Endbericht

Field Level Computation Cloud

PG 557, Fakultät für Informatik, TU Dortmund

Martin Bring, Iryna Denysenko, Katharina Diekmann, Tim Düllmann,
René Grzeszick, Katrin Holterbork, Sebastian Homann, Henning Meister,
Stanislav Nikolov, Marco Pfahlberg, Dirk Schalge, Nils Schmidt

BETREUER: Sascha Feldhorst und Christian Mosblech

Wintersemester 2011/2012
4. JUNI 2012

Inhaltsverzeichnis

KAPITEL 1

Einleitung

Die Projektgruppe 557 mit dem Titel „FiLeCC - Field Level Computation Cloud: Verlagerung von übergeordneten Steuerungsfunktionen in die Feldebene von Materialflusssystemen mithilfe eines Cloud-basierten Ansatzes“ wird vom Lehrstuhl für Förder- und Lagerwesen der Fakultät Maschinenbau betreut. Die Idee für die Projektgruppe ergibt sich aus kürzlich durchgeführten Forschungsarbeiten (siehe [FLt+09] und [FFt10]) im Bereich Materialflusssysteme. Diese beschreiben u. a. Ansätze zur Verlagerung von Intelligenz in die Feldebene, indem Fördermodule mit Rechen- und Speicherressourcen ausgestattet werden und sich an der Materialflusssteuerung beteiligen. Dabei kann es vorkommen, dass die Rechen- und Speicherressourcen durch die Steuerungsaufgaben für das Materialflusssystem nicht vollständig ausgelastet werden. Gleichzeitig müssen für übergeordnete Steuerungsfunktionen zusätzliche Ressourcen bereitgestellt werden. Deshalb besteht das Ziel, die ungenutzten Ressourcen in der Feldebene ebenfalls für übergeordnete Steuerungsfunktionen nutzbar zu machen.

Das Problem der ungenutzten Ressourcen besteht allerdings nicht ausschließlich in Materialflusssystemen. Ein Anwendungsbeispiel außerhalb der Logistik sind beispielsweise Set-Top-Boxen, wie sie zum Fernsehempfang verwendet werden. Sie sind in vielen Haushalten verfügbar und könnten in den Zeiten, in denen sie nicht durch den Fernsehempfang ausgelastet werden, ihre Ressourcen für andere Aufgaben z. B. über das Internet bereitstellen. Weitere Anwendungsgebiete ergeben sich durch die Entwicklung von Großrechnern hin zu vielen kleinen Recheneinheiten, die durch die zunehmende Miniaturisierung und kostengünstige Herstellung von digitaler Elektronik angetrieben wird. Solche Recheneinheiten können z. B. Mobiltelefone oder Multimediageräte sein, aber auch Steuerungselemente einer Logistik-Förderanlage. Oftmals sind die Recheneinheiten für ihre ursprüngliche Aufgabe überdimensioniert, sodass viele Rechen- und Speicherressourcen ungenutzt bleiben. Die Spezialisierung einzelner Einheiten führt ebenfalls dazu, dass immer mehr Ressourcen ungenutzt bleiben. Die Motivation für die Projektgruppe ist deshalb, die Ressourcen von Einheiten ab einer bestimmten Ressourcenkapazität nutzbar zu machen. Zentrale Anforderung bei der Lösung dieser Aufgabe ist ein dezentrales System,

dass flexibel erweitert und verkleinert werden kann und dadurch eine hohe Skalierbarkeit bietet. Diese Anforderung kann mit einer Cloud-Lösung erfüllt werden. Mit dem Begriff *Plug- & Play-Cloud* wird die von der Projektgruppe entwickelte Softwareplattform mit dem Namen *FiLeCC* bezeichnet. FiLeCC ist als Ausführungsplattform konzipiert, die nach außen hin eine einheitliche Schnittstelle nach dem Konzept der *Platform as a Service* anbietet. Die Ziele bei der Entwicklung sind neben der erwähnten Flexibilität und Skalierbarkeit eine Lastverteilung, Ausfallsicherheit, automatische Softwareverteilung, Ressourcenüberwachung und Speicherverwaltung. Die Lastverteilung teilt die Aufgaben anhand der durch die Ressourcenüberwachung ermittelten Daten auf die in der Cloud verfügbaren Ressourcen auf. Fällt eine Ressource aus, wird sie und alle von ihr wahrgenommenen Aufgaben durch eine andere Ressource ersetzt. Die automatische Softwareverteilung ermöglicht dem Systemadministrator, Software-Programme unkompliziert einzuspielen. Neben den Rechenressourcen können durch die Speicherverwaltung auch die Speicherressourcen genutzt werden.

Ein Minimalziel der Projektgruppe bei der Entwicklung von FiLeCC ist eine funktionierende Software zur Cloud-Steuerung. Dies entspricht der Entwicklung der genannten Ausführungsplattform und deren Erprobung in einem Testszenario. Ein weiteres Minimalziel ist die Implementation von mindestens einem Modul, das die Ressourcen der Cloud über die Ausführungsplattform nutzt. Dieses soll funktional eine Erkennung von Anlagenmodulen ermöglichen und daraus eine Topologie erstellen. Ein optionales Ziel ist die Implementierung eines Statistik-Moduls, welches Laufzeitdaten der Anlagenkomponenten erfasst und statistisch auswertet. Die dadurch gewonnenen statistischen Kennwerte sollen in die Steuerung einfließen und den Materialfluss optimieren.

In den folgenden Kapiteln wird die von der Projektgruppe geleistete Arbeit beschrieben und herausgestellt, welches Vorgehen der Projektgruppe zum Erreichen der o. g. Minimalziele führt. Dazu wird zunächst in Kapitel 2 die Organisation der Projektgruppe vorgestellt. Daran schließt sich in Kapitel 3 der Ablauf der Projektgruppenarbeit an, die in mehrere Phasen unterteilt ist. In Kapitel 4 werden anschließend die Grundlagen dargelegt, die für die Entwicklung der Cloud-basierten Ausführungsplattform benötigt werden. Hieran anschließend behandeln die folgenden Kapitel die für die Entwicklung der Ausführungsplattform erforderlichen Schritte und Maßnahmen. Kapitel 5 enthält, neben der Motivation für die zu entwickelnde Software, eine Definition der konkreten Anforderungen und beschreibt die daraus abgeleiteten Funktionalitäten. Die Motivation kann als Detaillierung der zu Beginn dieser Einleitung beschriebenen Problemstellung aufgefasst werden. Die durchgeführte Auswahl der verfügbaren Technologien wird in Kapitel 6 umfassend erläutert und bildet die Basis für den Entwurf und die Implementierung der Software-Architektur, die in Kapitel 7 beschrieben wird. Kapitel 8 beschreibt den Versuchsaufbau und die mit der Software durchgeführten Tests, anhand derer die Erfüllung der erwähnten Minimalziele demonstriert wird und nachvollzogen werden kann.
Abschließend fasst Kapitel 9 die Arbeit der Projektgruppe zusammen und gibt ein Fazit des Entwicklungsstandes. Zudem werden in diesem Kapitel Erweiterungsmöglichkeiten diskutiert, durch die die entwickelte Software ein größeres Aufgabenspektrum abdecken kann.

KAPITEL 2

Organisation der Projektgruppe

Das folgende Kapitel befasst sich mit wichtigen Aspekten der Organisation der Projektgruppe. Im Zuge dessen gibt es einen Einblick in die Verwantwortlichkeiten innerhalb der Gruppe (siehe Abschnitt 2.1), das genutzte Konfigurationsmanagement (beschrieben in Abschnitt 2.2) und die für das Projekt gewählte Lizenz, die im Abschnitt 2.3 beschrieben wird.

2.1 Verantwortlichkeiten

Zur transparenten Verteilung der Verantwortlichkeiten hat die Projektgruppe übergeordnete Organisationsaufgaben, wie z. B. die Projektleitung, an einzelne Mitglieder vergeben. Die Übersicht in Tabelle 2.1 stellt die Zuordnung der Verantwortlichen zu den jeweiligen Aufgaben über den gesamten Zeitraum der Projektgruppe dar.

2.2 Konfigurationsmanagement

Bei der Softwareentwicklung entstehen unabhängig vom genutzten Programmiermodell Ergebnisse in Form von Dokumenten oder Quelltexten. Das Konfigurationsmanagement gewährleistet die sichere Verwaltung dieser Ergebnisse und ermöglicht den Mitarbeitern eines Softwareentwicklungsprojektes einen kontrollierten Zugriff auf diese [Pop08]. Durch das Konfigurationsmanagement werden Richtlinien und Vorschriften vorgegeben, welche die Zusammenarbeit eines Teams fördern, kontrollieren und ein reibungsloses Arbeiten erleichtern. Das Konfigurationsmanagement hilft somit, die Qualität des erstellten Software-Produkts sicherzustellen. In den Unterkapiteln 2.2.1, 2.2.2 und 2.2.3 werden die Ziele des Konfigurationsmanagement, des Änderungsmanagement und die eingesetzten Werkzeuge vorgestellt.

Tabelle 2.1: Übersicht der fest vergebenen Organisationsaufgaben

Bereich	Verantwortlicher
Leitung der Projektgruppe stellv. Leitung	Sebastian Homann Nils Schmidt
Organisation des Projekt-Wikis	Nils Schmidt
Organisation des SVN-Repository	René Grzeszick
Serveradministration	Henning Meister Dirk Schalge
Strukturierung und Betreuung des Pflichtenhefts	Katharina Diekmann Martin Bring
Strukturierung und Betreuung des Zwischenberichts	Katharina Diekmann Katrin Holterbork
Strukturierung und Betreuung der Handbücher	Iryna Denysenko
Strukturierung und Betreuung des Endberichts	Iryna Denysenko Sebastian Homann

2.2.1 Ziele

Die Ziele eines Konfigurationsmanagementprozesses lassen sich in vier Teilbereiche einteilen [Pop08]. Dabei wird i. d. R. jeder Bereich durch ein oder mehrere Werkzeuge unterstützt, die meistens zu Beginn eines Projektes ausgewählt werden.

Änderungen kontrollieren Änderungen lassen sich in kleineren Entwicklerteams manuell kontrollieren. Die Kontrolle der Änderungen benötigt eine strikte Abstimmung aller Mitglieder, an welchem Teil einer Software sie gerade arbeiten. Dabei wächst der Kommunikationsaufwand überproportional zur Anzahl der Mitglieder, da jedes der n Mitglieder eines Teams mit $(n-1)$ Mitgliedern kommunizieren muss [Pop08]. Beispielsweise können Änderungen am Quelltext in einem zentralen Repository gepflegt und mit Kommentaren verbunden werden, wodurch die Kommunikation deutlich vereinfacht wird. Ein Repository ist ein Lagerort für geordnete Dokumente, der allen Mitgliedern zugänglich ist. Ohne ein solches Repository entstehen bei gleichzeitiger Bearbeitung einer Datei durch mehrere Entwickler Komplikationen. Es gibt verschiedene Ansätze festzulegen, welche Version in das Repository übernommen werden soll. Eine Version beschreibt dabei einen Zustand einer Datei. Ein möglicher Ansatz ist, die zuletzt gespeicherte Version zu wählen. Dann gehen alle Änderungen der Entwickler verloren, die zeitlich vor der letzten Speicherung gespeichert haben. Die in dieser PG genutzten Werkzeuge bieten bessere als das gerade genannte Verfahren, um Konflikte zu lösen. Die Werkzeuge werden in Abschnitt 2.2.3 beschrieben.

Produktivität steigern Die Werkzeuge, die zum Konfigurationsmanagement eingesetzt werden, helfen den Entwicklern, ihre Produktivität zu steigern. Eine Versionsverwaltung ermöglicht beispielsweise, zwei Versionen einer Datei zu vergleichen.

Qualität sichern Eine zum Konfigurationsmanagement eingesetzte Software kann automatisch Tests des Produktes ausführen, auswerten und gegebenenfalls die verantwortlichen Entwick-

ler benachrichtigen. Die Tests müssen im Vorfeld erstellt worden sein, da dies keine Aufgabe des Konfigurationsmanagements ist. Das Konfigurationsmanagement hilft ebenfalls dabei, Fehler zu dokumentieren, zu bewerten und zu priorisieren. Dadurch wird ein Überblick über den Zustand der Software realisiert. Die Qualität wird außerdem zusätzlich durch automatische Build-Skripte, die beispielsweise automatisch eine ausführbare Datei des Projektes erstellen, verbessert, welche eine verlässliche Erstellung des Produktes zur Auslieferung sicherstellen.

Transparenz verbessern Um eine höhere Transparenz zu erzielen, wird häufig eine Projekthomepage angelegt [Pop08]. Auf dieser kann der Zustand des Projektes anhand von (automatisch) erstellten Auswertungen dargestellt werden. Darunter fällt beispielsweise eine Übersicht über die erstellten Fehlermeldungen und über die automatischen Testläufe. Die Projekthomepage soll vor allem dem Projektleiter einen besseren Einblick in das Projekt gewähren.

2.2.2 Änderungs- und Fehlermanagement

Das Änderungs- und Fehlermanagement umfasst die schriftliche Dokumentation von Änderungswünschen, deren Umsetzung und Fehlerberichten. Es stellt sicher, dass eine Änderung erst durchgeführt wird, wenn sie geprüft ist und eine Bewertung der Auswirkungen stattgefunden hat. Die Änderungen werden vom Änderungsmanagement koordiniert und kontrolliert. Allen Änderungen wird eine Priorität und eine eindeutige Nummer zugewiesen. Weitere Parameter eines Änderungswunsches sind zum Beispiel: Name des Autors, Datum der Erstellung, Aktueller Status (z. B. „Vorgelegt“, „Angenommen“, „Umgesetzt“ oder „Freigegeben“), kurze inhaltliche Beschreibung, Bewertung hinsichtlich der Wichtigkeit und Aufwand oder zugewiesener Bearbeiter. Ein Fehlerbericht enthält die gleichen Daten, lediglich der Typ unterscheidet sich vom Änderungswunsch (siehe 2.2.3).

2.2.3 Eingesetzte Werkzeuge

Zur Unterstützung der Arbeit der Projektgruppe werden Softwarelösungen wie Subversion, Redmine und Maven eingesetzt. Diese Produkte zeichnen sich dadurch aus, dass sie kostenfrei verfügbar sind. Im Folgenden werden diese kurz vorgestellt.

Subversion Subversion ist eine Software, die die Versionierung von Dateien ermöglicht. Subversion beseitigt im folgenden beschriebene Schwächen von CVS [Pop08]. Beim Übertragen von Änderungen in ein CVS-Repository wird jede Datei einzeln gespeichert und erhält eine eigene Versionsnummer. Subversion hingegen verwendet eine Delta-Speicherung, die lediglich die geänderten Teile einer Datei im Dateisystem ablegt. Weiterhin gewährleistet Subversion Transaktionssicherheit. Ein Check-in ist dabei atomar und wird entweder vollständig oder gar nicht durchgeführt. Treten beispielsweise Netzwerkprobleme während des Check-ins auf, entsteht, im Gegensatz zu einem CVS-Repository, kein inkonsistenter Zustand, in dem nur ein Teil der Dateien im Repository aktualisiert wird. Weiterhin kann Subversion Binärdateien verwalten und versionieren. In einem CVS-Repository wird bei jeder Änderung die ganze Binärdatei ausgetauscht.

Redmine Redmine ist eine auf dem *Ruby On Rails*-Framework basierende Open-Source-Lösung, die eine flexible, funktionsreiche Projekthomepage zur Verfügung stellt. Die Hauptaufgabe von Redmine ist die Unterstützung des Änderungsmanagements (siehe Unterkapitel 2.2.2). Es können Arbeitsaufträge in Form von Tickets erstellt werden, die Fehler im Projekt oder Änderungswünsche dokumentieren. Jedes Ticket besitzt einen Ticket-Typ, beispielsweise Fehler oder Feature, einen Betreff, einen Status („neu“, „zugewiesen“, „erledigt“) und eine Priorität. Um Rückfragen der Entwickler zu vermeiden und dadurch eine Arbeitserleichterung zu erreichen, kann außerdem eine genauere Beschreibung angegeben werden. Weitere optionale bzw. später zu füllende Attribute sind „Bearbeiter“, „Beobachter“, „Frist“, „geschätzter Aufwand“ und der „aktuelle Bearbeitungsstatus in Prozent“. Aus den Ticket Start- und Endzeiten wird zur Unterstützung des Projektmanagements automatisch ein Eintrag im Projekt-Gantt-Diagramm und ein Projekt-Kalender generiert.

In der Redmine-Konfiguration kann u. a. ein Subversion-Repository angegeben werden. Das Repository kann anschließend auf der Projekthomepage durchsucht werden. Weiterhin können Vergleiche zwischen zwei Versionen direkt auf der Projekthomepage durchgeführt werden. Um die Teamarbeit weiter zu unterstützen, bietet Redmine zusätzlich die Möglichkeit, Dateien auszutauschen oder Neuigkeiten zu verbreiten. Weiterhin stehen ein Forum und ein Wiki zur Verfügung, welche für die Kommunikation genutzt werden können.

Maven Maven ist ein Werkzeug zur Durchführung eines automatischen Build-Prozesses. Maven führt den eigentlichen Build-Prozess durch, verwaltet die Abhängigkeiten zu externen Bibliotheken und erstellt eine Projektdokumentation. Es wird dabei ein modellbasierter, deklarativer Ansatz zur Umsetzung des Build-Prozesses genutzt [Pop08]. Statt ein Skript für den Build-Prozess zu entwickeln, wird ein Projektmodell (POM) angelegt, welches die Metadaten zum Projekt enthält. Diese Metadaten enthalten beispielsweise die verwendete Projektstruktur und die benötigten externen Bibliotheken. Eine wichtige Funktion von Maven ist dabei das automatische Verwalten von Bibliotheken. Dazu wird im Projektmodell der Name der Bibliothek und die gewünschte Version eingegeben und Maven lädt diese Version automatisch während des Build-Prozesses aus einem Maven-Repository herunter. Maven ist modular aufgebaut, so dass der Maven-Kern ausschließlich die Funktionen enthält die zum Interpretieren eines Projektmodells nötig sind. Für den eigentlichen Build-Prozess werden zusätzliche Plugins heruntergeladen. Diese Plugins finden beispielsweise automatisch Fehler im Quelltext, überprüfen dessen Lesbarkeit oder komprimieren ihn während des Build-Prozesses. Das Ergebnis eines Build-Prozesses sind eine oder mehrere Auslieferungsdateien. Dabei können bei umfangreichen Projekten mehrere Projektmodelle angelegt werden, die eigenständige Module beschreiben und jeweils eine Auslieferungsdatei erstellen. Ein übergeordnetes Projektmodell fasst alle Module zu einem Gesamtprojekt zusammen. Wird der Build-Prozess für das Projektmodell auf der obersten Ebene angestoßen, delegiert Maven die Ausführung automatisch an die untergeordneten Modelle.

2.3 Lizenz

Bei der Softwareentwicklung ist es üblich dem Produkt ein Lizenz zu geben. Deswegen ist in der Projektgruppe untersucht worden, unter welcher Lizenz die entwickelte Software veröffentlicht werden soll. Hierzu sind existierende Open-Source-Lizenzen und deren Eigenschaften in Betracht

gezogen worden, um eine optimale Auswahl treffen zu können. Diese wird im Folgenden vorgestellt und anschließend die Lizenz für FiLeCC bestimmt.

Public Domain Bei dieser Lizenzierungsart gibt der Urheber nicht nur alle persönlichen Rechte, sondern auch das Urheberrecht auf. Für Deutschland gilt diese Lizenz nicht, da es immer einen Urheber geben muss.

GNU General Public License (GPL) Diese Lizenz besagt, dass Software mit Quelltext zur Verfügung gestellt wird und von anderen Lizenznehmern nur unter dieser Lizenz verbreitet werden darf. Alle Veränderungen müssen der GPL-Lizenz unterliegen und die Namen der Rechteinhaber genannt werden.

GNU Lesser General Public License (LGPL) Software unter der LGPL darf für beliebige Zwecke genutzt, weitergegeben, verändert und die veränderte Version verbreitet werden. Die veränderte Version muss wieder unter LGPL oder GPL lizenziert werden und quelloffen sein. Wird die Software in eine andere Software integriert, so muss diese ebenfalls unter der LGPL oder GPL veröffentlicht werden. Wird eine Software unter der LGPL nur extern genutzt, also beispielsweise als Bibliothek, kann die verwendete Software unter einer anderen Lizenz veröffentlicht werden, also auch proprietär sein. Als proprietär wird Software bezeichnet, die urheberrechtlich (durch Lizenzpflicht) geschützt ist ([Hes04]). Eine Softwareveränderung ist nicht erlaubt.

Apache-Lizenz Eine unter der Apache-Lizenz gestellte Software darf frei genutzt, verändert und verteilt werden. Beim Verteilen muss aber unter Nennung des Rechteinhabers eindeutig darauf hingewiesen werden, dass die Software der Apache-Lizenz untersteht. Die Software muss außerdem eine Kopie der Apache-Lizenz enthalten. Es ist erlaubt, den Quelltext zu verändern, ohne dass der Rechteinhaber darüber informiert werden muss. Wird Software, die unter der Apache-Lizenz veröffentlicht ist, in einer anderen Software verwendet, so muss diese nicht der Apache-Lizenz unterliegen (im Gegensatz zu GPL). Mit Einverständnis der Apache Foundation, darf die Software Apache genannt werden.

Berkeley Software Distribution (BSD)-Lizenz Software unter der BSD-Lizenz darf frei verwendet, verändert und verbreitet werden, solange der ursprüngliche Urheberrechtshinweis nicht geändert und die Erwähnung der Universität von Berkeley beigefügt wird. Im Gegensatz zur GPL muss ein unter der BSD-Lizenz veröffentlichtes Produkt, nachdem es verändert wurde, nicht wieder unter der BSD-Lizenz verbreitet werden. Es ist nicht erforderlich den neuen Quelltext mit zu veröffentlichen.

Massachusetts Institute of Technology (MIT)-Lizenz Eine unter der MIT-Lizenz gestellte Software darf sowohl für Software, deren Quelltext frei einsehbar ist, als auch für Software, deren Quelltext nicht frei einsehbar ist, genutzt werden. Die Lizenz steht nicht unter Urheberrecht und der Lizenzinhalt darf daher beliebig verändert werden.

Die Eigenschaften der zuvor genannten Lizenzen sind in der Tabelle 2.2 zusammengefasst. Dort werden diese Lizenzen verglichen unter Betrachtung, ob sie proprietär oder frei sind, Verteilung mit bzw. ohne Änderung, und Kompatibilität zu GPL. Weitere Eigenschaften sind in der Spalte Anmerkung zu finden.

Tabelle 2.2: Lizenzenübersicht und deren Eigenschaften

Name	**Proprietär**	**Verteilen ohne Änderung**	**Verteilen mit Änderung**	**GPL kompatibel**	**Anmerkung**
GPL	nein	nur wenn Software ebenfalls GPL ist	nur wenn Software ebenfalls GPL ist	ja	weit verbreitet
LGPL	ja	ja, allerdings muss Sourcecode beigelegt werden	Änderungen müssen ebenfalls GPL oder LGPL sein	ja	weit verbreitet für Bibliotheken
Apache	ja	ja	ja	nein	Patente werden ebenfalls zur Nutzung freigegeben
BSD	ja	ja	ja	ja (modified BSD)	
MIT	ja	ja	ja	ja	

Die Teilnehmer der Projektgruppe haben eine Entscheidung für die GNU Lesser General Public License (LGPL) Version 3, 29 Juni 2007 herbeigeführt. Somit steht FiLeCC unter einer Lizenzierung als freie Software. Weiterführende Informationen über diese Lizenz können unter ([The12]) eingesehen werden.

KAPITEL 3

Ablauf der Projektgruppe

Im folgenden Kapitel werden die verschiedenen Projektphasen beschrieben, die von der Projektgruppe im Laufe des Bearbeitungszeitraumes durchlaufen wurden. Im ersten Semester hat die Projektgruppe eine Seminar- und Praktikumsphase absolviert (siehe Abschnitte 3.1.1 und 3.1.2), anschließend wurde die notwendige Planung und Recherche für das Projekt vorgenommen (siehe Abschnitt 3.1.3), die zum Ende des Semesters in den Entwurf von FiLeCC überging (siehe Abschnitt 3.1.4). Im zweiten Semester hat die Projektgruppe die Implementierung inklusive anschließender Tests und Dokumentation vorgenommen (siehe Abschnitte 3.2.1 und 3.2.2).

3.1 Erstes Semester

Um einen stetigen Fortschritt erzielen zu können, wurde das erste Semester der Projektgruppe in mehrere Phasen eingeteilt, in denen Kleingruppen verschiedene Themen erarbeiten und der restlichen Gruppe vorstellen mussten. Durch diese Vorgehensweise konnte die Vielzahl an Themenbereichen effizient bearbeitet werden. In wöchentlichen Besprechungen wurden Probleme, Fortschritte und aufkommende Fragen besprochen, sowie die grundlegende Architektur des zu entwickelnden Systems erarbeitet. Abbildung 3.1 zeigt eine Übersicht der einzelnen Phasen, welche im Folgenden vorgestellt werden.

3.1.1 Seminarphase

Vor Beginn des Semesters fand die Seminarphase statt. Das Ziel dieser Phase war es, allen Teilnehmern der Projektgruppe einen Einblick in das für das Projekt FiLeCC relevante Themen zu geben und den Wissensstand der einzelnen Teilnehmer auf diesen Gebieten anzugleichen. Die

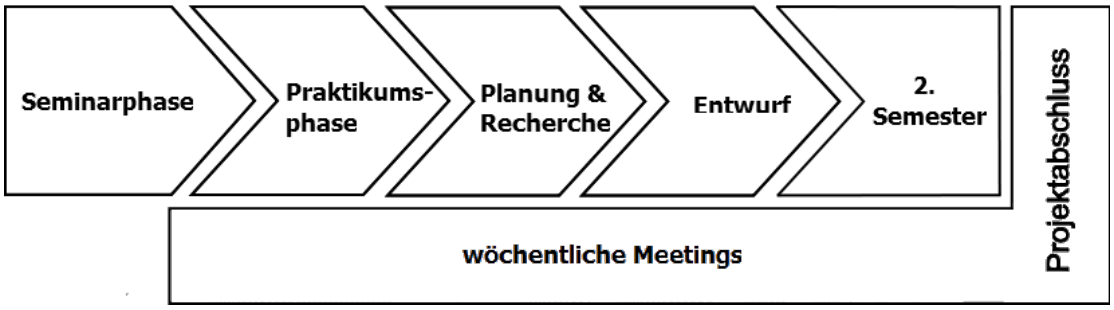


Abbildung 3.1: Übersicht der PG-Organisation

Teilnehmer beschäftigten sich mit grundlegenden Themen aus den Bereichen Logistik, Kommunikation und Cloud Computing. Jeder Teilnehmer erarbeitete eine ca. 10-seitige Ausarbeitung und einen 40-minütigen Vortrag. Das Präsentieren der Ergebnisse durch die jeweiligen Teilnehmer stellte den offiziellen Start der Projektgruppe dar. Die Themen der Seminarphase bilden die Grundlage für Kapitel 4, in dem die Ergebnisse der Seminarphase vorgestellt werden.

3.1.2 Praktikumsphase

In der Praktikumsphase wurden, in Gruppen von zwei Personen, Themen detailliert erarbeitet und Aufgaben erstellt, die vom Rest der Gruppe zu lösen waren. Diese Themen bezogen sich zum einen auf die Einarbeitung in verschiedene Werkzeuge und Betriebssysteme, zum anderen auf die Untersuchung möglicher Fremdsoftware, die innerhalb des zu programmierenden Systems Verwendung finden kann. Konkret waren das die Themen: Nutzung von Clouds, SOA und DPWS, Entwicklungsumgebungen und Laufzeitumgebungen. Ziel dieser Phase war es, sich in bestehende Technologien einzuarbeiten. Auch wurde in dieser Phase angestrebt, das erworbene Wissen unter den Teilnehmern zu verteilen.

3.1.3 Planung und Recherche

In den ersten Wochen wurde ein gemeinsamer Wissensstand aufgebaut und erste Technologien evaluiert. Danach begann die Planungs- und Recherchephase. Innerhalb dieser Phase wurde die grundlegende Architektur erarbeitet und in den wöchentlichen Meetings besprochen. Dafür wurden in Kleingruppen die für diese Basisarchitektur erforderlichen Kenntnisse erworben und die benötigten Technologien evaluiert. Diese wurden allen Teilnehmern vorgestellt. Es wurden eigene Softwareschnittstellen vorgeschlagen, die für die Umsetzung des FiLeCC-Systems benötigt wurden.

3.1.4 Entwurf

Im letzten Abschnitt des ersten Semesters wurde die Architektur des zu entwickelnden Systems entworfen. Diese Phase stellte einen iterativen Prozess dar. Die einzelnen Bestandteile (Module) des Systems wurden in Kleingruppen konzipiert und den anderen Teilnehmern präsentiert. Änderungsvorschläge konnten in der nächsten Iteration in die Module eingearbeitet werden. Die einzelnen Module sind in dem Kapitel 7 in den Unterkapiteln *Software Deployment and*

Runtime Management, UNO, Leaderwahl, Watchdog, Lastverteilung, Ressourcen Monitoring, Nachbarschaftserkennung und *Datenhaltung* beschrieben. Als Ergebnis dieser Phase ergab sich aus der Kombination der einzelnen Module die komplette Architektur des Systems. Zusätzlich wurden gezielt eigene Algorithmen entwickelt, um Teilprobleme des Projekts zu lösen. Mit den in dieser Phase definierten Softwareschnittstellen und Implementierungsvorschlägen, wurde die Grundlage geschaffen, um im zweiten Semester mit der Umsetzung der Architektur zu beginnen.

3.2 Zweites Semester

Nachdem im ersten Semester die Planung des Systems abgeschlossen wurde, wurde im zweiten Semester die Umsetzung der zuvor geplanten Architektur vorgenommen. Die in Abschnitt 3.2.1 beschriebene Implementierung wurde, wie auch der Entwurf (siehe Abschnitt 3.1.4), in Kleingruppen bearbeitet und der Fortschritt, sowie Probleme bei der Umsetzung in wöchentlichen Besprechungen diskutiert. Anschließend wurde das System getestet und eine ausführliche Dokumentation angefertigt (siehe Abschnitt 3.2.2) . Zum Abschluss der Projektgruppe wurde, wie in Abschnitt 3.2.3 beschrieben, eine Präsentation inklusive Vorführung des Systems erarbeitet, sowie ein Ausblick über die Stärken und Schwächen einer solchen Cloud-Plattform gegeben.

3.2.1 Implementierung

Mit dem Beginn des zweiten Semesters begann eine mehrwöchige Implementierungsphase, in der die entworfenen Module durch die Mitglieder der Projektgruppe umgesetzt wurden. Abhängig vom Arbeitsaufwand des jeweiligen Moduls wurde angestrebt, die Gruppen aus der Entwurfsphase beizubehalten. Im Zuge der Implementierung wurden Details des Entwurfs angepasst, die durch technische Gegebenheiten nicht wie geplant umgesetzt werden konnten (siehe Kapitel 7). Fehler, sowie unerwartete Schwierigkeiten im Cloud-Umfeld (z. B. im Bereich Netzwerk-Kommunikation) wurden erkannt und gelöst. Zusätzlich war jede Gruppe dafür verantwortlich, die Funktionalität des implementierten Moduls losgelöst vom Gesamtsystem durch JUnit-Tests (siehe [BG+12]) zu verifizieren.

3.2.2 Test und Dokumentation

Die Implementierungsphase ging nach der Fertigstellung der Rahmenfunktionalität fließend in die Testphase über, in der das Zusammenspiel der einzelnen Komponenten erprobt wurde. Fehler im Realbetrieb wurden aufgedeckt und behoben. Es wurden die im Pflichtenheft (siehe Anhang A) beschriebenen Testfälle der Cloud und der darauf ausgeführten Nachbarschaftserkennung (siehe Kapitel 7.8) auf ihre Korrektheit überprüft.
Parallel wurden ein Entwickler- und eines Administratorenhandbuches erstellt (siehe Appendix B und C. Das Entwicklerhandbuch soll Softwareentwicklern beim Entwurf und der Umsetzung neuer Webservices für die Plattform helfen und die Rahmenbedingungen bei der Cloudentwicklung herausstellen. Das Administratorhandbuch beschreibt die Einstellungsmöglichkeiten der Cloud-Infrastruktur und soll die Verwaltung von FiLeCC erleichtern.

3.2.3 Abschluss

Zum Abschluss der Projektgruppe wurde das System und seine Funktionalitäten in einer Testumgebung vorgestellt. Ziel war es, eine möglichst inhomogene Testumgebung aus verschieden leistungsfähigen Rechnern mit verschiedenen Architekturen und unterschiedlichen Betriebssystemen zu schaffen. Die Lauffähigkeit des Systems auf IPCs, Pandaboards und Laptops mit Linux- oder Windows-Betriebssystem wurde gezeigt. Im Rahmen der Abschlussvorstellung wurde auch auf Stärken und Schwächen der Architektur eingegangen und zukünftige Entwicklungsmöglichkeiten aufgezeigt.

3.3 Zeitplanung

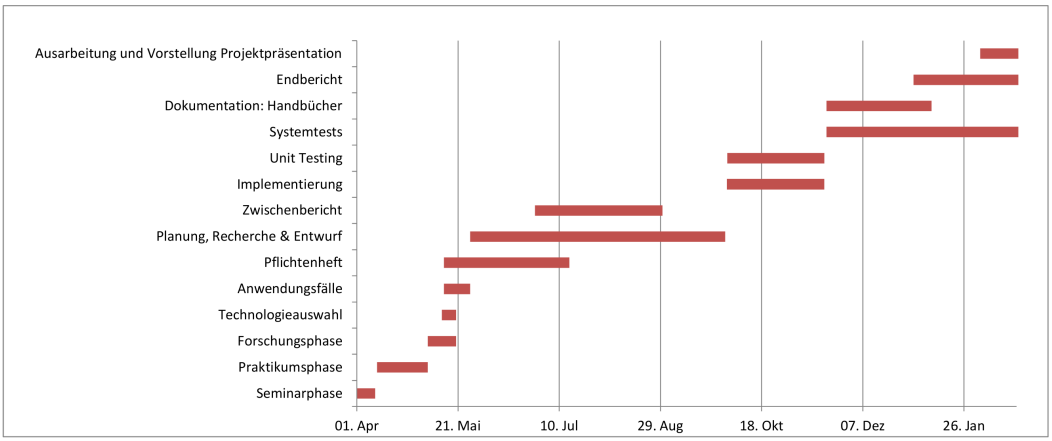


Abbildung 3.2: Zeitplan für den Ablauf der Projektgruppe im Sommersemester 2011 und Wintersemester 2011/12

Die in Abbildung 3.2 dargestellte Zeitplan, zeigt eine Übersicht über die jeweiligen Phasen der Projektgruppe.

Die Seminarphase wurde vor Beginn des 1. Semesters vorbereitet und die Ergebnisse wurden anschließend am 09. und 10.04.2011 der Gruppe präsentiert. Die vier darauffolgenden Wochen der Vorlesungszeit hat die Projektgruppe für die Praktikumsphase aufgewendet (bis 06.05.2011). Anschließend wurden, wie in Abschnitt 3.1.3 beschrieben, die ersten Technologie-Analysen in einer zweiwöchigen Forschungsphase vorgenommen (06.2011–20.05.2011). Auf dieser Basis war die Projektgruppe Mitte Mai in der Lage erste Anwendungsfälle für die zu erstellende Software konkret zu formulieren und die Arbeit an einem Pflichtenheft zu beginnen. Ein Lastenheft war in Form der ursprünglichen Aufgabenstellung gegeben. Die Anwendungsfälle gingen nahtlos in offene Fragen, Planung und weitere Recherche über und führten zum Entwurf der Architektur, die, inklusive der anzufertigenden Klassendiagramme, bis zum Ende der Vorlesungszeit am 15.07.2011 abgeschlossen waren. Einzelne Themengebiete, wie z. B. die Leaderwahl, Loadbalancing und die Auswahl der zu verwendenden verteilten Datenbank wurden bis zum Ende des ersten Semesters am 31.09.2011 fortgeführt.
Das zweite Semester begann mit der Implementierung und den damit zusammenhängenden Unittests, die bis zum 18.11.2011 umgesetzt wurden. Die anschließenden Systemtests haben

Änderungen erforderlich gemacht, die bis zur Abschlusspräsentation der Projektgruppe eingearbeitet wurden. In den folgenden Wochen wurde die Dokumentation des Systems in Form von je einem Entwickler- und Administrationshandbuch (18.11.11 - 10.01.12) abgeschlossen und die Arbeiten am Endbericht begonnen. Die Vorstellung des Projektes fand am 01.03.2012 statt.

KAPITEL 4

Grundlagen – Ergebnisse der Seminarphase

In der in Kapitel 3.1.1 vorgestellten Seminarphase wurden verschiedene Themen der Informatik und Logistik erarbeitet und so präsentiert, dass jeder Teilnehmer einen Überblick über das entsprechende Thema gewinnen konnte. Die Ergebnisse der Seminarphase bilden zugleich die Grundlagen der Projektgruppe und sind deshalb an dieser Stelle zusammenfassend aufgeführt.

In Abschnitt 4.1 werden die Grundladen zu Materialflusssystemen erklärt, bevor in Abschnitt 4.2 auf den derzeitigen Einsatz solcher Systeme eingegangen wird. Abschnitt 4.3 legt die Grundlagen für die Idee, wie Förderanlagen in Zukunft aussehen könnten. Dabei wird auf dafür notwendige Technologien der Hardware und Kommunikation eingegangen und speziell auch auf die Serviceorientierten Architekturen. In den Abschnitten 4.4 und 4.5 werden diese Technologien anschließend auf die aktuelle Sitation der Förderanlage am Lehrstuhl für Förder- und Lagerwesen der TU Dortmund und die Idee der Smart-Neighbourhood Module eingegangen. Da diese in der Zukunftsvision mit der Technologie des Cloud Computing verbunden werden sollen, werden in Abschnitt 4.6 die Grundlagen verschiedener Cloud-Plattformen erläutert.

4.1 Grundlagen der Intralogistik

Wichtige Funktionen in der Logistik sind nach [tSN07] das Lagern, Verteilen, Zusammenführen und Bewegen von Gütern. Das Zusammenspiel zwischen betrieblicher Organisation und technischer Umsetzung dieser Funktionen lässt sich unter dem Begriff des *Materialflusssystems* zusammenfassen (siehe [tSN07]). Materialflusssysteme bilden die Kernkomponente der Intralogistik und bestehen aus der Förder- und Lagertechnik in Verbindung mit einem Informations- und Steuersystem.

4.1.1 Fördersysteme

Fördern ist nach [tSN07] die Ortsveränderung von Gütern in einem räumlich begrenzten Gebiet. Die dazu erforderliche Technik, die Fördertechnik, besteht dabei aus allen notwendigen technischen, organisatorischen und personellen Mitteln.

In vielen Bereichen des innerbetrieblichen Materialflusses kommt Fördertechnik zum Einsatz. Neben dem Transport von Gütern dienen Fördermittel des Weiteren dem *Verteilen, Sammeln, Sortieren, Puffern* und *Zwischenlagern*. Fördermittel werden in [tSN07] als die technischen Transportmittel, „die innerhalb von örtlich begrenzten und zusammenhängenden Betriebsbereichen (z. B. innerhalb eines Werks) das Fördern bewerkstelligen“, definiert. Unter Verteilen wird die gezielte Verteilung von Gütern von einem Aufkommensort zu einem oder mehreren Zielorten verstanden, während Sammeln die Zusammenführung von Gütern mehrerer Quellen und somit das Gegenteil von Verteilen bezeichnet (siehe dazu [tSN07]).

Fördersysteme lassen sich durch die unterschiedliche Beschaffenheit der Fördergüter und deren Fortbewegung in *Stetig-* und *Unstetigförderer* für *Schütt-* oder *Stückgut* klassifizieren. Da das der Arbeit der Projektgruppe zugrunde liegende Steuerungskonzept *Paket Royale* (siehe [tFF11]) auf Systeme für den stetigen Stückguttransport ausgelegt ist, beschränkt sich diese Einführung auf die Beschreibung von diesen Systemen.

4.1.2 Stetigförderer

Stetigförderer fördern das Gut in einem *diskret kontinuierlichen Fördergutstrom*. Dies bedeutet, dass Stückgüter einzeln von einem Förderer gefördert werden, der dauerhaft im Betrieb ist. Innerhalb dieser Klasse von Fördermitteln gibt es eine Unterteilung in flurgebundene, aufgeständerte und flurfreie Fördermittel. Da im Beispielaufbau für das *Paket Royale*-System nur aufgeständerte Fördertechnik zum Einsatz kommt, werden flurgebundene und flurfreie Fördermittel nur am Rand vorgestellt. In der VDI-Richtlinie 4440 (siehe [Ver07a, Ver07b, Ver07c, Ver07d, Ver07e]) werden mehrere Stetigförderer definiert.

Flurgebundene Stetigförderer Fördermittel sind nach [tSN07] *flurgebunden*, „wenn sie Verkehrswege am Boden nutzen oder über Einrichtungen verfahren, die im Boden eingelassen sind.“ Ein Beispiel für ein flurgebundenes Stetigfördermittel ist der *Unterflur-Schleppkettenförderer*, der in Kapitel 3.6 von [Ver07b] definiert wird. Es handelt sich dabei um eine in sich geschlossene Kette, die in eine Führungsschiene im Boden eingelassen als Zugorgan dient. Die Kette ist mit Laufrollen und Mitnehmern ausgestattet, mit deren Hilfe die Transportwagen durch ein loslösbares Klinkensystem mit der Zugkette verbunden werden können. Obwohl die Kette stetig in Bewegung ist, können die Transportwagen über eine Stoppvorrichtung angehalten werden, die die Klinke von dem Mitnehmer der Kette trennt. Des Weiteren verfügen die Transportwagen über eine Vorrichtung, die das Auflaufen aufeinanderfolgender Wagen auf einen bereits Stehenden verhindern.

Aufgeständerte Stetigförderer Fördermittel sind nach [tSN07] *aufgeständert*, wenn sie sich „in definierter Höhe über dem Boden mit Stützen aufgeständert befinden.“ Sie sind ortsfest angebracht und stellen ein Hindernis für verfahrbare Fördermittel oder Personen dar. Je nach

Bauart und verwendetem Trag- oder Zugmittel wird zwischen *Bandförderern, Kettenförderern* und *Rollenbahnen* unterschieden.

Bandförderer werden in Blatt 1 der VDI-Richtlinie 4440 (siehe [Ver07a]) vorgestellt. Eine Ausprägung eines Bandförderers ist der *Gurtförderer* für Stückgut. Das Gut wird dabei in waagerechter oder schräger Ebene auf einem Fördergurt transportiert. Der über angetriebene Trommeln geleitete Fördergurt dient dabei sowohl als Tragmittel als auch als Zugorgan. Die maximal überbrückbare Steigung eines Gurtförderers wird in der Richtlinie mit 30° angegeben. Damit der Gurt horizontal ausgerichtet bleibt und nicht seitlich schräg abfällt, ist neben anderen Faktoren die richtige Gurtspannung Voraussetzung. Um die erforderliche Spannung zu erreichen, ist der Gurtförderer mit Spannvorrichtungen ausgestattet. Soll eine Richtungsänderung im Stückguttransport vorgenommen werden, kommen in Verbindung mit den Gurtförderern *Kurvengurtförderer* zum Einsatz. Die Trommeln haben dabei eine konische Form, über die der kegelförmig geschnittene Gurt Kurven zwischen 30° und 180° bilden kann.

Kettenförderer bilden eine weitere Klasse von aufgeständerten Stetigförderern und werden im Blatt 2 der VDI-Richtlinie 4440 (siehe [Ver07b]) definiert. Förderketten aus Kunststoff oder Stahl dienen als Zugorgan und Transportmittel. Die Glieder der Kette sind dabei so flexibel miteinander verbunden, dass Kurven und Steigungen umsetzbar sind. Während Bandförderer zur Realisierung von Strecken mit Kurven und Steigungen aus mehreren Bändern, die jeweils eine Geradeausstrecke oder eine Kurve darstellen, zusammengesetzt werden müssen, kann unter Einsatz von Kettenförderern eine variable Streckenführung mit nur einer Kette umgesetzt werden. Der *Schuhsortierförderer* (siehe [Ver07b]) oder *Schiebeschuhsorter* (siehe [tS10]) ist eine Ausprägung des Kettenförderers, die nicht nur zum Fördern sondern auch zum Sortieren von Stückgut geeignet ist. Der Schuhsortierförderer besteht aus zwei vertikal umlaufenden Ketten, die mit Tragelementen zur Aufnahme und zum Transport von Stückgut verbunden sind. Auf den Tragelementen befinden sich verschiebbare Elemente, die Schiebeschuh genannt werden, und die auf den Tragelementen befindliches Stückgut von dem Fördermittel abschieben können. Hierdurch ist eine Sortierfunktion des Förderers gegeben. Dieser Förderer kommt am Warenausgang des Beispielaufbaus des *Paket Royale*-Systems zum Einsatz.

In der Richtlinie VDI 4440 werden in Blatt 3 (siehe [Ver07c]) Rollen- und Kugelbahnen definiert. Es wird dabei zwischen angetriebenen und nicht angetriebenen Rollenbahnen unterschieden. Diese Förderer bestehen laut [tSN07] „aus vielen hintereinander angeordneten, frei drehbaren, zwischen zwei Profilen befestigten Tragrollen.“ Nach [Ver07c] muss der Rollenabstand je nach Fördergut so bestimmt werden, dass das Gut zu jeder Zeit von mindestens drei Rollen getragen wird. Herunterfallen des Guts wird somit vermieden. Diese Vorgabe ermöglicht aber auch eine Optimierung der Förderanlage in Abhängigkeit der Größe des Förderguts. *Angetriebene Rollenbahnen* werden durch Gurte oder Ketten angetrieben, die von unten durch Druckrollen an die Trag- und Förderelemente gepresst werden. Einige Ausführungen verfügen über in die Rollen integrierte Elektroantriebe (siehe [tSN07]). *Nicht angetriebene Rollenbahnen* werden nach [tSN07] auch *Schwerkraftrollenbahnen* genannt und gleichen vom Aufbau den angetriebenen Rollenbahnen. Sie haben allerdings keine Antriebe. Damit Schwerkraftrollenbahnen automatisch Fördergut transportieren, müssen sie mit einer leichten Neigung montiert werden. Dabei ist darauf zu achten, dass das zu fördernde Gut einerseits an jeder Stelle in Bewegung gesetzt werden kann, aber andererseits zu keinem Zeitpunkt zu schnell wird. In [tSN07] wird daher eine Neigung von 2-5% empfohlen, die aber nur experimentell genau ermittelt werden kann. Wie in [Ver07c] beschrieben ist, können schwerkraftbetriebene Rollenbahnen auch waagerecht montiert werden, damit das Fördergut manuell leicht bewegt werden kann.

Durch radial angeordnete Rollen ist es zusätzlich möglich, *Kurvenrollenbahnen* zu realisieren. Damit die Unterschiedlichen Geschwindigkeiten innerhalb der Kurve ausgeglichen werden können, sind die Rollen entweder konisch geformt oder mehrfach geteilt. Auch Kurvenrollenbahnen gibt es in angetriebenen und schwerkraftbetriebenen Ausführungen (siehe [Ver07c]).

Flurfreie Stetigförderer Im Gegensatz zu den flurgebundenen und den aufgeständerten Stetigförderern sind *flurfreie* Stetigförderer an der Hallendecke befestigt. Flurfreie Stetigförderer sind ortsfest angebracht, bilden aber kein Hindernis für verfahrbare Transportmittel oder Personen. Ein Beispiel für einen flurfreien Stetigförderer ist der *Kreisförderer*. Kreisförderer bestehen nach [tSN07] aus einer in sich geschlossenen Kette, die das Zugmittel darstellt und an die Laufrollen montiert sind. Mit diesen Laufrollen wird die Kette entlang einer an der Gebäudedecke fest montierten Laufbahn geführt. Der Verlauf der Strecke ist dabei nahezu beliebig – von horizontal über ansteigend bis vertikal. Kurven können wie schon bei Kettenförderern ebenfalls leicht realisiert werden. Durch die Verwendung mehrerer Antriebe ist die Kette in der Länge theoretisch unbeschränkt. Sie kommen nach [tSN07] häufig in der Automobil- und Textilindustrie zum Einsatz.

4.1.3 Materialflusssteuerung

Die Materialflusssteuerung ist Bestandteil von Materialflusssystemen und ist für die automatische, zielgerichtete Förderung der Güter über die Anlage verantwortlich. Die Aufgaben der Materialflusssteuerung werden in [Gt10] genannt und beinhalten das *Ausführen von Transportaufträgen* unter *Einhaltung bestimmter Transportregeln* wie FIFO, *Wegoptimierung* und *Stau- und Blockadenvermeidung*. Neben den wesentlichen Transportaufgaben übernimmt die Materialflusssteuerung zusätzlich Verwaltungsaufgaben, wie das *Erstellen von Statistiken*, die *Datenerfassung*, für welche automatische Identifizierungstechniken notwendig sind, und die *Visualisierung*. Erst das Zusammenspiel von Transport- und Verwaltungsaufgaben ermöglicht einen effizienten Materialfluss.

Die eingesetzte Steuerungstechnik verbindet, z. B. mit Hilfe logischer Schaltungen, die an der Fördertechnik angebrachten Sensoren und Aktoren miteinander und stellt somit sicher, dass das Fördergut von der Quelle zur richtigen Senke gefördert wird.

Sensoren Nach [tBF08] erfassen Sensoren physikalische Größen und wandeln diese in geeignete Ausgangsgrößen um, die von einem Automatisierungsgerät, wie einer speicherprogrammierbaren Steuerung (*SPS*), weiterverarbeitet werden können. Sensoren werden je nach eingesetztem Messprinzip und Messverfahren unterschieden. Im Folgenden werden [tBF08] folgend einige Beispiele verschiedener Sensortypen unter Angabe ihres Einsatzgebietes angegeben.

Taster und Schalter dienen sowohl der manuellen Betätigung als auch in der Automatisierung, beispielsweise als Endschalter. Endschalter sorgen unter anderem bei automatischen Rolltoren dafür, dass diese nicht zu weit hoch- oder heruntergefahren wird. *Optische Sensoren* arbeiten mit Lichtsignalen, die zwischen Sender und Empfänger/Detektor ausgetauscht werden. Dabei wird auf Empfängerseite eine vorher festgelegte Eigenschaft, wie z. B. Intensität oder Laufzeit des Lichtsignals erhoben und verarbeitet. Bei Lichtsignalen beispielsweise wird über die Intensität des Lichtsignals die Unterbrechung der optischen Strecke zwischen Sender und Empfänger

detektiert und ein Signalwechsel am Sensorausgang ausgelöst. Lichtschranken können zur Anwesenheitskontrolle oder Zählung genutzt werden. Die Vorteile von Lichtschranken sind die großen Entfernungen, die überwacht werden können, wenn die Umgebung dies zulässt, und die Verschleißfreiheit, die durch die berührungslose Arbeitsweise gegeben ist. *Weg-* oder *Winkelmesser* sind ebenfalls optische Sensoren, die ähnlich einer Lichtschranke mittels einer Skala aus lichtreflektierenden und lichtabsorbierenden Feldern aus dem Signalwechsel den Weg oder Winkel bestimmen können. Sie kommen beispielsweise bei Achswinkelmessung von Fördermitteln zum Einsatz. *Magnetische und induktive Sensoren* dienen der Anwesenheits- oder Lagerkontrolle von metallischen Objekten. *Ultraschall* kommt beispielsweise zur Kollisionsvermeidung fahrerloser Transportfahrzeuge zum Einsatz.

Aktoren Aktoren sind Antriebe oder Ventile, welche die eigentliche Bewegung innerhalb eines Materialflussprozesses ausführen. Es existieren unterschiedliche Varianten, u. a. vom einfachen Rollenfördererantrieb bis zum Bestandteil mehrachsiger Roboter. [tBF08] teilt Aktoren nach den geforderten Bewegungsabläufen in *Bewegungs-*, *Stell-* und *Positionierantriebe* ein. Bewegungsantriebe dienen dem kontinuierlichen Betrieb über längere Zeitintervalle. Sie kommen beispielsweise in Stetigförderern zum Einsatz. Stellantriebe werden für den diskontinuierlichen Betrieb mit festen Stellpositionen unter anderem für Hubeinrichtungen eingesetzt. Positionierantriebe werden im diskontinuierlichen Betrieb mittels stetig über die Förderstrecke verteilten Positionen genutzt und sind beispielsweise als Antrieb für eine Fahrtrichtung in Regalbediengeräten zu finden.

Zu jedem Antrieb gehört ein Stellglied, das die Energiezufuhr zur Antriebseinheit steuert und oftmals die Signalverarbeitung übernimmt. Es nimmt Steuersignale entgegen und überwacht die Antriebseinheit auf Überlast und korrekte Einhaltung der Sollwerte. Die Antriebseinheit erzeugt die mechanische Bewegung mittels zugeführter Energie. Unterscheiden wird zwischen translatorischen (Hubzylinder) und rotarischen (Motoren) Antrieben. Die Energie kann in elektrischer (Drehstrom-/Gleichstrom-/Schritt-/Linearmotoren) oder fluidischer Form (hydraulisch, pneumatisch) zugeführt werden.

Automatische Identifikation Um innerhalb eines Materialflusssystems objektbezogene Entscheidungen zu treffen, ist es notwendig, über eine eindeutige, zuverlässige und schnelle Identifikation der zu transportierenden Güter zu verfügen. Die häufigsten Methoden der Identifikation sind *Barcodes* oder *Radiofrequenzidentifikation (RFID)*. In der Versuchsanlage des Lehrstuhls kommt an den Identifizierungspunkten RFID zum Einsatz. [tBF08] defininiert *RFID* als „eine Technologie zur berührungslosen Erfassung und Übertragung binär kodierter Daten mittels induktiver oder elektromagnetischer Wellen“. Das System besteht aus einer Schreib-/Leseeinheit und einem Transponder, auch *Tag* genannt, und ist im direkten Gegensatz zum Barcode widerstandsfähig gegen Verschmutzung und benötigt keinen Sichtkontakt. Ein Transponder kann mit einem nichtbeschreibbaren Speicher ausgestattet sein, in den irreversibel ein Identifikator gespeichert werden kann. Transponder können zusätzlich einen beschreibbaren Speicher besitzen, der je nach Ausführung zwischen einem Bit und mehreren Kilobyte groß ist. RFID-Systeme können grundsätzlich verschiedene Frequenzbereiche eingeteilt werden, die unterschiedliche Einsatzzwecke erfüllen. In der Versuchsanlage kommen Transponder im *Ultra-High Frequency*-Bereich zum Einsatz. Solche Transponder sind häufig als sogenannte *Smart Label*-Aufkleber im Einsatz und haben eine Reichweite von ca. 1 m ([tBF08]).

4.2 Ist-Situation: aktuelle Logistiksysteme

Im Folgenden werden die derzeitigen Entwicklungen moderner intralogistischer Systeme beschrieben. Zunächst wird in Unterkapitel 4.3 der Stand der Technik heutiger Materialflusssysteme erläutert sowie bekannte Probleme im Bezug auf wachsende Marktvolatilität und -vielfalt aufgezeigt. Dies motiviert die Forschung nach neuen, flexibleren Technologien zum Zwecke der Modularisierung und Serviceorientierung zukünftiger Materialflusssysteme. Im Unterkapitel 4.4 wird ein dezentrales Steuerungsverfahren als Ansatz in dieser Richtung vorgestellt und in Unterkapitel 4.5 werden anschließend *Smart Neighbourhood Module*, also im laufenden Betrieb austauschbare Fördertechnikkomponenten, als Zukunftsvision einer dezentral konstruierten Förderanlage präsentiert. Dazu werden anhand eines Beispiels die möglichen positiven Auswirkungen auf Planungs- und Ausfallsicherheit, Effizienz sowie Zukunftssicherheit eines Smart-Neighbourhood-fähigen Materialflusssystems herausgestellt.

Der folgende Abschnitt basiert auf Beobachtungen aus dem Buch [AAK+08]. Aufgrund hoher Entwicklungs- und Konstruktionskosten werden intralogistische Systeme üblicherweise für den Betrieb über lange Zeiträume ausgelegt, wodurch die laufenden Betriebskosten die einmaligen Anschaffungskosten übersteigen. Ein wichtiges Ziel moderner Materialflussplanung ist daher die Verringerung von Betriebskosten. Schon in der Entwicklungsphase werden Fördersysteme für effiziente Prozessabläufe und hohe Betriebsmittelnutzungsgrade ausgelegt und dabei auf den jeweiligen Anwendungsfall hin spezialisiert. Dazu werden u. a. verschiedene Prozess- und Arbeitsmittelvarianten analysiert, korrekte Anlagendimensionierungen errechnet und alle Bestandteile eines Systems durch Simulation aufeinander abgestimmt. Für die eigentliche Konstruktion der Anlage müssen dann einerseits eine Vielzahl von mechanischen Bauteilen aufeinander abgestimmt werden, andererseits müssen alle Steuerungsaufgaben durch speziell anzufertigende Software abgedeckt werden. Mit wachsender Anlagengröße steigt auch die Zahl der benötigten Softwarekomponenten. Diese werden für komplexe Anwendungen, etwa Wegfindungsalgorithmen für Fördergüter, in der Feldebene eingesetzt und müssen an die jeweiligen Fördertechnikkomponenten angepasst werden, so dass deren Wartung durch wachsende Komplexität oft aufwändig ist. Jede Änderung einer solchen Anlage erzeugt erhebliche Kosten und ist daher im Sinne der Betriebskostenminimierung nach Möglichkeit zu vermeiden. Für lange Zeit wurde der Lebenszyklus von Produkten deshalb durch stark spezialisierte und damit unflexible Materialflusssysteme bestimmt. Die Folge sind lange Laufzeiten, feste Losgrößen in der Produktion und vergleichsweise langsame Marktveränderungen.

In neuerer Zeit unterliegt die Situation moderner Produktionsunternehmen jedoch einem massiven Wandel. Im Zuge der zunehmenden Individualisierung der Gesellschaft nimmt die Nachfrage nach immer neuen Produktvarianten und Innovationen stetig zu. Der Umschlag steigt, die Produktionszyklen verkürzen sich. In manchen Bereichen, etwa der Unterhaltungselektronik, veralten Produkte teilweise in Monaten. Das Internet unterstützt diesen Prozess durch neue, individuellere Vertriebswege. Durch den Onlinevertrieb hat sich insbesondere die mögliche Breite des Warenangebots vervielfacht, da keine natürlichen Begrenzungen der Angebotsbreite wie im klassischen Einzelhandel vorhanden sind; Ein Effekt, den der Journalist Chris Anderson als „The Long Tail“[And06] bezeichnet. Um in dieser Marktsituation wettbewerbsfähig zu bleiben, müssen statische Produktions- und Materialflusssysteme immer häufiger durch kostenintensive Modifikationen an neue Produktionsprozesse angepasst werden. Die entstehenden Kosten steigern letztlich den Preis und gefährden damit die Wettbewerbsfähigkeit von Unternehmen.

Laut [tNF+11] motiviert die derzeitige Alternativlosigkeit zu statischer Anlagenplanung die Suche nach neuen, flexibleren Methoden zur Konstruktion intralogistischer Systeme. Dabei kann ein modularer Ansatz viele weitere Probleme lösen:

Durch fortschreitende Modifikationen bestehender Anlagen werden zahllose Anpassungen und Korrekturen an den beteiligten Softwarekomponenten erforderlich. Diese Änderungen werden laut [tNF+11] nur mangelhaft dokumentiert, so dass Wartung und sukzessive Erweiterungen zunehmend schwieriger werden. Ein modular geplanter Ansatz kann mögliche Änderungen schon in der Planungsphase antizipieren und automatische Dokumentationsmechanismen bereitstellen.

Weiterhin können Ausfälle einzelner Komponenten eines automatischen Materialflusssystems den Stillstand großer Teile der Anlage zur Folge haben. Insbesondere beim Austausch von Modulen kann ein flexibles System mit eigenen Automatisierungseinrichtungen die Kosten solcher Ausfälle zum Teil erheblich vermindern, da die Funktionsfähigkeit der anderen Module erhalten bleibt.

Die Flexibilität einer modularen Anlage eröffnet neue Spielräume in der Planung moderner Materialflusssysteme. Beispielsweise kann schneller auf neue Qualitätssicherungsrichtlinien oder Nachfrageschwankungen reagiert werden.

4.3 Grundlagen für die Zukunftsvision

In diesem Unterkapitel werden grundlegende Techniken vorgestellt, die eine flexible Gestaltung von Logistiksystemen ermöglichen. Es soll die Möglichkeit diskutiert werden, modulare Systeme zu gestalten, die dezentral arbeiten können. Dazu soll jedes Modul eine eigene Modulintelligenz haben, d. h. einen eigenen Prozessor und Kommunikationsmöglichkeiten. Im weiteren Verlauf stellt dieses Unterkapitel die Grundlagen für das Steuerungssysstem Paket Royale (Unterkapitel 4.4) und die Zukunftsvision der Smart Neighbourhood Module (Unterkapitel 4.5) vor. Zunächst wird in Abschnitt 4.3.1 auf einsetzbare Rechnerressourcen und in Abschnitt 4.3.2 auf grundlegende Kommunikationsmethoden eingegangen. Anschließend wird das Konzept einer Serviceorientierte Softwarearchitektur (SOA) und eine mögliche Umsetzung dieses Konzepts in Abschnitt 4.3.3 vorgestellt.

4.3.1 Einsetzbare Rechnerressourcen

Die Rechnerressourcen sind ein wesentlicher Aspekt von jedem Projekt in der Automatisierungstechnik. Von der Hardware sind mehrere Seiten des Projekts abhängig: Preis, Leistung, Flexibilität. Um das optimale Preis-Leistung-Verhältnis bei der Hardwareanschaffung zu erziehen, ist es notwendig, dass die Alternativen bekannt sind und die benötigten Geräte ausgewählt werden.

In dieser Projektgruppe wird eine neue Cloud-Infrastruktur entwickelt. Dafür sind herkömmliche PC-kompatible Rechner geeignet. Allerdings passen Büro-PCs nicht zu den Arbeitsbedingungen in einer Industrieanlage [Gee01]. Diese PCs sind für die private Nutzung ausgelegt und stellen daher keine große Anforderungen an die Robustheit und die Zuverlässigkeit. Für den industriellen Einsatz sind robustere Varianten notwendig, die als Industrie-PCs (IPCs) bezeichnet

werden. Diese IPCs sind PC-kompatibel, aber stabiler und zuverlässiger gebaut, damit sie einen Industrieprozess in einer rauen Umgebung steuern und protokollieren können.

Auch wenn die IPCs über große Robustheit verfügen, haben sie Nachteile für den Einsatz in die *Smart Neighbourhood Module*-Architektur, die in der Projektgruppe entwickelt wird. Vor allem sind diese wegen ihrer hohen Qualität sehr teuer. Zusätzlich, sind sie nicht klein genug, um in autonomen Modulen eingebettet zu werden. Dies ist besonders wichtig für den gewünschten Einsatz in die Miniaturfördertechnik. Aus diesen Gründen, ist nach einer Alternative gesucht worden. Diese Alternative bieten verschiedene Hardwareboards, welche eine genügende Rechenleistung und eine kompakte Form haben. Die betrachteten Boards basieren auf der ARM-Rechnerarchitektur und verfügen über ein ausgewogenes Preis-Leistung-Verhältnis.

Diese beide Arten von Hardwareressourcen werden untersucht und die Ergebnisse werden im Abschnitt 6.1 zusammengefasst.

4.3.2 Kommunikation

Kommunikation stammt aus dem Lateinischen *communicare* und bedeutet „teilen, mitteilen, teilnehmen lassen“. Der Datenaustausch zwischen zwei oder mehr Kommunikationsteilnehmern findet i.d.R. durch die Übertragung von Signalen statt, die kabellos oder kabelgebunden ausgetauscht werden. Signale sind die physikalische Repräsentation von Daten. Die Datenumsetzung auf Signale und umgekehrt passiert auf physikalischer Ebene [Sch03].

Im Folgenden werden verschiedene Kommunikationstechnologien vorgestellt und bezüglich ihrer Einsetzbarkeit für die Projektgruppe gegenübergestellt.

Unter einem Rechnernetz versteht man eine Gruppe von Rechnern, die untereinander verbunden sind, um miteinander zu kommunizieren oder gemeinsame Ressourcen zu nutzen. Je nachdem, ob ein Rechnernetz sich auf einem begrenzten Raum beschränkt oder es sich weltweit erstreckt, spricht man von lokalen oder globalen Rechnernetzen. Im Folgenden werden verschiedene Rechnernetze vorgestellt. Ein *Local Area Network (LAN)* dient zur Verbindung von Rechnern und Servern in einem räumlich begrenzten Gebiet über Leitungen. Die Übertragungsraten liegen im Bereich von 1 bis 1.000 $\frac{\text{MBit}}{\text{s}}$. Ein *Metropolitan Area Network (MAN)* ist ein spezielles Rechnernetz, das für unterschiedlichste Übertragungsdienste wie Sprache, Daten, usw. ausgelegt ist. Ein MAN kann eine Ausdehnung bis zu 100 km haben und die Übertragungsraten liegen im Bereich von 100 $\frac{\text{MBit}}{\text{s}}$ bis 1 $\frac{\text{GBit}}{\text{s}}$. Ein *Wide Area Network (WAN)* verbindet einzelne Stationen innerhalb eines Landes oder in mehreren Ländern. Es erstreckt sich im Unterschied zu einem LAN oder MAN über einen sehr großen geografischen Bereich. Die Übertragungsgeschwindigkeiten liegen zwischen 64 $\frac{\text{kBit}}{\text{s}}$ und 600 $\frac{\text{MBit}}{\text{s}}$. Ein *Global Area Network (GAN)* verbindet Rechner weltweit. Es ist die logische Zusammenfassung verschiedener LANs oder MANs. Oft wird bei einem GAN Satelliten- oder Glasfaserübertragung eingesetzt.

Die Verbindungsstruktur der Teilnehmers in einem Netz wird als Netztopologie bezeichnet. Für spezielle Netztopologien gibt es geeignete Protokolle und Techniken, mit denen die Kommunikation der Teilnehmer untereinander gewährleistet wird. Typische Netztopologien sind Stern, Bus, (Token-) Ring, Baum und vermaschtes Netz. Bei der Stern-Topologie werden alle Netzteilnehmer über eigene Leitung mit einer zentralen Station verbunden. Die Bus-Topologie besteht aus mehreren Teilnehmern, die an eine gemeinsame Leitung (Bus) angeschlossen sind. Werden beide Enden miteinander verbunden, so heißt die entstehende Netztopologie (Token-)Ring. Die

Baum-Topologie entsteht indem mehrere Stern-Topologien miteinander verbunden werden. In einem vermaschten Netz sind alle Netzteilnehmer beliebig miteinander verbunden.

Es ist in vielen Fällen nicht möglich, alle potentiellen Teilnehmer an ein einziges Netz anzuschließen. Deswegen werden unterschiedliche Netze durch Vermittlungsrechner untereinander verbunden.

Kabelgebundene Kommunikation

Die physikalische Verbindung zwischen zwei Rechnern kann beispielsweise durch verdrillte Kupferdrähte realisiert werden. Nicht abgeschirmte verdrillte Kabel sind die günstigste und einfachste Verdrahtungsmöglichkeit. Sie werden im Telephoniebereich eingesetzt, z. B. bei *Integrated Services Digital Network (ISDN)*, mit einer Datenübertragungsrate von 2 Mbit/s. Bei *Digital Subscriber Line (DSL)*-Technologie ist die Geschwindigkeit von über 25 Mbit/s möglich. Koaxialkabel bestehen dagegen aus einem isolierten Kupferdraht, der mit dem zweiten Leiter ummantelt ist und somit gegen Störungen sehr robust ist. Mit dem Kabel lassen sich Übertragungsraten von 100 MBit/s erzielen. Glasfaserkabel zeichnen sich durch Unempfindlichkeit gegenüber äußeren Störungen aus und besitzen eine höchstmögliche Übertragungsrate. Diese Technologie ist jedoch teuer und erfordert einen höheren Aufwand für Sender und Empfänger. Die Übertragungsrate liegt ca. bei 1 Gbit/s, im Testbetrieb sogar 2,5 GBit/s [GS08].

USB - Universal Serial Bus USB ist eine Technologie, die genutzt wird, um verschiedene Geräte an den Rechner anzuschließen, wie z. B. Speichermedien, Eingabegeräte oder Drucker. USB-Geräte sind in der Lage Strom über den Bus zu beziehen. Somit benötigen sie in einigen Fällen kein eigenes Netzteil. Aus Sicht des Anwenders besteht der Vorteil des USB in der leichten Anwendbarkeit, in der zuverlässigen Datenübertragung, in der Flexibilität, in den geringen Kosten und im niedrigen Stromverbrauch [Axe06].

Ein USB-Kabel, das zwei miteinander kommunizierende Geräte verbindet, kann laut Spezifikation bis zu fünf Meter lang sein. Mit Hilfe von *Hubs* können Verbindungen über maximal 30 Meter hergestellt werden ([Axe06]). Ein *Hub* ist ein Gerät, das mehrere Geräte in einem Rechnernetz verbindet. Eine typische Konfiguration umfasst einen einzigen Host-PC mit mehreren Geräten, die über USB-Kabel miteinander verbunden sind. Der Host-PC verfügt über einen integrierten Hub, der typischerweise zwei oder mehr USB-Ports enthält. Durch diesen Hub können bis zu 126 Geräte miteinander verbunden werden ([Sta03]). Bei einer fehlerhaften Datenübertragung wird der Sender benachrichtigt, so dass die Daten erneut übertragen werden können.

Aktuell entsprechen die meisten Geräte dem USB-Standard in Version 1.0 oder 2.0. Mittlerweile gibt es die ersten Geräte mit dem Nachfolger USB 3.0, der eine deutlich höhere Geschwindigkeit bei der Datenübertragung ermöglicht, allerdings mehr Energie benötigt. Generell ist USB 3.0 abwärtskompatibel, alte Hardware kann deshalb weiter verwendet werden [tel10].

Ethernet Ethernet ist eine Technologie für lokale Netze. In einem Ethernet können verschiedene Netzwerkprotokollstapel eingesetzt werden, wie z. B. Transmission Control Protocol/Internet Protocol (TCP/IP) oder Internetwork Packet Exchange/Sequenced Packet Exchange (IPX/SPX).

Das ursprüngliche Ethernet nutzt ein Koaxialkabel als Übertragungsmedium. Die maximale Länge des gesamten Netzes liegt bei 2.500 m und enthält maximal bis zu 1.024 Knoten. Mittlerweile

ist Ethernet für verdrillte Kupfer- und auch Glasfaserkabel standardisiert, die das Koaxialkabel in den meisten Anwendungen ersetzt haben. Vor allem zur Überbrückung von längeren Strecken wird auf das Glasfaserkabel zurückgegriffen.

Das Ethernet ist ein paketvermittelndes Netzwerk. Die Daten werden in Pakete aufgeteilt. Diese Pakete werden *Frames* genannt. In einem Frame werden neben den Daten auch die Zieladresse, die Quelladresse und weitere Steuerungsinformationen zusammengefasst. Als Adressen dienen sogenannte MAC-Adressen, also die Hardware-Adresse, die zur eindeutigen Identifikation des Geräts im Netzwerk dient. Die MAC-Adresse wird i.d.R. fest in den Chip des Netzwerkadapters eingebrannt.

Kabellose Kommunikation

Im folgenden werden drei Beispiele für drahtlose lokale Netze vorgestellt. Diese sind flexibel und kostengünstig in Büroumgebungen, Privatwohnungen oder im industriellen Bereich einsetzbar. Es existieren zwei grundlegende Übertragungstechniken: Eine Technik verwendet infrarotes Licht und die andere nutzt Funkwellen.

Drahtlose Kommunikation lässt sich auf zwei Arten aufbauen. Eine Art ist ein sogenanntes *Infrastrukturnetz*. Die Kommunikation findet in einem solchen Netz nur zwischen den drahtlos angebundenen Endgeräten und einem Zugangspunkt (*access point*, Basisstation) statt. Eine direkte Kommunikation zwischen zwei Endgeräten ist nicht vorgesehen. Das Infrastrukturnetz bietet nicht nur einen Zugang zu anderen Netzen, sondern kann auch Daten zwischen verschiedenen Funknetzen weiterleiten und den Medienzugriff steuern. Ein Beispiel für solches Netz ist WLAN. Eine andere Art, die Geräte drahtlos kommunizieren zu lassen, ist *Ad-hoc-Netzwerk*. Dies benötigen keinerlei Infrastruktur um zu arbeiten. Jedes Endgerät kann mit einem anderem direkt kommunizieren. Die Kommunikation ist sofort möglich, sobald zwei oder mehr Geräte im gegenseitigen Übertragungsbereich liegen. Beispielsweise lässt sich ein Ad-hoc-Netzwerk mithilfe von Bluetooth aufbauen.

Infrarot Die *Infrared Data Association* spezifiziert Standards für die optische drahtlose Punkt-zu-Punkt Datenübertragung mittels infrarotem Licht, um Kommunikation zwischen unabhängigen Geräten zu ermöglichen. Infrarotübertragung nutzt entweder diffuses Licht, welches an Wänden oder Hindernissen reflektieren wird und somit auf Umwegen zum Empfänger gelangt, oder gerichtetes Licht, das eine direkte Sichtverbindung zwischen dem Sender und dem Empfänger erfordert [Sch03].

Der Vorteil liegt in der einfachen und sehr günstigen Sender- und Empfängerausrüstung. Nachteilig sind die niedrige Übertragungsrate von 4 $^{\text{MBit}}/_{\text{s}}$, eine Sichtverbindung und Störung durch Fremdlichteinfall. Das sind Gründe, warum Infrarot für komplexere Kommunikationsanwendungen selten eingesetzt wird.

Bluetooth Der Bluetooth-Standard spezifiziert eine drahtlose LAN-Technologie für kurze Entfernungen. Bluetooth ist ein lizenzfreies Nahbereichsfunkverfahren zur kabellosen Kommunikation. Die Entwicklung von Bluetooth geht auf eine Initiative der sogenannten Bluetooth Special Interest Group (Bluetooth SIG) von 1998 zurück, der eine große Zahl Hersteller angehört [Sch03].

Bluetooth basiert auf einer Master/Slave-Architektur, bei der jegliche Kommunikation mit Hilfe des Masters stattfindet. Bluetooth arbeitet im 2,4-GHz-ISM-Frequenzband auf 79 Kanälen, die jeweils 1 MHz umfassen und abwechselnd zur Datenübertragung genutzt werden. Das ISM-Frequenzband (*Industrial, Scientific, Medicine*) kann für Anwendungen in Industrie, Wissenschaft und Medizin frei und ohne Lizenz benutzt werden. Bei Bluetooth finden dabei 1.600 Wechsel pro Sekunde statt. An den Bandgrenzen sind 2 bzw. 3,5 MHz freigelassen, damit keine Störungen benachbarter Systeme auftreten. Die Datenübertragungsrate beträgt abhängig von der Betriebsart 58 $^{kBit}/_{s}$ bis maximal 2,2 $^{MBit}/_{s}$.

Derzeit existieren zwei Möglichkeiten mithilfe von Bluetooth ein Netzwerk aufzubauen: Piconet und Scatternet. Das *Piconet* ist eine Ansammlung von mobilen Endgeräten, die ein gemeinsames, örtlich begrenztes Netzwerk bilden. Ein Piconet beginnt mit zwei verbundenen Geräten und kann bis auf acht Endgeräte ausgedehnt werden. Ein Gerät operiert als Master und alle anderen Geräte als Slave. Die einzelnen Slaves haben untereinander keine Verbindung, sondern sind nur mit dem Master verbunden. Kommunikation unter Slaves ist nur möglich wenn die Pakete vom Master weitergeleitet werden. Wenn sich ein Gerät im Senderadius von zwei unterschiedlichen Piconets befindet, kann ein sogenanntes *Scatternet* aufgebaut werden. Die Unterscheidung zwischen Master und Slave ist hier auch eindeutig geregelt. Wenn ein Gerät die Master-Rolle in einem Piconet übernommen hat, kann es nicht gleichzeitig Master für das andere Piconet sein. Dafür ist es aber möglich, dass dieser Master gleichzeitig Slave im anderen Piconet ist. Es ist auch möglich, dass ein Slave -Gerät Slave für zwei verschiedene Piconets ist. Dieses Gerät bildet eine „Brücke“ (*Gateway*) zwischen den Netzen und leitet Datenpakete zwischen diesen weiter. Durch dieses Umschalten von Geräten verringert sich die Geschwindigkeit, mit der in den Netzen kommuniziert werden kann.

WLAN Wireless Lan (WLAN) ist eine weitere kabellose Kommunikationstechnologie. WLAN unterstützt, von den bisher vorgestellten drahtlosen Technologien, die größte räumliche Ausdehnung. Seit 1997 gibt es die Ethernet-Variante IEEE 802.11, die eine verbindliche drahtlose Schnittstelle spezifiziert. Die zur Übertragung benötigte Zeit ist länger als im drahtgebundenen LAN.

Bereits mit zwei WLAN-fähigen Geräten, sogenannten Stationen, lässt sich ein drahtloses Netz aufbauen. Bei der Einrichtung sind keine weiteren Elemente erforderlich. Die Stationen kommunizieren direkt über den WLAN-Adapter. Jede Station bildet mit seiner Netzwerkkarte eine Funkzelle. Solange sich die Zellen der Stationen überschneiden, ist eine Kommunikation zwischen den Stationen möglich.

Ist die Reichweite einer Zelle zu gering, lässt sie sich mit einem Zugangspunkt erweitern. Mittels zweier Zugangspunkte und Richtfunkantennen lässt sich auch die Reichweite eines drahtlosen Netzwerkes erhöhen. Mit WLAN besteht die Möglichkeit, Bereiche zu verbinden, die mit der herkömmlichen Verkabelung nicht erreicht werden können.

Protokolle

In der Informatik ist ein *Protokoll* eine Vereinbarung, nach der die Verbindung, Kommunikation oder Datenübertragung zwischen zwei Parteien besteht. Protokolle können durch Hardware, Software oder eine Kombination von beiden implementiert werden [Sch08].

Modbus Das *Modbus-Protokoll* ist ein Kommunikationsprotokoll, das auf einer Master/Slave-Architektur basiert. Es wurde 1979 vom US-amerikanischen SPS-Hersteller Gould-Modicon entwickelt und gilt zusammen mit einer seriellen Datenübertragung als eines der ersten Feldbussysteme. Das offene Protokoll, über das Steuerungen und Geräte unterschiedlicher Hersteller kommunizieren können, ist ein Quasi-Industriestandard. Es ist effektiv, einfach zu implementieren und sowohl für Anbieter von Geräten als auch für Anwender frei verfügbar. Als reines Anwendungsprotokoll ist Modbus unabhängig vom Übertragungsmedium: So nutzt Modbus-TCP dieselben Dienste und dasselbe Objektmodell wie die ursprünglichen Modbus-Varianten Modbus ASCII, Modbus RTU (RTU: Remote Terminal Unit, asynchrone Übertragung) oder Modbus Plus (Token Passing), verwendet aber Ethernet als Transportprotokoll und kapselt die Nachrichten in TCP/IP-Paketen. Ein Master kann mit einem oder mehreren Slaves kommunizieren. Nur der vom Master explizit angesprochene Slave darf Daten an den Master zurücksenden. Slaves kommunizieren nicht untereinander [HMS06].

Im Folgenden wird das Master/Slave-Prinzip von Modbus genauer beschrieben. An den Bus ist genau ein Master angeschlossen. Ein oder mehrere Slaves können an denselben seriellen Bus angeschlossen werden. Nur der Master ist zur Aufnahme der Kommunikation berechtigt, d. h. zum Senden von Anfragen an die Slaves. Der Master kann nur einen Vorgang gleichzeitig initiieren. Die Adressierung kann durch den Master an jeden Slave einzeln (Unicast) oder an alle Slaves gleichzeitig (Broadcast) erfolgen. Die Slaves können nur auf vom Master empfangene Anfragen antworten. Außerdem können die Slaves keine Kommunikation initiieren, weder mit dem Master noch mit einem anderen Slave. Wenn ein Fehler beim Empfang einer Meldung auftritt oder der Slave die geforderte Aktion nicht ausführen kann, erzeugen die Slaves eine Fehlermeldung und senden sie als Antwort an den Master [Sch08].

4.3.3 SOA und DPWS

In diesem Abschnitt wird das Konzept einer Serviceorientierte Architektur (SOA) und Webservices als Umsetzung dieses Konzepts vorgestellt. Dabei wird insbesondere das Webservice-Profil DPWS (*Devices Profile for Web Services*) erläutert. Unter einem Profil ist eine Kombination von Spezifikationen zu verstehen.

SOA

Eine Serviceorientierte Architektur ist ein Programmierkonzept, das als logisch nächster Schritt in einer Kette von Programmierkonzepten gesehen werden kann. Angefangen bei prozeduraler Programmierung über objektorientierte Programmierung folgt das Konzept der SOA [Mel10].

Bei einer SOA handelt es sich um ein abstraktes Konzept einer Softwareachitektur und nicht um eine konkrete Technik. Es wird vielmehr ein Bild der Wirklichkeit wiedergegeben [Mel10]. Wie bereits der Name „SOA“ andeutet, arbeitet ein serviceorientiertes System mit Diensten (*Services*). Ein Dienst ist dabei eine bestimmte Funktionalität. Anwendungen werden in Dienste zerlegt, die miteinander interagieren können. So lassen sich einzelne Komponenten einer Anwendung separat betrachten, aber auch zu der gesamten Anwendung zusammensetzen [Mel10]. SOA sind plattform- und programmiersprachenunabhängig. Um eine weite Akzeptanz zu ermöglichen, nutzt SOA offene Standards, wie z. B. TCP/IP oder XML [BBG06]. Ein weiteres Merkmal einer SOA ist die *lose Kopplung*. Lose Kopplung bedeutet, dass ein Dienst bei Bedarf von einer

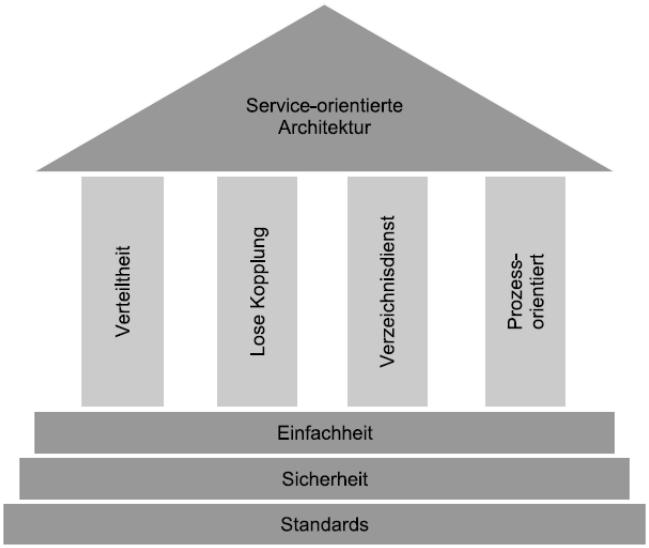


Abbildung 4.1: SOA Tempel [Mel10]

Anwendung oder einem anderen Dienst gesucht, gefunden und eingebunden werden kann[Mel10]. Dies kann zur Laufzeit geschehen. Um Dienste in einem Netzwerk finden zu können, kann ein Verzeichnisdienst oder eine Datenbank der verfügbaren Dienste, auch *Registry* genannt, angelegt werden, bei dem sich die Dienste anmelden. Bei diesem Verzeichnisdienst kann nach bestimmten Methoden gesucht werden.

Die Abbildung 4.1 fasst das Grundkonzept einer SOA zusammen. Als Fundament der SOA dienen offene Standards, Sicherheit und Einfachheit der Architektur. Getragen wird das Konzept der Serviceorientierten Architektur durch die verteilten Dienste, die lose Kopplung der Dienste, die Verzeichnisdienste und die Prozessorientierung.

Definitionen Es existiert keine einheitliche Definition einer SOA. Hier sollen daher Beispiele für eine solche gegeben werden.

„Serviceorientierte Architektur (SOA) ist ein Paradigma für die Strukturierung und Nutzung verteilter Funktionalität, die von unterschiedlichen Besitzern verantwortet wird.“ [MLM+06]

„Unter einer SOA versteht man eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wieder verwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachenunabhängige Nutzung und Wiederverwendung ermöglicht.“ [Mel10]

„Eine SOA ist eine mehrschichtige, verteilte Informationssystem (IS)-Architektur, die Teile von Applikationen für eine vereinfachte Prozessintegration als geschäftsorientierte Services kapselt und dabei bestimmte Designprinzipien berücksichtigt.“ [Heu07]

„Ein Service stellt ein abstraktes Software-Element bzw. eine Schnittstelle dar, die anderen Applikationen über ein Netzwerk einen standardisierten Zugriff auf Anwendungsfunktionen anbietet.“ [Heu07]

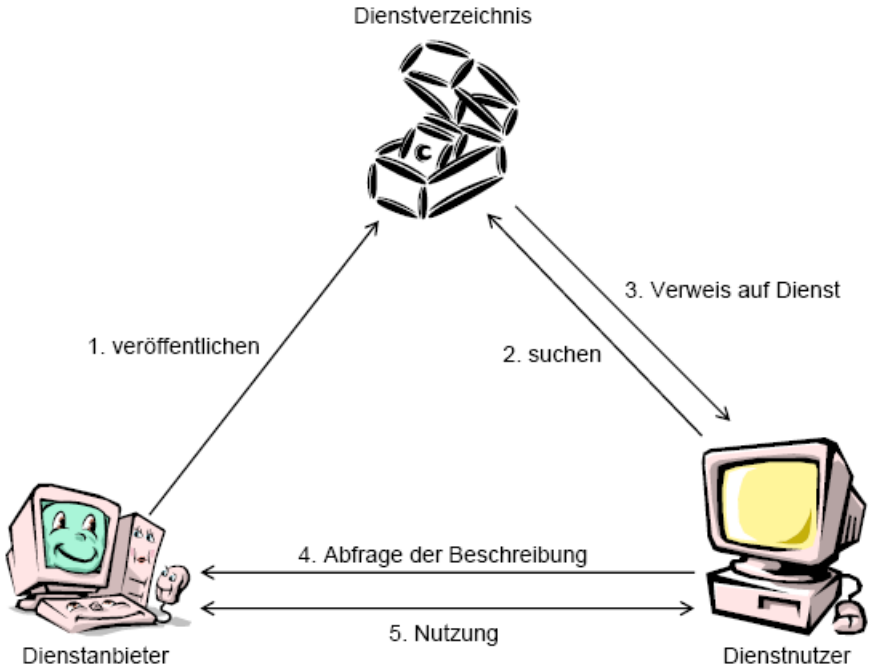


Abbildung 4.2: Rollen in einer SOA [Mel10]

Dienste und ihre Rollen Ein Dienst wird durch eine Schnittstelle eindeutig beschrieben, in der die Funktionen des jeweiligen Dienstes definiert werden [BBG06]. Diese Schnittstelle liegt in maschinenlesbarer Form vor, wobei dabei die Nutzung offener Standards wichtig ist [Mel10]. Schnittstelle und Implementierung der Dienste sind voneinander unabhängig. Die Schnittstellenbeschreibung wird genutzt, um die Dienste im Netzwerk zu integrieren, anzubieten, zu finden und ihre Funktionalitäten zu nutzen. Dienste können sich des Weiteren bei anderen Diensten anmelden, um Statusänderungen bei deren Eintreten zu empfangen [BBG06]. Dadurch, dass gekapselte Dienste genutzt werden, ist das Prinzip der Wiederverwendbarkeit umgesetzt [Mel10].

Dienste können in einer SOA drei verschiedene Rollen einnehmen: Sie können Anbieter, Nutzer oder Vermittler eines Dienstes sein. Abbildung 4.2 zeigt das Zusammenspiel dieser drei Rollen. Der Dienstanbieter stellt einen Dienst bereit, der von einem Dienstnutzer genutzt werden kann. Als Vermittler dient dabei beispielsweise ein Dienstverzeichnis. Jeder Dienstanbieter registriert sich bei diesem, damit der Dienst vom Dienstnutzer über das Dienstverzeichnis gefunden werden kann. So bekommt der Dienstnutzer bei einer Suche von dem Dienstverzeichnis eine Referenz auf den Dienst, über die der Dienstnutzer auf den Anbieter zugreifen kann.

Webservices und DPWS

Webservices sind mögliche Umsetzungen einer Serviceorientierten Architektur. Sie werden vom *World Wide Web Consortium* (W3C) als Technologie zur Maschine-Maschine-Interaktion über ein Netzwerk definiert [BHM+04]. Es existieren verschiedene modulare Spezifikationen zu Webservices. Diese müssen jedoch nicht alle für die Nutzung von Webservices umgesetzt werden. Es kann eine einzelne Spezifikation oder eine Kombination aus mehreren dieser Spezifikationen

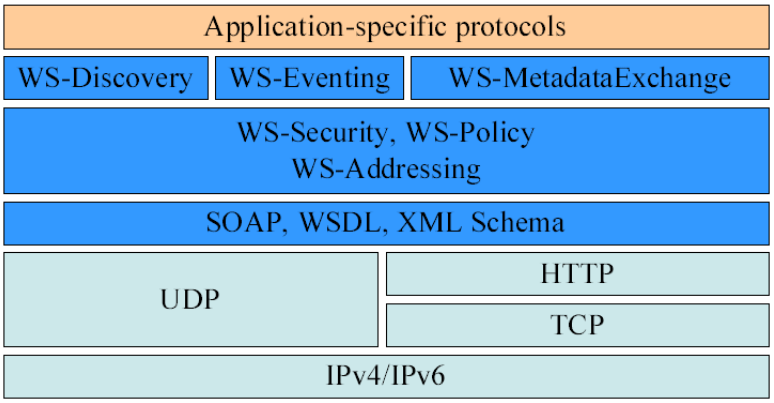


Abbildung 4.3: DPWS [BBG06]

genutzt werden. Es gibt z. B. Spezifikationen für das Auffinden (WS-Discovery [MK09]) oder Beschreiben (Web Service Description Language [CCM+01]) von Webservices.

Zur Kommunikation von Webservices werden SOAP-Nachrichten genutzt [GHM+07]. SOAP ist ein XML-basiertes Netzwerkprotokoll, das den entfernten Prozeduraufruf als Programmiermodell umsetzt. Dabei wird die Nachricht über ein beliebiges Transportprotokol, wie z. B. HTTP, gesendet. Die Beschreibung eines Webservices erfolgt mit Hilfe der *Web Service Description Language* (WSDL), die eine auf XML basierende Beschreibungssprache ist.

Devices Profile for Web Services Das *Devices Profile for Web Services* (DPWS) ist ein Profil, in dem verschiedene Webservice-Spezifikationen kombiniert werden. Zusätzlich werden Bereiche der Spezifikationen durch das Profil eingeschränkt bzw. erweitert. Zu den zusammengeführten Spezifikationen gehören Webservice-Spezifikationen aus den Bereichen des sicheren Nachrichtenverkehrs, des dynamischen Auffindens von Webservices, der Beschreibung der Webservices und der Verarbeitung von Ereignissen [DM09]. Dadurch soll DPWS den Einsatz von Webservices auf Geräten mit eingeschränkten Ressourcen ermöglichen [DM09]. Erstmals wird DPWS im Mai 2004 publiziert. Im Oktober 2005 wird eine überarbeitete Version veröffentlicht [BBG06]. Das aktuelle Profil ist vom Juli 2009. Es ist als Standard der *Organization for the Advancement of Structured Information Standards* (OASIS) publiziert. OASIS ist eine internationale Organisation, die sich mit der Weiterentwicklung von E-Buisness und Webservice-Standards beschäftigt (vgl. [OAS11]). Die Abbildung 4.3 zeigt den Aufbau des *Devices Profile for Web Services* und die darin kombinierten Webservice-Spezifikationen. Dazu gehören:

WS-Addressing: Eine Spezifikation, die ein transportunabhängiges Verfahren zur Adressierung von Webservices und Nachrichten beschreibt. Dabei definiert es die Identifikation des Endpunktes, also der Adresse eines Webservices (Endpunktreferenz) und die sichere Ende-zu-Ende Zustellung von Nachrichten. [GHR06]

WS-Discovery: Ein Standard, der das Auffinden von Webservices in einem Netzwerk ermöglicht. Er spezifiziert die Rolle des Vermittlers mithilfe eines dezentralen Protokolls. Nachrichten werden als Multicast-Nachrichten, also gleichzeitig an eine Gruppe von Empfängern, verschickt und die einzelnen Dienste, die als Empfänger der Multicast-Nachrichten dienen, antworten auf die Anfrage [MK09].

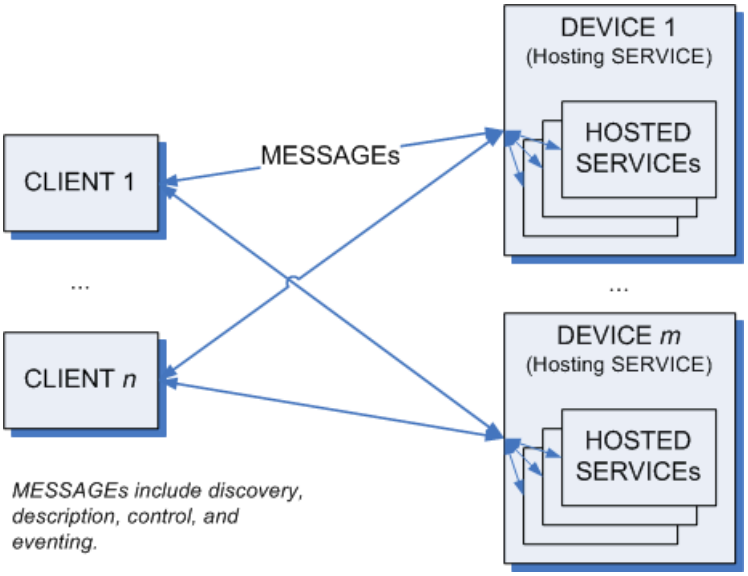


Abbildung 4.4: Geräte und Dienste [DM09]

WS-Eventing: Beschreibt ein Protokoll, das es einem Webservice erlaubt, Anmeldungen für Ereignisse entgegenzunehmen (Subscribe) und Ereignisbenachrichtigungen zu versenden (Publish) [BCC+06].

WS-MetadataExchange: Definiert, wie Metadaten eines Webservice dargestellt, wie sie in die Endpunktreferenz [GHR06] integriert und wie die Metadaten vom Endpunkt eines Webservices ermittelt werden [BBB+08].

WS-Security: Der Standard definiert eine Erweiterung der herkömmlichen SOAP-Nachrichten um sicherheitsrelevante Funktionen, wie Signaturen oder Verschlüsselung.

WS-Policy: Stellt ein generelles Modell zur Beschreibung der Richtlinien von Webservices zusammen [VOH+07].

Geräte und Dienste In der DPWS-Spezifikation wird zwischen Geräten (*Devices*) und Diensten (*Services*) unterschieden. Ein Gerät wird auch *Hosting Service* genannt. Es handelt sich um einen Webservice, der charakteristische Informationen über geräteweite Daten (z. B. Name des Herstellers oder Seriennummer des Gerätes) enthält und die Funktionen des Geräts als Dienste bereitstellt. Ein Dienst, auch *Hosted Service* genannt, enthält ausführbare Operationen, welche die eigentliche Programmlogik enthalten. Ein Dienst muss an einem Gerät registriert sein, um von einem Client in einem Netzwerk gefunden werden zu können. Der Nachrichtenaustausch zwischen Clients und Geräten ist in Abbildung 4.4 dargestellt. Ein Dienstnutzer kann mit einem Gerät kommunizieren und dadurch auf die beim Gerät registrierten Dienste zugreifen.

Die Beschreibung eines Gerätes erfolgt durch ein XML-Dokument. Es werden die charakteristische Informationen über die geräteweiten Daten und Informationen zu den registrierten Diensten angegeben.

Ein Dienst wird hingegen durch ein WSDL-Dokument beschrieben. Es enthält spezifische Informationen zu dem jeweiligen Dienst und beschreibt in maschinenlesbarer Form die Operationen,

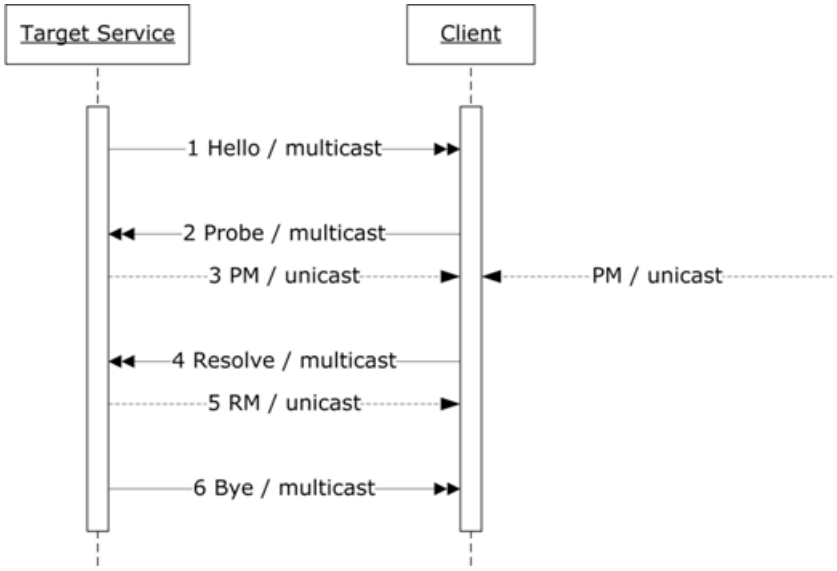


Abbildung 4.5: Discovery-Nachrichten [MK09]

die der Dienst zur Verfügung stellt. Es existieren vier verschiedene Operationstypen, die in einem WSDL-Dokument beschrieben werden können: *One-Way-*, *Request-Response-*, *Solicit-Response-* und *Notification*-Operationen [CCM+01]. Sie unterscheiden sich jeweils durch die Art des Nachrichtenaustauschs mit einem Client. Bei einer *One-Way*-Operation wird eine Nachricht vom Dienstnutzer gesendet und keine Antwort erwartet. Eine *Request-Response*-Operation erwartet eine Antwort oder eine Fehlermeldung auf eine durch den Dienstnutzer gesendete Anfrage. Bei einer *Solicit-Response*-Operation wird zuerst eine Nachricht an den Dienstnutzer gesendet, worauf dieser mit einer Antwort oder einer Fehlermeldung reagiert. Bei der *Notification*-Operation erhält der Dienstnutzer Nachrichten, aber sendet keine Antworten.

Auffinden von Geräten In DPWS wird festgelegt, dass in einem Netzwerk nur Geräte von einem Dienstnutzer direkt gefunden werden können, d. h. nach einem Dienst kann nicht explizit gesucht werden, sondern nur das Gerät bei dem der Dienst registriert ist. Das dynamische Auffinden des Gerätes geschieht mit Hilfe der in der WS-Discovery-Spezifikation festgelegten Mittel. Es werden spezielle Nachrichten als Multicast versandt. Geräte antworten, die in der entsprechenden Multicastgruppe sind. Abbildung 4.5 zeigt einen typischen Nachrichtenverkehr von WS-Discovery. Zum Zeitpunkt ihres Starts senden Geräte eine `Hello`-Nachricht. Dadurch melden sie sich bei bereits vorhandenen Dienstnutzern im Netzwerk. Um nach Geräten im Netzwerk zu suchen, sendet ein Dienstnutzer eine Suchanfrage (`Probe`- Nachricht) per Multicast in das Netzwerk und erhält als Antwort `ProbeMatch`-Nachrichten von den gefundenen Geräten. Diese Nachrichten enthalten Informationen über das Gerät und eine Transportadresse. Für den Fall, dass keine Transportadresse vom Gerät geliefert wird, wird diese gezielt durch eine `Resolve`-Nachricht angefordert und mit einer `ResolveMatch`-Nachricht vom Gerät beantwortet. Beim Beenden eines Gerätes sendet dieses eine `Bye`-Nachricht an alle Dienstnutzer. Dadurch erfahren diese, dass das Gerät nicht länger zur Verfügung steht.

Im Folgenden Unterkapitel 4.4 wird das Steuerungssystem Paket Royale vorgestellt, das SOA und DPWS für die Umsetzung einer dezentralen Anlagensteuerung nutzt. DPWS spielt auch für die Projektgruppe eine wichtige Rolle, da es zur Kommunikation im Netzwerk genutzt werden

soll. Daher werden DPWS und verschiedene Implementierungen dieses Profils in Kapitel 6.4 detaillierter betrachtet.

4.4 Das Steuerungssystem Paket Royale

Dieses Unterkapitel beschreibt die aktuellen Forschungsergebnisse des Lehrstuhls für Förder- und Lagerwesen der TU Dortmund. Diese stellen den bei der Beschreibung der Ist-Situation in Abschnitt 4.2 genannten grundlegenden Schritt zur Flexibilisierung und Modularisierung von Steuerungssoftware für automatisierte Anlagen dar. Abschnitt 4.4.1 zeigt zunächst die Motivation für die durchgeführten Forschungsprojekte auf. Daran anschließend wird im Abschnitt 4.4.2 die lehrstuhleigene Versuchsanlage vorgestellt, sowie in den Abschnitten 4.4.3 und 4.4.4 die aktuell eingesetzte Anlagensteuerung beschrieben, die das Ergebnis dieser Forschungsarbeiten darstellt. Dabei befasst sich der Abschnitt 4.4.3 mit einer einheitlichen Schnittstelle für den Zugriff auf die Gerätefunktionen, wohingegen Abschnitt 4.4.4 die Steuerungslogik der Anlage umfasst, welche auf die einheitliche Geräteschnittstelle zurückgreift. Den Abschluss bildet in Abschnitt 4.4.5 ein Ausblick auf weiterführende Ansätze zur Flexibilisierung der Anlagensteuerung auf die im Unterkapitel 4.5 näher eingegangen wird.

4.4.1 Motivation

Der Einsatz von großen, automatisierten Anlagen im industriellen Umfeld zeichnet sich vor allem durch sehr hohe Planungs- und Installationskosten aus. Dies hat zur Folge, dass die Anlagen über viele Jahre hinweg betrieben werden müssen, damit sich die hohen Investitionskosten rentieren (siehe [SF10]). Demgegenüber steht die stetig zunehmende Dynamik der Absatzmärkte, die häufige Änderungen in Geschäftsprozessen und immer kürzer werdende Produktzyklen mit sich bringt. Die benötigte Flexibilität können die derzeit eingesetzten automatisierten Anlagen nur begrenzt bieten. Änderungen sind i. d. R. sehr aufwändig, teuer und erfordern nicht selten einen kompletten Stillstand der gesamten Anlage (vgl. [SF10]). Dies ist darauf zurückzuführen, dass die Steuerung der Anlagen meist von wenigen monolithischen Softwareblöcken auf zentralen Rechnern durchgeführt werden. Selbst bei lokal begrenzten Änderungen an der Topologie der Anlage sind häufig unverhältnismäßig hohe Anpassungen an der Steuerungssoftware erforderlich (siehe [SF10]). Neuere Forschungsprojekte suchen deshalb nach Möglichkeiten, um die geforderte Flexibilität mit deutlich geringerem Aufwand und somit auch geringeren Kosten zu erreichen.

Ein Ansatz, dies zu erreichen, ist die Einführung einer Serviceorientierten Architektur (SOA, vgl. Unterkapitel 4.3.3) auf der Geräteebene. Hierbei können einzelne Sensoren und Aktoren mit heterogenen, herstellerabhängigen Schnittstellen zu Einheiten zusammengefasst werden, die nach außen hin eine einheitliche standardisierte Schnittstelle in Form von Diensten anbieten. Diese können von einer übergeordneten Steuerung flexibel komponiert werden, wobei die eigentliche Implementierung der Funktionalität der Dienste vor der Steuerung verborgen bleibt. Die Grundlagen dieser Idee sind in dem internationalen Forschungsprojekt SIRENA ([BBG06]) gelegt und speziell für Materialflusssysteme vom Lehrstuhl für Förder- und Lagerwesen der TU Dortmund auf diese übertragen worden (siehe [FLt+09]).

4.4.2 Versuchsanlage

Die Versuchsanlage des Lehrstuhls für Förder- und Lagerwesen besteht aus zwei miteinander verbundenen Ebenen. 37 angetriebene Förderer bilden eine Förderstrecke mit einer Länge von 120 Metern, auf der eine Fördergeschwindigkeit von bis zu 1,5 $^{m}/_{s}$ erzielt werden kann. Als Transportgüter werden Mehrzweckbehälter verwendet, die mit einem RFID-Chip mit einer Speicherkapazität von 512 Bit ausgestattet sind und im UHF-Bereich mit einer Frequenz von 868 MHz arbeiten. Mit auf der Strecke angebrachten RFID Lese- und Schreibgeräten können während des Transports die einzelnen Behälter identifiziert sowie Daten von den RFID-Chips gelesen und geschrieben werden. Insgesamt befinden sich etwa 80 verschiedene Sensoren wie z. B. Lichtschranken auf der Versuchsanlage, um die Positionen der Behälter zu erfassen.

Im Rahmen der Entwicklung verschiedener dezentraler Steuerungen hat der Lehrstuhl für Förder- und Lagerwesen sieben Industrie-PCs (IPCs) verbaut. Dies erweitert die bestehende Anlage um die Möglichkeit, zusätzliche Software, wie etwa die im folgenden Abschnitt vorgestellte SOA, auf der Feldebene der Anlage dezentral auszuführen. Diese IPCs besitzen eine Rechenleistung von 266 MHz, verfügen über 128 MB RAM und 1024 MB Datenspeicher auf einer Compact Flash Card. Als Betriebssystem wird ein Linux mit Echtzeiterweiterung (RTAI) eingesetzt.

Die Geräte auf der Feldebene sind mit den IPCs über den sog. K-Bus verbunden. Dieser wird für die Kommunikation und den Datenaustausch zwischen Geräten und IPCs verwendet und erlaubt den IPCs somit den Zugriff auf die technischen Anlagenfunktionen. Dabei sind Geräte, die zum selben Anlagenteil gehören (z. B. alle zu einem Förderband gehörenden Sensoren und Aktoren), stets mit dem selben IPC verbunden worden, sodass sich eine modulare Zuordnung von Geräten zu IPCs ergibt.

Für die Kommunikation der IPCs untereinander und mit anderen Steuerungsrechnern sind diese via Ethernet miteinander verbunden.

4.4.3 SOA für Geräte

Aufgrund von sich schnell ändernden Marktbedingungen ist es erforderlich, dass sich auch automatisierte Anlagen schnell und zu geringen Kosten an neue Anforderungen anpassen lassen (siehe [FLt+09]). Die bisherigen Anlagen müssen somit flexibler werden und Veränderungen sowie Erweiterungen der Anlage mit geringem Aufwand zulassen. Um dies zu erreichen hat der Lehrstuhl für Förder- und Lagerwesen das Konzept der SOA für Geräte auf Förderanlagen übertragen. Auf die Umsetzung dieses Konzepts für Förderanlagen wird in diesem Abschnitt näher eingegangen.

Ein generelles Problem von automatisierten Anlagen ist die Vielfalt der Geräteschnittstellen, verursacht durch den Einsatz von proprietären Kommunikationsprotokollen, welche die Hersteller für die Kommunikation mit ihren Geräten nutzen (vgl. [FLt+09]). Um diese heterogenen Schnittstellen vor der Steuerungsebene zu verbergen, ist die SOA für Geräte als *Hardware Abstraction Layer (HAL)* konzipiert worden. Der HAL ist eine zusätzliche Schicht zwischen der realen Hardware und deren Steuerung. Diese intermediäre Schicht abstrahiert von der realen Hardware und stellt für alle Geräte eine einheitliche Schnittstelle zur Verfügung. Die Transformation der einheitlichen Schnittstelle in die gerätespezifischen Protokolle geschieht innerhalb des HAL, sodass dies für übergeordnete Ebenen verborgen geschieht und diese mit einer einheitlichen Schnittstelle arbeiten können.

Für den HAL ist zusätzliche Software notwendig. Da die Geräte i. d. R. direkt an eine Speicherprogrammierbare Steuerung (SPS) oder eine andere Automatisierungseinrichtung angeschlossen sind und selbst über keine eigenen Berechnungsressourcen verfügen, kann die zusätzliche Software nicht auf den Geräten selbst ausgeführt werden. Deshalb ist zusätzliche Hardware erforderlich, auf der die zusätzliche Software für den HAL ausgeführt werden kann. Für diesen Zweck sind in der Versuchsanlage des Lehrstuhls die im Abschnitt 4.4.2 vorgestellten IPCs verbaut. Diese übernehmen die Ausführung der Software und besitzen sowohl Schnittstellen für die Geräte der Feldebene (z. B. K-Bus) als auch eine Netzwerkschnittstelle für die Anbindung an ein Netzwerk für die Kommunikation der IPCs untereinander und mit Rechnern auf der Steuerungsebene.

Die SOA für Geräte besteht aus drei Ebenen: die Geräteebene (*device layer*), die Integrationsebene (*integration layer*) und die Steuerungsebene (*control layer*) (vgl. Abbildung 4.6). Diese werden im Folgenden kurz vorgestellt.

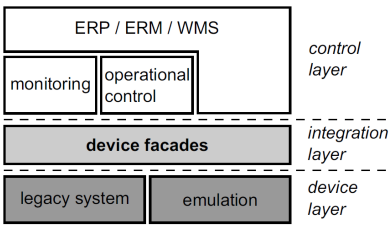


Abbildung 4.6: Aufbau der SOA für Geräte aus [FLt+09]

Geräteebene Die Geräteebene besteht aus Sensoren (z. B. Lichtschranken) und Aktoren (z. B. Motoren) und aus Kompositionen aus diesen bei komplexeren Geräten (z. B. Förderbändern).

Integrationsebene Der Kern der SOA für Geräte ist die Integrationsebene. Hier wird die Abstraktion von gerätespezifischen Schnittstellen und das Bereitstellen einer einheitlichen SOA-basierenden Schnittstelle für den Zugriff aus der Steuerungsebene heraus realisiert. Für die Implementierung ist dabei das Entwurfsmuster *Fassade* verwendet worden, das in der Softwaretechnik dazu eingesetzt wird heterogene Schnittstellen in eine einheitliche Schnittstelle mit Hilfe einer Zwischenschicht zu übersetzen. Die Hauptbestandteile einer Fassade sind in Abbildung 4.7 (a) grafisch dargestellt. Im einzelnen sind dies Zustandsverwaltung, Abhängigkeitsverwaltung und Sub-Devices. Aus Platzgründen wird an dieser Stelle jedoch nur auf die Sub-Devices näher eingegangen. Details zu den weiteren Bestandteilen einer Fassade können [FLt+09] entnommen werden. Die Sub-Devices bilden den Kern einer Fassade und führen die eigentliche Abstraktion von den gerätespezifischen Schnittstellen durch. Dazu besteht jedes Sub-Device aus den in Abbildung 4.7 (b) dargestellten drei Ebenen: Service, Interface und Implementation. Die Serviceebene dient der Bereitstellung eines einheitlichen Dienstes nach außen. Die Implementierungsebene enthält die Implementierung des Dienstes für das spezifische Gerät. Die zwischen diesen beiden Ebenen liegende Interfaceebene entkoppelt schließlich den Dienst von der gerätespezifischen Implementierung.

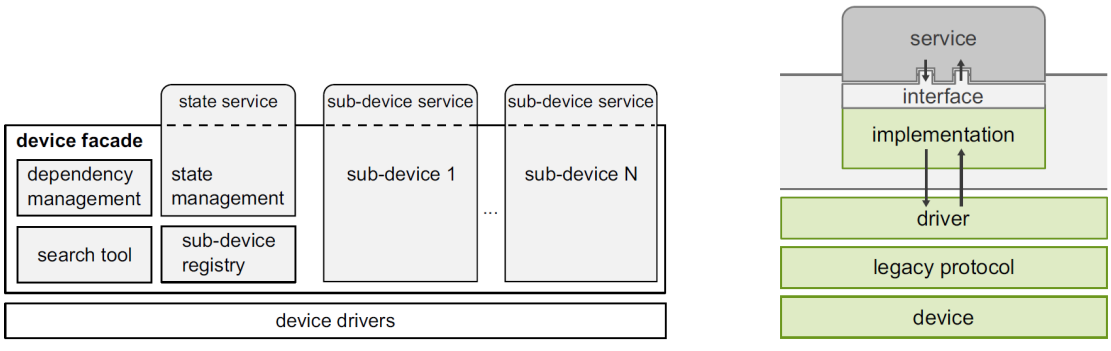
Steuerungsebene Mit der durch die Integrationsebene bereitgestellten einheitlichen SOA-Schnittstelle ist es möglich, die üblicherweise vorherrschenden vertikalen Hierarchien in der

Steuerungsebene aufzubrechen (siehe [FLt+09]). Mit Hilfe des SOA-Ansatzes kann flexible und erweiterbare Software erstellt werden, die auf die angebotenen Dienste der verschiedenen Geräte als Dienstnutzer zurückgreift. Durch die Möglichkeit eines ereignisgesteuerten Nachrichtenaustausches können Monitoring- oder ERP-Systeme direkt aus der Feldebene heraus über den aktuellen Anlagenzustand informiert werden anstatt diesen mit Hilfe eines polling-basierten Verfahrens bei einer zwischengeschalteten Materialflusssteuerungskomponente zu erfragen.

Der soeben vorgestellte Ansatz ist in einer *Controlled Device Application* (*CoDeA*) genannten Laufzeitumgebung implementiert worden, die eine oder bei Bedarf auch mehrere Fassaden ausführen kann. Die Laufzeitumgebung ist in Java entwickelt worden und lässt sich auf Basis von XML-Dokumenten konfigurieren. Um Ressourcen zu sparen, ist dabei die *Java Micro Edition (Java ME)* Plattform in Verbindung mit der *Connected Limited Device Configuration (CLDC)* eingesetzt worden, die speziell auf Geräte mit eingeschränkten Ressourcen abzielt. Für die Kommunikation mit Sensoren und Aktoren der Geräte, die über den K-Bus erfolgt, wird weiterhin eine eigene K-Bus Java-API genutzt. Für die SOA-basierende Kommunikation mit der Steuerungsebene ist der DPWS-Standard gewählt worden. Aufgrund technischer Restriktionen hat man auf den Einsatz der Echtzeitunterstützung für Java verzichten müssen (siehe [FLt+09]). Dies kann zu Fehlfunktionen der Anlage führen, wenn die Dienste zur operativen Steuerung genutzt werden und sich bei der Nachrichtenübertragung Verzögerungen ergeben. Beispielsweise kann ein verzögertes Eintreffen eines Weichenstellungsbefehls zur Folge haben, dass ein zu beförderndes Gut fehlgeleitet wird. Die Erweiterung dieser SOA für Geräte um eine Echtzeitunterstützung ist ein Ansatz für weitere zukünftige Verbesserungen.

4.4.4 Steuerungssystem Paket Royale

In diesem Abschnitt wird das ebenfalls vom Lehrstuhl für Förder- und Lagerwesen entwickelte Steuerungssystem *Paket Royale* vorgestellt. Es baut auf der im vorherigen Abschnitt beschriebenen SOA für Geräte auf und verbindet diesen Ansatz mit dem Konzept der Softwareagenten. Softwareagenten sind Softwareteile, die in der Lage sind vorgegebene Ziele autonom zu erreichen. Ziel dieses Systems ist es, die Steuerung der gesamten Anlage vollkommen dezentral zu gestalten, sodass auf den Einsatz übergeordneter Steuerungskomponenten, wie z. B. Materialflussrechner, verzichtet werden kann.



(a) (b)

Abbildung 4.7: Gerätefassade (a) und Sub-Device (b) aus [FLt+09]

Die Grundidee dabei ist, dass die einzelnen Geräte einer Anlage sich aktiv an der Anlagensteuerung beteiligen und Prozessinformationen für andere Anlagenteile oder übergeordneten Systemen über Dienste in Echtzeit zur Verfügung stellen. Die einzelnen Anlagenteile kommunizieren dabei untereinander über eine Ethernet-Verbindung. Weiterhin findet die Idee des „Internets der Dinge“ (siehe [Gt10]) Anwendung, indem die zu transportierenden Güter als Softwareagenten repräsentiert werden, die aktiv an Steuerungsentscheidungen für das entsprechende Gut mitwirken.

Das Steuerungssystem *Paket Royale* besteht aus vier Ebenen: Automatisierte Anlage, Basisdienste, Steuerungsagenten und Leitstandsoftware (vgl. Abbildung 4.8). Auf diese wird in den folgenden Abschnitten näher eingegangen.

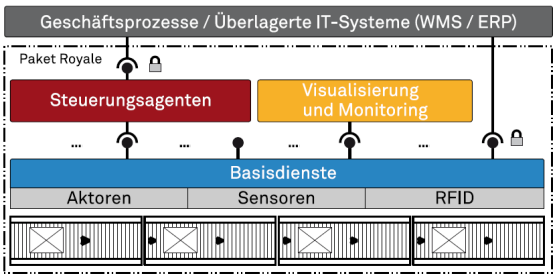


Abbildung 4.8: Aufbau von Paket Royale aus [FFt10]

Automatisierte Anlage Die unterste Ebene bilden die Sensoren und Aktoren und die Förder- und Identifizierungstechnik der zu steuernden Anlage. Im Rahmen der Entwicklung von *Paket Royale* am Lehrstuhl für Förder- und Lagerwesen ist dies die lehrstuhleigene Versuchsanlage, deren Aufbau in Abschnitt 4.4.2 dargestellt ist.

Basisdienste Die zweite Ebene bilden die Basisdienste. Jedes Gerät bietet entsprechend seiner Funktion spezielle Dienste an, die von verteilten Softwareblöcken, wie z. B. Agenten, genutzt werden können, um das Gerät zu steuern. Diese Dienste werden von der Laufzeitumgebung *CoDeA* auf den IPCs angeboten und ausgeführt und sind im vorangegangenen Abschnitt 4.4.3 beschrieben.

Steuerungsagenten Mit Hilfe der ersten beiden Ebenen werden Dienste angeboten. Steuerungsaufgaben werden von ihnen jedoch nicht übernommen. Dies ist die Aufgabe der dritten Ebene auf der so genannte Steuerungsagenten miteinander interagieren und so gemeinsam die Steuerung der Anlage übernehmen. Ein Agent ist ein Stück Software, dessen Funktion es ist, ein vorgegebenes Ziel zu erreichen. Dieses Ziel wird auch als „Mission“ bezeichnet. Dieser Softwareteil agiert selbstständig mit der Laufzeitumgebung, in der er ausgeführt wird, und versucht seine „Mission“ eigenständig zu erfüllen. Dabei kann er mit anderen Agenten interagieren und, falls dies erforderlich ist, auf eine andere Laufzeitumgebung migrieren. Die Laufzeitumgebungen, in denen die Agenten ausgeführt werden, werden von den IPCs ausgeführt und stellen weiterhin Standardfunktionalitäten für die Agenten bereit. Hierunter fallen z. B. Funktionalitäten zur Kommunikation der Agenten untereinander. Typische Handlungsszenarien für Agenten sind z. B. Vorfahrtsregelungen an Zusammenführungen oder Weichensteuerungen

an Verzweigungen. Dabei wird ein Agent stets auf dem IPC ausgeführt, dessen angeschlossene Geräte er steuert. Bei der Durchführung ihrer Aufgaben greifen die Agenten auf die Basisdienste zurück, die aufgrund der SOA für Geräte eine einheitliche und maschinenlesbare Schnittstelle besitzen.

Das Steuerungssystem *Paket Royale* unterscheidet zwei Typen von Agenten: stationäre und mobile Agenten. Die stationären Agenten werden auch Anlagenagenten genannt und sind für spezielle Teile der Anlage wie z. B. Weichensteuerungen oder Vorfahrtsregelungen zuständig. Diese Agenten verweilen dauerhaft auf den IPCs, die zu den Geräten korrespondieren, die sie steuern. Die mobilen Agenten hingegen, die auch als Paketagenten bezeichnet werden, repräsentieren ein zu beförderndes Gut und migrieren parallel zum Gut über die Anlage, d. h. sie werden stets auf dem IPC ausgeführt, der den Anlagenteil verwaltet, auf dem sich gerade das zu befördernde Gut befindet. Verlässt das Gut den Zuständigkeitsbereich eines IPCs, so migriert der Paketagent auf den IPC des folgenden Anlagenteils. Dabei interagiert der Paketagent mit den stationären Anlagenagenten, um zu veranlassen, dass das durch ihn repräsentierte Gut sein vorgesehenes Ziel erreicht. Beispielsweise teilt ein Paketagent einem weichensteuernden Anlagenagenten mit, zu welchem Ziel sein Gut transportiert werden soll, sodass der Anlagenagent die Weiche entsprechend stellen kann.

Abbildung 4.9 zeigt beispielhaft den Lebenszyklus eines Paketagentens. An jedem Wareneingang befindet sich ein stationärer Wareneingangsagent, der für übergeordnete Systeme (z. B. ein WMS oder ERP-System) einen Dienst zur Auftragseinlastung bereitstellt. Nach der Einlastung eines Auftrags wird vom Wareneingangsagenten ein neuer Paketagent erzeugt, dem die Daten des Auftrags als „Mission“ übergeben werden. Dieser hat die Aufgabe für die Erfüllung des Auftrags zu sorgen. Deshalb durchläuft er die Anlage (durch Migration von einem IPC zum anderen) parallel zum Gut und sorgt durch Interaktion mit stationären Anlagenagenten dafür, dass das Gut sein vorgegebenes Ziel erreicht. Bei Erreichen des Ziels vermeldet der Paketagent als abschließende Aktion die Erfüllung seiner Mission an den Wareneingangsagenten und beendet sich anschließend selbst.

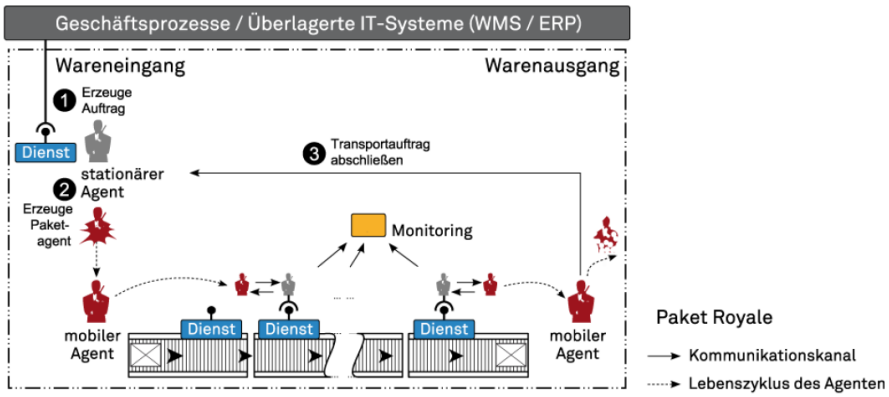


Abbildung 4.9: Der Lebenszyklus eines Paketagenten aus [FFt10]

Leitstandsoftware Die Leitstandsoftware dient der übergeordneten Steuerung, Visualisierung und Überwachung des Anlagenzustands in Echtzeit. Auch die Leitstandsoftware profitiert dabei von der SOA für Geräte und kann über den Aufruf von Diensten aktuelle Zustandsinformationen

der Anlage erhalten. Insbesondere ermöglicht die Diensteschnittstelle mit Hilfe von ereignisbasierten Nachrichten, die Leitstandsoftware in Echtzeit über Störungen zu informieren und erlaubt ein Echtzeit-Tracking der Güter. Auch Agenten können Dienste für die Leitstandsoftware anbieten. Ein Beispiel hierfür ist der Wareneingangsagent. Dieser ermöglicht es der Leitstandsoftware, über das Netzwerk einen Auftrag an die Anlagensteuerung zu übergeben und über dessen Fertigstellung informiert zu werden.

4.4.5 Ausblick

Mit dem Steuerungssystem *Paket Royale* ist es gelungen, die Idee des „Internets der Dinge“ zu realisieren und eine vollständig dezentrale Anlagensteuerung zu entwickeln. Somit entfällt die Notwendigkeit, zentrale Materialflussrechner zu betreiben, die die gesamte Steuerung in wenigen komplexen monolithischen Softwareblöcken vereinen. Stattdessen besteht die Steuerung aus vielen über die gesamte Anlage verteilten Modulen, den Agenten. Dies reduziert die Komplexität der Entwicklung deutlich, fördert zudem die Wiederverwendbarkeit und reduziert den Wartungsaufwand. Weiterhin ermöglicht dieser Ansatz den Austausch von Funktionalität zur Laufzeit. So können einzelne Agenten während des Betriebs ausgetauscht werden und somit die Steuerung der Anlage dynamisch verändert werden. Ein Beispiel hierfür ist der dynamische Austausch eines Agenten und somit der Strategie zur Vorfahrtsregelung an Zusammenführungen. Ebenfalls ermöglicht dieser Ansatz die partielle Stilllegung der Anlage bei Veränderungen der Anlagentopologie. Von Änderungen nicht betroffene Anlagenteile können dabei weiterbetrieben werden.

Durch den Einsatz der SOA für Geräte ist weiterhin die Schnittstelle des Steuerungssystems von den technischen Komponenten entkoppelt worden, sodass das gesamte System unabhängig von den Herstellern der Anlagenkomponenten ist. Die Kommunikation der einzelnen Anlagenteile (Geräte, Agenten, Leitstandsoftware) erfolgt durch die Verwendung von DPWS standardisiert, sodass eine verbesserte Integration der Feldebene in betriebliche IT-Systeme (z. B. WMS oder ERP-Systeme) möglich ist.

Durch die Kombination von SOA und Agentenansatz und der Nutzung der Vorteile aus beiden Ansätzen ist es möglich geworden, die Anlagensteuerung komplett in die Feldebene zu integrieren. Dies steigert die im Abschnitt 4.4.1 geforderte Flexibilität von großen automatisierten Analgen.

Das folgende Unterkapitel 4.5 über *Smart Neighbourhood Module* führt die Idee der Modularisierung und der einheitlichen dienstbasierten Schnittstelle fort und erweitert diesen Ansatz durch Hinzufügen weiterer Intelligenz in die Feldebene.

4.5 Smart Neighbourhood Module

In diesem Kapitel wird die Zukunftsvision modularisierbarer Fördertechnik an Hand der Kerntechnologie *Smart Neighbourhood Module* (auch: SNMs) beschrieben, welche eine mögliche Anwendung der von dieser Projektgruppe entwickelten Cloud-Umgebung darstellt. Grundlage für dieses Unterkapitel bildet der Forschungsantrag CREATE [Maz10]. Zunächst wird in Abschnitt 4.5.1 der Schritt von klassischen monolitischen Anlagenarchitekturen zu SNM-basierten

Architekturen erläutert, dann werden in Abschnitt 4.5.2 die spezifischen Eigenschaften dieser neuen Systeme zusammengefasst. In Abschnitt 4.5.3 werden schließlich die Vorteile von modularisierten Förderanlagen anhand eines konstruierten Beispiels demonstriert.

4.5.1 Motivation Smart Neighbourhood Module

Moderne Materialflusssysteme werden zunehmend mit Automatisierungstechnik kombiniert, wodurch kontinuierlich steigende Computerunterstützung erforderlich ist. In teilautomatisierten Systemen genügt der Einsatz von isolierter Steuerungstechnik auf der Feldebene, um die Funktionalität einzelner Bauteile zu gewährleisten. Für vollautomatisierte Anlagen werden vielschichtige Systemarchitekturen eingesetzt, die umfassende Führungs- und Monitoringaufgaben auf der Betriebsleitebene übernehmen können.

Die klassische Sichtweise auf die Softwarearchitektur von Materialflusssystemen teilt die Systeme vornehmlich in horizontale Schichten ein. Die Softwaremodule werden von der untersten Schicht mit Speicherprogrammierbaren Steuerungen für die Feldebene bis hin zu abstrakteren QoS- und Monitoringaufgaben jeweils einzeln bearbeitet.

Grundlage einer modularen Anlage mit *Smart Neighbourhood Modulen* ist dagegen eine vertikale Betrachtung der Architektur. Jedes SNM kapselt die notwendigen Softwarekomponenten für alle beteiligten Abstraktionsebenen, also low-level Steuerungsfunktionen genauso wie modulübergreifende Funktionen, etwa Paketrouting, sofern diese relevant sind. Über fest definierte Schnittstellen können die Module ihre Umgebung, insbesondere benachbarte SNMs und deren Funktionalität, erkennen. Ein Verbund aus SNMs kann daher vorhandene Module und Systemfunktionen erkennen und die einzelnen SNMs automatisch konfigurieren. Die ganzheitliche Betrachtung der Komponenten vereinfacht dabei viele Eingriffe in die Förderanlage, die im klassisch horizontalen Ansatz nur sehr aufwändig zu realisieren sind. So müssen etwa bei einer Layoutänderung des Systems keine Softwareänderungen vorgenommen werden, um das Zusammenspiel zwischen neuen Komponenten zu regeln. Auch erlaubt eine dynamische Implementierung den Austausch von Elementen im laufenden Betrieb, was insbesondere bei Ausfällen und Laständerungen von Bedeutung ist.

Zusamengefasst basiert die Grundidee von *Smart Neighbourhood Modulen* auf der vertikalen Betrachtung der Softwarearchitektur Materialflusssystemen bei vollständiger Kapselung aller Softwaremodule einer Fördertechnikkomponente. Dabei orientiert sich die Softwarearchitektur des Gesamtsystems an verteilten Systemen, d. h. jede Komponente stellt dem Gesamtsystem im Sinne der Serviceorientierung Funktionalität in Form von Services zur Verfügung. Die Steuerung des Systems erfolgt dezentral. Mittels definierter Schnittstellen und automatischer Nachbarschaftserkennung wird die automatische Konfiguration der Komponenten ermöglicht.

4.5.2 Eigenschaften von SNM-basierten Systemen

Ein ganzheitliches Framework für modulbasierte Fördertechnik bietet neben der automatischen Konfiguration weitere Vorteile. So kann eine Anlage als verteiltes System aufgefasst werden, das eine Reihe von Diensten ausführt, welche den robusten Betrieb vereinfachen. Wichtige Services umfassen dabei das kontinuierliche Erkennen des Systemlayouts, die Verteilung von

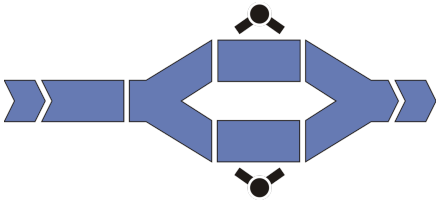


Abbildung 4.10: Basisszenario eines Produktionssystems mit zwei Arbeitsplätzen. Entnommen aus [Maz10]

Systeminformationen über bereitgestellte Funktionalität, Monitoringdienste für Trackingaufgaben, Protokollierung und Workflowanalysen sowie die automatische Dokumentation von Systemänderungen.

Durch die automatische Propagierung von Systemfunktionen einzelner SNMs werden Änderungen im laufenden Betrieb ermöglicht. Dadurch können Ausfälle kompensiert, Kapazitäten angepasst und neue Funktionen in die Fördertechnik integriert werden. Die gewonnene Flexibilität erhöht insbesondere die Planungssicherheit im Bereich der langfristigen Fabrikplanung sowie in der mittelfristigen Produktionsplanung.

Besonders deutlich wird die Flexibilität von SNM-basierten Systemen am Beispiel der automatischen Integration manueller Arbeitsplätze. So kann ein Arbeiter etwa mit einem tragbaren Interface eine *Smart-Neighbourhood*-fähige Förderstrecke ohne zusätzlichen Konfigurationsaufwand um neue Funktionalität – etwa eine Montagetätigkeit – erweitern. In Abschnitt 4.5.3 wird dieses Szenario in einem Anwendungsbeispiel genauer dargestellt.

Ein weiteres Merkmal ist die inhärente Evolutionsfähigkeit des Systems. Durch einen Frameworkgestützten Ansatz wird die Kompatibilität von Modulen verschiedener Hersteller forciert, und damit sowohl das Angebot für Anlagenbetreiber ausgeweitet, als auch die zukünftige Erweiterungsfähigkeit von SNM-fähigen Systemen ohne erhöhte Anpassungskosten gewährleistet.

4.5.3 Anwendungsbeispiel

Im CREATE-Forschungsantrag [Maz10] werden die Vorteile von SNMs u. a. anhand von Anwendungsfällen erläutert. Diese Szenarien sollen im Folgenden auszugsweise vorgestellt werden. Beispielhaft wird ein Produktionssystem für Set-Top-Boxen skizziert, welches im Basisszenario in Abbildung 4.10 zwei Arbeitsstationen umfasst.

Aufgrund eines neuen Zulieferers wird es notwendig, eine Warenkontrolle für die ankommenden Produktionsgüter einzurichten. Durch die Möglichkeit, nahtlos manuelle Arbeitsplätze in das Materialflusssystem einzubinden, kann diese Erweiterung im laufenden Betrieb direkt an einem bestehenden Transportelement durch einen Mitarbeiter eingerichtet werden. (vgl. Abbildung 4.11)

Eine Erweiterung des Produktportfolios macht anschließend die Einrichtung weiterer Arbeitsschritte in der Produktionslinie erforderlich. Dazu werden zwei neue Förderelemente in das bestehende Materialflusssystem integriert (Abb. 4.12).

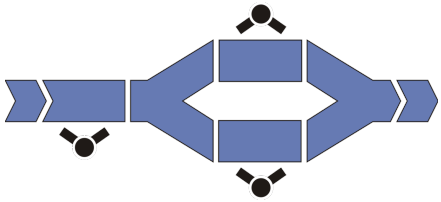


Abbildung 4.11: Nahtlose Integration von manuellen Arbeitskräften. Entnommen aus [Maz10]

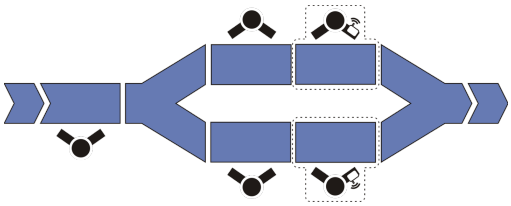


Abbildung 4.12: Erweiterung des Systems durch neue Arbeitsstellen. Entnommen aus [Maz10]

Schließlich werden die Qualitätsanforderungen an die Produktion durch einen neuen Kunden erhöht. Durch die Integration eines optischen Kontrollmoduls (vgl. Abbildung 4.13) und anschließendem Ausschleusmechanismus können Güter anhand von Qualitätsmerkmalen aussortiert werden. Nach Abschluss des Auftrages bzw. der Charge kann das Modul wieder aus der Fertigungsstrecke entfernt werden, um an anderer Stelle eingesetzt zu werden.

Durch die Flexibilität von *Smart Neighbourhood Modulen* lassen sich viele weitere Szenarien entwerfen, etwa schnelle Reaktionen auf Arbeitsplatzausfälle, Layoutänderungen für Spezialaufträge oder die kurzfristige Einführung von Sonderfunktionen. Die vollständige Evaluierung dieser und weiterer Möglichkeiten ist Forschungsgegenstand des CREATE-Projekts.

4.6 Cloud Computing

Obwohl der Begriff Cloud Computing häufig verwendet wird, existiert keine einheitliche Definition, welche ein grundlegendes Verständnis dieses Begriffs bietet. Es hängt somit vom Kontext ab, wie Cloud Computing definiert wird. Zum einen wird dieses Gebiet mit einer (einfachen) Abstraktion gleichgesetzt, welche dazu dienen soll, komplexe Strukturen zu verbergen („etwas wird in die Wolke ausgelagert“). Zum anderen existieren auch komplexe bzw. technische Definitionen, welche Details festlegen und somit eine eindeutigere Beschreibung ermöglichen.

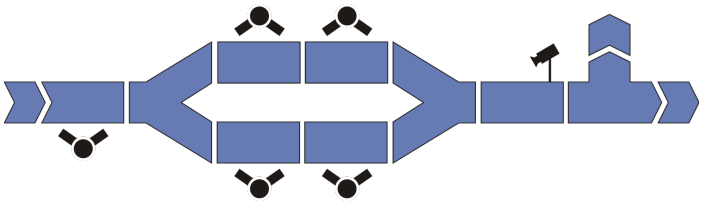


Abbildung 4.13: Installation einer optischen Kontrolleinheit und eines Moduls zur Ausschleusung von Waren. Entnommen aus [Maz10]

Cloud Computing kann als neues Paradigma angesehen werden, welches jedoch keine neuen Techniken bereitstellt, sondern auf bestehenden beruht. Der Vorteil liegt darin, dass im Kontext von Cloud Computing auf bewährte Technologien zurückgegriffen wird und diese neu kombiniert werden. Besonders lässt sich hierbei herausstellen, dass Cloud Computing auf verteilten Strukturen beruht, welche bereits für Grid Computing genutzt werden. Im Cloud-Umfeld wird sich auf bekannte Internet-Technologien verlassen. Beispiele für verwendete Technologien sind die Virtualisierung, mit der sich das Speichermanagement verbessern lässt (z. B. Pooling von Speicherressourcen), oder der Aufbau von serviceorientierten Strukturen (z. B. SOA & Webservices). Um die verschiedenen Varianten der angebotenen Cloud-Lösungen gliedern zu können, werden verschiedene Architekturschichten definiert, von denen die relevanten im Folgenden erläutert werden ([TBH11]). Einen Überblick über diese Schichten gibt Abbildung 4.14.

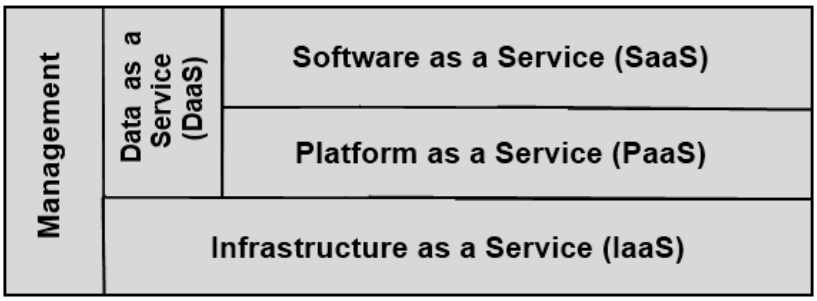


Abbildung 4.14: Ebenen der Cloud Dienste

4.6.1 Infrastructure as a Service

IaaS (*Infrastructure as a Service*) bezeichnet eine Cloud-Architektur, bei der eine Infrastruktur in Form einzelner physikalischer Ressourcen, wie Speicherplatz, oder zusammengesetzt als virtuelle Maschinen (im Folgenden als Instanzen bezeichnet) ohne weitere Vorgaben zur Verfügung gestellt wird [LKN+09, TBH11]. Der Nutzer ist selbst dafür verantwortlich, ein Betriebssystem sowie Software in die Cloud einzubringen, um diese Software innerhalb der Cloud zu nutzen. Die Infrastruktur kann bereits Grunddienste enthalten, wie z. B. einen Lastmonitor, der die automatische Skalierung der Cloud auf mehrere Server sicherstellt oder vordefinierte Software verteilt. Das geschieht i. d. R. über vordefinierte Images, sodass keine weitere Konfiguration der neuen Cloud-Instanzen nach deren Start nötig ist ([Ama11]).

Skalierung Die Cloud besteht aus einer variablen Anzahl von Instanzen, die in lastintensiven Zeiträumen heraufgesetzt wird und in Zeiträume mit geringerer Auslastung entsprechend herunterskaliert werden kann. Dies kann entweder automatisiert durch eine Steuersoftware des System oder durch manuelles Ansprechen der physikalischen Maschinen und Starten oder Stoppen dieser geschehen. Mittels Ressourcen-Monitoring kann sowohl die Auslastung der einzelnen Instanzen als auch deren Netzwerkauslastung festgestellt werden. Basierend auf diesen Kennzahlen können ab einer definierten Lastgrenze (z. B. 90% Systemauslastung) weitere Instanzen gestartet werden, um das Erreichen der Ressourcengrenze durch Lastspitzen zu verhindern. Ebenso kann bei anhaltender geringer Auslastung des Systems, durch das Stoppen einzelner Instanzen, dieses wieder herunterskaliert werden, um Kosten zu sparen und nicht verwendete Ressourcen freizugeben.

4.6.2 Platform as a Service

PaaS (*Platform as a Service*) beschreibt eine Cloud-Architektur, bei der der Nutzer frei innerhalb einer gegebenen Plattform arbeitet, an dieser selbst jedoch keine Veränderungen vornehmen kann. Eine Plattform kann z. B. ein Betriebssystem sein, aber auch ein Softwaresystem, in dessen Rahmen frei interagiert werden kann, jedoch nicht darüber hinaus [LKN+09, TBH11]. Bei einem PaaS wird die Hardwareebene im Gegensatz zum IaaS-Konzept abstrahiert und weitestgehend vor dem Nutzer verborgen. Die Plattform kann automatisch über mehrere Server skalieren, ohne dass der Nutzer davon weiß. Für den Nutzer ist die Plattform nur wie eine einzelne Maschine sichtbar. Beispiele für eine solche Plattform können Webanwendungen sein, die sich in Abhängigkeit von den Anfragen auf mehrere Maschinen replizieren (vgl. [Goo11a]) oder ein selbstständig skalierendes Betriebssystem, wie z. B. Windows Azure ([Mic11a]). Im Falle von Softwareplattformen ist ein anschauliches Beispiel ein Geoinformationsserver, der Kartendienste für weitere Softwarekomponenten bereitstellt und die Anfragen transparent auf mehrere Maschinen verteilt ([Env11]).

Die Grundidee einer PaaS-Architektur ist es, dem Nutzer eine leistungsstarke Plattform zur Verfügung zu stellen, die ressourcenintensive Aufgaben bewältigen kann. Dies geschieht i. d. R. durch das Verteilen des Systems auf mehrere Cloud-Instanzen. Der Nutzer arbeitet jedoch nur innerhalb der Plattform und die Verteilung wird dadurch vor ihm verborgen. Entsprechend der auf die Plattform anfallenden Last kann diese (sofern vorgesehen) dynamisch herauf- oder herunterskalieren.

Loadbalancing Für die verschiedenen Aufgaben, die in einer Plattform ausgeführt werden können, ist es nötig, dass die Plattform eine Form von Loadbalancing zur Verfügung stellt. Nicht alle Aufgaben können komplett verteilt stattfinden, sondern können unter Umständen nur auf einer einzelnen oder einer festgeschriebenen Anzahl von Instanzen ausgeführt werden. In diesem Fall ist es Aufgabe der Plattform, die Instanzen mit der geringsten Auslastung auszuwählen und die Berechnung an diese Instanzen zu verteilen, um eine möglichst zeitnahe Bearbeitung zu gewährleisten. Im einfachsten Fall kann an dieser Stelle ein *Round-Robin-Mechanismus* eingesetzt werden, der die Aufgaben in einer gleichbleibenden Reihenfolge an alle Instanzen eines Systems verteilt, um die Last von Beginn an gleichmäßig aufzuteilen [Tan07]. In komplexen Fällen können an dieser Stelle ebenfalls Lastmessungsmechanismen genutzt werden, die anhand der aktuellen Auslastung entscheiden, auf welche Instanzen die Aufgabe verteilt wird.

Verteiltes Rechnen Teilbare Aufgaben ermöglichen es, dynamisch über mehrere Instanzen skaliert zu werden. Ein einfaches Schema dafür ist MapReduce [DG08], dass wie folgt funktioniert:

Im Map-Schritt unterteilt der Aufrufer die Berechnung in mehrere Arbeitsschritte, die unabhängig voneinander gelöst werden können. Diese kleinstmöglichen Arbeitsschritte werden an unterschiedliche Instanzen verteilt, um die Rechenkapazitäten der einzelnen Instanzen optimal auszunutzen. Haben die Instanzen ihre Aufgabe gelöst, senden sie die jeweiligen Ergebnisse an den Aufrufer zurück. Dieser wartet, bis alle Teilergebnisse eingegangen sind, und kombiniert diese im Reduce-Schritt zu einer Gesamtlösung.

Ein anschauliches Beispiel dafür ist das Zählen von Buchstaben-Vorkommnissen in einem Text. Jeder Instanz wird ein Teiltext zugeteilt, für den sie die Buchstaben zählt, und liefert das Ergebnis

an den Aufrufer zurück. Dieser muss die einzelnen Ergebnisse aufsummieren, was eine weniger rechenintensive Aufgabe ist.

4.6.3 Software as a Service

Software, die auf Rechensystemen eines Anbieters installiert und betrieben wird und von Kunden über das Internet, beispielsweise per Webinterface, kostenlos oder kostenpflichtig, genutzt wird, bezeichnet man als *Software as a Service*, kurz SaaS [AFG+09]. In der Hierarchie der Cloudebenen (siehe Abbildung 4.14) befindet sich SaaS auf der höchsten Ebene. SaaS kann sowohl eine vollständige durch den Anwender benutzte Anwendung, wie beispielsweise Google Maps [Goo12], oder aber ein Softwaredienst, wie OpenID, [Ope06] sein, der in andere Anwendungen integriert werden kann. Die Bandbreite reicht von kleinen Anwendungen für eine persönliche Aufgabenliste, wie Remember The Milk [Rem12], über Büroanwendungspakete, wie die zu Windows Live [Mic12] gehörenden Microsoft Office Web Apps, bis hin zu von Kunden selbst erweiterbaren CRM Systemen, wie Salesforce [Sal00].

Von Vorteil für den Kunden ist die schnelle Einführung und kostengünstige Nutzung von Software, die nicht auf eigenen Computern möglicherweise umständlich installiert und gewartet werden muss [BKN+11]. Weiterhin kann die Software je nach Bedarf von einer unterschiedlichen Anzahl Anwendern genutzt und nur die tatsächliche Nutzung abgerechnet werden, sodass keine Lizenzen gekauft werden müssen, die potenziell ungenutzt bleiben. Auch für den Anbieter kann SaaS Vorteile bieten. So können Anwendungen beim Kunden plattformübergreifend genutzt werden und dennoch bis auf die Benutzerschnittstelle kostengünstig für genau eine Ausführungsplattform entwickelt werden. Ein wesentlicher Nachteil von SaaS für Kunden und Nutzer ist der Datenschutz, da Daten durch fremde Systeme verarbeitet und gespeichert werden.

4.6.4 Data as a Service

Eine weitere Kernkomponente jeder Cloud-Umgebung ist die Datenhaltung [BKN+11]. Hier hat sich die Abkürzung DaaS für *Data as a Service* etabliert. Wie in den Bereichen Infrastruktur und Rechenleistung wird auch im Bereich der Datenhaltung eine Programmierschnittstelle angeboten, die durch Virtualisierung von der eigentlichen Hardware des Betreibers vollständig abstrahiert. Dem Nutzer erscheinen die Ressourcen unbegrenzt. DaaS bildet dabei im Gegensatz zu IaaS oder PaaS keine eigene Ebene der Cloud-Dienste (siehe Abbildung 4.14), sondern ist je nach Cloud-Architektur auf einer anderen Ebene angesiedelt. Dabei müssen nicht alle Datenhaltungsdienste zu der gleichen Ebene einer Cloud-Architektur gehören.

Zu unterscheiden sind insbesondere solche Datenhaltungsdienste, die mittels verteilter Dateisysteme virtuelle Blockgeräte wie Festplatten anbieten, sowie Datenbankdienste. Verteilte Dateisysteme werden in erster Linie auf der Infrastrukturebene benötigt. Hier dienen sie z. B. dazu, die Images von virtuellen Maschinen zu speichern, oder persistenten Speicher für verschiedene andere Dienste, wie z. B. die Datenbanken, bereitzustellen. Dahingegen lassen sich Datenbanken in diesem Beispiel auf der Plattformebene ansiedeln, da sie die Datenhaltungsdienste der Infrastrukturebene nutzen und zusammen mit anderen Diensten eine Plattform für die Anwendungsebene bilden. Andererseits werden zur Bereitstellung einer Infrastruktur Datenbankdienste benötigt, woran nocheinmal deutlich wird, dass sich die Datenhaltungsdienste keiner festen Ebene zuordnen lassen, aber auch keine eigene Ebene bilden. In den folgenden Abschnitten

finden sich nähere Beschreibungen dieser unterschiedlichen Datenhaltungsdienstklassen sowie ihrer Unterklassen.

Verteilte Dateisysteme Verteilte Dateisysteme oder Netzwerkdateisysteme verbinden physische Datenträger auf unterschiedlichen Computern zu einem Dateisystem. Die Geschwindigkeit und Zugriffszeit auf das Dateisystem hängt dabei stark vom verwendeten Netzwerk ab. Der Zugriff auf das verteilte Dateisystem kann im Gegensatz zu den üblichen lokalen oder einem Cluster Dateisystem, welche auf Block-Level arbeiten, direkt über das Protokoll kontrolliert werden. Diese Kontrolle muss also nicht auf dem Client implementiert sein. Weiterhin gibt es je nach eingesetztem verteilten Dateisystem die Möglichkeit, Daten zu replizieren und die Daten so trotz des Ausfalles einer Node weiterhin zur Verfügung stellen zu können. Häufig sind verteile Dateisysteme transparent, sodass sie wie lokale Dateisysteme eingebunden und genutzt werden können. Dem Benutzer wird nicht sichtbar, dass er gerade auf einem verteilten Dateisystem arbeitet. Das große Problem der verteilen Dateisysteme ist die gleichzeitige Bearbeitung der Daten von mehreren Personen. Je nach eingesetztem Protokoll sind hier Mechanismen für die Behandlung von Konflikten oder zum Sperren vom Daten für andere Benutzer vorgesehen.

NoSQL-Datenbanken Die Entwicklung auf dem Gebiet der nichtrelationalen Datenbanken, auch NoSQL-Datenbanken genannt, ist ein entscheidender Faktor für den Erfolg der Idee vom Cloud Computing [Sur10a]. Im Gegensatz zur Gruppe der sehr unterschiedlich ausgeprägten NoSQL-Datenbanken, auf die im Folgenden eingegangen wird, weisen Relationale Datenbanksysteme (RDBS) einige typische Eigenschaften auf. Sie basieren auf streng tabellenorientierter bzw. relationaler Datenhaltung und bieten eine sehr mächtige Schnittstelle [Cod90] für komplexe Anfragen. Weiterhin zeichnen sie sich durch höchste Konsistenz aus, sichergestellt durch Transaktionen [EGL+76] sowie Persistenz. Unter Persistenz versteht man in der Informatik das Bewahren eines Zustandes bzw. von Daten, auch nach dem Abschalten eines Systems. Im Gegenzug skalieren sie schlecht beim Einsatz von Parallelisierung zur Anfrageverarbeitung.

Diese Eigenschaften von relationalen Datenbanksystemen können bei einem Einsatz in typischen Cloud-Umgebungen, die ein hohes Maß an Parallelität bei der Verarbeitung von Datenbankanfragen erfordern, nicht vollständig erreicht werden. Die geforderte höhere Performanz kann durch bessere Skalierung mittels Einschränkungen bei verschiedensten Eigenschaften von relationalen Datenbanksystemen erreicht werden. Da jede Anwendung individuelle Anforderungen an ein Datenbanksystem stellt, existieren sehr viele, sich in ihren Eigenschaften deutlich unterscheidende, Datenbanksysteme, die einzelne oder mehrere der genannten zentralen Eigenschaften von RDBS nur eingeschränkt oder gar nicht aufweisen. Diese Datenbanksysteme bilden die Klasse der NoSQL-Datenbanken. Die wesentliche Gemeinsamkeit dieser Datenbanken besteht darin, dass die Daten nicht nach dem relationalen Schema vorgehalten werden und somit keine zu SQL äquivalente Abfragesprache vollständig, effizient und mit ähnlicher Performanz unterstützt werden kann. Anstelle von Relationen dienen meistens eindeutige Schlüssel zur Referenzierung von kleinsten bis hin zu sehr großen Datensätzen. In den folgenden Abschnitten finden sich nähere Beschreibungen zu unterschiedlichen Klassen von NoSQL-Datenbanken.

Key-Value-Stores Die einfachste Form der NoSQL-Datenbank, der sogenannte *Key-Value-Store*, kurz KV-Store, ist eine Umsetzung des im vorherigen Absatz genannten Grundgedanken [Gua10]. Anstatt von Relationen werden eindeutige Schlüssel zur Referenzierung der Daten

benutzt. Ein KV-Store speichert die Daten zusammenhangslos. Komplexere Abfragen sind bei Unkenntnis der relevanten Schlüssel nur ineffizient durch Iterieren über alle Daten möglich. In verteilten Umgebungen ist dies parallel per MapReduce-Schema möglich, wobei jeder Datenbankknoten die Daten bearbeitet, die er vorhält. Aus diesem Grund ist eine einfache MapReduce-Unterstützung in einige KV-Store-Lösungen integriert [Haz11], falls solche Anfragen trotzdem notwendig sind. Durch diese Eigenschaften eignet sich ein einfacher KV-Store vor allem als Datenbank für Daten, über die nur wenige komplexe, nicht im Voraus bekannte Anfragen ausgeführt werden müssen, bei denen die Schlüssel der betroffenen Datensätze unbekannt sind. Für im Voraus bekannte, komplexere Anfragen lassen sich Hilfsdatensätze pflegen, die alle für die Anfrage relevanten Schlüssel enthalten. Für die Umsetzung von KV-Stores eignen sich die durch Amazons KV-Store Dynamo [DHJ+07] bekannt gewordenen Techniken *Eventual Consistency* und *Consistent Hashing*. Sie ermöglichen eine gute Skalierung bei Verteilung auf große Cluster. Die Eigenschaften dieser zwei Techniken werden im Folgenden kurz erläutert.

Das den Einsatz der *Eventual Consistency* motivierende CAP-Theorem [GL02] besagt, dass die drei Eigenschaften Konsistenz, Verfügbarkeit und Partitionstoleranz in einem verteilten System nicht gleichzeitig garantiert werden können. Die Konsistenz ist in diesem Zusammenhang definiert als die Eigenschaft, dass alle Knoten zur gleichen Zeit die gleichen Daten sehen. Verfügbarkeit bezeichnet die Eigenschaft, dass das System weiter arbeitet, solange nicht alle Knoten ausfallen. Wird der Verlust beliebiger Nachrichten toleriert, so wird dies als Partitionstoleranz bezeichnet. Bei verteilten Datenbanken auf vollständige Konsistenz zu verzichten, um die beiden anderen Eigenschaften garantieren zu können, führt zu dem Konsistenzmodell der *Eventual Consistency*. In der Praxis können sich bei Systemen, die sich an diesem Konsistenzmodell orientieren, Inkonsistenzen im Millisekundenbereich ergeben. Dieser Umstand muss bei der Planung eines Systems berücksichtigt werden, ist aber für sehr viele Anwendungen keine Einschränkung.

Das zweite Konzept ist das des *Consistent Hashing*. Der Einsatz konsistenter Hashing-Algorithmen in NoSQL-Datenbanken ist durch Amazons Dynamo populär geworden, dessen Techniken in [DHJ+07] beschrieben werden. Zur Verteilung der Daten auf die Knoten eines Clusters werden Hashing-Verfahren benutzt. Das hat den Vorteil, dass der Aufenthaltsort eines zu einem bestimmten Key gehörenden Datums im Cluster durch jeden Knoten gleichermaßen autonom und schnell festgestellt werden kann. Es ist kein zentraler Verwaltungsknoten notwendig, in dem der Aufenthaltsort aller Daten verzeichnet ist. Die Verfahren verteilen das Intervall der Zielfunktionswerte der Hashfunktion auf einen Ring. Der Hash-Wert eines Datums ordnet dieses dem Knoten zu, dessen einmalig zufällig berechnete aber eindeutige Identifikation den nächst größeren Hash-Wert aufweist. Der Nachteil eines solchen einfachen *Consistent Hashing*-Verfahrens liegt darin, dass Daten und Last nicht gleichmäßig über alle Knoten verteilt sind, da die Hash-Werte der Identifikationen nicht gleichmäßig über den Ring verteilt sind.

Die in [DHJ+07] beschriebene Variante löst dieses Problem dadurch, dass jeder Knoten eine Vielzahl von Indentifikationen und somit Positionen auf dem Ring zugewiesen bekommt. Um die Last in heterogenen Clustern gewichtet nach der Leistung der Knoten zu verteilen, wird die Anzahl der Identifikationen eines Knoten in Abhängikeit von seiner Leistungsfähigkeit gewählt. Zur n-fachen Replizierung der Daten werden diese zusätzlich auf den nächsten n Knoten, nach Reihenfolge auf dem Ring der Hashfunktion, abgelegt.

Zeilenorientierte Datenbanken Die auf wenige Methoden beschränkte Schnittstelle einfacher KV-Stores reicht für viele Anwendungen nicht aus bzw. bedingt eine aufwändige Modellie-

rung der Datenhaltungsschicht der Anwendung. Daher existieren NoSQL-Datenbanklösungen, die komplexere Anfragesprachen unterstützen, welchen meistens Konzepte von SQL zugrunde liegen. Wie aus der Bezeichnung dieser Klasse von NoSQL-Datenbanken hervorgeht, sind die Daten im Gegensatz zu RDBS nicht in spaltenorientierten Tabellen, sondern in zeilenorientierten Tabellen organisiert. Die einzelnen Datensätze sind wie bei KV-Stores durch eindeutige Schlüssel identifizierbar, weisen aber zusätzlich adressierbare Spalten auf. Diese Spalten sind allerdings im Gegensatz zu RDBS nicht explizit per Schema vorgegeben und können sich zwischen den einzelnen Datensätzen unterscheiden. Die Datenstruktur einer solchen Datenbank kann somit auch als schwach besetzte Tabelle mit variabler Anzahl von Zeilen und auch Spalten beschrieben werden, wobei die Spaltenmenge der Tabelle der Vereinigung der Spaltenmengen aller Datensätze entspricht. Anders als bei einfachen KV-Stores sind viele zeilenorientierte Datenbanken nach dem Vorbild von Googles BigTable [CDG+06] vollständig konsistent. Dafür wird auf eine höhere Performanz verzichtet. Entwicklungen wie z. B. Apache Cassandra hingegen verknüpfen den zeilenorientierten Ansatz mit den Techniken *Eventual Consistency* und *Consistent Hashing* aus dem Bereich der einfachen KV-Stores um eine höhere Performanz zu erreichen. Als Konsequenz werden die Daten durch Cassandra nicht vollständig konsistent vorgehalten.

Dokumentenorientierte Datenbanken Dokumentenorientierte Datenbanken [NL97, Len09] gibt es schon deutlich länger als die Begriffe Cloud Computing [Ama06] und NoSQL [Osk09]. Ein bekannter Vertreter dieser Klasse von Datenbanken ist Lotus Notes. Der Grundgedanke dokumentenorientierter Datenbanken besteht darin, eine Vielzahl von Dokumenten mit potenziell unterschiedlichen Datenfeldern derart abzuspeichern, dass Anfragen z. B. bezüglich einer bestimmten Belegung eines Datenfeldes effizient beantwortet werden können.

Das dokumentenorientierte Datenbankkonzept basiert auf den gleichen Grundgedanken wie das erst wesentlich später entwickelte Datenbankkonzept der zeilenorientierten NoSQL-Datenbanken. Die Dokumente einer dokumentenorientierten Datenbank sind vergleichbar mit den Zeilen einer zeilenorientierten Datenbank. Beide sind das Grundelement für Datensätze der jeweiligen Datenbank und durch eindeutige Schlüssel adressierbar. Weiterhin entsprechen die Datenfelder der Dokumente den Spalten der zeilenorientierten Datenbanken. Datenfelder und Spalten sind beide im Gegensatz zu Relationen nicht für alle Datensätze gleich und trotzdem in Anfragen einbeziehbar. Dies bedingt eine geringere Performanz als bei RDBS für Anfragen, von denen eine Menge von Datensätzen betroffen ist, deren Zusammensetzung nicht aus dem Schlüssel ihrer Elemente, sondern aus dem Inhalt der Datenfelder ihrer Elemente, ersichtlich ist.

4.6.5 Public- und Private-Clouds

Die hier beschriebenen Cloud Infrastrukturen können als Public- oder Private-Cloud betrieben werden. Als Private-Cloud bezeichnet man eine Cloud mit einem geschlossenen Nutzerkreis. Es handelt sich somit meist um eine firmeninterne Cloud, die nur den eigenen Mitarbeitern zur Verfügung steht und mit interner Infrastruktur betrieben wird. Der Vorteil in der Nutzung einer Private-Cloud liegt darin, dass sofern firmenintern genügend freie Rechenkapazitäten vorhanden sind, keine weiteren Kosten für Anschaffung oder Wartung von Rechnern anfallen. Zusätzlich werden Sicherheitsrisiken, wie in [JG11] beschrieben minimiert.

Von einer Public-Cloud spricht man bei Clouds, die von einem Unternehmen vermarktet werden und somit einem beliebigen Kundenkreis zur Verfügung gestellt werden. Bekannte Cloud-

Anbieter sind beispielsweise Amazon, Microsoft oder Google (siehe [Ama11, Mic11a, Goo11a]). Der Vorteil für diese Unternehmen liegt darin, dass Sie durch Ihr Kerngeschäft eine große Rechnerinfrastruktur besitzen, die gewartet wird und nicht komplett ausgelastet ist. Freie Ressourcen können also gewinnbringend verkauft werden. Der Vorteil für den Kunden liegt darin, dass keine Kosten für Anschaffung oder Wartung neuer Rechner anfallen. Zudem wird eine beliebige Skalierung der belegten Ressourcen möglich, sodass Lastspitzen in der Nutzung durch die Infrastruktur des Cloud-Anbieters problemlos ausgeglichen wird und die Verfügbarkeit nicht eingeschränkt wird.

4.7 Zusammenfassung

Die in diesem Kapitel vorgestellten Grundlagen bilden die Basis für die weitere Konzeption und Realisierung von FiLeCC. Materialflusssysteme und ihre Umsetzung (siehe Abschnitt 4.1, 4.2 und 4.4) dienen dabei als Anwendungsbeispiel für die Cloud-Plattform FiLeCC, die freie Ressourcen in der Feldebene zur Verfügung stellen soll. In Zukunft sollen Materialflusssysteme modular und flexibel gestaltet werden können (siehe dazu Abschnitt 4.5). Eine Cloud-Plattform bietet eine optimale Möglichkeit, auch die Rechenressourcen eines modularen Systems flexibel nutzen zu können. Die Grundlagen zu Cloud-Plattformen sind in Abschnitt 4.6 vorgestellt. Auf für die Steuerung von aktuellen und vor allem auch zukünftigen Materialflussanlagen geeignete Computersysteme, geht Abschnitt 4.3.1 ein. Die Abschnitte 4.3.2 und 4.3.3 führen in die Grundlagen zur Kommunikation zwischen den einzelnen Komponenten einer Cloud ein. In den folgenden Kapiteln wird auf diesen Grundlagen aufbauend die Cloud-Plattform FiLeCC konzipiert.

KAPITEL 5

Anforderungsanalyse

In diesem Kapitel werden die Anforderungen an die zu entwickelnde Ausführungsplattform motiviert und beschrieben, sowie auf die geforderten Produktfunktionalitäten eingegangen und herausgestellt, wie diese umgesetzt werden. Abschnitt 5.1 enthält die Motivation für die zu entwickelnde Ausführungsplattform und erweitert die in der Einleitung kurz skizzierte Problemstellung anhand des Anwendungsgebiets der logistischen Materialflusssysteme. Hieran anschließend werden in Abschnitt 5.2 die Anforderungen an die Ausführungsplattform definiert. Im darauf folgenden Abschnitt 5.3 werden die Produktfunkionen, die sich aus den Anforderungen ergeben, beschrieben und dessen geplante Umsetzung dargestellt. Sowohl die Anforderungen als auch die Produktfunktionen sind aus dem Pflichtenheft entnommen (siehe Anhang A). Zur besseren Eingliederung in diesen Bericht sind einige Anpassungen an den Anforderungen und Produktfunktionen vorgenommen worden. Inhaltlich stimmen diese jedoch weiterhin mit denen des Pflichtenheftes überein. Die Anforderungen an die Speicherschnittstelle der Plattform werden gesondert in Abschnitt 5.4 betrachtet. Den Abschluss dieses Kapitels bildet ein kurzes Fazit in Abschnitt 5.5.

5.1 Motivation und Zielbestimmung im Rahmen des Anwendungsgebiets Logistik

Aus der Einleitung in Kapitel 1 geht hervor, dass die zu entwickelnde Ausführungsplattform generell in allen Anwendungsgebieten eingesetzt werden kann, in denen Rechenressourcen mit ungenutzten Kapazitäten innerhalb eines Netzwerks miteinander verbunden sind. Die Projektgruppe fokussiert sich bei der Entwicklung der Ausführungsplattform auf das Anwendungsgebiet Logistik und insbesondere auf Materialflusssysteme, da sie vom Lehrstuhl für Förder- und Lagerwesen der Fakultät für Maschinenbau angeboten und betreut wird. In diesem Abschnitt

wird daher am konkreten Beispiel der Materialflusssysteme die Motivation für die zu entwickelnde Ausführungsplattform aufgezeigt und näher auf die Gründe für dessen Notwendigkeit eingegangen.

Die grundlegenden Anforderungen an heutige Materialflusssysteme sind der Betrieb über einen langen Zeitraum, damit sich die hohen Anschaffungskosten amortisieren, sowie eine hohe Flexibilität begründet durch die zunehmende Dynamik logistischer Prozesse aufgrund immer kürzer werdender Produktzyklen und sich schnell verändernden Marktsituationen (siehe [FLt+09]). Durch die geforderte Flexibilität ergibt sich die Anforderung bestehende Materialflusssysteme mit geringem Zeit- und Kostenaufwand anpassen zu können und Stillstandszeiten weitestgehend zu vermeiden und lokal zu begrenzen.

Heutige Materialflusssysteme besitzen auf Grund ihrer Komplexität einige Nachteile. Einige davon sind in [Gt10, S. 18ff] sowie in Unterkapitel 4.2 beschrieben. Insbesondere sind die mangelnde Ausfallsicherheit bei zentral gesteuerten Systemen und die zeit- und kostenaufwändigen Anpassungen in Zusammenhang mit langen Stillstandszeiten der gesamten Anlage zu nennen. Mit zunehmender Komplexität der Materialflusssysteme wirken sich diese Nachteile ebenfalls verstärkt aus.

Anhand der beschriebenen Nachteile wird deutlich, dass heutige Materialflusssysteme nicht allen Anforderungen gerecht werden können. Aktuelle Forschungsarbeiten wie z. B. [FLt+09] und [FFt10] nehmen sich diesem Problem an und zeigen Ansätze auf, mit denen die vorherrschende Komplexität von Materialflusssystemen aufgebrochen werden kann. Dabei schlagen die Arbeiten Konzepte der Modularisierung und Dezentralisierung vor, sodass einzelne Teile von Materialflusssystemen weitestgehend autonom agieren können und sich neue Anlagenmodule unter Verzicht von aufwändigen Softwareanpassungen in das bestehende System integrieren lassen. Solche dezentralen Systeme sollen auf Ausfälle eigenständig reagieren und ihre Auswirkungen lokal begrenzen können.

[FLt+09] beschreibt eine mögliche Realisierung der Verlagerung von Intelligenz in die Feldebene, indem diese mit Rechenressourcen ausgestattet wird. Mit Hilfe der Rechenressourcen werden die Anlagenmodule in die Lage versetzt ihre Funktionalitäten innerhalb einer Serviceorientierten Architektur anzubieten und ermöglichen so eine dynamische Kopplung von verschiedenen Diensten während dem Betrieb der Anlage. [FFt10] erweitert diesen Ansatz um ein Steuerungskonzept mit dem Namen Paket Royale, dass sich in die Serviceorientierte Architektur eingliedert und die Steuerungsaufgaben ebenfalls dezentral auf die Rechenressourcen der Feldebene verteilt. Beide Ansätzen werden ausführlich im Unterkapitel 4.4 vorgestellt. Ein weiteres aktuelles Forschungsthema ist die Entwicklung der in Unterkapitel 4.5 vorgestellten Smart Neighbourhood Module, denen das Konzept der Modularisierung zu Grunde liegt. Sie sind in der Lage eigenständig mit ihren benachbarten Modulen zu interagieren, um Informationen bzgl. ihrer Nachbarschaft auszutauschen und ihr Funktionsangebot entsprechend der Ausstattung der benachbarten Modulen anpassen zu können.

Die in der Feldebene verfügbaren Rechen- und Speicherressourcen wie beispielsweise die in Abschnitt 4.3.1 vorgestellten IPCs verfügen über Rechenleistungen von über 1 GHz und besitzen teilweise sogar mehrere Prozessorkerne. Die zuvor beschriebenen Ansätze nutzen diese Ressourcen nicht vollständig aus, sodass sich freie Kapazitäten ergeben. Das verfügbar machen dieser freien Ressourcen ist die Hauptaufgabe der zu entwickelnden Ausführungsplattform. Das konkrete Einsatzgebiet ist eine Förderanlage des Lehrstuhls für Förder- und Lagerwesen.

Die Leitsoftware, die in bisherigen Materialflusssystemen als ein monolithischer Softwareblock auf zentralen Materialflussrechnern ausgeführt wird, soll modularisiert werden, sodass einzelne Anlagenkomponenten jeweils Teile der übergeordneten Steuerungssoftware mitliefern können. Die Ausführung der von den Anlagenkomponenten mitgelieferten Software soll dabei innerhalb der zu entwickelnden Ausführungsplattform erfolgen. Damit die von den Anlagenkomponenten mitgelieferten Softwareteile im Gesamtsystem miteinander interagieren können, ist es eine Möglichkeit, diese in Form von Webservices zu realisieren, welche eine lose Kopplung zur Laufzeit ermöglichen. Die von der Projektgruppe zu entwickelnde Ausführungsplattform ermöglicht deshalb die Ausführung von Webservices auf den Rechenressourcen der Anlagenkomponenten und stellt eine lastabhängige Verteilung sicher, um Überlastungen einzelner Ressourcen und somit Beeinträchtigungen der zeitkritischen operativen Steuerung zu vermeiden.

Für die Umsetzung der Ausführungsplattform wird das Konzept des *Cloud-Computing* (siehe Unterkapitel 4.6) angewandt. Es ermöglicht eine Abstraktion von einzelnen Recheneinheiten und stellt nach außen hin eine einheitliche Schnittstelle zur Nutzung der einzelnen Ressourcen zur Verfügung. Die konkreten Anforderungen an die zu entwickelnde Ausführungsplattform sind in den folgenden Abschnitten beschrieben.

5.2 Anforderungen

In diesem Abschnitt sind alle Kriterien beschrieben, die als Anforderungen vorliegen. Diese Kriterien sind durch eine Unterteilung in Muss-, Soll- und Kannkriterien nach Wichtigkeit absteigend sortiert. Musskriterien unterliegen strengen Richtlinien und müssen nachweislich umgesetzt sein und dienen somit der Bewertung. Soll- und Kannkriterien hingegen sind nicht zwingend erforderlich, wobei die Umsetzung der Sollkriterien wiederum wünschenswerter ist als die der Kannkriterien. In Abschnitt 5.2.1 werden die Musskriterien vorgestellt. Die Sollkriterien sind in Abschnitt 5.2.2 und die Kannkriterien in in Abschnitt 5.2.3 zu finden.

5.2.1 Musskriterien

In diesem Kapitel werden die Anforderungen beschrieben, die zwingend umgesetzt werden müssen. Die Anforderungen sind im Folgenden untergliedert in die Bereiche Fördermodule, Hardwareressourcen und Webservices.

Für die Implementierung ist die Programmiersprache Java vorgesehen um eine Integration der zu entwickelnden Ausführungsplattform in die bestehenden Steuerung zu ermöglichen. Die in den folgenden Kriterien genannten Webservices müssen in der Programmiersprache Java realisiert sein und dem DPWS-Standard genügen.

Fördermodule Eine Förderanlage besteht aus verschiedenen Fördermodulen, wie beispielsweise Geradeausförderern, Kurvenelementen oder Zusammenführungen (siehe Unterkapitel 4.1). Es soll zur Laufzeit möglich sein, Fördermodule zur bestehenden Förderanlage hinzuzufügen oder zu entfernen. Spätestens bei einem erneuten Start der Anlage muss das neue Modul bzw. das Fehlen eines Moduls erkannt und in der anschließend neu berechneten Anlagentopologie berücksichtigt werden.

Innerhalb einer Förderanlage sind unterschiedliche Hardwareressourcen, wie beispielsweise IPCs, verteilt. An jede Hardwareressource ist mindestens ein Fördermodul angeschlossen, jedes Fördermodul wird dabei von einer Ressouce gesteuert.

Hardwareressourcen Die Förderanlage ist mit verschiedenen Hardwareressourcen ausgestattet, welche die Anlage mit Rechenkapazität ausstatten. Solche Ressourcen müssen zur Laufzeit zur Anlage hinzugefügt oder entfernt werden können, ohne dass Daten verloren gehen oder das System ausfällt.

Nach dem Anschließen der Hardwareressourcen an die Anlage muss eine Basissoftware gestartet werden. Anschließend müssen die Ressourcen automatisch erkannt und zum Gesamtsystem der Anlage hinzugefügt werden. Danach stehen sie zur Benutzung zur Verfügung. Zum Entfernen von Hardwareressourcen muss eine Abmeldefunktion realisiert sein, die die Hardwareressourcen vor dem Herunterfahren bei einem globalen Dienst abmeldet. Ein globaler Dienst ist eine Software, die über eine fest definierte Schnittstelle angesprochen werden kann und bei Verwendung eine bestimmte Funktionalität bereitstellt. Bewegliche Webservices, die auf der entfernten Ressource ausgeführt wurden, müssen anschließend auf einer anderen Ressource gestartet werden. Bei der Verteilung von Webservices muss das Fehlen einer entfernten Ressource im weiteren Verlauf berücksichtigt werden.

Damit Webservices lastabhängig auf den Rechenressourcen der Anlage verteilt werden können, muss die Möglichkeit bestehen, die Auslastung von Hardwareressourcen zur Laufzeit zu ermitteln. Diese Funktionalität wird für die Verteilung von Prozessen genutzt. Es ist dabei wichtig, dass Änderungen der Auslastung zeitnah erkannt werden, damit eine Ressource direkt nach Beendigung eines Prozesses für weitere Aufgaben zur Verfügung steht. Dies soll eine Beeinträchtigung der Anlagensteuerung bei Ressourcenengpässen vermeiden. Des Weiteren muss sichergestellt werden, dass Ausfälle von Hardwareressourcen, also Rechenleistung und/oder Speicherkapazität, automatisch erkannt und behandelt werden.

Die Zielplattform ist entweder eine x86- oder ARM-Architektur. Darüberhinaus sind weitere Plattformen optional.

Es muss dem Administrator des Anlagensystems eine Administrationsoberfläche zur Verfügung stehen, mit der er sowohl Webservices starten und stoppen als auch Statistiken erstellen kann.

Webservices Alle Funktionen liegen in Form von Webservices vor. Ein Teil der Webservices bezieht sich direkt auf die Funktion eines Fördermoduls und muss deshalb auf der Rechenressource ausgeführt werden, an die das Födermodul angeschlossen ist. Dies können beispielsweise Steuerfunktionen für Fördermodule sein. Diese Webservices werden im Folgenden als *lokale Webservices* bezeichnet. Andere Webservices, die beispielsweise für die Topologieberechnung zuständig und an keine Rechenressource gebunden sind, können auf jedem Knoten ausgeführt werden, die zum Ausführungszeitpunkt über genügend freie Kapazitäten verfügen. Der Endpunkt dieser Webservices (siehe Unterkapitel 4.3.3) ist also flexibel und kann sich über die Zeit hinweg verändern. Solche Webservices werden im Folgenden als *bewegliche Webservices* bezeichnet. Sowohl lokale als auch bewegliche Webservices können parallele Algorithmen enthalten, die verteilt auf dem System aus Rechenressourcen ausgeführt werden. Verteiltes Rechnen wird daher unterstützt.

Damit Webservices genutzt werden können, müssen sie über ein Netzwerk zur Verfügung gestellt werden, an das alle Komponenten der Anlage angeschlossen sind. Es muss sichergestellt sein, dass die Webservices zu jeder Zeit verfügbar sind, damit sie genutzt werden können. Dies gilt besonders für bewegliche Webservices. Fällt ein Teil des Systems bestehend aus Fördermodulen und Hardwareressourcen aus, so darf dies nicht zum Verlust von beweglichen Webservices führen. Es muss deshalb sichergestellt sein, dass bewegliche Webservices zu jeder Zeit auf redundant vielen und geeignet in der Topologie verteilt angeordneten Endpunkten angeboten werden. Für lokale Webservices ist dies kein erforderliches Kriterium, da das an die ausfallende Hardwareressource gekoppelte Fördermodul, für das der Webservice bereitgestellt wird, ebenfalls bei einem Ausfall nicht mehr zur Verfügung steht.

Angebotene Webservices müssen durch eine Suchfunktion auffindbar sein. Durch eine Suche gefundene Webservices werden im Folgenden als *bekannte Webservices* bezeichnet. Die Operationen, die durch bekannte Webservices bereitgestellt werden, müssen nutzbar sein.

Des Weiteren muss es die Möglichkeit geben, neue Software in Form von Webservices in das System zu integrieren. Die Integration der neu hinzukommenden Webservices in das bestehende System muss automatisch geschehen. Dies muss während des laufenden Betriebes möglich sein. Entsprechend dazu muss es ebenfalls möglich sein, Webservices aus dem Netzwerk zu entfernen. Dabei ist nicht relevant, ob die Webservices bereits bekannt sind. Alle zu den Webservices zugehörigen Komponenten und Daten müssen dabei automatisch aus dem System gelöscht werden. Dies muss auch während des laufenden Betriebes möglich sein. Bereits bestehende und möglicherweise bekannte Webservices müssen durch Einbringen veränderter Dateien, die Binärdaten der Webservices enthalten, verändert werden können. Auf Hinzufügen, Entfernen und Aktualisieren von Webservices muss das System automatisch reagieren.

Dem Benutzer muss eine Datenbank zur Verfügung gestellt werden, die unabhängig vom Rest des Systems nutzbar ist. In dieser Datenbank können beispielsweise Statistik-Daten hinterlegt werden.

Die Implementierung enthält die Umsetzung eines beispielhaften Services.

5.2.2 Sollkriterien

In diesem Abschnitt werden Kriterien beschrieben, deren Umsetzung wünschenswert aber nicht zwingend erforderlich ist. Die Umsetzung dieser Kriterien hat eine höhere Priorität als die der in Abschnitt 5.2.3 vorgestellten Kannkriterien.

Die Fördermodule der Anlage sind miteinander verbunden und sollen Informationen austauschen können, über welche Funktionen sie und ihre benachbarten Fördermodule verfügen. Es soll daher für jedes Fördermodul ein Webservice zur Verfügung stehen, der die Namen in Form einer UUID und die Funktionen der Nachbarschaft erkundet und bekannt macht. Die Nachbarschaft eines Anlagenmoduls besteht aus den angrenzenden Modulen.

Eine weitere Eigenschaft, die implementiert werden soll, ist das Anbieten von Speicherplatz im Sinne des Cloud-Modells *Infrastructure as a Service* (siehe dazu 4.6.1).

5.2.3 Kannkriterien

In diesem Abschnitt werden solche Kriterien beschrieben, deren Umsetzung wünschenswert aber nicht zwingend erforderlich ist. Sie sind wie die Sollkriterien optional, haben aber eine geringere Priorität.

Eine wünschenswerte Eigenschaft der zu entwickelnden Ausführungsplattform ist es, neben den Java-basierten Webservices auch solche Webservices zu unterstützen, die in anderen Programmiersprachen umgesetzt sind. Eine Bedingung ist dabei, dass sie sich an den in Unterkapitel 4.3.3 vorgestellten DPWS-Standard halten.

Webservices haben unterschiedliche Prioritäten. Während Webservices, die für die Steuerung der Fördermodule zuständig sind, zu jeder Zeit zuverlässig und schnell verfügbar sein müssen, kann ein Webservice, der eine unabhängige Funktion verwaltet, wie beispielsweise einen Teil eines ERP-Systems, auf seine Ausführung warten bis wieder freie Rechenkapazität vorhanden ist. Es können deshalb Prioritäten für Webservices vergeben werden, an denen sich deren Ausführungsreihenfolge orientiert. Dabei sollen steuerungsrelevante Webservices eine höhere Priorität bekommen, da sie niemals auf ihre Ausführung warten dürfen. Die Prioritäten sollen veränderbar sein.

5.3 Anwendungsfälle

Die Anwendungsfälle sind in Abbildung 5.1 grafisch dargestellt und werden im Folgenden näher erläutert.

Die Anwendungsfälle *P101* bis *P105* decken die Funktionalitäten ab, die direkt mit den Fördermodulen zusammenhängen. Hierunter fallen das Hinzufügen und Entfernen von Fördermodulen, die Nachbarschaftserkennung sowie das Berechnen und Anzeigen der sich durch die Anordnung der Fördermodule ergebenden Anlagentopologie.

P101: Neues Fördermodul hinzufügen
Der Betreiber der Anlage erweitert die bestehende Anlage um ein neues Fördermodul wie z. B. einen Geradeausförderer. Nachdem die technische Installation und Einrichtung abgeschlossen und insbesondere alle zur Kommunikation mit dem neuen Fördermodul nötigen Verbindungen hergestellt sind, muss das neue Fördermodul erkannt und die bestehende Anlagentopologie entsprechend aktualisiert werden. Für die Berechnung der Anlagentopologie wird auf die Nachbarschaftserkennungsfunktionalität zurückgegriffen, die im Anwendungsfall *P103* erläutert wird. Falls das neue Fördermodul Software in Form von Webservices mitliefert, so kann diese, wie im Anwendungsfall *P301* beschrieben, installiert werden.

P102: Fördermodul entfernen
Der Betreiber der Anlage entfernt ein Fördermodul aus der bestehenden Anlage. Hierzu bietet ein Client mit einer grafischen Benutzerschnittstelle eine entsprechende Funktionalität an. Der Betreiber der Anlage wählt im Client das zu entfernende Fördermodul aus. Anschließend wird dafür gesorgt, dass die bestehende Anlagentopologie entsprechend aktualisiert wird und die zu diesem Modul gehörenden Webservices deinstalliert werden. Die zu den deinstallierten Webservices gehörenden Daten, die sich in einem verteilten Datenspeicher auf verschiedenen Knoten im Netzwerk befinden, werden automatisch

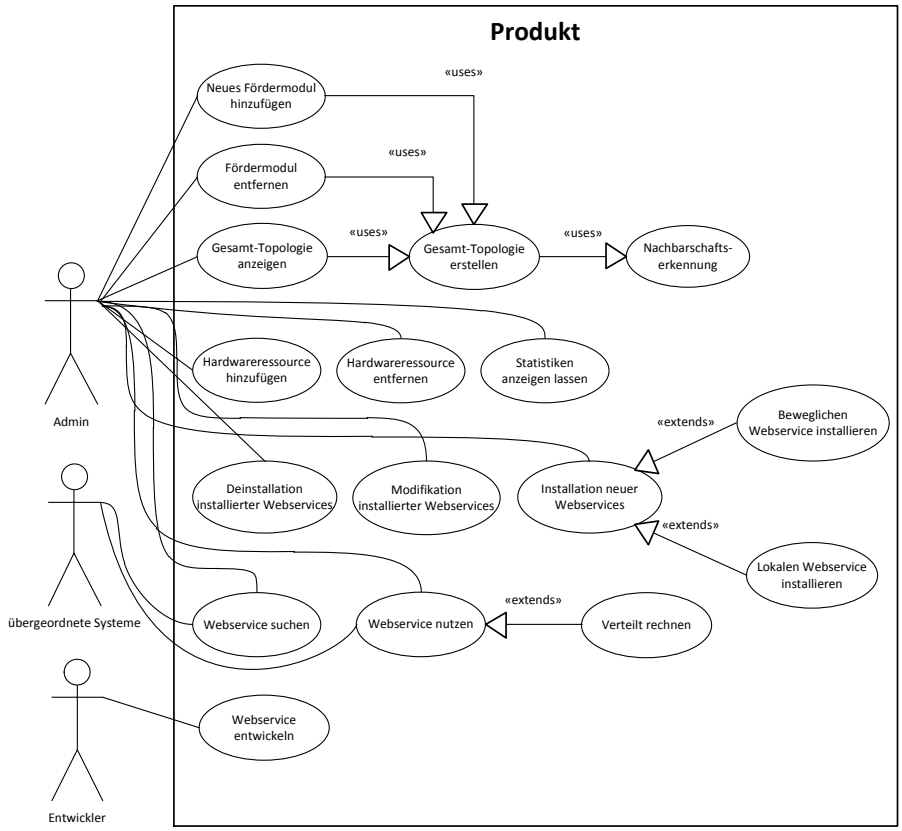


Abbildung 5.1: Anwendungsfalldiagramm

gelöscht. Die von diesen Webservices angebotenen Funktionalitäten stehen anschließend nicht mehr zur Verfügung.

P103: Nachbarschaftserkennung
Für die Berechnung einer Gesamttopologie der Anlage, die Informationen über alle angeschlossenen Module enthält, ist eine Nachbarschaftserkennung erforderlich. Bei Umsetzung des entsprechenden Sollkriteriums verfügt jedes Modul der Anlage über entsprechende Hardware, die über verschiedene Anschlüsse mit den jeweils benachbarten Modulen verbunden ist. Die Kommunikation wird über Ethernet durchgeführt. Hierfür ist es notwendig, ein Protokoll zu spezifizieren, das den genauen Ablauf der Kommunikation definiert. Über dieses Ethernet-Protokoll ist es möglich, die Nachbarschaftsinformationen eines Moduls von der an diesem Modul angeschlossenen Hardware zu erhalten. Diese Nachbarschaftsinformationen bestehen pro benachbartem Modul u. a. aus einer eindeutigen Kennung des Moduls (UUID) und dem Modulausgang, an dem das Modul angeschlossen ist. Falls dieses Sollkriterium nicht umgesetzt wird, erfolgt die Nachbarschaftserkennung durch Auslesen von Konfigurationsdateien, die die Kennung und den entsprechenden Modulausgang der benachbarten Module enthalten. Diese Konfigurationsdateien müssen vom Administrator für jedes Modul manuell erstellt und bei Änderungen an der Anlagentopologie aktuell gehalten werden. Die Nachbarschaftserkennung wird gestartet, wenn ein

neues Modul der Anlage hinzugefügt oder entfernt wird und somit die Anlagentopologie aktualisiert werden muss.

P104: Gesamttopologie erstellen
Dieser Anwendungsfall deckt das Erstellen einer Gesamttopologie ab. Mit Hilfe der Nachbarschaftserkennung wird die Topologie der Anlage erkundet und eine Gesamttopologie erstellt. Dieser Anwendungsfall kann sowohl dazu verwendet werden, um initial die Anlagentopologie zu bestimmen als auch dazu, die Topologie nach dem Hinzufügen oder Entfernen von Anlagenmodulen zu aktualisieren. Diese Funktionalität wird als beweglicher Webservice umgesetzt, sodass die Erstellung der Gesamttopologie von allen im Netzwerk vorhandenen Knoten bei Bedarf durchgeführt werden kann. Die erstellte Gesamttopologie wird in Form eines Topologiegraphen repräsentiert.

P105: Gesamttopologie anzeigen
Der Administrator lässt sich die Topologie der Förderanlage anzeigen. Hierfür bietet der Client mit einer grafischen Benutzerschnittstelle eine entsprechende Funktionalität an, mit der eine Übersicht der Gesamttopologie der Anlage aufgerufen werden kann.

Die Anwendungsfälle *P201* bis *P203* beschreiben die Funktionalitäten, die sich den Hardwareressourcen zuordnen lassen. Diese decken im Wesentlichen das Hinzufügen und Entfernen von Hardwareressourcen sowie das Anzeigen von Auslastungsstatistiken ab.

P201: Neue Hardwareressource hinzufügen
Der Administrator der Förderanlage integriert eine neue Hardwareressource in das System. Damit die neue Hardwareressource erkannt werden kann, ist es bei Rechenressourcen notwendig, dass der Administrator zunächst die Rechenressource durch eine Softwareinstallation vorbereitet und diese an das Netzwerk anschließt. Im Netzwerk wird ein Knoten zum so genannten *Leader* bestimmt. Dies geschieht durch ein Verfahren, das ein Knoten durch eine Broadcast-Nachricht initiiert. Hierbei generiert jeder Knoten eine ID, die im einfachsten Fall zufällig ist, aber auch auf aktuellen Auslastungsdaten beruhen kann. Derjenige Knoten mit der höchsten ID gewinnt die Leaderwahl und fungiert anschließend im Netzwerk als Leader. Das Verfahren ist dabei so ausgelegt, dass es parallele Leaderwahlen erkennt und zu einer gemeinsamen Wahl kombiniert. Die neue Rechenressource erkennt bei Eintritt in das Netzwerk, dass bereits ein Knoten als Leader fungiert und meldet sich als neuer Knoten bei dem Leader-Knoten an. Dieser ist für die vollständige Integration des neuen Knotens in das Netzwerk zuständig und sorgt dafür, dass die von dem neuen Knoten angebotenen beweglichen Webservices im gesamten Netzwerk verteilt und zur Sicherstellung der Verfügbarkeit mehrfach angeboten werden. Falls zum Zeitpunkt des Hinzufügens der neuen Rechenressource kein Leader im Netzwerk vorhanden ist, initiiert die neue Rechenressource eine Leaderwahl. Speicherressourcen werden dafür verwendet um Speicherplatz für den verteilten Datenspeicher bereitzustellen. Dieser beinhaltet die Binärdateien und Konfigurationsdaten der installierten Webservices sowie bei der Ausführung anfallende Daten, wie z. B. Auslastungsdaten der einzelnen Knoten. Das Hinzufügen von reinen Speicherressourcen wird nicht direkt unterstützt. Da als Speicherort jedoch nur ein in der Knotenkonfiguration angegebener Order verwendet wird, besteht für den Administrator die Möglichkeit, an diese Stelle beliebige Speichermedien in das genutzte Betriebssystem einzubinden und so Einfluss auf die verfügbare Speicherkapazität zu nehmen. Änderungen an der Konfiguration des Speicherordners sind zur Laufzeit nicht

möglich. Bei Umsetzung des entsprechenden Kannkriteriums kann dieser Speicherplatz nach außen hin für den Administrator zur Verfügung gestellt werden.

P202: Hardwareressource entfernen
Der Administrator entfernt eine Hardwareressource aus der Anlage. Falls es sich dabei um eine Rechenressource handelt, kann er im Client, der eine grafische Oberfläche anbietet, die zu entfernende Rechenressource auswählen. Der Client ruft anschließend einen Softwareteil auf, der im Netzwerk nur auf einem Knoten läuft und die Verfügbarkeit von beweglichen Webservices sicherstellt. Dieser Softwareteil sorgt dafür, dass alle beweglichen Webservices des zu entfernenden Knotens nach dem Entfernen weiterhin im Netzwerk auf anderen Knoten verfügbar sind und somit keine beweglichen Webservices versehentlich mitentfernt werden. Weiterhin benötigte Dateien des verteilten Datenspeichers, die sich auf dem zu entfernenden Knoten befinden, werden vor dem Entfernen ebenfalls auf andere Knoten ausgelagert, um deren Verfügbarkeit zu gewährleisten. Nachdem dies abgeschlossen ist, wird die Software auf dem zu entfernenden Knoten beendet. Das Entfernen von Speicherressourcen wird nicht direkt unterstützt. Allerdings besteht, ähnlich zum Anwendungsfall *P201*, für den Administrator die Möglichkeit den Speicherort in der Knotenkonfiguration anzugeben und somit Einfluss auf die verfügbare Speicherkapazität zu nehmen.

P203: Statistiken anzeigen lassen
Der Administrator sieht die Statistiken zur Auslastung der im System vorhandenen Hardwareressourcen ein. Hierfür bietet der grafische Client eine Oberfläche an, die die Auslastung der einzelnen Hardwareressourcen darstellt. Auf jedem Knoten wird zu diesem Zweck ein Softwareteil ausgeführt, über den die Auslastung des Knotens (z. B. CPU-Auslastung, verfügbarer/belegter Arbeitsspeicher und freie/belegte Speicherkapazität) ermittelt werden kann.

Die im folgenden beschriebenen Anwendungsfälle *P301* bis *P305a* beschreiben die Ausführung von Webservices auf der Ausführungsplattform. Dabei wird Bezug auf das Installieren, Modifizieren und Deinstallieren von Webservices genommen und beschrieben wie installierte Webservices genutzt werden können.

P301: Installation neuer Webservices
Der Administrator bringt einen neuen Webservice in das Netzwerk ein. Der zu installierende Webservice muss dabei einer vorgegebenen Schnittstelle entsprechen. Kompatible Webservices müssen die DPWS-Spezifikation erfüllen und als JAR-Datei vorliegen, die zusätzlich zu den eigentlichen Webservices eine Konfigurationsdatei enthält. Die Konfigurtionsdatei muss die eindeutigen Namen und Versionen der in der JAR-Datei definierten Webservices und Devices sowie die Zuordnung welche Webservice von welchem Device angeboten werden enthalten. Dies ist erforderlich, da eine JAR-Datei mehrere Webservices und Devices enthalten kann und die Installation, das Starten und das Loadbalancing der Webservices eine Zuordnung zur sie beinhaltenden JAR-Datei herstellen können müssen. Weiterhin muss der Typ der Webservices (lokal oder beweglich) spezifiziert sein und im Fall der Umsetzung des Kannkriteriums bzgl. der Vergabe von Prioritäten für Webservices, müssen diese ebenfalls in der Konfigurationsdatei angegeben werden. Ist die Konfigurationsdatei vom Entwickler des Webservices nicht mitgeliefert, so muss diese vom Administrator selbst erstellt werden. Da alle Webservices innerhalb einer JAR-Datei

vom *Load-Balancer* als eine Einheit aufgefasst werden, wird empfohlen, dass eine JAR-Datei aus genau einem Device besteht, um eine feine Granularität des Load Balancings zu ermöglichen. Für die Installation kompatibler Webservices werden drei verschiedene Verfahren angeboten. Eine Möglichkeit ist es, die Installation über den Client durchzuführen. In diesem kann die zu installierende JAR-Datei ausgewählt und installiert werden. Der Client leitet die JAR-Datei zur Deploymentschnittstelle eines beliebigen Knotens im Netzwerk weiter. Diese umfasst alle zur Installation und zum Start notwendigen Funktionen. Hierunter fallen Auswahl der Knoten, auf denen der Webservice angeboten werden soll, das Verteilen der JAR-Datei im verteilten Datenspeicher für die Sicherstellung der Verfügbarkeit und das Starten und Stoppen von Webservices.

Die zweite Möglichkeit zur Installation kompatibler Webservices besteht darin, einen USB-Stick an einen Knoten anzustecken. USB-Sticks, die sich als Massenspeicher zu erkennen geben, werden von der auf jedem Knoten gestarteten Deploymentschnittstelle automatisch nach kompatiblen Webservices durchsucht. Die gefundenen Webservices werden anschließend automatisch installiert.

Als dritte Möglichkeit zur Installation von kompatiblen Webservices besteht die Möglichkeit die JAR-Datei direkt in einen von der Deploymentschnittstelle auf Änderungen überwachten Ordner zu kopieren. Die Deploymentschnittstelle erkennt die neue JAR-Datei und startet das Deployment. Je nach Typ des neuen Webservices (beweglich oder lokal) variiert jeweils der Ablauf des Deployments. Die folgenden beiden Anwendungsfälle *P301a* und *P301b* beschreiben für diese beiden Typen den konkreten Ablauf des Deployments, nachdem der neue Webservice auf eine der drei beschriebenen Wege ins System eingebracht wurde.

Für den Fall, dass der zu installierende Webservice nicht kompatibel ist, wird in allen drei Installationsvarianten eine Fehlermeldung generiert und in eine Log-Datei geschrieben, die über den grafischen Client eingesehen werden kann.

P301a: Lokalen Webservice installieren
Dieser Anwendungsfall erweitert den Anwendungsfall *P301*. Bei einem lokalen Webservice muss innerhalb seiner Konfigurationsdatei spezifiziert sein, dass es sich um einen lokalen Webservice handelt. Erfolgt die Installation über den Client, muss während der Installation derjenige Knoten ausgewählt werden, der den neuen Webservice anbieten soll, sodass die JAR-Datei des Webservices an die Deploymentschnittstelle dieses Knotens geschickt werden kann. Bei den anderen Installationsvarianten wird der neue Webservice von demjenigen Knoten angeobten, an die der USB-Stick angesteckt bzw. in dessen Ordner die JAR-Datei kopiert worden ist. Das Deployment des Knotens, der den neuen Webservice anbieten soll, übernimmt das Starten des neuen Webservices. Dieser wird ausschließlich von diesem Knoten im Netzwerk angeboten. Typische Vertreter lokaler Webservices sind solche, die die direkte Steuerung der Aktoren eines Moduls übernehmen und deshalb nur auf jenen Knoten lauffähig sind, die mit diesen Aktoren direkt verbunden ist. Lokale Webservices können für parallele Berechnungen GridGain verwenden. GridGain wird zur Verfügung gestellt und startet auf allen Knoten eine GridGain-Node, sodass parallele Berechnungen unter Einhaltung der Schnittstelle von GridGain potentiell über alle im Netzwerk vorhandenen Knoten durchgeführt werden können.

P301b: Beweglichen Webservice installieren
Dieser Anwendungsfall erweitert den Anwendungsfall *P301*. Bei beweglichen Webser-

vices muss innerhalb der Konfigurationsdatei vermerkt sein, dass es sich um bewegliche Webservices handelt. Das Deployment ermittelt auf Grundlage der aktuellen Auslastungen der einzelnen Knoten diejenigen Knoten auf denen der neue Webservice angeboten werden soll. Um die Verfügbarkeit des neuen Webservices zu gewährleisten, wird dieser von mehreren Knoten mit ausreichend freien Ressourcen parallel angeboten. Die Auslastung der einzelnen Knoten kann das Deployment aus dem verteilten Datenspeicher entnehmen, in dem eine Datei gespeichert wird, die die aktuellen Auslastungen der Knoten enthält und in regelmäßigen Abständen durch die auf jedem Knoten gestartete Ressourcenüberwachung aktualisiert wird. Nachdem die Knoten ausgewählt sind, die den neuen Webservice anbieten sollen, werden die Daten des neuen Webservices (JAR-Datei und Konfigurationsdatei) zu den ausgewählten Knoten übertragen. Das Deployment auf den Ziel-Knoten sorgt anschließend für das Starten des neuen Webservices. Auch bewegliche Webservices können für parallele Berechnungen auf GridGain zurückgreifen und die Berechnungen über das Netzwerk parallelisieren.

P302: Modifikation installierter Webservices
Ein bereits installierter Webservice wird auf eine neue Version aktualisiert. Als Konvention zur Sicherstellung von Abwärtskompatibilität ist dabei zwingend zu beachten, dass die Beibehaltung des selben Namens für einen Webservice nur dann erlaubt ist, wenn der Webservice rückwärtskompatibel ist und sich seine Ein- und Ausgabeparameter in der neuen Version nicht verändert haben. Andernfalls ist für den aktualisierten Webservice ein neuer Namen zu vergeben. Bei einem Update eines Webservices ist es weiterhin zwingend erforderlich, dass die JAR-Datei des aktualisierten Webservice den selben Dateinamen besitzt wie der ursprünglich installierte Webservice. Der Update-Prozess erfolgt nun ähnlich dem Installieren eines neuen Webservices. Das Deployment erkennt, wenn die aktualisierte JAR-Datei in den überwachten Ordner eingefügt wird. Stellt das Deployment fest, dass dieser Webservice bereits im Netzwerk existiert, wird die aktualisierte Datei an alle den Webservice anbietenden Knoten übertragen und dort der Webservice in der aktualisierten Version neu gestartet. Weiterhin werden alle Kopien der ursprünglichen Version im verteilten Dateispeicher aktualisiert. Nach erfolgtem Update führt ein Aufruf des aktualisierten Webservices die neue Version aus.

P303: Deinstallation installierter Webservices
Der Administrator entfernt einen installierten Webservice. Hierfür bietet der Client eine Funktion an, mit deren Hilfe der Administrator den zu entfernenden Webservice auswählen kann. Der Client ermittelt auf welchen Knoten dieser Webservice ausgeführt wird und delegiert das Stoppen des Webservices an die Deployment-Schnittstellen dieser Knoten. Weiterhin veranlasst der Client das Löschen aller zu diesem Webservice gehörenden Daten im verteilten Datenspeicher. Anschließend ist der gelöschte Webservice weder im Netzwerk bekannt noch im verteilten Datenspeicher vorhanden.

P304: Webservice suchen
Es gibt die Möglichkeit, Webservices mit bestimmten Funktionalitäten zu suchen. Die Rückgabe besteht bei lokalen Webservices aus Netzwerkadressen, wie z. B. einer Endpunktreferenz oder IP-Adresse desjenigen Knotens, der den gesuchten Webservice anbietet. Bei beweglichen Webservices erfolgt eine Interaktion mit dem *Load-Balancer*. Es wird versucht das Load-Balancing transparent umzusetzen, d. h. ohne Anforderungen an Client und Webservice. Hierfür wird ein Discovery Proxy implementiert, der als Suchergebnis die Endpunktreferenz des zum gesuchten Webservice gehörenden Load-Balancers zurückgibt.

Ruft der Client anschließend den Webservice auf, leitet der Load Balancer diesen Aufruf lastabhängig an einen wenig ausgelasteten Knoten weiter. Aufgrund des clientseitigen Cachings von Endpunktreferenzen kann diese Umsetzung mittels Discovery Proxy nicht garantiert werden. Es wird in diesem Fall eine von zwei Alternativen ermöglicht. Entweder ist ein Load-Balancing nur bei deaktiviertem Caching der Endpunktreferenzen auf Clientseite möglich oder die Clients sorgen eigenständig für eine lastabhängige Verteilung ihrer Aufrufe, indem sie die Aufrufe gleichmäßig auf alle den Webservice anbietenden Knoten verteilen.

P305: Webservice nutzen
Sind Endpunktreferenzen von Webservices durch eine Suche gefunden worden, kann der gesuchte Webservice aufgerufen und genutzt werden. Dies gilt sowohl für lokale als auch für bewegliche Webservices.

P305a: Verteilt rechnen
Erweitert den Anwendungsfall „Webservice nutzen“ bezüglich der Möglichkeit einen Webservice zu nutzen, der eine verteilte Berechnung über GridGain auf mehreren Rechenressourcen parallel durchführt. Der Entwickler des Webservices muss dabei den Webservice bereits bei der Entwicklung an die GridGain-Schnittstelle anpassen, sodass der Webservice direkt auf die auf den Knoten gestarteten GridGain-Nodes zugreifen kann.

Der Anwendungsfall *P401* deckt das Entwickeln von Webservices für die Ausführungsplattform ab. Im Gegensatz zu den vorangegangenen Anwendungsfällen ist für die Entwicklung von Webservices nicht der Administrator zuständig sondern ein Softwareentwickler.

P401: Webservice entwickeln
Der Entwickler entwickelt einen kompatiblen Webservice. Sofern das Kannkriterium bzgl. der Ausführbarkeit von Webservices, die in anderen Programmiersprachen als Java implementiert sind, nicht umgesetzt wird, ist es zwingend erforderlich, dass der Entwickler für die Entwicklung eines Webservices die Programmiersprache Java verwendet.

Abschließend werden die Anwendungsfälle *P501* bis *P503* vorgestellt. Diese befassen sich mit einer lastabhängigen Verteilung von Webservices, der Sicherstellung ihrer Verfügbarkeit sowie einer Möglichkeit zur Priorisierung von Webservices.

P501: Lastabhängige Verteilung von Webservices
Bewegliche Webservices werden so auf die einzelnen Knoten verteilt, sodass sich eine möglichst gleichmäßige Auslastung der einzelnen Knoten ergibt und kein Knoten überlastet ist. Für die Umsetzung wird ein Softwareteil entwickelt, der *UNO* genannt wird. Der Name *UNO* ist dabei keine Abkürzung und besitzt keine nähere Bedeutung. Er wurde in Anlehnung an die Vereinten Nationen gewählt, da Nachrichten von überlasteten Knoten an die UNO als Hilferufe bezeichnet werden. Die UNO wird auf genau einem Knoten ausgeführt. Stellt ein Knoten fest, dass er überlastet ist, kann dieser die UNO benachrichtigen. Die UNO sorgt anschließend dafür, dass bewegliche Webservices, die der überlastete Knoten anbietet, auf andere weniger ausgelastete Knoten übertragen werden und somit der Knoten entlastet wird. Um Situationen, in denen einzelne Knoten überlastet sind, so weit es möglich ist zu vermeiden, wird für bewegliche Webservices weiterhin ein *Load-Balancing* durchgeführt. Dieses ist im Anwendungsfall *P304* beschrieben und hat die Aufgabe, einzelne Webserviceaufrufe auf aktuell wenig ausgelastete Knoten zu

verteilen und verhindert somit, dass aktuell stark ausgelasteten Knoten weitere Last zugewiesen wird. Da lokale Webservices an einen bestimmten Knoten gebunden sind, ist diese lastabhängige Verteilung lediglich für bewegliche Webservices möglich.

P502: Sicherstellung der Verfügbarkeit von Webservices
Für die Sicherstellung der Verfügbarkeit von beweglichen Webservices werden spezielle Mechanismen angeboten. Für die lokale Sicherstellung der Verfügbarkeit auf einem Knoten wird auf jedem Knoten ein lokaler Watchdog ausgeführt. Dieser überwacht sowohl die internen Prozesse, als auch die installierten Webservices, die auf dem Knoten angeboten werden. Erkennt der lokale Watchdog, dass ein Prozess oder Webservice nicht mehr reagiert, veranlasst dieser den Neustart des betreffenden Prozesses bzw. Webservices. Zur globalen Sicherstellung der Verfügbarkeit im gesamten Netzwerk wird auf genau einem Knoten zusätzlich zum lokalen Watchdog ein globaler Watchdog ausgeführt. Dieser sorgt dafür, dass auf allen im Netzwerk vorhandenen Knoten stets ein lokaler Watchdog vorhanden ist. Fällt ein lokaler Watchdog aus, so initiiert der globale Watchdog den Neustart des lokalen Watchdogs auf dem entsprechenden Knoten. Weiterhin kann durch den globalen Watchdog der Ausfall eines oder mehrerer Knoten erkannt werden. In diesem Fall benachrichtigt der globale Watchdog die UNO, welche dafür sorgt, dass die beweglichen Webservices auf den ausgefallenen Knoten von anderen noch aktiven Knoten weiterhin angeboten werden.

P503: Priorisierung von Webservices
Bei Umsetzung des entsprechenden Kannkriteriums bzgl. der Priorisierbarkeit von Webservices, wird die Möglichkeit angeboten, den Webservices Prioritäten zuzuweisen. Die Zuweisung der Prioritäten erfolgt statisch durch einen Eintrag in den Konfigurationsdateien der Webservices.

5.4 Anforderungen an die Datenhaltung

Aufgabe der Datenhaltung (siehe Abschnitt 4.6.4) ist es, Anwendungsdaten der verschiedenen Komponenten von FiLeCC sowie der auf der Plattform betriebenen Software, in einer verteilten Datenbank (siehe Abschnitt 6.6.2) hochverfügbar, automatisch mit der sich ändernden Knotenanzahl skalierend und mit automatischer Lastverteilung abzuspeichern. Die Komponenten von FiLeCC werden in Abschnitt 5.3, in Form von Anwendungsfällen (siehe z. B. P301-303, P501), vorgestellt und in Kapitel 7 detailliert beschrieben. Die Anforderungen der Komponenten von FiLeCC an die Datenhaltung sollen mittels einer kompakten Programmierschnittstelle, der Speicherschnittstelle, erfüllt und die verteilte Datenbank effizient angebunden werden. Es soll eine datenbankunabhängige Speicherschnittstelle geschaffen werden, die von der Anfragesprache der Datenbank abstrahiert. Die Methoden der Schnittstelle sollen dabei alle benötigten Anfragetypen abdecken und auf Datenbankoperationen abbilden. Dies ermöglicht ggf. ein Austauschen des Datenbanksystems, ohne dass Veränderungen an der Schnittstelle notwendig sind.

Da die Anforderungen durch externe Software, welche mittels FiLeCC betrieben wird, unterschiedlich und im Vorfeld nicht bekannt sind, ist es hier notwendig eine Möglichkeit anzubieten, flexibel auf die Datenbank zugreifen zu können. Es muss lediglich verhindert werden, dass durch externe Software, die auf der Plattform FiLeCC betrieben wird, die internen Daten von FiLeCC verändert werden. Die Umsetzung dieser Schnittstelle ist in Abschnitt 7.9.4 beschrieben. Im

Folgenden sind die Anforderungen an die interne Datenhaltung von FiLeCC, aufgeteilt nach Lastdaten und SDRM-Daten, aufgeführt.

5.4.1 Lastdaten

Die Lastdaten aller Knoten werden durch mehrere Komponenten von FiLeCC ausgewertet. Das Longterm-Loadbalancing durch die UNO (Abschnitt 7.3) stellt eine gleichmäßige Verteilung der Last auf alle Knoten sicher. Bei Ausfall oder Überlastung eines Knotens werden neue Instanzen, der auf dem betroffenen Knoten ausgeführten Software, auf anderen Knoten gestartet. Hierfür werden die Lastdaten der einzelnen Knoten benötigt. Weiterhin benötigt das in Abschnitt 7.2 beschriebene SDRM die Lastdaten, um feststellen zu können, welche Knoten im System vorhanden sind. Um sicherzustellen, dass Software nur auf Knoten verteilt und ausgeführt wird, die über genügend freien Speicher verfügen, werden die Lastdaten durch das SDRM ebenfalls ausgewertet. Die Lastdaten bestehen z. B. aus den verfügbaren Speicherressourcen und der ungenutzten Rechenleistung. Diese Informationen werden vom Ressourcenmonitor, welcher in Kapitel 7.7 beschrieben wird, periodisch aktualisiert.

Die für die Schnittstelle zu den Lastdaten notwendigen Operationen beschränken sich auf das Einfügen bzw. Verändern der Lastdaten eines einzelnen Knotens sowie das Abfragen einer Menge von Lastdaten. Diese Menge kann die Lastdaten aller Knoten umfassen oder aber ausschließlich die Knoten, die über ausreichend freien Hauptspeicher verfügen. In beiden Fällen soll die Menge nach der Prozessorauslastung sortiert sein.

Da Lastdaten den aktuellen Zustand eines Systems beschreiben und nach einem Systemneustart nicht mehr aktuell sind, müssen sie nicht persistent gespeichert werden. Ein Sichern des Verlaufs der Lastdaten ist nicht vorgesehen. Weiterhin muss die Konsistenz der Daten nicht explizit sichergestellt werden, da es keine Datensätze gibt, die von mehreren Knoten verändert werden. Jeder Knoten ändert nur den Datensatz, der die eigenen Lastdaten enthält.

Die sich aus den Lesezugriffen der UNO ergebenden Leistungsanforderungen sind vernachlässigbar, da die Frequenz der Lesezugriffe sehr gering ist. Lesezugriffe der UNO treten lediglich bei Knotenüberlast oder -ausfall auf. Die Anforderungen an die Aktualität und somit die Schreibfrequenz durch den Ressourcenmonitor sind ebenso gering, da die UNO das Longterm-Loadbalancing durchführt, für welches keine hochaktuellen Lastdaten notwendig sind. Eine maximales Alter der Lastdaten von weniger als einer Minute ist ausreichend. Aus den Lesezugriffen des SDRM ergeben sich in etwa die gleichen Anforderungen. Die Lesefrequenz ist vergleichbar mit der der UNO, da das SDRM die Lastdaten bei UNO-Operationen abfragt, welche die Lastdaten ohnehin benötigen. Darüber hinaus benötigt das SDRM die Lastdaten nur bei Administrationseingriffen, welche nicht zum Normalbetrieb gerechnet werden. Zusammengefasst stellen UNO und SDRM geringe Anforderungen an die Leistungsfähigkeit der Datenbank und der Schnittstelle.

5.4.2 Daten des Software Deployment and Runtime Management

Das in Kapitel 7.2 beschriebene SDRM verwaltet Daten welche die in FiLeCC installierte Software beschreiben. Hierfür wird eine Tabelle mit verschiedenen Metadaten für jedes Softwarepaket vorgehalten. Darüber hinaus müssen Informationen, über die Verteilung von Binärdateien sowie Instanzen der Software auf Knoten des Systems, in jeweils einer weiteren Tabelle gespeichert

werden. Die Schnittstelle zu diesen Daten muss Methoden zum Einfügen und Ändern sowie zum Löschen einzelner Datensätze jeder dieser drei Tabellen bieten. Weiterhin wird jeweils eine Methode zum Abfragen aller Datensätze der drei Tabellen benötigt. Für den Zugriff auf die Tabellen mit den Verteilungsdaten von Binärdateien und Softwareinstanzen müssen darüber hinaus Methoden zum Abrufen aller Knoten zu einem gegebenen Softwarepaket, sowie umgekehrt aller Softwarepakete zu einem gegebenen Knoten, bereitgestellt werden.

Die Daten des SDRM müssen im Gegensatz zu den Lastdaten sowohl persistent als auch konsistent gespeichert werden. Die Leistungsanforderungen unterscheiden sich ebenfalls. Für jedes Starten und Stoppen einer Softwareinstanz müssen Daten aus allen drei Tabellen gelesen werden. Dies geschieht abgesehen von Administrationseingriffen in Überlast- oder Ausfallsituationen innerhalb des Systems. Schreibzugriffe auf die Tabelle mit den Metadaten werden bei Neuinstallation, Aktualisierung oder Deinstallation von Software durchgeführt. In diesen Fällen wird ebenfalls die Tabelle mit den Daten für die Verteilung von Binärdateien angepasst. Änderungen an dieser Tabelle sind zusätzlich bei der Berechnung einer neuen Softwareverteilung notwendig. Eine Neuberechnung der Softwareverteilung wird lediglich bei Veränderung der Infrastruktur oder Deployment-Strategie durchgeführt. Die Tabelle, die die Zuordnung von Softwareinstanzen zu Knoten enthält, wird bei Start- und Stoppvorgängen von Softwareinstanzen verändert. Änderungen werden also lediglich an der letztgenannten Tabelle zur Verwaltung der Softwareinstanzen automatisch durch FiLeCC ausgelöst. Änderungen an den zwei anderen Tabellen werden durch Administrationseingriffe ausgelöst. Keine der einen Datenbankzugriff auslösenden Situationen findet mehr als einmal pro Knoten und Minute statt. Zusammengefasst stellt das SDRM geringe Anforderungen an die Leistungsfähigkeit der Datenbank und der Schnittstelle.

5.4.3 Zusammenfassung

Die grundlegenden Anforderungen an die Speicherschnittstelle und Datenhaltung sind eine hohe Verfügbarkeit, Skalierbarkeit und Lastverteilung des Datenhaltungssystems. Dazu sind vollständig verteilte Datenbanksysteme, sowie eine automatische Anpassung an eine sich verändernde Menge von Rechenknoten des Systems notwendig. Weder die Lastdaten noch die Daten des SDRM weisen eine komplexe Struktur auf. In dieser Hinsicht genügen Datenbanksysteme, wie die in 4.6.4 beschriebenen KV-Stores. Darüber hinaus sind die Anforderungen an die Haltung der Lastdaten einerseits und die der Daten des SDRM andererseits gegensätzlich. Während die Lastdaten zeitnah bereitgestellt und häufig angepasst werden müssen, werden für die Daten des SDRM Konsistenz und maximale Ausfallsicherheit gefordert. Die Anforderungen an die Datenbank für externe Software lassen sich nicht allgemein und ohne Kenntnis der zu betreibenden Software ermitteln. Grundvoraussetzung ist allerdings eine vollständige Abgrenzung von den internen Daten von FiLeCC. Welche Datenbanksysteme den beschriebenen Anforderungen gerecht werden, wird in Abschnitt 6.6.2 betrachtet.

5.5 Fazit

Die Umsetzung der in diesem Kapitel vorgestellten Muss-, Soll- und Kannkriterien und den zugehörigen Produktfunktionen erfordert eine detaillierte Auswahl an verfügbaren Technologien sowie eine umfangreiche Planung der Architektur der Software. Die Auswahl verfügbarer

Technologien wird in Kapitel 6 beschrieben. Die in diesem Kapitel im Rahmen der Produktfunktionen vorgestellten Komponenten werden in Kapitel 7 erneut aufgegriffen und im Zuge der Architekturplanung detailliert vorgestellt.

KAPITEL 6

Auswahl verfügbarer Technologien

Wie in Kapitel 5 beschrieben, ist das Ziel der Projektgruppe die Entwicklung einer Cloud-Infrastruktur, um ungenutzte Ressourcen in einer Logistik-Förderanlage für Software nutzen zu können, die derzeit z. B. noch auf zentralen Leitrechnern ausgeführt wird. Da der Begriff „Cloud“ in unterschiedlichen Kontexten verwendet wird, muss der Begriff „Cloud-Infrastruktur“ im vorliegenden Kontext zunächst genauer spezifiziert werden. So gibt es bereits eine große Vielfalt an Technologien und Cloud-Frameworks, die allerdings unter dem Begriff „Cloud“ sehr verschiedene Lösungen anbieten, was anhand der in Abschnitt 4.6 beschriebenen Clouddefinitionen nachvollzogen werden kann. Die Schwierigkeit ist es die adäquate Hard- und Software zu finden, die alle in Kapitel 5.2 beschriebenen Anforderungen erfüllt.

Abbildung 6.1 zeigt ein Schema der Architektur, das sich aus diesen Anforderungen ergibt. In dem Schema sind zwei als „Wolken“ dargestellte Elemente enthalten, welche die einzelnen mit HR1 bis HRn bezeichneten Knoten verbinden: die obere „Kommunikationswolke“ und die untere „Berechnungswolke“. In der Kommunikationswolke können die einzelnen Programme und Programmteile miteinander kommunizieren, was eine sehr lose Kopplung dieser Komponenten ermöglicht. Komponenten können entweder Webservices sein, die extern entwickelt wurden und auf der Plattform ausgeführt werden (im Bild mit WS1 bis WS4 dargestellt) oder Webservices, die zum Betrieb der Plattform notwenig sind (im Bild als FiLeCCWS dargestellt). Die verfügbaren und eingesetzten Kommunikationsprotokolle und -implementierungen werden in den Abschnitten 6.3 und 6.4 ausführlich betrachtet.

Eine Kommunikation ist ebenfalls in der Berechnungswolke nötig, allerdings liegt hier das Schwerpunkt beim verteilten Rechnen. Auf jedem Knoten wird dafür eine spezielle Software-Komponente ausgeführt, die in der Abbildung als „GridNode“ dargestellt ist. Auch für das verteilte Rechnen existieren Lösungen, die in Abschnitt 6.5 vorgestellt werden.

Zudem muss untersucht werden, welche Art von Hard- und Software in der Cloudumgebung als Basis eingesetzt werden kann, um den Einsatz der ausgewählten Technologien zu ermöglichen.

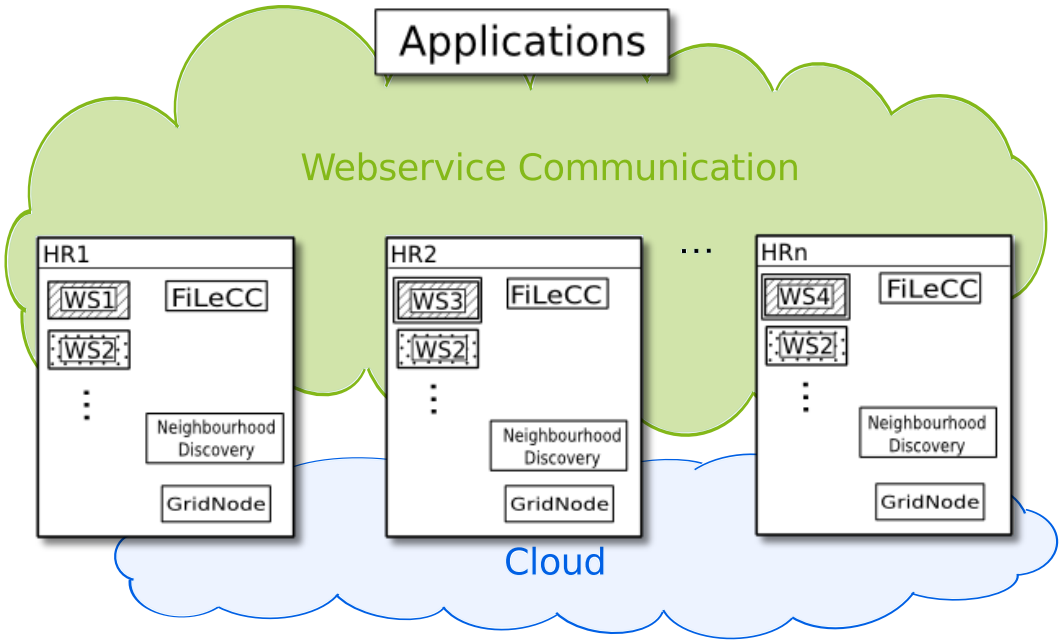


Abbildung 6.1: Architekturübersicht

Dafür werden im Abschnitt 6.1 zunächst typische Hardwareplattformen für Förderanlagen betrachtet, um entscheiden zu können, welche der beschriebenen Laufzeitumgebungen insbesondere Betriebssysteme eingesetzt werden können (vgl. Abschnitt 6.2).

In Abbildung 6.1 wird außerdem der Funktionsblock „Neighbourhood Discovery“ dargestellt. Darunter ist die Erkennung der physikalischen Nachbarn eines Knotens und die daraus berechenbare Gesamttopologie des Systems zu verstehen. Dazu wird vom Lehrstuhl für Förder- und Lagerwesen der TU Dortmund eine Mikrocontrollerschaltung entwickelt, mit deren Hilfe die Nachbarschaftsbeschreibungen innerhalb der Förderanlage automatisch erkannt werden können. Eine fertige Software zur Ansteuerung dieser Platine existiert derzeit noch nicht, weshalb ein Teil der Software im Rahmen der Projektgruppe selbst entwickelt wird (siehe auch 7.8).

In der Komponente „FiLeCC“ sind alle diejenigen Anpassungen und zusätzlichen Programme zusammengefasst, die für das reibungslose Zusammenspiel der ausgewählten Technologien nötig sind. Der Entwurf dieser Programmteile wird in Kapitel 7 beschrieben.

Die in Abbildung 6.1 mit *Applications* bezeichnete Komponente steht stellvertretend für alle Programme, die von außen auf Webservices zugreifen wollen und im Sinne einer SOA die Rolle des Dienstnutzers einnehmen. Sie unterliegen keiner Kontrolle durch die Plattform und müssen sich lediglich an die eingesetzten Kommunikationsverfahren halten.

Im letzten Abschnitt 6.7 wird ein zusammenfassender Überblick über alle ausgewählten Technologien gegeben.

6.1 Hardwareressourcen

Wie im Abschnitt 4.3.1 beschrieben ist, werden zwei Typen von Hardwareressourcen für den Projekteinsatz berücksichtigt: die Industrie PCs und die kompakten Miniaturrechner mit integrierten ARM-Prozessoren. Im Folgenden werden verschiedene Hintergrundinformationen über die verfügbaren Lösungen beschrieben.

Industrie-PCs

Ein Industrie-PC (IPC) ist ein Rechner, der für den Einsatz in der Industrie speziell konzipiert ist. Diese PC-kompatiblen Geräte werden zunehmend in industriellen Anlagen benutzt und ersetzen dabei die spezielle Hardware-SPS, welche früher zwingend notwendig war [Gee01]. Ein solcher Rechner kann in einer rauen Industrieumgebung betrieben werden und muss daher vor störende Umwelteinflüssen geschützt werden. Insbesondere muss er mit hohen Temperaturen, Vibrationen und Staub umgehen können [Sie10, Mer10, IPC10]. Deswegen sind die IPCs erheblich robuster gebaut und werden unter schweren Arbeitsbedingungen getestet.

Es existieren verschiedene Standards, die die Robustheit und die Industrieeignung der IPCs beschreiben. Eine mögliche Kennzahl ist die sogenannte Schutzart (engl. *International Protection Rating*, auch *IP Code*) [Bis06]. Sie besteht aus der Zeichenfolge „IP“ und zwei nachfolgenden Ziffern, welche die Berührungs-, Fremdkörper- und Wasserschutzart des getesteten Geräts angeben. Weiterhin, wird die erlaubte Betriebstemperatur angegeben, die für den industriellen Einsatz zwischen 0 °C und 60 °C liegt. Erschütterungstests werden durchgeführt, um sicherzustellen, dass die spezifizierten mechanischen Störungen keinen schädlichen Effekt auf die technischen Komponenten haben. Außerdem werden die Geräte auf ihre elektromagnetische Verträglichkeit (EMV) geprüft und zertifiziert.

Um diese Bedingungen auszustehen, werden verschiedene Maßnahmen bei dem Aufbau der IPCs ergriffen. Die Gehäuse und die Hauptplatinen der Rechner werden stabil aufgebaut, wobei Stahlrahmen eingesetzt werden. Wegen der Vibrationen wird auf konventionellen Festplatten mit beweglichen Teilen verzichtet. Stattdessen werden *Solid State Disks* (SSDs) oder Flash-Speicher verwendet, welche über keine beweglichen Teilen verfügen. Falls Festplatten wegen eines großen Speicherplatzbedarfs zu verwenden sind, sind diese auf speziellen Festplattenaufhängungen montiert. Um die Zuverlässigkeit zu verbessern, werden redundante Stromversorgungen eingebaut. Zusätzlich wird *Error-correcting code* (ECC) Arbeitsspeicher integriert, der das Risiko von fehlerhaften Berechnungen minimiert. Weiterhin ist ein Watchdog-Timer vorhanden, der dafür sorgt, das Gerät rückzusetzen, wenn das laufende Programm nicht mehr funktionsfähig ist. Eine passive Kühlung wird verwendet um bewegliche Komponenten zu vermeiden und das Risiko von Überhitzung durch Ausfall dieser aktiven Komponente zu minimieren.

IPCs haben verschiedene Anwendungsbereiche. Hauptsächlich werden sie als Ausführungsumgebung für Soft-SPS verwenden. Auf diese Weise können sie andere Geräte ansteuern. Dies kann sowohl direkt, als auch über einen Feldbus erfolgen [Ler05]. Sie können die Prozessvisualisierung durchführen und damit den Echtzeitzustand des Betriebsprozesses zugänglich machen. Zusätzlich kann ein IPC Protokollfunktionen erbringen, indem er die wesentlichen Parameter eines Prozesses beobachtet und speichert. Diese Funktion ist z. B. für die Problembehebung von wichtiger Bedeutung. Die Analyse von gesammelten Daten kann auch eine Grundlage zur Optimierung des Betriebsprozesses sein.

(a) WAGO-I/O-IPC-G2 (b) WAGO-I/O-IPC-P14

Abbildung 6.2: WAGO IPCs [WAG11a, WAG11b]

Es sind verschiedene Bautypen von IPCs verfügbar. Ein Typ ist der *Rack-mounted* IPC, der in einem 19″-Einbauschacht montiert wird. Diese Art der Montage wird benutzt, wenn eine zentrale Topologie vorliegt und alle Hardwaregeräte in einem Schaltschrank zu integrieren sind. Weiterhin können IPCs auch als sogenannte Panel-PCs mit eingebetteten Touch-Display realisiert werden. Aufgrund des integrierten Displays sind diese für Visualisierungs- und Steurungsganwendungen gut geeignet. Darüber hinaus existieren Embedded- oder auch Kompakt-IPCs, welche die Rechnerressourcen in einer kompakten Bauform bereit stellen. Die Vorteile PCs dieser Art bestehen darin, dass sie platzsparend sind und für verteilte Steuerung verwendet werden können, indem sie auf einer Tragschiene montiert werden.

Außer ihrer Robustheit, haben die IPCs die Fähigkeit, Peripheriegeräte direkt zu steuern. Dies erfolgt durch Steckkarten, die digitale und analoge Ein- und Ausgänge realisieren. Ein auf dem Rechner installiertes Betriebssystem kann diese Ein- und Ausgänge beobachten, erfassen und kontrollieren. Im Zusammenspiel mit einer Laufzeitumgebung für SPS-Programme kann ein IPC als eine Plattform für Soft-SPS betrachtet werden [tS05, Ler05].

Deutsche Hersteller von IPCs und dazugehörigen Komponenten sind unter anderem die Siemens AG (`http://siemens.net`), die WAGO Kontakttechnik GmbH (`http://wago.net`) und die Beckhoff Automation GmbH (`http://www.beckhoff.de`). Die angebotenen Lösungen dieser Firmen ähneln einander in der Regel stark. Da IPCs von WAGO bereits bei dem Lehrstuhl für Förder- und Lagerwesen verfügbar sind, konzentriert sich der Text auf die WAGO Produkte.

WAGO IPCs Die WAGO Kontakttechnik GmbH ist ein deutsches Unternehmen, welches Komponente für die Industrie, die Prozesstechnik und die Gebäudeautomation herstellt. Insbesondere bietet WAGO komplette Lösungen für die Industrieautomatisierung, wie z. B. Klemmen, Kabel, IPCs oder Feldbuskoppler an. Das IPC-Angebot besteht aus Linux-basierten Embedded-PCs. In diesem Endbericht werden zwei Modelle beschrieben: das veraltete Modell „WAGO-I/O-IPC-G2“ [WAG11a] und das neuere leistungsfähigere Modell „WAGO-I/O-IPC-P14"[WAG11b]. Die beiden Modelle sind in Abbildung 6.2 zu sehen.

Diese IPCs sind Vertreter des Embedded-Bautyps und werden an einer Tragschiene montiert. Beide unterstützen die feldbusunabhängigen E/A-Steckkarten von WAGO, die den Namen „WAGO-I/O-SYSTEM“ tragen [WAG07]. Diese Steckkarten ermöglichen den direkten Anschluss von beliebigen Sensoren oder Aktoren und lassen sich einfach an die IPCs anschließen. Zusätzlich

verfügen die Rechner über eine serielle RS-232 Schnittstelle, zwei USB-Ports und Status-LEDs. Zwei integrierte digitale Ausgänge und zwei digitale Eingänge sind vorhanden und erlauben den direkten Anschluss von Mess- und Steuergeräten. Es ist ebenfalls eine DVI-Schnittstelle präsent, die zum direkten Anschluss eines Monitors dient. Die Kommunikationsfähigkeiten werden durch zwei *Fast Ethernet*-Schnittstellen vervollständigt. Diese Netzwerkanschlüsse ermöglichen die Verwendung verschiedener Netzwerkprotokolle wie z. B. Modbus/TCP. Als Datenspeicher benutzten die WAGO IPCs eine CompactFlash-Karte mit einer maximalen Größe von 4 GB. Die Geräte verfügen weiterhin über einen integrierten Permanentspeicher, der die Konfiguration des Gerätes speichert und sie im Falle eines Stromausfalls erhält. Beide Modelle sind für die IP20-Schutzart spezifiziert. Damit sind sie vor Fremdkörpern geschützt, bleiben aber anfällig für Flüssigkeiten. Die Geräte sind passiv gekühlt und operieren damit lüfterlos. Sie dürfen unter Temperaturen von 0 °C bis zu 55 °C betrieben werden.

Der größte Unterschied zwischen den beiden Modellen besteht im Prozessortyp und dessen Geschwindigkeit. Der IPC der ersten Generation verfügt über einen AMD Geode SC1200 Prozessor, der eine Taktfrequenz von 266 MHz besitzt. Der neue IPC ist mit einem 1,4 GHz Intel Pentium M ausgestattet. Das erste Modell ist kompakter (172 mm × 65 mm × 100 mm), wobei das Neuere die Dimensionen 240 mm × 70 mm × 100 mm hat. Das Pentium-basierte Modell besitzt 256 MB Arbeitsspeicher und 512 MB internen Flash-Speicher und das andere – 128 MB Arbeits- und 128 MB Flashspeicher.

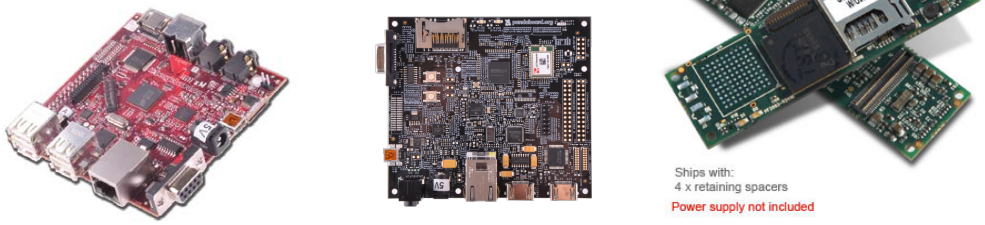
Beide IPCs basieren auf einem Linux-Betriebssystem mit Echtzeiterweiterungen. Diese Erweiterungen ermöglichen die Realisierung von Steuerungs- und anderen echtzeitkritischen Aufgaben. Das Programmieren kann in Hochsprachen wie C/C++ oder Java erfolgen. Zusätzlich ist eine CoDeSys-Umgebung vorhanden, die es ermöglicht, die IPCs in SPS-Programmiersprachen nach IEC 61131-3 zu programmieren [Sma12]. Weitergehende Informationen über diese Sprachen sind in [Ler05] vorhanden.

Zusammenfassend kann festgehalten werden, dass es sich bei den IPCs vom WAGO grundsätzlich um PC-kompatible Rechner handelt, die in einem robusten Gehäuse verbaut sind. Eine weitere Form von Hardware – die verschiedenen ARM-Architektur-basierte Miniaturrechner – wird im nächsten Abschnitt erläutert.

Miniaturrechner

Die zuvor beschriebene IPCs sind nicht für alle Aufgaben geeignet. Beispielsweise sind selbst die kompakten IPCs relativ groß und eignen sich nicht zur Integration in kleine Geräte. Außerdem sind die Anschaffungskosten für einen IPC vergleichsweise hoch, insbesondere wenn die bereitgestellte Leistung nicht benötigt wird. Deswegen werden Alternativen wie kleinere, günstigere Miniaturrechner betrachtet, die besser den Preis- und Größenanforderungen entsprechen. Diese Module sind in einem autonomen Modul (z. B. SNM) leicht integrierbar.

Bei diesen Miniaturrechner gibt es verschiedene Formfaktoren [HDR07]. Eine Bauart ist der Einplatinenrechner (*Single-board Computer*, SBC). Diese Bauart integriert alle wichtigen Teile eines Rechners auf einer einzigen Platine, wie z. B. den Prozessor, den Arbeitsspeicher und den Massenspeicher. Die konventionellen Anschlüsse, wie USB oder Ethernet, sowie VGA- oder DVI-Ausgänge sind ebenfalls auf dieser Platine untergebracht.



(a) BeagleBoard [Bea10] (b) PandaBoard [Pan11] (c) Overo Earth [Gum11b]

Abbildung 6.3: ARM-basierte Hardwareboards

Eine andere Bauart heißt *Computer-on-module* (COM) oder auch *System-on-module* (SOM). In diesem Fall ist ebenfalls ein funktionsfähiger Rechner auf einer Platine vorhanden. Die Anschlüsse sind als Kontaktflächen oder als proprietäre Erweiterungsbuchsen vorhanden. Um die Platine nutzbar zu machen, sind zusätzliche Erweiterungsmodule notwendig. Es gibt Erweiterungsmodule, die vom Hersteller angeboten werden und bestimmte Signale nach außen führen. Diese Bauart hat den Vorteil, dass die Basismodule vergleichsweise klein sind und die Erweiterungsmodule passend zum Einsatz ausgewählt werden können.

Im Folgenden werden beide Arten von Miniaturrechner anhand von Beispielen kurz beschrieben.

BeagleBoard Das BeagleBoard [Bea10] ist ein Einplatinenrechner, welcher auf der *Open Multimedia Application Platform* (OMAP) [Tex12] von Texas Instruments basiert. Diese Plattform kombiniert einen Prozessor und ausgewählte Peripheriegeräte wie USB- und Kamerakontroller auf einem Chip. Der Entwurf des BeagleBoards stammt von Texas Instruments und dient als Einsatzbeispiel der OMAP-Plattform. Das Boarddesign ist frei zugänglich und wird in Zusammenarbeit mit den Benutzern weiterentwickelt. Die aktuelle Hardwarerevision heißt „xM“ und umfasst Verbesserungen gegenüber ursprünglicher Version wie höhere Taktfrequenzen und mehr Arbeitsspeicher. Die folgende Beschreibung basiert auf dieser letzten Revision. Diese ist in Abbildung 6.3(a) dargestellt.

Die Platine ist 78,74 mm × 76,2 mm groß [Bea10]. Sie verfügt über einen DM3730 *Application Processor*. Dieser Prozessor enthält einen ARM-Cortex-A8-Kern und ist eine Weiterentwicklung der OMAP3530-Plattform [Tex11]. Im Wesentlichen bietet dieses Prozessormodul eine 1 GHz-CPU, einen *Digital Signal Processor* und eine 3D-Grafikkarte an. Das BeagleBoard verfügt über 512 MB Arbeitsspeicher und hat einen microSD-Karten-Steckplatz, der microSD-Karten bis zu 4 GB Größe unterstützt. Es stehen vier USB Ports zur Verfügung. Zusätzlich kann das BeagleBoard über Fast-Ethernet kommunizieren. Stereoaudioeingang und -ausgang sind vorhanden, sowie ein DVI-D und ein S-Video-Videoausgang. Das Board integriert einen LCD-Connector, der direkt mit kompatiblen LCD-Modulen verbunden werden kann. Auf dem Board befindet sich zusätzlich eine serielle RS-232-Schnittstelle. Weitere Signale werden an eine 28-polige Erweiterungsbuchse weitergeleitet. Das Board kann über eine vorhandene Schnittstelle angesprochen und ausgetestet werden.

Das BeagleBoard wird ausschließlich als eine Teststellung angeboten und wird für den Einsatz in kommerziellen Produkten nicht empfohlen [Bea10]. Das Board unterstützt eingebettete Betriebssysteme wie Windows CE oder Linux.

PandaBoard Das PandaBoard (siehe Abbildung 6.3(b)) ist der Nachfolger des BeagleBoards. Es ist ebenfalls eine offene Plattform, welche aber auf dem neueren OMAP4-Prozessor basiert. Die Platine verfügt über einen 2-Kern Cortex-A9-Prozessor, der eine um 150% verbesserte Leistung gegenüber dem Cortex-A8 anbietet [Pan11]. Weiterhin gibt es Verbesserungen bei der Videoausgabe, wobei in diesem Modell „Full HD“-Videowiedergabe unterstützt wird. Der 3D-Beschleuniger ist leistungsfähiger und die Größe des Arbeitsspeichers beträgt 1 GB. Eine weitere wichtige Neuerung ist die Verfügbarkeit von Bluetooth- und WLAN-Adaptern [Pan11].

Gumstix Die Gumstix-Produkte [Gum11b] sind Computer-on-modules, welche eine Größe von 58 mm × 17 mm haben. Die Overo-Produktserie besteht aus verschiedenen Modulkonfigurationen, welche unterschiedliche Hardwareausstattungen anbieten. Alle Produkte basieren auf der OMAP3-Plattform und beinhalten einen ARM Cortex-A8-Prozessorkern. Laut Herstellerangabe, unterstützen einige Modelle 3D-Beschleunigung- und DSP-Funktionalität sowie Bluetooth- und WLAN-Konnektivität. Die Gumstix-Produkte sind ausdrücklich für den Einsatz in neuen Produkten geeignet, im Gegensatz zu dem Entwurfsstatus der BeagleBoard-Serie. Ein Beispielmodel ist das Overo Earth COM (siehe Abbildung 6.3(c)). Es beinhaltet einen OMAP3530-Prozessor und verfügt über 256 MB Arbeitsspeicher, sowie 256 MB Flash-Speicher [Gum11c]. Das COM akzeptiert eine Erweiterungskarte, welche über zwei 70-polige Buchsen angeschlossen wird. Zusätzlich ist eine 27-polige Buchse für den Anschluss eines Kameramoduls vorgesehen. Das Gerät kann bei Temperaturen von 0 °C bis 85 °C betrieben werden.

Der Prozessor basiert auf der OMAP3-Plattform und weist Ähnlichkeiten zum BeagleBoard auf. Weil die Basismodule von Gumstix COMs sind, brauchen sie zusätzliche Erweiterungsmodule, um verschiedene Funktionen nutzbar zu machen. Der Hersteller bietet unterschiedliche vorkonfigurierte Erweiterungsmodule, die verschiedene integrierte Funktionen haben, wie z. B. die Erweiterungsmodule „Tobi“ [Gum11d] und „Chestnut43“ [Gum11a], welche das Basismodul unter anderem um einen Ethernet-Anschluss ergänzen. Das Erweiterungsmodul Tobi verfügt darüber hinaus über USB- und DVI-Anschlüsse, sowie eine Schnittstelle für Stereo-Audiosignale. Die Größe des Moduls beträgt 105 mm × 40 mm. Das Chestnut43 Modul bietet einen Anschluss für einen 4, 3″ Touch-Screen, sowie USB- und Audioanschlüsse. Beide Boards stellen 40 E/A-Kanäle zur Verfügung, die zur Kommunikation und Steuerung anderer Geräte dienen können.

Die Gumstix-Produkte unterstützen Embedded-Linux als Standardbetriebsystem. Es besteht die Möglichkeit, Windows zu verwenden. Dies wird herstellerseitig jedoch nicht offiziell unterstützt.

6.2 Laufzeitumgebungen

Neben den Anforderungen an die Hardware entstehen durch die Cloud-Plattform gewisse Voraussetzungen im Bereich Software. Die Softwaregrundlage bildet wie bei jedem Computersystem das Betriebssystem. Die detaillierten Anforderungen an das Betriebssystem sind im Pflichtenheft (siehe Anhang A) dokumentiert und in den folgenden Abschnitten aufgegriffen. Im ersten

Unterabschnitt 6.2.1 werden ausgewählte Betriebssysteme vorgestellt, die die Anforderungen aus dem Pflichtenheft erfüllen, und erläutert, nach welchen zusätzlichen Kriterien das Betriebssystem ausgewählt wird. Die von der Projektgruppe erstellte Software ist in Java programmiert, weshalb das Betriebssystem die Ausführung der Java-Laufzeitumgebung unterstützen muss. Im zweiten Unterabschnitt 6.2.2 wird diese Entscheidung begründet und erläutert, welche konkrete Java-Implementierung sich für die Ausführung der Cloud-Plattform eignet.

6.2.1 Betriebssysteme

Zwei der Hauptkriterien für die Auswahl eines Betriebssystems sind eine hohe Flexibilität und eine frei verfügbare Dokumentation. Außerdem ist eine minimale Hardwarebelastung durch das Betriebssystem ein wichtiges Kriterium, da möglichst viele Ressourcen für die eigentliche Plattform und damit zum Ausführung von Programmen zur Verfügung stehen soll. Da bereits der Steuerungsteil der Versuchsanlage *CoDeA* auf einer Linux-Plattform betrieben wird und Linux alle genannten Anforderungen erfüllen kann, wird als Betriebssystemkern Linux verwendet. Das Microsoft Betriebssystem Windows kann nicht eingesetzt werden, da wichtige Hardwareplattformen wie z. B. die gesamte ARM-Prozessorarchitektur nicht unterstützt werden. Außerdem ist der Speicherbedarf zu hoch. So brauchte eine Installation von Windows 7 ohne zusätzliche Software in Tests etwa 10 GB nichtflüchtigen Speicher. Das von Apple entwickelte Mac OS X kann nur auf der von Apple vertriebenen Hardware eingesetzt werden und stellt deshalb keine Alternative dar. Andere Unix-Betriebssysteme haben oft einen geringen Verbreitungsgrad und bieten unter anderem deshalb weniger Treiber als Linux oder Windows an. Trotzdem wird bei der Entwicklung der Plattform darauf geachtet, dass alle Komponenten betriebssystemunabhängig gestaltet sind und eine Portierung auf andere Betriebssysteme grundsätzlich möglich ist.

Neben dem Kern werden in einem Linux-System noch weitere Programme und Bibliotheken benötigt. Dazu gehört beispielsweise die *GNU C Library* (glibc), die eine Schnittstelle für Systemaufrufe bereitstellt, über die Programme Betriebssystemaufgaben wie das Lesen und Schreiben von Dateien nutzen können. Ein weiteres Beispiel ist der *init*-Prozess, der als erster Prozess beim Booten eines Linux-Systems gestartet wird. Er ist für den Start aller weiteren Prozesse zuständig, die zum Betrieb des System nötig sind. Welche dies sind, kann der Systemadministrator in der Regel konfigurieren. In den meisten Fällen dürfte z. B. ein Netzwerkverwaltungsdienst darunter sein. Die Zusammenstellung von Kernel, Systembibliotheken und -programmen übernehmen Linux-Distributionen. Für die Plattform soll eine Distribution ausgewählt werden, die in den Punkten Paketvielfalt, Aktualität von Software, Anpassungs- und Wartungsaufwand in Bezug auf unterstützte Hardwareplattformen und Speicherplatzbedarf sowohl flüchtiger als auch nichtflüchtiger Art die besten Ergebnisse erzielt. Die in den folgenden Unterabschnitten vorgestellten Distributionen konzentrieren sich diejenigen, die das Kriterium der Ressourcenbeschränkung erfüllen. Eine Übersichtstabelle und die Auswahl sind im letzten Unterabschnitt beschrieben.

Debian Squeeze

Die aktuelle Version von Debian ist 6.0 (Squeeze), das seit dem 6. Februar 2011 verfügbar ist und mit 29.000 Paketen eine große Auswahl an zusätzlicher Software bietet. Zudem unterstützt die Distribution mit ARM, MIPS, PowerPC und x86 alle gängigen PC-Architekturen [Deb11]. Eine aktuelle Basisinstallation von Debian Squeeze belegt etwa 300 MB, bietet dafür den vollen Komfort der Linux Standardprogramme/-bibliotheken. Das Paketsystem ermöglicht eine einfache

Verwaltung von Softwareinstallationen und Programmaktualisierungen, die durch eine große Entwicklergemeinde zeitnah bereitgestellt werden. Allerdings ist diese Distribution nicht direkt auf den Einsatz in ressourcenbeschränkten Umgebungen optimiert, was besonders im Rahmen des EmDebian-Projekts umgesetzt wird.

EmDebian

Im Wiki des EmDebian Projekts wird dessen Aufgabe so zusammengefasst: „In short, what EmDebian does is wrap around the regular debian package building tools to provide a more fine grained control over package selection, size, dependencies and content to enable creation of very small and efficient debian packages for use on naturally resource limited embedded targets.“ ([Wil11]). EmDebian unterscheidet zwischen drei verschiedenen Varianten: Grip, Crush und Baked.

Bei EmDebian Grip handelt es sich um eine zu Debian Squeeze binärkompatible Distribution, die die Standard-Debianpakete verwendet, diese aber verschlankt. Binärkompatibel heißt in diesem Fall, dass ein für Debian Squeeze kompiliertes Programm auch unter EmDebian Grip ausgeführt werden kann. Dies geschieht z. B. dadurch, dass Dokumentationen und Sprachpakete separat verfügbar sind. Dazu werden die Installationspakete (.deb Dateien) von Debian Squeeze durch automatisierte Tools entpackt, die unnötigen Dateien gelöscht bzw. verschoben und das Paket neu verpackt. Dadurch ändern sich die Paketnamen im Vergleich zu Debian Squeeze nicht [Emb11c]. Dies ermöglicht einem mit Debian Squeeze vertrauten Anwender einen einfachen Umstieg auf EmDebian Grip. Außerdem werden die Abhängigkeiten unter den Paketen verändert, so dass insgesamt weniger Pakete installiert werden müssen. Eine Testinstallation ist unter 100 MB groß, das Projekt selbst geht von einer Platzersparnis bis zu 70% aus. Trotzdem setzt Grip auf die Standardprogramme von Linux, d. h. die Programmiersprache Perl, die Systembibliothek glibc und die Werkzeugsammlung Coreutils werden als Basis verwendet. Dadurch verhält sich Grip genau wie Debian Squeeze. Die aktuelle Version von Grip enthält einen großen Teil der Pakete aus Squeeze. Als Architekturen stehen ARM, MIPS, PowerPC und x86 zur Verfügung [Emb11b].

Einen anderen Ansatz verfolgt Debian Crush. Es setzt im Gegensatz zu Grip nicht auf der *Standard-Linux-Toolchain* auf, sondern verwendet Busybox als Ersatz für die Coreutils und Perl wird nicht genutzt. Dadurch ist keine Binärkompatibilität mit den Paketen aus Debian Squeeze gegeben, d. h. alle Pakete werden neu kompiliert. Deshalb werden wenige Basispakete fertig zur Verfügung gestellt, die Portierung der restlichen Pakete bleibt dem Anwender überlassen. Die Einsparungsmöglichkeiten beim Speicherbedarf sind dadurch allerdings noch größer als bei Grip. Die aktuellste Version basiert auf Debian 5.0 (Lenny) und ist nur für die ARM-Plattform verfügbar [Emb11a]. Die Entwicklung neuerer Versionen, die auf Squeeze basieren, sind vorerst aus Personalmangel gestoppt worden [Wil09].

EmDebian Baked bezeichnet eine vorkonfigurierte Installation von Grip oder Crush, die auf dem Zielsystem nicht mehr verändert werden kann. Damit ist ein Paketsystem überflüssig und i. d. R. nicht mehr enthalten. Updates können nur durchgeführt werden, indem das gesamte System neu aufgespielt wird [Emb11a]. Dies spart nochmals Speicherplatz und ist dann sinnvoll, wenn die Zielsysteme eine homogene Hardwarelandschaft bilden und die Anpassungen an individuelle Hardware zentral vorgenommen werden kann. Für die geplante Cloud-Umgebung eignet sich der Einsatz nicht, da eine flexible Erweiterung mit neuen Rechen- und Speicherressourcen

auf jeden Fall möglich sein muss. Über einen längeren Zeitraum ist dies mit der gleichen Hardwarekonfiguration aus wirtschaftlichen Gründen nicht sinnvoll, weshalb zwangsläufig eine heterogene Systemlandschaft entsteht. Deshalb wird EmDebian Baked im Weiteren nicht mehr betrachtet.

Damn Small Linux (DSL)

Eine weitere Distribution, die ebenfalls auf Debian basiert, ist *Damn Small Linux* (DSL). Die Entwickler garantieren bei Damn Small Linux, dass die LiveCD niemals mehr als 50 MB groß ist und trotzdem ein vollwertiges System mit grafischer Oberfläche enthält. Die Hardwareanforderungen sind gering: mit 24 MB RAM lässt sich das System booten, ab 128 MB wird es komplett in den Hauptspeicher geladen und von dort ausgeführt [Dam12]. Allerdings hat das selbstgesetzte Limit von 50 MB für das komplette Betriebssystemabbild den Nachteil, dass Damn Small Linux immernoch auf den alten Linux Kernel 2.4 (aktuell ist 3.0) setzt und damit neuere Hardware nicht unterstützt. Als Nachfolgeprojekt existiert deshalb DSL-N, dass auf den Kernel 2.6 aufsetzt und keinerlei Größenbeschränkungen unterliegt. Allerdings wird diese Version derzeit nicht mehr aktiv weiterentwickelt. Die letzte Version von Damn Small Linux wurde 2008 veröffentlicht. Trotzdem gibt es sehr viele Zusatzpakete, die die gängigste Software zum Nachinstallieren liefern [Dam10]. Insgesamt kann Damn Small Linux vor allem auf Grund des Linux 2.4 Kernel und der stagnierenden Entwicklung als unterstütztes Betriebssystem ausgeschlossen werden.

SliTaz

Eine ähnliche Zielgruppe wie Damn Small Linux hat die Distribution SliTaz. Sie stellt ein großes Paketrepository (über 2000 Pakete, darunter auch eine aktuelle Java-Version) bereit und benötigt lediglich 100 MB Festplattenspeicher. Die aktuelle Version ist am 28.03.2010, die letzte Aktualisierung der instabilen Version am 4.11.2010 veröffentlicht worden. Zudem enthält die Distribution aktuelle Softwarepakete, darunter auch der Linux Kernel 2.6 [Sli12]. Durch das Entfernen der grafischen Oberfläche wird der Speicherplatzverbrauch auf 50 MB reduziert. Damit hat SliTaz eine ähnliche Größe wie *Damn Small Linux*, bietet im Gegensatz dazu aber aktuelle Softwarepakete.

Zusammenfassung und Fazit

Tabelle 6.1: Merkmalanalyse der vorgestellten Betriebssysteme

Name	Paketvielfalt	Aktualität	Anpassungsaufwand	Speicherbedarf
Debian (Squeeze)	++	++	++	-
EmDebian Grip	+	+	++	+
EmDebian Crush	- -	- -	- -	++
Damn Small Linux	+	-	-	++
SliTaz	+	+	+	+

Tabelle 6.1 zeigt eine Merkmalanalyse der vorgestellten Betriebssysteme. Die Spalte Speicherbedarf enthält sowohl die gemessenen Werte für flüchtigen als auch nichtflüchtigen Speicher. Der Anpassungsaufwand bezieht sich auf das Erstellen eigener Softwarepakete und auf die

Installation bzw. Wartung des Systems. Insgesamt ist die Bandbreite an ressourcenschonenden Linux-Distributionen groß. Die Distributionen *EmDebian Crush* und *Damn Small Linux* haben den geringsten Speicherbedarf, allerdings ist der Wartungs- und Anpassungsaufwand an die Zielplattform sehr hoch. Es hat sich außerdem gezeigt, dass Debian Squeeze geringe Anforderungen an flüchtigen und nichtflüchtigen Speicherplatz stellt. Wegen der Linux Standardprogramme und -bibliotheken, der großen Auswahl an Paketen und einfachen Konfigurierbarkeit werden die Testsysteme mit einer Minimalinstallation von Debian betrieben. In einem zweiten Schritt kann dann auf das sehr ähnliche *EmDebian Grip* umgestellt werden, welches weniger nichtflüchtigen Speicher verbraucht.

6.2.2 Java – Programmiersprache und Spezifikationen

Als Programmiersprache wird sowohl für die selbstentwickelte Software als auch für die Software, die auf der entwickelten Plattform ausgeführt werden soll, Java verwendet. Der Hauptgrund dafür ist ein funktionaler Aspekt der Sprache Java: Java-Programme werden in plattformunabhängigen *Bytecode* übersetzt. Dieser Bytecode kann auf allen Systemen ausgeführt werden, auf denen eine *Java Virtual Maschine* (JVM, interpretiert den Bytecode) verfügbar ist [LY99]. Da es sich bei einer Cloud um ein heterogenes System handelt, ist Plattformunabhängigkeit ein zentrales Kriterium. Denn i. d. R. wird der Quellcode der Programme, die in der Cloud laufen sollen, nicht verfügbar sein. Deshalb ist ein nachträgliches Anpassen an eine andere Architektur nur unter hohem Aufwand oder überhaupt nicht möglich. Außerdem wird Java aktiv weiterentwickelt. Am 28.07.2011 wurde die aktuelle Version 7 der Java-Spezifikation veröffentlicht, die kleine Spracherweiterungen im Vergleich zur Version 6 enthält, die in den folgenden Abschnitten betrachtet wird. Java hat zudem eine hohe Verbreitung und dadurch auch eine große Entwicklergemeinde, die viele Bibliotheken und auch Dokumentationen bereitstellen.

Neben der Programmiersprache Java gehören auch die JVM und Klassenbibliotheken zu einer Java-Plattform. Der genaue Funktionsumfang einer Java-Plattform wird in Spezifikationen wie *Java Standard Edition* (SE) und *Java Mobile Edition* (ME) festgelegt. Zur den verschiedenen Spezifikationen gibt es Implementierungen, die sich vor allem in Zielplattform, Geschwindigkeit und Ressourcenverwaltung unterscheiden. In den folgenden Unterabschnitten werden zuerst die unterschiedlichen Java-Spezifikationen erläutert und danach die Vor- und Nachteile der Java SE aufgezeigt. Eine Nutzwertanalyse zum Abschluss zeigt, welche Implementierungen der Java-SE-Spezifikation sich für die zu erstellende Plattform am besten eignet.

Die Java-SE-Plattform

Abbildung 6.4 zeigt eine Übersicht der Java-Plattformen, die von Oracle entwickelt werden. Diese unterscheiden sich vor allem durch die eingesetzte *Virtual Maschine*, aber auch durch die mitgelieferten Bibliotheken, die als Klassen zur Verfügung stehen. Die Abbildung zeigt, dass zur Java-SE-Plattform eine umfangreiche Bibliothek gehört.

Eine Übersicht über alle Funktionen bietet Abbildung 6.5. Alle Funktionen der Java-SE-Plattform sind in der offiziellen Java-SE-Dokumentation unter [Ora11c] beschrieben, von denen die wichtigsten im Folgenden kurz betrachtet werden.

Neben der eigentlichen Programmiersprache Java enthält die Java-SE-Plattform eine große Anzahl von Werkzeugen zum Übersetzen und Verwalten von Java-Programmen, auf die an dieser

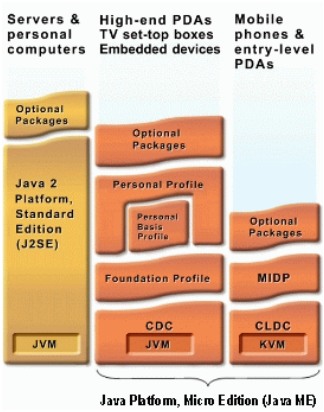


Abbildung 6.4: Unterschiede zwischen den Oracle Java Plattformen. Quelle: [Ora11a]

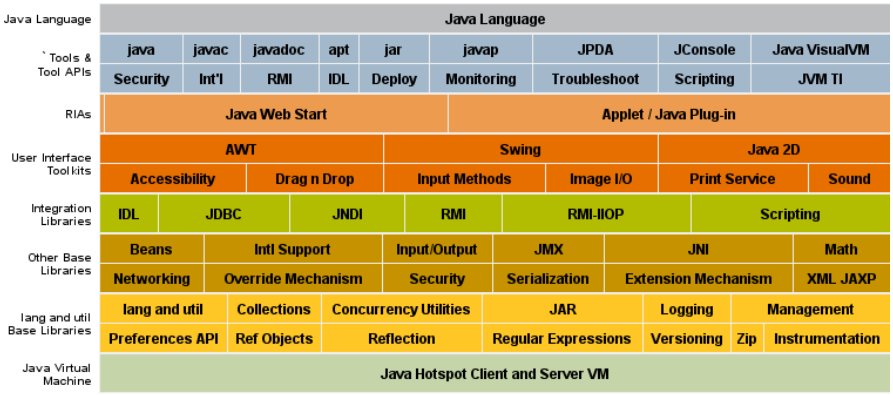


Abbildung 6.5: Plattformübersicht Java Standard Edition 6. Quelle: [Ora11c]

Stelle nicht näher eingegangen wird. Auch *Rich Internet Applications* (RIA) und *User Interface Toolkits* (AWT, Swing, Java2D) werden nicht näher betrachtet. Die von der Java-SE-Plattform mitgelieferten Bibliotheken werden im Folgenden betrachtet, da sie in der Java-ME-Plattform (siehe Abschnitt 6.2.2) nicht enthalten sind.

Die Grundlage der Java-Plattform bilden die Klassen aus dem Namespace `java.lang` und `java.util`. In `java.lang` sind die Basisklassen für Datentypen (`Integer`, `String`, ...), Klassen (`Class`, `ClassLoader`, ...) und Prozesse (`Process`, `Thread`, ...) enthalten. Zudem wird eine Schnittstelle für die Spracheigenschaft *Reflection* bereitgestellt, die es ermöglicht Eigenschaften von Klassen abzufragen und zu manipulieren, wie z. B. die Abfrage von verfügbaren Methoden einer Klasse. Die *Concurrency Utilities* bieten Funktionen für nebenläufige Programme wie z. B. Threads, Semaphore, Mutex, `synchronized` Methoden (automatischer gegenseitiger Ausschluss bei parallelen Methodenzugriffen), atomare Variablen und die Möglichkeit ein eigenes Scheduling zu implementieren.

Im `java.util` Paket sind u. a. die Klassen für *Collections* (Datenstrukturen) enthalten, wie z. B. verkettete Listen, Arrays oder Heaps. Über *Ref Objects* können explizit die Referenzen von Objekten verwaltet werden und erlauben so die Interaktion mit dem *Garbage Collector* über die sogenannten *Reference Queues*. Die *Instrumentation*-Schnittstelle bietet die Möglichkeit externe Agenten (Programme) zum Monitoring und sammeln von Leistungsdaten anzubinden, während

die *Preferences API* eine einheitliche Schnittstelle zur Speicherung von Konfigurationsdaten in unterschiedlichen Datenstrukturen wie z. B. XML-Dateien oder Datenbanken erlaubt.

Unter dem Namen *Other Base Libraries* sind weitere Bibliotheken gruppiert, die einen große Anzahl von Kernfunktionen bieten. Besonders hervorzuheben ist dabei das `java.io` Paket, weil es Klassen zur Dateiverwaltung und Netzwerkanbindung enthält. Auch `java.math` mit Mathematikfunktionen, Klassen zur Bearbeitung von XML-Dateien und Klassen mit Kryptofunktionen (z. B. für verschlüsselte Netzwerkübertragung) sind enthalten. Eine weitere Sprachfunktion ist die Umwandlung von Klassen in einen Strom von Bytes (*Bytestream*). Dieser Vorgang wird *Serialization* genannt. Zudem wird in Java die Internationalisierbarkeit (*i18n*) von Programmen ermöglicht. Zeitzonen- und Spracheinstellungen können vorgenommen werden, an die sich bereits enthaltene Datums- und Zeitklassen halten. Das *Java Native Interface* (JNI) wird benötigt, um nativen ðkompilierten Programmcode ausführen zu können, und bietet oft die einzige Möglichkeit, um auf Hardware direkt zuzugreifen. Die *Java Management Extensions* (JMX) stellt eine erweiterte Schnittstellen zum externen Monitoring (wie z.B. Konfigurationsänderungen, Statistiken, Statusänderungen von Threads) bereit. Die Java Beans sind ein Sprachmittel, das die Entwicklung von Java-Programmen in Komponenten aufbricht und eine Kommunikation durch Ereignisse ermöglicht.

Die *Integration Libraries* ermöglichen die Einbettung von Java-Applikationen in bestehende Infrastrukturen. Darunter fällt z. B. die *Java Interface Definition Language* (Java IDL), die eine *CORBA* (Middleware für verteiltes Rechnen) Anbindung ermöglicht. Über die *Java Database Connectivity* (JDBC) kann außerdem auf relationale Datenbanken über SQL zugegriffen werden. Das *Java Naming and Directory Interface* (JNDI) bietet eine Schnittstelle für Verzeichnis- und Namensdienste wie *LDAP*. Über *Java Remote Method Invocations* (JRMI) wird ein *Remote Procedure Call* Mechanismus implementiert, bei dem über einen zentralen *Registry*-Server Objekte zur Ausführung von entfernten Methodenaufrufen gefunden werden können. Über eine spezielle Schnittstelle für Scriptsprachen werden bekannte interpretierte Sprachen wie z. B. Python (im Jython Projekt) direkt in Java-Bytecode übersetzt und können so über die JVM ausgeführt werden.

Dieser Funktionsumfang hat den Nachteil, dass die Plattform viel Speicherplatz benötigt. Das *Java Runtime Environment* (JRE) der Java SE 6 von Oracle umfasste in einer Testinstallation ca. 100 MB. Im Vergleich zu der vorher betrachteten Betriebssysteminstallation von Debian Squeeze mit 300 MB wird der Umfang der JRE deutlich.

Die JVM selbst ist auf den Betrieb in Mehrprozessorsystemen (Mehrkernsystemen) hin optimiert. Dies umfasst mehrere Programmthreads und einen *Garbage Collector*, der ebenfalls von mehreren Prozessoren Gebrauch machen kann [Ora11d]. Daneben wird eine Technik mit dem Namen *HotSpot* eingesetzt. Der Unterschied zu reinen Java-Bytecode-Interpretern besteht darin, dass bei der Hotspot-Implementierung ein adaptiver *Just-In-Time* (JIT) Compiler eingesetzt wird. Er analysiert zur Laufzeit den Programmcode, erkennt Engpässe (*hot spots*) und übersetzt diese direkt in Maschinencode. Damit braucht dieser Programmteil nicht mehr interpretiert werden, was die Geschwindigkeit der JVM erhöht [Ora11e].

Ingesamt steht mit der Java Standard Edition eine sehr umfangreiche und auf Geschwindigkeit optimierte Java Plattform zur Verfügung.

Die Java-ME-Plattform

Ebenfalls von Oracle wird die *Java Micro Edition* entwickelt. Anders als bei Java SE existieren bei der Java ME zwei Konfigurationen, die eine minimale Auswahl an Bibliotheken mitliefern [Sun03, 2.2.3]. Zusätzliche Funktionalität kann der Entwickler über sogenannte Profile hinzufügen (siehe auch Abbildung 6.4). In den zwei folgenden Abschnitten werden die zwei unter Java ME verfügbaren *Configurations* CLDC und CDC erläutert.

CLDC Die *Java Connected Limited Device Configuration* (CLDC) ist für Zielgeräte mit geringen Hardwareressourcen und Batteriebetrieb entwickelt worden. Typische Einsatzgebiete sind Mobiltelefone, Heimvernetzung, TV-Geräte und Kassensysteme. Die Spezifikation enthält das Konzept, dass während der Laufzeit einzelne Teile des Programms (Klassen) über Netzwerk auf eine sichere Art nachgeladen werden können. Dies liegt u. a. daran, dass Programme (mangels eines Dateisystems) ggf. nicht persistent auf dem Zielgerät gespeichert werden können [Sun03, 2.1].

Die Anforderungen an die Hardware sind bei der CLDC gering: es müssen lediglich 160 KB nichtflüchtiger Speicher (ROM, HDD, etc.) für die VM und Bibliotheken bereitstehen. Zur Laufzeit benötigt die VM mit nur 32 KB flüchtigem Speicher (RAM) für den Heap (dynamischer Speicher). Auch die Softwareanforderungen sind gering: lediglich ein Kernel, der eine Instanz der VM ausführen kann, wird benötigt. Die Bereitstellung eines Dateisystems, einer Prozessverwaltung oder einer Adressraumseparierung sind optional, aber nicht notwendig. Trotzdem bildet die CLDC eine direkte Untermenge der Java SE [Sun03, 2.2]. Teil der Spezifikation sind die Java Sprachedefinition, die VM, Kernbibliotheken aus den Paketen `java.lang`, `java.util` und `java.io`, sowie ein Sicherheits-, Netzwerk- und Internationalisierungskonzept. Explizit nicht Teil der Spezifikation ist die Applikationsverwaltung, graphische Benutzeroberfläche, Ereignis-Behandlung und Interaktion zwischen Benutzer und Programm [Sun03, 2.3].

Sicherheit ist ein zentraler Aspekt der CLDC, insbesondere auf unterer Ebene in der virtuellen Maschine. Das Hauptaugenmerk liegt darauf, dass die Semantik der Sprache Java befolgt wird und dass die VM nicht zum Absturz gebracht oder anderweitig kompromittiert wird. Dies kann entweder durch den regulären Klassenverifizierer zur Laufzeit oder im Rahmen der Präverifizierung durchgeführt werden. Durch die Präverifizierung wird das Verifizieren zur Laufzeit in Hinsicht auf Speicherverbrauch (nur 10 KB Code und weniger als 100 KB RAM) und Ausführungszeit (Methoden werden teilweise `inline` deklariert, d. h. die Methoden werden nicht aufgerufen sondern direkt an die einzelnen Aufrufstellen kopiert) effizienter gemacht. Wird die Präverifizierung nicht unterstützt, dann wird das Standardverfahren verwendet [Sun03, 5.2]. Das sogenannte Sandbox-Modell sieht vor, dass ein Gerät (eine Implementierung), welches die CLDC unterstützt, sicherstellen muss, dass der Classloader nicht umgangen werden kann und nur eine geschlossene und vordefinierte Menge an Bibliotheken auf der Zielplattform vorhanden ist. Dadurch hat der Benutzer keine Möglichkeit eigene Bibliotheken mit nativem Code nachzuinstallieren. Auch die Reihenfolge, in der nach Klassen gesucht wird, muss immer identisch bleiben, damit die mitgelieferten Bibliotheken nicht verändert bzw. übergangen werden können. Außerdem ist es einer Java-Applikation nur erlaubt, Dateien aus der eigenen JAR-Datei zu verwenden. Andere Ebenen der Sicherheit sind nicht Teil der Spezifikation, wie beispielsweise die Sicherheit auf Anwendungsebene, bei der der Zugriff auf Ressourcen nach Vorgaben einschränkt wird, oder die Ende-zu-Ende Sicherheit, bei der die gesamte Funktion auf dem Gerät sicher ist, indem beispielsweise ein verschlüsselter Zugang verfügbar ist [Sun03, 3].

Generell ist die CLDC zur *Java Virtual Maschine Specification* (JVMS) [LY99] kompatibel, mit Außnahme der folgenden Einschränkungen: es ist kein benutzerdefiniertes Laden von Klassen erlaubt (siehe Sandbox-Modell oben), Thread-Gruppen und Hintergrundprozesse, die automatisch mit dem Programm beendet werden, sind nicht verfügbar und es gibt keine Möglichkeit, Objekte zu finalisieren (`Object.finalize()`). Auch bei der Fehlerbehandlung gibt es Unterschiede: Exceptions treten nur synchron auf und die Behandlung von `Error` muss plattformabhängig gelöst werden. Asynchrone Exceptions, die durch externen Zugriff auf einen Thread entstehen können, werden nicht unterstützt [LY99, 11]. Beim Auftreten von `Error` muss der Fehler vom Programm selbst gelöst werden, ansonsten wird er in der Aufrufhierarchie immer weiter hochgereicht. Auf diese Unterschiede muss bei der Erstellung eines Programms geachtet werden. Wird ein Programm für die Java SE programmiert, so ist eine nachträgliche Quelltextrevision in Hinblick auf diese Unterschiede unerlässlich.

Die CLDC sieht es vor, dass Klassen nach Möglichkeit direkt aus der Java SE übernommen werden. Dies ist allerdings auf Grund von Abhängigkeiten vor allem im I/O-Bereich nicht immer möglich, weshalb ein Ersatz durch das Paket `java.microedition.io` bereitgestellt wird. Ansonsten werden die Pakete `java.lang`, `java.util` und `java.io` teilweise übernommen. Insgesamt werden so 39 Basisklassen mitgeliefert, die neben `Object`, `System`, `Thread` und den Typklassen auch die Klassen `Math`, ein Teil der Datumklassen und die `Vector`, `Stack` und `Hashtable` Klassen aus der Java Collection Schnittstelle enthalten. Obwohl virtuelle Maschinen nach der CLDC den Unicode Zeichensatz untersützen müssen, ist die Standardkodierung auf Latin1 festgelegt. Dies führt zu einem geringerem Speicherverbrauch. Außerdem unterstützen die Datums- und Zeitklassen keine Internationalisierung. Für die Netzwerkfunktionen wird eine von der Java SE abweichende Schnittstelle angeboten, die im wesentlichen aus der Klasse `Connector` besteht. An diese kann eine URI (z. B. socket://, comm:0 oder datagram://) übergeben werden. Die eigentliche Unterstützung einzelner Protokolle muss durch Profile eingebunden oder selbst entwickelt werden.

Da viele Funktionen in der CLDC fehlen, wird i. d. R. das Profil *Mobile Information Device Profile* (MIDP) verwendet, dass die fehlende Netzwerkunterstützung (mind. eine Form von HTTP unterstützt), grundlegende Elemente für Benutzerschnittstellen und den Zugriff auf Dateien bereitstellt. MIDP-Applikationen werden auch MIDlets genannt. Trotzdem haben MIDlets Einschränkungen. Weil die JVM keine Unterstützung für das *Java Native Interface* (JNI) mitbringt, sind die einzigen Möglichkeiten nativen Programmcode aufzurufen, direkt Veränderungen an der JVM durchzuführen oder die sogenannte KNI-Schnittstelle zu verwenden, bei der die nativen Quelltextanteile zusammen mit der JVM kompiliert werden müssen [Top02, Seite 12].

Laut [Ora05a] braucht die offizielle CLDC HotSpot VM zusammen mit dem Profil MIDP mindestens 300 KB flüchtigen Speicher (RAM), 1 MB nichtflüchtigen Speicher (ROM oder HDD) und einen 50 MHz Prozessor. Empfohlen wird aber die doppelte Leistung. Die HotSpot VM implementiert wie bei der Java SE einen JIT Compiler, der allerdings durch einen *Ahead-Of-Time* (AOT) Compiler ergänzt wird. Dieser verkürzt zusätzlich die Startzeit der Programme und verlängert damit potentiell die Batterielaufzeit zu Lasten der Programmcodegröße. Seit der Version 1.1.2 wird außerdem Multitasking implementiert, sodass mehrere MIDlets auf einer VM zeitgleich ausgeführt werden können.

Insgesamt ist Java nach der CLDC Spezifikation sehr eingeschränkt und erhält erst mit der MIDP Erweiterung einen größeren Funktionsumfang. In diesem Fall fehlen trotzdem noch komplexere Funktionen, wie z.B. die Finalisierung von Objekten. Zudem werden einige Schnittstellen, vor allem im Bereich der Netzwerkkommunikation, angepasst und unterscheiden sich deutlich von

dem Äquivalent in der Java SE und CDC (siehe unten) [Top02, Seite 179]. Die Stärken der CLDC liegen klar in den niedrigen Hard- und Softwareanforderungen. Zusammenfassend ist sie nur eine Alternative, wenn die Java SE oder CDC auf Grund von zu hohen Anforderungen an die Hardware nicht eingesetzt werden können.

CDC Der zweite Vertreter der Java-ME-Plattform ist die *Connected Device Configuration* (CDC). Ihre Zielgeräte liegen in Bezug auf Leistung zwischen denen, die Java SE einsetzen können, und denen, die CLDC auf Grund von Hardwarebeschränkungen einsetzen müssen. Typischerweise sind diese Systeme mit mehr Arbeitsspeicher ausgestattet, weshalb die CDC mindestens 2 MB Hauptspeicher benötigt [Top02, Seite 5]. Die wichtigsten Profile für CDC sind das *Foundation Profile*, das *Personal Basis Profile* und *Personal Profile*. Das *Foundation Profile* wird i. d. R. für die Entwicklung verwendet, da mit ihm ein Großteil der original Basisklassen aus der Java-SE-Plattform übernommen werden. Dazu gehören die Klassen aus `java.lang`, `java.util`, `java.math`, `java.io`, `java.net`, `java.security` und `java.text` [Top02, Seite 237]. Die beiden *Personal Profiles* enthalten Klassen zur Interaktion mit dem Benutzer über GUI-Elemente. Klassen, für die ein Äquivalent in der Java SE existiert, werden unverändert in die CDC übernommen, abgesehen von als veraltet markierten Schnittstellen [Top02, Seite 235].

Implementierungen Die *phoneME MR3* Implementierung ist die Open-Source-Variante des Java-ME-SDK und ist anders als das SDK unter Linux Betriebssystemen ausführbar [Ora11g]. Sie ist auf den Architekturen ARM, MIPS und x86 lauffähig und umfasst neben der CLDC auch das *Mobile Information Device Profile* (MIDP) und in der Variante `phoneME Advaned MR1` die CDC. Eine offizielle Veröffentlichung gibt es allerdings nicht, lediglich eine Entwicklerversion zum Testen [Ora09].

Ein wesentlicher Unterschied zwischen der Java-SE-Plattform und der CDC findet sich bei der *Virtual Machine*: die Referenzimplementierung von Oracle heißt *C Virtual Maschine* (CVM) und basiert auf dem *phoneME Advanced* Projekt, das allerdings proprietäre Funktionen wie die HotSpot Technologie oder JIT Compilation nicht enthält. Bei der CVM handelt es sich also um einen reinen Java-Bytecode Interpreter [Top02, Seite 229]. Daneben existiert zusätzlich die HotSpot Implementierung der CDC von Oracle. Diese ist nur für die Architekturen ARM und MIPS verfügbar und außerdem ausschließlich unter Linux-Betriebssystemen ausführbar. Beide unterstützen zusätzlich Funktionen der Java SE wie benutzerdefinierte *Class Loader* (bei der CLDC explizit ausgeschlossen), *Serialization*, *Reflection*, JNI und alle I/O-Funktionen der Java SE [Ora05b]. Allerdings basiert die HotSpot-Implementierung auf der Java SE 1.4.2, d. h. Spracheigenschaften neuerer Java-SE-Generationen sind nicht verfügbar [Ora11b].

Java EE

Unter dem Namen *Java Enterprise Edition* (EE) ist eine Erweiterung der Java SE zu finden. Der Schwerpunkt der Spezifikation liegt auf der Unterstützung von komplexen Geschäftsanwendungen. Deshalb ist die Spezifikation fast durchgehend durch Anwendungsserver (engl. *Application Server*) umgesetzt, die eine komplette Laufzeitumgebung mit Prozessverwaltung anbieten. Die zusätzlichen Java-Bibliotheken umfassen die folgenden Funktionen:

Webservices Es werden Schnittstellen für die Verwendung von REST, SOAP, JAX-RPC, JAX-WS, WS-Addressing und Klassen zur allgemeinen Verwaltung von XML-Streams bereitgestellt.

Servlet Ein Containerformat, das sehr stark dem Webservicegedanken entspricht. Servlets können HTTP zur Kommunikation verwenden. Beinhaltet auch Java Server Pages (Format zur Generierung von Servlets).

Enterprise JavaBeans Hierbei wird der Programmcode in kleine ausführbare Teile (Beans) gespalten, die lose gekoppelt werden. Die Identifizierung eines Endpunktes erfolgt wie bei Webservices durch URIs. Durch Annotationen können Eigenschaften wie "stateless" oder "stateful" angegeben werden. Außerdem werden Transaktionen bei der Ausführung von *Enterprise Beans* angeboten, die einen Sitzungsmechanismus implementieren. Diese Transaktionen erfüllen die ACID (*atomicity, consistency, isolation, durability*) Anforderungen. Die *Persistence API* erlaubt die persistente Speicherung von Daten ("stateful") in relationalen Datenbanken über einen *Object-Relational Mapper*.

Java Message Service Ein Nachrichtensystem, bei dem Daten entweder mittels Punkt-zu-Punkt-Verbindung (wie z.B. TCP) oder über ein Abonnentenmodell übertragen werden können. Dieses System kann sowohl zur Kommunikation zwischen einzelnen Programmteilen (Threads, Prozesse) als auch über Rechnergrenzen hinweg zur Kommunikation in Netzwerken eingesetzt werden.

Java EE Connector Architecture (JCA) Abstraktionsschicht für unterschiedliche Datenbanksysteme (noch abstrakter als JDBC aus der Java SE).

JavaServer Faces Ermöglicht es Webinterfaces nach dem MVC-Konzept (*model, view, controller*) aufzubauen. Das MVC-Konzept ist eine Umsetzung des *separation of concerns* Gedanken. In diesem Fall wird die Steuerlogik in *controller*, die Datenverarbeitung in *models* und die Anzeigelogik in *views* getrennt. Außerdem erlaubt *JavaServer Faces* das dynamische Nachladen einzelner Webseitenelemente mittels AJAX (*Asynchronous JavaScript And XML*).

Diese Funktionen können zum Aufbau einer Cloud-Plattformen verwendet werden. Insbesondere die Webservice-Schnittstellen sind für eine lose Kopplung einzelner Komponenten geeignet. Allerdings kann an der Liste abgelesen werden, dass viele Funktionen mehrfach abgedeckt werden. So implementieren die Enterprise Beans eine sehr ähnliche Funktionalität wie Webservices. Dies führt zu einem Mehrbedarf an Speicher- und Rechenressourcen, der noch dadurch erhöht wird, dass die Anwendungsserver, die die Java EE implementieren, selbst noch Systemressourcen vor allem im Bereich des flüchtigen Speichers belegen.

Android

Google bietet mit dem Android-System ebenfalls eine Java-Plattform an. Es handelt sich dabei um mehr als nur eine Java-Plattform, weil in Android auch viele Teile des Betriebssystems enthalten und ein Programmiermodell festgelegt sind, wie Abbildung 6.6 zeigt.

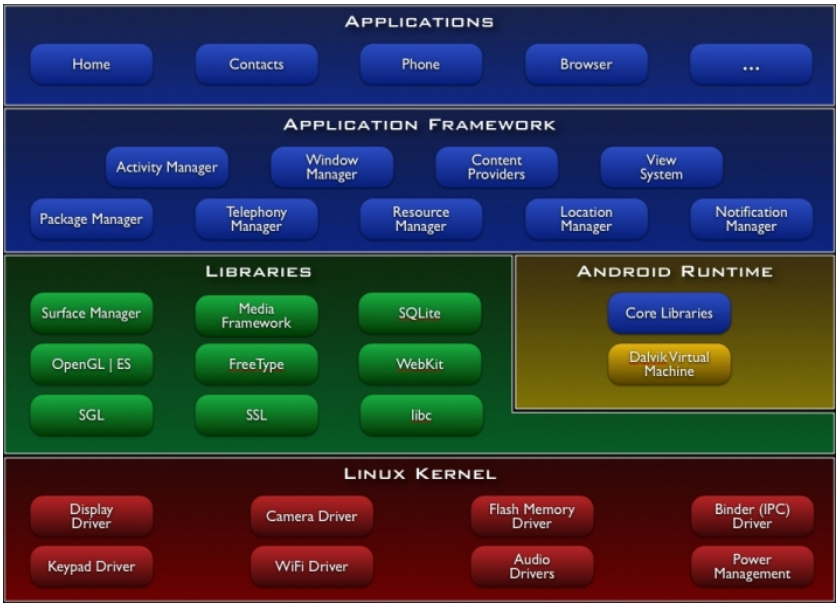


Abbildung 6.6: Aufbau der Android Plattform. Quelle: [Goo11b]

Ähnlich wie bei den Linux Distributionen EmDebian oder DSL enthält Android den Linux-Kernel und nur die grundlegenden Bibliotheken, um auf Hardwarefunktionen bzw. den Kernel zugreifen zu können. Dieser Teil des Systems besteht im wesentlichen aus C/C++ Applikationen, die weitgehend auf bereits existierenden Projekten wie z. B. die Webengine WebKit, die SQL-Datenbank SQLite oder der FreeType Font-Engine basieren [Goo07]. Der Linux-Kernel enthält verschiedene Anpassungen von Google, die z. B. ein anderes Speichermanagement implementieren. Im Vergleich zum von Linus Torvalds herausgegeben Kernel sind über die Hälfte der Änderungen Gerätetreiber [KH10].

Zwei weitere Komponenten des Android System sind die Android Laufzeitumgebung und der Anwendungsrahmen. Kernstück der Android Laufzeitumgebung ist die virtuelle Maschine *Dalvik*, die in der Lage ist, speziellen Java-Bytecode auszuführen. Diese VM wurde von Google auf Basis der virtuellen Maschine des *Apache Harmony* Projekts für die Android Plattform weiterentwickelt. Die Anforderungen für die Dalvik-VM ähneln den VMs der Java-ME-Plattform: sie kann auf System mit langsamen Prozessor (250 – 500 MHz), wenig Hauptspeicher (ab 64 MB) und keinem Auslagerungsspeicher ausgeführt werden und ist in Bezug auf Energieeffizienz optimiert [Bor08].

Insgesamt ist eine Vielzahl von Optimierungen in der Dalvik-VM umgesetzt worden. Dazu zählt ein ressourcenschonenderes Klassenformat, ein neues Bytecodeformat, bei dem viele kleine durch wenige mächtige Befehle ersetzt werden, was zu geringerem Speicherverbrauch führt, geteilter Speicherbereich für Standardbibliotheken, wodurch Klassen einmal geladen und von mehreren Programmen genutzt werden können, und ein effizienterer Just-In-Time Compiler [BP10] [Bor08] [CB10]. Einige dieser Änderungen haben zur Folge, dass die Dalvik VM stark in das umgebende Android-System eingebettet ist. Damit wird dem Entwickler gleichzeitig das bereits erwähnte Programmiermodell vorgeschrieben. Dies ist auf die Bedürfnisse von Programmen mit Benutzerinteraktion optimiert und für Cloud-Anwendungen ungeeignet. So erfodert z. B. das vollständige Deaktivieren der Benutzeroberfläche einen Eingriff in den Android-Quelltext. Zudem gestaltet sich auch das Starten und Stoppen von Programmen schwierig, da dies über den Systemdaemon *Zygote* geschieht.

Google entwickelt Android für ARM-Prozessoren, es gibt allerdings einen von Freiwilligen betreuten x86-Zweig, dessen Betrieb in einer Testinstallation funktioniert. Auch MIPS-Technologies bietet für ihre MIPS-Plattform eine spezielle Android-Version an, die ebenfalls als Open-Source-Software erhältlich ist [MIP11].

Fazit Java-Spezifikationen

Die im letzten Abschnitt vorgestellte Android-Plattform mit ihrer Dalvik VM ist in Hinsicht auf Speicherverbrauch und Geschwindigkeit optimiert und damit für den Einsatz als Cloud-Laufzeitumgebung geeignet. Allerdings werden Entwickler gleichzeitig an ein Programmiermodell gebunden, welches mit seiner Konzentration auf graphische Oberflächen die Entwicklung von Cloud-Anwendungen verkompliziert. Da die Entwicklung von Cloud-Anwendungen möglichst ohne größeren Mehraufwand im Vergleich zu monolithische Anwendung möglich sein soll, ist die Android-Plattform als Java-Umgebung nicht einsetzbar.

Die Java EE Spezifikation kann nicht verwendet werden, weil die verfügbaren Implementierungen einen sehr großen Bedarf an flüchtigem und nichtflüchtigem Speicherplatz haben. Genau in diesem Kriterium ist die Zielplattform allerdings beschränkt.

Gegen die CLDC spricht der verringerte Funktionsumfang. Dadurch müssen Programme, die auf der Cloud-Plattform ausgeführt werden sollen, an diese Einschränkungen angepasst werden. Dies ist insbesondere bei bereits bestehenden Programmen aufwändig.

Die Unterschiede zwischen der CDC und der Java-SE-Plattform sind gering, da die wichtigsten Bibliotheken übernommen werden und die CDC-VM mit HotSpot-Unterstützung alle nötigen Eigenschaften der Java-SE-VM umsetzt und dabei trotzdem einen geringen Hauptspeicherbedarf hat. Liegt allerdings keine ARM- oder MIPS-Architektur vor, muss zu Lasten der Geschwindigkeit die CVM verwendet werden.

Die Java SE hat einen großen Funktionsumfang und ist weit verbreitet, wodurch der Anpassungsaufwand bestehender Programme an die Cloud-Plattform verringert wird. Der Bedarf an Speicher- und Rechenressourcen liegt aus dem gleichen Grund allerdings über denen der CDC und CLDC, hängt aber auch von der eingesetzten Implementierung ab (siehe Abschnitt 6.2.3).

Insgesamt lässt sich festhalten, dass die Arbeit seitens Oracle sowohl für die CDC als auch für die CLDC fast vollständig zu Gunsten von Java Embedded, einer Java-SE-Implementierung, die im folgenden Abschnitt 6.2.3 beschrieben wird, eingestellt worden ist. Bislang ist keine Implementierung der CDC oder CLDC von anderen Anbietern bekannt, weshalb trotz der guten Eignung der CDC die Entscheidung auf die Java SE gefallen ist. Im folgenden Abschnitt werden deshalb die existierenden Implementierungen der Java SE evaluiert.

6.2.3 Evaluierung von Java-SE-Implementierungen

Tabelle 6.2 zeigt eine Übersicht über Java-SE-Implementierungen, die aktive Projekte mit Linux-Unterstützung sind. Kommerzielle Lösungen werden nur berücksichtigt, wenn sie kostenlose Forschungslizenzen vergeben.

Sowohl die OpenJDK-, Oracle-HotSpot- und Oracle-Java-Embedded-Implementierungen werden von der Firma Oracle entwickelt. Durch die Übernahme von Sun Microsystems sind das

Tabelle 6.2: Marktübersicht: Java-SE-Implementierungen

Name	Lizenz	Architekturen	J2SE
OpenJDK	GPLv2	x86, SPARC, ARM	6
Oracle HotSpot	proprietär	x86, SPARC	6
Oracle Java Embedded	proprietär	x86, ARM, PPC	6
Apache Harmony	Apache	x86	1.5
JamaicaVM	proprietär	x86, ARM, MIPS, PPC, SPARC	1.2
JamVM	GPLv2	x86, ARM, MIPS, PPC	JVMS 2
CACAO	GPLv2	x86, Alpha, ARM, MIPS, PPC, S390	JVMS 2
GCJ	GPL	x86, SPARC, Tru64 Unix, Alpha, PPC	1.4

OpenJDK und die HotSpot-Implementierung an Oracle übergegangen und zeichnen sich durch volle Kompatibilität zum Java-Standard aus. Dabei basiert HotSpot auf OpenJDK, allerdings mit zusätzlichen Inhalten, die Oracle nicht unter einer Open-Source-Lizenz veröffentlicht hat [Ora11f]. Als Ablösung der älteren Java-ME-Implementierungen ist Java Embedded entwickelt worden, das ebenfalls die volle Java-SE-Spezifikation mit Ausnahme der graphischen Oberfläche unterstützt („headless“). Die eigentliche JVM ist bei Java Embedded und HotSpot identisch, allerdings werden für Java Embedded speziell angepasste Bibliotheken verwendet. Diese sind neben der x86-Prozessorarchitektur auch für die ARM-Prozessorplattform verfügbar, was das Einsatzgebiet von Java Embedded erhöht.

Apache Harmony ist eine Java-SE-Implementierung, die von der Apache Software Foundation als Open-Source-Projekt verwaltet wird. Vor allem IBM war in den letzten Jahren an der Entwicklung von Harmony beteiligt, die allerdings zu Gunsten der OpenJDK-Unterstützung seitens IBM aufgegeben wurde. Harmony bildet außerdem die Grundlage für die Android Java-Bibliotheken [Apa11a].

Eine Besonderheit bildet die proprietäre JamaicaVM, die Java um Echtzeitfähigkeiten erweitert. Laut Herstellerangaben zeichnet sie sich durch einen geringen Speicherbedarf und eine hohe Geschwindigkeit aus. Außerdem besteht über JNI die Möglichkeit Programmcode in anderen Hochsprachen wie C oder C++ auszuführen. Wie bereits in Kapitel 6.5 beschrieben, wird die Software GridGain zum verteilten Rechnen eingesetzt. Diese ist auf neuere Sprachfunktionen wie Annotationen angewiesen. Da die Klassenbibliotheken der JamaicaVM allerdings nur mit der Java SE 1.2 kompatibel sind, ist das Projekt als Cloud-Laufzeitumgebung nicht einsetzbar [Aic11].

JamVM und CACAO sind reine virtuelle Maschinen, die keine eigenen Klassenbibliotheken enthalten und daher Klassenbibliotheken aus externen Projekten verwenden müssen. Bei beiden können daher entweder die Klassenbibliotheken der OpenJDK oder des GNU Classpath Projekts verwendet werden. GNU Classpath ist eine Implementierung der Java-SE-6-Bibliotheken, bei der die Implementierung einzelner Klassen nicht vollständig ist. Eine Vergleichsmatrix ist unter [Fre10] zu finden. Eine unvollständige Implementierung der Java SE als Cloud-Plattform ist nicht sinnvoll, da Entwickler stabile Schnittstellen für die Erstellung von Programmen benötigen. Deshalb wird für die im Verlauf des Kapitels beschriebenen Tests von JamVM und CACAO lediglich die OpenJDK-Bibliotheken verwendet.

Die Entwickler von JamVM werben mit einer hohen Geschwindigkeit, die z.B. durch die Verwendung eines JIT-Compilers erreicht werden soll. Außerdem ist die VM nur 200 KB groß

und ist Grundlage für die Dalvik VM von Android. Das letzte Version wurde am 01.01.2010 veröffentlicht [Lou11].

CACAO wird von der Universität Wien entwickelt und arbeitet als ein erweiterter JIT-Compiler. Das bedeutet, dass keine Interpretation von Bytecode geschieht, sondern vielmehr bei der ersten Verwendung eines Codeabschnitts der Bytecode in Maschinencode übersetzt wird [CAC11].

Ähnlich arbeitet der GCJ (*GNU Compiler for Java*), der Java-Quellcode vor der Ausführung direkt in Maschinencode übersetzt (*Ahead of Time* Kompilierung). Dadurch besteht der Vorteil von plattformunabhängigem Bytecode nicht mehr. Gerade für Cloud-Anwendungen in einer heterogenen Systemlandschaft ist diese Eigenschaft entscheident: Da nicht davon ausgegangen werden kann, dass auf jedem Knoten exakt die gleiche Software installiert ist (z. B. Prozessorarchitekturbedingt), muss die Software auf jedem Knoten separat kompiliert werden. Besonders deutlich wird dieser Nachteil, wenn ein Update der libc im Linux-System durchgeführt werden muss. Da jedes Programm diese Bibliothek verwendet, müssen bei einem Update alle abhängigen Programme erneut kompliert werden. Im Falle von Java ist das wiederum lediglich die *Java Virtual Maschine*. Zudem stellt die *Virtual Maschine* Sicherheitsfunktionen (z. B. Speicherschutz) zur Laufzeit zur Verfügung, die ein Compiler prinzipbedingt nicht anbieten kann (siehe Abschnitt 6.2.2). Außerdem verwendet der GCJ die Bibliotheken des bereits erwähnten GNU Classpath Projekts. Der Nachteil der unvollständigen Java-SE-Implementierung gilt also auch für den GCJ. Deshalb wurde der GCJ als Referenz getestet, aber nicht in die Auswahl der verwendbaren Java-SE-Implementierungen aufgenommen [Fre11].

Versuchsaufbau

Für einen objektiven Vergleich der verschiedenen Java-Implementierungen werden sechs unterschiedliche Java-Testprogramme auf drei unterschiedlich ausgestatteten x86-Rechnern ausgeführt und verglichen. Es kann kein Test auf einer ARM-Architektur durchgeführt werden, da die dafür notwendige Hardware zum Testzeitpunkt nicht verfügbar gewesen ist. Die Ausstattung der Testsysteme ist in Tabelle 6.3 dargestellt. Insbesondere unterscheiden sich die Testsysteme durch die Größe des Hauptspeichers und der Anzahl bzw. Geschwindigkeit der Prozessorkerne. Auf Rechner #1 ist ein Prozessorkern deaktiviert worden, um zusätzlich eine möglichst große Vergleichbarkeit der Ergebnisse zwischen diesem und Rechner #3 zu erreichen. So kann die Rechenzeit, die Rechner #3 zum Ausführen eines Computerprogrammes benötigt, nach einer Ausführung auf Rechner #1 besser geschätzt werden. Rechner #2 stellt den aktuellen Hardware Entwicklungsstand dar und zeigt, welche Rechenzeiten minimal für die Tests erzielbar sind.

Tabelle 6.3: Leistungsmerkmale der Testsysteme

Nr.	CPU	MHz pro Kern	Anzahl Kerne	RAM	Speicher
1	Intel Core 2 Duo M	2000 MHz	2 (1 deaktiviert)	2 GiB	2 GiB
2	Intel Core i3-530	2933 MHz	2+2	4 GiB	>5 GiB
3	IPC	1400 MHz	1	512 MiB	500 MiB

Die Tabelle 6.4 zeigt eine Übersicht aller verwendeten Testprogramme, deren Arbeitsweise und ggf. die verwendeten Parameter. Die Quelltexte aller Programme sind in Anhang D ab Seite 259 zu finden.

Tabelle 6.4: Testprogramme

Name	Art des Tests	Parameter
`MemoryTest`	Hauptspeicher schreiben (RAM)	–
`PrimeTest`	Primzahlberechnung (CPU)	1.000.000
`CompressionTest`	Komprimierung von Daten (CPU)	10 MB random data
`DijkstraTest`	Kürzeste Wege Algorithmus (CPU/RAM)	50 50 30
Whetstone	Benchmark	–
Dhrystone	Benchmark	–

Testprogramme

Die Geschwindigkeit und der Speicherverbrauch werden mit den im Folgenden näher erläuterten Testprogrammen gemessen, welche die verfügbaren Ressourcen der Rechner wie Speicher oder Prozessor beanspruchen und vergleichbare Ergebnisse liefern. Bei einigen Tests können Parameter angegeben werden, welche die Komplexität des Tests beeinflussen. Daher ist darauf zu achten, dass auf allen Rechnern die selben Parameter genutzt werden, um eine Vergleichbarkeit zu gewährleisten. Außerdem werden mit Whetstone [Mil11] und Dhrystone [Wei11] zwei standardisierte Benchmarks ausgeführt, die häufig zur Bestimmung der Leistungsfähigkeit von Rechnern eingesetzt werden.

Speichertest Dieser Test fügt eine zufällige Zeichenkette zwei Millionen mal in eine Liste ein. Dafür muss von der virtuellen Maschine Speicher reserviert und beschrieben werden. Anschließend wird die Liste geleert und wieder beschrieben. Dies wird 100 mal wiederholt. Dieser Test kann abhängig vom Betriebsystem nicht vergleichbare Ergebnisse liefern, da der Speicher nach unterschiedlichen Verfahren reserviert wird. Auf den getesteten Rechnern wird jeweils Linux ausgeführt, wodurch dieser Umstand abgeschwächt wird.

Primzahltest Der Primzahltest bestimmt alle Primzahlen zwischen zwei und einer Zahl, die als Parameter angegeben werden kann. Um zu überprüfen, ob eine Zahl eine Primzahl ist, wird das Verfahren der Probedivision eingesetzt. Dabei wird überprüft, ob die Zahl durch eine Zahl von zwei bis $\sqrt{Zahl}$ teilbar ist. Wenn dies nicht zutrifft, ist die Zahl eine Primzahl. Das Probedivisionsverfahren wird aufgrund seiner geringen Geschwindigkeit normalerweise nicht zur Primzahlberechnung genutzt, da effizientere Verfahren existieren. Dies ist für den Test allerdings kein Nachteil. Für die Tests werden alle Primzahlen zwischen zwei und einer Million gesucht.

Kompressionstest Beim Kompressionstest wird eine Datei, welche im Speicher liegt, mittels des in Java eingebauten Gzip Ausgabestreams komprimiert, anschließend wieder dekomprimiert und das Resultat mit dem Original verglichen. Bei den Tests wird eine 10 MB große Datei mit Zufallsdaten komprimiert.

Dijkstratest Für diesen Test wird die quellenoffene Java-Bibliothek *JUNG - Java Universal Network/Graph Framework* genutzt. Diese bietet eine generische Schnittstelle an, um Graphen zu erzeugen und auf diesen verschiedene Algorithmen auszuführen. Der Dijkstra-Algorithmus zur Bestimmung von kürzesten Wegen ist in dieser Bibliothek enthalten. Dieser Test erwartet als

Parameter die Größe des Graphen in *x*- und *y*-Richtung und die Anzahl der zu berechnenden Wege. Es wird dann im ersten Schritt ein Gitter aus Knoten mit der als Parameter übergebenen *x*- und *y*-Ausdehnung erstellt. Dabei ist jeder Knoten mit dem direkten Nachbar in jeder Himmelsrichtung verknüpft. Im zweiten Schritt werden zufällige Wege berechnet. Bei diesem Test werden 30 Wege auf einem 50x50 Knoten großen Gitter berechnet.

Whetstone Der Whetstone-Benchmark ist 1976 von H. J. Curnow entwickelt und veröffentlicht worden [CW76]. Der Benchmark bestimmt dabei die Geschwindigkeit eines Rechners in „Whetstone-Instruktionen pro Sekunde“ (kurz WIPS). Der Test nutzt dabei unterschiedliche funktionale Einheiten der CPU. Es werden sowohl Gleitkomma-Operationen, als auch Ganzzahl-Operationen ausgeführt. Tarquin Mills hat eine Java-Implementierung dieses Benchmarks entwickelt, welche unter [Mil11] bezogen werden kann.

Dhrystone Im Gegensatz zum Whetstone Benchmark nutzt der Dhrystone Benchmark ausschließlich Ganzzahl-Operationen. Der Benchmark ist von Reinhold Weicker im Jahr 1984 entwickelt worden. Eine Java Implementierung kann unter [Wei11] bezogen werden.

Testprotokolle

Die Ergebnisse umfassen die benötigte Rechenzeit und den Speicherverbrauch während der einzelnen Tests. Die Rechenzeit wird in Sekunden angegeben und entsteht durch eine einfache Zeitdifferenzmessung vor Teststart und nach erfolgreichem Test. Der Speicherverbrauch wird unter Linux mit drei Werten angegeben: VIRT, RES und SHR. VIRT entspricht dem gesamten Speicherverbrauch eines Prozesses, der reservierten, aber ungenutzten Speicher enthält. Zudem wird in VIRT auch sogenannter *shared memory* einberechnet, der von mehreren Prozessen gleichzeitig verwendet wird. In diesem *shared memory* sind i. d. R. Systembibliotheken gespeichert, die so nur einmal in den Speicher geladen werden, aber von mehreren Programmen verwendet werden können. Der Anteil an tatsächlichem *shared memory* wird zudem über den Wert SHR angegeben. Der Wert RES gibt den tatsächlich belegten Speicher (ohne *shared memory*) an. Sind also beispielsweise 10 MB Speicher vom Kernel angefordert, aber nur 1 MB an Daten vom Prozess tatsächlich verändert worden, so steigt der Wert von RES um 1 MB und der Wert von VIRT um 10 MB.

Der Wert VIRT ist bei JVMs zu vernachlässigen, da die virtuelle Maschine selbst einen Stack- und Heapspeicher verwaltet und dafür direkt beim Start Speicherplatz reserviert. Das führt dazu, dass der VIRT Wert so groß wie gefordert wird, obwohl ein Programm wie z. B. „Hello World“ ausgeführt wird. Deshalb kann bei den getesten JVMs die Größe des Heapspeichers angepasst werden, wie z. B. mit den Parametern -Xms (Minimum) und -Xmx (Maximum) bei den Implementierungen von Oracle. Für die Auswertung der Tests ist der Wert von VIRT, genau wie SHR, nicht relevant, da Java wegen seiner Plattformunabhängigkeit auf externe Bibliotheken weitgehend verzichtet. Der genutzte Wert zur Beurteilung des Speicherverbrauchs ist deshalb RES.

Rechner #1 Auf Rechner #1 wird ein neu installiertes Debian Squeeze Minimal ohne Swap-Partition verwendet. Als Massenspeicher kommt ein 2 GB großer USB-Stick zum Einsatz. Damit die Geschwindigkeit des USB-Sticks keinen Einfluss auf den `CompressionTest` hat, wird der Test mit einer Datei im Dateisystem im Hauptspeicher durchgeführt. Alle Tests werden mehrfach ausgeführt und der Mittelwert gebildet, damit durch andere Systemkomponenten erzeugte Schwankungen ausgeglichen werden. Messergebnisse mit hohen Schwankungen werden bei der Mittelwertbildung vollständig ignoriert. Die Werte der Speicherauslastung (VIRT, RES und SHR) werden in einem separaten sechsten Lauf ausgelesen, um das Ergebnis der Laufzeitmessung nicht zu verfälschen.

Die Tabelle 6.5 zeigt die Bezeichnungen der getesteten Java-SE-Implementierungen und in den Tabellen 6.6 und 6.7 sind die Ergebnisse der Tests dargestellt.

Tabelle 6.5: Bezeichnungen der Java-SE-Implementierungen Rechner #1

Name	Version
OpenJDK	java version "1.6.0_18" OpenJDK Runtime Environment (IcedTea6 1.8.3) (6b18-1.8.3-2) OpenJDK 64-Bit Server VM (build 16.0-b13, mixed mode)
HotSpot	java version "1.6.0_24" Java(TM) SE Runtime Environment (build 1.6.0_24-b07) Java HotSpot(TM) 64-Bit Server VM (build 19.1-b02, mixed mode)
Java Embedded	java version "1.6.0_25" Java(TM) 2 Runtime Environment, Standard Edition for Embedded (build 1.6.0_25-b06, headless) Java HotSpot(TM) Client VM (build 20.0-b11, mixed mode)
Apache Harmony	java version "1.6.0" Apache Harmony (1.6.0) DRLVM (1.6.0-r991881)
JamVM	java version "1.6.0_18" OpenJDK Runtime Environment (IcedTea6 1.8.3) (6b18-1.8.3-2) JamVM (build 1.6.0-devel, inline-threaded interpreter)
CACAO	java version "1.6.0_18" IcedTea Runtime Environment (IcedTea6 1.8.3) (6b18-1.8.3-2) CACAO (build 0.99.4, compiled mode)
GCJ	java-1.5.0-gcj-4.4 gcc version 4.4.5 (Debian 4.4.5-2)

Die Geschwindigkeit des *OpenJDK*, der *HotSpot*-Implementierung und von *Java Embedded* sind auf einem ähnlichen Niveau. Dies ist darauf zurückzuführen, dass alle drei auf der Technologie von Oracle basieren. *Java Embedded* benötigt vor allem im `PrimeTest`, `CompressionTest`, `Whetstone`- und `Dhrystone`-Test deutlich weniger Speicher als die anderen zwei von Oracle entwickelten JVMs. Dies resultiert in einer höheren Rechenzeit, die allerdings noch deutlich geringer ist als die Rechenzeit der anderen getesten JVMs *Apache Harmony*, *JamVM* und *CACAO*. Diese benötigen teilweise beim `MemoryTest`, `CompressionTest`, `Whestone`- und `Dhrystone`-Test mehr als die doppelte Rechenzeit als die Oracle Produkte. Beim `PrimeTest` stehen Rechenzeiten von unter einer Sekunde Zeiten jenseits von 10 Sekunden gegenüber. Der Speicherverbrauch ist lediglich bei *CACAO* im Vergleich zu den anderen JVMs akzeptabel, *JamVM* und *Apache*

Tabelle 6.6: Testergebnisse Rechner #1

Implementierung	`MemoryTest`	`PrimeTest`	`CompressionTest`
OpenJDK	~39,5 s VIRT: 943m RES: 220m SHR: 8284	~0,55 s VIRT: 943m RES: 14m SHR: 8232	~28,6 s VIRT: 1008m RES: 23m SHR: 8296
HotSpot	~36 s VIRT: 687m RES: 125m SHR: 8580	~0,55 s VIRT: 687m RES: 14m SHR: 8524	~26,5 s VIRT: 752m RES: 23m SHR: 8620
Java Embedded	~38,5 s VIRT: 366m RES: 125m SHR: 3816	~1,55 s VIRT: 366m RES: 8668 SHR: 3812	~33,5 s VIRT: 368m RES: 13m SHR: 3864
Apache Harmony	~47,7 s VIRT: 1646m RES: 289m SHR: 5312	~12,3 s VIRT: 1642m RES: 44m SHR: 5304	~52 s VIRT: 1706m RES: 39m SHR: 5336
JamVM	~88 s VIRT: 650m RES: 118m SHR: 4628	~33,3 s VIRT: 711m RES: 9512 SHR: 4624	~57,2 s VIRT: 712m RES: 41m SHR: 4656
CACAO	~58 s VIRT: 210m RES: 115m SHR: 4892	~32 s VIRT: 110m RES: 14m SHR: 4888	~54,3 s VIRT: 110m RES: 16m SHR: 4920
GCJ	~29,1 s VIRT: 185m RES: 134m SHR: 23m	~1,55 s VIRT: 86304 RES: 33m SHR: 23m	~19,4 s VIRT: 92700 RES: 45m SHR: 23m

Harmony brauchen besonders im `CompressionTest` deutlich mehr Speicher. *GCJ* zeigt, dass kompilierte Software deutlich leistungsfähiger bezüglich der Rechenzeit sind. Das schlechtere Ergebnis im `PrimeTest` und `Dhrystone`-Test ist mit dem Mehraufwand der Java-Übersetzung zum Programmstart, der bei einer sehr kurzen Rechenzeit von unter einer Sekunde deutlicher ins Gewicht fällt, zu begründen.

Die Größe des Heapspeichers kann die Rechenzeit beeinflussen. Dies macht sich beim *OpenJDK* bemerkbar, da die Rechenzeit durch einen festen Heapspeicherwert von 256 MB im `MemoryTest` auf 25 Sekunden reduziert werden kann. Diese Ersparnis von über 10 Sekunden lässt sich darauf zurückführen, dass der Speicherbereich weniger fragmentiert vom Kernel angefordert wird und daher direkt als ein großer Block beim Start der JVM bereitsteht. Der gesamte Speicher wird so beim Start des Tests reserviert und nicht während der Laufzeit in mehreren Teilen angefordert. Da alle JVMs mit einer dynamischen Heapgröße getestet werden, sind die Ergebnisse trotzdem vergleichbar.

Tabelle 6.7: Weitere Testergebnisse Rechner #1

Implementierung	Whetstone	Dhrystone
OpenJDK	~225400 KWIPS VIRT: 1015m RES: 22m SHR: 8351	~17 ms VIRT: 946m RES: 17m SHR: 8296
HotSpot	~203300 KWIPS VIRT: 696m RES: 24m SHR: 8680	~19 ms VIRT: 696m RES: 14m SHR: 7931
Java Embedded	~151000 KWIPS VIRT: 366m RES: 9016 SHR: 3872	~16 ms VIRT: 366m RES: 9024 SHR: 3632
Apache Harmony	~66200 KWIPS VIRT: 1642m RES: 45m SHR: 5332	~70 ms VIRT: 1644m RES: 42m SHR: 4816
JamVM	~29000 KWIPS VIRT: 647m RES: 9560 SHR: 4632	~227 ms VIRT: 711m RES: 9624 SHR: 4592
CACAO	~40500 KWIPS VIRT: 110m RES: 14m SHR: 4896	~30 ms VIRT: 110m RES: 14m SHR: 4943
GCJ	~95800 KWIPS VIRT: 86m RES: 33m SHR: 23m	~41 ms VIRT: 86m RES: 33m SHR: 23m

Rechner #2 Rechner #2 nutzt die Server-Version von Debian Squeeze, welche ohne Swap-Partition auf einer 10 GB großen Festplatte installiert ist. In Tabelle 6.8 sind die getesteten JVMs zu finden. Aufgrund eines späteren Testzeitpunktes liegt das *OpenJDK* in einer neueren Version vor. Die Ergebnisse der auf diesem Rechner ausgeführen Belastungstests sind in den Tabellen 6.9 und 6.10 dargestellt. Es hat sich bestätigt, dass die von Oracle implementierten JVMs den anderen Lösungen sowohl im Speicherverbrauch, als auch in der Rechenzeit, deutlich überlegen sind. Die quelloffene JVM-Implementierung der Apache Software Foundation benötigt verglichen mit den Oracle-Implementierungen in einigen Tests fast die vierfache Rechenzeit. Lediglich im `DijkstraTest` kann diese virtuelle Maschine konkurrenzfähige Ergebnisse erzielen und benötigt nur 40% mehr Rechenzeit als das *OpenJDK*. Der Speicherverbrauch ist ebenfalls deutlich höher. Die *Cacao* VM liefert ein ähnliches Ergebnis bei der Rechenzeit und des Speicherverbrauches wie Apache Harmony, liegt aber im `PrimeTest` deutlich zurück. Die *JamVM* benötigt hinstichtlich der Rechenzeiten und des Speicherverbrauches ähnlich viele Ressourcen wie die *Cacao* VM und Apache Harmony. Verglichen mit den von Oracle implementierten JVMs sind die *JamVM* gemessenen Werte deutlich schlechter. Die mittels GCJ kompilierten Binärdateien erzeu-

gen in allen Tests gute Ergebnisse. Die übrigen drei getesteten JVMs befinden sich etwa auf dem gleichen Niveau, wobei das *OpenJDK* und die *HotSpot*-Implementierung etwas weniger Rechenzeit benötigen, dafür allerdings mehr Speicher nutzen als die Java-Embedded-Implementierung.

Tabelle 6.8: Versionsnummern der Java-SE-Implementierungen Rechner #2

Name	Version
OpenJDK	java version "1.6.0_18" OpenJDK Runtime Environment (IcedTea6 1.8.7) (6b18-1.8.7-2 squeeze1) OpenJDK 64-Bit Server VM (build 14.0-b16, mixed mode)
HotSpot	java version "1.6.0_24" Java(TM) SE Runtime Environment (build 1.6.0_24-b07) Java HotSpot(TM) 64-Bit Server VM (build 19.1-b02, mixed mode)
Java Embedded	java version "1.6.0_25" Java(TM) 2 Runtime Environment, Standard Edition for Embedded (build 1.6.0_25-b06, headless) Java HotSpot(TM) Client VM (build 20.0-b11, mixed mode)
Apache Harmony	java version "1.6.0" Apache Harmony (1.6.0) DRLVM (1.6.0-r991881)
JamVM	java version "1.6.0_18" OpenJDK Runtime Environment (IcedTea6 1.8.3) (6b18-1.8.3-2) JamVM (build 1.6.0-devel, inline-threaded interpreter)
CACAO	java version "1.6.0_18" IcedTea Runtime Environment (IcedTea6 1.8.3) (6b18-1.8.3-2) CACAO (build 0.99.4, compiled mode)
GCJ	java-1.5.0-gcj-4.4 gcc version 4.4.5 (Debian 4.4.5-2)

Rechner #3 Rechner #3 ist ein Wago IPC, der auf der Versuchsanlage des Lehrstuhl für Förder- und Lagerwesen der TU Dortmund eingesetzt wird. Dieser ist in Kapitel 6.1 beschrieben. Auf dem Rechner ist ein Linux mit einem vom Hersteller zusammengestellten Linux Kernel installiert. Da sich aufgrund der Ergebnisse von Rechner #1 und #2 bereits eine Tendenz für die Java-Embedded-Implementierung abzeichnet, geht es in diesem Test nur noch darum, die Funktionstüchtigkeit der VMs nachzuweisen. Die Tests werden jeweils mit eingeschalteter und ausgeschalteter SPS-Laufzeitumgebung durchgeführt. Dabei wird nur der RES-Speicherverbrauch protokolliert, da die anderen Werte keine Grundlage für einen Vergleich bieten. Die Ergebnisse dieses Rechners sind in Tabelle 6.11 zu finden.

Merkmalsanalyse und Fazit

FiLeCC wird für Zielsysteme mit vergleichsweise geringen Rechen- und Speicherressourcen ausgelegt. Daraus ergeben sich spezielle Anforderungen an die verwendete Ausführungsplattform, insbesondere geringer Speicherverbrauch und hohe Effizenz in der genutzten Rechenzeit. In diesem Abschnitt werden daher die derzeit verfügbaren Java-SE-Plattformen hinsichtlich ihrer Funktionalität untersucht und mit Hilfe von Testläufen jeweils Speicherverbrauch und

Tabelle 6.9: Testergebnisse Rechner #2

Implementierung	`MemoryTest`	`PrimeTest`	`DijkstraTest`
OpenJDK	~18,9s VIRT: 943m RES: 226m SHR: 8076	~0,2 s VIRT: 949m RES: 20m SHR: 8152	~1 s VIRT: 943m RES: 20m SHR: 8428
HotSpot	~20,5s VIRT: 1137m RES: 128m SHR: 8592	~0,2 s VIRT: 1137m RES: 15m SHR: 8524	~1 s VIRT: 1150m RES: 53m SHR: 9072
Java Embedded	~25,2s VIRT: 366m RES: 135m SHR: 3796	~0,6 s VIRT: 366m RES: 8852 SHR: 3784	~0,5 s VIRT: 367m RES: 22m SHR: 3844
Apache Harmony	~73s VIRT: 1648m RES: 289m SHR: 5456	~16,8 s VIRT: 1644m RES: 45m SHR: 5452	~1,4 s VIRT: 1646m RES: 43m SHR: 4836
JamVM	~83,5 s VIRT: 1087m RES: 125m SHR: 5312	~19 s VIRT: 1063m RES: 10473 SHR: 5284	~1,4 s VIRT: 1074m RES: 42m SHR: 5111
CACAO	~33,2 s VIRT: 211m RES: 116m SHR: 5220	~18,7 s VIRT: 111m RES: 16m SHR: 5200	~1,3 s VIRT: 146m RES: 44m SHR: 5314
GCJ	~15 s VIRT: 185m RES: 133m SHR: 21m	~0,6 s VIRT: 88692 RES: 31m SHR: 21m	~0,8 s VIRT: 95183 RES: 48m SHR: 21m

benötigte Rechenzeit verglichen. Die Ergebnisse dieser Untersuchung sind in Tabelle 6.12 zusammengefasst.

Der Merkmalanalyse liegen folgende Bewertungskriterien zu Grunde:

Architekturen Das Pflichtenheft schreibt vor, dass sowohl die x86- als auch die ARM-Architektur unterstützt werden muss. Laufzeitumgebungen, die für eine dieser Architekturen nicht verfügbar sind, werden mit - bewertet, ansonsten mit +.

Aktualität Die Java-Plattform durchläuft eine stetige Entwicklung, sodass veraltete und nicht fortgeführte virtuelle Maschinen schnell an Kompatibilität und damit an Nutzen verlieren. Die Bewertung dieses Kriteriums erfolgt anhand des Datums der zuletzt veröffentlichten Version. Sehr aktuelle Plattformen erhalten eine ++-Bewertung, veraltete hingegen eine - Bewertung. Eine Abstufung der Bewertung einer Plattform erfolgt im Vergleich mit den übrigen Plattformen.

Tabelle 6.10: Weitere Testergebnisse Rechner #2

Implementierung	Whetstone	Dhrystone
OpenJDK	~369800 KWIPS VIRT: 949m RES: 20m SHR: 8152	~14,5 ms VIRT: 946m RES: 17m SHR: 8088
HotSpot	~334000 KWIPS VIRT: 1146m RES: 23m SHR: 8700	~13,1 ms VIRT: 1141m RES: 13m SHR: 8084
Java Embedded	~237000 KWIPS VIRT: 366m RES: 9148 SHR: 3828	~11,6 ms VIRT: 366m RES: 9032 SHR: 3792
Apache Harmony	~75600 KWIPS VIRT: 1644m RES: 45m SHR: 5488	~50 ms VIRT: 1644m RES: 42m SHR: 4816
JamVM	~34100 KWIPS VIRT: 943m RES: 9356 SHR: 4965	~198 ms VIRT: 943m RES: 9485 SHR: 5216
CACAO	~61400 KWIPS VIRT: 111m RES: 16m SHR: 5224	~22 ms VIRT: 111m RES: 16m SHR: 5196
GCJ	~123500 KWIPS VIRT: 83m RES: 36m SHR: 21m	~26 ms VIRT: 89m RES: 36m SHR: 21m

Tabelle 6.11: Testergebnisse Rechner #3

Implementierung	`MemoryTest`	`PrimeTest`	`DijkstraTest`
Java Embedded	~128,9 s RES: 98m	~4,8 s RES: 8672	~2,3 s RES: 21m
Java Embedded (SPS abgeschaltet)	~67,5 s RES: 98m	~2,4 s RES: 8672	~1,3 s RES: 21m

Speicherverbrauch Da der Arbeitsspeicher der Zielsysteme sehr begrenzt ist, muss der benötigte Speicher der Java-Plattform möglichst gering sein. Für die Bewertung werden die gemittelten Resultate der Testläufe verglichen und in ein relatives Bewertungsschema mit den Noten ++, +, -, - - übertragen.

Tabelle 6.12: Merkmalanalyse der untersuchten Java-SE-Implementierungen

Name	Architekturen	Aktualität	kum. Rechenzeit	kum. Speicherv.
OpenJDK	+	++	+	-
HotSpot	-	++	+	+
Java Embedded	+	++	+	++
Apache Harmony	-	++	- -	- -
JamVM	+	+	- -	+
CACAO	+	-	- -	+
(*GCJ*)	-	++	++	-

Werden die Ergebnisse der Leistungstests betrachtet, so sticht der *GCJ* aufgrund des zu erwartenden Geschwindigkeit heraus, da hier der gesamte Quelltext vorkompiliert wird. Zusätzlich zu der zuvor bereits genannten eingeschränkten Nutzbarkeit für Cloudanwendungen kann diese Implementierung jedoch durch die fehlende Architekturunterstützung, sowie den hohen Speicherbedarf, in diesem Projekt nicht genutzt werden. Trotz ähnlich guter Testergebnisse kann auch Oracles *Hotspot*-VM aufgrund fehlender ARM-Unterstützung nicht berücksichtigt werden. Die geeignetesten Testkandidaten sind *OpenJDK* und *Java Embedded*, die in beinahe allen Testpunkten ähnliche Resultate erzielen. Die getestete *Java-Embedded*-Implementierung benötigt dabei deutlich weniger Arbeitsspeicher und ist damit für dieses Projekt von allen Alternativen am besten geeignet.

6.2.4 Anzahl der verwendeten JVMs

Da FiLeCC auf Systemen lauffähig sein soll, die gemessen an aktuellen Maßstäben wenig Hauptspeicher zur Verfügung stellen, ist eine effiziente Nutzung dieses Speichers nötig. Die vorgestellten JVM-Implementierungen verfügen über eine interne Speicherverwaltung, allerdings gibt es keinen expliziten Mechanismus, um reservierten und ggf. ungenutzten Speicher wieder freizugeben. Das liegt insbesondere daran, dass die Speicherverwaltung von der virtuellen Maschine von Java selbstständig durchgeführt wird. Deshalb wird bereits beim Start der JVM ein Teil des Hauptspeichers für die JVM reserviert. Die Größe des reservierten Teils ist beim Programmstart einstellbar. Allerdings wird keine Rückskalierung von ungenutztem Speicher durchgeführt. Für die Umsetzung von FiLeCC stellt sich deshalb die Frage, wie viele JVMs auf den beteiligten Knoten eingesetzt werden sollen. Deshalb werden in diesem Abschnitt die folgenden vier Einsatzvarianten der JVMs betrachtet und diskutiert, welche für dieses Projekt am besten geeignet ist.

1. Eine JVM für das gesamte System
2. Eine JVM für die Cloud-Plattform, eine JVM für alle Anwendungen
3. Eine JVM für die Cloud-Plattform, eine JVM für pro Anwendung
4. Eine JVM für jede Komponente der Cloud-Plattform, eine JVM für jede Anwendung

Tabelle 6.13 zeigt die Merkmalanalyse der einzelnen Szenarien. Sie unterscheiden sich in der Anzahl der eingesetzten JVMs, da jede JVM ihre eigene Speicherverwaltung und damit einen Verwaltungsmehraufwand hat. Insbesondere der reservierte Speicher führt bei vielen JVM-Implementierungen zu einem größeren Anteil an ungenutztem Speicher. Mehrfach verwendete

Tabelle 6.13: Merkmalanalyse JVM Einsatzszenarien

Szenario	Speichernutzung	Geschwindigkeit	Ausfallsicherheit	Komplexität
1	++	++	- -	+
2	+	++	-	-
3	-	-	+	- -
4	- -	- -	++	- -

Klassen (z. B. aus der Java Standardbibliothek) können innerhalb einer JVM wiederverwendet werden. Das führt zu einem geringeren Speicherverbrauch. Aus diesen zwei Gründen haben die Szenarien 1 und 2 einen deutlichen Vorteil bei der effizienten Speichernutzung gegenüber den Szenarien 3 und 4. Außerdem gilt, dass bei effizienter Speichernutzung gleichzeitig das Abfangen von Lastspitzen beim Speicherverbrauch erleichtert wird, da der freie Speicher effizient genutzt werden kann. Gleichzeitig kann der Hauptspeicher bei den Szenarien 1 und 2 vollständig beim Start reserviert werden, was die Ausführung von Programmen beschleunigt.

Szenario 3 und 4 haben dagegen Vorteile bei der Ausfallsicherheit. Im ersten Szenario bedeutet ein Ausfall der JVM gleichzeitig den Ausfall des gesamten Knotens. Beim zweiten Szenario ist ein Ausfall der Anwendungs-JVM weniger kritisch, da diese von der Cloud-Plattform erneut gestartet werden kann. Ein Ausfall der JVM für die Cloud-Plattform hat zur Folge, dass die Überwachung und Steuerung der Anwendungsprogramme unmöglich wird. Die Anwendungen sind allerdings weiterhin lauffähig. Das dritte Szenario unterscheidet sich vom zweiten dadurch, dass jede Anwendung ihre eigene JVM verwendet. Dadurch hat der Absturz einer Anwendungs-JVM lediglich die Auswirkung, dass eine Anwendung bis zum Neustart durch die Cloud-Plattform nicht zur Verfügung steht. Allerdings hat ein Ausfall der JVM für die Cloud-Plattform die gleichen Folgen wie im zweiten Szenario. Deshalb wird im vierten Szenario die Cloud-Plattform auf mehrere JVMs aufgeteilt, um so bei einem Absturz einzelner Komponenten-JVMs diese neustarten zu können.

Mit mehreren JVMs steigt gleichzeitig die Komplexität. So muss bei den Szenarien 3 und 4 für die Rückskalierung des Speichers gesorgt werden. Außerdem muss der Speicherverbrauch für die Cloud-Plattform in den Szenarien 2, 3 und 4 abgeschätzt werden, damit beim Start ein geeigneter Anteil des Speichers reserviert wird. In den Szenarien 3 und 4 muss dies für jede Anwendung durchgeführt werden. Außerdem sind nur in Szenario 1 direkte Methodenaufrufe bzw. die direkte Überwachung von Anwendungen möglich, da die Cloud-Plattform und die Anwendungen in der selben JVM betrieben werden. In den andern Szenarien müssten spezielle Schnittstellen zur Steuerung in den JVMs der Anwendungen zur Verfügung stehen, auf die die Cloud-Plattform zugreifen kann. Die Überwachung einzelner JVMs kommt zusätzlich zur Anwendungsüberwachung in den Szenarien 2, 3 und 4 hinzu. Allerdings müssen in den Szenarien 1 und 2 drei zusätzliche Maßnahmen für die Isolation einzelner Anwendungen getroffen werden. Dazu gehört die Implementierung eines *ClassLoader*, der Klassen virtuell im Speicher isoliert. So kann eine Klasse mehrfach in unterschiedlichen Kontexten geladen und verwendet werden. Ein Zugriff auf Klassen aus externen Kontexten ist nicht möglich. Daneben muss der Zugriff auf einige Funktionen der JVM für die Anwendungsprogramme eingeschränkt werden. Beispielsweise darf ein Anwendungsprogramm die JVM im Szenario 1 und 2 nicht beenden (`System.exit()` aufrufen). Dies kann durch die Implementierung eines *Security Manager* gelöst werden. Zudem muss sichergestellt werden, dass keine Programmfehler (*Exceptions*) zu einem Ausfall der JVM führen. Dies kann durch eine allgemeine Fehlerbehandlung in der Cloud-Plattform realisiert wer-

den, bei der nicht behandelte Fehler der Anwendungen protokolliert und verworfen werden. Alle drei Maßnahmen sind von Java bereits vorgesehen und erfordern nur geringeren Mehraufwand und erhöhen die Komplexität kaum.

Wie in der Tabelle ablesbar ist, entsteht ein Zielkonflikt zwischen Ressourceneffizienz und Ausfallsicherheit. Die ressourceneffizienten Lösungen haben eine geringere Ausfallsicherheit, während bei Lösungen mit einer hohen Ausfallsicherheit die Ressourcen nicht effizient verwendet werden. Da der Schwerpunkt bei der Entwicklung der Cloud-Plattform auf ressourcenbeschränkten Geräten liegt, wird das in Szenario 1 beschriebene Verfahren umgesetzt, da bei diesem Verfahren die effizienteste Speichernutzung mit der höchsten Geschwindigkeit vorliegt. Außerdem ist durch die geringe Komplexität eine robuste Implementierung möglich, durch die die negativen Aspekte der Ausfallsicherheit kompensiert werden.

6.3 Cloud-Lösungen

In Kapitel 4.6 wird auf die verschiedenen Arten von Cloud-Architekturen, ihre Ebenen (siehe Abbildung 4.14) und die zugrunde liegenden Techniken eingegangen. Da die grundlegende Anforderung an FiLeCC darin besteht, die Ressourcen eines Rechnernetzwerkes von leistungsschwachen Computern zu einer Cloud zu verbinden, wird FiLeCC als PaaS (siehe Abschnitt 4.6.2) realisiert. Zu klären bleibt, in welcher Weise die Infrastruktur den darüberliegenden Schichten zur Verfügung gestellt wird. Die in diesem Zusammenhang interessanten Ansätze sind, Public Clouds anzubinden, um die verfügbaren Ressourcen zu erweitern, und virtuelle Maschinen zur Abstraktion von den eigentlichen Rechnern einzusetzen. Diese Ansätze werden in den folgenden Abschnitten erläutert und ihre Eignung für FiLeCC geprüft.

6.3.1 Public-Cloud-Lösungen

Public-Cloud-Lösungen bieten Dienstleistungen, in denen Rechenkapazität oder Speicher in Form einer Cloud von einem externen Dienstleister für eine große Gruppe von Nutzern zur Verfügung gestellt wird. Es wird eine flexible Infrastruktur angeboten, um deren Wartung sich der Nutzer nicht mehr kümmern muss und bei der nur genau die Leistung bezahlt wird, die auch genutzt wurde. Dies hat für den Nutzer Vorteile und der Dienstleister kann ein eventuelles Überangebot an eigener Rechenkapazität vermarkten. Die Nutzung von Public Clouds bietet somit die Möglichkeit, die eigene Infrastruktur zu erweitern und bei plötzlichen Lastsituationen schnell herauf zu skalieren. Der Markt für Public Clouds ist breit gefächert und bietet sowohl IaaS, als auch PaaS Lösungen an, unter denen sich Anbieter wie Windows Azure (PaaS) oder Amazon EC2 (IaaS) befinden [Mic11a, Ama11].

Durch die Integration von Public-Cloud-Lösungen in FiLeCC ist eine Erweiterung der vorhandenen Infrastruktur, sowie Backup-Konzepte zur Kompensation eventueller Systemausfälle denkbar. Ausfälle einer oder mehrerer IPCs einer Förderanlage können zu einem kurzfristigen Ressourcenmangel führen, der den Ausfall einer Förderanlage verursachen kann. In diesem Fall ist es wünschenswert, innerhalb kurzer Zeit, externe Ressourcen schnell und unkompliziert zuschalten zu können, um die benötigte Rechenkapazität abzudecken. Eine einzelne Cloud-Serverinstanz übertrifft i.d.R. die Rechenleistung eines durchschnittlichen IPCs und befindet sich auf einem sehr niedrigen Preisniveau, wenn nur temporäre Ausfälle in der eigenen Anlage abgedeckt werden

sollen. Eine Cloud-Instanz kann innerhalb weniger Minuten voll automatisch gestartet werden, sodass ein eventueller Ressourcenmangel ausgeglichen wird. Die Ausfallzeiten einer Public Cloud sind bedingt durch ihre Größe relativ gering und große Anbieter garantieren in ihren *Service Level Agreements* in der Regel eine Verfügbarkeit von 99.9% auf einen Monat gerechnet [Ama12]. Softwareprodukte wie FiLeCC können in *deploybare Images* eingearbeitet werden oder auf zusätzlichem Speicher in der Cloud hinterlegt werden, sodass diese mit dem Start einer Instanz ausgeführt und in die interne Struktur integriert werden können ([Ama11, Mic11b]).

Die Integration einer Public Cloud in eine Cloud von IPCs ist technisch realisierbar und kann einen gewissen Nutzen bieten. Dennoch gibt es Argumente, die dagegen sprechen.

Durch die Nutzung von externen Servern werden Daten aus dem eigenen Verwaltungsbereich heraus gegeben, die sowohl auf dem Transportweg, als auch in der Public Cloud Sicherheitsrisiken ausgesetzt sind. Sensible Daten müssen daher in der firmeninternen Serverstruktur bleiben, sodass die Gefahren und Bedenken bezüglich der Sicherheit minimiert werden. Dies ist jedoch, wie in den *NIST - Guidelines on Security and Privacy in Public Cloud Computing* [JG11] beschrieben, nicht immer sicherzustellen. Hinzu kommt, dass eine Förderanlage zukünftig eine hinreichende Anzahl an IPCs mitbringt, um die konzeptionell vorgesehenen Aufgaben (z. B. das Routing von Paketen) zu bewältigen. Desweiteren ist anzunehmen, dass bei einem Ausfall eines IPCs die physikalische Verbindung zu den ausgefallenen Teilen der Förderanlage ebenfalls ausfällt. In diesem Fall ist auch eine Backup-Lösung nur von begrenztem Nutzen für die Aufrechterhaltung der Lauffähigkeit der Anlage.

Die Motivation des FiLeCC-Projekts ist, die freien Ressourcen innerhalb der Anlage zu nutzen, um Kosten zu sparen und effizienter zu arbeiten. Die Anbindung einer vergleichsweise großdimensionierten Public Cloud widerspricht diesem Ansatz, da die Rechenressourcen der Förderanlage in vielen Fällen kleiner dimensioniert sind und an Bedeutung verlieren. Somit kann in diesem Fall komplett auf eine kommerzielle Cloud-Lösung gesetzt werden und der Effizienzgewinn durch die firmeneigene Infrastruktur geht verloren. Zudem hat die Nutzung von Public Clouds einen nicht unerheblichen Preis und ist im Kosten/Nutzen Verhältnis auf lange Zeit gerechnet wesentlich teurer als die Nutzung fester, lokal vorhandener Ressourcen. Somit wird die dauerhafte Nutzung von Public Clouds im Zusammenhang mit FiLeCC ausgeschlossen.

6.3.2 Infrastrukturlösungen auf Basis virtueller Maschinen

Essentiell für die meisten großen Public-Cloud-Umgebungen ist die Virtualisierung der Serverhardware [BKN+11]. Kunden nutzen die Ressourcen dieser Clouds in Form von virtuellen Maschinen (VMs). Der Cloud stehen dazu bei den großen Anbietern häufig mehrere Rechenzentren zur Verfügung, über die die virtuellen Maschinen der Kunden ausfallsicher verteilt werden. Neben den kommerziellen Angeboten existieren Open-Source-Lösungen, mit denen dieser Bereich einer IaaS-Lösung (siehe Abschnitt 4.6.1) durch die Virtualisierungstechniken *Xen* [Cit05], *KVM* [Lin12] oder *VMWare* [VMw12b] realisiert werden kann. Hier sind besonders *Eucalyptus* [Euc12] und *OpenNebula* [Ope02] zu nennen, die sich dadurch unterscheiden, dass *Eucalyptus* weitgehend kompatibel zu Amazons Public Cloud ist, wohingegen *OpenNebula* im Bereich der Virtualisierungstechniken weiter entwickelt ist [BKN+11]. Es unterstützt beispielsweise VMWare auch in der Open-Source-Version, Migration von VMs im Betrieb und spezielle Clusteringoptionen für *High Performance Computing*.

Für FiLeCC sind Konzepte auf der Basis virtueller Maschinen nicht geeignet, da die Ziele bei FiLeCC grundlegend anders sind. Während Virtualisierungstechniken für IaaS zur Konsolidierung dienen, also den Betrieb mehrerer virtueller Maschinen auf einer realen Hardware ermöglichen, ist bei FiLeCC eine Lösung notwendig, die es ermöglicht, die Ressourcen vieler ressourcenschwacher Rechner gemeinsam zu nutzen. Dadurch können auch solche Aufgaben erfüllt werden, die einen einzelnen dieser Rechner überfordern. Weiterhin erzeugt Virtualisierung einen deutlichen Mehraufwand in Form von Rechenressourcen, insbesondere auf Plattformen ohne Hardwareunterstützung für Virtualisierung, wie sie typisch für ressourcenschwächere Rechner sind. Das gesamte Konzept bietet für FiLeCC also keinen Vorteil, weil das Ziel gegensätzlich ist und bringt zudem Leistungseinbußen mit sich. Der Einsatz von virtuellen Maschinen ist also nicht geeignet für dein Einsatz in FiLeCC.

6.3.3 Zusammenfassung

Für FiLeCC ist der Einsatz von virtuellen Maschinen, als Abstraktion von den realen Maschinen, aus den genannten Gründen nicht sinnvoll. FiLeCC erbringt die Plattformdienste daher direkt auf der jeweiligen Systemhardware. Weiterhin ist die Anbindung von Public Clouds aus verschiedenen Gründen derzeit nicht vorgesehen. Die Cloud-Plattform FiLeCC ist daher unabhängig von verbindlichen Vorgaben, die bei Anbindung einer Public Cloud zu beachten sind.

6.4 Kommunikation auf Basis von Webservices

Zur Kommunikation der einzelnen Programme und Programmteile untereinander soll eine SOA genutzt werden, dessen generelle Architektur in Kapitel 4.3.3 beschrieben ist. Zur Umsetzung der SOA werden Webservices gewählt. Es existiert eine Vielzahl von Webservicespezifikationen. Aus diesen Spezifikationen muss eine für das Projekt FiLeCC geeignete Menge ausgewählt werden. Dazu kann ein sogenanntes Profil genutzt werden. Im Zusammenhang mit Webservices ist ein Profil eine Kombination aus mehreren Teilspezifikationen. Somit kann ein Profil auf eine Anwendung zugeschnitten werden. Dadurch können kompakte Implementierungen dieser Profile erstellt werden, die nicht sämtliche Teile der gesamten Webservicespezifikation unterstützen müssen. Ein bereits existierendes Profil bietet den Vorteil, dass es bereits anerkannt und seine Verbreitung größer ist.

6.4.1 Auswahl eines Webservice Profiles

Das in Kapitel 4.3.3 beschriebene Webservicesprofil DPWS eignet sich in vielerlei Hinsicht für den Einsatz im Projekt FiLeCC. DPWS beschreibt ein kompaktes, aber dennoch vollständiges Profil. Die in der DPWS-Spezifikation beschriebene Kombination bestehender Webservicespezifikationen zielt darauf ab, insbesondere auf ressourcenschwachen Geräten eingesetzt zu werden. Eine automatisierte Anlage, deren verteilte Rechenressourcen zu einer Cloud zusammengefasst werden sollen, besteht zu großen Teilen aus solchen ressourcenbeschränkten Geräten. Für den Einsatz von Webservices auf diesen Geräten erscheint die DPWS-Spezifikation deshalb als geeignet. Die DPWS-Spezifikation ermöglicht weiterhin einen vollständig dezentralen Betrieb. Dies wird u. a. durch die Integration der WS-Discovery-Spezifikation erreicht, die eine dynamische Suche nach Webservices ohne zentrale Vermittlungsstelle ermöglicht (vgl. [MK09]). Die

WS-Discovery-Spezifikation sucht nach sogenannten *Target Services*. Dabei ergibt sich für das Projekt FiLeCC ein Nachteil der DPWS-Spezifikation: die Unterteilung von Webservices in *Geräte* und *Dienste* (vgl. 4.3.3), wobei nur die Geräte als *Target Services* gelten und über die WS-Discovery-Spezifikation auffindbar sind. Nach der DPWS-Spezifikation kapseln die *Geräte* die *Dienste*, welche die eigentlichen Funktionalitäten bereitstellen. In dem Projekt FiLeCC werden nach der DPWS-Definition von Webservices nur die DPWS-*Dienste* benötigt. Ein positiver Aspekt von DPWS ist die Verfügbarkeit von Implementierungen in Programmiersprachen wie Java, C/C++ oder .NET. Einige dieser Implementierungen können auf eingeschränkten Ausführungsplattformen wie etwa der *Java Connected Limited Device Configuration* (Java CLDC) (vgl. 6.2.2) oder dem .NET-Micro-Framework eingesetzt werden. Letzteres unterstützt DPWS nativ, sodass keine zusätzlichen Bibliotheken benötigt werden.

Aufgrund der Vorteile, die DPWS für das Projekt FiLeCC bietet, soll das Profil eingesetzt werden. Im nächsten Abschnitt werden deshalb verschiedene DPWS-Implementierungen vorgestellt.

6.4.2 DPWS-Implementierungen

Für den Einsatz von DPWS in dem Projekt FiLeCC ist eine externe DPWS-Implementierung notwendig, da in keiner mitgelieferten Standardbibliothek der Java Laufzeitumgebung eine eigene Implementierung der DPWS-Spezifikation bereitgestellt wird, ausgenommen dem .NET-Framework. Aufgrund der Tatsache, dass auf den eingesetzten Geräten ein Linux-basiertes Betriebsystem installiert ist (vgl. 6.2.1), muss die Auswahl auf solche DPWS-Implementierungen beschränkt werden, die auf diesem Betriebssystem einsetzbar sind. Die Verwendung des .NET-Frameworks ist somit nicht möglich. Mit *Mono* existiert zwar eine Open-Source-Implementierung von .NET für Linux, diese implementiert allerdings einen älteren .NET-Standard, welcher DPWS nicht enthält (siehe dazu auch [Xam11]). Im Folgenden werden einige DPWS-Implementierungen vorgestellt und deren Eignung für den Einsatz in diesem Projekt untersucht.

Apache Axis2/Java *Apache Axis2*/Java ist ein Java-basiertes Framework für die Erstellung von Webservices, das eine Neuentwicklung des Vorgängers Apache Axis 1.0 darstellt und seit dem Jahr 2005 unter der *Apache License 2.0* verfügbar ist (siehe [Apa04]). *Apache Axis2* ist ein Framework, das nicht nur auf die DPWS-Spezifikation beschränkt ist, sondern durch zahlreiche Module eine Vielzahl von Webservicespezifikationen vereint. Dies hat zur Folge, dass dieses Framework für den hier vorliegenden Anwendungsfall zu viele Funktionalitäten mitbringt, die nicht benötigt werden, und somit unnötig viele Ressourcen verbraucht. Weiterhin ist für den Betrieb von Webservices unter *Apache Axis2* ein Servlet-Container wie z. B. Apache Tomcat erforderlich, dessen Einsatz auf ressourcenbeschränkten Geräten wiederum Probleme verursachen kann.

WS4D-Axis2 Web Services for Devices (WS4D) ist eine Initiative, die sich u. a. mit der Verbindung von industriellen Automatisierungsanlagen mit Technologien von Serviceorientierten Architekturen sowie Webservices befasst. Hauptmitglieder dieser Initiative sind die Universität Rostock, die TU Dortmund und die in Dortmund ansässige Firma MATERNA

GmbH Information & Communication (siehe [WS412]). Diese Initiative bietet u. a. die DPWS-Implementierung WS4D-Axis2 an, welche das im vorherigen Abschnitt vorgestelle *Apache-Axis2*-Framework um ein Modul erweitert, das für das *Apache-Axis2*-Framework eine Implementierung der DPWS-Spezifikation bereitstellt (siehe [Uni11d]). Diese Erweiterung ist wie das *Apache-Axis2*-Framework selbst unter der *Apache License 2.0* veröffentlicht. Da es sich bei *WS4D-Axis2* um eine Erweiterung des *Apache-Axis2*-Frameworks handelt, sind hier ebenfalls der Ressourcenverbrauch und der zwingende Einsatz eines Servlet-Containers als Nachteile für die Nutzung im Projekt FiLeCC zu nennen. Weiterhin verfügt diese Erweiterung über eine eingeschränkte Dokumentation und wird darüber hinaus derzeit nicht aktiv weiterentwickelt, was daran festzumachen ist, dass die letzte Aktualisierung im Projekt-Repository bereits über zwei Jahre zurückliegt.

WS4D-gSOAP WS4D-gSOAP ist ein Webservice-Framework für die Entwicklung von DPWS-basierten Webservices in C/C++, das von der Universität Rostock, einem Mitglied der WS4D-Initiative, entwickelt wird (siehe [Uni11b]). Dieses Framework steht unter der GPL/LGPL-Lizenz und ist somit frei verfügbar. Es nutzt das gSOAP-Toolkit, „ein Open Source C und C++ Softwareentwicklungs-Toolkit für SOAP/XML Webservices“ (siehe [EG02]), zur Umsetzung der DPWS-Spezifikation. Dadurch nutzt das WS4D-gSOAP-Framework den Mechanismus von gSOAP, aus den Metadaten eines Gerätes und einer Servicebeschreibung in der WSDL, das benötigte Quelltextgerüst für die Webservices automatisch zu generieren. Der Programmierer beschränkt sich anschließend darauf, die Dienstoperationen zu implementieren. WS4D-gSOAP ist ein ressourcenschonendes Framework, das außerdem umfangreich dokumentiert ist. Es ist für den Einsatz auf ressourcenbeschränkten Geräten geeignet.

Java Multi Edition DPWS Stack (JMEDS) Der *Java Multi Edition DPWS Stack* (JMEDS) wird von der Dortmunder Firma MATERNA GmbH Information & Communication und der TU Dortmund entwickelt (siehe [Uni11c]). Es ist ein in Java entwickeltes Framework, mit dessen Hilfe DPWS-basierte Webservices für Geräte entwickelt werden können. Als Lizenz liegt die *Eclipse Public License* zugrunde (vgl. [Uni11c]). Durch den modularen Aufbau, der es ermöglicht, einzelne Module nicht zu verwenden, ist das JMEDS-Framework konfigurierbar. Weiterhin ist ein Einsatz auf den speziell für ressourcenbeschränkte Geräte gedachten Java-Plattformen CDC (vgl. 6.2.2) und CLDC (vgl. 6.2.2) möglich. Somit ist JMEDS für das Projekt geeignet. Die Erstellung von WSDL-Beschreibungen für Services ist unter JMEDS nicht erforderlich. Stattdessen werden die Dienste zur Laufzeit durch Methodenaufrufe definiert und daraus die WSDL-Beschreibung automatisch generiert. Zum Zeitpunkt des Auswahlprozesses und der Dauer der Projektgruppe ist JMEDS weiterentwickelt und die Dokumentation erweitert worden. Begründet durch das Auslaufen des OSAMI-Projektes bei der Firma MATERNA GmbH Information & Communication (siehe [ITE12]) ist es zum Zeitpunkt der Berichterstellung jedoch nicht abzusehen, ob eine zukünftige Weiterentwicklung stattfinden wird. Diese Information lag der Projektgruppe während des Auswahlprozeses noch nicht vor, weshalb sie in der anschließenden Bewertung der DPWS-Implementierungen in Abschnitt 6.4.3 keine Berücksichtigung findet.

Service-Oriented Architecture for Devices (SOA4D) Die *Service-Oriented Architecture for Devices* (SOA4D) ist in den Auswahlprozess nicht mit einbezogen worden, da sie den Authoren zu diesem Zeitpunkt noch nicht bekannt gewesen ist.

6.4.3 Bewertung der DPWS-Implementierungen

In diesem Abschnitt erfolgt eine Bewertung der DPWS-Implementierungen. Die Tabelle 6.14 zeigt eine Gegenüberstellung der in Abschnitt 6.4.2 vorgestellten DPWS-Frameworks anhand der Kriterien Programmiersprache, Einsatz auf ressourcenbeschränkten Geräten, Dokumentation, Projektpflege und Lizenz. Die Eignung für ressourcenbeschränkte Geräten ist mit einem '+'

Tabelle 6.14: Übersicht über die Eigenschaften der beschriebenen DPWS-Implementierungen

	Apache Axis 2	**WS4D-Axis2**	**WS4D-gSOAP**	**JMEDS**
Programmiersprache	Java	Java	C/C++	Java
Einsatz auf ressourcenbeschränkten Geräten	-	-	+	+
Dokumentation	+	-	+	+
Projektpflege	+	-	+	+
Lizenz	*Apache License 2*	*Apache License 2*	GPL/LGPL	*Eclipse Public License*

bewertet worden, wenn eine DPWS-Implementierung Rücksicht auf ressourcenbeschränkte Geräte nimmt und ein Einsatz der Bibliothek keinen großen Ressourceneinsatz erfordert. Andernfalls ist die Wertung '-' vergeben worden. Die Wertung der Dokumentation berücksichtigt den Umfang der Dokumentation sowie deren Nachvollziehbarkeit. Die Projektpflege ist mit einem '+' bewertet worden, wenn der Zeitpunkt der Veröffentlichung der letzten Version zum Bewertungszeitpunkt nicht länger als ein Jahr zurückgelegen hat.

Das *Apache Axis2* und WS4D-Axis2 Framework sind für den Einsatz auf ressourcenbeschränkten Geräten nicht geeignet, da für ihren Einsatz ein großer Ressourceneinsatz erforderlich ist. Deshalb sind beide für dieses Projekt nicht geeignet. Das WS4D-Axis2 Framework erweckt derzeit den Anschein nicht mehr gepflegt zu werden, da die letzte veröffentlichte Version auf den 12.11.2008 datiert ist (siehe [Uni11e]). Sowohl das WS4D-gSOAP-Framework als auch JMEDS erweisen sich beide als für dieses Projekt geeignet. Aufgrund der Präferenz für Java als Programmiersprache fällt die Entscheidung auf JMEDS als DPWS-Framework für dieses Projekt.

6.5 Verteiltes Rechnen

Ein essentieller Bestandteil einer Cloud ist die Möglichkeit, Berechnungen effizient über mehrere Rechner verteilen zu können. Da es bereits eine Vielzahl an Frameworks gibt, die verteiltes Rechnen ermöglichen, werden im Folgenden drei häufig verwendete Frameworks, denen verschiedene Ansätze zu Grunde liegen, getestet. Dabei handelt es sich um GridGain (siehe Abschnitt 6.5.1), Hadoop (siehe Abschnitt 6.5.2) und das Globus Toolkit (siehe Abschnitt 6.5.3).

6.5.1 GridGain

GridGain ist ein in Java entwickeltes Framework für Rechencluster, das das Verteilen von JVMs über eine oder mehrere Maschinen erlaubt. Es handelt sich um eine reine Softwareplattform.

GridGain ist damit nicht vergleichbar mit virtuellen Instanzen, die komplette Betriebssysteme ausführen können. Die einzelnen Instanzen (*GridGain Nodes*) kommunizieren dabei per Multicast und bringen keine eigene Programmlogik mit. Stattdessen wird die zum verteilten Rechnen benötigte Logik an einer Node in das System eingebracht, an die anderen Nodes verteilt und anschließend ausgeführt. Dies erlaubt auch das Verteilen und Ausführen externer, nicht Java-basierter Binärdateien. Zudem verwalten die Nodes selbstständig ihre Topologie und bringen Komfortfunktionen mit, die z. B. die Verwaltung von Arbeitsaufträgen, Lastverteilung oder Metriken zur Ressourcenverwaltung umfassen ([Gri05]).

6.5.2 Hadoop

Hadoop ist ebenfalls ein Java-basiertes Framework für verteiltes Rechnen, das ohne separate Instanzen, Aufgaben mit Hilfe von MapReduce verteilt. Hadoop wird von der *Apache Software Foundation* entwickelt und konstant vorangetrieben. Zudem bietet Hadoop ein eigenes, verteiltes Dateisystem, das in seiner Konzeption auf dem Google-Dateisystem (vorgestellt in [GGL03]) basiert. Da Hadoop nicht mit separaten Nodes arbeitet, muss das entsprechende Programm, das den MapReduce-Algorithmus implementiert, manuell in den Rechencluster eingebracht werden. Zudem schränkt die Nutzung von MapReduce das verteilte Rechnen auf Aufgaben ein, die in ein Ein- und Ausgabemuster für diesen Algorithmus überführt werden können (siehe [Apa11d]).

6.5.3 Globus Toolkit

Das Globus Toolkit ist eine Sammlung von Werkzeugen, die es ermöglichen, ein sicheres Grid zur Datenspeicherung und zum verteilten Rechnen zu betreiben. Kern dieser Lösung sind die Sicherheitsmodule, die ein fein granulares Rechte- und Verschlüsselungsmanagement auf Benutzerbasis bieten. Das Datengrid wird mittels GridFTP umgesetzt und setzt somit auf das weit verbreitete FTP-Protokoll als Zugangsmittel. Verteiltes Rechnen wird mit dem Modul GRAM5 implementiert. Innerhalb eines GRAM5-Grids kann alles berechnet werden, das auch auf den einzelnen Node des Grids berechenbar ist. Es stehen für verschiedene Programmiersprachen Bibliotheken (C, Java, Python, ...) zur Verfügung, um die Verteilbarkeit von Programmen auf einem GRAM5-Grid zu unterstützen. GRAM5 benötigt Gridnodes mit einem unixartigen Betriebssystem. Eine Bibliothek für Java-basierte Webservices umgeht diese Einschränkung und ist auf allen Java-SE-fähigen Rechnern ausführbar. Das Globus Toolkit ist modular aufgebaut, sodass einzelne Module weggelassen und ausgetauscht werden können. Dadurch ist es sehr flexibel einsetzbar, allerdings auch unübersichtlich. Es ist ohne umfangreiche Recherche schwierig, die Grenzen der Nutzungsmöglichkeiten des Globus Toolkits zu erkennen ([Uni11a]).

6.5.4 Vergleich der Frameworks

Zum Vergleich dieser drei Frameworks müssen softwaretechnisch relevante Aspekte, wie z. B. Funktionalität, Wartbarkeit oder Performanz untersucht werden. Im Folgenden werden die Analyseergebnisse der Recherche durch die Projektgruppe dargestellt und kurz zusammengefasst. Eine Übersicht der Ergebnisse ist in Tabelle 6.15 aufgelistet.

Die Aktivität der jeweiligen Entwickler- und Nutzergemeinden und die Aktualität des Projekts sind wichtige Gradmesser für die Zukunftssicherheit eines Frameworks. Ausserdem ist die

Tabelle 6.15: Analyse der Frameworks für verteiltes Rechnen

Kriterium	**GridGain**	**Hadoop**	**Globus Toolkit**
Aktivität der Community	+	+	+
Releaseaktualität	+	-	+
Dokumentation	+	+/-	-
API- / Benutzerfreundlichkeit	+/-	+	-
Mächtigkeit & Flexibilität	+	-	+
Ressourcenverbrauch	+/-	+/-	n.v.
Load-Balancing	+	-	n.v.
Betriebssystem-Unterstützung	Windows, Linux, OSX	Linux, Windows nur mit Cygwin	Linux, OSX, Windows nur mit Cygwin
Abhängigkeiten zu externen Bibliotheken	keine	Cygwin	Schwankt mit Modulauswahl

Dokumentation des Projekts ein essentieller Orientierungspunkt dafür, wie leicht Enwicklern der Einstieg in das Framework gemacht wird.
Für GridGain erscheinen regelmäßige Releases, News und es gibt eine Vielzahl aktueller Tutorials, die den Einstieg stark erleichtern. Die aktuelle Version 3.0.5 von GridGain ist im Januar 2011 erschienen. Dies zeigt sowohl, dass GridGain bereits über Jahre entwickelt wird, als auch, dass dieses Framework nach wie vor eine aktive Entwicklergemeinde hat. Die Dokumentation ist vollständig und übersichtlich. Neben einer *Javadoc*-Dokumentation ist ein *Documentation Center* vorhanden, mit kurzen und anschaulichen (Code-)Beispielen. Diese funktionieren zwar, auf Grund von Änderungen der Programmierschnittstellen, nicht immer fehlerfrei, sind aber hilfreich, um Antworten auf praktische Fragen und Probleme zu finden.
Für das Hadoop Framework sind hauptsächlich Mailinglisten zur Kommunikation vorhanden; diese zeigen jedoch eine hohe Aktivität und sind u.a. unterteilt in Entwickler- und Benutzer-Listen. Zusätzlich ist ein Wiki vorhanden, um Wissen über das Framework zu sammeln und bereitzustellen. Das Framework ist mit der aktuellen Version 0.21 jedoch in einem frühen Entwicklungsstadium, es sind somit Schnittstellenänderungen möglich.
Die API-Dokumentation ist als *Javadoc*-Dokumentation vorhanden. Weiterhin liegen Tutorials für den ersten Betrieb vor. Dies bedeutet, dass ein schneller Einstieg möglich ist. Einzelne Kapitel der Dokumentation beschäftigen sich mit Befehlen oder den nativen Bibliotheken, sind jedoch speziell zur Nutzung auf UNIX-Shells ausgelegt. Bessere Erklärungen sind häufig nur für vorherige Versionen verfügbar (die Dokumentation liegt hinter dem Entwicklungsstatus zurück). Die Community des Globus Toolkits ist aktiv und die gesamte Bandbreite einer Entwicklerwebseite ist vorhanden. Es findet trotz des hohen Alters des Toolkits eine kontinuierliche Weiterentwicklung statt. Die Dokumentation ist jedoch wenig strukturiert und schwierig verständlich. Teilweise wird dabei bei der aktuellsten Version auf vorherige Kapitel verwiesen, weil noch keine aktualisierte Dokumentation vorhanden ist.

Ein weiterer Gesichtspunkt zu Bewertung der Frameworks ist Benutzerfreundlichkeit der jeweiligen APIs. Dies ist für die Entwicklung von Software ein signifikanter Faktor, da eine

umständliche Syntax oder technische Einschränkungen die Entwicklung erschweren.
GridGain bietet die Nutzung von Annotationen, wodurch der Programmieraufwand reduziert werden kann. Da diese jedoch nicht von allen JVMs unterstützt werden, müssen eventuell Implementierungen ohne Annotationen genutzt werden, welche komplizierter sind und unter Umständen unübersichtlichere Codeabschnitte erfodern.
Hadoop hingegen bietet insgesamt übersichtliche und in sich schlüssige Implementierungsmöglichkeiten und erlaubt somit einen leichten Einstieg und effizientes Entwickeln.
Wie bereits auf Dokumentationsebene erkennbar ist, ist das Globus Toolkit ebenfalls im Hinblick auf seine API unübersichtlich. Dies kann zum Teil durch die Modularität erklärt werden, ist aber dennoch ein erkennbarer Nachteil.

Weitere funktionale Aspekte der Frameworks sind, neben dem Funktionsumfang, der Ressourcenverbrauch und die Möglichkeiten zur Lastverteilung. Insbesondere für die Erstellung einer Cloud-Plattform auf ressourcenbeschränkten Geräten, sind diese beiden Punkte wichtige Faktoren.
GridGain ist ein mächtiges Framework, bei dem es z. B. möglich ist, separate Nodes zu starten, die vorerst auf Aufgaben warten und beliebige Rechenaufgaben annehmen können. Berechnungen können unabhängig vom MapReduce-Paradigma (siehe [DG08]) ausgeführt werden, bzw. findet eine weichere Umsetzung des Paradigmas statt als z. B. bei Hadoop. Es wird ausdrücklich hervorgehoben, dass das Verteilen und Ausführen von nicht Java-Code möglich ist, dabei wird jedoch empfohlen diese Programme außerhalb der JVM auszuführen, um die Stabilität der Node nicht durch externe Fehler zu beeinflussen. Zudem hat GridGain im Test einen äußerst leistungsfähigen Eindruck hinterlassen; der GridGain-Nodes ohne Aufgabe benötigt nur 500 kB Arbeitsspeicher. Allerdings wird zusätzlich eine JVM gestartet, die entsprechend mehr Ressourcen verbraucht. Da FiLeCC bereits eine JVM startet, kann die GridGain-Node innerhalb dieser vorhandenen virtuellen Maschine gestartet werden. Die Kommunikation der Nodes erfolgt per Multicast-Nachrichten. Die Netzwerkauslastung, die von den Nachrichten erzeugt wird, ist im Rahmen von Tests kaum messbar. Für die Lastverteilung bietet GridGain eine *Loadbalancing API* mit bereits vorimplementierten Algorithmen an, sodass die Last gemäß gewählter Kriterien verteilt werden kann, wenn verteilte Algorithmen entwickeln werden. So ist beispielsweise eine Verteilung adaptiv zur Last, basierend auf Caching, RoundRobin oder gewichtete Randomisierung möglich.
Hadoop setzt im Gegensatz zu GridGain auf eine strikte Umsetzung des MapReduce-Schemas. Veränderungen im *MapReduce-Dictionary* oder Erweiterungen im verteilten Rechenschritt führten in Tests zu Fehlern beim Zusammenführen der Berechnungsergebnisse, da dies im Schema nicht vorgesehen ist. Im Hinblick auf seine Leistungsfähigkeit ist das Hadoop-Framework ebenfalls ressourcenschonend. Bei größeren Berechnungen sollte der Entwickler darauf achten, dass nicht die gesamte Eingabe in den Speicher geladen wird. Bei kleineren Berechnungen muss abgewägt werden, ob nicht eine nicht-verteilte Berechnung schneller zum Ziel führt. Eine Lastverteilung kann ebenfalls nur über das Format der Eingabe simuliert werden, da sich die Anzahl der einzelnen *Map-Schritte* nach dem Format der Eingabe richtet. Es wird jedoch keine echte Lastverteilung angeboten.
Das Globus Toolkit ist mit Abstand das umfangreichste Framework; die benötigten Module können einzeln zusammengestellt und bei Bedarf komplett ausgetauscht werden. Somit ist es sehr flexibel und für viele Aufgaben (u. a. Java Webservices) geeignet. Leider konnten aufgrund der Undurchsichtigkeit der API und Kompabilitätsproblemen keine sinnvollen Tests zum Ressourcenverbrauch oder Loadbalancing entwickelt werden.

Neben den funktionalen Aspekten bietet GridGain auch im Hinblick auf die Interoperabilität

Vorteile. Es werden alle gängigen Betriebssysteme unterstützt, ohne dass weitere Abhängigkeiten anfallen. Hadoop ist unter Windows nur mit der gleichzeitigen Nutzung von Cygwin ausführbar und auch im Globus Toolkit können abhängig vom verwendeten Modul Abhängigkeiten anfallen.

6.5.5 Fazit

GridGain hat sowohl in Bedienungsfreundlichkeit als auch im Hinblick auf Funktionalitäten Vorteile gegenüber den anderen beiden Frameworks. Das Globus Toolkit bietet vielversprechende Ansätze, ist in der Implementierung jedoch umständlich und auf Architekturebene eine Ebene zu niedrig für die Umsetzung von FiLeCC angesiedelt, da es auf die Virtualisierung auf Betriebssystemebene ausgelegt ist. Hadoop wirkte in den vorgenommenen Tests nicht ausgereift genug und in seinem Funktionsumfang gegenüber GridGain eingeschränkt.

Ein großer Vorteil sind die in GridGain verfügbaren Metriken zur Ressourcenverwaltung, sowie die *Loadbalancing API* zum Verteilen von Berechnungen. Es ist technisch möglich diese Komponenten auch für die Verteilung von Webservices innerhalb von FiLeCC zu nutzen, die Ergebnisse sind aber nicht optimal. Dennoch bietet das GridGain-Framework Möglichkeiten, sicher und effizient verteilt, z. B. innerhalb eines Webservices, zu rechnen und verteilte Algorithmen zu implementieren. Zudem bietet GridGain viele Freiheiten, wie diese Aufgaben ausgestaltet werden.

6.6 Verteilte Datenhaltung

Für eine Cloud-Plattform wie FiLeCC ist eine Möglichkeit Daten zu speichern grundlegend. Dies betrifft sowohl die Daten, die nur zur Laufzeit benötigt werden, wie z. B. Zustandsinformationen, als auch solche Daten, die persistent vorgehalten werden müssen, wie z. B. Konfigurationseinstellungen und Binärdateien. Für strukturierte Daten bietet sich der Einsatz von Datenbanken an. Dahingegen eignen sich für große zusammenhängende Datensätze, wie Binärdateien, Dateisysteme besser. Im Folgenden werden daher verteilte Open-Source-Lösungen für diese beiden Klassen von Datenhaltungssystemen evaluiert.

Die wichtigsten Innovationen aus dem in Abschnitt 4.6.4 beschriebenen Bereich DaaS, sind von Google und Amazon entwickelt worden. Vorreiter auf diesem Gebiet sind die NoSQL-Datenbanksysteme Dynamo und BigTable, welche nicht frei zur Verfügung stehen. Die Konzepte sind dennoch in wissenschaftlichen Artikeln, wie beispielsweise [DHJ+07] und [CDG+06], veröffentlicht. Daher existiert eine Vielzahl Open-Source-Lösungen, die auf diesen Konzepten basieren. Abschnitt 6.6.2 geht auf viele dieser Softwarepakete ein. Zuvor werden in Abschnitt 6.6.1 die Ergebnisse der Auswertung verschiedener verteilter Dateisysteme beschrieben. Die der Bewertung der Lösungen zugrundeliegenden Anforderungen sind in Kapitel 5 beschrieben. Abschnitt 5.4 enthält dabei die Anforderungen, die speziell an die verteilte Datenhaltung gestellt werden.

6.6.1 Verteilte Dateisysteme

Auf dem Gebiet der verteilten Dateisysteme existieren Systeme, welche für Cluster und nicht für Cloud-Umgebungen konzipiert sind. Die Systeme GlusterFS und Lustre werden im Laufe dieses

Abschnitts genauer beschrieben. Daneben gibt es mit HDFS ein speziell auf Cloud-Umgebungen ausgelegtes verteiltes Dateisystem, welches ebenfalls betrachtet wird.

Apache Hadoop/HDFS Zu dem in Kapitel 6.5.2 beschriebenen Apache-Hadoop-Projekt gehört das in Java implementierte, verteilte Dateisystem Hadoop File System kurz HDFS, welches unabhängig von anderen Hadoop-Komponenten einsetzbar ist. Eine Übersicht über die Funktionsweise von HDFS bietet [Apa10]. Die Architektur von HDFS ist nicht beliebig verteilt und besteht aus einem Netzwerk aus genau einer *NameNode* und vielen *DataNodes*. Die *NameNode* verwaltet die Metadaten des Dateisystems, welche vollständig und aktuell im Hauptspeicher gehalten werden. Weiterhin speichert die *NameNode* den Startzustand der Metadaten in einem sogenannten *FsImage* und alle Änderungen im sogenannten *EditLog*. Da die Metadaten nur von der *NameNode* vorgehalten werden, beginnen alle Anfragen an das Dateisystem mit einer Verbindung an die *NameNode*, um mittels der Metadaten zu bestimmen, auf welcher *DataNode* sich die Daten befinden. Die eigentlichen I/O-Operationen werden direkt auf den *DataNodes* ausgeführt. Da nicht mehr als eine *NameNode* integriert werden kann, ist sie eine Schwachstelle bzgl. der Ausfallsicherheit des Systems.

Um die Verfügbarkeit des Systems zu erhöhen, kann eine *BackupNode* eingerichtet werden. Diese enthält eine Kopie der Metadaten, die kontinuierlich mit der *NameNode* synchronisiert wird. Außerdem führt die *BackupNode* periodische Checkpoints durch, bei denen aus dem aktuellen Zustand der Metadaten ein aktuelles *FsImage* generiert wird. Dadurch kann der Umfang des *EditLog* der *NameNode* klein gehalten werden. Allerdings besteht nur die Möglichkeit eine einzige BackupNode zu betreiben, wodurch die Verfügbarkeit des Systems nur geringfügig verbessert werden kann. Das Durchführen der Checkpoints ist auch ohne die vollständige Backup-Funktion mit beliebig vielen *CheckpointNodes* möglich, solange keine *BackupNode* in das System integriert ist.

HDFS sieht die beschriebenen Mechanismen vor, um Knotenausfälle auszugleichen, kann aber nicht vollständig verteilt betrieben werden. Da die Zuverlässigkeit des Gesamtsystems immer von der Zuverlässigkeit von *NameNode* und *BackupNode* abhängt, ist HDFS nicht für den Einsatz in FiLeCC geeignet.

GlusterFS GlusterFS ist ein GPL-lizenziertes Dateisystem, welches über ein FUSE-Kernel-Modul eingebunden wird. FUSE (*Filesystem in Userspace*) ist ein im Linux-Kernel verfügbares Modul, das Benutzern ohne Administrationsrechte ermöglicht, Dateisysteme einzubinden. GlusterFS unterstützt unter anderen folgende Betriebsvarianten [RH12].

Standalone Storage: Ähnlich einer Netzwerk-Freigabe wird der Inhalt des Dateisystems über ein Netzwerk zur Verfügung gestellt.

Distributed Storage: Die Dateien werden auf mehrere Servern verteilt. Eine Datei ist aber immer komplett auf einem Server gespeichert.

Replicated Storage: Die Dateien werden auf mehrere Server repliziert. Dabei wird die Größe des Dateisystems durch die kleinste Speicherkapazität eines Servers bestimmt.

Striped Storage: Die Dateien werden in kleinen Teilen auf alle Server verteilt. Dadurch wird die Geschwindigkeit des gesamten Dateisystem erhöht, da eine Datei von mehreren Servern gleichzeitig gelesen wird.

Variantenkombination: Die oben genannten Speichervarianten können kombiniert werden. Bei einem *Distributed and Replicated Storage* werden die Dateien beispielsweise sowohl auf mehrere Server verteilt, als auch repliziert.

Jeder Server, der Speicher zur Verfügung stellen möchte, bietet diesen über einen Dienst an. Die Konfiguration der Struktur des verteilten Dateisystems wird in Konfigurationsdateien auf dem Client vorgenommen. Diese Struktur muss auf allen Clients gleich sein, da die zur Verfügung gestellten Speicherkapazitäten ansonsten unterschiedlich genutzt werden. Die Verteilung der Daten übernimmt nicht ein Server, sondern jeder Client verteilt seine Daten eigenständig auf alle verfügbaren Server. Dadurch ist bei unterschiedlichen Konfigurationen auf den Clients ein Datenverlust sehr wahrscheinlich, da die Blöcke des Dateisystems unterschiedlich genutzt würden. Das GlusterFS-Dateisystem kann dann wie andere lokale Dateisysteme eingebunden und genutzt werden. Wird ein Server dauerhaft aus dem Netzwerk entfernt, muss diese Änderung in der Konfigurationsdatei jedes Clients durchgeführt werden und das Dateisystem neu gestartet werden. GlusterFS ist daher für ein festes Netzwerk aus Servern zu empfehlen. Aufgrund der fehlenden Flexibilität ist GlusterFS für FiLeCC nicht geeignet.

Lustre Lustre ist ein verteiltes Dateisystem, das eine hohe Performance durch eine Verteilung der Daten auf tausende Knoten bietet. Daraus resultiert eine sehr hohe Speicherkapazität. Im Dezember 2010 hat Oracle, der Entwickler von Lustre, angekündigt, die Entwicklung einzustellen. Das Projekt ist aber weiterhin als Open-Source-Lösung nutzbar. Für FiLeCC ist Lustre daher wegen schlecht vorhersagbarer Weiterentwicklung und Support nicht interessant.

6.6.2 NoSQL-Datenbanken

Wie in der Einleitung dieses Abschnitts erwähnt, basieren die meisten hier vorgestellten quelloffenen NoSQL-Datenbanken für Cloud-Umgebungen auf Techniken der beiden frühen Anbieter großer Public Clouds Amazon und Google. Diese Techniken sowie die verschiedenen Klassen von NoSQL Datenbanken werden in Abschnitt 4.6.4 beschrieben. Im Folgenden findet sich die Auswertung verschiedener Open-Source-Implementierungen bezüglich der Einsetzbarkeit in FiLeCC.

JBoss Infinispan JBoss Infinispan bietet einen über ein Grid verteilten virtuellen Cache-Speicher, der insbesondere für Javaprojekte geeignet ist. Dazu implementiert es das `Cache` Interface, das wiederum auf der Schnittstelle `java.util.Map` basiert. Infinispan gehört somit zur Klasse der KV-Stores wie in Abschnitt 4.6.4 beschrieben. Im folgenden werden die Eigenschaften von Infinispan [Mui11] näher erläutert.

Zur Erhöhung der Datensicherheit können die Daten über beliebig viele Knoten repliziert werden. Infinispan skaliert in solchen verteilten Umgebungen sehr gut, da jegliche Kommunikation, außer der, die zum Auffinden neuer Knoten notwendig ist, auf Peer-to-Peer-Basis abläuft. Weiterhin kann eine Vergrößerung oder Verkleinerung des Clusters um eine beliebige Anzahl Rechner im laufenden Betrieb stattfinden und erfordert keinen Administrationsaufwand. Neue Knoten, auf denen Infinispan ausgeführt wird, werden automatisch gefunden und in den Cluster integriert.

Um die Konsistenz der Daten bei parallelem Zugriff gewährleisten zu können, unterstützt Infinispan Transaktionen und Zugriffssperren. Zur Verwaltung der Transaktionen ist ein Transaktionsmanager notwendig. Da kein Transaktionsmanager für verteilte Umgebungen in Infinispan integriert ist, muss zusätzlich ein zur *Java Transaction API* konformer Transaktionsmanager betrieben werden, falls die Unterstützung für Transaktionen notwendig ist. Dazu kann Infinispan in einer JavaEE-Umgebung betrieben werden, die einen Transaktionsmanager beinhaltet. Alternativ kann ein eigenständiger verteilter Transaktionsmanager eingerichtet und an Infinispan gekoppelt werden. Dies kann beispielsweise JBossTS sein. Beide Varianten sind wenig geeignet für die Verwendung in FiLeCC, da JavaEE allgemein nicht für ressourcenschwache Systeme geeignet ist (siehe Abschnitt 6.2.2) und auch eigenständige verteilte Transaktionsmanager häufig auf komplexer Middleware wie CORBA aufsetzen. Eine zu Infinispan kompatible, aber dennoch ressourcenschonende Lösung ist nicht verfügbar.

Infinispan unterstützt weiterhin das persistente Speichern der Daten im Cache über eine spezielle `CacheStore`-Schnittstelle mit unterschiedlichen Möglichkeiten zur Datenspeicherung. Dazu gehören die Unterstützung von Datenbanken per JDBC und das Speichern im Dateisystem. Das persistente Speichern kann dabei zu verschiedenen Zwecken genutzt werden, wie dem „Warmstarten“ mit bereits gefüllten Caches, dem Schützen der Daten vor Verlust durch vollständige Grid-Neustarts oder -Ausfälle, sowie als Überlaufspeicher, für den Fall, dass der Hauptspeicher aller Knoten bereits gefüllt ist.

Die Konfiguration von Infinispan ist sowohl durch eine XML-Datei, als auch zur Laufzeit durch Methodenaufrufe möglich. Diese Eigenschaft, die Konfiguration zur Laufzeit ändern zu können in Kombination mit der hohen Anzahl sehr feingranularer und tiefgreifender Einstellungsmöglichkeiten, unterscheidet Infinispan von Konkurrenzprodukten, wie dem im nächsten Abschnitt beschriebenen Hazelcast. Ein weiterer Nachteil, zusätzlich zu der für FiLeCC ungeeigneten Transaktionsimplementierung, ist die momentan fehlende Unterstützung für Datenbankabfragen nach dem MapReduce-Schema. Diese wird erst ab Version 5.0.0 enthalten sein [Sur10b], welche zum Zeitpunkt der Recherche kurz vor der Veröffentlichung steht.

Zusammengefasst hängt die Einsetzbarkeit von Infinispan für FiLeCC davon ab, ob parallele Schreibzugriffe auf die Daten erfolgen, welche einen anforderungsgerechten Lockingmechnismus (siehe 5.4.2) erfordern. Da ein solcher Mechanismus, durch das Fehlen eines geeigneten und kompatiblen Transaktionsmanagers, nicht vorhanden ist, ist Infinispan nur dann ein Kandidat für den Einsatz in FiLeCC, wenn parallele Zugriffe ausgeschlossen werden können.

CouchDB CouchDB ist eine dokumentenorientierte Datenbank (vgl. Kapitel 4.6.4) [Apa11c]. Im Gegensatz zu relationalen Datenbanken werden die Daten in einer CouchDB-Datenbank in Dokumenten gespeichert. Diese Dokumente liegen im JSON-Format (*JavaScript Object Notation*) vor [Int06]. Das JSON-Format ist ein kompaktes, für Menschen lesbares und zur Datenübertragung entwickeltes Format. Ein Dokument kann mit einem Anhang versehen werden, wodurch Binärdateien gespeichert werden können. Innerhalb der Datenbank wird JavaScript als Steuersprache verwendet. Mittels dieser Skriptsprache können beispielsweise Filter auf eine Menge von Dokumenten angewendet werden.

CouchDB ist speziell für die verteilte Speicherung von Dokumenten entwickelt. So kann eine Datenbank auf einen anderen Server repliziert und die Aktualisierungen der Datenbanken synchronisiert werden. Es wird bei der Replizierung mit einer zweiten Datenbank eine komplette

Kopie der lokalen Datenbank angelegt. Diese Synchronisation kann manuell oder kontinuierlich von der Datenbank ausgeführt werden. Die Synchronisation kann Konflikte hervorrufen, wenn parallele Änderungen an einem gespeicherten Dokument vorgenommen werden. Konflikte werden nicht automatisch aufgelöst, sondern müssen manuell behandelt werden. Die Wahl der Datenbanken, mit denen die lokale Datenbank synchronisiert werden soll, kann frei getroffen werden. Sollen alle Knoten miteinander synchronisiert werden, wie es aufgrund der Forderung nach höchster Verfügbarkeit in FiLeCC der Fall ist, muss für jeden Knoten die Replikation mit allen anderen Knoten eingerichtet werden. In einer dynamischen Umgebung erfordert die Verwaltung der aktuellen Topologie und der notwendigen Replikationsverbindungen einen großen Aufwand, da beides kontinuierlich angepasst werden muss.

Ansonsten ist der Konfigurationsaufwand für eine CouchDB-Infrastruktur gering und die Konfigurationsmöglichkeiten sind flexibel. Einige für FiLeCC notwendige Funktionen fehlen allerdings und müssen selbst implementiert werden. Eine Lastverteilung der Datenbankzugriffe kann z. B. nur durch eine HTTP-Proxysoftware erreicht werden. Zu den Nachteilen gehört zusätzlich die lediglich inoffizielle Unterstützung für das Betriebssystem Windows.

MongoDB ist ebenfalls eine dokumentenorientierte Datenbank [10g11b]. Die Dokumente werden ebenfalls im JSON-Format gespeichert. Replika der Datenbank können nur in einer nicht rekursiven Master/Slave-Struktur erstellt werden. In dieser Struktur kann nur der Master schreibend auf die Datenbank zugreifen. Die Slaves werden dazu verwendet, die lesenden Zugriffe zu verteilen. Die Master/Slave-Struktur kann zusätzlich als Replika-Set definiert werden, welches aus einem Master und einem oder mehreren Slaves besteht, wodurch die Möglichkeit besteht, im Falle eines Ausfalls des Masters, einen Slave zum neuen Master zu wählen.

MongoDB bietet darüber hinaus die Möglichkeit, die Datenbank in sogenannte *Shards* aufzuteilen [10g11a]. Ein *Shard* besteht dabei aus einem Replika-Set, also einer Master/Slave-Struktur, welche die Verfügbarkeit der Daten sicherstellt. In Abbildung 6.7 sind im oberen Teil vier Shards zu sehen. Diese bestehen in der dargestellten Konfiguration aus je drei Servern, die ein Replika-Set bilden. Im Gegensatz zu CouchDB werden so nicht alle Daten auf jedem Knoten bereit gehalten, sondern gleichverteilt auf den Shards gespeichert. Dadurch wird eine Parallelisierung der schreibenden Datenbankzugriffe ermöglicht, da in jedem Shard ein Master verfügbar ist. Außerdem wird die gesamte Speicherkapazität der Datenbank erhöht, da die Daten auf die Shards verteilt werden. Weiterhin ist ein Konfigurationsserver (und dessen Backup(s)) notwendig, wenn eine solche Shard-basierte Struktur eingesetzt wird, welcher die Metadaten der Shards speichert. Die Konfigurationsserver sind in Abbildung 6.7 auf der linken Seite dargestellt. Nach außen ist diese Struktur wie ein einzelner Server sichtbar. Die Nutzung auf den Clients erfolgt dabei über einen Dienst, der als eine Schnittstelle zur Datenbank agiert. Dieser Dienst ist an der unteren Seite von Abbildung 6.7 über dem Client zu sehen. Der Konfigurationsaufwand ist jedoch durch die unflexible Struktur, welche fest in Konfigurationsdateien angegeben werden muss, als vergleichsweise hoch einzuschätzen.

Cassandra Das Apache Projekt Cassandra hat sich zum Ziel gesetzt, die Vorteile von hoch verteilten Systemen mit *Eventual Consistency* (siehe Abschnitt 4.6.4) mit den Vorteilen strukturierter Systemen zu kombinieren. Ursprünglich ist Cassandra ein internes Projekt von Facebook, das seit 2008 Open-Source-Projekt bei der Apache Software Foundation ist. Die Cassandra zugrunde liegenden Techniken sind in [LM10] beschrieben. Cassandra nutzt das *Eventual Consistency*

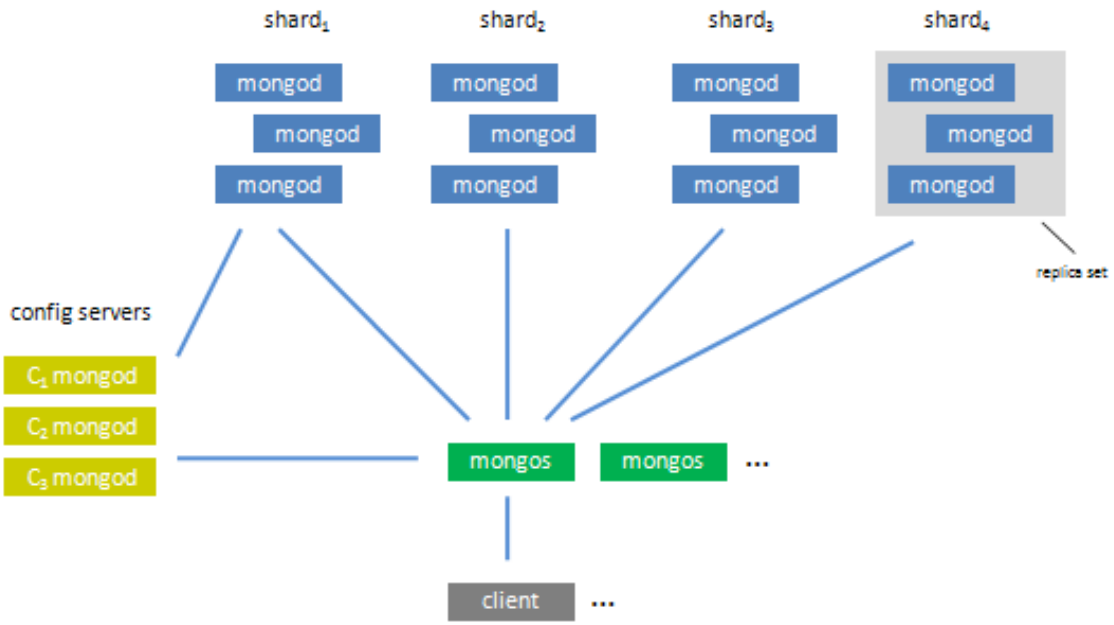


Abbildung 6.7: MongoDB-Struktur mit Shards. Quelle: [10g11a]

Modell und *Consistent Hashing* (siehe Abschnitt 4.6.4), strukturiert die Daten aber in schwach besetzten Tabellen anhand von Spalten.

Ein wesentlicher Nachteil von Cassandra ist der hohe Speicherbedarf. Die Entwickler des Projektes empfehlen eine minimale RAM-Größe von 4 GB pro Knoten [Apa11b]. Daran wird deutlich, dass Cassandra ausdrücklich für Enterpriseanwendungen ausgelegt ist. Mit den begrenzten Ressourcen der FiLeCC-Zielplattform ist es also nicht sinnvoll Cassandra zu betreiben. Aus diesem Grund wird diese Datenbank nicht weiter getestet und es wird nach effizienteren Lösungen gesucht.

Apache HBase HBase ist eine verteilte Datenbank mit einem Ansatz, der sich zwischen dem KV-Modell und relationalen Datenbanken einordnet. Vorbild in technologischer Hinsicht ist Googles BigTable. Die Daten werden über einen Schlüssel identifiziert und enthalten Daten zu Spalten, die sich zwischen den Datensätzen unterscheiden dürfen. HBase ist daher eine Option für den Ersatz von relationalen Datenbanken, wenn nur eine schwache Strukturierung der Daten nötig ist, auf SQL verzichtet werden kann, aber Datenmengen zu bewältigen sind, für die Cluster einer Größe notwendig sind, bei der RDBS nicht mehr skalieren.

Da HBase auf Hadoop aufbaut und weiterhin stark auf Konsistenz und gleichzeitig hohe Geschwindigkeit ausgerichtet ist, ist die Architektur nicht vollständig verteilt. Die Hadoop *NameNode* und der HBase *Master* Konten sind beide *Single Points of Failure* [Ein10]. Weiterhin hat HBase nach Aussage von Experten Probleme mit *Rolling Restarts*, bei denen ein Knoten nach dem anderen neu gestartet wird, während der übrige Cluster weiter in Betrieb ist [Ein10]. Aus diesem Grund muss der Cluster nach Konfigurationsänderungen vollständig heruntergefahren und dann neu gestartet werden [Ein10]. Für die Änderung der Konfiguration von *NameNode* und

HBase Master sind ohnehin Wartungsfenster nötig, in denen die Datenbank nicht verfügbar ist. Dennoch ist HBase das weitentwickeltste Open-Source-Projekt seiner Klasse. Für FiLeCC ist HBase aufgrund seiner nicht vollständig verteilten Architektur dennoch nicht geeignet.

Riak Riak ist ein weiteres auf den Technologien von Amazons Dynamo basierendes Open-Source-Projekt mit dem Ziel eine NoSQL-Datenbank für das vollständig verteilte, persistente und hoch ausfallsichere Speichern von Daten in einem Grid zu entwickeln. Implementiert ist Riak hauptsächlich in den Programmiersprachen Erlang und C. Weiterhin wird eine Verarbeitung der gespeicherten Daten nach dem MapReduce-Schema unterstützt. Eine Übersicht über die Funktionen von Riak findet sich in [Bas12b].

Es existieren Riak-Clients für diverse Programmiersprachen [Bas12a]. Für Java ist ein Client verfügbar, der die MapReduce-Fähigkeiten von Riak unterstützt. Die eigentlichen MapReduce-Skripte müssen allerdings in Erlang oder JavaScript implementiert sein, wobei mit JavaScript lediglich Lesezugriffe auf die Daten möglich sind.

Die Datenbank ist für den Einsatz in FiLeCC nicht geeignet, weil das Windows-Betriebssystem und die ARM-Plattform nicht unterstützt werden. Dadurch ist die geforderte Plattformunabhängigkeit in zwei wichtigen Bereichen nicht erfüllt.

Voldemort Voldemort basiert wie Riak auf den technischen Grundlagen von Amazon Dynamo [DHJ+07], die es ermöglichen große Datenmengen als Schlüssel/Wert-Paare vollständig verteilt in einem Grid zu speichern. Die grundlegende Funktionsweise wird auf der Projektwebsite [Pro12] vorgestellt. Voldemort ist vollständig auf Java ausgelegt und in Java implementiert. Andere Clients stehen nicht zur Verfügung. Weiterhin verfügt Voldemort über ein automatisches Caching, um Zugriffe zu beschleunigen. Die Konfiguration des Clusters kann allerdings nur statisch erfolgen. Vor dem Starten der Datenbank müssen alle Knoten bekannt sein. In FiLeCC ist dies nicht der Fall, weswegen Voldemort für dieses Projekt nicht eingesetzt werden kann.

Hypertable Hypertable ist ein weiteres Projekt, das eine zeilenorientierte Datenbank (vgl. Kapitel 4.6.4) implementiert. Auch hier sind die Daten in schwach besetzten Tabellen gespeichert und durch Schlüssel identifizierbar.

Mit Hypertable wird über eine eigene Abfragesprache namens HQL kommuniziert, die an SQL angelehnt ist. Zur Speicherung der Daten im Grid benutzt Hypertable verteilte Dateisysteme, wie HDFS, welches in Abschnitt 6.6.1 beschrieben wird. Auch verteiltes Rechnen auf den Daten ist durch Unterstützung von Apache Hadoop in Form seiner MapReduce-Techniken möglich. Außerdem gibt es die Möglichkeit Map- und Reduce-Skripte, die in anderen Sprachen geschrieben sind, mit Hadoops Streamingtechniken auf die Daten anzuwenden. Eine Übersicht über die Funktionsweise aus Anwendersicht und den Funktionsumfang bietet [Hyp11].

Hypertable setzt auf verteilten Dateisystemen wie HDFS auf. Die in Abschnitt 6.6.1 untersuchten Dateisysteme werden aber alle nicht den Anforderungen gerecht. Gefordert sind vollständig verteilte Ausführung und höchste Verfügbarkeit, bei gleichzeitig automatischer Erweiterung und Verkleinerung um hinzukommende und wegfallende Knoten. Aus diesem Grund ist Hypertable nicht für den Einsatz in FiLeCC geeignet.

Hazelcast Hazelcast ist ein quelloffener KV-Store, welcher eine verteilte Version der Java-Schnittstellen `java.util.{Queue, Set, List, Map}` implementiert [Haz11]. Die Software ist klein (< 2 MB) und einfach zu benutzen, wobei nur eine JAR-Datei einzubinden ist.

Die Konfiguration erfolgt über eine XML-Datei. Jede Instanz der von Hazelcast unterstützten Datenstrukturen kann individuell konfiguriert werden. Beispielsweise kann ein Replizierungsfaktor *n* eingestellt werden, um die Ausfalltoleranz des Systems zu erhöhen. Dieser gibt an, wie viele Backup-Kopien der Datenstruktur existieren sollen. Im Falle eines Knotenausfalls, wird sichergestellt, dass der Replizierungsfaktor für die betroffenen Daten wieder hergestellt wird. In der Originalsoftware ist *n* auf einen Wert von maximal 3 beschränkt. Wenn ein größerer Replizierungfaktor benötigt wird, z. B. für die Garantie, dass Daten auf allen Knoten vorliegen, lässt sich dieser durch eine Änderung im Quellcode erhöhen. Um Hazelcast als Cache zu benutzen, kann für jede Datenstruktur eine Maximalanzahl an Elementen bestimmt werden. Die überzähligen Elemente werden nach einem vorgegebenen Verfahren verdrängt. Das Gleiche ist für veraltete Elemente möglich, welche eine einstellbare Zeitgrenze haben, die abhängig von der Erstellungs- oder Zugriffszeit ist.

Hazelcast unterstützt dynamische Clusterbildung durch Multicast-Nachrichten. Bei dem Neustart eines Knotens, kommuniziert dieser mit den anderen und wird automatisch in die Datenverteilung einbezogen. Die Verteilung der Daten wird durch eine Hashfunktion gesteuert, die sicherstellt, dass alle Knoten gleich ausgelastet sind.

Weiterhin wird Persistenz unterstützt, indem der Entwickler eine beliebige Implementierung der `MapStore`-Klasse zur Verfügung stellt. Darüber hinaus, wird eine einfache, SQL-ähnliche Schnittstelle zur Datendurchsuchung und Filterung angeboten. Als weitere Zusatzfunktion, werden cloudübergreifende Ereignisse unterstützt, mit einer API für das Abonnieren und Verschicken von Nachrichten. Auch einfache Transaktionen sind mit Hazelcast möglich, die es erlauben, eine Folge von Schritten atomar auszuführen. Verteiltes Rechnen ist ebenfalls verfügbar, jedoch sind hier die Möglichkeiten wesentlich eingeschränkter als bei dem in Kapitel 6.5.1 beschriebenen GridGain.

Ein Nachteil von Hazelcast ist die ausschließlich statische Konfigurierbarkeit der Datenstrukturen. Eine dynamische Änderung der Datenstrukturkonfigurationen wird nicht unterstützt. Durch den Einsatz von Wildcards in den Datenstrukturnamen bei der Konfiguration besteht jedoch die Möglichkeit verschiedene Gruppen von Datenstrukturinstanzen zu konfigurieren und zur Laufzeit zu erzeugen. Ein weiterer Nachteil von Hazelcast ergibt sich aus der Annahme, dass alle Knoten äquivalent sind und die gleichen Speichergrößen haben. In einem Netz heterogener Knoten erschwert dies eine ressourcenbezogene Last- und Datenverteilung.

GridGain Die Software GridGain (vgl. Kapitel 6.5.1) ist in erster Linie für verteiltes Rechnen konzipiert. Darüber hinaus wird eine Data-Grid-Infrastruktur angeboten. Diese Infrastruktur erlaubt es dem GridGain-Rechencluster als verteilte Datenbank zu funktionieren. Es handelt sich um eine KV-Datenbank, bei der beliebige Java-Objekte als Schlüssel-Wert-Paare gespeichert werden können.

Die Datenbank ist durch die eingesetzte Java-Technologie plattformunabhängig. Sie bietet eine Multicast-basierte automatische Clusterbildung und -replikation. Die Konfigurationsmöglichkeiten sind sehr umfangreich. Beispielsweise kann der Replizierungfaktor, bis hin zu einer

Verteilung von jedem Datum auf jeden Knoten, frei gewählt werden. Allerdings ist diese Konfiguration statisch anzugeben und zur Laufzeit nicht anpassbar. Darüber hinaus kann jeder Aspekt des Data-Grid-Verhaltens durch eine Neuimplementierung der zuständigen Schnittstellen beeinflusst werden.

Die Datenhaltung besteht aus beliebig vielen verschiedenen Cache-Datenstrukturen, die unabhängig voneinander konfiguriert werden können. Die Schreibzugriffe auf jeden Cache werden von GridGain automatisch serialisiert. Dadurch wird die Konsistenz der Daten bei parallelen Schreibzugriffen sichergestellt. Weiterhin besteht die Möglichkeit, die Daten mittels verschiedener Backends, wie z. B. JDBC [Gri12b] oder Hibernate [Gri12a], persistent zu speichern.

Die GridGain Datenbank ist anhand ihres Leistungsumfangs gut für FiLeCC geeignet. Weil die Software ohnehin auf allen Knoten gespeichert ist, um verteiltes Rechnen in FiLeCC zu ermöglichen, ist ihr Einsatz ressourceneffizient, da keinezusätzliche Software installiert werden muss.

6.6.3 Fazit

Auf dem Gebiet der verteilten Dateisysteme ist keine der betrachteten Lösung in der Lage, den Anforderungen von FiLeCC gerecht zu werden. Alle Lösungen sind nicht vollständig verteilt, oder nur statisch konfigurierbar. Für den Einsatz in FiLeCC ist allerdings eine Lösung mit höchster Ausfallsicherheit und dynamischer Skalierbarkeit notwendig. Idealerweise soll ein solches Dateisystem weiterhin eine von der Topologie der Anlage abhängige Verteilung der Kopien jeder Datei erlauben, um Ausfälle großer zusammenhängender Anlagenteile kompensieren zu können. Da keine fertige Lösung existiert, die den Anforderungen gerecht wird, wird für das persistente und ausfallsichere Speichern von Binärdateien in FiLeCC das in Abschnitt 7.2 beschriebene *Software Deployment and Runtime Management* (kurz SDRM) eingesetzt. Es handelt sich dabei um eine Eigenentwicklung für FiLeCC, bei der die Binärdateien mittels Betriebssystemschnittstellen in dem vorhandenen Dateisystem auf den Knoten gespeichert werden. Die Informationen darüber, welche Binärdatei auf welchem Knoten gespeichert ist, wird in einer verteilten Datenbank gespeichert. Der Algorithmus zur Berechnung einer Verteilung von Sicherheitskopien auf Knoten ist austauschbar, so dass eine Verteilung z. B. topologiebasiert möglich ist.

Im Bereich der verteilten NoSQL-Datenbanken müssen einige der betrachteten Systeme ausgeschlossen werden, da sie nicht vollständig verteilt arbeiten und somit einen oder mehrere *Single Points of Failure* aufweisen. Dies ist in einigen Fällen dadurch begründet, dass sie auf verteilten Dateisystemen aufbauen, welche durchweg nicht für FiLeCC geeignet sind. Andere Datenbanksysteme sind für Clouds bestehend aus Clustern von leistungsstarken Servern ausgelegt und belegen bereits im Ruhezustand zu viele Ressourcen für den Einsatz auf ressourcenbeschränkten Geräten. Ein weiteres Kriterium, das zum Ausschluss verschiedener Datenbanksysteme führt, ist die geforderte Platformunabhängigkeit. Lediglich die Datenbanksysteme Hazelcast, Infinispan und GridGain erfüllen alle gestellten Anforderungen.

Aus den in Abschnitt 5.4 aufgezeigten Anforderungen an die Datenhaltung folgt, dass ein einzelnes Datenbanksystem nicht gleichzeitig den Anforderungen durch die SDRM- und Lastdaten gerecht werden kann. Die Daten des SDRM müssen bei parallelem Zugriff konsistent bleiben und dürfen in keinem Fall bei Ausfall von Knoten verloren gehen. Das macht eine Verteilung auf alle Knoten des Systems erforderlich. Die Lastdaten hingegen weisen eine regelmäßige Schreibfrequenz auf, müssen aber nicht vollständig vor Verlust geschützt werden. Wichtiger ist

hier zu vermeiden, dass permanent alle Knoten des Systems Ressourcen aufwenden müssen, um die Lastdaten vorzuhalten. Die Lastdaten werden also idealer Weise in einem verteilten Cache vorgehalten, der nur auf den leistungsfähigsten Knoten des Systems betrieben wird und dessen Instanzanzahl mit der Größe des Systems skaliert. Minimaler Ressourcenverbrauch durch die Lastdatenbank ist notwendig, um die Geschwindigkeitsvorteile zu erreichen, die durch das Sammeln der Lastdaten in einem Cache ermöglicht werden.

Für die Realisierung der SDRM-Datenbank ist lediglich GridGain geeignet, da mit Hazelcast und Infinispan die Konsistenz der Daten nicht gewährleistet werden kann. GridGain in FiLeCC als Datenbank zu nutzen hat weiterhin den Vorteil, dass ein Grid ohnehin betrieben wird, um der in FiLeCC betriebenen Software eine Möglichkeit für verteiltes Rechnen anzubieten. Somit kann auf die Verwendung weiterer Fremd-Software verzichtet werden. Aus diesen Gründen und aufgrund der hohen Flexibilität der GridGain-Datenbank wird diese in FiLeCC auch für die Daten externer Software genutzt.

Da für die Lastdaten eine zweite Datenbank betrieben werden muss, um sie unabhängig von der Datenbank für die Daten des SDRM und die der externen Software skalieren zu können, wird für diesen Zweck Hazelcast ausgewählt. Dieses Datenbanksystem erfüllt alle Anforderungen wie automatische Clusterbildung und ist gleichzeitig sehr kompakt und ressourcenschonend. Außerdem ist Hazelcast leistungsfähiger als GridGain, weil das Paradigma der *Eventual Consistency* verwendet wird. Infinispan eignet sich ebenfalls weniger als Hazelcast für den Einsatz als Lastdaten-Cache in FiLeCC, da es deutlich mehr Ressourcen belegt und die identifizierten Stärken wie die Konfiguration zur Laufzeit für diese Anwendung nicht notwendig sind.

Die Implementierung der verteilten Datenhaltung, wird in Abschnitt 7.9 ausführlich beschrieben.

6.7 Fazit Technologieauswahl

Tabelle 6.16 zeigt abschließend eine Übersicht, welche Technologien für die Umsetzung von FiLeCC verwendet werden. Die Plattformen werden vom Lehrstuhl für Förder- und Lagerwesen der TU Dortmund gestellt, während die anderen Technologien auf Grund der durchgeführten Evaluation bereits getestet und einsatzbereit sind.

Tabelle 6.16: Zusammenfassung der ausgewählten Technologien

Aufgabe	**Ausgewählte Technologie**
Hardwareplattform	PandaBoard, IPCs
Betriebssystem	Debian Squeeze
Java-Implementierung	Oracle Java Embedded
Cloud Architektur	PaaS / private Cloud
Webservice Kommunikationsprotokoll	DPWS
Webservice Referenzimplementierung	JMEDS
Verteiltes Rechnen	GridGain
Verteilte Datenhaltung	Hazelcast, GridGain
Binärdatenhaltung	eigene Lösung (SDRM)

Als Laufzeitumgebung wird für FiLeCC die Linux-Distribution Debian Squeeze in Kombination mit Java Embedded verwendet. Diese wird auf Hardwareplattformen wie dem Pandaboard

oder den vorgstellten IPCs (siehe Kapitel 6.1) installiert und konfiguriert. Als Architektur für die angebotene *private Cloud* wird *PaaS* verwendet. Auf dieser Plattform kann Software als Webservices ausgeführt werden. Die Webservices müssen sich an die *DPWS*-Spezifikation halten. Zusätzlich haben Webservices die Möglichkeit, die Rechenkapazität der Cloud durch verteiltes Rechnen auszunutzen. Dazu wird eine Anbindung an die Software *GridGain* zur Verfügung gestellt. Zur Datenverwaltung stehen den Webservices Schnittstellen für eine verteilte Datenbank zur Verfügung. Als verteilte Datenbank steht eine Implementierung mit GridGain und Hazlecast zur Verfügung, die automatisch von FiLeCC verwaltet werden.

Um die Zusammenarbeit der eingesetzten Technologien zu gewährleisten, fehlen noch einige Steuerungskomponenten. Zu dessen Aufgaben gehört das Verteilen von Software innerhalb der Cloud, das Starten und Stoppen von einzelnen Webservices, die gleichmäßige Nutzung der zur Verfügung gestellten Rechenressourcen und die Verfügbarkeitsgarantie für Webservices zu gewährleisten. Diese Komponenten sind im nachfolgenden Kapitel 7 beschrieben.

KAPITEL 7

Software-Architektur

FiLeCC ist eine Cloud-Plattform, die die Nutzung freier Kapazitäten eines aus mehreren Ressourcen bestehenden Rechensystems für andere Anwendungen ermöglicht. Diese Anwendungen werden im Folgenden „Applikationen“ genannt. Sie können optional DPWS Webservices anbieten oder nutzen. In diesem Kapitel wird beschrieben, wie die in Kapitel 5.2 definierten Anforderungen in Form der Cloud-Plattform FiLeCC umgesetzt werden. Nach einer anfänglichen Übersicht des Gesamtsystems werden die Einzelkomponenten detailliert vorgestellt. Im Rahmen dieser Vorstellung erfolgt jeweils eine Beschreibung des konzeptionellen Lösungsansatzes sowie ausgewählter Implementationsdetails, die zum besseren Verständnis der weiteren Abläufe dienen. Eine Übersicht aller Komponenten des Gesamtsystems ist in Abbildung 7.1 zu sehen.

Die beteiligten Rechner sind in einem Netzwerk miteinander verbunden. Sie sind in Abbildung 7.1 mit *HR*1, *HR*2 und *HRn* bezeichnet. Durch die Netzwerkverbindung sind sie in der Lage, miteinander zu kommunizieren und können u. a. eine DPWS-Schnittstelle anbieten. In Abbildung 7.1 ist diese als eine Wolke dargestellt. Da FiLeCC sowohl komponentenabhängige als auch komponentenunabhängige Applikationen unterstützen muss (siehe Kapitel 5.2.1), ist die folgende Unterscheidung wichtig: Applikationen, deren Ausführung an eine Hardwarekomponente gebunden ist, werden als *lokale Applikation* bezeichnet. Webservices, deren Ausführung komponentenunabhängig ist, heißen entsprechend *bewegliche Applikation*. Siehe hierzu auch Kapitel 5.2.1. Diese beweglichen Applikationen müssen jederzeit verfügbar sein. Das gilt auch dann, wenn ein Teil des Systems ausfällt oder es zu einer Trennung der Netzwerkverbindung innerhalb eines Netzwerks (*Netsplit*) kommt. In Abbildung 7.1 sind die lokalen Applikationen mit schraffiertem und die beweglichen Applikationen mit gepunktetem Hintergrund dargestellt. Applikationen mit einer zweiten Umrandung enthalten parallele Algorithmen und verwenden daher GridGain (siehe Kapitel 6.5.1) zur Umsetzung der verteilten Berechnungen. Die GridGain-Komponente ist ebenfalls in Abbildung 7.1 als Wolke dargestellt. Die ständige Verfügbarkeit von beweglichen Applikationen wird von einem Watchdog-System, bestehend aus einer globalen und

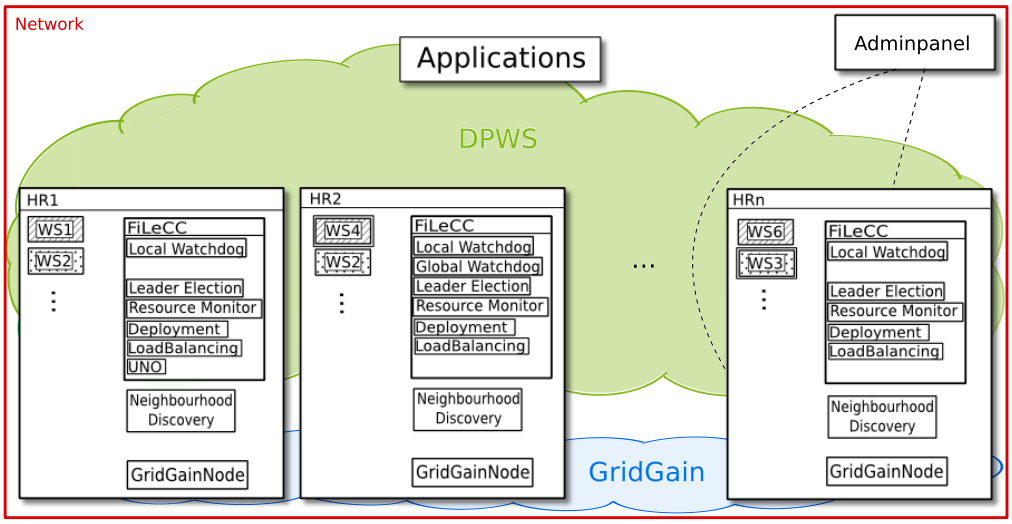


Abbildung 7.1: Aufbau des Systems. Lokale Applikationen (hier nur Webservices) sind mit schraffiertem, bewegliche mit gepunktetem Hintergrund dargestellt. Webservices verwenden DPWS zur Kommunikation. Sie können über DWPS gesucht und genutzt werden. Die Webservices sind daher innerhalb der DPWS-Wolke zu finden. Die doppelt umrandeten Applikationen enthalten parallele Algorithmen und können in der GridGain-Wolke für verteiltes Rechnen ausgeführt werden.

mehreren lokalen Komponenten, (siehe Kapitel 7.5) sichergestellt. Es handelt sich dabei um einen globales System zur kontinuierlichen Überwachung der Applikationen, der zu einem Zeitpunkt nur auf einem der teilnehmenden Rechnern gestartet ist. Dieser sorgt dafür, dass zu jeder Zeit die Ausführung aller gestarteter beweglicher Applikationen garantiert ist. Um dies zu überprüfen, ist auf jedem Knoten ein lokaler Watchdog gestartet, der die lokalen Prozesse überwacht und sich in regelmäßigen Abständen bei dem globalen Watchdog meldet. Durch die Nachrichten der lokalen Watchdogs kann dieser alle anderen Knoten im Netzwerk und damit auch deren lokale Prozesse überwachen. Erhält dieser von einem lokalen Watchdog keine regelmäßigen Nachrichten mehr, bedeutet dies, dass der lokale Watchdog abgestürzt, stark ausgelastet, der zugehörige Knoten möglicherweise ausgefallen oder vom Netz getrennt ist. In diesem Fall sendet der globale Watchdog eine Nachricht, die im Folgenden Hilferuf genannt wird, an eine weitere Systemkomponente, die *UNO* (siehe Kapitel 7.3). UNO ist dabei keine Abkürzung, sondern eine Anspielung auf die *Organisation der Vereinten Nationen*. Die UNO wird beauftragt, die Aufgaben des ausgefallenen Knotens auf die verbliebenen zu verteilen.

Damit die Rechner ihre primäre Aufgabe, also beispielsweise die Steuerung einer Förderanlage, erfüllen können, müssen die Applikationen lastabhängig auf den Rechenressourcen des Systems verteilt werden können. Die Auslastung von Hardwareressourcen muss also zur Laufzeit ermittelt und entsprechend darauf reagiert werden. Diese Aufgabe übernimmt der *lokale Ressourcenmonitor* (siehe Kapitel 7.7). Bewegliche Applikationen werden immer auf den Knoten gestartet, die zum Startzeitpunkt die größte Menge frei verfügbarer Rechenkapazität haben. Wird durch die vom lokalen Ressourcenmonitor ermittelten Daten festgestellt, dass der zugehörige Knoten zu wenig freie Rechenkapazität hat oder möglicherweise sogar überlastet ist, muss die bewegliche Applikation auf einen anderen Knoten verschoben werden. Der lokale Ressourcenmonitor zeichnet die Belastung eines Knotens auf und sendet bei Überlastung einen Hilferuf an die UNO, die

einen anderen Knoten auswählt und diesen anweist, bewegliche Applikationen des überlasteten Knoten zu übernehmen. Das Verfahren zur Auswahl eines neuen Knotens ist in Kapitel 7.3 beschrieben.

Sowohl die UNO als auch der globale Watchdog werden jeweils nur auf einem Knoten im Netzwerk gestartet und solange ausgeführt, bis der Knoten ausfällt oder überlastet ist. Es muss sich dabei nicht um denselben Knoten handeln. Welcher Knoten für die Ausführung des globalen Watchdogs und welcher für die UNO ausgewählt wird, wird durch ein Wahlverfahren, der sogenannten *Leaderwahl*, entschieden. In Kapitel 7.4 ist die Leaderwahl erläutert.

In Abbildung 7.1 sind die Komponenten lokaler und globaler Watchdog, Leaderwahl, Ressourcenmonitor, *Software Deployment and Runtime Management* (siehe Kapitel 7.2), *Loadbalancing* (siehe Kapitel 7.6) und UNO unter einer Sammlung von Systemdiensten mit der Bezeichnung *FiLeCC* zusammengefasst. Bis auf die UNO und den globalen Watchdog sind diese unabhängig von allen Applikationen auf jedem beteiligten Rechner gestartet. Die UNO und der globale Watchdog sind zwar auf jeder Hardwareressource installiert, werden aber jeweils nur auf einer ausgeführt.

Die Nachbarschaftserkennung ist ein Beispiel für eine Applikation für FiLeCC. Sie gehört nicht zu den Systemdiensten und ist deshalb in Abbildung 7.1 gesondert dargestellt. Details zu Nachbarschaftserkennung sind in Kapitel 7.8 zu finden. Die Aufgaben der Komponenten und deren Verknüpfung ist in den folgenden Abschnitten erläutert. Der *AppServer*, der u. a. alle Komponenten in der Reihenfolge startet, in der sie voneinander abhängen, ist in Abschnitt 7.1 erläutert. Eine Beschreibung des in Abbildung 7.1 aufgeführten Admin-Panels ist im Administratorhandbuch in Anhang B enthalten.

Im Folgenden werden nun die Entwurfskonzepte und ausgewählte Implementierungsdetails der Komponenten des Gesamtsystems detailliert erklärt.

7.1 AppServer

Der AppServer ist die Komponente von FiLeCC, die alle in den folgenden Abschnitten beschriebenen Komponenten in einzelnen Threads startet. Die Reihenfolge, in der die Anwendungen gestartet werden, ist dabei wichtig, da einige Komponenten von der Ausführung anderer abhängen. Das Diagramm in Abbildung 7.2 gibt eine Übersicht über die Startreihenfolge der Komponenten. Zunächst wird dabei auf einem Knoten der lokale Watchdog (siehe Abschnitt 7.5.1) gestartet. Er ist dafür zuständig, dass alle Anwendungen lokal überwacht werden. Anschließend werden die Threads für die GridGain-Umgebung für verteiltes Rechnen und die GridGain-Benutzerdatenbank gestartet. Es folgen die SDRM-Komponente (siehe Abschnitt 7.2) und der Ressourcenmonitor (siehe Abschnitt 7.7). Anschließend werden die notwendigen Simulatoren für die Module der Förderanlage und die Nachbarschaftserkennung (siehe Abschnitt 7.8) gestartet. Die Simulatoren werden im Realbetrieb durch spezielle Steuerungsplatinen ersetzt und müssen in diesem Fall nicht vom AppServer gestartet werden. Anschließend wird vom AppServer eine Leaderwahl angestoßen, die einen Knoten für die UNO und einen für den globalen Watchdog sucht und diese dort startet. Die UNO ist in Abschnitt 7.3 beschrieben und der globale Watchdog in Abschnitt 7.5.2.

Des Weiteren ist der AppServer dafür zuständig, neue JAR-Dateien zu laden und in derselben JVM wie FiLeCC auszuführen. Siehe dazu auch Kapitel 6.2.2. Die externen Anwendungen

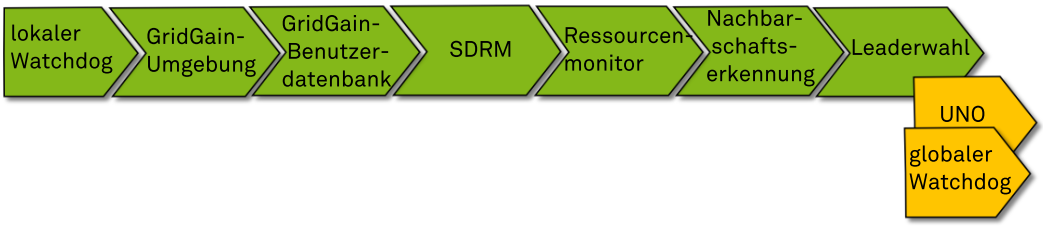


Abbildung 7.2: Reihenfolge, in der die Komponenten des FiLeCC-Gesamtsystems gestartet werden. Die Komponenten UNO und globaler Watchdog werden dabei von der Leaderwahl und nicht vom AppServer selbst gestartet.

müssen dafür das `FiLeCCApplication`- oder `MonitoredFiLeCCApplication`-Interface implementieren. Eine genaue Beschreibung dieser Interfaces ist im Entwicklerhandbuch in Ahnhang C zu finden. Nachdem eine Anwendung geladen ist, kann sie von der SDRM-Komponente (siehe Abschnitt 7.2) gestartet werden. Ein neuer Thread führt dann die Methode `start()` des `FiLeCC-Application`-Interface aus und setzt den Status der Anwendung anschließend auf `running`. Wird durch das SDRM eine Anwendung gestoppt, so wird in einem neuen Thread die Methode `stop()` aufgerufen, die dafür zuständig ist, dass alle Threads, die zu der entsprechenden Anwendung gehören, beendet werden.

Wird von einer Anwendung statt des `FiLeCCApplication`-Interfaces das `MonitoredFiLeCC-Application`-Interface implementiert, so ist es möglich, die Methode `alive()` zu verwenden. Diese wird regelmäßig von der Watchdog-Komponente (siehe Abschnitt 7.5) aufgerufen. Falls der Watchdog ein `false` empfängt, gilt die Anwendung als nicht mehr im Betrieb und wird neugestartet, falls der Status der Anwendung `running` ist.

Da die geladenen Anwendungen im selben Speicherkontext (JVM) wie die FiLeCC-Komponenten ausgeführt werden, sind Sicherheitsvorkehrungen notwendig. Zunächst lädt der AppServer externe JAR-Dateien mit einem speziellen `ClassLoader`, der das Laden von Klassen der FiLeCC-Komponenten verhindert. Andernfalls ist ein Zugriff auf globale Variablen der FiLeCC-Komponenten möglich, was ein Sicherheitsrisiko darstellt. Außerdem werden Programmausnahmen (Exceptions) innerhalb der geladenen Anwendungen im AppServer abgefangen. Dadurch führen sie nicht zu einem Ausfall der gesamten FiLeCC-Plattform.

7.2 Software Deployment and Runtime Management

Die Hauptaufgaben der *Software Deployment and Runtime Management*-Komponente (SDRM) sind, die Verteilung von Applikationen in der Cloud und das Anbieten von Start- und Stopp-Möglichkeiten für diese Applikationen.

Grund für die Selbstentwicklung einer eigenen Verteilungskomponente sind die Probleme, die bei der Evaluierung verschiedener verteilter Dateisysteme und Datenbanken auffallen (siehe Abschnitt 6.6.1). So benötigt FiLeCC die Möglichkeiten, Binärdateien auf dem System zu installieren, eine Aktualisierung bereits verteilter Binärdateien durchzuführen und Binärdateien wieder zu entfernen (siehe Anwendungsfälle P301-P303 in Kapitel 5.3). Die Verteilung muss den Zielkonflikt zwischen Ausfallsicherheit und effizienter Ressourcenausnutzung lösen. Es darf

kein einzelner Knoten existieren, der den Zugang zu den auszuführenden Daten durch einen Ausfall gefährdet. Außerdem soll die Verteilung im Idealfall optimiert zur Netzwerktopologie erfolgen.

Die SDRM-Komponente ist so gestaltet, dass eine größtmögliche Ausfallsicherheit bezüglich der Binärdateien garantiert wird ohne eine Kopie auf jedem Knoten vorzusehen. In der geplanten Standardkonfiguration darf keine vollständige Redundanz der Binärdateien durch eine Verteilung auf jeden Knoten existieren, sondern es muss eine Verteilung sein, die eine Erreichbarkeit auch nach Ausfall großer Teile des Systems garantiert. Die Verteilungsfunktion dafür wird in Unterkapitel 7.2.1 näher beschrieben.

Grundsätzlich können beliebige Binärdateien in der Cloud verteilt werden, wobei das SDRM nach *installierten* und *aktiven* Applikationen unterscheidet. Auf einem Knoten „aktiv“ bedeutet, dass der Knoten die Applikation momentan ausführt, wohingegen „installiert“ nur bedeutet, dass die Binärdatei aktuell auf diesem Knoten vorhanden ist. Ist eine Applikation aktiv, so ist sie implizit auch installiert.

Bei den Applikationen wird nach *lokalen* und *beweglichen* Applikationen (siehe Einleitung Kapitel 7) unterschieden. Lokale Applikationen werden nicht von der SDRM-Komponente verteilt, da sie nur lokal ausgeführt werden dürfen. Daher sind für die Verteilung nur bewegliche Applikationen von Bedeutung. Starten und Stoppen kann die SDRM-Komponente beide Arten von Applikationen, ebenso wie beliebige ausführbare Binärdateien. Bei beliebigen Binärdateien geht allerdings die Betriebssystemunabhängigkeit verloren, so dass der Betreiber der Cloud die Ausführbarkeit selbst sicherstellen muss.

Das Klassendiagramm dieser Komponente ist in Abbildung 7.3 dargestellt. Die SDRM-Komponente wird auf jedem Knoten im Netzwerk ausgeführt. Für die Kommunikation untereinander und mit anderen Komponenten bietet jede Instanz des SDRM ein *DPWS-Gerät* an, das mittels der `UUID` des jeweiligen Knotens identifizierbar ist. Das `SDRMDevice` bietet die Webservices `AdminService`, `FileService` und `StartStopService` an.

7.2.1 Softwareverteilung

Um Applikationen in der Cloud verteilen zu können, müssen sie zunächst im Netzwerk integriert werden. Das Integrieren der Applikationen geschieht über die sogenannte *Installation* auf einem der im Netzwerk befindlichen Knoten.

Installation bedeutet in diesem Fall das Ablegen der zu verteilenden Dateien in einem dafür vorgesehenen Ordner auf dem Knoten. Der Ordner wird beim Start von FiLeCC von der SDRM-Komponente unter dem Namen *welcome* angelegt und im Betrieb von der SDRM-Komponente auf neue Dateien überwacht. Die Installation kann durch das direkte Kopieren von Dateien in den *welcome*-Ordner durchgeführt werden. Eine andere Möglichkeit ist der Aufruf der für die Installation vorgesehenen Operation `InstallWebService` aus dem SDRM-Service `AdminService`. Diese Operation nimmt als Parameter unter anderem die Applikation als Binäranhang entgegen. Ein solcher Aufruf kann z. B. über das Adminpanel (siehe Adminhandbuch Anhang B) geschehen.

Jede Applikation, die in der Cloud verteilt werden soll, muss als Archiv vorliegen. Dabei kann es sich sowohl um eine *JAR*-Datei als auch um eine *ZIP*-Datei handeln. Das Archiv muss eine Beschreibungsdatei enthalten. In dieser sind Informationen zu der Applikation hinterlegt, die zur

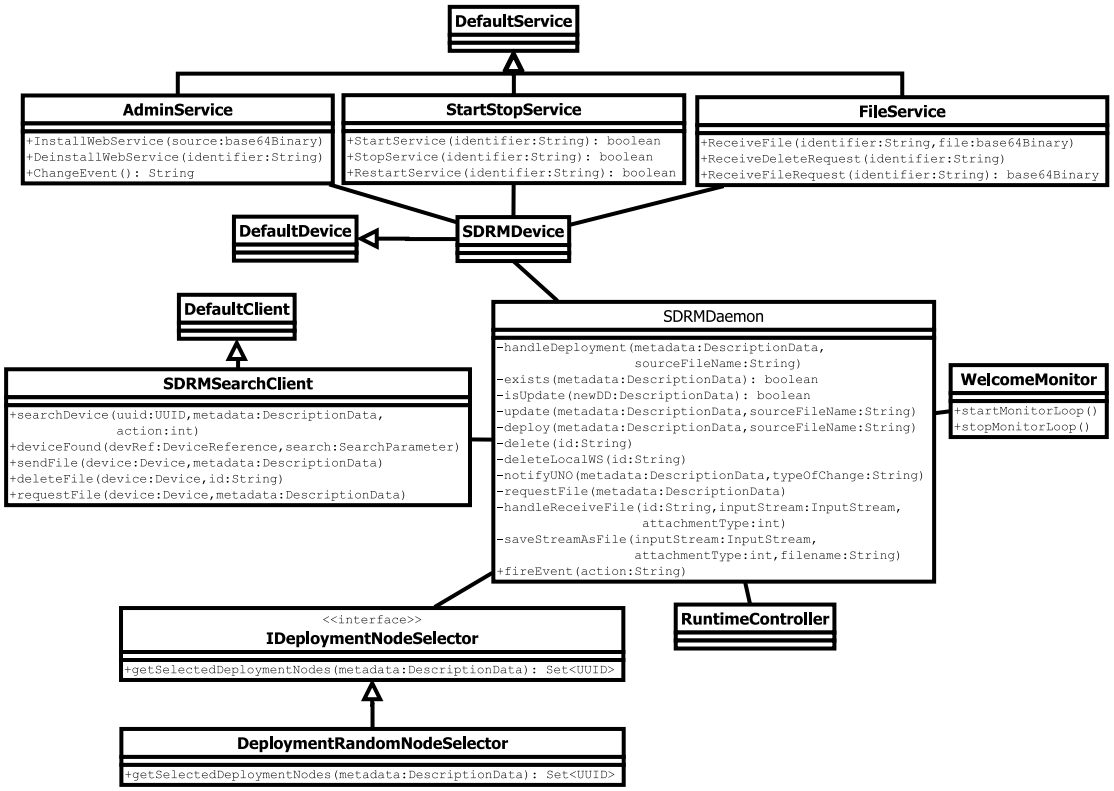


Abbildung 7.3: Klassendiagramm der SDRM-Komponente

weiteren Verarbeitung benötigt werden. Die Beschreibungsdatei liegt im XML-Format vor und befindet sich unter dem Namen *filecc.xml* im Stammordner der Archivdatei.

Neu installierte Applikationen werden anhand dieser Beschreibungsdatei überprüft. Die Beschreibungsdatei wird mit Hilfe eines XML-Schemas deklariert (vgl. Listing 7.1).

Listing 7.1: XML-Schema der Beschreibungsdatei

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://filecc.flw.mb.tu-dortmund.de"
    xmlns="http://filecc.flw.mb.tu-dortmund.de">
    <xs:element name="fileccDescription">
        <xs:complexType >
            <xs:sequence>
                <xs:element name="id" type="xs:string"/>
                <xs:element name="version" type="xs:string"/>
                <xs:element name="cmdline_arguments" type="xs:string" default="" ↩
                    minOccurs="0"/>
                <xs:element name="multicast_address" type="xs:string" default="" ↩
                    minOccurs="0"/>
                <xs:element name="executable_filename" type="xs:string" default="" ↩
                    minOccurs="0"/>
                <xs:element name="keep_local" type="xs:boolean" default="false" ↩
                    minOccurs="0"/>
                <xs:element name="main_class" type="xs:string" default="" minOccurs=↩
                    "0"/>
            </xs:sequence>
        </xs:complexType>
```

```
    </xs:element>
</xs:schema>
```

Die Beschreibungsdatei enthält einen eindeutigen Identifikator für die Applikation sowie eine Versionsnummer. Die restlichen Informationen sind optional und können bei Bedarf angegeben werden. Weitere Informationen zu den unterschiedlichen Konfigurationsparametern befinden sich im Kapitel 3 des Entwicklerhandbuchs in Anhang C.

Wenn eine Beschreibungsdatei vorliegt und diese mittels des XML-Schema validiert werden kann, werden die enthaltenen Daten in einem `DescriptionData`-Objekt (vgl. Abbildung 7.4) gekapselt und zur weiteren Verarbeitung an den `SDRMDaemon` übergeben.

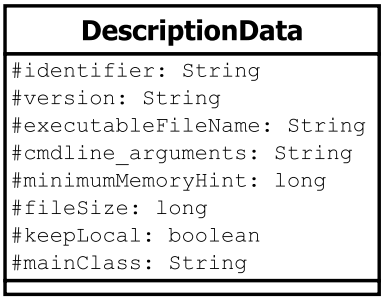


Abbildung 7.4: `DescriptionData`-Objekt

In der Klasse `SDRMDaemon` befindet sich die Verarbeitungslogik der SDRM-Komponente. Über die Methode `handleDeployment` wird die Verarbeitung der neu installierten Applikation gestartet. Zunächst wird anhand der Identifikator- und Versionsinformationen aus der Beschreibungsdatei entschieden, was mit der neuen Applikation geschehen soll. Dies kann das Verteilen der Applikation, die Aktualisierung bereits im System vorhandener Applikationen oder das Ignorieren und Löschen der neuen Applikation sein.

Das SDRM speichert die Daten zu jeder installierten Applikation in einer verteilten Datenbank, damit diese auf jedem Knoten zur Verfügung stehen. Zusätzlich wird in der Datenbank gespeichert, welche Applikation auf welchem Knoten installiert und auf welchem Knoten aktiv ist. In Kapitel 7.9.3 werden die von der SDRM-Komponente zu speichernden Daten genauer beschrieben und Beispieldaten gegeben.

Da die Metadaten (`DescriptionData`) von jeder in der Cloud installierten Archivdatei in der verteilten Datenbank gespeichert sind, kann nach dem Installieren zunächst geprüft werden, ob die Archivdatei bereits vorhanden ist. Dies geschieht anhand des Identifikators. Der Identifikator muss in der gesamten FiLeCC-Umgebung eindeutig sein. Daher wird davon ausgegangen, dass ein noch nicht in der Datenbank befindlicher Identifikator eine neue Applikation impliziert und zu einer Verteilung führt. Ein bereits bekannter Identifikator bedeutet folglich, dass es sich bei der neu installierten Archivdatei nicht um eine neue Applikation handelt und daher eine Aktualisierung der bereits vorhanden Applikation erfolgen muss. Bevor eine Aktualisierung durchgeführt wird, werden die Versionsnummern verglichen. Nur im Fall einer höheren Version der neu installierten Applikation wird eine Aktualisierung eingeleitet. Eine niedrigere oder gleiche Version führt dazu, dass die Archivdatei nicht weiter beachtet und gelöscht wird.

Bei der Verteilung einer neuen Applikation muss zunächst eine Gruppe von Knoten ausgewählt werden, auf welche die Applikation verteilt werden soll. Dies geschieht über eine Implemen-

tierung der Java-Schnittstelle `IDeploymentNodeSelector`, die die Verteilungsfunktion für die Applikation definiert. Die Standardimplementierung dieser Schnittstelle (`DeploymentRandomNodeSelector`) geht davon aus, die Netzwerktopologie nicht zu kennen. Sie nimmt eine Verteilung mittels einer Zufallsfunktion vor. Dabei werden zufällig `x` Prozent der Knoten, die sich im Netzwerk befinden ausgewählt. Bei dieser Auswahl wird der freie Festplattenspeicherplatz auf den Knoten beachtet, sodass die zu verteilende Archivdatei bei der Verteilung genug Platz auf den Knoten hat. Die Prozentzahl wird über eine Konfigurationsvariable bestimmt. Sie gibt den Faktor an, der den Anteil an der Gesamtzahl der Knoten des Netzwerks angibt, auf die verteilt werden soll. Vorkonfiguriert ist zunächst ein Faktor von 0.25, sodass zufällig ein Viertel aller Knoten ausgewählt wird, die Applikation zu erhalten. Der Verteilungsfaktor muss sich an der Größe des Netzwerks orientieren um die Ausfallsicherheit zu gewährleisten. Bei einem kleinen Netzwerk mit wenigen Knoten wird der Faktor z. B. höher gewählt als bei einem großen Netzwerk mit sehr vielen Knoten. Die genauen Kennwerte dieser Funktion können durch Experimente gewonnen werden. Ist die Topologie des Netzwerks bekannt, so kann die Knotenermittlung auf Grundlage der Netzwerktopologie erfolgen. Es wird darauf geachtet, dass jede Archivdatei höchstens `n` Knoten von jedem Knoten entfernt vorhanden ist. Auf diese Weise können auch Härtefälle wie eine komplette Zweiteilung des Netzwerks beherrscht werden, indem sicher ist, dass die Applikation noch in naher Netzwerknachbarschaft vorhanden ist. Der Wert von `n` kann aus Experimenten gewonnen werden.

Bei der eigentlichen Verteilung wird zunächst die Applikation auf dem Knoten, auf dem sie installiert ist, in den *storage*-Ordner verschoben. Dieser wird wie der *welcome*-Ordner beim Start von FiLeCC angelegt. Im *storage*-Ordner eines Knotens liegen alle Archivdateien, die auf dem Knoten installiert sind. Bei dem Verschieben wird die Archivdatei zusätzlich so umbenannt, dass sie den selben Namen, wie der dazugehörige Identifikator trägt. Dadurch ist eine Zuordnung der Daten in der Datenbank zu der konkreten Archivdatei vereinfacht und es wird einem versehentlichen Überschreiben anderer Applikationen vorgebeugt. Die Informationen über die neue Applikation werden in die Datenbank eingetragen. Anschließend wird die Archivdatei an alle ausgewählten Knoten verteilt.

Für den eigentlichen Dateitransfer wird das `SDRMDevice` auf den jeweiligen Knoten gesucht und die `ReceiveFile`-Operation des `FileService` aufgerufen. Die Operation erhält als Parameter den Identifikator der Archivdatei und als Anhang die Archivdatei selbst. Die Verarbeitung auf der Empfängerseite findet im `SDRMDaemon` des Empfängerknotens statt. Die Archivdatei wird von der Operation als `InputStream` bereitgestellt und an den `SDRMDaemon` weitergeleitet. Dort wird der `InputStream` wieder in eine Archivdatei geschrieben und im *storage*-Ordner des Knotens gespeichert. Anschließend erfolgt ein Eintrag in die Datenbank, der angibt, dass die Archivdatei auf dem Knoten vorliegt.

Ein Update funktioniert fast identisch wie die Verteilung von Applikationen. Anstatt einer neuen Gruppe von Knoten werden die Knoten aus der Datenbank gelesen, auf denen die Applikation bereits vorliegt. An diese Knoten wird die Archivdatei über die `ReceiveFile`-Operation gesendet. Bei der Verarbeitung der ankommenden Archivdatei wird zunächst geprüft, ob die ältere Version momentan ausgeführt wird. Ist das der Fall, so wird die Applikation vorübergehend gestoppt. Das Update wird durchgeführt, indem die Archivdatei mit der neuen Version überschrieben wird. Anschließend wird die Applikation wieder auf den Knoten gestartet, auf denen sie zuvor lief.

Neben der Verteilung von Applikationen ist das SDRM auch dafür zuständig, sie wieder aus dem Gesamtsystem zu entfernen. Dafür ist die `DeinstallWebService`-Operation des `AdminService` vorgesehen. Die Operation bekommt den Identifikator der zu entfernenden

Applikation übergeben. Der `FileService` des `SDRMDevice` enthält die `deleteWebService`-Operation. Diese Operation wird auf jedem Knoten, auf dem sich die Archivdatei befindet, aufgerufen. Die Verteilung der Archivdatei ist dabei aus der Datenbank ersichtlich. Der Aufruf der `deleteWebService`-Operation hat das Löschen der Archivdatei auf dem jeweiligen Knoten zur Folge. Zuvor wird geprüft, ob die Applikation gerade ausgeführt wird, und gegebenenfalls gestoppt, bevor sie gelöscht wird. Dadurch wird ein sauberes Entfernen von Applikationen aus dem System ermöglicht.

7.2.2 Starten und Stoppen von Software

Die SDRM-Komponente stellt den lokalen Webservice `StartStopService` zur Verfügung, mit dessen Hilfe die anderen Komponenten des Systems die installierten Applikationen starten und stoppen können.

Um eine Applikation zu starten, wird die Operation `startService` mit ihrem Identifikator (siehe Abschnitt 7.2.1) aufgerufen. Der `StartStopService` prüft zuerst, ob die entsprechende Binärdatei auf dem Knoten vorhanden ist. Ist die Binärdatei nicht lokal verfügbar, nutzt die Startoperation den Webservice `FileService`, um die Binärdatei von einem anderen Knoten anzufragen. Der Webservice entnimmt der verteilten Datenbank die Informationen auf welchem Knoten die Binärdatei der Applikation liegt. Erhält er die Daten von einem anderen Knoten, werden sie lokal gespeichert und ihre Verfügbarkeit in der verteilten Datenbank vermerkt. Da die Binärdatei der zu startenden Applikation nun lokal vorhanden ist, kann sie gestartet werden.

Beim Starten gibt es drei verschiedene Mechanismen, die FiLeCC unterstützt. Entweder nutzt die Applikation den `AppServer` (siehe Abschnitt 7.1), und wird somit in derselben Java-VM ausgeführt wie FiLeCC, und bietet weiterhin dem *Local Watchdog* die Möglichkeit der Statusüberwachung der Applikation (siehe Abschnitt 7.5). Oder sie wird als eigener Prozess auf dem Knoten gestartet. Der letzte Fall unterscheidet sich nochmals zwischen Java und beliebigem Code. Welcher Mechanismus benutzt wird, ergibt sich implizit aus der Applikation, die zu starten ist. FiLeCC benötigt diese Information aus der Beschreibungsdatei der externen Applikation. Prozessnummern und Verweise auf auf Threads von `AppServer`-Applikationen werden im Speicher behalten, um weiterhin Kontrolle über die gestarteten Applikationen zu haben. Beim Starten einer Applikation in derselben Java-VM wie FiLeCC wird zusätzlich zum Start der Applikation noch der `LocalWatchdog` darüber informiert, dass eine Applikation gestartet wurde. Dies geschieht über den Aufruf der Methode `startMonitoringApp` des `LocalWatchdog`.

Zum Stoppen bereits aktiver Applikationen stellt der `StartStopService` die Operation `stopService` zur Verfügung. Als Parameter wird der Identifikator der zu stoppenden Applikation benötigt. Geht ein solcher Befehl ein, so prüft der `StartStopService`, ob die Applikation lokal aktiv ist, und beendet diese gegebenenfalls. Läuft die Applikation in derselben Java-VM wie FiLeCC, so wird die Methode `stopMonitoringApp` des `LocalWatchdog` aufgerufen, um diesen darüber zu informieren, dass die Applikation nicht weiter überwacht werden muss.

Die Operation `restartService` ist eine einfache Kapselung der nacheinander ausgeführten Operationen `stopService` und `startService`.

7.3 UNO

Die UNO ist ein Baustein der FiLeCC-Architektur, an den bei Ausfall eines Knotens eine Nachricht, auch Hilferuf genannt, geschickt werden kann. Anschließend versucht die UNO alle auf dem ausgefallenen Knoten ausgeführten beweglichen Webservices auf anderen Knoten zu starten. Lokale Webservices werden von der UNO nicht behandelt, da diese nicht auf beliebigen Knoten ausgeführt werden. Der Ausfall eines Knotens wird vom *Global Watchdog* (siehe Abschnitt 7.5.2) erkannt und an die UNO gemeldet. Weiterhin wird bei Überlastung eines Knotens eine Nachricht an die UNO geschickt, die anschließend andere Knoten anweist, zusätzliche Instanzen der Webservices des überlasteten Knotens zu starten. Die Überlast-Nachricht wird im folgenden Hilferuf genannt und wird durch den *Ressourcenmonitor* (siehe Unterkapitel 7.7) initiiert. Durch einen Hilferuf wird die Last auf einem oder mehreren Knoten verringert, da ein Teil der Webserviceanfragen nach dem Hilferuf von dem Knoten, auf dem der Webservice neu gestartet wird, beantwortet wird. Bei Neuinstallation eines Webservices ist die UNO dafür zuständig, einen geeigneten Knoten auszuwählen, auf dem dieser gestartet wird.

Die UNO darf nur einmal in einer FiLeCC-Cloud ausgeführt werden. Um dies sicherzustellen, wird ein selbstentwickeltes Leaderwahlverfahren eingesetzt, das in Unterkapitel 7.4) vorgestellt wird. Im Rahmen dieses Verfahrens wird auch ein Ausfall der UNO erkannt und entsprechend reagiert.

Für die Implementierung der Webservices werden Basisklassen aus der JMEDS-Bibliothek genutzt. Eine Skizzierung der Implementierung ist in Abbildung 7.5 dargestellt.

Um die zuvor beschriebene Funktionalität bereitzustellen, bietet die UNO drei Operationen als Webservices an. Die Operation `LostNodeCompensationOperation` wird vom globalen Watchdog aufgerufen, wenn ein Knoten ausgefallen ist. Die UNO ist anschließend dafür zuständig, alle aktiven beweglichen Webservices des ausgefallenen Kontens, auf andere Knoten zu verteilen. Dazu wird der Operation `LostNodeCompensationOperation` der Identifikator des ausgefallenen Knotens als Parameter übergeben. Die aktiven Webservices dieses Kontens sind in der SDRM-Datenbank zu finden. Für jeden dieser Webservices wird ein Knoten aus einer speziellen Knotenliste ausgewählt. Diese Knotenliste ist nach verfügbarer Rechenleistung sortiert und wird mithilfe der Lastdatenbank berechnet. Das *SDRM* (siehe Kapitel 7.2) des Knotens wird anschließend angewiesen, den Webservice zu starten. Dazu werden die Methoden der Klasse `UNOOperations` genutzt, welche mit Hilfe der JMEDS-Bibliothek eine Verbindung zum *SDRM* herstellt. Kann der Webservice auf diesem Knoten nicht gestartet werden, wird der nächste Knoten aus der Liste gewählt. Existiert kein geeigneter Knoten, auf dem der Webservice gestartet werden kann, wird eine Nachricht an den Administrator geschickt, die auf diesen kritischen Zustand hinweist. Der Administrator empfängt diese Nachricht über die Administrationsoberfläche (siehe Anhang B).

Die Operation `HelpRequestOperation` implementiert die Operation, die den Hilferuf entgegen nimmt und ist identisch zur Operation `LostNodeCompensationOperation`. Dieser Operation wird der Identifikator des Knotens übergeben, der überlastet ist. Diese Operation wird bei Überlast eines Knotens vom *Ressourcenmonitor* (siehe Unterkapitel 7.7) aufgerufen.

Als dritte Operationen bietet die UNO die Operation `StartWebserviceOperation` an, welche aufgerufen wird, wenn ein neuer Webservice erstmals im System gestartet werden soll. Auch diese Operation verfährt analog zur Operation `LostNodeCompensationOperation`, mit dem

Unterschied, dass die Liste der Webservices nicht aus der SDRM-Datenbank gelesen werden muss, sondern dieser Operation der Identifikator des zu startenden Webservice übergeben wird.

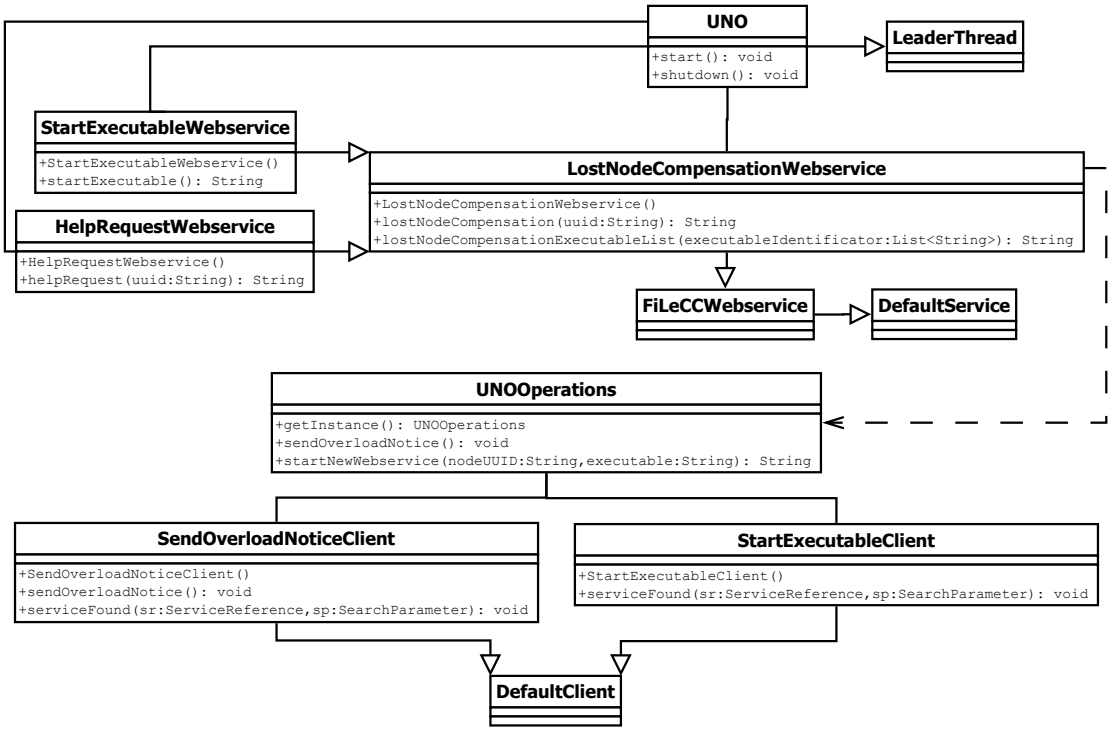


Abbildung 7.5: Klassendiagramm UNO-Komponenten

7.4 Leaderwahl

Das Leaderwahlverfahren dient dazu, einen Knoten des Netzes zu bestimmen, welcher eine systemweit einzigartige Rolle (Leader) übernimmt und die zugehörige Aufgaben erhält. Eine solche Aufgabe ist zum Beispiel die die UNO oder der globale Watchdog. Auch benötigen gegebenenfalls durch den Endbenutzer vorgenommene Programmerweiterungen (*Applikationen*) die Nutzung eines *Leaders*. Die Hauptaufgabe ist also, dafür zu sorgen, dass ein Leader nur einmal vorhanden ist. Ansonsten kann derselbe Prozess mehrfach ausgeführt und dieselbe Aufgabe mehrfach ausgeführt werden. Die mit der Leaderrolle verbundenen Aufgaben müssen zusätzlich zu den normalen Prozessen bearbeitet werden, so dass die aktuelle Auslastung der Knoten bei der Wahl erücksichtigt werden muss. Deshalb wird im Zuge der Leaderwahl der Knoten mit der aktuell geringsten Auslastung bestimmt und als Leader gekennzeichnet. Diese so gewählten Leader sollen die gesonderten Aufgaben übernehmen. Zum Beispiel wird ein Leader benötigt, welcher die Aufgabe des *globalen Watchdogs* (siehe Kapitel 7.5) übernimmt. Dabei wird von den durchzuführenden Aufgaben in soweit abstrahiert, dass beliebige Aufgaben in das Leaderwahlverfahren integriert werden können. Dazu erhalten bei der Integration der Applikationen alle Knoten, auf denen die entsprechenden Klassen der Applikation durch das SDRM verteilt wurden, die Aufgabe als ausführbares Programm. Der als Leader gewählte Knoten kann so direkt den entsprechenden Code ausführen und mit der Bearbeitung der Aufgabe beginnen. Die einzelnen, relevanten Teilaspekte des Leaderwahlverfahrens werden in den folgenden Unterab-

schnitten näher beschrieben. Im Abschnitt Algorithmus (Abschnitt 7.4.1), wird der Algorithmus der Leaderwahl erläutert, sowie der Mechanismus zum behandeln aufgetretender Konfliktfälle. Abschnitt 7.4.2 beschreibt die integrierte Ausfallkontrolle, welche selbstständig ausgefallene Leader erkennt. Im Anschluss daran, wird die Implementierung des beschriebenen Algorithmus in Abschnitt 7.4.3 erläutert. Dieser Algorithmus dient dazu, Leader für die Aufgaben zu wählen, welche einen solchen benötigen. Im folgenden werden diese Aufgaben einheitlich Applikationen genannt, welche den Teil der Anwendung darstellen, der einen Leader benötigt.

7.4.1 Algorithmus

Obwohl für verteilte Systeme eine Vielzahl effizienter Algorithmen existieren, welche die Wahl eines Stellvertreters ermöglichen (z.B. Echo-Algorithmus), muss für die FiLeCC-Architektur ein neuer Ansatz entwickelt werden. Die Gründe hierfür liegen darin, dass keine Informationen über das zugrunde liegende Netz vorhanden sind. Weiterhin existieren unterschiedliche Nachrichtentypen (z.B. bedingt durch die Leaderwahl selbst oder die Ausfallkontrolle), welche durch die einzelnen Knoten ausgetauscht werden müssen und von bestehenden Algorithmen nicht unterstützt werden. Dies bedeutet der Leaderwahlalgorithmus muss alle unterschiedlichen Komponenten gleichzeitig berücksichtigen. Um das verwendete Netzwerk hierbei nicht mit unnötigen Nachrichten zu überlasten, werden sämtliche Nachrichten via Multicast zwischen den entsprechenden Knoten ausgetauscht. Da die Topologie des Netzes die Form einer Stern-Topologie aufweisen kann, erwirken bestehende Ansätze keine nennenswerte Effizienzsteigerung im Sinne der Nachrichtenreduktion. Da jede Station im Netz aufgrund der Umsetzung der Kommunikation mittels Multicast, lediglich notwendige Nachrichten empfängt. Die auftretenden Konfliktfälle, welche durch durch nahezu gleichzeitig startende Knoten ausgelöst werden, werden durch einen entsprechenden Mechanismus erkannt und aufgelöst (siehe Abschnitt 7.4.1). Das erfolgreiche Beenden einer Leaderwahl wird durch das anschließende Bekanntmachen des neu gewählten Leaders mitgeteilt. Im Folgenden wird der Ablauf der implementierten Leaderwahl erläutert.

Ablauf Die eigentliche Leaderwahl, deren Ablauf in Abbildung 7.6 grafisch veranschaulicht wird, basiert darauf, den Knoten mit der geringsten Auslastung zu bestimmen, um diesem eine bestimmte Aufgabe zuweisen zu können.

Beim Verteilen einer Applikation wird ebenfalls das Leaderwahlverfahren auf den entsprechenden Knoten gestartet. Dies wird in dem Aktivitätsdiagramm in Abbildung 7.6 dargestellt. Wird die Leaderwahl gestartet, sendet dieser Knoten eine so genannte *ELECTION*-Nachricht an alle Knoten in der entsprechenden Multicastgruppe. Beim Verteilen einer Applikation durch das SDRM, werden alle entsprechenden Knoten gleichzeitig in einer Multicastgruppe zusammengefasst. Diese Knoten sind potentielle Leader für diese Applikation und nehmen an der Leaderwahl für diese Applikation teil. In dieser Nachricht sendet der Leaderwahl-Initiator seine UUID, um mitzuteilen, wer diese Leaderwahl gestartet hat. Im Anschluss daran wartet der Leaderwahl-Algorithmus eine in der Anwendung fest definierte Zeit auf Antworten der anderen Knoten. Diese Zeitspanne resultiert aus Tests, welche dazu dienten die Kommunikation zeitlich zu Analysieren. In dieser Antwort quittiert jeder Knoten, den Empfang der *ELECTION*-Nachricht, seine aktuelle Auslastung, sowie seine *UUID*, welche der Zuordnung der Auslastung zum entsprechenden Knoten dient. Nach Ablauf der Wartezeit, müssen zwei Fälle unterschieden werden. Zum einen ist es möglich, dass keine Antwort eingegangen ist, sodass angenommen wird, dass kein anderer Knoten im Netz verfügbar ist. In diesem Fall wird der Initiator selbst zum Leader

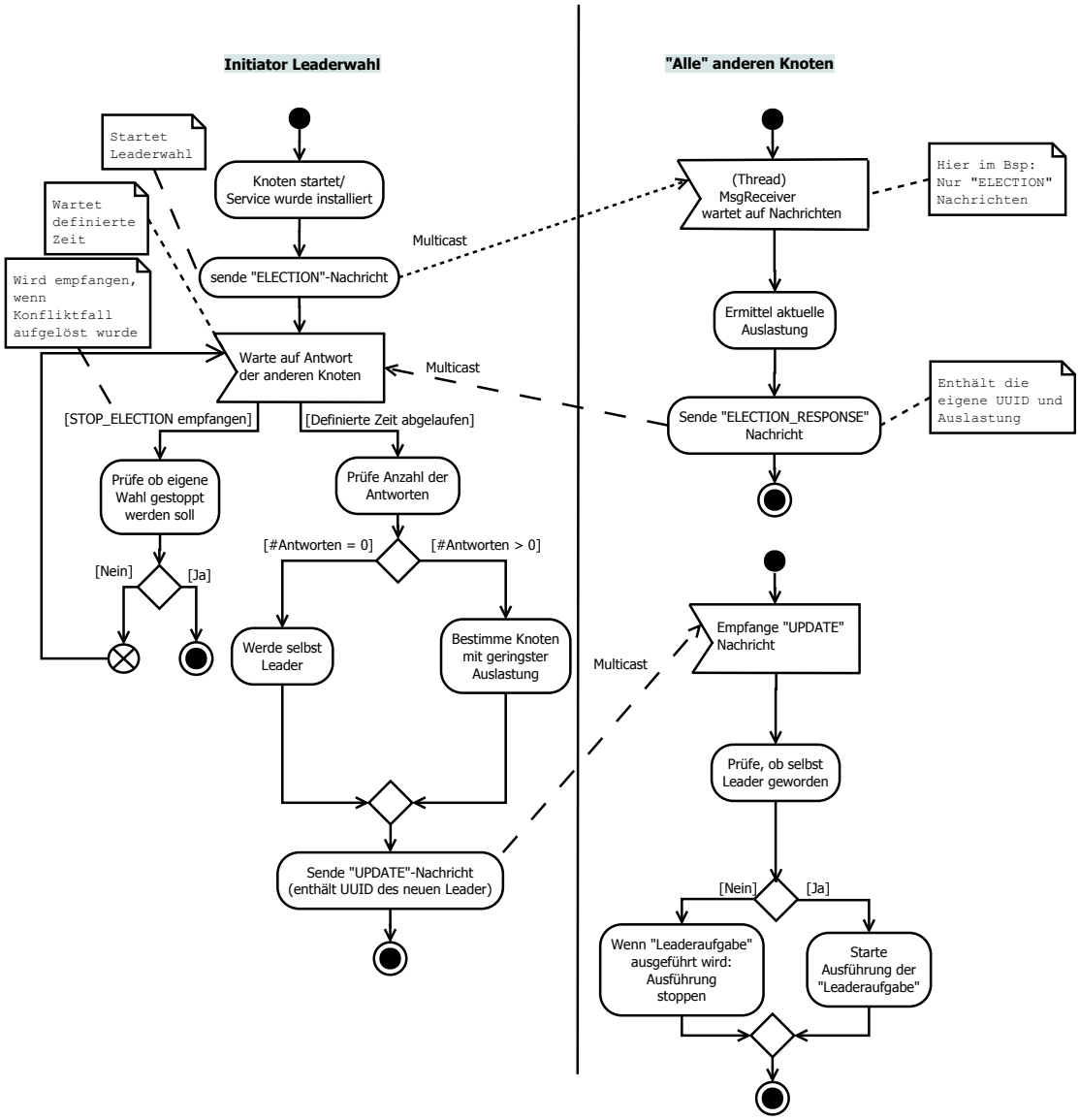


Abbildung 7.6: Ablauf der Leaderwahl

gewählt. Zum anderen ist es möglich, dass mehrere Antworten eingegangen sind. Ist dies der Fall, so wird anhand der darin empfangenen Auslastungen der am wenigsten ausgelastete Knoten durch den Leaderwahl-Initiator ermittelt. Dieser Knoten nimmt dann die Rolle des Leaders ein. Zum Beenden der Leaderwahl wird nun die *UUID* des als Leaders bestimmten Knoten als *UPDATE*-Nachricht den anderen Knoten der Multicastgruppe mitgeteilt.

Aufgrund des beschriebenen Ablaufs, muss jeder Knoten im Rahmen der Leaderwahl auf eingehende Nachrichten reagieren und eine entsprechende Verarbeitung anstoßen. Das Empfangen dieser Nachrichten muss zu allen anderen Prozessen parallel laufen, damit eine zeitnahe Verarbeitung gewährleistet ist. Anhand des Aktivitätsdiagramm in Abbildung 7.7 wird deutlich, dass im normalen Ablauf der Leaderwahl drei Nachrichtentypen empfangen werden können. Beim Start einer Leaderwahl werden *ELECTION*-Nachrichten versendet. Da diese eine laufende Leaderwahl bekannt machen, muss der empfangende Knoten aufgrund dieser Nachricht seine

aktuelle Auslastung ermitteln und als Antwort auf diese Nachricht senden. Die Behandlung von Konfliktfällen, die durch diesen Nachrichtentyp verursacht werden, wird im folgenden Abschnitt beschrieben. Weiterhin können *UPDATE*-Nachrichten empfangen werden. Diese signalisieren das erfolgreiche Beenden einer Leaderwahl. Wird diese Nachricht empfangen, müssen die Verlierer der Leaderwahl lediglich die darin enthaltene UUID speichern. Der Gewinner der Leaderwahl, der durch die enthaltene UUID identifiziert werden kann, muss die zu verteilende Aufgabe übernehmen. Wie in Abbildung 7.7 veranschaulicht wird, muss dann die Applikation gestartet werden, welche durch den Leader ausgeführt werden soll. Dieses Aktivitätsdiagramm veranschaulicht den Standardnachrichtenfluss ohne eigene Leaderwahl sowie ohne Ausfallkontrolle. Eine *STOP_ELECTION*-Nachricht bedeutet, dass eine vom empfangenden Knoten initiierte Leaderwahl abgebrochen werden muss. Verursacht wird ein solcher Abbruch durch zwei parallel laufende Wahlen, welche dieselbe Aufgabe verteilen wollen. Dieser Konfliktfall wird im folgenden genauer erläutert.

Konfliktfall Wie zuvor beschrieben, kann nicht ausgeschlossen werden, dass beliebig viele weitere Knoten eine Leaderwahl für dieselbe Applikation starten. Dies stellt einen Konfliktfall dar, welcher aufgelöst werden muss. Durch dieses Auflösen können eventuell auftretende Probleme, sowie unnötig versendete Nachrichten verhindert werden. Aus diesem Grund werden die eigentliche Leaderwahl und das Empfangen weiterer Nachrichten als parallele Prozesse ausgeführt. Der Ablauf der Konfliktlösung wird in Abbildung 7.8 dargestellt.

Ein Konfliktfall tritt auf, sobald ein Knoten während einer selbst initiierten Leaderwahl eine *ELECTION*-Nachricht empfängt. Da dies bedeutet, dass nun zwei Leaderwahlen für eine Applikation durchgeführt werden, muss eine dieser laufenden Wahlen gestoppt werden. Für die Entscheidung, welche Wahl gestoppt wird, vergleicht der empfangene Knoten die eigene UUID mit der aus der empfangenden *ELECTION*-Nachricht. Ist die eigene UUID lexikografisch größer sein, wird die fremde Leaderwahl gestoppt. Dazu wird eine *STOP_ELECTION*-Nachricht mit der UUID der zu stoppenden Leaderwahl als Inhalt an die entsprechende Multicast-Gruppe versendet. Ist hingegen die eigene UUID kleiner, muss die eigene Wahl gestoppt werden. Dazu muss lediglich der eigene Wahlprozess gestoppt werden. Alle eingehenden Antworten können verworfen werden.

7.4.2 Ausfallkontrolle

Da ein Leaderknoten eine einzigartige Aufgabe übernimmt, muss sichergestellt sein, dass dieser Knoten zu jeder Zeit erreichbar ist. Ansonsten muss sofort ein Nachfolger bestimmt werden, welcher diese Aufgabe übernimmt. Um den Ausfall eines Leaders zeitnah zu bemerken, sender jeder Leader nach einer fest definierten Zeit ein Lebenszeichen in Form einer *STILL_ALIVE*-Nachricht. Die zu wartende Zeitspanne wurde durch eine Analyse des Kommunikationsmechanismus ermittelt. Mit dieser Nachricht teilt er den anderen Knoten mit, dass er seine Funktion als Leader noch wahrnehmen kann. Dieser Ablauf wird in Abbildung 7.9 dargestellt, wobei die linke Seite die Rolle des Leaders darstellt.

Alle anderen Knoten führen für jede Leaderaufgabe eine Instantz der *MalfunctionDetection* aus (siehe Aktivitätsdiagramm 7.9). Diese wartet eine vorgegebene Zeit auf die Benachrichtigung des Leaders, welche anzeigt, dass dieser noch funktionsfähig ist. Da die Wartezeiten der Knoten zu jeweils unterschiedlichen Zeitpunkten gestartet wird, reagieren nicht alle Knoten auf einmal.

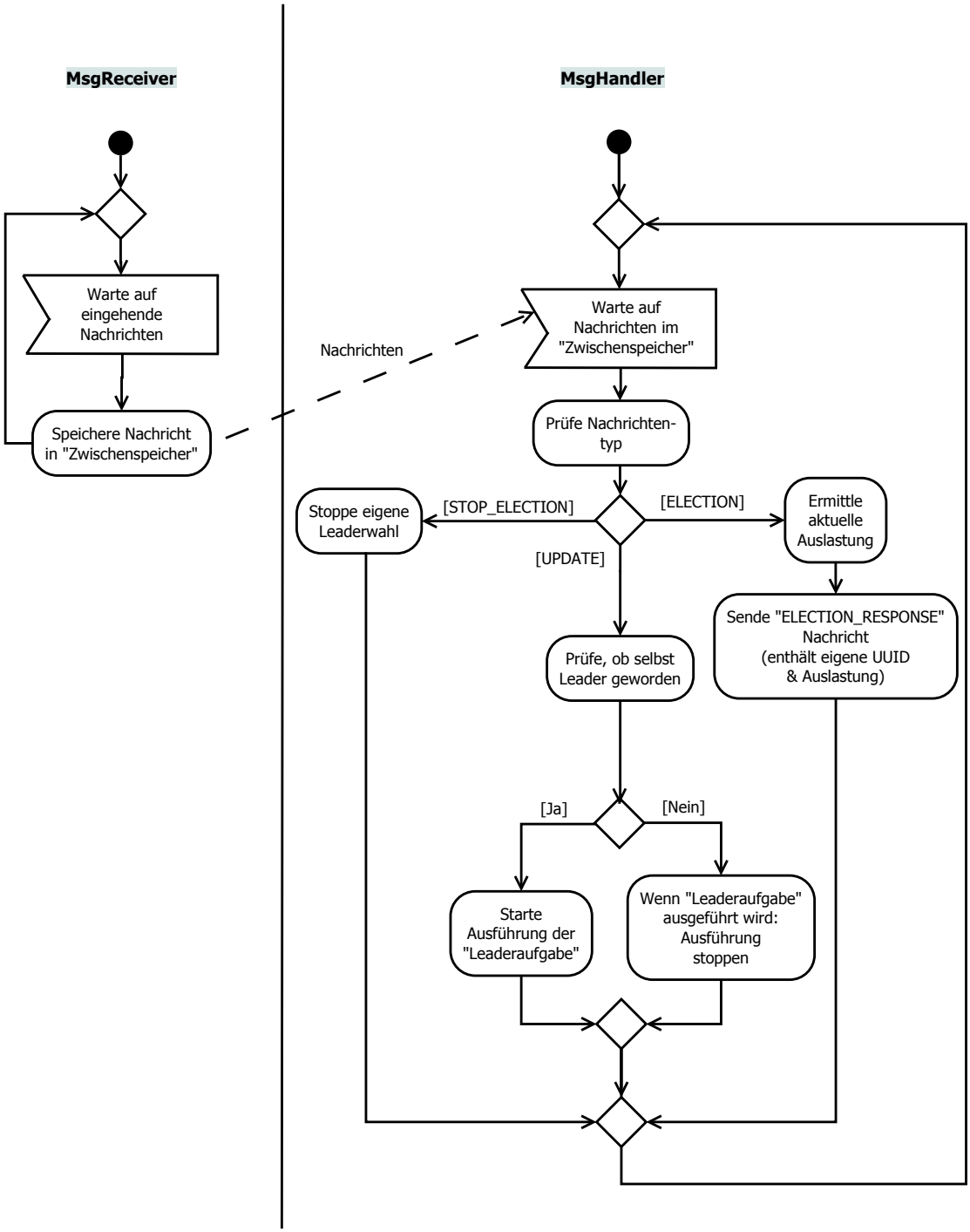


Abbildung 7.7: Verarbeitung eingehender Nachrichten

Wird diese Nachricht vor Eintreten einer Zeitschranke (Ablauf der Wartezeit) empfangen, wird der Timer zurück gesetzt und auf die nächste Benachrichtigung gewartet. Läuft hingegen die Wartezeit ab, so wird eine neue Leaderwahl gestartet, da davon ausgegangen wird, dass der Leader ausgefallen ist.

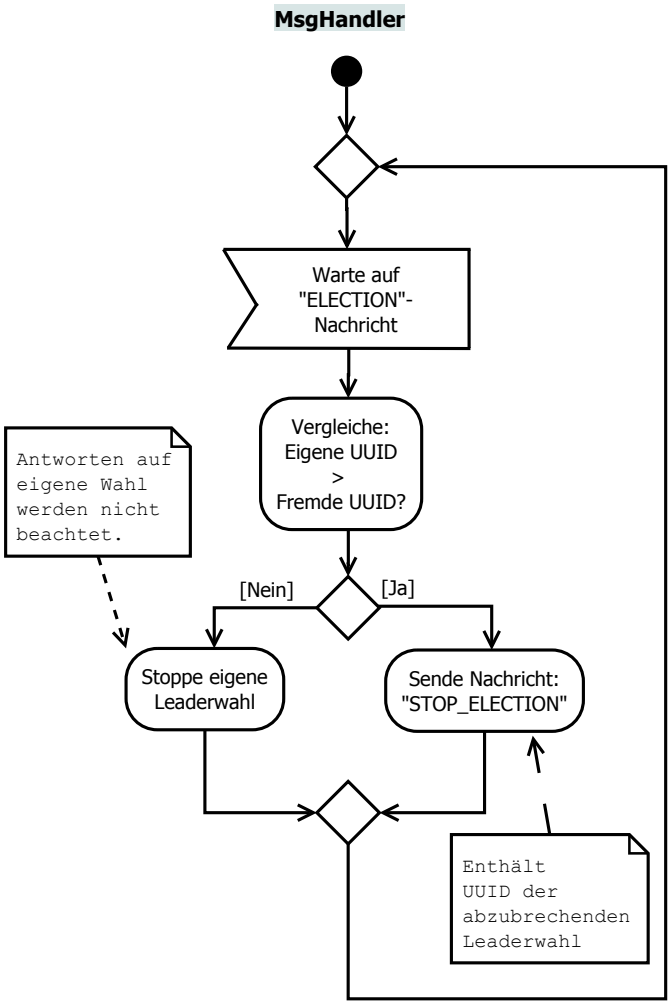


Abbildung 7.8: Auflösung möglicher Konfliktfälle

7.4.3 Implementierung

In diesem Abschnitt wird die Umsetzung des oben beschriebenen Leaderwahlalgorithmus beschrieben. In Abbildung 7.10 wird die Implementierung anhand eines UML Klassendiagramms verdeutlicht. Um den Ablauf der Leaderwahl zu verdeutlichen, wird zunächst die Klasse *LeaderManagement* beschrieben. Diese Klasse stellt den Einstiegspunkt in die Leaderwahl dar. Dies bedeutet, dass beim Verteilen von neuen Applikationen durch das SDRM eine Instanz dieser Klasse erzeugt werden muss, da diese sämtliche Verwaltungsaufgaben für die Leaderwahl übernimmt. Nach dem Erzeugen werden durch die Instanz dieser Klasse automatisch alle nötigen Informationen gesammelt und gespeichert. Nötige Informationen sind z.B. die eigene UUID oder die aktuelle Auslastung. Anschließend wird die Leaderwahl angestoßen, indem ein Objekt der Klasse *LeaderElection* erzeugt wird. In dieser Klasse sind alle Abläufe der eigentlichen Leaderwahl implementiert. Wie bereits beschrieben, wird u.a. parallel auf eingehende Nachrichten reagiert. Dies wird von der Klasse *MsgReceiver* übernommen. Aus Effizienzgründen werden die Nachrichten von dieser Klasse lediglich in einem Puffer gespeichert, damit keine Nachrichten verloren gehen. Die eigentliche Auswertung der gesammelten Nachrichten wird von einer *MsgHandler*-Instanz übernommen. Diese Instanz reagiert auf Nachrichten der Typen:

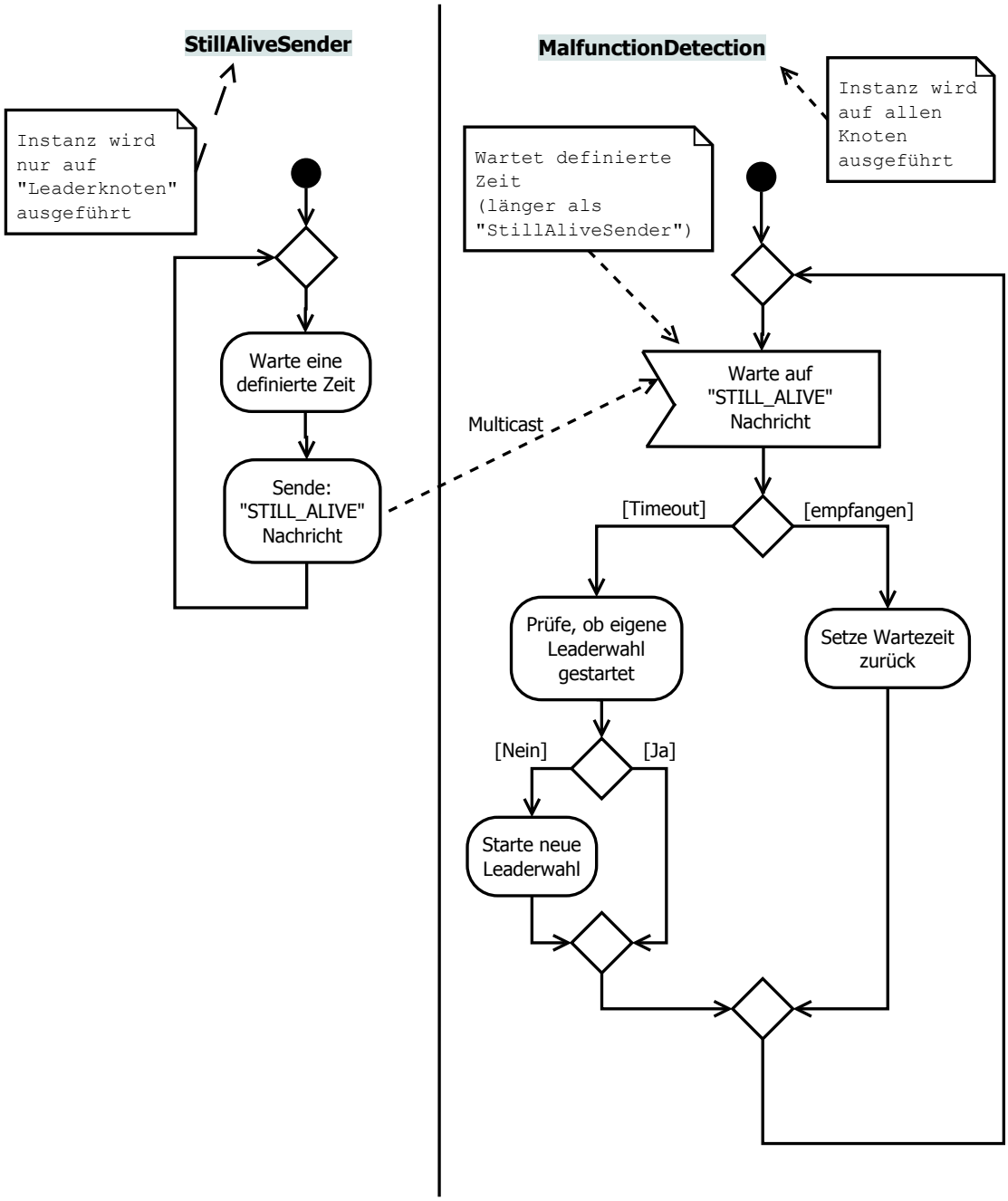


Abbildung 7.9: Ausfallkontrolle

ELECTION, *STOP_ELECTION* und *UPDATE*. Alle anderen Nachrichten werden von spezialisierten Mechanismen empfangen und behandelt. Aufgrund der Vielzahl an Nachrichten soll so eine zeitnahe und korrekte Bearbeitung sicher gestellt werden.

Ausfallsicherung Um das Ausfallen eines Leader wie beschrieben festzustellen, wird beim Start des Wahl-Algorithmus vom *LeaderManagement* ein Thread vom Typ *MalfunctionDetection* (siehe Klassendiagramm in Abbildung 7.10) erzeugt. Dieser Thread wartet auf *STILL_ALIVE*-

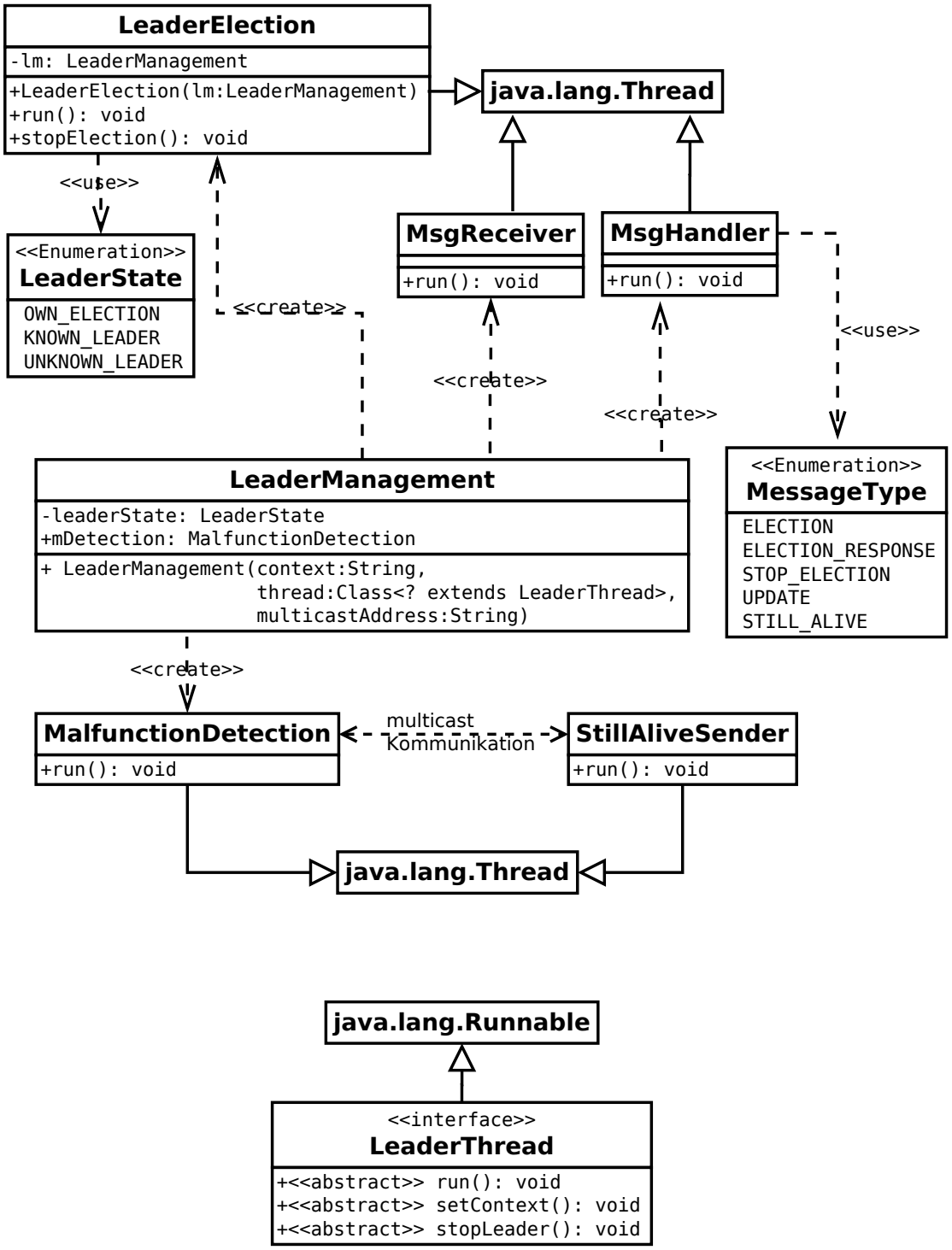


Abbildung 7.10: Klassendiagramm der Leaderwahl-Implementierung

Nachrichten des aktuellen Leaders. Der Leader-Knoten sendet diese Nachrichten mit Hilfe einer Instanz vom Typ *StillAliveSender*. Um mit diesen Nachrichten keine Verzögerung der anderen Abläufe zu bewirken, werden die *STILL_ALIVE*-Nachrichten intern über einen eigenen Socket-Port versendet. Ein Socket-Port dient lediglich der Unterscheidung der einzelnen Nachrichten

innerhalb einer Multicastgruppe. Somit wird diese Kommunikation des Leaders mit allen anderen Knoten isoliert durchgeführt.

Ergänzungen Wie in Abbildung 7.10 dargestellt ist, existieren weitere Klassen. Diese sind u.a. die Enumeration *MessageType*, mit dessen Hilfe die einzelnen Nachrichtentypen differenziert werden. Die Klasse *LeaderState* dient dazu, den Zustand des Knotens im Kontext der Leaderwahl zu repräsentieren. Der Zustand gibt an, dass ein Knoten eine Leaderwahl gestartet hat, den aktuellen Leader kennt, oder dass der Leader derzeit unbekannt ist. Damit eine Applikation als Leader gewählt werden kann, muss diese die Java-Schnittstelle *LeaderThread* implementieren. Diese Schnittstelle schreibt unter anderem vor, dass diese Klasse als selbstständiger Thread ausführbar sein muss. Weiterhin muss der eigentliche Ablauf der Applikation in der Methode *run()* implementiert werden, da diese zum Starten der Applikation aufegerufen wird. Auch muss eine Methode *stopLeader()* existieren. Diese Methode wird vom Leadermanagement verwendet, um den laufenden LeaderThread zu stoppen.

Kommt es im laufenden Betrieb zu einer Auftrennung des Netzwerks (Netsplit), ist also das Netz gestört, so kann die implementierte Leaderwahl diese Ausnahmesituation behandeln. Da für eine Applikation nur in einem der entstandenen Teilnetze ein Leader vorhanden ist, werden im Teilnetz ohne Leader keine *STILL_ALIVE* Nachrichten eines Leaders verschickt. Nach Ablauf der Zeitschranke in einer Instanz der Klasse *MalfunctionDetection* wird dieser Fehler erkannt und eine neue Leaderwahl gestartet. Somit können beide Teilnetze ihre Aufgaben weiter durchführen. Sollten nach dem Zusammenführen der beiden Teilnetze beide Leader parallel weiter laufen, so wird dies anhand der versendeten *STILL_ALIVE*-Nachrichten erkannt. In diesen ist die UUID des Leaders enthalten, welche von einem empfangenden Knoten mit der gespeicherten UUID verlgichen wird. Sollte hierbei festgestellt werden, dass diese UUIDs nicht zusammen passen, wird eine neue Leaderwahl ausgelöst.

7.5 Watchdog

Das Watchdog-System hat die Aufgabe, die Verfügbarkeit von Applikationen innerhalb der Cloud zu garantieren. Dazu ist eine ständige Überwachung aller Applikationen nötig. Sobald eine Applikation oder ein Knoten ausfällt, auf dem die Applikation angeboten wird, müssen die ausgefallenen Applikationen auf einem anderen Knoten wieder gestartet werden. Zur Umsetzung dieser Aufgabe wird ein Konzept mit zwei getrennten Komponenten genutzt, die mit den Namen *Local Watchdog* und *Global Watchdog* bezeichnet werden. Der *Local Watchdog* ist dafür verantwortlich, dass Applikationen (JAR-Dateien, siehe Abschnitt 7.1) oder einzelne Webservices in Form von *Target Services* (siehe Abschnitt 6.4.1) bei einem Absturz auf dem gleichen Knoten neugestartet werden. Dies löst die Problematik allerdings nicht vollständig, da ein Knoten als Ganzes ebenfalls ausfallen oder vom Netzwerk getrennt werden kann. Deshalb überwacht der *Global Watchdog* alle vorhandenen Knoten. Die genaue Reaktion des Global Watchdog wird in Unterabschnitt 7.5.2 beschrieben, während die Details zum Local Watchdog im folgenden Unterabschnitt 7.5.1 erläutert werden.

7.5.1 Local Watchdog

Zur Überwachung der Applikationen bzw. Webservices wird auf jedem Knoten eine *Local-Watchdog*-Instanz gestartet. Bei Webservices wird deren Erreichbarkeit in einem periodischem Abstand überprüft. Stellt der *Local Watchdog* fest, dass ein Webservice nicht reagiert, muss er einen Neustart des ausgefallenen Webservice initiieren. Dieses Konzept wird im Folgenden als *Webservice-Überwachung* bezeichnet. Für diese Form der Überwachung sind keine Veränderungen am Quelltext der zu überwachenden Webservices nötig.

Alternativ kann eine Applikation, die auf der FiLeCC-Plattform ausgeführt wird und die Schnittstelle `MonitoredFiLeCCApplication` implementiert, direkt überwacht werden. Die Überwachung ist deshalb direkt, weil sie als ein Methodenaufruf über die vom AppServer bereitgestellte Instanz der Applikation erfolgt (siehe Abschnitt 7.1). Dieses Konzept wird im Folgenden *Instanz-Überwachung* genannt. Dazu bietet die Schnittstelle `MonitoredFiLeCCApplication` die Methode `alive()` an, die vom *Local Watchdog* periodisch aufgerufen wird. Der wesentliche Unterschied zum vorher vorgestellten Verfahren der Webservice-Überwachung liegt darin, dass nicht nur die Erreichbarkeit einer Schnittstelle überprüft wird, sondern die Applikation selbst die Verantwortung trägt, ihren Status korrekt zurückzuliefern. Beide Überwachungsverfahren werden in den folgenden Unterabschnitten detailliert erläutert.

Webservice-Überwachung

Für die Webservice-Überwachung benötigt der *Local Watchdog* die Information, welche Webservices (*Target Services*) überwacht werden sollen. Diese Information erhält der *Local Watchdog* beim Start eines Webservice automatisch: laut der Spezifikation von DPWS [DM09] bzw. genauer WS-Discovery [MK09] müssen sich neue *Target Services* nach dem Start mittels Multicast mit einer `HELLO`-Nachricht bei den anderen Teilnehmern melden. In dieser `HELLO`-Nachricht ist die Endpunktreferenz des entsprechenden *Target Service* enthalten, über die ein Zugriff auf den Webservice möglich ist. Der Aufbau einer Endpunktreferenz ist in WS-Addressing [GHR06] festgelegt. Der *Local Watchdog* wartet auf diese `HELLO`-Nachrichten und überprüft, ob die Endpunktreferenz dem lokalen System entspricht. Dies wird durch einen Vergleich der lokalen IP-Adressen mit der IP-Adresse realisiert, die in der Endpunktreferenz enthalten ist. Die Überwachung des Webservice durch den *Local Watchdog* wird gestartet, wenn der Webservice auf dem gleichen Knoten wie der *Local Watchdog* ausgeführt wird. Damit nur Webservices überwacht werden, die nicht das Instanz-Überwachungsverfahren nutzen, muss der Webservice speziell von Entwickler gekennzeichnet werden. Dies erfolgt über Metadaten, die vom *Local Watchdog* für jeden Target Service (z. B. ein DPWS-Gerät) abgerufen werden können. Die Metadaten-Felder haben einen Namen und einen Wert, der zwecks Internationalisierung in mehreren Sprachen angegeben werden kann. Diese Eigenschaft der Metadaten-Felder wird für die Kennzeichnung zweckentfremdet. Zur Kennzeichnung muss der Entwickler innerhalb der Webservice-Metadaten in das Feld mit dem Namen `dpws:ModelName` und der Sprache `filecc` eine beliebige Zeichenkette eingetragen. Die Sprache `filecc` wird verwendet, um Kollisionen mit existierenden Sprachen zu vermeiden. Findet der *Local Watchdog* im Feld `dpws:ModelName` in der Sprache `filecc` eine Zeichenkette, so wird die Überwachung des *Target Services* gestartet. Webservices ohne diesen Eintrag werden nicht überwacht. Für den Fall, dass ein Webservice regulär beendet wird, sieht WS-Discovery den Versand einer optionalen `BYE`-Nachrichten vor. Diese Nachrichten müssen in FiLeCC für eine korrekte Abmeldung beim *Local Watchdog* verpflichtend gesendet werden. Ansonsten wird ein Ausfall registriert und der Webservice erneut gestartet. Empfängt ein

Local Watchdog eine solche `BYE`-Nachricht, so überprüft er, ob der Absender von ihm überwacht wird. Wenn dies der Fall ist, dann wird die Überwachung beendet.

Die Überwachung selbst wird durch das periodische Senden einer `Probe`-Nachricht realisiert, die ebenfalls in WS-Discovery [MK09] definiert ist. Antwortet der Webservice, so wird davon ausgegangen, dass er korrekt ausgeführt wird. Dies basiert auf der Annahme, dass der Webservice selbst überprüft, ob er korrekt ausgeführt wird und nur in diesem Fall antwortet. Stellt der *Local Watchdog* fest, dass ein Webservice nicht mehr reagiert, muss dieser neu gestartet werden. Diese Funktionalität bietet die *SDRM*-Komponente (siehe Abschnitt 7.2) als Webservice an.

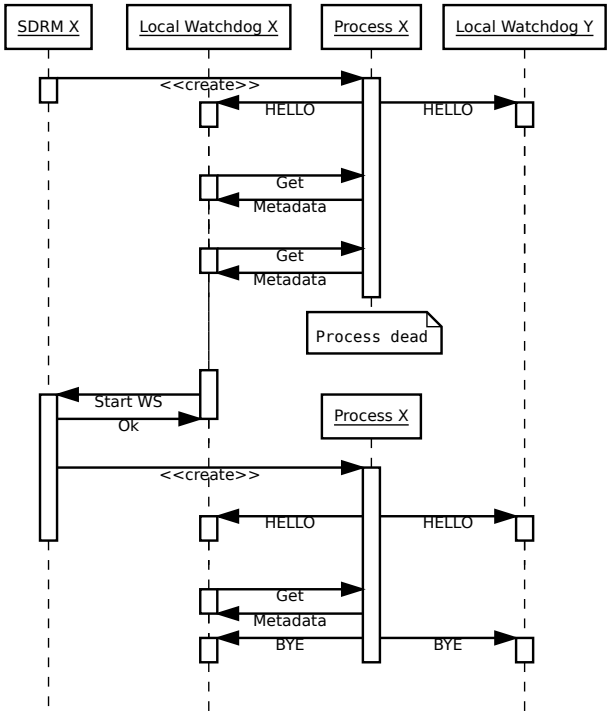


Abbildung 7.11: *Local Watchdog* – Sequenzdiagramm zur Überwachung eines Webservice. Die Buchstaben X und Y benennen die Knoten, auf denen der Webservice ausgeführt wird.

Abbildung 7.11 zeigt zusammenfassend ein Sequenzdiagramm, in dem die Überwachung eines Webservice durch den *Local Watchdog* dargestellt ist. Die Buchstabensuffixe *X* und *Y* bezeichnen die Knoten, wobei identische Buchstaben für identische Knoten stehen. Es wird davon ausgegangen, dass sich die Knoten *X* und *Y* in einer Multicast-Gruppe befinden, sodass der *Local Watchdog* auf Knoten *Y* die `HELLO`- und `BYE`-Nachrichten des Prozesses auf Knoten *X* empfängt. In dem konkreten Fall ignoriert er diese Nachrichten, da er nicht für die Prozesse auf Knoten *X* zuständig ist.

Instanz-Überwachung

Bei der Instanz-Überwachung besteht die Pflicht des Applikationsentwicklers die Java-Schnittstelle `MonitoredFiLeCCApplication` und damit insbesondere die Methode `alive()` zu implementieren. Innerhalb dieser Methode muss geprüft werden, ob die Applikation voll funktionsfähig ist. Im einfachsten Fall kann überprüft werden, ob der Thread zur Ausführung der Applikation

(falls vorhanden) noch ausgeführt wird. Werden in erster Linie Webservices angeboten, kann z. B. deren Erreichbarkeit überprüft werden. Welche Prüfverfahren verwendet werden, wird vom Entwickler der Applikation selbst entschieden. Dieser hat i. d. R. ein genaues Bild darüber, wie die Applikation am besten überprüft werden kann. Über den booleschen Rückgabewert von `alive()` wird dem *Local Watchdog* mitgeteilt, ob ein Neustart erforderlich ist (`alive() == false`) oder nicht (`alive() == true`). Die `alive()`-Methoden werden aus Sicherheitsgründen in einem separaten Thread ausgeführt. Dies muss der Entwickler der Applikation gegebenenfalls berücksichtigen. Insbesondere gibt das Konstrukt `Thread.currentThread()` nicht den Thread der Applikation zurück, sondern einen isolierten Thread, der nur zur Ausführung von `alive()` verwendet wird. Ein Informationsaustausch zwischen Applikationsinstanz und dem `alive()`-Thread kann über Klassenvariablen umgesetzt werden. In diesem Fall muss der Entwickler auf geeignete Synchronisationsmechanismen zurückgreifen.

Der Aufruf der Methode `alive()` durch den *Local Watchdog* erfolgt nur dann, wenn die Instanz der Applikation mit Hilfe der Methode `start()` hochgefahren wird. Dies kann bei der Implementierung von `alive()` vorausgesetzt werden. Die Ausführung der Methode wird durch einen sogenannten Java `TimerTask` realisiert. Das heißt insbesondere, dass maximal ein `alive()`-Aufruf zur selben Zeit durchgeführt wird. Um nachfolgende Aufrufe nicht zu verzögern, darf `alive()` keine Arbeiten durchführen, die längere Zeit blockieren, sondern muss möglichst schnell den Programmfluss beenden.

Implementierung

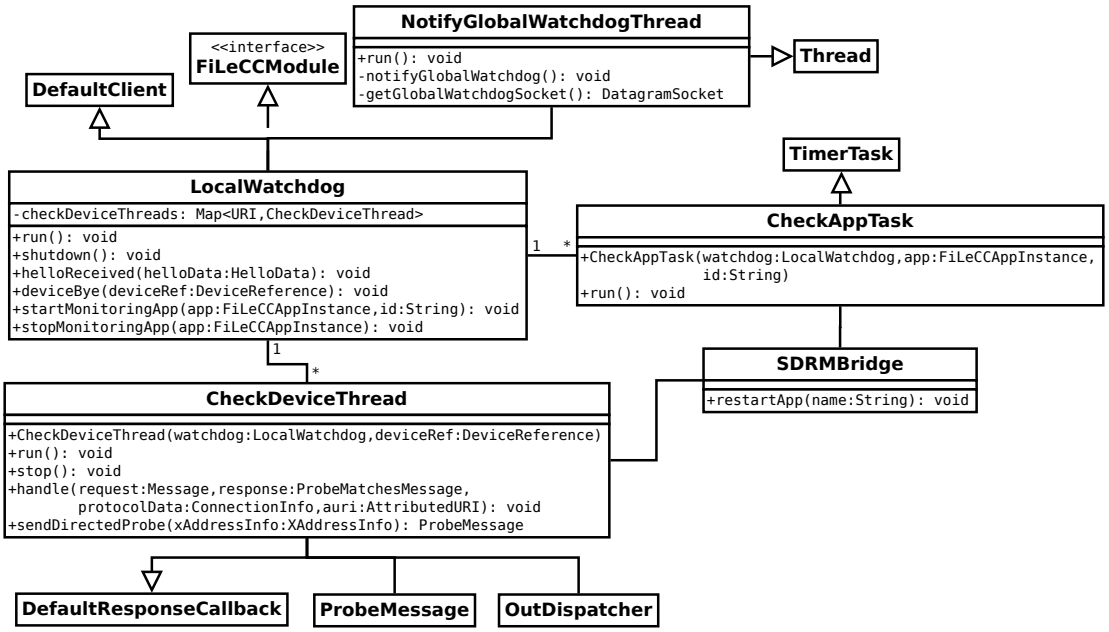


Abbildung 7.12: *Local Watchdog* Klassendiagramm

Abbildung 7.12 zeigt die Implementierung des *Local Watchdog*, die aus den fünf Klassen `LocalWatchdog`, `CheckDeviceThread`, `CheckAppTask`, `SDRMBridge` und `NotifyGlobalWatchdogThread` besteht.

Beim *Local Watchdog* handelt es sich um eine FiLeCC-Komponente, die vom AppServer erzeugt und gestartet wird. Die Verarbeitung beginnt in der Methode `run()` der Klasse `LocalWatchdog`. Die Klasse `NotifyGlobalWatchdogThread` hat die Aufgabe, ein periodisches Signal in Form eines UDP-Pakets an den *Global Watchdog* zu senden. Der Inhalt des Pakets ist lediglich die UUID des Absenderknotens, damit der *Global Watchdog* die Pakete eindeutig zuordnen kann. Das genaue Konzept des *Global Watchdog* erläutert der nachfolgende Abschnitt.

Durch das Erben von der Klasse `DefaultClient` in der Klasse `LocalWatchdog` können die Methoden zur Verarbeitung von `HELLO`- und `BYE`-Nachrichten überschrieben werden, die für die Webservice-Überwachung benötigt werden. In der Methode `helloReceived()` wird durch Aufruf der Methode `isLocalDevice()` überprüft, ob der *Local Watchdog* für diesen Webservice zuständig ist. Ist dies der Fall, wird eine neue Instanz der Klasse `CheckDeviceThread` erzeugt und in eine Hashtabelle eingefügt. Letzteres ist nötig, da beim Empfang einer `BYE`-Nachricht der Thread aus der Tabelle herausgesucht und über dessen Methode `stop()` beendet werden muss. Die Klasse `CheckDeviceThread` erzeugt in einer Schleife Instanzen der Klasse `ProbeMessage`, die über den `OutDispatcher` per Unicast an den zu überwachenden Webservice versendet werden. Antworten auf die `ProbeMessage` signalisieren, dass der Webservice noch aktiv ist und werden über die `handle()`-Methode verarbeitet, die aus der Klasse `DefaultResponseCallback` geerbt wird. Bleiben die Antworten aus, so wird die Methode `restartApp()` der Klasse `SDRMBridge` verwendet, um einen Neustart des Webservice über die *SDRM*-Komponente anzufordern.

Die Instanz-Überwachung kann durch das *SDRM* gestartet werden, indem die öffentliche Methode `startMonitoringApp()` der Klasse `LocalWatchdog` aufgerufen wird. Entsprechend wird die Überwachung durch Aufruf der Methode `stopMonitoringApp()` wieder beendet. Für jede überwachte Instanz wird ein `CheckAppTask` erzeugt. Diese `TimerTask`-Instanzen können so konfiguriert werden, dass sie in periodischem Abstand ihre `run()`-Methode aufrufen. Innerhalb der *run()*-Methode wird die *alive()*-Methode der Instanz aufgerufen. Ist der Rückgabewert der Methode `false`, so wird ein Neustart der Instanz über die Methode `restartApp()` der Klasse `SDRMBridge` beim *SDRM* angefordert.

Durch die Webservice- und Instanzüberwachung wird sichergestellt, dass alle auf der Plattform ausgeführten Programme bei einem Ausfall automatisch neugestartet werden. Allerdings ist ein Ausfall des kompletten Knotens (z. B. durch Netzwerkprobleme oder Hardwarefehler) möglich. Der *Global Watchdog* ermöglicht es bei einem Knotenausfall die Verfügbarkeit der betroffenen Applikationen weiterhin zu garantieren.

7.5.2 Global Watchdog

Die Aufgabe des *Global Watchdog* ist die Überwachung aller verfügbaren *Local-Watchdog*-Instanzen. Der *Global Watchdog* bietet einen Webservice an, über den die *Local-Watchdog*-Instanzen die Netzwerkadresse des *Global Watchdog* ermitteln können. Diese senden während ihrer Ausführungsdauer periodisch das im Abschnitt 7.5.1 beschriebene UDP-Paket. Ein einzelnes UDP-Paket erzeugt eine geringe Netzwerkbelastung, wodurch eine hohe Anzahl von Knoten über diesen Mechanismus überwacht werden können. Werden die Pakete eines *Local Watchdog* für eine bestimmte Zeitspanne nicht empfangen, muss davon ausgegangen werden, dass der *Local Watchdog* nicht mehr reagiert bzw. der gesamte Knoten ausgefallen ist. Als erste Maßnahme versucht der *Global Watchdog* daraufhin die *SDRM*-Komponente auf dem Knoten zu

erreichen, auf dem der *Local Watchdog* nicht mehr reagiert. Ist dies möglich, wird über diese Komponente der *Local Watchdog* neugestartet. Kann die Komponente allerdings nicht erreicht werden, wird ein Ausfall des Knotens angenommen. In diesem Fall wird ein Hilferuf (siehe Abschnitt 7.3) an die *UNO* gesendet, sodass die beweglichen Webservices, die auf dem Knoten ausgeführt werden, auf anderen Knoten erneut gestartet werden.

Es existiert stets nur ein Knoten im Netzwerk, der die Aufgabe des *Global Watchdog* übernimmt. Um dies zu garantieren, wird das in Abschnitt 7.4 beschriebene Leaderwahlverfahren für den *Global Watchdog* verwendet. Es stellt sicher, dass keine zweite Instanz des *Global Watchdog* dauerhaft im Netzwerk existiert. Ist dies z. B. durch Verbinden von zwei vorher getrennten Netzen kurzzeitig der Fall, erkennt der Leaderwahlalgorithmus (siehe Abschnitt 7.4) die Kollision und startet eine neue Wahl. Nach der Wahl steht fest, welche Instanz weiterhin ausgeführt werden soll. Die Instanz, die gestoppt wird, übergibt eine Liste mit allen derzeit überwachten Knoten an die gewählte Instanz und beendet sich daraufhin selbst. Dadurch werden auch Knotenausfälle erkannt, die während des Wahlzeitraums stattgefunden haben.

Implementierung

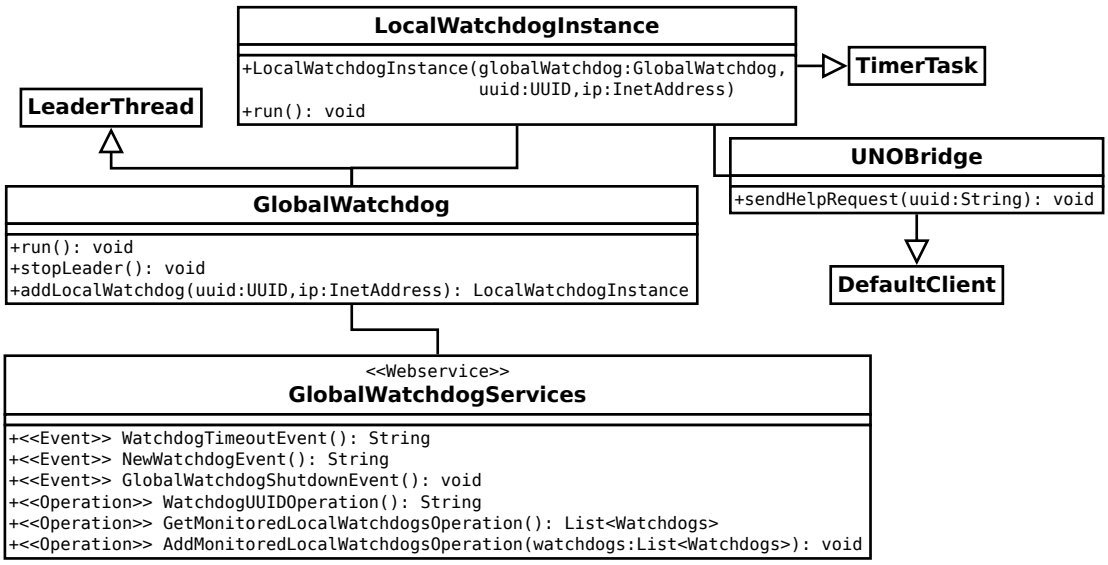


Abbildung 7.13: Global Watchdog Klassendiagramm

In Abbildung 7.13 ist das Klassendiagramm des *Global Watchdog* dargestellt. Der *Global Watchdog* ist ein `LeaderThread`, der gegebenenfalls vom Leaderwahlverfahren gestartet (`run()`) oder gestoppt (`stopLeader()`) wird. Im Wesentlichen besteht die Methode `run()` aus einer Schleife, die eingehende UDP-Pakete den *Local-Watchdog*-Instanzen zuordnet. Für jede *Local-Watchdog*-Instanz existiert eine Instanz der Klasse `LocalWatchdogInstance`, die einen Zähler enthält, welcher fortlaufend verringt wird. Durch Empfang eines UDP-Pakets wird die Zähler-Variable auf den Wert einer vorher vom Administrator festgelegten Zeitschranke gesetzt. Erreicht der Zähler Null, so hat der *Local Watchdog* nicht innerhalb der Zeitschranke geantwortet und die `run()`-Methode von `LocalWatchdogInstance` signalisiert den Ausfall dieser *Local-Watchdog*-Instanz. Ist dies der Fall, so wird über die Methode `sendHelpRequest()` der Klasse `UNOBridge`

eine Nachricht an die UNO gesendet. Durch die Weitergabe der in den UDP-Paketen enthaltenen UUID kann der Knoten auch bei der *UNO* eindeutig identifiziert und ersetzt werden.

Zusätzlich bietet der *Global Watchdog* Webservices in der Klasse `GlobalWatchdogServices` an, die zur Administration und von anderen FiLeCC-Komponenten verwendet werden. Über die Operation `WatchdogUUIDOperation` kann die UUID des Knotens abgefragt werden, auf dem der *Global Watchdog* derzeit ausgeführt wird. Dies wird insbesondere für die Netzwerktopologie benötigt, die durch Suchen und Aufrufen dieser Methode den *Global Watchdog* einem Knoten zuordnen kann. Die Operation `GetMonitoredLocalWatchdogsOperation` gibt eine Liste mit allen UUIDs der überwachten Knoten zurück und dient Administrationszwecken. Zudem wird die Operation `AddMonitoredLocalWatchdogsOperation` angeboten, über die manuell Knoten in die Überwachung aufgenommen werden können. Zusätzlich dient diese Operation zur Konsolidierung mehrerer *Global-Watchdog*-Instanzen in einem Netzwerk (z. B. nach Zusammenführen von zwei Teilnetzen). In diesem Fall wird sie verwendet, um eine Liste der überwachten Knoten an eine Instanz zu übergeben und die andere Instanz zu beenden. Das Ereignis `NewWatchdogEvent` signalisiert das Auftauchen eines neuen *Local Watchdog* im Netzwerk, sobald dessen Überwachung gestartet wird. Wenn ein *Local Watchdog* das Netzwerk (durch einen Timeout) verlässt, wird das Ereignis `WatchdogTimeoutEvent` gesendet. Das `GlobalWatchdogShutdownEvent` wird ausgelöst, wenn die *Global-Watchdog*-Instanz durch die Leaderwahl beendet wird. Dieses Ereignis wird von den *Local-Watchdog*-Instanzen empfangen, die danach die Suche nach einer neuen *Global-Watchdog*-Instanz starten.

Durch das beschriebene Verfahren kann der Ausfall eines Knotens vom System erkannt und behandelt werden. Dabei dient der *Global Watchdog* lediglich der Erkennung eines Knotenausfalls, während die *UNO* für die Behandlung eines Ausfalls zuständig ist.

7.6 Lastverteilung

Zu den grundlegenden Aufgaben einer Cloud-Umgebung gehört die Verringerung von Entwicklungskomplexität (siehe [TBH11]). Dies lässt sich u. a. durch das Verbergen von Ausführungsdetails erreichen: Berechnungsaufgaben sollen ohne Systemkenntnis an die Cloud übergeben werden. Der Nutzer benötigt keine Informationen über Architektur und Netztopologie des zugrunde liegenden Systems, die über die API-Funktionen hinausgehen. Dafür muss er sich darauf verlassen können, dass das System automatisch mit Ausfällen und Lastspitzen umgehen kann und die aktive Software sinnvoll auf alle verfügbaren Knoten verteilt wird, sofern das Cloud-System auf mehr als einem Knoten agiert. Für den Umgang mit Ausfallsicherheit wird das Watchdog-System eingesetzt, welches in Abschnitt 7.5 beschrieben ist. Für den Umgang mit Belastungsschwankungen sind verschiedene Ansätze denkbar, sodass die Auswahl einer eingehenden Untersuchung bedarf. Die möglichen Ansätze sowie die dazugehörigen Arbeitsergebnisse werden in diesem Unterkapitel zusammengefasst.

FiLeCC nutzt zum einen verteilte Jobs in Form von GridGain-Tasks. Dabei stellt GridGain bereits Mechanismen zur Lastverteilung und zum Umgang mit ausgefallenen Knoten bereit und bedarf daher keiner weiteren Anpassung. Andererseits wird innerhalb von FiLeCC Software betrieben, deren Funktionalität in Form von Webservices erreichbar ist. Diese Software ist zunächst fest an den Knoten gebunden, auf dem sie ausgeführt und über den sie aufgerufen werden kann. Wenn an einen Webservice mehr Anfragen gestellt werden, als ein einzelner Knoten verarbeiten kann,

so muss zur Umsetzung einer Lastverteilung der Service auf weitere Knoten repliziert werden und die Anfragen an diesen müssen auf alle Instanzen verteilt werden.

Die Replikation eines Webservices auf einen neuen Knoten ist eine vergleichsweise aufwändige Operation, da sie durch das Kopieren der Ausführungsdateien hohe Netzlast erzeugt. Ziel ist es daher, dies so selten wie möglich durchzuführen. Die gleichmäßige Verteilung von Anfragen an alle Instanzen ist dagegen eine sehr häufig auftretende Operation und muss dementsprechend schnell und ressourcenschonend durchgeführt werden können. Dies motiviert die Verwendung eines zweischichtigen Verfahrens: Die Replikation entspricht einer *Long-Term-Scheduling*-Strategie (vgl. [Tan07]), d. h. der Behandlung von langsam ablaufenden Belastungsschwankungen (im Unterschied zum *Short-Term-Scheduling*, bei welchem kurze Lastspitzen vermieden werden sollen). Letzteres wird durch die UNO implementiert, welche in Abschnitt 7.3 beschrieben ist. Dem gegenüber steht die Vermeidung von Lastspitzen auf einzelnen Knoten, welche durch Anfragenverteilung auf alle verfügbaren Knoten mit dem entsprechenden Webservice sichergestellt wird. Der Entwurf eines passenden Mechanismus wird im Folgenden beschrieben.

Die Aufgabe des gesuchten Scheduling-Mechanismus ist zunächst einfach: Gegeben sei eine Liste von Knoten, auf denen der gewünschte Webservice angeboten wird. Wähle aus dieser Liste mit Hilfe einer sinnvollen Methode einen passenden Knoten aus. Mögliche Methoden können sich an Lastkennzahlen der Knoten orientieren, zufallsbasiert arbeiten oder nach dem *Round-Robin*-Verfahren (vgl. [Tan07]) vorgehen, d. h. die Liste der Webservices zyklisch durchlaufen. Da es mit der UNO bereits einen Mechanismus für langfristige Belastungsschwankungen gibt, muss das System sich im Sinne von *Short-Term-Scheduling* [Tan07] nur an der aktuellen Situation im System orientieren. Eine aktuelle Erfassung der Lastdaten aller Knoten bei jeder Anfrage an einen Webservice bedeutet eine unverhältnismäßige Netzbelastung und erzeugt eine zusätzliche Verzögerung, und kann daher keine sinnvolle Option sein. Um für eine gleichmäßige Auslastung zu sorgen, sind das *Round-Robin*-Verfahren und das zufallsbasierte Zuordnen daher die sinnvollste Auswahlmethoden.

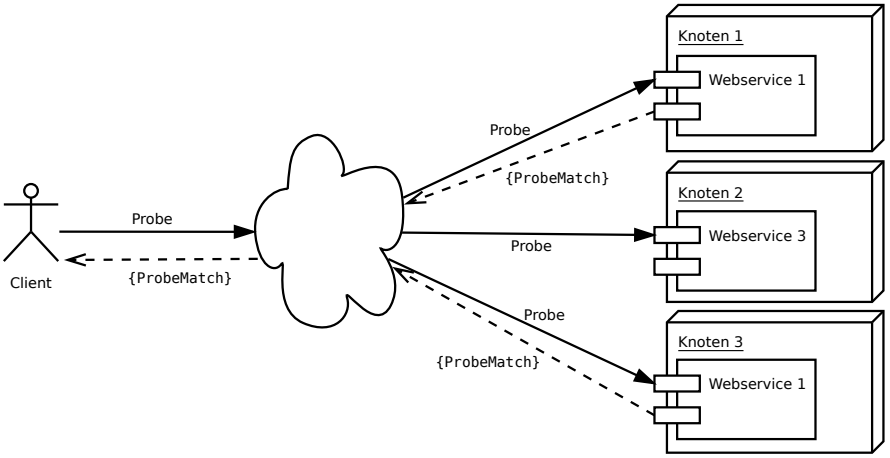


Abbildung 7.14: Webservicesuche mit WS-Discovery [MK09]. Die Wolke repräsentiert das unterliegende Netzwerk.

Für ein solches Schedulingverfahren gibt es in einer DPWS-konformen Umgebung verschiedene Möglichkeiten, die jeweils mit Vor- und Nachteilen verbunden sind. In Abbildung 7.14 ist der Ablauf einer Webservicesuche durch einen Client schematisch dargestellt. Der Client sendet zunächst eine `Probe`-Nachricht (vgl. Kapitel 4.3.3) an die entsprechende Multicast-Gruppe. Alle

passenden Webservices antworten dann per Unicast mit einer `ProbeMatch`-Nachricht. Daraus ergeben sich drei mögliche Ansatzpunkte für einen lastverteilenden Zuteilungsmechanismus:

Serviceseitig Jeder Webservice behandelt die eingehenden Anfragen nach einem bestimmten Zuteilungsverfahren.

Clientseitig Clients verteilen Anfragen auf vorhandene Services auf der Basis eines Zuteilungsmusters.

Netzebene Anfragen werden im Netz abgefangen und transparent auf Webservices verteilt.

Jedes dieser Verfahren bietet unterschiedliche Vor- und Nachteile, die gegeneinander abgewägt werden müssen, um die beste Option zu finden. Zwei Aspekte der Webservicesuche mit DPWS sind in diesem Zusammenhang von Bedeutung: Bei der Webservicesuche mit Hilfe von WS-Discovery antworten alle passenden Webservices auf `Probe`- und `Resolve`-Nachrichten (vgl. mit Kapitel 4.3.3), sodass der Client eine Liste aller verfügbaren Services erhält. Die Spezifikation von WS-Discovery (vgl. [MK09]) schreibt kein Auswahlverfahren vor, es ist also jedem Client freigestellt, an welche konkreten Instanzen er die eigentliche Anfrage stellt. Insbesondere haben Anfrageverteilungsmechanismen auf Webserviceseite damit keinen Einfluss auf die Entscheidungen der Clients, so dass serviceseitige Anfrageverteilung keine sinnvolle Lösung darstellt.

Weiterhin empfiehlt die WS-Discovery-Spezifikation (vgl. [MK09, S.16]) das Speichern von einmal gefundenen Webservices auf Clientseite. In der Praxis binden sich Clients somit fest an einmal gefundene Services und umgehen damit jeden Lastverteilungsmechanismus, der sich lediglich auf gleichmäßige `Probe`-Verteilung stützt. Eine Anfrageverteilung auf Netzebene ist daher ebenfalls nicht sinnvoll, da Clients sich nach einmaliger Zuteilung eines Webservices durch den Verteilmechanismus fest an den entsprechenden Webservice binden und anschließend keine Rücksicht auf die Auslastung des ausgewählten Webservice nehmen.

Die einzige sinnvolle Möglichkeit für eine Verteilung von Clientanfragen auf mehrere DPWS-konforme Webservices ist daher die Entwicklung einer Clientbibliothek, die Anfragen gleichmäßig über alle bekannten Webservices verteilt. Das konzipierte Verfahren wird im Folgenden Abschnitt näher erläutert.

7.6.1 Grundlagen der clientseitigen Anfragenverteilung

Bei der Suche nach Webservices gehen Clients laut der WS-Discovery-Spezifikation (vgl. [MK09]) in drei Schritten vor. Zuerst wird die Transportadresse des gesuchten Webservices mit Hilfe von `Probe`- und `Resolve`-Nachrichten sowie lokal gespeicherten Ergebnissen früherer Anfragen gesucht. Um die Zahl der Suchanfragen gering zu halten, werden im zweiten Schritt die Transportadressen gefundener Webservices für zukünftige Anfragen lokal gespeichert. Im dritten Schritt werden dann ein oder mehrere Webservices durch den Client ausgewählt und verwendet. Dabei gibt die Spezifikation kein Auswahlverfahren vor, so dass verschiedene Implementierungen unterschiedliche Auswahlstrategien verfolgen können. Um die Anfragelast gleichmäßig auf alle Webserviceinstanzen zu verteilen, müssen Clients bei der Auswahl eines Webservices in Schritt drei eine sinnvolle Auswahlstrategie verwenden. Nicht sinnvoll ist es, immer den selben zufällig gewählten Knoten zu verwenden, da es dann zu ungleichmäßigen Belastungen einzelner

Knoten kommen kann. Wie oben beschrieben ist das Round-Robin-Verfahren aus Sicht der Lastverteilung eine der sinnvollsten Auswahlmethoden.

Ein wesentlicher Vorteil der clientseitigen Anrageverteilung ist, dass alle notwendigen Informationen dem Client ohnehin vorliegen und die Anfrageverteilung genau dort stattfindet, wo die Anfrage erzeugt wird. Zudem ist sie einfach zu implementieren und bietet die Möglichkeit, durch eine Clientbibliothek einige der DPWS-Ausführungsdetails durch eine Client-API zu vereinfachen. Der Nachteil dieser Methode ist, dass sie eine spezielle Bibliothek für den Zugriff auf Services in der FiLeCC-Cloud erfordert.

7.6.2 Implementierung als Client-API

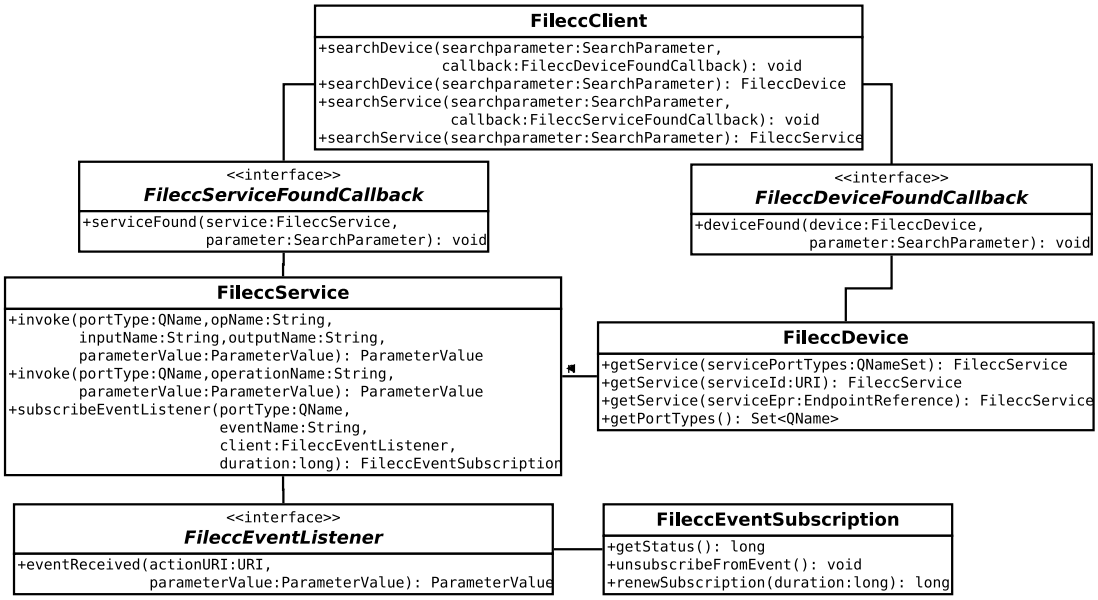


Abbildung 7.15: Klassendiagramm der Client-API

Benötigt wird eine Bibliothek, mit deren Hilfe Clients Webservices anhand ihres Namens sowie ihrer Ein- und Ausgabeparameter suchen und nutzen können. Die Bereitstellung einer solchen Bibliothek bietet zusätzlich die Möglichkeit, einige Funktionen der JMEDS-Client-API zu verbergen, die für den Einsatz in einer Cloud ohnehin nicht von Bedeutung sind. Es ist beispielsweise nicht sinnvoll, die UUID des Knotens eines Webservices abzufragen, da dieser auf mehreren Knoten gleichzeitig ablaufen und durch die Lastverteilung sogar jederzeit seine Position in der Cloud ändern kann. Andere DPWS-Funktionen müssen darüber hinaus modifiziert werden, um sie mit Webservices zu nutzen, die von FiLeCC verwaltet werden. Ein Beispiel ist die Registrierung und Behandlung von Events, da der Client bei der Registrierung nicht wissen kann, wie oft ein Service mit dem passenden Event derzeit in FiLeCC verfügbar ist und bei welchem er sich registrieren muss.

Alle oben genannten Vorgaben werden durch das `clientapi`-Paket implementiert, welches im Klassendiagramm in Abbildung 7.15 dargestellt wird. Die Suche nach neuen Webservices ist über die zentrale Klasse `FileccClient` sowohl synchron als auch asynchron möglich und ermöglicht

das Zwischenspeichern von einmal gefundenen Webservices im Hintergrund. Jeder Suche wird ein eindeutiges `FileccService`-Objekt zugeordnet, welches alle zu den Suchparametern passende Webservices kapselt und als Mediator (vgl. [GHJ+94]) fungiert. Alle Webserviceaufrufe werden über die `invoke`-Methode des entsprechenden `FileccService`-Objekts durchgeführt, welches die Anfragen nach dem *Round-Robin*-Verfahren auf alle bekannten Webservices verteilt. Mit der Methode `subscribeEventListener` der Klasse `FileccService` können Clients sich für DPWS-Events entsprechend der WS-Eventing Spezifikation registrieren. Empfangene Events von einem Webservice werden über das `FileccEventListener`-Interface an den Client weitergeleitet.

Entsprechend der DPWS-Spezifikation kann über die `FileccClient`-Klasse auch nach Devices gesucht werden. Analog zu Webservices werden gefundene Devices durch ein Mediatorobjekt vom Typ `FileccDevice` gekapselt. Vergleiche dazu auch die entsprechenden Klassen in Abbildung 7.15 unten.

7.7 Ressourcen Monitoring

Der Ressourcenmonitor ist eine lokale Komponente, die auf jedem Knoten der Cloud ausgeführt wird und die Aufgabe hat, die zur Verfügung stehenden Ressourcen dieses Knotens kontinuierlich zu überwachen. Dadurch wird ermöglicht, dass die Verteilung von Services innerhalb von FiLeCC effizient durchgeführt wird (siehe Abschnitt 7.2 und 7.3). Dazu werden kurze Abfrageintervalle definiert (durch den Admin konfigurierbar; vordefiniert sind 15sek.), in denen die freien Ressourcen gemessen und anschließend den anderen Modulen über eine verteilte Datenbank (siehe Abschnitt 7.9) bekannt gemacht werden. Die wesentlichen Kennzahlen für die Messung der freien Ressourcen sind die CPU-Geschwindigkeit, die prozentuale Leerlaufzeit der CPU, der freie Arbeitsspeicher und der freie Festplattenspeicher.

Das Klassendiagramm 7.16 stellt die Implementierung des FiLeCC Ressourcenmonitors dar.

Der Ressourcenmonitor wird mit einer frei konfigurierbaren Sekundenanzahl *s* instanziiert und startet im Konstruktor einen `MetricTask`, der von der Java Util-Klasse `TimerTask` erbt. Mit Hilfe von *Java-Events* wird der im Hintergrund wartende `MetricTask` anschließend alle *s* Sekunden angestoßen. Dabei werden die Lastdaten eines Knotens über eine Reader-Klasse ausgelesen und anschließend an den Ressourcenmonitor zurück geliefert. Der Reader ist in erster Linie über die Java-Schnittstelle `IRessourceReader` spezifiziert, die zwei essentielle Operationen definiert, welche das Aktualisieren und die Bereitstellung der Daten als `LoadData`-Klasse an den `MetricTask` realisieren. Anschließend werden die Lastdaten in die verteilte Datenbank geschrieben, um sie den anderen in der Cloud befindlichen Knoten zugänglich zu machen. Die gemessenen Lastdaten werden in der Modell-Klasse `LoadData` abgebildet.

In der Implementierung des `IResourceReader` ist es wünschenswert, die freien Ressourcen, soweit möglich, auf Betriebssystemebene auszulesen. Da dies mit Java nicht direkt möglich ist, wird eine Implementierung mittels *Java Native Interface* verwendet. Um dies zu ermöglichen ist die Implementierung des `IResourceReader` mit dem *Hyperic Sigar Framework* umgesetzt, welches eine vollständige Java-Schnittstelle anbietet. Ein weiterer Vorteil ist, dass der *Sigar*-Kern in C implementiert ist und alle erforderlichen Lastdaten zurückliefert. Es werden zudem alle gängigen Betriebssysteme (Linux, FreeBSD, Windows, Solaris, AIX, HP-UX und Mac OSX) und

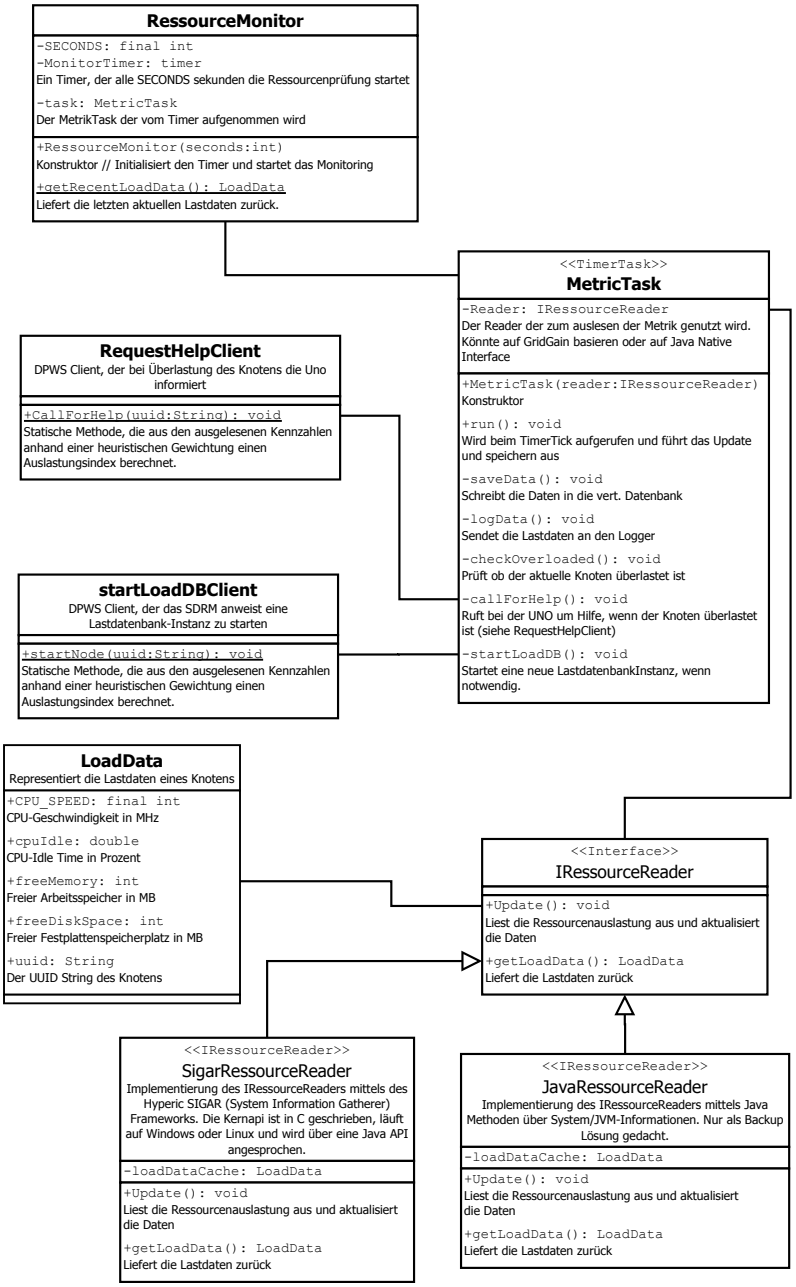


Abbildung 7.16: Klassendiagramm Ressourcenmonitor

Architekturen (x86 und x64) unterstützt, sodass die Interoperabilität des Ressourcenmonitors gewährleistet ist (weitere Details sind unter [VMw12a] ersichtlich). Stehen die *Sigar*-Bibliotheken auf einem System nicht zur Verfügung, wird auf eine reine Java-Implementierung zurückgegriffen. Diese Implementierung greift auf die von der Java Virtual Machine gelieferten Lastdaten zurück und tätigt basierend auf diesen Informationen Annahmen über den Zustand des gesamten Systems. Die auf diese Weise berechneten Werte sind ungenauer als die vom Betriebssystem gelieferten Daten, garantieren jedoch die Funktionalität des Systems ohne die zusätzlichen Bibliotheken des *Sigar Frameworks*. Bei Bedarf kann die Implementierung der `IRessourceReader`-Schnittstelle jederzeit ersetzt werden.

Nach dem Auslesen der Daten, nutzt der Ressourcenmonitor die Datenbank-Schnittstelle (siehe Abschnitt 7.9), um die Daten in die verteilte Datenbank einzutragen. Auf diese Weise werden die Daten den anderen Systemkomponenten zur Verfügung gestellt. Diese werden z. B. zum Verteilen von Software im System oder von der UNO (siehe Abschnittl 7.2 und 7.3) genutzt, um weniger ausgelastete Knoten zu finden. Ist noch keine Datenbank zur Verwaltung der Lastdaten in der Cloud gestartet, prüft der Ressourcenmonitor, ob es möglich ist auf dem eigenen Knoten eine Datenbank-Instanz zu starten und benachrichtigt das SDRM, sodass eine neue Datenbank erstellt wird.

Zusätzlich prüft der Ressourcenmonitor in jedem Intervall, ob der Knoten nahe seiner Auslastungsgrenze oder eventuell bereits überlastet ist. Wird in n aufeinander folgenden Intervallen ein gesetzter Schwellwert überschritten, wird die UNO vom Ressourcenmonitor über die Überlastung des Knotens informiert, so dass entsprechende Maßnahmen ergriffen werden können (siehe Abschnitt 7.3). Voreingestellt sind an dieser Stelle drei Intervalle, sodass eine zu schnelle Alarmierung der UNO durch eine einmalige sehr kurze Lastspitze auf dem Rechner unterbunden wird. Die Werte, die festlegen ab wann ein Knoten als überlastet betrachtet wird, können durch den Administrator des Systems angepasst werden. Weitere Details hierzu sind im Administratorenhandbuch zu finden, siehe Anhang B.

7.8 Nachbarschaftserkennung und Topologieberechnung

Im Gegensatz zu den in den vorherigen Unterkapiteln vorgestellten Systemkomponenten beschreibt dieses Unterkapitel eine Applikation für die Ausführungsplattform FiLeCC. Da der Fokus der Projektgruppe auf dem Anwendungsgebiet der logistischen Anlagen liegt, ist die Applikation ebenfalls auf dieses Anwendungsgebiet zugeschnitten und umfasst die Erstellung einer Anlagentopologie von einer modular aufgebauten Materialflussanlage. Für davon abweichende Anwendungsgebiete kann auf die Integration dieser Applikation verzichtet werden. Die Applikation stellt eine Beispielapplikation dar. Ihr Ziel ist aufzuzeigen, dass sowohl die Rechen- als auch Speicherressourcen der Cloud über die Ausführungsplattform FiLeCC von externen Applikationen genutzt werden können. Weiterhin dient die Implementierung dieser Beispielapplikation der Erfüllung eines in Kapitel 1 beschriebenen Minimalziels der Projektgruppe, das ein Softwaremodul zur Berechnung von Anlagentopologien fordert.

Dieses Unterkapitel ist dabei wie folgt gegliedert: In Abschnitt 7.8.1 werden die Rahmenbedingungen und Anforderungen aufgelistet. Daran anschließend werden in Abschnitt 7.8.2 zwei Konzepte zu möglichen Realisierungen gegenübergestellt, wovon eines von der Projektgruppe realisiert worden ist. Details zu der Realisierung beinhaltet Abschnitt 7.8.3. Den Abschluss dieses

Unterkapitels bildet Abschnitt 7.8.4 mit einem kurzen Ausblick auf mögliche Erweiterungen und Anknüpfungspunkte der vorgestellten Beispielapplikation.

7.8.1 Rahmenbedingungen & Anforderungen

Die Berechnung der Anlagentopologie einer Materialflussanlage folgt dem Konzept der im Grundlagenkapitel 4.5 vorgestellten Smart Neighbourhood Module (SNM) und leistet somit einen Beitrag für deren Realisierung. Daraus ergeben sich die im Folgenden vorgestellten Rahmenbedingungen. Die für die Topologieberechnung betrachteten Materialflussanlagen müssen aus vielen einzelnen Modulen bestehen, die nach dem Plug&Play-Prinzip miteinander verbunden sind und in der Lage sind, mit ihren benachbarten Modulen zu kommunizieren. Die Kommunikation, die den Austausch von Informationen bzgl. der Nachbarschaftsinformationen von direkt benachbarten Modulen beinhaltet, wird dabei durch eine *SNM-Platine* realisiert, die u. a. über einen Mikrocontroller verfügt. Eine detaillierte Beschreibung dieser SNM-Platine enthält Abschnitt 7.8.3. Die Entwicklung der SNM-Platine und Beispielapplikation hat sich an der Miniaturfördertechnik orientiert, die im Rahmen einer Projektarbeit am Lehrstuhl für Förder- und Lagerwesen entworfen worden ist (siehe [WHS+11], [VO10], [RRW12] und [FS10]). Diese besteht aus kleinen quadratischen Modulen mit einer Grundfläche von 300×300 mm, die zu einer Miniaturförderanlage kombiniert werden können. Dabei kann an jede Seite eines Moduls nur ein weiteres Modul angeschlossen werden, sodass ein Modul maximal vier benachbarte Module besitzt. Aus diesem Grund unterstützt sowohl die SNM-Platine als auch die Beispielapplikation nur maximal vier Nachbarn zu einem Modul. Damit die Beispielapplikation Zugriff auf die von den SNM-Platinen bereitgestellten Informationen hat, müssen diese mit den Knoten der FiLeCC-Cloud verbunden sein.

Die zentrale Anforderung an die Beispielapplikation besteht in der korrekten Auswertung und Verdichtung der lokalen Nachbarschaftsinformationen der einzelnen Module zu einer globalen Anlagentopologie. Änderungen an der Anlagentopologie sollen dynamisch und konfigurationsarm möglich sein. Für die Topologieberechnung bedeutet dies, dass sie Änderungen eigenständig erkennen und auf diese reagieren können muss. Diese Änderungen müssen des Weiteren zeitnah erkannt werden, um den zum Teil zeitkritischen Anforderungen von Materialflussanlagen gerecht zu werden. Neben dem regulären Hinzufügen und Entfernen von Modulen soll die Beispielapplikation auch auf Fehlersituationen wie z. B. den Ausfall einer Netzwerkverbindung reagieren können und die davon betroffenen Module aus der Topologie entfernen. Damit andere Applikationen ebenfalls auf Veränderungen an der Topologie reagieren können, soll die Beispielapplikation außerdem über einen Benachrichtigungsmechanismus verfügen. Von diesem können u. a. Applikationen für die Steuerung der Materialflussanlage profitieren und die veränderte Topologie in ihre Steuerungsentscheidungen miteinbeziehen.

Neben der Berechnung der Anlagentopologie soll die Beispielapplikation einen weiteren Schritt in Richtung der Smart Neighbourhood Module ermöglichen und Informationen zu den in die Anlage integrierten Modulen bereitstellen. Zu diesem Zweck muss ein Modul über einen Modultyp-Index, einer Zahl zwischen 0 und 65.535, verfügen. Der Modultyp-Index kann zur Ermittlung eines zum Modul gehörenden Datensatzes aus einer sich in der Cloud befindenden Datenbank genutzt werden. Diese Datenbank soll für jedes Modul seine Metadaten enthalten. Zu diesen gehören u. a. Hersteller, Herstellungsdatum, Gerätetyp sowie beliebig viele weitere Daten. Des Weiteren sollen Module in der Lage sein sich eigenständig ohne manuelle Konfiguration zu

Verbünden zusammenzuschließen und durch Aggregation ihrer Einzelfunktionalitäten modulübergreifende Funktionen bereitstellen. Dafür verfügt jedes Modul über eine Liste der Funktionen, die es selbst anbietet. Anhand dieser Funktionslisten können Verbünde von Modulen automatisiert entscheiden, ob für das Anbieten einer modulübergreifenden Funktion alle benötigten Basisfunktionen vorhanden sind. Darüber hinaus kann ein Modul über sog. Zusatzgeräte (*Addons*) verfügen. Diese realisieren zusätzliche Funktionen eines Moduls, die dynamisch aktiviert und deaktiviert werden können und als Webservice angeboten werden. Diese sollen von der Beispielapplikation ebenfalls berücksichtigt werden. Neben dem Modultyp-Index und der Funktionsliste besitzt jedes Modul eine eindeutige UUID, die zur Identifikation eines Moduls in der Anlagentopologie verwendet wird. Darüber hinaus muss ein Modul in der Lage sein, die UUIDs seiner benachbarten Module abfragen zu können.

Die Beispielapplikation muss somit eine Nachbarschaftserkennung enthalten, die den Austausch der o. g. Informationen ermöglicht. Die darauf aufbauende Berechnung der Anlagentopologie muss ebenfalls Teil der Beispielapplikation sein und diese Informationen zu einer globalen Sicht aggregieren.

7.8.2 Konzeption

Dieser Abschnitt beschreibt zwei Konzepte zur Umsetzung der im vorherigen Abschnitt beschriebenen Anforderungen. In beiden Konzepten ist die Beispielapplikation als Webservice konzipiert.

Das erste Konzept für die Beispielapplikation basiert auf der folgenden Grundüberlegung: Jeder Knoten fügt seine Nachbarschaftsinformationen in eine verteilte Datenbank ein und hält diese entsprechend eventueller Änderungen aktuell. Ein zur Beispielapplikation gehörender Topologie-Webservice, der nur auf einem Knoten innerhalb der FiLeCC-Cloud gestartet ist, fragt in regelmäßigen Abständen die Datenbank nach Veränderungen ab und bemerkt so nach kurzer Zeit Veränderungen an der Anlage. Gemäß der in der verteilten Datenbank gespeicherten Informationen kann dieser Webservice zu jeder Zeit die aktuelle Anlagentopologie berechnen. Die Nachbarschaftserkennung erfolgt in diesem Konzept ebenfalls durch einfaches Abfragen der benötigten Informationen aus der verteilten Datenbank.

Der Vorteil dieses Vorgehens ist, dass die Daten nur einmal in die Datenbank geschrieben werden müssen und der Aufwand dafür unabhängig von der Anzahl der auf sie zugreifenden Topologie-Webservices ist. Als nachteilig erweist sich jedoch das periodische Abfragen der Datenbank, um an Informationen über Änderungen zu gelangen, da unabhängig von tatsächlich stattgefundenen Änderungen periodisch die Daten ausgelesen und auf eventuelle Änderungen hin überprüft werden müssen, um die Aktualität der Anlagentopologie zu gewährleisten. Ergänzend, jedoch nicht ausschlaggebend, kommt hinzu, dass es sich bei dem Topologie-Webservice um den im Pflichtenheft (siehe Anhang A) geforderten Beispielwebservice handelt. Für diesen ist ein über das Auslesen von Daten aus einer Datenbank hinausgehender Aufbau, wie z. B. eine Interaktion mit anderen Webservices, wünschenswert. Aus diesen Gründen ist auf eine Realisierung dieses Konzepts verzichtet worden.

Das zweite Konzept ergibt sich aus den genannten Nachteilen des ersten Konzepts und ist im Folgenden beschrieben. Kernpunkt des zweiten Konzepts ist das in Abbildung 7.17 dargestellte Ebenenmodell. Die untere Ebene, die als Modul-Ebene bezeichnet wird, umfasst die

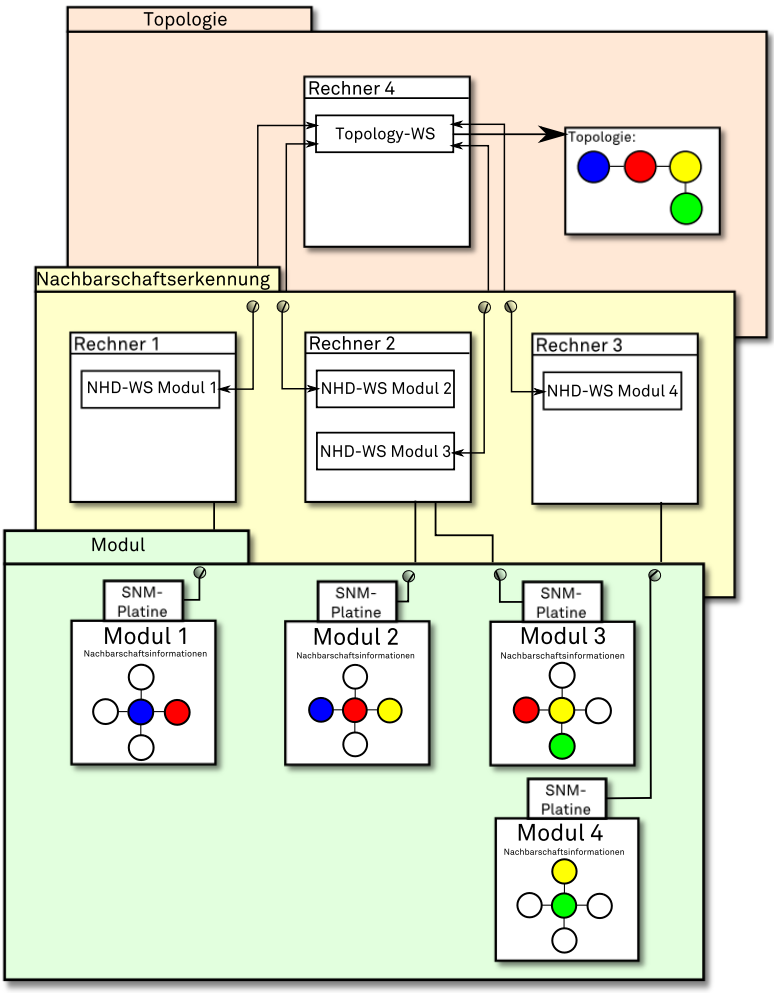


Abbildung 7.17: Ebenenmodell des zweiten Konzepts

realen Module der Anlage. Diese sind mit einer SNM-Platine ausgestattet, die jeweils mit einem Knoten der FiLeCC-Cloud verbunden sind. Dabei können mehrere Module an demselben Knoten angeschlossen sein. Auf die Nachbarschaftsinformationen der einzelnen Module kann über die SNM-Platine zugegriffen werden. Dieser Zugriff erfolgt aus der mittleren Ebene, der Nachbarschaftserkennungs-Ebene. Auf Knoten, an denen Module angeschlossen sind, ist für jedes angeschlossene Modul ein sog. Nachbarschaftserkennungswebservice gestartet. Dieser liest die Nachbarschaftsinformationen von dem ihm zugeordneten Modul aus, bereitet sie auf und stellt sie als Webservice den anderen Knoten der Cloud zur Verfügung. Die obere Ebene übernimmt die Berechnung der Anlagentopologie und wird deshalb Topologie-Ebene genannt. Auf einem beliebigen Knoten der Cloud wird ein Topologiewebservice ausgeführt. Dieser interagiert mit den Nachbarschaftserkennungswebservices der darunter liegenden Ebene und berechnet anhand der erhaltenen Informationen die Topologie der Anlage.

Dieses Konzept zergliedert die Beispielapplikation in zwei separate Applikationen. Zum einen ist

dies die Nachbarschaftserkennungsapplikation, die die Aufgaben der Nachbarschaftserkennungs-Ebene übernimmt und zum anderen die Topologieapplikation, die für die Berechnung der Anlagentopologie zuständig und der Topologie-Ebene zugeordnet ist. Die Realisierung dieses Konzepts ist Gegenstand des folgenden Abschnitts 7.8.3.

7.8.3 Realisierung

Die Beschreibung der Realisierung des im vorangegangenem Abschnitt 7.8.2 beschriebenen zweiten Konzepts orientiert sich an dem ebenfalls in Abschnitt 7.8.2 vorgestellten Ebenenmodell aus Abbildung 7.17.

Modul-Ebene

Die Module der Modul-Ebene werden mit der FiLeCC-Cloud über SNM-Platinen verbunden. Die SNM-Platinen, die für das Speichern und Bereitstellen der Nachbarschaftsinformationen zuständig sind, werden der Projektgruppe durch den Lehrstuhl für Förder- und Lagerwesen der TU Dortmund zur Verfügung gestellt. Eine Ethernetschnittstelle dient zur Verbindung der SNM-Platine mit einem Knoten der Cloud, um die Nachbarschaftsinformationen von der SNM-Platine abfragen zu können. Weiterhin soll die SNM-Platine für die Ansteuerung des zugehörigen Moduls zuständig sein. Die hierfür notwendigen Steuerungsbefehle sollen ebenfalls über die Ethernetverbindung übertragen werden. Als Protokoll für den Datenaustausch kommt das Modbus-Protokoll zum Einsatz, welches in der Industrie, insbesondere im Zusammenhang mit speicherprogrammierbaren Steuerungen, weit verbreitet ist (siehe [Mod06]). Als zugrunde liegendes Transportprotokoll wird TCP/IP verwendet. Damit die SNM-Platine in der Lage ist, Informationen zu ihren benachbarten Modulen zu ermitteln, muss diese über eine direkte Verbindung zu den benachbarten Modulen verfügen. Aufgrund der Orientierung an der Miniaturfördertechnik (siehe Rahmenbedingungen in Abschnitt 7.8.1) wird für die Smart Neighbourhood Module eine quadratische Modulgröße vorausgesetzt. Die Anordnung der Module orientiert sich dabei an einem fest vorgegebenen Raster, sodass ein Modul an jeder Seite maximal ein anderes Modul als Nachbarn besitzen kann. Demzufolge sind für die SNM-Platine vier Schnittstellen zu benachbarten Modulen erforderlich. Diese SNM-Platinen sind Eigenentwicklungen des Lehrstuhls für Förder- und Lagerwesen. Die Entwicklung hat aufgrund der Planung, dem Layouting und der Bestückung der Platine viel Zeit gekostet, sodass im Rahmen der Projektgruppe noch keine einsatzfähigen SNM-Platinen zur Verfügung gestanden haben. Aus diesem Grund ist ein Simulator implementiert worden, der in der Lage ist, eine SNM-Platine zu simulieren. Der Simulator ist als Konsolenanwendung ausführbar und repräsentiert pro Instanz ein Modul mitsamt SNM-Platine.

Für den Zugriff auf die von den SNM-Platinen bzw. den Simulatoren bereitgestellten Nachbarschaftsinformationen ist eine Systemkomponente für FiLeCC entwickelt worden. Ziel dieser Systemkomponente ist eine Abstraktion von dem verwendeten Modbus-Protokoll zu dem Zweck, die Kommunikation mit den SNM-Platinen zu vereinfachen und die SNM-Platinen aufgrund ihrer eingeschränkten Rechenleistung nicht mit zu vielen parallelen TCP-Verbindungen zu überlasten. Aus diesem Grund ist diese Komponente, die als *Modulmediator* bezeichnet wird, als Zwischenschicht zwischen den SNM-Platinen und den auf sie zugreifenden Komponenten angelegt. Der Modulmediator ist somit die einzige Komponente des Systems, die direkt mit den SNM-Platinen

kommuniziert. Dabei existiert für jede SNM-Platine genau ein Modulmediator, der die Kommunikation aller Komponenten mit dieser SNM-Platine bündelt. Unter Verwendung einer einzigen TCP-Verbindung liest er Daten auf Anfrage anderer Komponenten aus und leitet Schreibzugriffe weiter. Für die Kommunikation mit den SNM-Platinen, die über das Modbus-Protokoll erfolgt, wird die *jamod*-Bibliothek verwendet, welche eine einfach zu handhabende Modbus-Abstraktion bereitstellt.

Das Klassendiagramm zum Modulmediator ist in Abbildung 7.18 dargestellt. Die zentrale Klasse

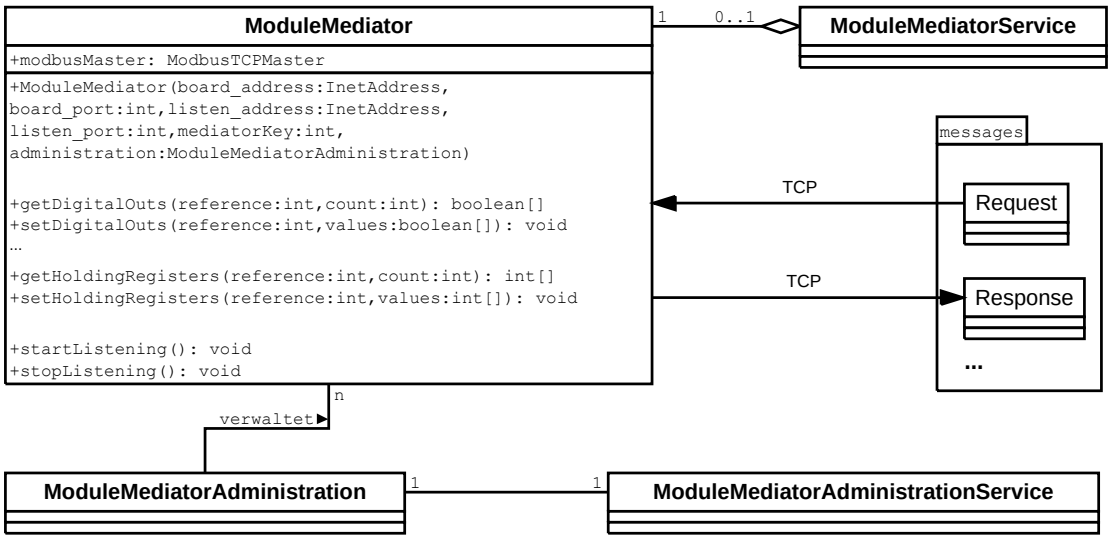


Abbildung 7.18: Klassendiagramm des Modulmediators

ist die Klasse `ModuleMediator`. Beim Instanziieren eines Objekts dieser Klasse muss diesem zum einen die IP-Adresse (Parameter `board_address`) und der Port (Parameter `board_port`) der SNM-Platine übergeben werden. Adresse und Port hängen davon ab, wie die Ethernet-Verbindung zwischen SNM-Platine und Systemknoten konfiguriert ist. Mit diesen Verbindungsdaten verbindet sich der Modulmediator mit der SNM-Platine. Zum anderen muss eine Adresse (`listen_address`) mit zugehörigem Port (`listen_port`) spezifiziert werden, an dem das `ModuleMediator`-Objekt auf eingehende TCP-Anfragen (Requests) von anderen Komponenten wartet, um diese an die SNM-Platine weiterzuleiten. Andere Komponenten haben somit die Möglichkeit, mit dem Modulmediator in Verbindung zu treten, um so mit der SNM-Platine kommunizieren zu können. Neben der Möglichkeit, Anfragen über eine TCP-Verbindung an den Modulmediator zu senden, besteht auch die Möglichkeit, die entsprechenden Kommunikationsmethoden (u. a. `getDigitalOuts()` und `setDigitalOuts()`) direkt auf dem `ModuleMediator`-Objekt aufzurufen. Eine dritte Möglichkeit, über den Modulmediator mit der SNM-Platine zu kommunizieren, besteht in der Verwendung der Klasse `ModuleMediatorService`, welche die Funktionalitäten des Modulmediators optional als Webservice zur Verfügung stellt. Zu beachten ist hierbei jedoch, dass diese Möglichkeit unter den drei genannten aufgrund der XML-Umwandlung der Nachrichten die langsamste ist. Die Verwaltung der Modulmediatoren geschieht durch die Klasse `ModuleMediatorAdministration`, von der pro Knoten jeweils genau eine Instanz existiert. Mit ihr können neue Modulmediatoren auf einfache Weise erstellt und beendet, sowie Informationen zu aktiven Modulmediatoren ermittelt werden. Die Klasse `ModuleMediatorAdministrationService` stellt die soeben genannten Funktionen als Webservice bereit.

Um Applikationsentwicklern die Kommunikation mit den SNM-Platinen zu vereinfachen, wurde eine Hilfsklasse `BoardCommunicator` entwickelt, dessen Klassendiagramm in Abbildung 7.19 dargestellt ist. Diese verbirgt das Erstellen der Netzwerkverbindung zum Modulmedia-

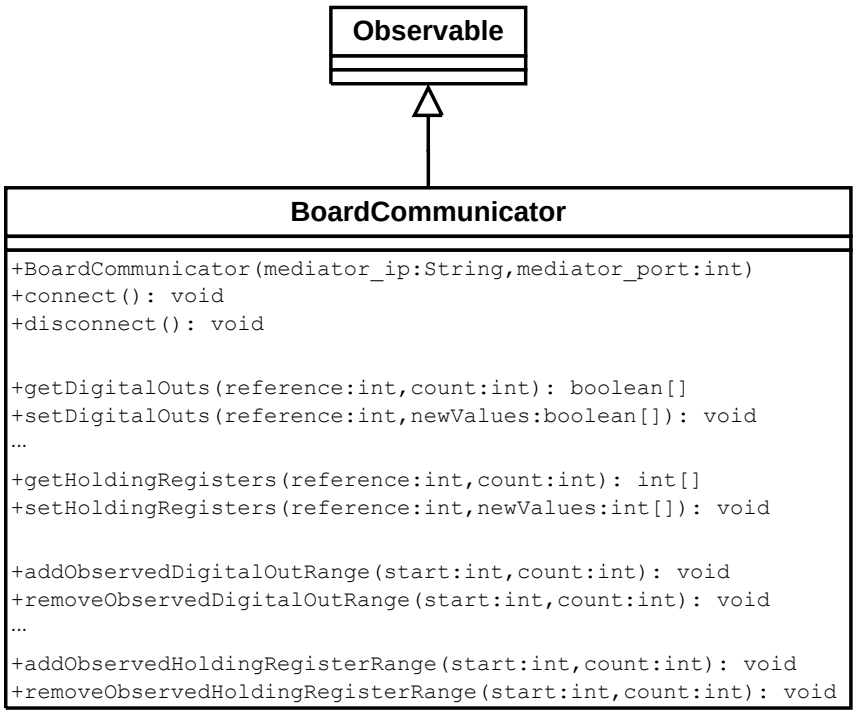


Abbildung 7.19: Klassendiagramm des BoardCommunicators

tor und den Aufbau der `Request`-Objekte vor dem Entwickler. Weiterhin bietet die Klasse `BoardCommunicator` Methoden zum Überwachen von Änderungen von digitalen Ein- und Ausgängen sowie Registern der SNM-Platinen an. Die Überwachung ist durch das Implementieren des Interfaces `Observable` im Zusammenhang mit einer periodischen Abfrage der aktuellen Werte vom Modulmediator realisiert worden. Dem Entwickler wird somit eine Abstraktion bereitgestellt, die einen einfachen Datenaustausch mit den SNM-Platinen ermöglicht. Hiervon profitiert u. a. die im folgenden Unterabschnitt vorgestellte Nachbarschaftserkennungsapplikation.

Die Modul-Ebene enthält neben den Modulen mitsamt SNM-Platinen bzw. den Simulatoren auch die Softwarekomponente, die eine Schnittstelle zur Kommunikation mit ihnen bereitstellt. Diese Schnittstelle wird von der Nachbarschaftserkennungs-Ebene verwendet um die benötigten Nachbarschaftsinformationen von den SNM-Platinen auszulesen. Das Auslesen dieser Informationen ist im folgenden Unterabschnitt beschrieben.

Nachbarschaftserkennungs-Ebene

Die Nachbarschaftserkennungs-Ebene liest die lokalen Nachbarschaftsinformationen der einzelnen Module aus, bereitet diese auf und stellt sie als Webservice der Cloud zur Verfügung. Diese Aufgaben übernimmt die als externe Applikation entwickelte Nachbarschaftserkennungsapplikation, welche als Webservice konzipiert worden ist und die Nachbarschaftsinformationen der Cloud zur Verfügung stellt. Dieser Nachbarschaftserkennungsapplikation ist, im Sinne der

in der Einleitung zu Kapitel 7 genannten Definition, ein lokaler Webservice. Auf Knoten, an denen mindestens ein Modul angeschlossen ist, wird eine Instanz dieser Applikation gestartet, die anschließend für jedes angeschlossene Modul einen Nachbarschaftserkennungswebservice anbietet. Zwar sind die Nachbarschaftsinformationen bereits mit den Webservices der Modulmediatoren abrufbar, dies erfordert jedoch Kenntnis der Registernummern der SNM-Platinen, in denen die entsprechenden Nachbarschaftsinformationen vorgehalten werden. Weiterhin müssen die erhaltenen Daten anschließend entsprechend aufbereitet werden. Zu dieser Aufbereitung zählt u. a. das Zusammensetzen der UUIDs, die auf der Mikrocontrollerplatine aufgrund ihrer Größe jeweils in acht separaten Registern gespeichert sind. Diese und vergleichbare Aufgaben kapselt der Nachbarschaftserkennungswebservice und stellt die in Abschnitt 7.8.1 beschriebenen Nachbarschaftsinformationen in einem aufbereiteten Format zur Verfügung. Die aufbereiteten Nachbarschaftsinformationen pro Modul sind in Tabelle 7.1 dargestellt. Sowohl Metadaten als

Tabelle 7.1: aufbereitete Nachbarschaftsinformationen

Nachbarschaftsinformation	Erläuterung
eigene UUID	UUID des Moduls (dient der eindeutigen Identifikation des Moduls)
UUIDs der Nachbarn	UUIDs der direkt benachbarten Module
Knoten-UUID	UUID des Knotens, an den das Modul angeschlossen ist
Metadaten	Metadaten des Moduls (u. a. Hersteller, Herstellungsdatum, Gerätetyp)
Funktionsliste	Liste der vom Modul angebotenen Funktionen
Geltungsbereich	Geltungsbereich des Moduls
Zusatzgeräte (Addons)	Zum Modul gehörende Zusatzgeräte innerhalb des Geltungsbereichs des Moduls

auch Funktionsliste werden von der Nachbarschaftserkennungsapplikation aus einer verteilten Datenbank innerhalb der Cloud gelesen. Zur Identifikation des passenden Datensatzes wird der Modultyp-Index verwendet, der von den SNM-Platinen abgefragt werden kann. Die Datenbank muss einmalig vom Administrator angelegt werden. Als Hilfsmittel ist eine zusätzliche Applikation implementiert worden, die genau diese Aufgabe erfüllt. Der Geltungsbereich des Moduls muss beim Erstellen des Nachbarschaftserkennungswebservices angegeben werden. Dieser wird dafür benötigt, um nach Zusatzgeräten des Moduls innerhalb dessen Geltungsbereichs suchen zu können und die Daten der gefundenen Zusatzgeräte, worunter u. a. Metadaten und angebotene Funktionen fallen, in die Nachbarschaftsinformationen zu integrieren.

Das Klassendiagramm der Nachbarschaftserkennungsapplikation ist in Abbildung 7.20 dargestellt. Beim Erstellen eines `NeighbourhoodDetectionService`-Objekts müssen diesem die Adresse und der Port des Modulmediators mitgeteilt werden, über den die Kommunikation mit der SNM-Platine abgewickelt wird. Über diesen Modulmediator bezieht der Nachbarschaftserkennungswebservice die Nachbarschaftsinformationen des korrespondierenden Moduls. Die Nachbarschaftsinformationen können mit Hilfe der Methode `getModule()` abgefragt werden, welche auch als Webservice-Operation angeboten wird, sodass ein Zugriff von allen Knoten der Cloud möglich ist. Als Rückgabewert liefert die `getModule()`-Methode ein Objekt der Klasse `Module` bzw. dessen XML-Repräsentation im Fall eines Webserviceaufrufs. Diese Klasse enthält alle in Tabelle 7.1 aufgelisteten Nachbarschaftsinformationen und stellt `get`-Methoden bereit, um auf diese zuzugreifen.

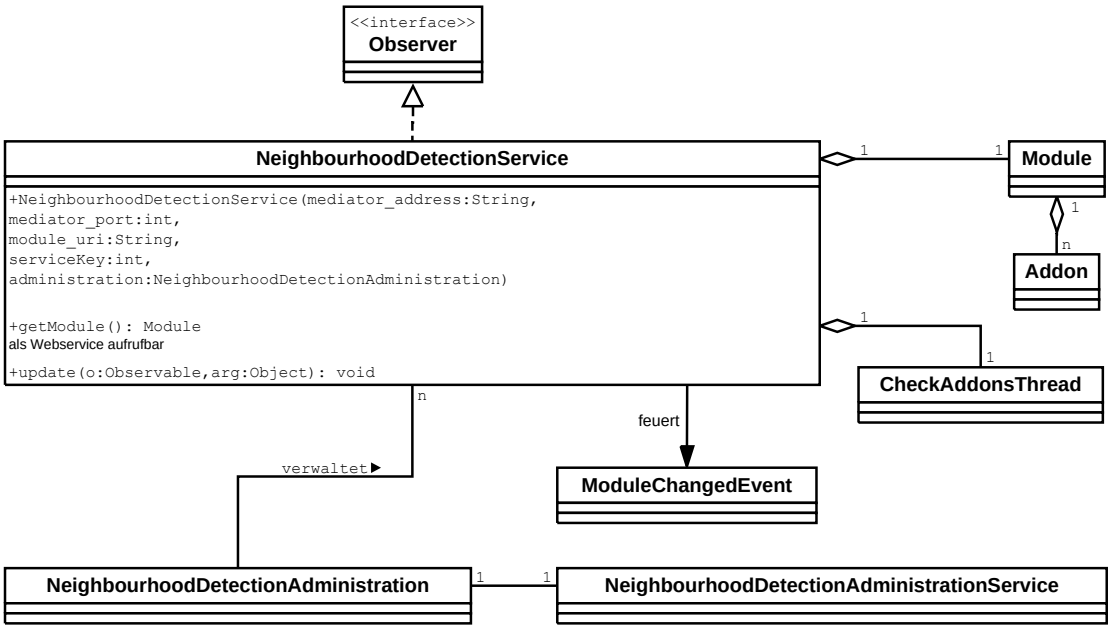


Abbildung 7.20: Klassendiagramm der Nachbarschaftserkennungsapplikation

Zum Auffinden der zu einem Modul gehörenden Zusatzgeräte anhand seines Geltungsbereichs dient die Klasse `CheckAddonsThread`, welche periodisch eine Suche nach Addon-Webservices innerhalb des Geltungsbereichts des Moduls startet. Die gefundenen Addons werden durch die Klasse `Addon` repräsentiert, die Informationen wie z. B. Metadaten und angebotene Funktionen eines Addons bereitstellt.

Durch das Zurückgreifen auf die Klasse `BoardCommunicator` und deren Überwachungsfunktionalität, die im Unterabschnitt zur Modul-Ebene beschrieben ist, wird der Nachbarschaftserkennungswebservice über Änderungen an den Nachbarschaftsinformationen des Moduls informiert. Im Anschluss an die Verarbeitung dieser Veränderungen verschickt der Nachbarschaftserkennungswebservice ein Event, das durch die Klasse `ModuleChangedEvent` repräsentiert wird. Für dieses Event können sich andere Komponenten, wie z. B. die im folgenden Unterabschnitt beschriebene Topologieapplikation, registrieren, um über Änderungen informiert zu werden.

Für das Erstellen, Konfigurieren und Beenden von Nachbarschaftserkennungswebservices bietet die Nachbarschaftserkennungsapplikation die Klassen `NeighbourhoodDetectionAdministration` und `NeighbourhoodDetectionAdministrationService` an. Diese arbeiten nach demselben Prinzip wie die entsprechenden Administrationsklassen des Modulmediators, die im vorangegangenem Unterabschnitt zur Modul-Ebene beschrieben sind.

Die Nachbarschaftserkennungsapplikation erfüllt somit alle Aufgaben der Nachbarschaftserkennungs-Ebene. Für jedes Modul wird von ihr ein Webservice angeboten, der einen Zugriff auf die aufbereiteten Nachbarschaftsinformationen des Moduls erlaubt. Für die Berechnung der Anlagentopologie müssen diese Informationen nun noch miteinander kombiniert werden. Dies ist Aufgabe der Topologie-Ebene, deren Realisierung Inhalt des folgenden Unterabschnitts ist.

Topologie-Ebene

Die Topologie-Ebene umfasst die Erstellung der Anlagentopologie. Dies realisiert die Topologieapplikation, die für die Berechnung der Topologie zuständig ist und diese anschließend als Webservice innerhalb der Cloud verfügbar macht. Die Topologieapplikation ist ein beweglicher Webservice und kann auf jedem beliebigen Knoten der Cloud ausgeführt werden. Aus den einzelnen Nachbarschaftsinformationen, die von der im vorherigen Unterabschnitt beschriebenen Nachbarschaftserkennungsapplikation bereitgestellt werden, wird ein Topologiegraph aufgebaut. Dieser Topologiegraph wird als Webservice angeboten, sodass andere Komponenten Zugriff auf diesen erhalten können. Das Schema der durch den Webservice angebotenen XML-Repräsentation des Topologiegraphen ist Listing 7.2 zu entnehmen.

Listing 7.2: XML-Schema des Topologiegraphen

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://flw.mb.udo.edu/filecc/neighbourhood/"
    xmlns="http://flw.mb.udo.edu/filecc/neighbourhood/">
 <xs:element name="topologyGraph">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="vertex" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="module">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="uuid" type="xs:string" />
          <xs:element name="uri" type="xs:string" />
          <xs:element name="metadata">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
               <xs:simpleContent>
                <xs:extension base="xs:string">
                 <xs:attribute name="value" type="xs:string" />
                </xs:extension>
               </xs:simpleContent>
              </xs:complexType>
             </xs:element>
            </xs:sequence>
           </xs:complexType>
          </xs:element>
          <xs:element name="functions">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="function" type="xs:string" maxOccurs="unbounded" />
            </xs:sequence>
           </xs:complexType>
          </xs:element>
          <xs:element name="addons">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="uri" type="xs:string" />
             <xs:element name="type" type="xs:string" />
             <xs:element name="function" type="xs:string" />
             <xs:element name="metadata" type="xs:string">
             <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
               <xs:simpleContent>
                <xs:extension base="xs:string">
                 <xs:attribute name="value" type="xs:string" />
```

```
                </xs:extension>
               </xs:simpleContent>
              </xs:complexType>
             </xs:element>
            </xs:sequence>
           </xs:complexType>
          </xs:element>
          <xs:element name="neighbours">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="uuid" type="xs:string" minOccurs="4" maxOccurs="4" />
            </xs:sequence>
           </xs:complexType>
          </xs:element>
          <xs:element name="nodeuuid" type="xs:string" />
         </xs:sequence>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="edge" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="start" type="xs:string"/>
       <xs:element name="end" type="xs:string"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Das im Folgenden beschriebene Klassendiagramm ist in Abbildung 7.21 dargestellt. Die Klasse `TopologyGraph` repräsentiert den soeben beschriebenen Topologiegraphen, wobei die Informationen zu den einzelnen Knoten des Topologiegraphen in den Datenklassen `TopologyVertex`, `Module` und `Addon` gekapselt sind. Die Klasse `TopologyClient` dient der Kommunikation mit den von der Nachbarschaftserkennungsapplikation angebotenen Webservices, die die benötigten Nachbarschaftsinformationen anbieten. Dabei obligen dem Topologie-Client einige wichtige Aufgaben. Er ist dafür zuständig, die Nachbarschaftserkennungswebservices zu überwachen. Kommen aufgrund neu angeschlossener Module neue Webservices hinzu oder melden sich bestehende Webservices aufgrund der Trennung von Modulen ab, so muss der Topologiegraph entsprechend angepasst werden. Durch das Überwachen von `Hello`- und `Bye`-Nachrichten werden derartige Veränderungen vom Topologie-Client bemerkt, sodass die Topologie entsprechend modifiziert werden kann. Fällt ein Knoten aus und damit auch die an ihm angeschlossenen Module, entfällt der Versand einer `Bye`-Nachricht. Für diesen Fall registriert sich der Topologie-Client beim globalen Watchdog für das `WatchdogTimeoutEvent` und entfernt beim Auftreten eines solchen Events alle Module aus dem Topologiegraphen, die an dem ausgefallenen Knoten angeschlossen sind. Ändern sich die Nachbarschaftsinformationen eines Moduls, wird der Topologie-Client darüber über das `ModuleChangedEvent` der Nachbarschaftserkennungsapplikation informiert. Das Anpassen des Topologiegraphen delegiert der Topologie-Client jeweils an die Klasse `TopologyService`, die den Topologiegraphen verwaltet und den Dienst zum Abfragen der Topologie anbietet. Nach erfolgten Änderungen an der Topologie versendet der Topologie-Service ein `TopologyChangedEvent`, das daran interessierte Komponenten über die veränderte Topologie informiert.

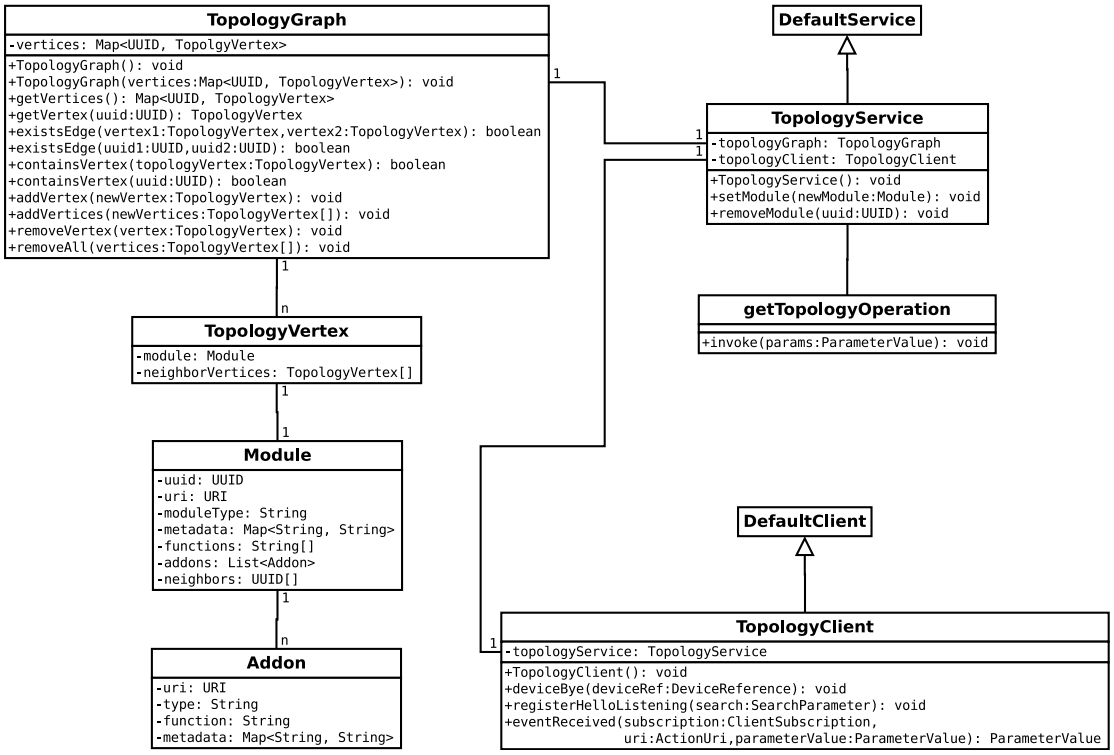


Abbildung 7.21: Klassendiagramm des TopologyService

Zusammenfassung

Die Realisierung der Nachbarschaftserkennung und Topologieberechnung ist auf drei Ebenen aufgeteilt. Die Modul-Ebene umfasst die Module selbst sowie in FiLeCC integrierte Komponenten, die eine Kommunikation mit den Modulen ermöglichen. Darauf aufbauend nutzt in der Nachbarschaftserkennungs-Ebene die externe Nachbarschaftserkennungsapplikation diese Komponenten für das Auslesen der Nachbarschaftsinformationen und stellt diese aufbereitet der Cloud in Form von Webservices zur Verfügung. In der Topologie-Ebene werden diese einzelnen Nachbarschaftsinformationen schließlich durch die externe Topologieapplikation zu einer globalen Sicht aggregiert und diese in Form eines Topologiegraphen als Dienst anderen Komponenten zugänglich gemacht. Abbildung 7.22 enthält ein Klassendiagramm, das die wichtigsten Klassen der in diesem Kapitel vorgestellten Komponenten und Applikationen auflistet und deren Zusammenspiel demonstriert.

7.8.4 Ausblick

Die Nachbarschaftserkennungsapplikation erhält die Nachbarschaftsinformationen über die Module mittels eines Mediators, der auf Basis des Modbus-Protokolls mit Simulatoren kommuniziert. Die Simulatoren stellen die SNM-Platinen dar, da sich diese zum Zeitpunkt der Berichtserstellung noch in der Entwicklung befunden haben. Der nächste Schritt ist, die SNM-Platinen einzurichten und an reale Module anzuschließen. Sie sollen den Simulator ersetzen. Des Weiteren soll die im Zuge der Projektgruppe entwickelte Software FiLeCC auf den Smart Neighbourhood Modulen

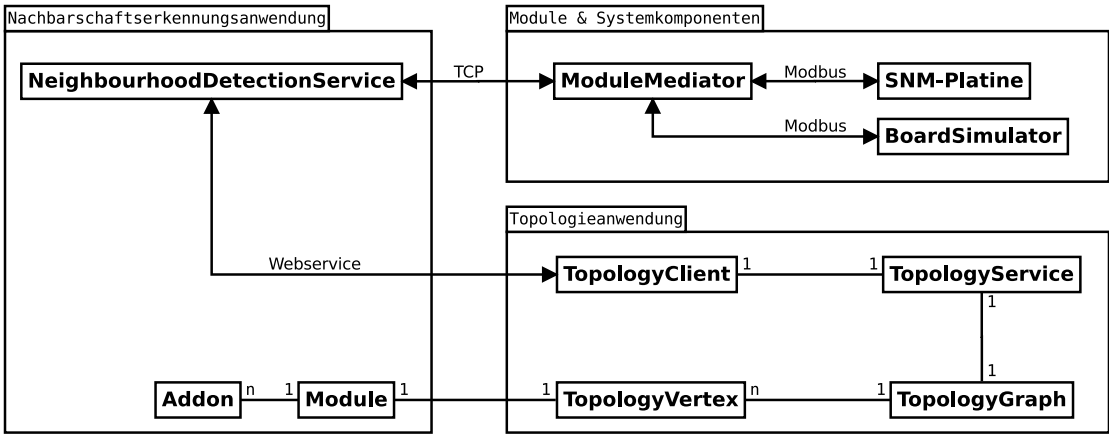


Abbildung 7.22: Klassendiagramm: Zusammenhang der einzelnen Komponenten und Applikationen

eingesetzt werden. Die simulierte Umgebung, wie sie derzeit verwendet wird, soll so durch eine reale Umgebung ersetzt werden.

Eine mögliche Optimierung im Hinblick auf die Ressourceneffizienz der Topologieberechnung ist, die fertig berechnete Topologie in der verteilten Datenbank zu speichern. Dies vermeidet eine komplette Neuberechnung der Topologie, wenn die Topologieapplikation als beweglicher Webservice z. B. auslastungsbedingt auf einen anderen Knoten verdrängt wird oder eine weitere Instanz der Topologieapplikation hinzukommt. Die Topologie kann in der vorhandenen verteilten Datenbank abgelegt und bei Änderungen an der Anlage modifiziert werden.

7.9 Datenhaltung

Die Datenhaltung von FiLeCC basiert auf den Datenbanksystemen GridGain und Hazelcast. Es sind Schnittstellen zu den FiLeCC-internen Last- und SDRM-Datenbanken sowie zur Nutzerdatenbank für Applikationen vorhanden. Die Bereitstellung der Implementierungen der Schnittstellen erfolgt durch zwei nach dem Entwurfsmuster *Factory* [Gam95] entworfene Klassen. Eine Übersicht über die Implementierung der Datenhaltungsebene von FiLeCC bieten die folgenden Abschnitte.

7.9.1 Speicherschnittstellen

Für die Bereitstellung der Schnittstellen zu den verschiedenen Datenbanken von FiLeCC sind zwei Klassen notwendig, um Applikationen, die auf der Plattform betrieben werden, den Zugriff auf die internen Datenbanken zu verwehren. Beide Klassen sind nach dem Entwurfsmuster *Factory* [Gam95] entworfen, um ggf. die Implementierung austauschen zu können, ohne den Quelltext zu ändern, der die Datenbanken nutzt. Eine Darstellung der *Factory* für die internen Datenbanken mit der Bezeichnung `DatabaseFactory` findet sich in Abbildung 7.23. Diese Klasse verwendet zusätzlich das Entwurfsmuster *Singleton* [Gam95] um das mehrfache Instanziieren der Datenbankschnittstellen zu verhindern. Eine detailiertere Beschreibung der zwei ebenfalls in Abbildung 7.23 enthaltenen Schnittstellen für die Last- und SDRM-Datenbank

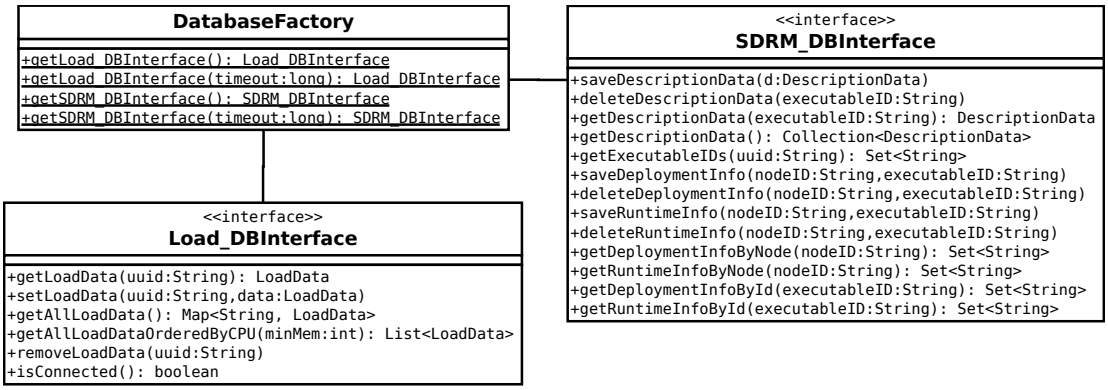


Abbildung 7.23: Klassendiagramm der Klasse `DatabaseFactory` und der dazugehörigen Schnittstellen

folgt in den Abschnitten 7.9.2 und 7.9.3. Die Klasse `UserDatabaseFactory`, für die Bereitstellung der Schnittstelle zur Datenbank für Applikationen, ist in Abbildung 7.24 dargestellt. Die Beschreibung der Schnittstelle folgt in Abschnitt 7.9.4.

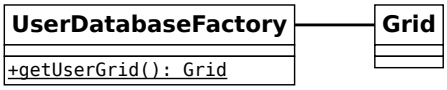


Abbildung 7.24: Klassendiagramm der Klasse `UserDatabaseFactory`

7.9.2 Lastdaten Schnittstelle

In Abschnitt 7.7 werden die zu speichernden Lastdaten definiert. Tabelle 7.2 stellt diese Daten tabellarisch dar. Die Schnittstelle für die Lastdatenbank ist im Java-Interface `Load_DBInterface`

Tabelle 7.2: Tabelle mit Beispiellastdaten zu drei Knoten

Node	CPU Clock	CPU Idle	Free RAM	Free Diskspace
UUID1	1.000 MHz	68%	223 MB	2.333 MB
UUID3	666 MHz	13%	87 MB	-
UUID2	1.200 MHz	42%	121 MB	451 MB

definiert. In Abbildung 7.23 sind die Methoden des Interfaces dargestellt. Konkret implementiert sind die Methoden für die Speicherung der Daten in einer Hazelcast-Datenbank, in der Klasse `Load_DBHazelcast`. Die Factory instanziiert diese konkrete Implementierung.

In der Schnittstelle sind Methoden vorhanden, welche den aktuellen Lastzustand eines Knoten setzen, auslesen oder löschen. Die Methode `getLoadDataOrderedByCPU(minMem:int)` liefert eine sortierte Liste aller Knoten zurück, welche eine bestimmte minimale Größe des freien Arbeitspeichers besitzen. Diese Liste kann z. B. für Lastverteilungsentscheidungen genutzt werden.

7.9.3 SDRM-Daten Schnittstelle

In Abschnitt 7.2 werden die für das SDRM zu speichernden Daten definiert. Die Tabellen 7.3, 7.4 und 7.5 stellen diese Daten tabellarisch dar. Die Schnittstelle, welche die SDRM-Daten speichert,

Tabelle 7.3: Tabelle des SDRM mit Konfigurationsdaten der im System installierten Software

Identifier	wsTopology	someBinary	wsSomethingElse	...
Version	1.0	3.87-r2	2.0	...
File name		start.exe		...
Cmd arguments		-some-parameter=4		...
Main class	org.test.TopologyMain			...
Multicast address	240.0.1.4/24			...
Min free RAM		16M		...
Filesize	45678421	4568797	12055640	...
Keep local	true	false	false	...

Tabelle 7.4: Tabelle des SDRM mit den Daten der Verteilung von Binärdateien auf Knoten des Systems

Identifier	**Node**
wsTopology	UUID1
wsTopology	UUID2
wsTopology	UUID3
someBinary	UUID1
wsSomethingElse	UUID3

Tabelle 7.5: Tabelle des SDRM mit den Daten der Verteilung von Softwareinstanzen auf Knoten des Systems

Identifier	**Node**
wsTopology	UUID2
someBinary	UUID1
someBinary	UUID3

wird durch das Java-Interface `SDRM_DBInterface` beschrieben. In Abbildung 7.23 sind die angebotenen Methoden dargestellt. Zur Implementierung des Interfaces werden die Methoden auf eine GridGain-Datenbank abgebildet. Dabei wird GridGain auf allen Knoten ausgeführt und die Daten werden vollständig repliziert, was eine hohe Ausfallsicherheit ermöglicht.

Die Aufgabe der SDRM Schnittstelle ist es, drei Datenmengen zu verwalten. Eine Beschreibung der Datenmengen findet sich in Abschnitt 5.4. Für jede Datenmenge wird ein `GridCache` eingesetzt.

Die SDRM Schnittstelle bietet Methoden an, welche die Verwaltung der Datenmengen erlauben. Dabei besteht die Möglichkeit neue Datensätze einzufügen oder bestehende Datensätze zu löschen. Über diese Methoden ist weiterhin ableitbar, auf welchen Knoten eine Binärdatei bereitgestellt und wo sie aktuell ausgeführt wird. Darüber hinaus sind Informationen über die verfügbaren Binärdateien für jeden Knoten abrufbar. Diese komplexe Schnittstelle mit speziellen

Methoden für alle Anwendungsfälle verlagert einen großen Teil der Datenverarbeitung von den die Daten nutzenden Komponenten in die Schnittstelle. Das senkt den Entwicklungsaufwand für Software die die Datenbank nutzt. Darüberhinaus können effizientere Anfragemechanismen wie Map-Reduce (siehe Abschnitt 4.6) eingesetzt und für jeden Anwendungsfall optimiert werden. Nachteilig wirkt sich eine solche Architektur aus, wenn Änderungen an den Daten vorgenommen werden. In diesem Fall müssen viele Methoden der Schnittstelle und die Implementierungen für jedes Datenbank-Backend angepasst werden.

7.9.4 Nutzerdaten Schnittstelle

Bei FiLeCC handelt es sich um eine Ausführungsplattform für Applikationen. Weil die Anforderungen der externen Entwickler im Vorfeld nicht feststehen und sich stark voneinander unterscheiden können, wird eine verteilte Datenbank zur Verfügung gestellt, welche frei konfigurierbar ist. Aufgrund von Flexibilitätsaspekten, auf die im Folgenden eingegangen wird, wird dazu eine separate Datenbank mit eigenem Grid verwendet. Weiterhin können dadurch die externen Entwickler das FiLeCC-Basissystem nicht beeinflussen.

Dieses zweite, nebenläufige Grid wird ebenfalls auf jedem Knoten betrieben. Das FiLeCC-Grid und das Nutzer-Grid sind unabhängig voneinander und haben verschiedene Konfigurationen. Weil bei GridGain keine Konfigurationsänderungen zur Laufzeit möglich sind, wird den Entwicklern eine Schnittstelle zur Verfügung gestellt, mittels welcher sie das Grid neu starten können. Durch den Austausch der Konfigurationdatei des Nutzer-Grid ist es somit möglich die Nutzerdatenbank beliebig zu rekonfigurieren, ohne das Basissystem zu stören.

7.9.5 Starten und Skalieren der Datenbanksysteme

Aufgrund spezieller Anforderungen (siehe Abschnitt 5.4) und einiger Einschränkungen der von FiLeCC verwendeten Datenbanksysteme (siehe Abschnitt 6.6.3) erfordert das Starten und Skalieren der Datenbanksysteme zusätzlichen Aufwand. Eine Beschreibung findet sich, getrennt nach Datenbanksystem, in den folgenden Abschnitten.

GridGain Da von GridGain sowohl das System für verteiltes Rechnen (siehe 6.5) als auch die verteilte Datenbank auf jedem FiLeCC-Knoten benötigt wird, wird GridGain automatisch vom `AppServer` (vgl. Abschnitt 7.1) gestartet. Wie bereits in 7.9.4 beschrieben, werden auf jedem Knoten zwei Grid-Instanzen gestartet. Eine Instanz für das FiLeCC-Grid und eine weitere für das Nutzer-Grid. Die Konfiguration erfolgt über zwei XML-Dateien, welche in FiLeCC enthalten sind.

Bei einem GridGain-Einsatz ohne Datenbank-Funktionalität, funktioniert die Clusterbildung automatisch per Multicast. Weil für FiLeCC auch die Datagrid-Funktionalität notwendig ist, muss auf eine Clusterbildung per TCP/IP zurückgegriffen werden. Die Schwierigkeit besteht darin, die IP-Adresse eines bereits in das Grid eingebundenen Knotens zu finden, oder festzustellen, dass noch kein Grid existiert. Dazu wird ein Webservice angeboten, welcher die IP-Addresse eines Knotens des Grids zurückgibt. Um beim Systemstart zu verhindern, dass mehrere Grids parallel aufgebaut werden, existiert genau ein solcher Webservice im System. Der den Webservice anbietende Knoten wird durch das in Abschnitt 7.4 beschriebene Leaderwahlverfahren bestimmt.

Wird ein Knoten zum Leader gewählt, so startet er ein neues Grid. Alle übrigen Knoten verbinden sich über die vom Webservice abfragbare IP-Adresse mit diesem Grid. Durch dieses Verfahren wird es sichergestellt, dass nur ein Grid im ganzen System vorhanden ist und alle dazukommende Knoten sich daran anschließen.

Hazelcast Das Datenbanksystem Hazelcast wird bei FiLeCC nicht auf jedem Knoten betrieben. Eine höhere Ausfallsicherheit durch eine permanente Verteilung aller Daten auf alle Knoten ist nicht notwendig, weil Hazelcast lediglich als Cache für aktuelle Lastdaten fungiert. Diese Daten werden bei Verlust automatisch vom Ressourcenmonitor neu geschrieben, da sie ohnehin kontinuierlich aktualisiert werden. Das Fehlen von Lastdaten oder der gesamten Datenbank ist dennoch unerwünscht, da es zu Verzögerungen oder Fehlentscheidungen von Diensten führt, die auf die Lastdaten zurückgreifen. Da das Vorhalten der Daten auf vielen Knoten durch den Synchronisationsaufwand allerdings zu mehr Belastung der Knoten und des Netzwerkes führt, ist eine geeignete Balance von Ausfallsicherheit und Ressourcenverbrauch notwendig. Hieran orientiert sich die Mindestanzahl der Hazelcast-Instanzen, sowie die Anzahl der Backups eines jeden Datums. Diese Werte müssen vom Administrator passend gewählt werden. Für kleinere Systeme mit bis zu acht Knoten hat sich in den durch die Projektgruppe durchgeführten Tests sowohl für die Mindestanzahl der Hazelcast-Instanzen als auch für die Anzahl der Backups jedes Datums der Wert drei bewährt. Für Clouds die aus einer größeren Anzahl Knoten gebildet sind, ist ggf. eine Anpassung der Werte notwendig. Hierbei sollen die Werte allerdings deutlich langsamer als linear in Abhängigkeit zur Anzahl der Knoten wachsen. Eine Formel zur Berechnung eines geeigneten Wertes ist nicht verfügbar.

Darüberhinaus muss sichergestellt werden, dass die den Hazelcast-Instanzen zur Verfügung stehenden Ressourcen ausreichen, um alle Daten speichern und alle Abfragen zeitnah beantworten zu können. Falls dies nicht der Fall ist, müssen weitere Instanzen gestartet werden, da Hazelcast nicht auf allen Knoten betrieben wird und wie im Folgenden beschrieben die Speicherressourcen pro Instanz beschränkt sind. Die Verfügbarkeit von Rechenzeit wird dabei durch die in Kapitel 7.3 beschriebene UNO sichergestellt. Das Vorhandensein ausreichender Speicherressourcen stellen die Hazelcast-Instanzen selbstständig sicher. Dazu wird Hazelcast für FiLeCC innerhalb der Wrapper-Klasse `HazelcastNode` betrieben. Ein Klassendiagramm findet sich in Abbildung 7.25.

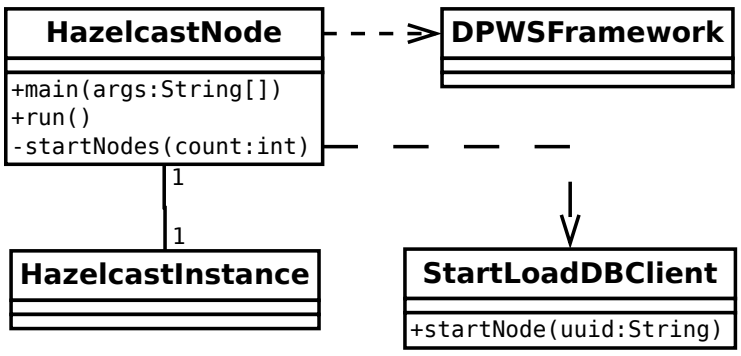


Abbildung 7.25: Klassendiagramm HazelcastNode

Die Methode `run()` übernimmt dabei das Monitoring der lokalen Instanz sowie das des gesamten Clusters. Alle zehn Sekunden werden lokal die RAM-Speicherauslastung sowie cluster-weit

die Anzahl der Hazelcast-Instanzen von FiLeCC überprüft. Liegt die Speicherauslastung der Java-VM, in welcher die `HazelcastNode` ausgeführt wird, bei über 80%, so wird per Webservice-Aufruf an das SDRM eine neue Hazelcast-Instanz auf dem am wenigsten ausgelasteten Knoten gestartet. Dazu wird die Methode `startNodes(int count)` aufgerufen, welche wiederum die Methode `startNode(String uuid)` der Klasse `StartLoadDBClient` benutzt, um den Webserviceaufruf auszuführen. Ist die Speicherauslastung unter 20% gesunken und die Anzahl der Hazelcast-Instanzen größer als die vorher festgelegte Mindestanzahl, so beendet sich die lokale Instanz, um Ressourcen freizugeben. Liegt die Anzahl der Hazelcast-Instanzen unabhängig von der Speicherauslastung unter der Mindestanzahl, so werden neue Instanzen gestartet. Dieser Vorgang wird höchstens alle zwei Minuten durchgeführt, um es den neuen Instanzen zu ermöglichen, sich mit dem Cluster zu verbinden. Ohne dieses Warteintervall werden ggf. über einen kurzen Zeitraum alle zehn Sekunden neue Instanzen gestartet, weil die zuerst gestarteten sich noch nicht mit dem Cluster verbunden haben und die feststellbare Anzahl der Hazelcast-Instanzen nach wie vor unter dem Minimum liegt. Erst wenn sich so viele neue Instanzen vollständig mit dem Cluster verbunden haben, dass die Mindestanzahl erreicht ist, werden keine weiteren Instanzen gestartet. Die überhöhte Steigerung der Anzahl der Instanzen wird durch das Warteintervall von zwei Minuten verhindert. Die Werte für das periodische Überprüfungsintervall, die Wartezeit zwischen zwei Instanzstarts sowie die Ober- und Untergrenze der Speicherauslastung resultieren aus Systemtests der Projektgruppe mit Systemen aus weniger als zehn Knoten. Größere Systeme erfordern eine Anpassung dieser Werte, wobei keine Formel zur Berechnung geeigneter Werte verfügbar ist.

Um die Speicherauslastung bestimmen zu können, wird jede Hazelcast-Instanz, im Gegensatz zu allen anderen Komponenten von FiLeCC, in einer eigenen Java-VM betrieben. Über die Aufrufe `Runtime.getRuntime().totalMemory()` und `Runtime.getRuntime().maxMemory()` kann dadurch die aktuelle Speicherauslastung bestimmt werden. Dieser Mechanismus zur Steuerung der Anzahl der Hazelcast-Instanzen ist notwendig, da FiLeCC für heterogene Umgebungen konzipiert ist, Hazelcast die Daten allerdings gleichmäßig auf alle Instanzen verteilt. Durch eine Beschränkung der Speichermenge, die der Java-VM maximal zur Verfügung steht, wird es möglich, Hazelcast auch auf ressourcenschwachen Systemen zu betreiben. Dabei richtet sich die Speichergrenze nach dem am schlechtesten mit Arbeitsspeicher ausgestatteten Knoten des Systems, auf dem eine Hazelcast-Instanz ausgeführt werden soll. Dies muss durch den Administrator sichergestellt werden. Der bereits beschriebene Mechanismus zum Starten weiterer Hazelcast-Instanzen bei unzureichend werdenden Speicherressourcen führt dazu, dass in heterogenen Umgebungen auch die Knoten mit viel Arbeitsspeicher ausgelastet werden, da die neuen Instanzen bevorzugt auf diesen Knoten gestartet werden. Durch die Wrapper-Klasse `HazelcastNode` wird es also möglich eine Hazelcast-Datenbank dynamisch zu skalieren und dabei die Ressourcen von heterogenen Umgebungen auszunutzen.

KAPITEL 8

Erprobung

Das folgende Kapitel gibt einen Überblick über das Testverfahren. Nach den Tests der Einzelkomponenten ergibt sich die Notwendigkeit, das Gesamtsystem auf Funktion zu prüfen. Für diesen Zweck gibt es eine Testumgebung mit einem choreografierten Testlauf, der alle Anwendungs- und Testfälle des Pflichtenheftes umfasst. Diese Testumgebung und die Choreografie werden in Abschnitt 8.1 beschrieben. Die Auswertung des Tests geschieht in Abschnitt 8.2, in dem auch die Probleme, die beim Testen aufgetreten sind, beschrieben werden.

8.1 Testumgebung

Die Testumgebung besteht aus einem Netzwerk, an das vier verschiedene Geräte angeschlossen sind. Diese Geräte unterscheiden sich hinsichtlich Gerätetyp und eingesetztem Betriebssystem. Als Gerätetypen werden in der Testumgebung sowohl Notebooks als auch ein Netbook verwendet. Die eingesetzten Betriebssysteme sind Microsoft Windows auf dem Netbook und Linux sowohl in der 32- als auch 64-Bit Version auf den Notebooks. Bei der Konfiguration der Geräte ist zu beachten, dass IPv6 deaktiviert und eine Multicastroute vorhanden ist, um eine korrekte Funktionsweise der Webserviceaufrufe und der Übermittlung von Multicast-Nachrichten zu gewährleisten. Für den auf dem Pandaboard, das im Gegensatz zu den tatsächlich eingesetzten Geräten nicht auf der x86- sondern auf der ARM-Architektur basiert, vorhandenen Sound-Chipsatz hat zum Zeitpunkt der Testdurchführung kein funktionsfähiger Soundtreiber für die ARM-Architektur unter Linux existiert. Da die Testumgebung jedoch zur Vorführung von FiLeCC eingesetzt werden soll und der im Folgenden beschriebene Testablauf eine funktionsfähige Soundausgabe voraussetzt, wird auf den ursprünglich geplanten Einsatz des Pandaboards verzichtet.

Weiterhin wird ein Gerät zur Steuerung des Testablaufs mit Hilfe der Administrationsoberfläche benötigt, das dieselben Netzwerkvorraussetzungen hat wie die eigentlichen Testgeräte.

Die vier zur Testumgebung gehörenden Geräte sowie das Administrationsgerät sind innerhalb eines Netzwerks miteinander verbunden. Die zugehörige Netztopologie ist in Abbildung 8.1 dargestellt. Sie besteht aus zwei Switches, die jeweils zwei der vier Geräte verbinden. Das

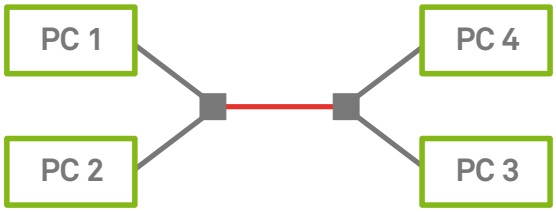


Abbildung 8.1: Netzwerktopologie

Administrationsgerät ist mit einem der beiden Switches verbunden, wird aber im Folgenden nicht mehr explizit erwähnt. Die Switches selbst sind untereinander ebenfalls verbunden, sodass durch das Trennen dieser Verbindung eine Teilung des Netzwerks simuliert werden kann, die das Netzwerk in zwei Teilnetze zu je zwei Geräten aufteilt.

Für die in Kapitel 7.8 beschriebene Nachbarschaftserkennung, die durch den in diesem Abschnitt vorgestellten Testablauf ebenfalls getestet wird, müssen Vorbereitungen getroffen werden. Um eine Anlagentopologie zu simulieren, die aus mehr als vier Fördermodulen besteht, sind jedem Gerät der Testumgebung mehrere solcher Module zugeordnet. Die simulierte Anlagentopologie besteht insgesamt aus 9 Fördermodulen. Die Zuordnung der simulierten Fördermodule zu den am Testablauf beteiligten Geräten ist Abbildung 8.2 zu entnehmen. Für die Simulation der

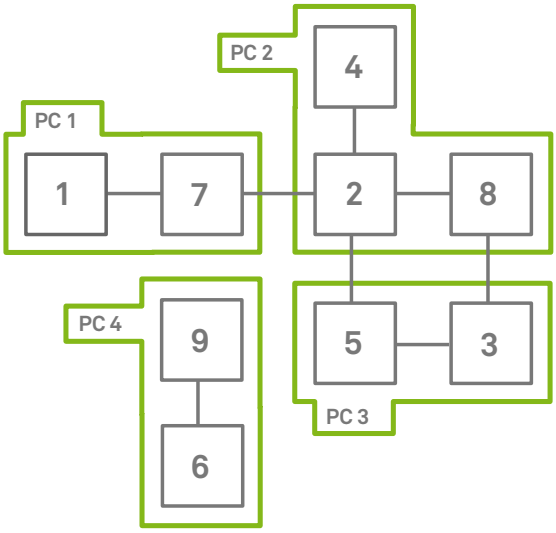


Abbildung 8.2: Anlagentopologie

Fördermodule muss auf den Geräten pro simuliertem Fördermodul ein Simulator gestartet werden. Entsprechend der Anlagentopologie müssen die Simulatoren weiterhin entsprechend konfiguriert werden, sodass sich die in Abbildung 8.2 dargestellten Nachbarschaftsbeziehungen ergeben. Im Anschluss an das Starten der Simulatoren auf den einzelnen Geräten kann die Software FiLeCC gestartet werden. FiLeCC liegt dabei als Kompilat in einer einzelnen JAR-Datei vor, sodass diese lediglich auf alle am Test beteiligten Geräte verteilt und anschließend gestartet werden muss. Weiterhin muss die Testanwendung, die bei Ausführung eine Sounddatei

abspielt, auf dem Administrationsgerät als JAR-Datei vorliegen und dem in Abschnitt 7.2.1 beschriebenen Aufbau entsprechen. Nachdem die Vorbereitungen abgeschlossen sind, kann mit der Durchführung des Testablaufs begonnen werden. Einen Überblick über den Testablauf enthält Tabelle 8.1. Sofern nicht anders angegeben, werden sämtliche Operationen mittels der Administrationsoberfläche durchgeführt.

Tabelle 8.1: Die einzelnen Schritte des Testablaufs

	Anweisung
0.	Vorbereitung, Starten des Simulator und von FiLeCC
1.	Kontrolle der Topologie
2.	Installation Testanwendung
3.	Starten auf beliebigem Knoten
4.	Prüfen der Ausgabe
5.	Entfernen des ausgewählten Knotens
6.	Kontrolle der Topologie, Prüfen der Ausgabe
7.	Entfernen des Knotens auf dem die Testanwendung neu gestartet wurde
8.	Kontrolle der Topologie, Prüfen der Ausgabe
9.	Simulieren einer Überlast auf Testanwendungsknoten im Netz mit zwei Knoten
10.	Auslesen der Auslastungsstatistik
11.	Prüfen der Ausgabe auf allen Knoten
12.	Hinzufügen der beiden einzelnen Knoten
13.	Kontrolle der Topologie
14.	Stoppen der Testanwendung auf allen Knoten
15.	Starten von zwei Instanzen der Testanwendung
16.	Entfernen des Netzwerkkabels zwischen den Switches zur Erzeugung eines Netsplits
17.	Prüfen, ob auf beiden Seiten des Netsplits je zwei Instanzen laufen
18.	Wiederverbinden der beiden Netze
19.	Kontrolle der Topologie
20.	Deinstallation der Testanwendung

Im ersten Schritt des Tests wird mit Hilfe der Administrationsoberfläche überprüft, ob die Topologie korrekt berechnet und angezeigt wird und ob UNO und Global Watchdog gewählt worden sind. Das bedeutet, dass alle beteiligten Knoten sich miteinander verbunden haben und funktionstüchtig sind. Nach positiver Bestätigung muss die Testanwendung in FiLeCC installiert werden. Nach der Installation ist die ausführbare Datei der Testanwendung von jedem Knoten aus zugänglich, aber noch nicht gestartet. Ein beliebiger Knoten wird gewählt, auf dem die Anwendung gestartet wird. Nun sieht der Testablauf vor, auf die Ausgabe der Testanwendung zu warten, um die korrekte Funktionsweise zu überprüfen.

Die Applikation läuft nun auf einem der vier Knoten. Dieser wird vom Restnetz getrennt, indem z. B. das Netzwerkkabel gezogen wird. Ergebnis dieser Operation soll sein, dass es zwei getrennte Netze gibt, eines aus einem und eines aus drei Knoten, in welchen jeweils eine Instanz des Global Watchdogs, der UNO und der Testanwendung gestartet sein soll. Die Leaderwahl der beiden Komponenten Global Watchdog und UNO bemerkt das Fehlen einer jeweiligen Instanz in einem der beiden Teilnetze und startet eine neue Wahl um dies zu korrigieren. Sobald im Netz mit drei Knoten beide Instanzen vorhanden sind, erkennt der Global Watchdog den Ausfall des vierten Knotens und gibt der UNO den Befehl eine neue Instanz der Testanwendung im Netz zu starten.

Nachdem überprüft wurde, ob die Testanwendung in beiden Netzen gestartet worden ist, und die Topologie in beiden Netzen korrekt ist, wird wiederum ein Knoten entfernt, und zwar derjenige, auf dem die neue Instanz der Testanwendung gestartet worden ist. Ergebnis dieser Operation soll sein, dass es drei Netze, eines mit zwei Knoten und zwei Netze mit je einem Knoten, gibt, in welchen jeweils eine Instanz des Global Watchdogs, der UNO und der Testanwendung läuft.

Im Netz der zwei Knoten wird nun mittels einer Hilfsfunktion der UNO eine Überlast auf dem Testanwendungsknoten simuliert und über die Administrationsoberfläche verifiziert. Die angebliche Überlast veranlasst die UNO eine weitere Instanz aller Anwendungen des überlasteten Knotens auf einem anderen Knoten des Netzes anzubieten. Die Testanwendung wird nun auch dem vierten Knoten gestartet, so dass sie auf jedem der vier Knoten der Testumgebung ausgeführt wird.

Nacheinander werden im nächsten Schritt die beiden zuvor entfernten Knoten wieder mit den anderen verbunden, so dass es nur noch ein Netz gibt. Dabei beendet die Leaderwahl jeweils überflüssige Instanzen des Global Watchdogs und der UNO, womit zum Abschluss dieses Schritts noch jeweils eine aktive Instanz dieser beiden Services existiert. Die Testanwendung läuft weiterhin auf allen vier Knoten, und nach einer Kontrolle der Ausgaben und der Topologie, wird der Befehl gegeben, die Ausführung der Testanwendung auf allen Knoten zu beenden.

Nach dem letzten Schritt ist das System in demselben Zustand wie nach *Schritt 2*, der Installation der Testanwendung. Ein weiterer Testfall ist der Netsplit, bei dem das Netz in zwei Teile getrennt wird. Zu diesem Zweck wird die Testanwendung zunächst auf zwei beliebigen Knoten gestartet. Danach wird das in Abbildung 8.1 rot dargestellte Kabel, das die beiden Switches miteinander verbindet, entfernt, so dass das Netzwerk in zwei Teile zerfällt. Der Global Watchdog und die UNO werden nach kurzer Zeit in beiden Netzen wieder zur Verfügung stehen, und dafür sorgen, dass auf beiden Seiten zwei Instanzen der Testanwendung gestartet sind.

Nach der Kontrolle dieses Zustands wird das Kabel wieder verbunden, und die beiden Netze vereinen sich wieder. Die Topologie aktualisiert sich, die Leaderwahl sorgt für die Einzigartigkeit von Global Watchdog und UNO. Die vier noch laufenden Instanzen der Testanwendung können gestoppt werden. Mit der zuletzt folgenden Deinstallation der Testanwendung ist der Testablauf beendet.

8.2 Testauswertung

Die Testumgebung demonstriert gleichermaßen Korrektheit und Vollständigkeit der vorgelegten Implementierung und präsentiert diese anhand der in Tabelle 8.1 beschriebenen anschaulichen Choreographie. Die volle Funktionsfähigkeit der erstellten Software impliziert ihre Abnahmefähigkeit und wird im Pflichtenheft (siehe Anhang A) anhand von Testfällen definiert, welche die zugesicherten Anwendungsfälle aus Kapitel 5.2 vollumfänglich abdecken. Im Folgenden wird gezeigt, dass jeder Testfall durch die beschriebene Choreographie repräsentiert wird.

Die Testfälle im Pflichtenheft beziehen sich auf vier Testwebservices. Diese decken einerseits den beweglichen sowie den unbeweglichen Fall ab, und andererseits Fälle mit Verwendung von GridGain sowie ohne Verwendung von GridGain. Die Verwendbarkeit von GridGain aus einem Webservice heraus wird automatisch durch GridGain sichergestellt und braucht nicht weiter betrachtet zu werden. Daher genügt es, die Testfälle jeweils mit einem beweglichen und einem unbeweglichen Webservice durchzuführen, soweit diese Unterscheidung beim jeweiligen

Testfall sinnvoll ist. Abweichend vom Pflichtenheft wird der bewegliche Webservice in der Choreographie zum Zwecke der Anschaulichkeit nicht anhand eines Primzahltest durchgeführt, sondern es wird auf dem ausführenden Knoten eine Musikdatei abgespielt. Weiterhin verwendet die Choreographie die Nachbarschaftserkennungsanwendung (siehe Kapitel 7.8.3), die als unbeweglicher Webservice implementiert ist und damit den unbeweglichen Fall abdeckt.

8.2.1 Auswertung der Testfälle

Im Folgenden werden die Testfälle aus dem Pflichtenheft in Anhang A im Kontext des durchgeführten Testablaufs betrachtet um die tatsächlichen Ergebnisse mit den erwarteten Ergebnissen zu vergleichen. Die Testfälle nehmen Bezug auf die Anwendungsfälle, die in Kapitel 6.1 des Pflichtenheftes definiert sind und in Kapitel 5.2 diesem Bericht aufgeführt werden. Die einzelnen Schritte des Testablaufs sind in Tabelle 8.1 beschrieben. Diese Schritte werden im Folgenden in Beziehung zu den Testfällen des Pflichtenheftes gesetzt, die in Tabelle 8.2 zusammen mit den jeweiligen Testnummern aufgeführt sind. Diese Testnummern korrespondieren wiederum mit den Nummern derjenigen Anwendungsfälle aus der Anforderungsanalyse in Kapitel 5.2, welche durch die entsprechenden Tests abgedeckt werden. Test P305 wird im Pflichtenheft auch für einen verteilt rechnenden Webservice ausgeführt. Dieser Fall wird wie erwähnt durch GridGain abgedeckt und braucht nicht zusätzlich untersucht werden.

Tabelle 8.2: Die Testfälle aus Kapitel 7 des Pflichtenheftes in Anhang A

	Bezeichnung
T101	Neues Fördermodul hinzufügen
T102	Fördermodul entfernen
T201a	Neue Rechenressourcen hinzufügen (mit unbeweglicher Anwendung)
T201b	Neue Rechenressourcen hinzufügen (mit beweglicher Anwendung)
T202	Rechenressourcen entfernen
T203	Statistiken anzeigen lassen
T301	Installation neuer Webservices
T302	Modifikation installierter Webservices
T303	Deinstallation installierter Webservices
T304	Webservice suchen
T305	Webservice nutzen
T501	Lastabhängige Verteilung von Webservices
T502	Sicherstellung der Verfügbarkeit von beweglichen Webservices

Einige der Tests aus Tabelle 8.2 korrespondieren sehr direkt mit Schritten des Testablaufs. So schließen Schritte 0 und 1, der Start des Systems sowie die Kontrolle der Topologie, bereits den Testfall T101 „Neues Fördermodul hinzufügen“ ein. Da dabei die unbewegliche Topologieanwendung installiert, gesucht und ausgeführt wird, werden außerdem die Testfälle T301, T304 und T305 für unbewegliche Webservices abgedeckt. Schritt 2 „Installation Testanwendung“, Schritt 3 „Starten auf beliebigem Knoten“ und Schritt 4 Prüfen der Ausgabe decken dementsprechend T301, T304 und T305 für bewegliche Webservices ab. Schritt 5 „Entfernen des ausgewählten Knotens“ deckt T202 ab. Der entfernte Knoten in Schritt 5 ist gleichzeitig auch Träger von Simulatoren für Fördermodule. Durch die Entfernung sind diese Fördermodule für das System nicht länger verfügbar und gelten als entfernt. Zusammen mit Schritt 6 „Kontrolle der Topologie“

entspricht dieser Schritt Testfall T102. Schritt 7 und 8 wiederholen Schritte 5 und 6 und decken Testfall T502 ab, da gezeigt wird, dass wiederholtes Entfernen einer Rechenressource nicht zu Ausfällen von Webservices führt. In Schritt 9 wird eine Überlast simuliert um die lastabhängige Verteilung von Webservices zu testen, zusammen mit der Überprüfung in Schritt 11 wird damit Testfall T501 abgedeckt. Das Auslesen der Auslastungsstatistik über die Administrationsoberfläche in Schritt 10 korrespondiert direkt mit T203. In Schritt 12 werden die vormals entfernten Knoten wieder hinzugefügt. Dabei gibt es im Netz mit dem Topologiewebservice eine unbewegliche Anwendung und mit der Testanwendung eine bewegliche Anwendung, die nach dem integrieren der neuen Knoten weiterhin korrekt ausgeführt werden. Damit sind Testfälle T201a und T201b abgedeckt. Die Modifikation eines Webservices entspricht der Deinstallation der alten Version und anschließender Installation der neuen, modifizierten Version. Dementsprechend ist dieser Testfall abgedeckt, wenn sowohl die Installation als auch die Deinstallation getestet werden. Dies ist der Fall, da in Schritt 20 die Testanwendung deinstalliert wird. Damit werden auch T302 und T303 im Testablauf repräsentiert.

Zusammengefasst deckt der Testablauf nach Tabelle 8.1 jeden laut Pflichtenheft zu erfüllenden Testfall ab, so dass die erfolgreiche Durchführung dieses Testablaufs die vollständige Erfüllung aller Testfälle zeigt. Daraus folgt, dass die Anwendungsfälle aus Kapitel 5.2 eine Teilmenge des Funktionsumfangs von FiLeCC bilden und damit die Abnahmekriterien vollumfänglich erfüllt werden.

8.2.2 Diskussion des Globaltests

Bei der Planung und Durchführung der Erprobung des Gesamtsystems sind einige Besonderheiten, Schwächen und Probleme aufgefallen, die im Folgenden aufgeführt werden.

Für die verteilte Datenhaltung (vgl. Kapitel 5.4) sind die Datenbanksysteme GridGain sowie Hazelcast ausgewählt (vgl. Kapitel 6.6), die die SDRM-Daten und die Lastdaten speichern sollen. Bei intensiven Testläufen mit unvorhersagbaren Knotenausfällen und Netztrennungen können beide Systeme nicht in jeder Situation die Konsistenz der jeweiligen Daten sicherstellen und führen daher in einigen Fällen zu fehlerhaftem Verhalten. Hazelcast-Knoten verweigeren zuweilen die Verbindung aller Knoten zu einem einzigen Cluster. Diese Fehler treten zwar vermehrt in heterogenen Umgebungen auf, d. h. im Mischbetrieb von Windows- und Linuxplattformen, eine Korrelation mit den Fehlern ist jedoch nicht zu ermitteln. Ähnliche Probleme treten im Zusammenhang mit der GridGain-Datenbank auf, die nach der Trennung einzelner Knoten oder ganzer Teilnetze nur innerhalb eines sehr begrenzten Zeitfensters die erneute Verschmelzung zu einer konsistenten Datenbank leisten kann. Da dieses Verhalten nicht dokumentiert ist, und bei kleineren Testumgebungen nicht auftritt, wurde dieses Problem zu spät erkannt und konnte im Entwurf nicht mehr berücksichtigt werden. Für die Präsentation existiert eine eigene Implementierung einer verteilten Datenhaltung für Last- und SDRM-Daten, welche die Daten auf jeden aktiven Knoten repliziert und Ausfälle anhand von Timeouts erkennt. Diese Methode hat gegenüber einer fertigen verteilten Datenhaltungslösung den Nachteil, dass sie nicht für beliebige Netzgrößen skaliert, da die Zahl der direkten Datenverbindungen quadratisch mit der Zahl der Knoten steigt. Die entsprechenden Schnittstellen zu GridGain bzw. Hazelcast sind in FiLeCC integriert. Daher können zukünftige Versionen der entsprechenden Datenbanken, in denen die genannten Probleme behoben worden sind, direkt eingebunden und verwendet werden.

Es folgen weitere Bemerkungen zu aufgetretenen Problemen und Besonderheiten, deren Lösung im Rahmen der Projektgruppe nicht oder nur in geringem Umfang möglich ist.

Es gibt keine betriebssystemunabhängige Möglichkeit einen USB Massenspeicher einzubinden und aus Java heraus auf diesen zuzugreifen, damit dieser überwacht und neue Software ausgelesen werden kann. An dieser Stelle ist die Plug&Play-Fähigkeit des Systems beeinträchtigt, da das Einbringen von neuer Software in FiLeCC einen manuellen Eingriff durch einen Administrator erfordert, der die Installation etwa über die Administrationsoberfläche anstößt.

Eine weiteres Problem ist, dass die vorhandenen Bibliotheken zur Ressourcenüberwachung keine Unterstützung für alle Rechnerarchitekturen bieten. So bietet das verwendete *Hyperic Sigar Framework* keine offizielle Unterstützung für ARM-Architekturen an, weshalb insbesondere bei der Verwendung des *Pandaboards* nur approximative Angaben, basierend auf den durch die JVM belegten Ressourcen, möglich sind.

Zuletzt bleibt darauf hinzuweisen, dass sich die verwendete DPWS-Bibliothek *JMEDS* noch im Beta Status befindet. Die Projektgruppe hat versucht Probleme in Zusammenarbeit mit der Materna GmbH zu beheben, das Auftreten weiterer Fehler kann jedoch nicht ausgeschlossen werden. Da die Förderung des *OSAMI-Projektes* ausläuft ist zudem derzeit unklar, ob und in welchem Rahmen die Entwicklung von *JMEDS* weitergeführt wird.

KAPITEL 9

Zusammenfassung und Ausblick

Die Motivation für die Projektgruppe ergibt sich daraus, dass in modernen Materialflusssystemen zunehmend Intelligenz in die Feldebene verlagert wird. Die Anzahl der Rechner sowie deren Leistung wächst dabei stetig. Oftmals sind die Rechen- und Speicherkapazitäten allerdings für ihre ursprüngliche Aufgabe überdimensioniert, sodass viele Ressourcen ungenutzt bleiben. Diese Überdimensionierung ist jedoch notwendig, da einzelne Aufgaben die ihnen exklusiv zugewiesenen Rechenkapazitäten kurzfristig vollständig ausnutzen. Die durchschnittliche Ausnutzung der Kapazitäten ist allerdings gering, sodass durch gemeinsames Nutzen aller Ressoucen Kapazitäten für zusätzliche Aufgaben entstehen. Eine Cloud-Plattform bietet die Möglichkeit, dieses Ziel zu erreichen.

Um schnell und preisgünstig auf sich ändernde Anforderungen reagieren zu können, besteht der Wunsch, Förderanlagen modular und die einzelnen Module flexibel kombinierbar zu gestalten. Ein Forschungsschwerpunkt des Lehrstuhl für Förder- und Lagerwesen der TU Dortmund ist daher die Entwicklung von *Smart Neighbourhood Modulen* (SNMs). Dabei handelt es sich um mit Rechenkapazität ausgestattete Förderelemente, die über eine einheitliche Schnittstelle mit ihren Nachbarelementen verbunden werden können und somit eine komplexere Förderkomponente bilden. Ein Anlagenbetreiber verfügt durch SNMs über eine flexible Förderanlage, die ohne zusätzlichen Konfigurationsaufwand betriebsbereit ist. Die Idee der SNMs ist in Abschnitt 4.5 beschrieben.

Das Ziel der Projektgruppe ist die Entwicklung einer Cloud-Plattform für SNMs, um die Rechenkapazität der Module zu kombinieren und freie Kapazitäten für zusätzliche Aufgaben nutzbar zu machen. Hieraus leitet sich der Name *Field-Level Computation Cloud* (FiLeCC) ab. Eine solche Cloud-Plattform unterscheidet sich vom klassischen Cloud-Computing derart, dass die Rechenleistung der einzelnen Rechenknoten deutlich geringer ist und sich deren Anzahl und Zusammensetzung dynamisch ändert. Durch *Ubiquitous Computing* ergeben sich weitere

Anwendungsbereiche, da die Anzahl ressourcenbeschränkter, miteinander vernetzter Geräte kontinuierlich steigt.

Die wichtigste Anforderung ist ein vollständig dezentrales System, das flexibel um weitere Hardwarekomponenten erweitert und verkleinert werden kann und dadurch eine hohe Skalierbarkeit bietet. Weitere Ziele sind die Realisierung von Ausfallsicherheit, Lastverteilung, automatischer Softwareverteilung, Ressourcenüberwachung und Speicherverwaltung.

Diese Anforderungen werden durch ein System aus mehreren Komponenten (siehe Kapitel 7) erfüllt, die in Java implementiert sind und über Webservices miteinander kommunizieren können. Eine zentrale Komponente, die die Ausfallsicherheit garantiert, ist das Watchdogsystem. Es stellt die Verfügbarkeit aller Rechenknoten, FiLeCC-Softwarekomponenten, sowie externer Anwendungen sicher. Stellt das Watchdogsystem einen Ausfall fest, meldet er dies an eine weitere Komponente, die UNO. Die UNO ermittelt anhand der Systemauslastung einen Rechenknoten für eine neue Instanz der betroffenen Software und stellt somit die Lastverteilung sicher. Anschließend wird die neue Instanz mittels der *Software Deployment and Runtime Management*-Komponente (SDRM) gestartet. Das SDRM verteilt des Weiteren die im System installierten Softwarepakete unter Berücksichtigung der Speicherkapazität auf die Rechenknoten und hält sie somit verfügbar. Zur Speicherung der Verwaltungsinformationen von FiLeCC-Komponenten und Anwendungen ist eine verteilte Datenbank vorgesehen. Darüber hinaus wurde ein Leaderwahlverfahren implementiert, das für bestimmte FiLeCC-Komponenten und externe Software die Ausführung genau einer Instanz garantiert.

Ein Minimalziel der Projektgruppe ist die Entwicklung einer Cloud-Plattform für ein dynamisches Netzwerk ressourcenbeschränkter Geräte, die die oben genannten Anforderungen erfüllt. Die durchgeführten Tests (siehe dazu Kapitel 8) demonstrieren gleichermaßen Korrektheit und Vollständigkeit der Implementierung, wobei alle Testfälle aus dem Pflichtenheft (siehe Anhang A) abgedeckt sind. Durch die Tests sind einige Besonderheiten und Schwächen erkannt worden (siehe Abschnitt 8.2). Beispielsweise funktionieren die ausgewählten Datenbanksysteme nicht in allen Situationen, die in hochdynamischen Systemen auftreten, korrekt.

Ein weiteres Minimalziel der Projektgruppe ist die Implementierung einer Anwendung, die die Ressourcen der Cloud mittels der Ausführungsplattform FiLeCC nutzt. Die Projektgruppe hat sich für die Entwicklung einer Anwendung zur Erkennung der Nachbarschaftsbeziehungen von SNMs und Berechnung der Topologie der Förderanlage entschieden (siehe Kapitel 7.8). Deren Funktion ist ebenfalls mittels der durchgeführten Tests gezeigt worden. Diese Anwendung ist die Grundlage zur Realisierung von Smart Neighbourhood Modulen. Die Funktionen eines Fördermoduls sowie deren Topologie werden durch FiLeCC bereits erkannt.

Für die vollständige Realisierung von SNMs fehlt jedoch eine Software, die die Funktionalitäten intelligent zusammenfügt. So sollen beispielsweise ein Geradeausförderer mit Pufferfunktion, eine Kamera und eine Weiche ihre Funktionalitäten zu einer Kontrollstation kombinieren. Diese komplexe Aufgabe bleibt für zukünftige Forschungsprojekte offen.

Eine weitere wünschenswerte Funktion, die durch die Projektgruppe nicht umgesetzt wurde, ist eine topologiebasierte Softwareverteilung. Das SDRM soll dabei anhand der berechneten Topologie von Hardwareressourcen externe Softwarepakete so verteilen, dass sie auch bei einer Netsplit-Situation in allen entstehenden Subnetzen zur Verfügung stehen, ohne dass jede Software auf jeder Ressouce gespeichert ist. Diese intelligente Verteilung von Software muss noch entwickelt werden. Des Weiteren soll die Plattform zur Laufzeit neu konfigurierbar sein. Da es sich dabei um eine Zusatzfunktion handelt, die einen großen Aufwand erfordert, da

Änderungen an Konfigurationsdateien konsistent im Netz zu propagiert werden müssen, wurde diese Funktion nicht realisiert. Externe Softwarepakete müssen derzeit in Java implementiert sein, um auf der Plattform FiLeCC ausgeführt werden zu können. Eine Unterstützung für weitere Programmiersprachen ist wünschenswert.

Rechnernetze aus ressourcenbeschränkten Geräten, die nicht optimal ausgelastet sind, finden sich nicht ausschließlich in Materialflusssystemen. Ein weiteres Beispiel sind Set-Top-Boxen, die zum Fernsehempfang über Kabel verwendet werden. Sie sind bereits in vielen privaten Haushalten installiert, verfügen über einen Breitbandinternetzugang und können in den Zeiten, in denen sie nicht durch den Fernsehempfang ausgelastet sind, ihre Ressourcen für andere Aufgaben z. B. über das Internet bereitstellen. FiLeCC bietet sich an, in Zukunft die ungenutzten Ressourcen dieser Geräte zu einer Cloud zusammenzufassen und somit zu nutzbar zu machen.

Zusammengefasst bietet die im Rahmen dieser Projektgruppe entworfene hochdynamische Cloudarchitektur, motiviert durch die aktuelle Forschung im Bereich modularer Materialflusssysteme, einen neuen forschungsrelevanten Ansatz, um überschüssige Rechenkapazitäten in dezentralen ressourcenbeschränkten Umgebungen nutzbar zu machen.

Literaturverzeichnis

[10g11a] 10gen, Inc.: *MongoDB Dokumentation - Sharding*. 2011. – http://www.mongodb.org/display/DOCS/Sharding

[10g11b] 10gen, Inc.: *MongoDB Homepage*. 2011. – http://www.mongodb.org/

[AAK+08] Arnold, D.; Arnold, D.; Kuhn, A.; Isermann, H.; Furmans, K.; Tempelmeier, H.: *Handbuch Logistik*. Springer, 2008 (VDI-Buch)

[AFG+09] Armbrust, Michael; Fox, Armando; Griffith, Rean; Joseph, Anthony D.; Katz, Randy H.; Konwinski, Andrew; Lee, Gunho; Patterson, David A.; Rabkin, Ariel; Stoica, Ion; Zaharia, Matei: Above the Clouds: A Berkeley View of Cloud Computing / EECS Department, University of California, Berkeley. Version: Feb 2009. http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html. 2009 (UCB/EECS-2009-28). – Forschungsbericht

[Aic11] Aicas GmbH: *JamaicaVM Homepage*. 2011. – http://www.aicas.com/jamaica.html

[Ama06] Amazon Web Services LLC: *Announcing Amazon Elastic Compute Cloud (Amazon EC2) - beta*. 2006. – http://aws.amazon.com/de/about-aws/whats-new/2006/08/24/announcing-amazon-elastic-compute-cloud-amazon-ec2---beta/, Stand 4.5.2012

[Ama11] Amazon Web Services LLC: *Amazon EC2 Produkthomepage*. 2011. – http://aws.amazon.com/de/ec2/, Stand: 13.08.2011

[Ama12] Amazon Web Services LLC: *Amazon EC2 SLA*. 2012. – http://aws.amazon.com/de/ec2-sla/, Stand: 21.04.2012

[And06] Anderson, Chris: *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion, 2006

[Apa04] Apache Software Foundation: *Apache Axis2/Java*. 2004. – http://axis.apache.org/axis2/java/core/, Stand: 23.08.2011

[Apa10] Apache Software Foundation: *HDFS Users Guide*. 2010. – http://hadoop.apache.org/hdfs/docs/current/hdfs_user_guide.html

[Apa11a] Apache Software Foundation: *Apache Harmony Homepage*. 2011. – http://harmony.apache.org/

[Apa11b] Apache Software Foundation: *Cassandra Hardware*. 2011. – http://wiki.apache.org/cassandra/CassandraHardware, Stand: 16.04.2012

[Apa11c] Apache Software Foundation: *CouchDB Homepage*. online, 2011. – http://couchdb.apache.org/, Stand: 05.03.2012

[Apa11d] Apache Software Foundation: *Hadoop Projekthomepage*. 2011. – http://hadoop.apache.org/, Stand:13.08.2011

[Axe06] Axelson, Jan: *USB 2.0 Handbuch für Entwickler*. zweite Auflage. mitp Professional, 2006

[Bas12a] Basho Technologies, Inc.: *Riak – Client Libraries*. online, 2012. – http://wiki.basho.com/Client-Libraries.html, Stand: 05.03.2012

[Bas12b] Basho Technologies, Inc.: *Riak – Concepts*. online, 2012. – http://wiki.basho.com/Concepts.html, Stand: 05.03.2012

[BBB+08] Ballinger, Keith; Bissett, Bobby; Box, Don; Ferguson, Don; others; Curbera, Francisco (Hrsg.); Schlimmer, Jeffrey (Hrsg.); Parastatidis, Savas (Hrsg.): *Web Services Metadata Exchange 1.1 (WS-MetadataExchange)*. 2008. – http://www.w3.org/Submission/WS-MetadataExchange/

[BBG06] Bohn, Hendrik; Bobek, Andreas; Golatowski, Frank: SIRENA - Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains. In: *Mobile Communications and Learning Technologies, Conference on Networking, Conference on Systems, International Conference on* (2006), S. 43

[BCC+06] Box, Don; Cabrera, Luis F.; Critchley, Craig; Curbera, Francisco; others: *Web Services Eventing (WS-Eventing)*. 2006. – http://www.w3.org/Submission/WS-Eventing/

[Bea10] BeagleBoard.org: *BeagleBoard-xM Rev A2 System Reference Manual*. Revision 0.1, Juli 2010. – http://focus.ti.com/lit/ug/spruf98p/spruf98p.pdf

[BG+12] Beck, Kent; Gamma, Erich; others: *JUnit*. 2012. – http://junit.sourceforge.net/, Stand: 02.03.2012

[BHM+04] Booth, David; Haas, Hugo; McCabe, Francis; Newcomer, Eric; Champion, Michael; Ferris, Chris; Orchard, David: *Web Services Architecture*. 2004. – http://www.w3.org/TR/ws-arch/

[Bis06] Bisenius, William S.: Ingress Protection: The System of Tests and Meaning of Codes. In: *Compliance Engineering* (2006)

[BKN+11] Baun, Christian; Kunze, Marcel; Nimis, Jens; Tai, Stefan; Günter, O. (Hrsg.); Karl, W. (Hrsg.); Lienhart, R. (Hrsg.); Zeppenfeld, K. (Hrsg.): *Cloud Computing: Web-basierte dynamische IT-Services*. Springer-Verlag, 2011

[Bor08] Bornstein, Dan: *Google I/O 2008 – Dalvik Virtual Machine Internals*. 2008. – https://sites.google.com/site/io/dalvik-vm-internals

[BP10] Becker, Arno; Pant, Marcus: *Android 2 – Grundlagen und Programmierung*. 2., aktualisierte und erweiterte Auflage. dpunkt.verlag, 2010

[CAC11] CACAOVM - Verein zur Förderung der freien virtuellen Maschine CACAO: *CACAOVM Homepage*. 2011. – http://www.cacaojvm.org/

[CB10] Cheng, Ben; Buzbee, Bill: *Google I/O 2010 – A JIT Compiler for Android's Dalvik VM*. 2010. – http://www.google.com/events/io/2010/sessions/jit-compiler-androids-dalvik-vm.html

[CCM+01] Christensen, Erik; Curbera, Francisco; Meredith, Greg; Weerawarana, Sanjiva: *Web Services Description Language (WSDL) 1.1*. 2001. – http://www.w3.org/TR/wsdl

[CDG+06] Chang, Fay; Dean, Jeffrey; Ghemawat, Sanjay; Hsieh, Wilson C.; Wallach, Deborah A.; Burrows, Mike; Chandra, Tushar; Fikes, Andrew; Gruber, Robert E.: Bigtable: A Distributed Storage System for Structured Data. In: *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*. Berkeley, CA, USA: USENIX Association, 2006 (OSDI '06), S. 205–218

[Cit05] Citrix Systems, Inc.: *Xen hypervisor, the powerful open source industry standard for virtualization*. 2005. – http://xen.org/, Stand 4.5.2012

[Cod90] Codd, E. F.: *The relational model for database management: version 2*. Addison-Wesley Longman Publishing Co., Inc., 1990

[CW76] Curnow, H. J.; Wichmann, B. A.: A synthetic benchmark. In: *Computer Journal* 19 (1976), Nr. 1, S. 43–49. – http://freespace.virgin.net/roy.longbottom/whetstone.pdf

[Dam10] Damn Small Linux: *FAQ – http://www.damnsmalllinux.org/wiki/index.php/FAQ*. 07 2010. – unpublished

[Dam12] Damn Small Linux: *DSL Information*. 2012. – http://www.damnsmalllinux.org/, Stand 01.06.2012

[Deb11] Debian Project: *Debian 6.0 »Squeeze« veröffentlicht*. 2011. – http://www.debian.org/News/2011/20110205a.de.html

[DG08] Dean, J.; Ghemawat, S.: MapReduce: Simplified data processing on large clusters. In: *Communications of the ACM* 51 (2008), Nr. 1, S. 107–113

[DHJ+07] DeCandia, Giuseppe; Hastorun, Deniz; Jampani, Madan; Kakulapati, Gunavardhan; others: Dynamo: Amazon's Highly Available Key-Value Store. In: Stevenson (Hrsg.): *Proceedings of the 21st ACM Symposium on Operating Systems Principles*, 2007

[DM09] Driscoll, Dan; Mensch, Antoine: *Devices Profile for Web Services Version 1.1*. 2009. – http://docs.oasis-open.org/ws-dd/dpws/wsdd-dpws-1.1-spec.html

[EG02] van Engelen, Robert A.; Gallivan, Kyle: The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks. In: *Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid2002)* (2002), S. 128–135. – http://www.cs.fsu.edu/~engelen/soap.html, Stand: 23.08.2011

[EGL+76] Eswaran, K. P.; Gray, J. N.; Lorie, R. A.; Traiger, I. L.: The notions of consistency and predicate locks in a database system. In: *Commun. ACM* 19 (1976), November, Nr. 11, 624–633. http://dx.doi.org/10.1145/360363.360369. – DOI 10.1145/360363.360369. – ISSN 0001–0782

[Ein10] Einspanjer, Daniel: *Riak and Cassandra and HBase, oh my!* Mai 2010. – http://blog.mozilla.com/data/2010/05/18/riak-and-cassandra-and-hbase-oh-my/, Stand 03.04.2011

[Emb11a] Embedded Debian Project: *Emdebian Distributions*. 2011. – http://www.emdebian.org/emdebian/flavours.html

[Emb11b] Embedded Debian Project: *Emdebian GNU/Linux Grip 2.0 (based on Debian 6.0 squeeze) released*. 2011. – http://emdebian.org/News/2011/20110205.html

[Emb11c] Embedded Debian Project: *Emdebian Grip packages*. 2011. – http://emdebian.org/grip/

[Env11] Environmental Systems Research Institute: *ArcGIS Cloud*. 2011. – http://www.esri-germany.de/products/arcgis/arcgis10/features.html#f8, Stand: 13.08.2011

[Euc12] Eucalyptus Systems, Inc.: *Eucalyptus Community*. 2012. – http://open.eucalyptus.com/, Stand 4.5.2012

[FFt10] Feldhorst, Sascha; Fiedler, Martin; ten Hompel, Michael: Paket Royale – Dezentrale Steuerung für das Internet der Dinge. In: *Tagungsband zum 6. Fachkolloquium der Wissenschaftlichen Gesellschaft für Technische Logistik (WGTL)*, 2010, S. 94–105

[FLt+09] Feldhorst, Sascha; Libert, Sergey; ten Hompel, Michael; Krumm, Heiko: Integration of a legacy automation system into a SOA for devices. In: *ETFA'09: Proceedings of the 14th IEEE international conference on Emerging technologies & factory automation*, IEEE Press, 2009, S. 782–789

[Fre10] Free Software Foundation, Inc.: *Results of comparison between icedtea6 and classpath*. 2010. – http://builder.classpath.org/japi/openjdk6-classpath.html

[Fre11] Free Software Foundation, Inc.: *GCJ Homepage*. 2011. – http://gcc.gnu.org/java/

[FS10] Freund, Marco; Schäfers, David: *Realisierung der Datenübertragung in einer Miniaturfördertechnik durch Einsatz von Radio Frequency Identification*. 2010. – Seminararbeit

[Gam95] Gamma, Erich: *Design Patterns*. Boston: Addison-Wesley, 1995

[Gee01] Gee, Dave: The how's and why's of PC based control. In: *Pulp and Paper Industry Technical Conference, 2001 Conference Record of.*, 2001, S. 67–74

[GGL03] Ghemawat, S.; Gobioff, H.; Leung, S.T.: The Google file system. In: *ACM SIGOPS Operating Systems Review* Bd. 37 ACM, 2003, S. 29–43

[GHJ+94] Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994

[GHM+07] Gudgin, Martin; Hadley, Marc; Mendelsohn, Noah; Jean-Jacques; others: *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. 2007. – http://www.w3.org/TR/soap12-part1/

[GHR06] Gudgin, Martin; Hadley, Marc; Rogers, Tony: *Web Services Addressing 1.0 – Core*. 2006. – http://www.w3.org/TR/ws-addr-core

[GL02] Gilbert, Seth; Lynch, Nancy: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. In: *ACM SIGACT News* 33 (2002), Nr. 2. – http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.20.1495&rep=rep1&type=pdf, Stand 11.07.2011

[Goo07] Google Inc.: *Androidology – Part 1 of 3 – Architecture Overview*. 2007. – http://www.youtube.com/watch?v=Mm6Ju0xhUW8

[Goo11a] Google, Inc.: *Google App Engine*. 2011. – http://code.google.com/intl/de-DE/appengine/, Stand: 13.08.2011

[Goo11b] Google Inc.: *What is Android?* 2011. – http://developer.android.com/guide/basics/what-is-android.html

[Goo12] Google Inc.: *Google Maps*. 2012. – http://maps.google.de/, Stand 5.5.2012

[Gri05] GridGain Systems, Inc.: *GridGain Projekthomepage*. 2005. – http://www.gridgain.com/, Stand:13.08.2011

[Gri12a] GridGain Systems: *GridCacheHibernateBlobStore*. online, 2012. – http://www.gridgain.com/javadoc30C/org/gridgain/grid/cache/store/hibernate/GridCacheHibernateBlobStore.html, Stand: 07.03.2012

[Gri12b] GridGain Systems: *GridCacheJdbcBlobStore*. online, 2012. – http://www.gridgain.com/javadoc30C/org/gridgain/grid/cache/store/hibernate/GridCacheJdbcBlobStore.html, Stand: 07.03.2012

[GS08] Gumm, H. P.; Sommer, Manfred: *Einführung in die Informatik*. 8rd. Oldenbourg Verlag München, 2008

[Gt10] Günthner, Willibald; ten Hompel, Michael: *Internet der Dinge in der Intralogistik*. Springer-Verlag Berlin Heidelberg, 2010

[Gua10] Gualtieri, Mike: *NoSQL And Elastic Caching Platforms Are Kissing Cousins*. Februar 2010. – http://blogs.forrester.com/application_development/2010/02/nosql.html

[Gum11a] Gumstix Inc.: *Chestnut43 Product Overview*. 2011. – http://www.gumstix.com/store/catalog/product_info.php?products_id=237, Stand: 15.03.2011

[Gum11b] Gumstix Inc.: *Gumstix small open source hardware*. 2011. – https://www.gumstix.com/index.html, Stand 06.05.201

[Gum11c] Gumstix Inc.: *Overo Earth COM Product Overview*. 2011. – http://www.gumstix.com/store/catalog/product_info.php?products_id=211, Stand: 15.03.2011

[Gum11d] Gumstix Inc.: *Tobi Product Overview*. 2011. – http://www.gumstix.com/store/catalog/product_info.php?products_id=230, Stand: 15.03.2011

[Haz11] Hazel Ltd.: *Hazelcast Documentation*. 2009-2011. – http://www.hazelcast.com/documentation.jsp

[HDR07] Hauser, Norbert; Dumsky, Günther; Rupp, Stephan: Trends und Technologien im Bereich Embedded Computing. In: *Information Management & Consulting* (2007), S. 36–43

[Hes04] Hesse, Friedrich W.: *e-teaching.org*. online, 2004. – http://www.e-teaching.org/glossar/proprietaer, Stand: 04.03.2012

[Heu07] Heutschi, Roger: *Serviceorientierte Architektur – Architekturprinzipien und Umsetzung in der Praxis*. Springer, 2007

[HMS06] HMS Industrial Networks GmbH: *Modbus-TCP*. online, 2006. – http://www.anybus.de/technologie/modbustcp.shtml, Stand: 04.03.2012

[Hyp11] Hypertable Inc.: *The Hypertable Scalable Database Manual*, 2011. – http://code.google.com/p/hypertable/wiki/HypertableManual, Stand: 16.04.2012

[Int06] Internet Society: *The application/json Media Type for JavaScript Object Notation (JSON)*. online, 2006. – https://www.ietf.org/rfc/rfc4627.txt, Stand: 07.03.2012

[IPC10] IPC2U GmbH: *iROBO Industrie Computer Systeme*. 2010. – http://www.ipcpro.de/catalog/R/description.html, Stand: 30.03.2011

[ITE12] ITEA Office Association: *ITEA2*. 2012. – http://www.itea2.org/project/index/view/?project=230, Stand: 15.04.2012

[JG11] Jansen, Wayne; Grance, Timothy: *NIST – Guidelines on Security and Privacy in Public Cloud Computing*. 2011

[KH10] Kroah-Hartmans, Greg: *Presentation on Android, CELF conference 2010*. 2010. – https://github.com/gregkh/android-presentation/zipball/CELF_2010_v2

[Len09] Lennon, Joe: *Exploring CouchDB*. 2009. – http://www.ibm.com/developerworks/opensource/library/os-couchdb/index.html

[Ler05] Lerch, Reinhard: Rechnergestützte Messdatenerfassung. In: *Elektrische Messtechnik*. Springer Berlin Heidelberg, 2005 (Springer-Lehrbuch), S. 397–534

[Lin12] Linux KVM: *Kernel Based Virtual Machine Project Website*. 2012. – http://www.linux-kvm.org/, Stand 4.5.2012

[LKN+09] Lenk, A.; Klems, M.; Nimis, J.; Tai, S.; Sandholm, T.: What's inside the Cloud? An architectural map of the Cloud landscape. In: *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing* IEEE Computer Society, 2009, S. 23–31

[LM10] Lakshman, Avinash; Malik, Prashant: Cassandra - A Decentralized Structured Storage System. In: *SIGOPS Oper. Syst. Rev.* 44 (2010), April, S. 35–40

[Lou11] Lougher, Robert: *JamVM Homepage*. 2011. – http://jamvm.sourceforge.net/

[LY99] Lindholm, Tim; Yellin, Frank: *The Java(TM) Virtual Maschine Specification*. Second Edition. Prentice Hall, 1999

[Maz10] de la Maza, Silvia: *CREATE: Creating Evolution Capable Co-operating Applications in Industrial Automation*. 2010. – Full Project Proposal, ITEA 2

[Mel10] Melzer, Ingo: *Service-orientierte Architekturen mit Web Services*. 4. Auflage. Spektrum Akademischer Verlag, 2010

[Mer10] Mertes, Wilbert: Industrietaugliche Mainboards ermöglichen zuverlässige IPC. In: *Maschinenmarkt* (2010)

[Mic11a] Microsoft Corporation: *Windows Azure*. 2011. – http://www.microsoft.com/de-de/azure/, Stand: 13.08.2011

[Mic11b] Microsoft Corporation: *Windows Azure Produkthomepage*. 2011. – http://www.microsoft.com/windowsazure/

[Mic12] Microsoft Corporation: *Windows Live*. 2012. – https://home.live.com/, Stand 5.5.2012

[Mil11] Mills, Tarquin: *Whetstone Benchmark Java Implementierung*. 2011. – http://www.aicas.com/download/Whetstone.java

[MIP11] MIPS Technologies: *MIPSANDROID*. 2011. – http://www.mipsandroid.org/

[MK09] Modi, Vipul; Kemp, Devon: *Web Services Dynamic Discovery (WS-Discovery) Version 1.1*. 2009. – http://docs.oasis-open.org/ws-dd/discovery/1.1/wsdddiscovery-1.1-spec.html

[MLM+06] MacKenzie, C. M.; Laskey, Ken; McCabe, Francis; Brown, Peter F.; others: *Reference Model for Service Oriented Architecture 1.0*. 2006. – http://docs.oasis-open.org/soa-rm/v1.0/

[Mod06] Modbus Organization: *MODBUS Protocol Specification*. 2006. – http://modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf, Stand: 02.03.2012

[Mui11] Muir, Pete: *What is Infinispan?* 2011. – https://docs.jboss.org/author/pages/viewpage.action?pageId=3737165

[NL97] Newbold, Dave; Lipton, Russ: Notes: A sustainable platform architecture. In: *Iris Today Webzine* (1997). – http://www.ibm.com/developerworks/lotus/library/ls-Notes_platform_architecture/, Stand 4.5.2012

[OAS11] OASIS: *OASIS Homepage*. 2011. – http://www.oasis-open.org/org

[Ope02] OpenNebula Project Leads: *OpenNebula: The Open Source Solution for Data Center Virtualization*. 2002. – http://www.opennebula.org/, Stand 4.5.2012

[Ope06] OpenID Foundation: *OpenID Foundation website*. 2006. – http://openid.net/, Stand 5.5.2012

[Ora05a] Oracle Corporation: *CLDC HotSpot Implementation Virtual Machine*. 2005. – http://java.sun.com/j2me/docs/pdf/CLDC-HI_whitepaper-February_2005.pdf

[Ora05b] Oracle Corporation: *Connected Device Configuration (CDC) HotSpot Implementation*. 2005. – http://java.sun.com/j2me/docs/cdc_hotspotds.pdf

[Ora09] Oracle Corporation: *phoneME FAQ*. 2009. – http://java.net/projects/phoneme/sources/svn/content/trunk/www/phone_me_faq.html?rev=20546

[Ora11a] Oracle Corporation: *Java ME Technology*. 2011. – http://www.oracle.com/technetwork/java/javame/tech/index.html

[Ora11b] Oracle Corporation: *Java ME Technology – CDC*. 2011. – http://www.oracle.com/technetwork/java/javame/tech/index-jsp-139293.html

[Ora11c] Oracle Corporation: *Java SE 6 Documentation*. 2011. – http://download.oracle.com/javase/6/docs/index.html

[Ora11d] Oracle Corporation: *Java SE 6 Performance White Paper*. 2011. – http://java.sun.com/performance/reference/whitepapers/6_performance.html

[Ora11e] Oracle Corporation: *Java Virtual Machine Technology*. 2011. – http://download.oracle.com/javase/6/docs/technotes/guides/vm/index.html

[Ora11f] Oracle Corporation: *OpenJDK FAQ*. 2011. – http://openjdk.java.net/faq/

[Ora11g] Oracle Corporation: *Subversion Log für Revision 20546*. 2011. – http://java.net/projects/phoneme/sources/svn/content/trunk/www/content/phoneme_platforms.html?rev=20546

[Osk09] Oskarsson, Johan: *NOSQL meetup*. 2009. – http://blog.oskarsson.nu/2009/05/nosql-meetup.html, Stand 30.4.2012

[Pan11] PandaBoard.org: *PandaBoard Product Page*. 2011. – http://www.pandaboard.org/, Stand: 20.03.2011

[Pop08] Popp, Gunther: *Konfigurationsmanagement mit Subversion, Ant und Maven*. zweite aktualisierte Auflage. dpunkt.verlag, 2008

[Pro12] Project Voldemort: *Project Voldemort – Design*. online, 2012. – http://project-voldemort.com/design.php, Stand: 05.03.2012

[Rem12] Remember The Milk Pty Ltd: *Remember The Milk: Online to-do list and task management*. 2012. – http://www.rememberthemilk.com/, Stand 5.5.2012

[RH12] Red Hat, Inc.: *GlusterFS Dokumentation*. 2012. – http://gluster.org/community/documentation/index.php/Main_Page, Stand: 01.06.2012

[RRW12] Rotgeri, Mathias; Rupflin, Thomas; Werth, Michael: *Konstruktion und Realisierung einer modularen Miniaturfördertechnik*. 2012. – Fachwissenschaftliche Projektarbeit

[Sal00] Salesforce.com Inc.: *CRM, the cloud, and the social enterprise - Salesforce.com*. 2000. – http://www.salesforce.com/, Stand 5.5.2012

[Sch03] Schiller, Jochen: *Mobilkommunikation*. 2rd. Pearson Studium, 2003

[Sch08] Schneider Electric: *XBTN/R, Modbus Slave-Protokoll*. online, 2008. – http://www.downloads.schneider-electric.com/sites/oreo/de/document-detail.page?p_docId=2020146&p_Conf=i#http://www.schneider-electric.de, Stand: 31.05.2012

[SF10] Sadowsky, Volker; Feldhorst, Sascha: Paket Royale – Dezentrales Steuerungskonzept für das Internet der Dinge. In: *Tagungsband zum 5. BVL Wissenschaftssymposium Logistik Strukturwandel in der Logistik*, 2010, S. 296–306

[Sie10] Siemens AG: *SIMATIC IPC – Mehr Industrie-PC*. 2010. – http://www.automation.siemens.com/mcms/pc-based-automation/de/industrie-pc/highlights/Seiten/home.aspx

[Sli12] SliTaz GNU/Linux: *http://www.slitaz.org/*. 2012. – unpublished

[Sma12] Smart Software Solutions GmbH: *IEC 61131-3 Development System (IDE) CoDeSys*. 2012. – http://www.3s-software.com/index.shtml?en_CoDeSysV3_en, Stand: 06.05.2012

[Sta03] Stahl, Michael: *Java USB API for Windows*, ETH Zürich, Diplomarbeit, 2003. – http://www.steelbrothers.ch/jusb/, Stand: 04.03.2012

[Sun03] Sun Microsystems: *Connected Limited Device Configuration Specification*. Version 1.1. 2003

[Sur10a] Surtani, Manik: *Data-as-a-Service: The data fabric for clouds*. 05 2010. – http://java.dzone.com/articles/data-service-data-fabric

[Sur10b] Surtani, Manik: *Infinispan Roadmap*. 2010. – http://community.jboss.org/wiki/InfinispanRoadmap

[Tan07] Tanenbaum, Andrew S.: *Modern Operating Systems*. dritte Auflage. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007

[tBF08] ten Hompel, Michael; Büchter, Hubert; Franzke, Ulrich: *Identifikationssysteme und Automatisierung*. Springer-Verlag Berlin Heidelberg, 2008

[TBH11] Tietz, Vincent; Blichmann, Gregor; Hübsch, Gerald: Cloud Entwicklungsmethoden. In: *Informatik Spektrum* 34 (2011), Nr. 4

[tel10] teltarif.de Onlineverlag GmbH: *USB 3.0 kommt mit angezogener Handbremse*. online, 2010. – http://www.teltarif.de/usb-3-0-kommt-mit-angezogener-handbremse/news/x12143.html, Stand: 04.03.2012

[Tex11] Texas Instruments Inc.: *OMAP35x To AM37x Hardware Migration Guide*. 2011. – http://processors.wiki.ti.com/index.php/OMAP35x_To_AM37x_Hardware_Migration_Guide, Stand: 30.03.2011

[Tex12] Texas Instruments Inc.: *OMAP Mobile Processors*. 2012. – http://www.ti.com/general/docs/gencontent.tsp?contentId=46946&DCMP=WTBU&HQS=Other+OT+omap, Stand: 06.05.2012

[tFF11] ten Hompel, Michael; Feldhorst, Sascha; Fiedler, Martin: Paket Royale – Dezentrale Steuerung für das Internet der Dinge. In: *Logistics Journal* (2011)

[The12] The GNU Operating System: *GNU Lesser General Public License v3.0 - GNU Project - Free Software Foundation (FSF)*. online, 2012. – http://www.gnu.org/licenses/lgpl.html, Stand: 04.03.2012

[tNF+11] ten Hompel, Michael; Nettsträter, Andreas; Feldhorst, Sascha; Schier, Arkadius: Engineering von modularen Förderanlagen im Internet der Dinge. In: *at-Automatisierungstechnik* 59 (2011), Nr. 4, S. 248–256

[Top02] Topley, Kim: *J2ME in a Nutshell*. O'Reilly, 2002

[tS05] ten Hompel, Michael; Schmidt, Thorsten: Materialflussautomatisierung. In: *Warehouse Management*. Springer Berlin Heidelberg, 2005 (VDI-Buch), S. 157–202

[tS10] ten Hompel, Michael; Schmidt, Thorsten: *Warehouse Management. Organisation und Steuerung von Lager- und Kommissioniersystemen*. Springer-Verlag Heidelberg, 2010

[tSN07] ten Hompel, Michael; Schmidt, Thorsten; Nagel, Lars: *Materialflusssysteme. Förder- und Lagertechnik*. Springer-Verlag Berlin Heidelberg, 2007

[Uni11a] University of Chicago: *Globus Toolkit Projekthomepage*. 2011. – http://www.globus.org/toolkit/, Stand: 13.08.2011

[Uni11b] Universität Rostock: *gSOAP*. 2011. – http://ws4d.e-technik.uni-rostock.de/gsoap/, Stand: 23.08.2011

[Uni11c] Universität Rostock: *JMEDS*. 2011. – http://ws4d.e-technik.uni-rostock.de/jmeds/, Stand: 23.08.2011

[Uni11d] Universität Rostock: *WS4D Axis 2*. 2011. – http://ws4d.e-technik.uni-rostock.de/axis2/,Stand: 23.08.2011

[Uni11e] Universität Rostock: *WS4D Axis 2*. 2011. – http://trac.e-technik.uni-rostock.de/projects/ws4d-axis2,Stand: 15.04.2012

[Ver07a] Verein Deutscher Ingenieure (VDI): *Entwurf VDI 4440, Blatt 1 – Übersichtsblätter Stetigförderer für Stückgut – Bandförderer*. 2007

[Ver07b] Verein Deutscher Ingenieure (VDI): *Entwurf VDI 4440, Blatt 2 – Übersichtsblätter Stetigförderer für Stückgut – Kettenförderer*. 2007

[Ver07c] Verein Deutscher Ingenieure (VDI): *Entwurf VDI 4440, Blatt 3 – Übersichtsblätter Stetigförderer für Stückgut – Rollen- und Kugelbahnen*. 2007

[Ver07d] Verein Deutscher Ingenieure (VDI): *Entwurf VDI 4440, Blatt 5 – Übersichtsblätter Stetigförderer für Stückgut – Hängeförderer*. 2007

[Ver07e] Verein Deutscher Ingenieure (VDI): *Entwurf VDI 4440, Blatt 6 – Übersichtsblätter Stetigförderer für Stückgut – Vertikalförderer*. 2007

[VMw12a] VMware, Inc.: *Hyperic SIGAR API*. 2012. – http://www.hyperic.com/products/sigar, Stand: 22.04.2012

[VMw12b] VMware, Inc.: *VMware Deutschland - Virtualisierung der Business-Infrastruktur*. 2012. – http://www.vmware.com/de, Stand 4.5.2012

[VO10] Vollmari, Dennis; Ochowiak, Gregor: *Entwicklung und Bau einer Miniaturfördertechnik zur Veranschaulichung einer dezentralen Steuerung – Geradeausförderer*. 2010. – Studienarbeit

[VOH+07] Vedamuthu, Asir S.; Orchard, David; Hirsch, Frederick; Hondo, Maryann; others: *Web Services Policy 1.5 – Framework*. 2007. – http://www.w3.org/TR/ws-policy

[WAG07] WAGO Kontakttechnik GmbH: *Das WAGO-I/O-SYSTEM 750/753 – Ein System für alle Anwendungen*, 2007

[WAG11a] WAGO Kontakttechnik GmbH: *WAGO-I/O-IPC-G2 Linux 2.6 Datenblatt*, Februar 2011

[WAG11b] WAGO Kontakttechnik GmbH: *WAGO-I/O-IPC-P14 Linux 2.6 Datenblatt*, Februar 2011

[Wei11] Weicker, Reinhold P.: *Dhrystone Benchmark Java Implementierung*. 2011. – http://www.aicas.com/download/Dhrystone.java

[WHS+11] Wilp, Johannes; Hahn, Jannik; Szukat, Philipp; Wellinghausen, Tobias: *Entwicklung und Bau einer Miniaturfördertechnik zur Veranschaulichung einer dezentralen Steuerung – Drehtisch*. 2011. – Studienarbeit

[Wil09] Williams, Neil: *Crush 2.0 abandoned*. 2009. – http://lists.debian.org/debian-embedded/2009/08/msg00005.html

[Wil11] Williams, Neil: *Debian Wiki – Embedded Debian*. 2011. – http://wiki.debian.org/Embedded_Debian

[WS412] WS4D: *Web Services for Devices (WS4D)*. 2012. – urlhttp://ws4d.e-technik.uni-rostock.de/about/, Stand: 15.04.2012

[Xam11] Xamarin: *Mono Web Services Enhancements*. 2011. – http://www.mono-project.com/WSE, Stand: 22.08.2011

ANHANG A

Pflichtenheft

Pflichtenheft

PG 557 – Field Level Computation Cloud

Fakultät für Informatik, TU Dortmund

Martin Bring, Iryna Denysenko, Katharina Diekmann, Tim Düllmann, René Grzeszick, Katrin Holterbork, Sebastian Homann, Henning Meister, Stanislav Nikolov, Marco Pfahlberg, Dirk Schalge, Nils Schmidt

Betreuer: Sascha Feldhorst und Christian Mosblech

Sommersemester 2011
17. Juli 2011
Version 1.4

Inhaltsverzeichnis

1 Einleitung

Dieses Pflichtenheft dokumentiert die Anforderungen der von der Projektgruppe 557 zu entwickelnden Software. Diese Software wird im Folgenden Produkt genannt.

In diesem Kapitel wird zunächst das Produkt beschrieben sowie die Motivation für dessen Entwicklung skizziert. In Kapitel 2 werden die in diesem Dokument verwendeten Fachbegriffe kurz erläutert. Daran anschließend werden in Kapitel 3 die Anforderungen an das Produkt beschrieben sowie in Kapitel 4 der Produkteinsatz und in Kapitel 5 die Produktumgebung spezifiziert. Die Produktfunktionen und dessen geplante Umsetzungen werden in Kapitel 6 beschrieben. Den Abschluss dieses Pflichtenhefts bildet Kapitel 7, das zu den zuvor definierten Produktfunkionen geeignete Testszenarien beschreibt, um die korrekte Umsetzung der Produktfunktionen überprüfen zu können.

Bei der Planung von neuen automatisierten Förderanlagen ist derzeit der Trend der Modularisierung zu beobachten. Diese soll die Flexibilität der Anlagen erhöhen und Stillstandszeiten bei Änderungen an der Anlagentopologie reduzieren. Neben der Modularisierung der Hardware in einzelne Module kann auch die Steuerungssoftware modularisiert und dezentral auf eingebetteten Rechensystemen ausgeführt werden. Zu diesem Zweck besitzen bereits heute viele Anlagenteile eigene Recheneinheiten wie z. B. Industrie-PCs (IPCs), die die dezentrale Ausführung von Software ermöglichen. Die Steuerungssoftware wird dabei von den einzelnen Anlagenteilen mitgeliefert. Durch mehrere IPCs, die von verschiedenen Anlagenteilen mitgeliefert werden, entsteht ein Netzwerk, das über viele Rechen- und Speicherressourcen verfügt. Es zeichnet sich derzeit ab, dass die dezentral auf den IPCs ausgeführten und von den Anlagenteilen mitgelieferten Steuerungsfunktionen das vorhandene Ressourcenpotential bei Weitem nicht ausschöpfen.

Das Produkt soll die Nutzung des bisher ungenutzten Ressourcenpotentials ermöglichen. Dazu soll ein Cloud-basierter Ansatz verwendet werden, der von den einzelnen Recheneinheiten abstrahiert und nach außen hin eine einheitliche Schnittstelle zur Nutzung der einzelnen Ressourcen anbietet. Das Produkt ist demnach als *Platform as a Service* (PaaS) zu verstehen. Es stellt eine Ausführungsplattform für Webservices bereit, die lastabhängig innerhalb der Cloud auf wenig beanspruchte Ressourcen verteilt werden. Die Software, die durch das Produkt ausgeführt werden soll, muss in Form von Webservices vorliegen. Im Rahmen der ersten Realisierung sollen zunächst grundlegende/globale Verwaltungskomponenten, die für die Steuerung einer Anlage benötigt werden, innerhalb des Produkts ausgeführt werden. Dazu gehören beispielsweise die Topologieerkennung oder die Bereitstellung von Konfigurationsdaten für die Steuerung. Für die Zukunft sind dezentrale Ausprägungen übergeordneter Steuerungssysteme angedacht, wie z. B. WMS oder ERP-Systeme, welche über die von diesem Produkt definierte Schnittstelle die Ressourcen innerhalb der Cloud nutzen.

Zusätzlich zu der Bereitstellung der Ressourcen der Cloud durch eine einheitliche Schnittstelle soll das Produkt die dynamische Erweiterung der Förderanlage um neue Module

und somit auch zusätzliche Ressourcen unterstützen. Diese neuen Ressourcen sollen automatisch in die bestehende Cloud integriert werden können. Weiterhin soll es auch möglich sein, Module und zugehörige Ressourcen zur Laufzeit zu entfernen.

2 Glossar

In diesem Kapitel werden einige wichtige Fachbegriffe aus den Bereichen Informatik und Logistik definiert. Auf diese Definitionen beziehen sich die verwendeten technischen Begriffe im weiteren Verlauf des Pflichtenhefts.

Cloud Eine Cloud ist eine abstrakte IT-Infrastruktur, die Dienste in Form von Rechen- oder Speicherkapazität zur Verfügung stellt. Diese Dienste kann ein Benutzer in Anspruch nehmen. Der eigentliche Aufbau der Architektur bleibt dem Benutzer verborgen. Wird Rechenkapazität zur Verfügung gestellt, so ist von *Platform as a Service* (PaaS) die Rede. Zur Verfügung gestellte Speicherkapazität wird als *Infrastructure as a Service* (IaaS) bezeichnet.

CoDeA Kurzform für *Controlled Device Application*. Vom Lehrstuhl für Förder- und Lagerwesen der TU Dortmund entwickelte Laufzeitumgebung für die Bereitstellung der Anlagenfunktionen in Form von Webservices als Basis für die dezentrale Steuerung einer Förderanlage.

CPU Kurzform für *Central Processing Unit*. Die CPU ist die zentrale Verarbeitungseinheit in einem Computer. Sie ist in der Lage, Berechnungen durchzuführen oder Programme auszuführen.

DDR-SDRAM Kurzform für *Double Data Rate Synchronous Dynamic Random Access Memory*. Direktzugriffsspeichertyp, der in Computern als Arbeitsspeicher vorkommt.

DPWS Kurzform für *Devices Profile for Web Services*. DPWS spezifiziert einen Standard, der es ermöglicht, Webservices auf eingebetteten Systemen mit Ressourcenbeschränkungen einzusetzen.

Endpunktreferenz Eine Referenz auf eine Instanz eines Webservices. Diese besteht im Wesentlichen aus einer Adresse um den anbietenden Netzwerkknoten zu identifizieren (z. B. IP-Adresse) und einer Informationen für die Adressierung des Webservices innerhalb des Netzwerkknotens wie z. B. Ports.

ERP-System Kurzform für *Enterprise Resource Planning*-System. Ein Enterprise Resource Planning-System ist ein Softwaresystem, das die Ressourcenplanung eines Unternehmens unterstützen soll. Der Funktionsumfang deckt dabei nahezu alle Bereiche eines Unternehmens ab und umfasst u. a. die Bereiche Materialwirtschaft, Produktion, Finanz- und Rechnungswesen und Controlling.

FiLeCC Kurzform für *Field Level Computing Cloud*. FiLeCC beschreibt den Teil des Produkts, der von der Projektgruppe 557 vollständig selbst implementiert wird.

Fördermodul Der Begriff Fördermodul bezeichnet eine Stetigförderkomponente eines intralogistischen Materialflusssystems. Stetige Fördermodule sind beispielsweise Geradeausförderer, Kurvenelemente oder Weichen.

Fördertechnik Fördertechnik umfasst alle notwendigen technischen, organisatorischen und personellen Mittel zum Bewegen von Gütern.

GridGain GridGain ist ein auf Java basierendes Framework für Rechencluster, das das verteilte Rechnen/Ausführen von Java-Programmen mittels MapReduce ermöglicht.

IPC Kurzform für Industrie-PC. Ein zum Personal Computer ähnlicher Rechner, der für den Einsatz in industriellen Umgebungen konzipiert und optimiert wurde und sich durch seine Robustheit vor Umwelteinflüssen und elektromagnetischen Störungen auszeichnet.

JVM Kurzform für *Java Virtual Machine*. Die JVM gehört zur Ausführungsumgebung für Java-Programme. In ihr wird der Java-Bytecode ausgeführt.

MapReduce Das MapReduce-Verfahren wird bei der Ausführung verteilter Algorithmen eingesetzt. In einem ersten Schritt, dem sog. *Map*-Schritt, wird die Eingabe zunächst in kleinere Subprobleme zerlegt. Diese Subprobleme werden anschließend an mehrere verteilte Recheneinheiten weitergegeben und von diesen bearbeitet. Nachdem alle Subprobleme bearbeitet wurden, werden diese im sog. *Reduce*-Schritt wieder zusammengefügt und als Lösung des ursprünglichen Problems ausgegeben.

Multicast Multicast ist das Verschicken einer Nachricht über ein Netzwerk an mehrere Empfänger, die zu einer bestimmten Interessengruppe gehören. Zur Bildung solcher Multicast-Gruppen ist der IP-Adressbereich 224.0.0.0 - 239.255.255.255 reserviert.

Node Ein Rechenkapazität anbietendes Gerät innerhalb des Netzwerkes. Dies können beispielsweise IPCs sein.

Pandaboard Das Pandboard ist ein kleiner kostengünstiger und energiesparender Computer, der auf dem Texas Instruments OMAP4430 Prozessor basiert. Er ist mit einer Dual-Core 1 GHz ARM Cortex-A9 MPCore CPU und einer PowerVR SGX540 GPU ausgestattet und verfügt über 1 GB DDR2-SDRAM.

RAM Kurzform für *Random Access Memory*. Direktzugriffsspeicher, der in Computern als Arbeitsspeicher vorkommt.

Rechenressource Eine Rechenkapazität anbietende Ressource. Beispielsweise ein IPC.

Serviceorientierte Architektur Eine Serviceorientierte Architektur (SOA) ist ein abstraktes Konzept einer Software-Architektur. Es beschreibt die Zusammenarbeit zwischen Dienstanbieter und Dienstnutzer mit dem Ziel eine möglichst große Interoperabilität zu erreichen

Speicherressource Eine Speicherkapazität anbietende Ressource. Beispielsweise eine SD-Karte oder ein USB-Stick.

TCP Kurzform für *Transmission Control Protocol*. TCP ist ein Protokoll zum Austausch von Daten zwischen Computern. Unter Verwendung von TCP wird eine Verbindung zwischen den Kommunikationspartnern aufgebaut, über die diese anschließend Pakete austauschen können.

UDP Kurzform für *User Datagram Protocol*. UDP ist ein Protokoll zum Austausch von Daten zwischen Computern. Es handelt sich im Gegensatz zu TCP dabei jedoch um ein verbindungsloses Protokoll.

UUID Kurzform für *Universally Unique Identifier*. Eine eindeutige Identifikationsnummer mit deren Hilfe z. B. Fördermodule identifiziert werden können.

Watchdog Ein Softwareteil, der andere Softwareteile überwacht und das Abstürzen dieser überwachten Softwareteile erkennt und geeignete Gegenmaßnahmen wie beispielsweise einen Neustart einleitet.

Webservice Der Begriff Webservice bezeichnet Software zur Maschine-Maschine-Interaktion über ein Netzwerk. Die Schnittstelle eines Webservice wird in einem von Maschinen verarbeitbaren Format beschrieben.

Lokaler Webservice Als lokaler Webservice wird in diesem Dokument ein Webservice bezeichnet, dessen Ausführung an eine bestimmte Hardwarekomponente gekoppelt ist. Der lokale Webservice kann nur an dieser bestimmten Hardwarekomponente genutzt werden und liegt nur dort vor.

Beweglicher Webservice Als beweglicher Webservice wird in diesem Dokument ein Webservice bezeichnet, dessen Ausführbarkeit nicht an eine bestimmte Hardwarekomponente gebunden ist.

WMS Kurzform für *Warehouse Management System*. Ein Warehouse Management System ist ein Softwaresystem für die Verwaltung von Warenlagern und umfasst die Verwaltung der Positionen der Waren innerhalb des Lagers, der Wareneingänge, der Kommissionierung und der Warenausgänge sowie einiger weiterer Funktionen.

WS-Discovery WS-Discovery ist ein Standard zum dynamischen Auffinden von Webservices. Es basiert auf Multicast und kommuniziert über die IP-Adresse 239.255.255.250 und den Port 3702.

3 Anforderungen

In diesem Abschnitt sind alle Kriterien des Produkts beschrieben, die als Anforderungen für das Produkt vorliegen. Die Kriterien sind je nach Wichtigkeitsgrad in Muss-, Soll- und Kannkriterien unterteilt. Musskriterien unterliegen strengen Richtlinien und müssen nachweislich im Produkt umgesetzt sein. Soll- und Kannkriterien hingegen sind nicht zwingend erforderlich, wobei die Umsetzung der Sollkriterien wiederum wünschenswerter ist als die der Kannkriterien. In Tabelle 3.1, die auf Seite 11 zu finden ist, sind alle zuvor beschriebenen Anforderungen übersichtlich dargestellt.

3.1 Musskriterien

In diesem Kapitel werden die Anforderungen an das Produkt beschrieben, die zwingend umgesetzt werden müssen. Die Anforderungen sind im Folgenden zur Übersicht Bereichen zugeordnet, in denen sie anfallen. Unterschieden werden die Bereiche Fördermodule, Hardwareressourcen und Webservices.

Für die Implementierung ist die Programmiersprache Java vorgesehen. Die in den folgenden Kriterien genannten Webservices müssen in der Programmiersprache Java realisiert sein und dem DPWS-Standard vollständig genügen.

Fördermodule Das Produkt wird für innerbetriebliche Förderanlagen erstellt. Eine Förderanlage besteht aus verschiedenen Fördermodulen, wie beispielsweise Geradeausförderern, Kurvenelementen oder Weichen. Es muss zur Laufzeit möglich sein, Fördermodule zur bestehenden Förderanlage hinzuzufügen oder zu entfernen. Falls technisch machbar, soll dies zur Laufzeit möglich sein. Spätestens bei einem erneuten Start der Anlage muss das neue Modul bzw. das Fehlen eines Moduls erkannt und in der anschließend neu berechneten Topologie berücksichtigt werden.

Auf der Förderanlage sind mehrere Harwareressourcen, wie beispielsweise IPCs, verteilt. An jede Hardwareressource ist mindestens ein Fördermodul angeschlossen, jedes Fördermodul wird dabei von einer Ressouce gesteuert.

Hardwareressourcen Die Förderanlage ist mit verschiedenen Hardwareressourcen ausgestattet, welche die Anlage mit mehr Rechenkapazität ausstatten. Solche Ressourcen müssen zur Laufzeit zur Anlage hinzugefügt oder entfernt werden können, ohne dass Daten verloren gehen.

Nach dem Anschließen der Hardwareressourcen an die Anlage muss eine Basissoftware des Produkts gestartet werden. Anschließend müssen die Ressourcen automatisch erkannt und zum Gesamtsystem der Anlage hinzugefügt werden. Danach stehen sie zur Benutzung zur Verfügung. Zum Entfernen von Hardwareressourcen muss eine Abmeldefunktion realisiert sein, die die Hardwareressourcen vor dem Herunterfahren bei einem globalen Dienst abmeldet. Ein globaler Dienst ist ein Stück Software, das über eine festdefinierte Schnittstelle angesprochen werden kann und bei Verwendung eine bestimmte Funktionalität bereitstellt. Bewegliche Webservices, die auf der entfernten Ressource ausgeführt wurden, müssen anschließend auf einer anderen Ressource gestartet werden. Bei der Verteilung von Webservices muss das Fehlen der Ressource im weiteren Verlauf berücksichtigt werden.

Damit Webservices lastabhängig auf den Rechenressourcen der Anlage verteilt werden können, muss die Möglichkeit bestehen, die Auslastung von Hardwareressourcen zur Laufzeit zu ermitteln. Diese Funktionalität wird intern vom System für die Verteilung von Prozessen genutzt. Es ist dabei von großer Wichtigkeit, dass Änderungen der Auslastung zeitnah erkannt werden, damit eine Ressource direkt nach Beendigung eines Prozesses für weitere Aufgaben zur Verfügung steht. Des Weiteren muss sichergestellt werden, dass Ausfälle von Hardwareressourcen, also Rechenleistung und/oder Speicherkapazität, automatisch erkannt und, falls möglich, behandelt werden.

Das Produkt muss auf einer x86-, sowie einer ARM-Plattform lauffähig sein. Darüberhinausgehend sind alle weiteren Plattformen optional.

Es muss dem Administrator des Anlagensystems eine Administrationsoberfläche zur Verfügung stehen, mit der er sowohl Webservices starten und stoppen als auch Statistiken erstellen kann.

Webservices Alle Aufgaben, die durch das Produkt bearbeitet werden, liegen in Form von Webservices vor. Manche Webservices beziehen sich direkt auf ein Fördermodul und sollten deshalb auch auf der Rechenressource behandelt werden, an die das Födermodul angeschlossen ist. Dies können beispielsweise Steuerfunktionen für Fördermodule sein. Solche Webservices werden im Folgenden als *lokale Webservices* bezeichnet. Andere Webservices, die beispielsweise für die Topologieberechnung zuständig und demnach an keine Rechenressource gebunden sind, können auf jeder Node ausgeführt werden, die zum Ausführungszeitpunkt über genügend freie Kapazitäten verfügt. Der Endpunkt dieser Webservices ist also flexibel. Solche Webservices werden im Folgenden als *bewegliche Webservices* bezeichnet. Sowohl lokale als auch bewegliche Webservices können parallele Algorithmen enthalten, die verteilt auf dem System aus Rechenressourcen ausgeführt werden können. Das Produkt muss deshalb verteiltes Rechnen unterstützen.

Damit Webservices genutzt werden können, müssen sie über ein Netzwerk zur Verfügung gestellt werden, an das alle Komponenten der Anlage angeschlossen sind. Es ist wichtig, dass Webservices zu jeder Zeit verfügbar sind, damit sie genutzt werden können. Dies gilt vorallem für bewegliche Webservices. Fällt ein Teil des Systems, bestehend aus Fördermodulen und Hardwareressourcen, aus, so darf dies nicht zum Verlust von beweglichen Webservices führen. Es muss deshalb sichergestellt sein, dass bewegliche Webservices zu jeder Zeit auf redundant vielen und geeignet in der Topologie verteilt angeordneten Endpunkten angeboten werden. Für lokale Webservices ist dies kein erforderliches Kriterium, da das an die ausfallende Hardwareressource gekoppelte Fördermodul, für das der Webservice bereitgestellt wird, ebenfalls anschließend nicht zur Verfügung steht.

Angebotene Webservices müssen durch eine Suchfunktion auffindbar sein. Durch eine Suche gefundene Webservices werden im Folgenden als *bekannte Webservices* bezeichnet. Die Operationen, die durch bekannte Webservices bereitgestellt werden, müssen nutzbar sein. Des Weiteren muss es die Möglichkeit geben, neue Software in Form von Webservices in das System zu integrieren. Die Integration der neu hinzukommenden Webservices in das bestehende System muss automatisch geschehen. Dies muss auch während des laufenden Betriebes möglich sein. Äquivalent dazu muss es ebenfalls möglich sein, Webservices aus dem Netzwerk zu entfernen. Dabei ist es nicht relevant, ob die Webservices bereits bekannt sind. Alle zu den Webservices zugehörigen Komponenten und Daten müssen dabei automatisch aus dem System gelöscht werden. Dies muss auch während des laufenden Betriebes möglich sein. Bereits bestehende und möglicherweise bekannte Webservices müssen durch Einbringen veränderter Dateien, die die Webservices enthalten, verändert werden können. Auf Hinzufügen, Entfernen und Aktualisieren von Webservices muss das System automatisch reagieren.

Das Produkt muss dem Benutzer eine Datenbank zur Verfügung stellen, die er unabhängig vom Rest des Systems nutzen kann.

Mit dem Produkt muss ein Webservice als Implementierungsbeispiel mitgeliefert werden.

3.2 Sollkriterien

In diesem Abschnitt werden solche Kriterien beschrieben, deren Umsetzung wünschenswert aber nicht zwingend erforderlich ist.

Die Fördermodule der Anlage sind miteinander verbunden und sollen wissen, über welche Funktionen ihre Nachbarmodule verfügen. Es soll daher für jedes Fördermodul ein Webservice zur Verfügung stehen, der die Nachbarschaft erkundet und bekannt macht. Mit Nachbarschaft sind die angrenzenden Module eines Anlagenmoduls gemeint.

Eine weitere Eigenschaft, die implementiert werden soll, ist das Anbieten von Speicherplatz im Sinne des Cloud-Modells *Infrastructure as a Service*.

3.3 Kannkriterien

Eine wünscheswerte Eigenschaft des Produkts ist es, neben den Java basierten Webservices auch solche Webservices zu unterstützen, die in anderen Programmiersprachen umgesetzt sind, sich aber an den DPWS-Standard halten.

Manche Webservices sind wichtiger als andere. Während Webservices, die für die Steuerung der Fördermodule zuständig sind, zu jeder Zeit zuverlässig und schnell verfügbar sein müssen, kann ein Webservice, der eine unabhängige Funktion wie beispielsweise einen Teil eines ERP-Systems verwaltet, auf seine Ausführung warten bis wieder freie Rechenkapazität vorhanden ist. Es können deshalb Prioritäten für Webservices vergeben werden, an denen sich deren Ausführungsreihenfolge orientiert. Dabei sollen steuerungsrelevante Webservices eine höhere Priorität bekommen. Die Prioritäten sollen veränderbar sein.

4 Produkteinsatz

In diesem Abschnitt werden die Anwendungsbereiche und die Zielgruppe der Software beschrieben und die erforderlichen Betriebsbedingungen festgelegt.

Tabelle 3.1: Übersicht über alle Anforderungen an das Produkt.

Musskriterien
Fördermodul hinzufügen
Fördermodul entfernen
Hardwareressource hinzufügen
Hardwareressource entfernen
Hardwareressourcen in die Topologie aufnehmen
Hardwareressourcen abmelden
Auslastung ermitteln
Hardwareausfälle erkennen
Administrationsoberfläche
Verwendung von lokalen Webservices
Verwendung von beweglichen Webservices
Unterstützung von verteiltem Rechnen
Verfügbarkeit beweglicher Webservices sicherstellen
Webservices auffinden
Operationen der Webservices nutzen
Neue Webservices automatisch integrieren
Webservices entfernen
Webservices verändern
Datenbank nutzen
Webservice als Implementierungsbeispiel mitliefern
Sollkriterien
Nachbarschaftserkennung
Kannkriterien
Verwendung von Webservices in anderen Programmierprachen
Priorisierung von Webservices
Speicherplatz zur Verfügung stellen

4.1 Anwendungsbereiche

Das Produkt wird für dezentral gesteuerte Förderanlagen entwickelt, deren steuernde Computer, wie beispielsweise IPCs, über freie Hardwareressourcen verfügen und diese verschiedener Software zur Verfügung stellen können.

4.2 Zielgruppe

Das Produkt richtet sich an Forschungseinrichtungen, die sich mit dezentralen Steuerungen für stetig fördernde Förderanlagen beschäftigen.

Des Weiteren soll das Produkt auf beliebigen Anlagen in der Intralogistik zum Einsatz kommen können. Benutzer sind sowohl die Betreiber der Anlage als auch Administatoren

des Anlagensystems. Sie kommen mit der Software nur beim Starten der Basissoftware auf hinzugefügten Hardwareressourcen und bei der Administration des Produkts in Kontakt. Darum gelten für die Zielgruppe keine weiteren Beschränkungen bezüglich der benötigten Qualifikationen. Entwickler von Webservices für das System müssen sich an die vorgegebenen Schnittstellen halten. Sie sollten deshalb mit den Schnittstellen vertraut sein. Zwecks Einarbeitung wird zusammen mit dem Produkt eine entsprechende API-Dokumentation zur Verfüfung gestellt.

4.3 Betriebsbedingungen

Das Produkt soll für den stetigen Betrieb über einen langen Zeitraum lauffähig sein. Deshalb muss darauf geachtet werden, dass das Produkt nicht mehr verwendete Ressourcen der Ausführungsumgebung, wie z. B. Arbeitsspeicher, wieder freigibt und möglichst schonend mit diesen Ressourcen umgeht.

Für den Betrieb müssen Rechenressourcen zur Verfügung stehen und so miteinander verbunden sein, dass eine Kommunikation über TCP und UDP möglich ist. Das Netzwerk muss multicastfähig sein. Da zum Auffinden der Webservices *WS-Discovery* verwendet wird, das Multicast über die IP-Adresse 239.255.255.250 und den UDP-Port 3702 verwendet, müssen diese Adresse und der Port für WS-Discovery reserviert werden. Das zum verteilten Rechnen verwendete *GridGain* kommuniziert ebenfalls über Multicast über die standardmäßig festgelegte IP-Adresse 228.1.2.4 und den Port 47200. Beide sollten ebenfalls reserviert werden. Weitere Informationen zu GridGain sind in Kapitel 5 zu finden. Die Webservices benutzen zufällig ausgewählte Ports. Es müssen dafür also keine Ports reserviert werden.

Das System arbeitet lokal und soll nach außen abgeschlossen sein. Eine Firewall muss deshalb nicht zum Einsatz kommen. Weitere Sicherheitsmechanismen sind ebenfalls nicht vorgesehen.

5 Produktumgebungen

In diesem Abschnitt werden die benötigten Systemkomponenten und die notwendige Fremdsoftware beschrieben, die für die Benutzung der Software erforderlich sind. Des Weiteren werden die Mindestanforderungen für die Hardwarekomponenten festgelegt.

5.1 Software

Die hier beschriebene Software setzt sich zum einen aus einem bereits bestehenden und zum anderen aus einem selbstimplementierten Teil zusammen. Als Betriebssystem ist ein Linux 2.6 auf den Steuerungsrechnern erforderlich. Das verwendete Betriebssystem

muss USB-Massenspeichergeräte automatisch einbinden können. Es muss des Weiteren *Java Embedded* von Oracle betrieben werden können.

Um verteilt rechnen zu können, wurde aus mehreren bereits existierenden Softwaresystemen *GridGain* ausgewählt. GridGain ist im Vergleich zu entsprechenden Alternativen sehr flexibel und bringt viele Funktionalitäten wie u. a. eine eigene lastabhängige Verteilung von Prozessen mit. Neben GridGain selbst muss die von GridGain unterstützte Ausführungsumgebung, Java SE, ebenfalls vorhanden sein. Damit ein lastabhängiges Verteilen auch unabhängig von GridGain funktioniert, kommt das *Hyperic Sigar Framework* zum Einsatz. Die Hyperic Sigar Bibliotheken müssen deshalb installiert sein.

Neben der bereits aufgezählten Software muss auf den Rechnern, die für die Steuerung der Förderanlage zuständig sind, ebenfalls das vom Lehrstuhl für Förder- und Lagerwesen der TU Dortmund entwickelte *CoDeA* ausgeführt werden, mit dessen Hilfe die Gerätesteuerung realisiert wird.

5.2 Hardware

Das Produkt muss sowohl auf x86- als auch auf ARM-Plattformen ausführbar sein. Die IPCs, auf denen das Produkt u. a. ablaufen soll, haben eine 1,4 GHz CPU und 512 MB RAM. Sie verfügen über eine interne Flashkarte mit 500 MB Speicher und haben zusätzlich eine 2 GB Flashkarte eingebaut.

Im Vorfeld wurde getestet, wieviel Arbeitsspeicher von den erforderlichen Softwarekomponenten (Betriebssystem, JVM, GridGain und FiLeCC) verbraucht wird. Auf dieser Basis wird die Mindestanforderung für die Arbeitsspeichergröße auf 512 MB festgelegt. Die eingesetzte Hardware muss zudem eine CPU verwenden, die über eine vergleichbare Rechenleistung wie die der OMAP4430 ARM®Cortex™-A9 MPCore des *Pandaboards* verfügt.

Aufgrund von Tests bezüglich des verbrauchten persistenten Speicherplatz sind 1 GB persistenter Speicher mindestens erforderlich, eine Speichergröße von 2 GB wird zur Benutzung aber empfohlen.

6 Produktfunktionen

In diesem Kapitel werden die Produktfunktionen beschrieben. Dabei werden zunächst im Abschnitt 6.1 die Anwendungsfälle des Produkts erläutert. Im Abschnitt 6.2 werden weitere Produktfunktionen beschrieben, die keinem Anwendungsfall direkt zugeordnet werden können. Den Abschluss dieses Kapitels bildet Abschnitt 6.3, der den grundsätzlichen Aufbau dieses Produkts beschreibt.

6.1 Anwendungsfälle

Die Anwendungsfälle sind in Abbildung 6.1 grafisch dargestellt und werden im Folgenden näher erläutert.

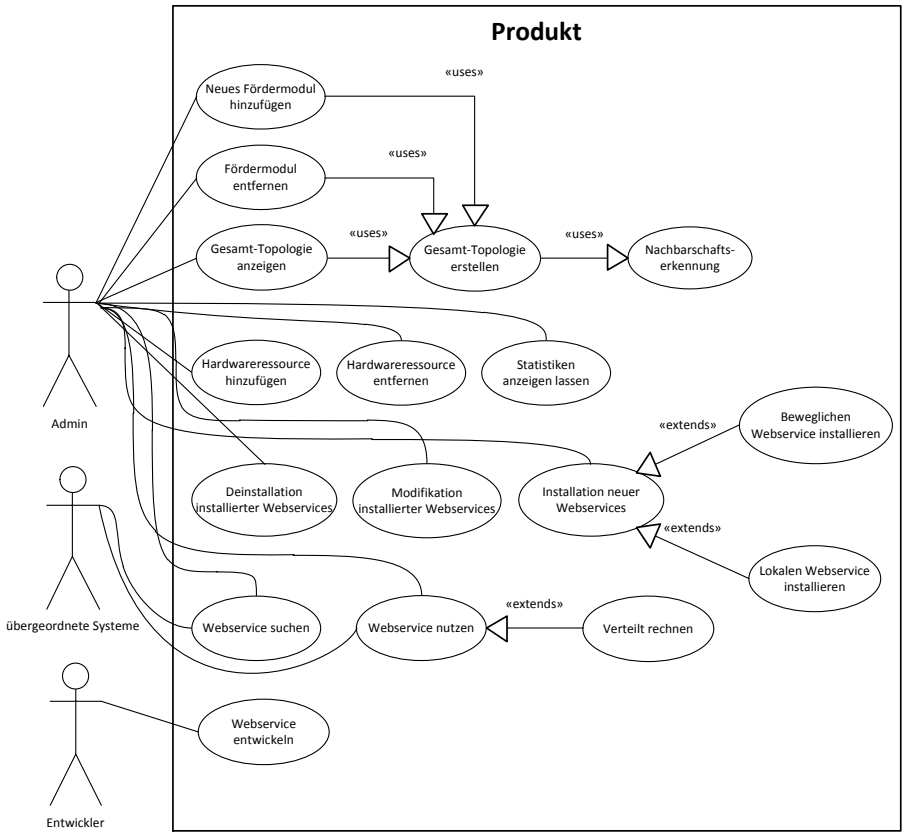


Abbildung 6.1: Anwendungsfalldiagramm

P101: Neues Fördermodul hinzufügen
Der Betreiber der Anlage erweitert die bestehende Anlage um ein neues Fördermodul, wie z. B. einen Geradeausförderer. Nachdem die von diesem Produkt nicht abgedeckte technische Installation und Einrichtung abgeschlossen und insbesondere alle zur Kommunikation mit dem neuen Fördermodul nötigen Verbindungen hergestellt wurden, muss das Produkt das neue Fördermodul erkennen und die bestehende Anlagentopologie entsprechend aktualisieren. Für die Berechnung der Anlagentopologie wird auf die Nachbarschaftserkennungsfunktionalität zurückgegriffen, die im Anwendungsfall *P103* näher erläutert wird. Falls das neue Fördermodul Software in Form von Webservices mitliefert, so kann diese, wie im Anwendungsfall *P301* beschrieben, installiert werden.

P102: Fördermodul entfernen
Der Betreiber der Anlage entfernt ein Fördermodul aus der bestehenden Anlage. Hierzu bietet ein zum Produkt gehöriger Client mit einer grafischen Benutzerschnittstelle eine entsprechende Funktionalität an. Der Betreiber der Anlage wählt im Client das zu entfernende Fördermodul aus. Das Produkt sorgt anschließend dafür, dass die bestehende Anlagentopologie entsprechend aktualisiert wird und die zu diesem Modul gehörenden Webservices deinstalliert werden. Die zu den deinstallierten Webservices gehörenden Daten, die sich in einem verteilten Datenspeicher auf verschiedenen Nodes im Netzwerk befinden, werden vom Produkt gelöscht. Die von diesen Webservices angebotenen Funktionalitäten stehen anschließend nicht mehr zur Verfügung.

P103: Nachbarschaftserkennung
Für die Berechnung einer Gesamttopologie der Anlage, die Informationen über alle angeschlossenen Module enthält, ist eine Nachbarschaftserkennung erforderlich. Bei Umsetzung des entsprechenden Sollkriteriums zur automatischen Ermittlung der benachbarten Module verfügt jedes Modul der Anlage über einen eigenen Mikrocontroller, der über verschiedene Anschlüsse mit den jeweils benachbarten Modulen verbunden ist. Dieser Mikrocontroller ist nicht Teil dieses Produkts und wird als vorhanden vorausgesetzt. Das Produkt kann über Ethernet mit diesen Mikrocontrollern kommunizieren. Hierfür ist es notwendig, ein Protokoll zu spezifizieren, das den genauen Ablauf der Kommunikation definiert. Über dieses Ethernet-Protokoll ist es dem Produkt möglich, die Nachbarschaftsinformationen eines Moduls von dem an diesem Modul angeschlossenen Mikrocontroller zu erhalten. Diese Nachbarschaftsinformationen bestehen pro benachbartem Modul u. a. aus einer eindeutigen Kennung des Moduls (UUID) und dem Modulausgang, an dem das Modul angeschlossen ist. Falls dieses Sollkriterium im Produkt nicht umgesetzt wird, erfolgt die Nachbarschaftserkennung des Produkts durch Auslesen von Konfigurationsdateien, die die Kennung und den entsprechenden Modulausgang der benachbarten Module enthalten. Diese Konfigurationsdateien müssen vom Administrator für jedes Modul manuell erstellt und bei Änderungen an der Anlagentopologie aktuell gehalten werden. Die Nachbarschaftserkennung wird gestartet, wenn ein neues Modul der Anlage hinzugefügt oder entfernt wurde und somit die Anlagentopologie aktualisiert werden muss.

P104: Gesamttopologie erstellen
Dieser Anwendungsfall deckt das Erstellen einer Gesamttopologie ab. Mit Hilfe der Nachbarschaftserkennung wird die Topologie der Anlage erkundet und eine Gesamttopologie erstellt. Dieser Anwendungsfall kann sowohl dazu verwendet werden, um initial die Anlagentopologie zu bestimmen als auch dazu, die Topologie nach dem Hinzufügen oder Entfernen von Anlagenmodulen zu aktualisieren. Diese Funktionalität wird als beweglicher Webservice umgesetzt, sodass die Erstellung der Gesamttopologie von allen im Netzwerk vorhandenen Nodes bei Bedarf durchgeführt werden kann.

P105: Gesamttopologie anzeigen
Der Administrator lässt sich die Topologie der Förderanlage anzeigen. Hierfür bietet der zum Produkt gehörige Client mit einer grafischen Benutzerschnittstelle eine entsprechende Funktionalität an, mit der eine Übersicht der Gesamttopologie der Anlage aufgerufen werden kann.

P201: Neue Hardwareressource hinzufügen
Der Administrator der Förderanlage integriert eine neue Hardwareressource in das System. Damit die neue Hardwareressource vom Produkt erkannt werden kann, ist es bei Rechenressourcen notwendig, dass der Administrator zunächst das Produkt auf der neuen Rechenressource installiert und diese an das Netzwerk anschließt. Das Produkt bestimmt im Netzwerk eine Node zum so genannten *Leader*. Dies geschieht durch ein Verfahren, das eine Node durch eine Broadcast-Nachricht initiiert. Hierbei generiert jede Node eine ID, die im einfachsten Fall zufällig ist, aber auch auf aktuellen Auslastungsdaten beruhen kann. Diejenige Node mit der höchsten ID gewinnt die Leaderwahl und fungiert anschließend im Netzwerk als Leader. Das Verfahren ist dabei so ausgelegt, dass es parallele Leaderwahlen erkennt und zu einer gemeinsamen Wahl kombiniert. Die neue Rechenressource erkennt bei Eintritt in das Netzwerk, dass bereits eine Node als Leader fungiert und meldet sich als neue Node bei der Leader-Node an. Diese ist für die vollständige Integration der neuen Node in das Netzwerk zuständig und sorgt dafür, dass die von der neuen Node angebotenen beweglichen Webservices im gesamten Netzwerk verteilt und zur Sicherstellung der Verfügbarkeit mehrfach angeboten werden. Falls zum Zeitpunkt des Hinzufügens der neuen Rechenressource kein Leader im Netzwerk vorhanden ist, initiiert die neue Rechenressource eine Leaderwahl. Speicherressourcen werden vom Produkt dafür verwendet um Speicherplatz für den verteilten Datenspeicher bereitzustellen. Dieser beinhaltet die jar-Dateien und Konfigurationsdaten der installierten Webservices sowie bei der Ausführung des Produkts anfallende Daten, wie z. B. Auslastungsdaten der einzelnen Nodes. Das Hinzufügen von neuen Speicherressourcen wird vom Produkt nicht direkt unterstützt. Da das Produkt als Speicherort jedoch nur einen in der Nodekonfiguration angegebenen Order verwendet, besteht für den Administrator die Möglichkeit, an diese Stelle beliebige Speichermedien zu mounten und so Einfluss auf die verfügbare Speicherkapazität des Produkts zu nehmen. Für Änderungen an der Konfiguration des Speicherordners darf das Produkt auf der betroffenen Node nicht ausgeführt werden, sondern muss vorher beendet und anschließend neugestartet werden. Bei Umsetzung des entsprechenden Kannkriteriums kann dieser Speicherplatz auch nach außen hin für den Administrator zur Verfügung gestellt werden.

P202: Hardwareressource entfernen
Der Administrator entfernt eine Hardwareressource aus der Anlage. Falls es sich dabei um eine Rechenressource handelt, kann er im zum Produkt gehörenden Client, der eine grafische Oberfläche anbietet, die zu entfernende Rechenressource auswählen. Der Client ruft anschließend ein zum Produkt gehörenden Softwareteil auf, der im Netzwerk nur auf einer Node läuft und die Verfügbarkeit von beweg-

chen Webservices sicherstellt. Dieser Softwareteil sorgt dafür, dass alle beweglichen Webservices der zu entfernenden Node nach dem Entfernen weiterhin im Netzwerk auf anderen Nodes verfügbar sind und somit keine beweglichen Webservices beim Entfernen der Rechenressource versehentlich mitentfernt werden. Weiterhin benötigte Dateien des verteilten Datenspeichers, die sich auf der zu entfernenden Node befinden, werden vor dem Entfernen ebenfalls auf andere Nodes ausgelagert um deren Verfügbarkeit zu gewährleisten. Nachdem dies abgeschlossen wurde, wird die Software dieses Produkts auf der zu entfernenden Node beendet. Das Entfernen von Speicherressourcen wird nicht direkt vom Produkt unterstützt. Allerdings besteht, ähnlich zum Anwendungsfall *P201*, für den Administrator die Möglichkeit den Speicherort des Produkts in der Nodekonfiguration anzugeben und somit Einfluss auf die verfügbare Speicherkapazität des Produkts zu nehmen.

P203: Statistiken anzeigen lassen
Der Administrator sieht die Statistiken zur Auslastung der im System vorhandenen Hardwareressourcen ein. Hierfür bietet der zu diesem Produkt gehörende grafische Client eine Oberfläche an, die die Auslastung der einzelnen Hardwareressourcen darstellt. Auf jeder Node wird zu diesem Zweck ein Softwareteil des Produkts ausgeführt, über den die Auslastung der Node (CPU-Auslastung, verfügbarer/belegter Arbeitsspeicher und freie/belegte Speicherkapazität) ermittelt werden kann.

P301: Installation neuer Webservices
Der Administrator bringt einen neuen Webservice in das Netzwerk ein. Der zu installierende Webservice muss dabei mit dem Produkt kompatibel sein. Produktkompatible Webservices müssen die DPWS-Spezifikation erfüllen und als jar-Datei vorliegen, die zusätzlich zu den eigentlichen Webservices noch eine Konfigurationsdatei enthält. Die Konfigurtionsdatei muss die eindeutigen Namen und Versionen der in der jar-Datei definierten Webservices und Devices sowie dessen Zuordnung zueinander enthalten. Dies ist erforderlich, da eine jar-Datei mehrere Webservices und Devices enthalten kann und die Produktteile, die für die Installation, Starten und Loadbalancing der Webservices zuständig sind, eine Zuordnung zur sie beinhaltenden jar-Datei herstellen können müssen. Weiterhin muss der Typ der Webservices (lokal oder beweglich) spezifiziert sein und im Fall der Umsetzung des Kannkriteriums bzgl. der Vergabe von Prioritäten für Webservices, müssen diese ebenfalls in der Konfigurationsdatei angegeben werden. Wurde die Konfigurationsdatei vom Entwickler des Webservices nicht mitgeliefert, so muss diese vom Administrator selbst erstellt werden. Da alle Webservices innerhalb einer jar-Datei von dem zum Produkt gehörenden Load Balancer als eine Einheit aufgefasst werden, wird empfohlen, dass eine jar-Datei aus genau einem Device besteht um eine feine Granularität des Load Balancings zu ermöglichen. Für die Installation produktkompatibler Webservices bietet das Produkt drei verschiedene Verfahren an. Eine Möglichkeit ist es, die Installation über den zum Produkt gehörenden Client durchzuführen. In diesem kann die zu installierende jar-Datei ausgewählt und installiert werden. Der Client leitet die jar-Datei zur Deploymentschnittstelle einer beliebigen Node im Netzwerk weiter. Diese umfasst alle zur Installation und zum

Start notwendigen Funktionen. Hierunter fallen Auswahl der Nodes, auf denen der Webservice angeboten werden soll, das Verteilen der jar-Datei im verteilten Datenspeicher für die Sicherstellung der Verfügbarkeit und das Starten und Stoppen von Webservices. Die zweite Möglichkeit zur Installation produktkompatibler Webservices besteht darin einen USB-Stick an eine Node anzustecken. USB-Sticks, die sich als Massenspeicher zu erkennen geben, werden von der auf jeder Node gestarteten Deploymentschnittstelle des Produkts automatisch nach produktkompatiblen Webservices durchsucht und die gefundenen Webservices werden anschließend automatisch installiert. Als dritte Möglichkeit zur Installation von produktkompatiblen Webservices besteht die Möglichkeit die jar-Datei direkt in einen von der Deploymentschnittstelle auf Änderungen überwachten Ordner zu kopieren. Die Deploymentschnittstelle erkennt die neue jar-Datei und startet das Deployment. Je nach Typ des neuen Webservices (beweglich oder lokal) variiert jeweils der Ablauf des Deployments. Die folgenden beiden Anwendungsfälle *P301a* und *P301b* beschreiben für diese beiden Typen den konkreten Ablauf des Deployments nachdem der neue Webservice auf eine der drei beschriebenen Wege ins System eingebracht wurde. Für den Fall, dass der zu installierende Webservice nicht produktkompatibel ist, wird in allen drei Installationsvarianten eine Fehlermeldung generiert und in eine Log-Datei geschrieben, die über den im Produkt enthaltenen grafischen Client eingesehen werden kann.

P301a: Lokalen Webservice installieren
Dieser Anwendungsfall erweitert den Anwendungsfall *P301*. Bei einem lokalen Webservice muss innerhalb seiner Konfigurationsdatei spezifiziert sein, dass es sich um einen lokalen Webservice handelt. Erfolgt die Installation über den zum Produkt gehörenden Client, muss während der Installation diejenige Node ausgewählt werden, die den neuen Webservice anbieten soll, sodass die jar-Datei des Webservices an die Deploymentschnittstelle dieser Node geschickt werden kann. Bei den anderen Installationsvarianten wird der neue Webservice von derjenigen Node angeobten, an die der USB-Stick angesteckt wurde bzw. in dessen Ordner die jar-Datei kopiert wurde. Das Deployment der Node, die den neuen Webservice anbieten soll, übernimmt das Starten des neuen Webservices. Dieser wird ausschließlich von dieser Node im Netzwerk angeboten. Typische Vertreter lokaler Webservices sind solche, die die direkte Steuerung der Aktoren eines Moduls übernehmen und deshalb nur auf jener Node lauffähig sind, die mit diesen Aktoren direkt verbunden ist. Lokale Webservices können für parallele Berechnungen GridGain verwenden. Das Produkt liefert GridGain mit und startet auf allen Nodes eine GridGain-Node, sodass parallele Berechnungen unter Einhaltung der Schnittstelle von GridGain potentiell über alle im Netzwerk vorhandenen Nodes durchgeführt werden können.

P301b: Beweglichen Webservice installieren
Dieser Anwendungsfall erweitert den Anwendungsfall *P301*. Bei beweglichen Webservices muss innerhalb der Konfigurationsdatei vermerkt sein, dass es sich um einen beweglichen Webservice handelt. Das Deployment ermittelt auf Grundlage der aktuellen Auslastungen der einzelnen Nodes diejenigen Nodes auf denen der neue

Webservice angeboten werden soll. Um die Verfügbarkeit des neuen Webservices zu gewährleisten, wird dieser von mehreren Nodes mit ausreichend freien Ressourcen parallel angeboten. Die Auslastung der einzelnen Nodes kann das Deployment aus dem verteilten Datenspeicher entnehmen, in dem eine Datei gespeichert wird, die die aktuellen Auslastungen der Nodes enthält und in regelmäßigen Abständen durch die zum Produkt gehörende und auf jeder Node gestarteten Ressourcenüberwachung aktualisiert wird. Nachdem die Nodes ausgewählt wurden, die den neuen Webservice anbieten sollen, werden, wie bereits bei der Installation eines lokalen Webservices erläutert, die Daten des neuen Webservices (jar-Datei inklusive Konfigurationsdatei) zu den ausgewählten Nodes übertragen. Das Deployment auf den Ziel-Nodes sorgt anschließend für das Starten des neuen Webservices. Auch bewegliche Webservices können für parallele Berechnungen auf GridGain zurückgreifen und die Berechnungen über das Netzwerk parallelisieren.

P302: Modifikation installierter Webservices
Ein bereits installierter Webservice wird auf eine neue Version aktualisiert. Als Konvention zur Sicherstellung von Abwärtskompatibilität ist dabei zwingend zu beachten, dass die Beibehaltung des selben Namens für einen Webservice nur dann erlaubt ist, wenn der Webservice rückwärtskompatibel ist und sich seine Ein- und Ausgabeparameter in der neuen Version nicht verändert haben. Andernfalls ist für den aktualisierten Webservice ein neuer Namen zu vergeben. Bei einem Update eines Webservices ist es weiterhin zwingend erforderlich, dass die jar-Datei des aktualisierten Webservice den selben Dateinamen besitzt wie der ursprünglich installierte Webservice. Der Update-Prozess erfolgt nun ähnlich dem Installieren eines neuen Webservices. Das Deployment erkennt, wenn die aktualisierte jar-Datei in den überwachten Ordner eingefügt wird. Stellt das Deployment fest, dass dieser Webservice bereits im Netzwerk existiert, wird die aktualisierte Datei an alle, den Webservice anbietenden, Nodes übertragen und dort der Webservice in der aktualisierten Version neu gestartet. Weiterhin werden alle Kopien der ursprünglichen Version im verteilten Dateispeicher aktualisiert. Nach erfolgtem Update führt ein Aufruf des aktualisierten Webservices die neue Version aus.

P303: Deinstallation installierter Webservices
Der Administrator entfernt einen installierten Webservice. Hierfür bietet der zum Produkt gehörende Client eine Funktion an, mit deren Hilfe der Administrator den zu entfernenden Webservice auswählen kann. Der Client ermittelt auf welchen Nodes dieser Webservice läuft und delegiert das Stoppen des Webservices an die Deployment-Schnittstellen dieser Nodes. Weiterhin veranlasst der Client das Löschen aller zu diesem Webservice gehörenden Daten im verteilten Datenspeicher. Anschließend ist der gelöschte Webservice weder im Netzwerk bekannt noch im verteilten Datenspeicher vorhanden.

P304: Webservice suchen
Es gibt die Möglichkeit, Webservices mit bestimmten Funktionalitäten zu suchen. Die Rückgabe besteht bei lokalen Webservices aus Netzwerkadressen, wie z. B. einer

Endpunktreferenz oder IP-Adresse derjenigen Node, die den gesuchten Webservice anbietet. Bei beweglichen Webservices erfolgt eine Interaktion mit dem zum Produkt gehörenden *Load Balancer*. Es wird versucht das Load Balancing transparent umzusetzen, d. h. ohne Anforderungen an Client und Webservice. Hierfür soll ein Discovery Proxy implementiert werden, der als Suchergebnis die Endpunktreferenz des zum gesuchten Webservice gehörenden Load Balancers zurückgibt. Ruft der Client anschließend den Webservice auf, leitet der Load Balancer diesen Aufruf lastabhängig an eine wenig ausgelastete Node weiter. Aufgrund des clientseitigen Cachings von Endpunktreferenzen kann diese Umsetzung mittels Discovery Proxy nicht garantiert werden. Das Produkt ermöglicht in diesem Fall eine von zwei Alternativen. Entweder ist ein Load Balancing nur bei deaktiviertem Caching der Endpunktreferenzen auf Clientseite möglich oder die Clients sorgen eigenständig für eine lastabhängige Verteilung ihrer Aufrufe, indem sie die Aufrufe gleichmäßig auf alle den Webservice anbietende Nodes verteilen.

P305: Webservice nutzen
Sind Endpunktreferenzen von Webservices durch eine Suche gefunden worden, kann der gesuchte Webservice darüber aufgerufen und genutzt werden. Dies gilt sowohl für lokale als auch für bewegliche Webservices.

P305a: Verteilt rechnen
Erweitert den Anwendungsfall „Webservice nutzen“ bezüglich der Möglichkeit einen Webservice zu nutzen, der eine verteilte Berechnung über GridGain auf mehreren Rechenressourcen parallel durchführt. Der Entwickler des Webservices muss dabei den Webservice bereits bei der Entwicklung an die GridGain-Schnittstelle angepasst haben, sodass dieser, ohne Einwirken des Produkts, direkt auf die auf den Nodes gestarteten GridGain-Nodes zugreifen kann.

P401: Webservice entwickeln
Der Entwickler entwickelt einen Webservice, der kompatibel zu diesem Produkt ist. Sofern das Kannkriterium bzgl. der Ausführbarkeit von Webservices, die in anderen Programmiersprachen als Java implementiert wurden, von diesem Produkt nicht umgesetzt wurde, ist es zwingend erforderlich, dass der Entwickler für die Entwicklung eines Webservices die Programmiersprache Java verwendet.

6.2 Weitere Produktfunktionen

P501: Lastabhängige Verteilung von Webservices
Bewegliche Webservices werden vom Produkt so auf die einzelnen Nodes verteilt, sodass sich eine möglichst gleichmäßige Auslastung der einzelnen Nodes ergibt und keine Node überlastet ist. Für die Umsetzung beinhaltet das Produkt einen Softwareteil, der *UNO* genannt wird und auf genau einer Node ausgeführt wird. Stellt eine Node fest, dass sie überlastet ist, kann diese die UNO benachrichtigen. Die UNO sorgt anschließend dafür, dass bewegliche Webservices, die die überlastete Node

anbietet, auf andere weniger ausgelastete Nodes übertragen werden und somit die Node entlastet wird. Um Situationen, in denen einzelne Nodes überlastet sind, so weit es möglich ist zu vermeiden, wird vom Produkt für bewegliche Webservices weiterhin ein *Load Balancing* durchgeführt. Dieses wurde bereits im Anwendungsfall *P304* erwähnt und hat die Aufgabe, einzelne Webserviceaufrufe auf aktuell wenig ausgelastete Nodes zu verteilen und verhindert somit, dass aktuell stark ausgelasteten Nodes weitere Last zugewiesen wird. Da lokale Webservices an eine bestimmte Node gebunden sind, ist diese lastabhängige Verteilung lediglich für bewegliche Webservices möglich.

P502: Sicherstellung der Verfügbarkeit von Webservices
Für die Sicherstellung der Verfügbarkeit von beweglichen Webservices bietet das Produkt einige Mechanismen an. Für die lokale Sicherstellung der Verfügbarkeit auf einer Node wird auf jeder Node ein lokaler Watchdog ausgeführt. Dieser überwacht sowohl die Prozesse des Produkts, als auch die installierten Webservices, die auf der Node angeboten werden. Erkennt der lokale Watchdog, dass ein Prozess oder Webservice nicht mehr reagiert, veranlasst dieser den Neustart des betreffenden Prozesses bzw. Webservices. Zur globalen Sicherstellung der Verfügbarkeit im gesamten Netzwerk wird auf genau einer Node zusätzlich zum lokalen Watchdog ein globaler Watchdog ausgeführt. Dieser sorgt dafür, dass auf allen im Netzwerk vorhandenen Nodes stets ein lokaler Watchdog vorhanden ist. Fällt ein lokaler Watchdog aus, so initiiert der globale Watchdog den Neustart des lokalen Watchdogs auf der entsprechenden Node. Weiterhin kann durch den globalen Watchdog der Ausfall einer oder mehrerer Nodes erkannt werden. In diesem Fall benachrichtigt der globale Watchdog die UNO, welche dafür sorgt, dass die beweglichen Webservices auf den ausgefallenen Nodes von anderen noch aktiven Nodes weiterhin angeboten werden.

P503: Priorisierung von Webservices
Bei Umsetzung des entsprechenden Kannkriteriums bzgl. der Priorisierbarkeit von Webservices, bietet das Produkt die Möglichkeit den Webservices Prioritäten zuzuweisen. Die Zuweisung der Prioritäten erfolgt statisch durch einen Eintrag in den Konfigurationsdateien der Webservices.

6.3 Aufbau

In Abbildung 6.2 ist der Aufbau des Systems dargestellt. Die Hardwareressourcen sind in einem Netzwerk miteinander verbunden und können über eine DPWS-Schnittstelle miteinander kommunizieren. Auf den Hardwareressourcen werden Webservices angeboten, die zum Teil lokal (grüne Umrandung) und zum Teil beweglich (rote Umrandung) sind. Zusätzlich blau umrandete Webservices sind solche, die eine verteilte Berechnung über GridGain durchführen. Hierzu greifen sie auf die *GridGainNode* zu, die wiederum die Berechnungen auf mehrere Nodes im Netzwerk verteilt. Diese Webservices können von verschiedenen Applikationen genutzt werden.

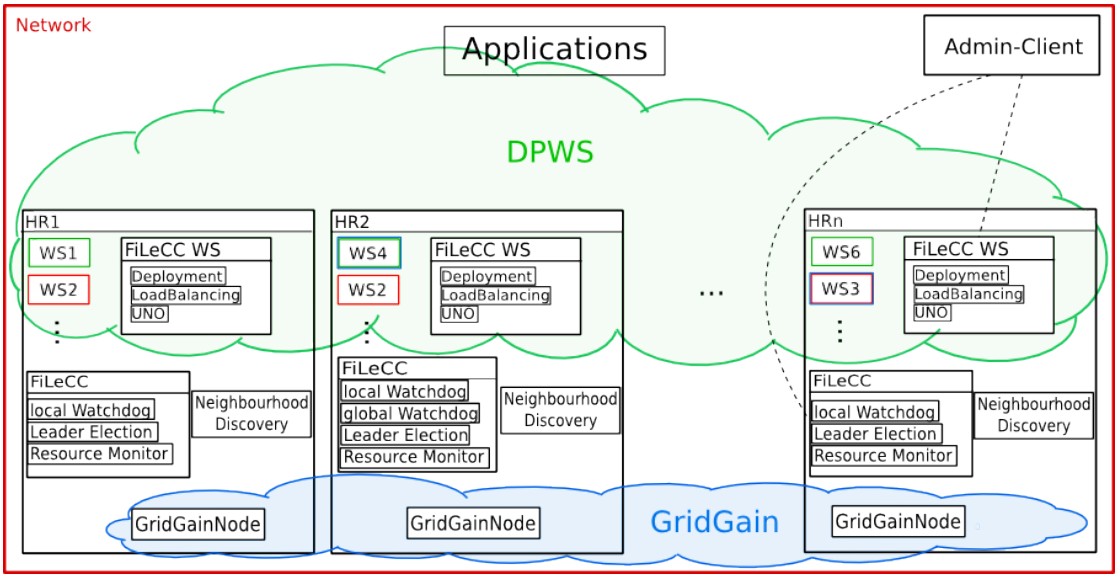


Abbildung 6.2: Aufbau des Systems

Neben diesen nicht zum Produkt gehörenden Webservices werden von jeder Hardwareressource weiterhin auch zum Produkt gehörende Webservices angeboten, die unter der Bezeichnung *FiLeCC WS* zusammengefasst wurden. Im Einzelnen sind dies Webservices der folgenden Komponenten:

Deployment Dieser Webservice ist u. a. dafür zuständig Kanäle zu Massenspeichergeräten, wie USB-Sticks oder externen Festplatten, zu überwachen und bei Bedarf neue Software in Form von Webservices zu installieren.

LoadBalancing Der LoadBalancing-Webservice übernimmt die lastabhängige Verteilung der beweglichen Webservices.

UNO Die UNO bietet einen Webservice an, den überlastete Nodes nutzen können um der UNO ihre Überlastung zu signalisieren. Diese leitet anschließend die in der Produktfunktion *P501* beschriebenen Gegenmaßnahmen ein.

Die weiteren Komponenten des Produkts, die keine Webservices sind, wurden unter der Bezeichnung *FiLeCC* zusammengefasst und umfassen folgende Komponenten:

Local Watchdog Der lokale Watchdog überwacht die Prozesse und Webservices auf Node-Ebene und startet diese bei einem Absturz neu.

Global Watchdog Der globale Watchdog stellt sicher, dass auf jeder Node ein lokaler Watchdog vorhanden ist. Dieser wird auf genau einer Node im Netzwerk ausgeführt.

Leader Election Diese Komponente übernimmt die Aufgabe der Leaderwahl.

Resource Monitor Der Resource Monitor ermittelt die Auslastungsdaten der Hardwareressource und stellt diese den anderen Komponenten, insbesondere der UNO und dem LoadBalancing, zur Verfügung.

Die *Neighbourhood Discovery* ist eine Komponente, mit deren Hilfe die benachbarten Module eines Anlagenteils ermittelt werden können (vgl. Anwendungsfall *P103*). Da die Hardware in Form eines Mikrocontrollers für die *Neighbourhood Discovery* vom Lehrstuhl für Förder- und Lagerwesen der TU Dortmund entwickelt wird und lediglich die Kommunikation über Ethernet mit diesem Mikrocontroller von dem Produkt übernommen wird, wurde die *Neighbourhood Discovery* in Abbildung 6.2 separat eingezeichnet.

Weiterhin ist in Abbildung 6.2 der Administrations-Client dargestellt. Dieser stellt eine grafische Benutzeroberfläche bereit, mit der der Administrator das Produkt administrieren kann. Die Kommunikation des Clients mit den einzelnen Komponenten des Produkts erfolgt über Ethernet.

7 Globale Testszenarien und Testfälle

Die Testszenarien und Testfälle für die in diesem Projekt entwickelte Software können auf zwei verschiedenen Testumgebungen durchgeführt werden. Diese werden im folgenden Unterabschnitt 7.1 kurz beschrieben. Daran anschließend erfolgt im Unterabschnitt 7.2 eine detaillierte Beschreibung einzelner Testszenarien und Testfälle für die Produktfunktionen. Die Durchführung der Tests ist auf beiden vorgestellten Testumgebungen möglich. Rechenressourcen, wie z. B. IPCs, werden dabei im Folgenden auch als Nodes bezeichnet.

7.1 Testumgebungen

Eine Testumgebung ist die vom Lehrstuhl für Förder- und Lagerwesen betriebene Versuchsanlage. Diese ist für den Transport von Mehrzweckbehältern ausgelegt und besteht aus zwei miteinander verbundenen Ebenen, die eine Förderstrecke mit einer Länge von 120 Metern bilden. Einzelne Anlagenteile wurden zu Segmenten zusammengefasst, für deren Steuerung jeweils ein IPC verantwortlich ist. Insgesamt gibt es sieben IPCs, die auf der gesamten Anlage verteilt sind und deren Steuerung übernehmen. Zum Testen des Produkts muss dieses zunächst auf den IPCs installiert werden. Anschließend können die im folgenden Abschnitt beschriebenen Testszenarien und Testfälle durchgeführt werden.

Als zweite Testumgebung kommt die im Rahmen einer Projektarbeit am Lehrstuhl für Förder- und Lagerwesen entworfene Miniaturfördertechnik zum Einsatz. Diese besteht aus kleinen Modulen mit einer Grundfläche von $300 \times 300\,\text{mm}$, die zu einer Miniaturförderanlage kombiniert werden können. Derzeit verfügbare Module sind ein Geradeausförderer und ein Drehtisch. Die Steuerung dieser Module erfolgt auch hierbei de-

zentral über an den Modulen montierte Recheneinheiten. Auch bei dieser Testumgebung ist eine Installation des Produkts Vorraussetzung für die Durchführung der Tests.

7.2 Testszenarien und Testfälle

In den folgenden Testfällen werden die Webservices WS-LP, WS-BP, WS-LMM und WS-BMM verwendet.

Der Webservice WS-LP ist ein lokaler Webservice, der Primzahlen bis zu einer hinreichend großen Zahl x berechnet, so dass eine sehr hohe Auslastung der einzelnen im System vorhandenen Nodes während der Prüfung der Ergebnisse einzelner Testfälle gewährleistet ist. Als eingesetztes Verfahren bietet sich das extrem rechenaufwändige Probedivisionsverfahren an.

Der Webservice WS-BP ist ebenfalls ein Webservice zur Primzahlberechnung. Er unterscheidet sich vom Webservice WS-LP dadurch, dass er ein beweglicher Webservice ist. Die Implementierung dieser beiden Webservices ist somit identisch, einzig die Konfiguration des Webservices benötigt eine entsprechende Anpassung.

Die Webservices WS-LMM und WS-BMM sind Webservices zur parallelen Berechnung einer Matrix-Multiplikation mithilfe von GridGain. Der Webservice WS-LMM wird als lokaler Webservice dabei nur von einer vorher konfigurierten Node angeboten, nutzt jedoch für die Berechnung der Matrix-Multiplikation auch Rechenkapazitäten anderer Nodes. Im Unterschied dazu wird der Webservice WS-BMM als beweglicher Webservice von einer beliebigen Node angeboten.

Der Webservice WS-T ist ein Webservice zur Topologieberechnung der Förderanlage. Er liefert einen ungerichteten Graphen, in dem alle Nachbarschaften von Fördermodulen erkennbar sind.

7.2.1 T101: Neues Fördermodul hinzufügen

Dieser Test dient der Überprüfung, ob die im Anwendungsfall *P101* beschriebene Funktionalität zum Hinzufügen neuer Fördermodule im Produkt korrekt umgesetzt wurde. Weiterhin lässt sich mit diesem Test feststellen, ob die Topologieerkennung (Anwendungsfall *P104*) und die Nachbarschaftserkennung (Anwendungsfall *P103*) des Produkts korrekt umgesetzt wurden. Sofern der resultierende Topologiegraph, der nach dem Hinzufügen des neuen Fördermoduls ausgegeben wird, mit dem vorgegebenen Topologiegraphen übereinstimmt und die im resultierenden Topologiegraphen enthaltene UUID identisch zu der im Vorfeld bekannten UUID ist, hat das Produkt das neue Fördermodul korrekt erkannt und sowohl Topologie- als auch Nachbarschaftserkennung funktionieren korrekt.

Vorbedingung:

- Auf allen bestehenden Nodes ist die Basissoftware des Produkts installiert und gestartet
- Ein Topologiegraph, der das neue Fördermodul enthält, ist vorgegeben
- Die UUID des neuen Fördermoduls ist bekannt

Durchzuführende Schritte:

- Neues Fördermodul in die Anlage bringen und entsprechend anschließen
- WS-T ausführen

Erwartete Ausgabe:

- Ungerichteter Graph, der das neue Fördermodul enthält

Erwartete Nachbedingung:

- Neues Fördermodul ist in die Förderstrecke integriert und kann verwendet werden
- Die UUID des neuen Fördermoduls wurde durch die Nachbarschaftserkennung korrekt ermittelt

Prüfanweisungen:

- Ergebnis des WS-T mit dem vorgegebenen Graphen vergleichen
- UUID vergleichen: Vergleich der im Vorfeld bekannten UUID mit der im Graphen angezeigten UUID

7.2.2 T102: Fördermodul entfernen

Ziel dieses Tests ist die Überprüfung der korrekten Umsetzung des Anwendungsfalls *P102*. Auch mit diesem Test lässt sich zusätzlich die korrekte Funktionsweise der Topologie- und Nachbarschaftserkennung testen, die in den Anwendungsfällen *P104* und *P103* beschrieben wurden. Nachdem ein Fördermodul entfernt wurde, darf dieses anschließend nicht mehr im Topologiegraphen vorhanden sein. Dies überprüft der vorliegende Test.

Vorbedingung:

- Auf allen bestehenden Nodes ist die Basissoftware des Produkts installiert und gestartet
- Ein Topologiegraph besteht, der den Aufbau der Förderanlage beschreibt

Durchzuführende Schritte:

- Fördermodul aus der Anlage mit Hilfe des zum Produkt gehörenden Clients entfernen
- WS-T ausführen

Erwartete Ausgabe:

- Ungerichteter Graph, der die neue reduzierte Anlagentopologie beschreibt

Erwartete Nachbedingung:

- Fehlen des Fördermoduls wurde erkannt und im Topologiegraphen vermerkt

Prüfanweisungen:

- Ergebnis des WS-T mit dem vorherigen Topologiegraphen vergleichen

7.2.3 T201a: Neue Rechenressourcen hinzufügen (Test-Szenario 1)

Das Hinzufügen neuer Rechenressourcen (Anwendungsfall *P201*) wird von diesem und dem folgenden Test *T201b* abgedeckt. Zum Testen, ob eine neue Rechenressource korrekt in das System integriert wird, wird auf ihr ein lokaler Webservice installiert. Kann dieser anschließend gesucht und genutzt werden und hat die Ausführung ausschließlich auf der neuen Rechenressource stattgefunden, wurde die neue Rechenressource offensichtlich korrekt durch das Produkt in das bestehende Netzwerk integriert.

Vorbedingung:

- Auf allen bestehenden Nodes ist die Basissoftware des Produkts installiert und gestartet
- Es ist kein systemfremder Webservice installiert

Durchzuführende Schritte:

- Neue Rechenressource an das Netzwerk anschließen
- Basissoftware des Produkts auf der neuen Rechenressource installieren und starten
- Konfiguration des lokalen Webservice zur Primzahlberechnung (WS-LP) so anpassen, dass dieser nur auf der soeben hinzugefügten neuen Rechenressource angeboten wird
- Webservice WS-LP installieren
- Webservice WS-LP suchen
- Webservice WS-LP nutzen

Erwartete Ausgabe:

- Die Ergebnisse der Primzahlberechnung

Erwartete Nachbedingung:

- Eine Suche nach dem Webservice WS-LP liefert als Dienstanbieter lediglich die neu hinzugefügte Rechenressource (Node)

- Die Ausführung des Webservice WS-LP hat nur auf der neuen Rechenressource stattgefunden

Prüfanweisungen:

- Anbietende Nodes ermitteln: Funktionalität zum Suchen nach Webservices verwenden und erhaltene Endpunktreferenz mit derjenigen der neuen Rechenressource vergleichen
- Ausführungsort ermitteln: CPU-Auslastung der Nodes überwachen

7.2.4 T201b: Neue Rechenressourcen hinzufügen (Test-Szenario 2)

Dieser Test deckt, wie bereits der vorangegangene Test *T201a* das Hinzufügen neuer Rechenressourcen ab. Im Unterschied zum Test *T201a* wird die neue Rechenressource zu einem Zeitpunkt hinzugefügt, zu dem alle bereits im System vorhandenen Nodes durch lokale Webservices stark ausgelastet sind. Weiterhin ist im System ein beweglicher Webservice installiert. Der Test prüft nun, ob nach dem Hinzufügen der neuen Rechenressource das Produkt die neue Rechenressource erkannt hat und für die Entlastung der weiteren Nodes nutzt. Dies ist daran zu erkennen, dass der vorher von den stark ausgelasteten Nodes angebotene bewegliche Webservice anschließend auf der neuen kaum ausgelasteten Rechenressource angeboten wird.

Vorbedingung:

- Das Netzwerk besteht aus genau fünf Nodes
- Auf allen bestehenden Nodes ist die Basissoftware des Produkts installiert und gestartet
- Es ist kein systemfremder Webservice installiert

Durchzuführende Schritte:

- Beweglichen Webservice WS-BP installieren
- Ursprüngliche fünf Nodes jeweils durch den lokalen Webservice WS-LP auslasten
  - Webservice WS-LP jeweils auf diesen 5 Nodes installieren (Konfiguration entsprechend anpassen)
  - Webservice WS-LP nutzen (für jede Node einzeln aufrufen)
- Neue Rechenressource an das Netzwerk anschließen
- Basissoftware des Produkts auf der neuen Rechenressource installieren und starten

Erwartete Ausgabe:

- Die Ergebnisse der Primzahlberechnung (mehrfach aufgrund mehrfacher Ausführung)

Erwartete Nachbedingung:

- Die neu hinzugefügte Rechenressource bietet den beweglichen Webservice WS-BP an, da für diesen auf den vorher bereits vorhandenen Nodes aufgrund der Auslastung durch die lokalen Primzahlberechnungen keine Rechenressourcen mehr verfügbar sind

Prüfanweisungen:

- Anbietende Node ermitteln: Funktionalität zum Suchen nach Webservices verwenden und erhaltene Endpunktreferenzen mit derjenigen der neuen Rechenressource vergleichen

7.2.5 T202: Rechenressourcen entfernen

Mithilfe dieses Tests kann getestet werden, ob das Produkt Rechenressourcen korrekt entfernt und somit den zu Rechenressourcen gehörenden Teil des Anwendungsfalls *P202* korrekt umsetzt. Auf einem initial mindestens fünf Nodes umfassendem Netzwerk sind zwei bewegliche Webservices installiert. Es werden nach und nach Rechenressourcen aus dem Netzwerk entfernt. Solange noch mind. eine Rechenressource vorhanden ist, müssen die zwei installierten beweglichen Webservices weiterhin angeboten werden, da bewegliche Webservices beim Entfernen von Rechenressourcen nicht mitentfernt werden dürfen. Anhand dieses Tests lässt sich nach jedem Entfernen einer Rechenressource die Verfügbarkeit der beweglichen Webservices überprüfen, indem jeweils die Nodes ermittelt werden, welche die Webservices anbieten.

Vorbedingung:

- Das Netzwerk besteht aus mindestens fünf Nodes
- Auf allen bestehenden Nodes ist die Basissoftware des Produkts installiert und gestartet
- Es ist kein systemfremder Webservice installiert

Durchzuführende Schritte:

- Webservice WS-BP installieren
- Webservice WS-BMM installieren
- Webservices WS-BP und WS-BMM suchen und diejenigen Nodes ermitteln, die die Webservices anbieten
- Entfernung einer oder mehrerer Rechenressourcen mit jeweils 30 Sekunden Abstand zwischen dem Entfernen zweiter Nodes

Erwartete Ausgabe:

- Keine

Erwartete Nachbedingung:

- Sowohl der Webservice WS-BP als auch der WS-BMM wird weiterhin von mindestens einer Node angeboten
- Die entfernten Nodes werden von einem Aufruf der Suchfunktion nicht mehr zurückgegeben

Prüfanweisungen:

- Anbietende Nodes ermitteln: Funktionalität zum Suchen nach Webservices verwenden und Endpunktreferenzen mit denjenigen vergleichen, die zuvor ermittelt wurden

7.2.6 T203: Statistiken anzeigen lassen

Dieser Test prüft, ob die Auslastungsstatistiken der im Netzwerk vorhandenen Hardwareressourcen abgerufen werden können, wie dies im Anwendungsfall *P203* beschrieben wurde. Zu diesem Zweck wird eine Node durch einen rechenintensiven lokalen Webservice stark ausgelastet, wobei die übrigen im System vorhandenen Nodes nicht ausgelastet werden. In den Auslastungsstatistiken sollte dies deutlich zu erkennen sein.

Vorbedingung:

- Auf allen bestehenden Nodes ist die Basissoftware des Produkts installiert und gestartet
- Es ist kein systemfremder Webservice installiert

Durchzuführende Schritte:

- Webservice WS-LP installieren (Konfiguration entsprechend anpassen)
- Webservice WS-LP suchen
- Webservice WS-LP nutzen
- Zum Produkt gehörenden Client ausführen und die Statistiken anzeigen lassen

Erwartete Ausgabe:

- Anzeige der Auslastung der einzelnen Nodes

Erwartete Nachbedingung:

- Die Auslastung derjenigen Node, die den Webservice WS-LP ausführt liegt bei etwa 100%
- Die Auslastung der übrigen Nodes sollte nahe bei 0% liegen

Prüfanweisungen:

- Angezeigte Auslastungen des Administrationspanels mit den unter *Erwartete Nachbedingung* genannten Werten vergleichen.

7.2.7 T301: Installation neuer Webservices

Dieser Test ermöglicht die Überprüfung, ob die Produktfunktionalität bzgl. dem Installieren neuer Webservices (Anwendungsfall *P301*) korrekt umgesetzt wurde. Aufgrund der drei verschiedenen Möglichkeiten einen neuen Webservice zu installieren, ist dieser Test in drei Varianten durchführbar. Je eine Variante pro Installationsverfahren:

Variante a: Jar-Datei im zum Produkt gehörenden Client auswählen

Variante b: USB-Stick anschließen

Variante c: Jar-Datei in den vom Deployment überwachten Ordner kopieren

Weiterhin kann bei der Durchführung des Tests zwischen lokalen und beweglichen Webservices variiert werden. Der Test ist erfolgreich, wenn nach der Installation der installierte Webservice gefunden und genutzt werden kann.

Vorbedingung:

- Auf allen bestehenden Nodes ist die Basissoftware des Produkts installiert und gestartet
- Es ist kein systemfremder Webservice installiert

Durchzuführende Schritte:

- Webservice WS-LP oder WS-BP mit dem jeweiligen Installationsverfahren installieren (Konfiguration entsprechend anpassen)

Erwartete Ausgabe:

- Keine

Erwartete Nachbedingung:

- Mind. eine Node bietet den Webservice WS-LP bzw. WS-BP an
- Der Webservice WS-LP bzw. WS-BP kann genutzt werden

Prüfanweisungen:

- Anbietende Nodes ermitteln: Funktionalität zum Suchen nach Webservices verwenden
- Nutzbarkeit feststellen: Webservice nutzen

7.2.8 T302: Modifikation installierter Webservices

Dieser Test überprüft die korrekte Umsetzung des Anwendungsfalls *P302*. Dabei wird ein beweglicher Webservice erstellt, der bei Aufruf einen bestimmten Text zurück gibt. Dieser wird anschließend modifiziert, sodass ein vom ursprünglichen Text verschiedener Text zurückgegeben wird. Unterscheiden sich die Aufrufe dieses Webservices vor und nach der Modifikation entsprechend der gewählten Texte, wurde die Modifikation installierter Webservices offensichtlich korrekt durch das Produkt umgesetzt.

Vorbedingung:

- Auf allen bestehenden Nodes ist die Basissoftware des Produkts installiert und gestartet
- Es ist kein systemfremder Webservice installiert

Durchzuführende Schritte:

- Beweglichen Webservice schreiben, der den Text „Hello World!“ zurück gibt
- Webservice installieren
- Webservice suchen
- Webservice nutzen
- Webservice modifizieren, sodass dieser nun den Text „Hello World! I’ve been modified!“ zurück gibt
- Webservice suchen
- Webservice nutzen

Erwartete Ausgabe:

- Der Text „Hello World!“ wird zurückgegeben
- Der Text „Hello World! I’ve been modified!“ wird zurückgegeben

Erwartete Nachbedingung:

- Keine

Prüfanweisungen:

- Ausgaben überprüfen

7.2.9 T303: Deinstallation installierter Webservices

Mithilfe dieses Tests wird die Deinstallation von bereits installierten Webservices geprüft, welche im Anwendungsfall *P303* beschrieben wurde. Dabei kann dieser Test sowohl für lokale als auch für bewegliche Webservices durchgeführt werden. Nachdem ein lokaler oder beweglicher Webservice installiert wurde, wird dieser mithilfe des zum Produkt gehörenden Clients deinstalliert. Der Test ist erfolgreich, wenn eine der Deinstallation folgende Suche nach dem deinstallierten Webservice erfolglos bleibt und der Webservice somit von keiner Node mehr angeboten wird.

Vorbedingung:

- Auf allen bestehenden Nodes ist die Basissoftware des Produkts installiert und gestartet
- Es ist kein systemfremder Webservice installiert

Durchzuführende Schritte:

- Webservice WS-LP oder WS-BP installieren (Konfiguration entsprechend anpassen)
- Webservice WS-LP bzw. WS-BP deinstallieren

Erwartete Ausgabe:

- Keine

Erwartete Nachbedingung:

- Keine Node bietet den Webservice WS-LP bzw. WS-BP an

Prüfanweisungen:

- Anbietende Nodes ermitteln: Funktionalität zum Suchen nach Webservices verwenden

7.2.10 T304: Webservice suchen

Die korrekte Umsetzung der im Anwendungsfall *P304* beschriebenen Suche nach Webservices kann mit diesem Test geprüft werden. Es wird dabei ein lokaler Webservice auf einer vorher bestimmten Node installiert und durch eine anschließende Suche ermittelt, ob der installierte Webservice gefunden werden kann.

Vorbedingung:

- Auf allen bestehenden Nodes ist die Basissoftware des Produkts installiert und gestartet
- Es ist kein systemfremder Webservice installiert

Durchzuführende Schritte:

- Webservice WS-LP installieren (Konfiguration entsprechend anpassen)
- Webservice WS-LP suchen

Erwartete Ausgabe:

- Es wird diejenige Node zurückgegeben, die den Webservice WS-LP anbietet

Erwartete Nachbedingung:

- Die Node, die durch die Suchfunktion zurückgegeben wird, stimmt mit derjenigen überein, die in der Konfigurationsdatei angegeben wurde

Prüfanweisungen:

- Endpunktreferenz der in der Konfiguration angegebenen Node mit derjenigen der von der Suchfunktion zurückgegebenen Endpunktreferenz vergleichen

7.2.11 T305a: Webservice nutzen (Test-Szenario 1)

Dieser und der folgende Test *T305b* dienen dem Überprüfen, ob das Produkt ermöglicht, Webservices zu nutzen, wie dies im Anwendungsfall *P305* bzw. *P305a* beschrieben wurde. Dieser Test betrachtet ausschließlich den Anwendungsfall *P305*. Der Test kann in zwei Varianten durchgeführt werden. Die erste Variante deckt den Fall ab, dass ein lokaler Webservice genutzt werden soll. Dieser wird ausschließlich auf der in der Konfiguration angegebenen Node ausgeführt und darf von keiner weiteren Node angeboten werden. Die zweite Variante deckt hingegen den Fall ab, dass ein beweglicher Webservice genutzt werden soll. Dieser darf im Unterschied zur ersten Variante von beliebig vielen Nodes angeboten werden. Ein Aufruf des beweglichen Webservices darf jedoch nur eine Ausführung auf genau einer Node zur Folge haben.

Vorbedingung:

- Auf allen bestehenden Nodes ist die Basissoftware des Produkts installiert und gestartet
- Es ist kein systemfremder Webservice installiert

Durchzuführende Schritte:

- Webservice WS-LP installieren (Konfiguration entsprechend anpassen) (nur erste Variante)
- Webservice WS-LP suchen (nur erste Variante)
- Webservice WS-LP nutzen (nur erste Variante)
- Webservice WS-BP installieren (nur zweite Variante)

- Webservice WS-BP suchen (nur zweite Variante)
- Webservice WS-BP nutzen (nur zweite Variante)

Erwartete Ausgabe:

- Das Ergebnis der Primzahlberechnung

Erwartete Nachbedingung:

- Eine Suche nach dem Webservice WS-LP liefert als anbietende Node lediglich die in der Konfiguration angegebene Node (nur erste Variante)
- Die Ausführung des Webservice WS-LP hat nur auf der in der Konfiguration angegebenen Node stattgefunden (nur erste Variante)
- Eine Suche nach dem Webservice WS-BP liefert als anbietende Nodes mind. eine Node zurück (nur zweite Variante)
- Die Ausführung des Webservice WS-BP hat nur auf genau einer Node stattgefunden (nur zweite Variante)

Prüfanweisungen:

- Anbietende Nodes ermitteln: Funktionalität zum Suchen nach Webservices verwenden
- Ausführungsort ermitteln: CPU-Auslastung der Nodes überwachen

7.2.12 T305b: Webservice nutzen (Test-Szenario 2)

Auch dieser Test überprüft die korrekte Umsetzung des Produkts bzgl. der Nutzung von Webservices, bezieht sich dabei jedoch im Unterschied zum Test *T305a* auf Webservices, welche eine verteilte Berechnung durchführen, wie dies im Anwendungsfall *P305a* beschrieben wurde. Der Test kann ebenfalls sowohl für lokale (Variante a) als auch für bewegliche (Variante b) Webservices durchgeführt werden. Variante a deckt den Fall ab, dass ein lokaler Webservice genutzt werden soll, welcher eine parallele Berechnung auf mehreren Nodes durchführt. Dieser darf ausschließlich von der in der Konfiguration angegebenen Node angeboten werden. Die Durchführung der Berechnung muss jedoch parallel erfolgen. Die Variante b deckt hingegen den Fall ab, dass ein beweglicher Webservice genutzt werden soll, welcher eine parallele Berechnung auf mehreren Nodes durchführt. Dieser darf im Unterschied zu Variante a von beliebig vielen Nodes angeboten werden. Zur Überprüfung, ob die Berechnung tatsächlich verteilt stattgefunden hat, können die Auslastungen der einzelnen Nodes überwacht werden.

Vorbedingung:

- Auf allen bestehenden Nodes ist die Basissoftware des Produkts installiert und gestartet

- Es ist kein systemfremder Webservice installiert

Durchzuführende Schritte:

- Webservice WS-LMM installieren (Konfiguration entsprechend anpassen) (nur Variante a)
- Webservice WS-LMM suchen (nur Variante a)
- Webservice WS-LMM nutzen (nur Variante a)
- Webservice WS-BMM installieren (nur Variante b)
- Webservice WS-BMM suchen (nur Variante b)
- Webservice WS-BMM nutzen (nur Variante b)

Erwartete Ausgabe:

- Das Ergebnis der Matrix-Multiplikation

Erwartete Nachbedingung:

- Eine Suche nach dem Webservice WS-LMM liefert als anbietende Node lediglich die in der Konfiguration angegebene Node (nur Variante a)
- Eine Suche nach dem Webservice WS-BMM liefert als anbietende Nodes mind. eine Node zurück (nur Variante b)
- Die Ausführung des Webservice WS-LMM bzw. WS-BMM hat parallel auf mehreren Nodes stattgefunden

Prüfanweisungen:

- Anbietende Nodes ermitteln: Funktionalität zum Suchen nach Webservices verwenden
- Ausführungsort ermitteln: Log-Dateien von GridGain überprüfen / CPU-Auslastung der Nodes überwachen

7.2.13 T501: Lastabhängige Verteilung von Webservices

Dieser Test deckt die Produktfunktion *P501* ab. Auf denjenigen Nodes, die einen zuvor installierten beweglichen Webservice anbieten, wird zusätzlich ein lokaler Webservice installiert, der für eine sehr hohe Auslastung der Node sorgt. Das Produkt hat die lastabhängige Verteilung korrekt umgesetzt, wenn es dafür sorgt, dass der bewegliche Webservice anstatt weiterhin von den ausgelasteten Nodes von weniger ausgelasteten Nodes angeboten wird.

Vorbedingung:

- Auf allen bestehenden Nodes ist die Basissoftware des Produkts installiert und gestartet
- Es ist kein systemfremder Webservice installiert

Durchzuführende Schritte:

- Webservice WS-BP installieren
- Webservice WS-BP suchen und diejenigen Nodes ermitteln, die den Webservice WS-BP anbieten
- Diejenigen Nodes, die den Webservice WS-BP anbieten jeweils durch den lokalen Webservice WS-LP auslasten
  - Webservice WS-LP jeweils auf diesen Nodes installieren (Konfiguration entsprechend anpassen)
  - Webservice WS-LP nutzen (für jede Node einzeln aufrufen)

Erwartete Ausgabe:

- Die Ergebnisse der Primzahlberechnung (mehrfach aufgrund mehrfacher Ausführung)

Erwartete Nachbedingung:

- Das Produkt hat die hohe Auslastung durch die lokalen Webservices WS-LP erkannt und veranlasst weitere Nodes dazu den beweglichen Webservice WS-BP anzubieten
- Der bewegliche Webservice WS-BP wird nicht mehr von den ursprünglich anbietenden Nodes angeboten, da diese aufgrund der Auslastung durch die lokalen Primzahlberechnungen keine Rechenressourcen mehr verfügbar haben
- Der bewegliche Webservice WS-BP wird von Nodes angeboten, die verschieden von den ursprünglich anbietenden Nodes sind

Prüfanweisungen:

- Anbietende Nodes ermitteln: Funktionalität zum Suchen nach Webservices verwenden

7.2.14 T502: Sicherstellung der Verfügbarkeit von beweglichen Webservices

Die in der Produktfunktion *P502* beschriebene Sicherstellung der Verfügbarkeit von beweglichen Webservices kann anhand dieses Test geprüft werden. Es werden dabei nacheinander diejenigen Nodes vom Netzwerk getrennt, die einen zuvor installierten beweglichen Webservice anbieten. Für die Sicherstellung der Verfügbarkeit muss das System auf den Verlust der anbietenden Nodes reagieren und den beweglichen Webservice zusätzlich auf weiteren Nodes anbieten. Der Test wurde erfolgreich absolviert, wenn das Produkt dies leistet.

Vorbedingung:

- Das Netzwerk besteht aus mindestens fünf Nodes
- Auf allen bestehenden Nodes ist die Basissoftware des Produkts installiert und gestartet
- Es ist kein systemfremder Webservice installiert

Durchzuführende Schritte:

- Webservice WS-BP installieren
- Webservice WS-BP suchen und diejenigen Nodes ermitteln, die den Webservice WS-BP anbieten
- Trennung der Verbindung einer den Webservice WS-BP anbietenden Node zum Netzwerk
- Wiederholung des vorherigen Schrittes nach Abwarten von jeweils 30 Sekunden bis alle der in Schritt 3 ermittelten Nodes vom Netzwerk getrennt wurden

Erwartete Ausgabe:

- Keine

Erwartete Nachbedingung:

- Das Produkt hat die Trennung der anbietenden Nodes vom Netzwerk erkannt und und veranlasst weitere Nodes dazu den beweglichen Webservice WS-BP anzubieten
- Der bewegliche Webservice WS-BP wird von Nodes angeboten, die verschieden von den ursprünglich anbietenden Nodes sind

Prüfanweisungen:

- Anbietende Nodes ermitteln: Funktionalität zum Suchen nach Webservices verwenden und Endpunktreferenzen der Nodes vergleichen

ANHANG B

Administrator Manual

Administration Manual

Field Level Computation Cloud

PG 557, Department of Computer Science, TU Dortmund

Martin Bring, Iryna Denysenko, Katharina Diekmann, Tim Düllmann,
René Grzeszick, Katrin Holterbork, Sebastian Homann, Henning Meister,
Stanislav Nikolov, Marco Pfahlberg, Dirk Schalge, Nils Schmidt

SUPERVISING TUTORS: Sascha Feldhorst and Christian Mosblech

winter semester 2011/2012
DRAFT, JUNE 1, 2012

Contents

1 Introduction

The purpose of this manual is to help administrators operate the FiLeCC System. This manual covers every aspect about FiLeCC starting with installation and setup, and closing with customary common tasks.

1.1 Contents of this Book

This manual is divided in to three chapters. The first chapter of the book introduces this manual and the manuals contents, with a quick look on the licence of FiLeCC and marketing purposes.

The second chapter (2) describes the product environment and following the hardware and software requirements for FiLeCC.

Chapter 3 describes all administration tasks and working in the life cycle of FiLeCC, beginning with a description of the user roles in section 3.1, and followed by a use case diagram that gives a quick overview over FiLeCC in section 3.2.

The administrator responsibilities and the admin panel are the main issue of the following section 3.3.

The main issue in section 3.4 is working with the basic configuration of FiLeCC and section 3.5 describes the user database configuration.

The last section 3.6 of the third chapter describes the logger mechanism used to store status data.

1.2 Licensing

Licence: *GNU Lesser General Public License* (LGPL), Version 3, 29th June 2007.

LGPL is a license for free software and was published by the Free Software Foundation (FSF). A software under the LGPL can be used for any purpose. It can be changed, modified and republished. For details see ([Sys12]).

2 Setting up the Environment – Technical Aspects

FiLeCC is a software system being able to host and manage applications in a cloud environment. Every computer running FiLeCC automatically becomes a node in the cloud. For this purpose it is necessary for all nodes to be in one local network. Though FiLeCC can manage situations in which the network is disconnected, those particular situations should be exceptional cases. Further hardware (see section 2.1) and software (see section 2.2) requirements are listed in the following sections.

2.1 Hardware Requirements

FiLeCC has low hardware requirements because the whole system is intended for resource-restricted systems. The used hardware platform has to be either the x86-platform or the ARM-platform. Both require at least 512MB of main memory and a 1 GHz CPU. Furthermore, the system needs at least 1GB of persistent memory, while 2GB of persistent memory are recommended to ensure a unhindered programm execution.

2.2 Software Requirements

To run FiLeCC, at least Linux 2.6 is required as the operating system. To avoid interaction with software being responsible for managing the conveyor system, FiLeCC has to run in an own virtual machine. Therefore, Oracle's *Java Embedded* is used to execute FiLeCC. To provide the opportunity of distributed computing, the cloud computing platform *GridGain* together with the execution environment *Java SE* is required and therefore must be supported.

2.3 Operating Conditions

Communication between the nodes is achieved by the TCP and UDP protocols. Therefore, the underlaying network has to support these protocols. Since some components of FiLeCC are using multicast, this also has to be supported. Furthermore, FiLeCC utilizes Web services and therefore (according to the *WS-Discovery* specification) the IP address 239.255.255.250 and the UDP port 3702 have to be reachable and must not be blocked by a firewall. The cloud computing platform *GridGain* also uses multicast so the IP address 228.1.2.4 and port 47200 should be reachable as well. For internal purposes, particularly for the leader election processes and the logging mechanism which sends log messages to the admin panel, there are additional multicast IP addresses and ports which have to be reachable by FiLeCC. In particular these are the IP address 224.0.1.1 and the port 51001 for the logging mechanism and the IP addresses 224.0.1.10 up to 224.0.1.12 for the leader election processes. The corresponding ports are 51000 and 51020.

3 Working with FiLeCC

This chapter deals with the administration of FiLeCC. Section 3.1 presents the three different actors and their roles regarding FiLeCC. The use cases of these actors are presented in section 3.2 and explained in detail in section 3.3. After the description of the use cases, section 3.4 explains how to set system parameters of FiLeCC in order to adjust the software to the needs of specific environments. Section 3.5 handles the configuration of the user database. Finally section 3.6 describes a logging mechanism which can be used by developers to integrate their own log messages into the ones of FiLeCC. This enables the logging messages of external applications to be shown in the admin panel, too.

3.1 Roles

FiLeCC can be used by three different roles (see Figure 1). The first role is the administrator which is responsible for all administrative tasks like managing hardware resources and conveyor modules and taking care of the Web services which should be executed on the FiLeCC platform.

The developer creates software to be executed on the FiLeCC platform. This software can be installed by the administrator.

The control layer systems are systems which interact with FiLeCC by using the Web services offered by FiLeCC.

3.2 Use Cases

FiLeCC supports several use cases illustrated in Figure 1. It is possible to add or remove a conveyor module or a hardware resource. Furthermore the topology of conveyor modules can be displayed and statistical data can be accessed.

Since FiLeCC is a Web-service-based system, the most important job is to manage the Web services. The administrator has the possibility to install new Web services and remove or modify already existing ones. All installed Web services can be searched and used by the administrator and systems of the control layer.

A detailed description of these use cases is given in the following section.

3.3 Administrator Responsibilities & Admin Panel

This subsection describes the duties of the administrator. The use cases are explained by a step by step documentation.

3.3.1 Add hardware resources

Adding new hardware resources, i.e. nodes, to FiLeCC is simple. After integrating the new node into the network the FiLeCC software has to be started. This is done by executing the start script "start.sh". Optionally, the administartor can modify the standard configuration of FiLeCC before starting it. This is detailed in section 3.4. After starting FiLeCC, the new node will integrate itself into the FiLeCC cloud.

3.3.2 Remove hardware resources

To remove a node from the FiLeCC cloud the FiLeCC software has to be stopped on that particular node. One possible way to achieve this is sending `SIGTERM` to the FiLeCC process.

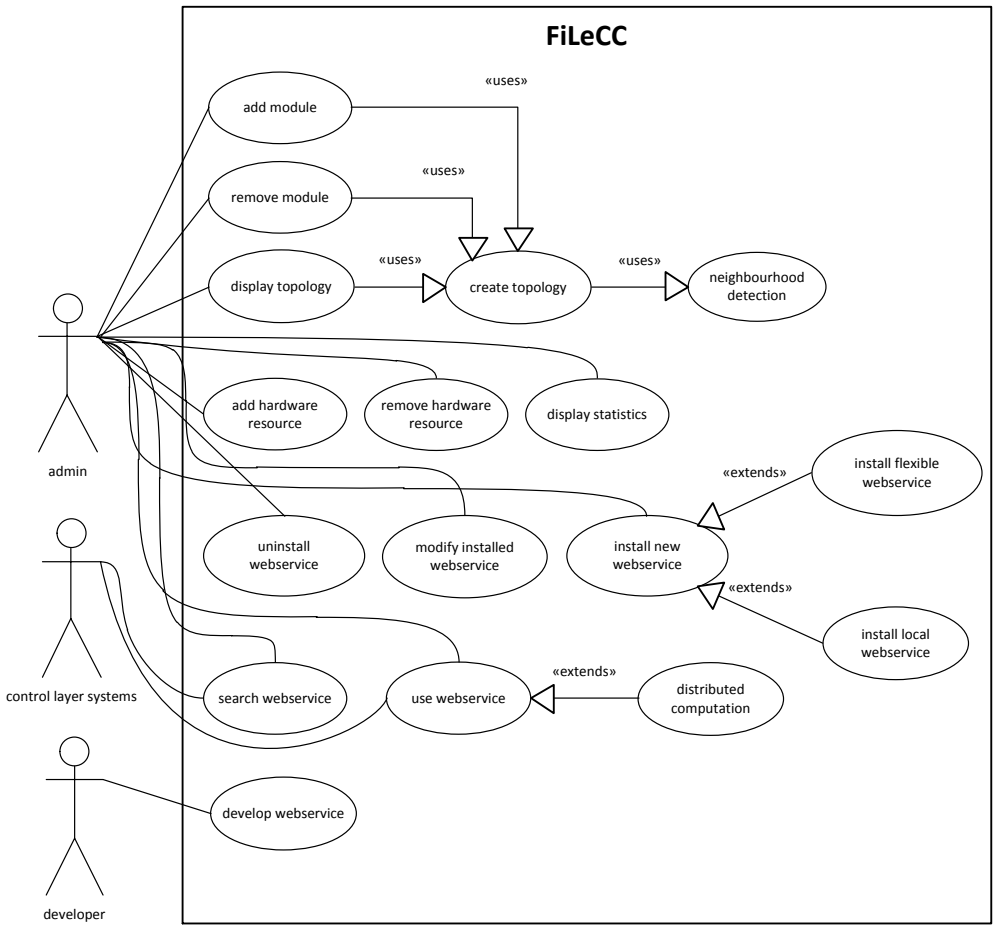


Figure 1: Use Case Diagramm for FiLeCC.

3.3.3 Installing new application

To install new software into the cloud, a compiled Java `jar` file fitting the FiLeCC requirements described in the "Add and remove Apps to and from FiLeCC " chapter of the FiLeCC Developer's Manual is needed. In order to install the file, open the binary tab and left-click the "Install new binary" button. In the following dialog, select the file to be installed and left-click "Open". See screenshot in Figure 8.

3.3.4 Uninstall application

Go to the binary tab and select the binary entry in the list on the left-hand side. Then left-click "Remove from platform" to stop and undeploy the binary on all nodes. See screenshot in Figure 8.

3.3.5 Start application

To start another instance of a software, open the binary tab, select a binary in the list on the left-hand side and left-click "Start on new node". The FiLeCC will select a suitable node depending on its load and start the software on that node. Another way to start the binary on another node is to select the node directly, by selecting a node in the topology tab and right-clicking it to open the edit dialog for that node. Then select a the binary in the list of deployed binaries or the list of binaries on the platform and left-click the "Start binary" button below the list of the selected binary. See the screenshot in Figure 7.

3.3.6 Stop application

To stop an application on all nodes, open the binary tab and select the application you want to stop in the list on the left-hand side. Then left-click the "Stop on all nodes" button. To stop an application only on one node, select the node in the topology tab and right-click it to open the node edit dialog. Then select the application entry in the "Binaries running on this node" list and left-click the "Stop binary" button. See screenshot in Figure 7.

3.3.7 Changing leader

The administrator is allowed to change the elected leader. In order to change the leader open the node edit dialog of a leader by right-clicking on the node in the topology tab and left-click the "Elect other UNO" or "Elect other watchdog." Using the "Elect this node as UNO" or "Elect this node as Watchdog" button elects the selected node as UNO or Watchdog. See screenshot in Figure 7.

3.3.8 Order a node to call for help

If a node exceeded its resources, it calls for help itself. When calling for help manually, a help request is sent to the UNO. The UNO then tries to start all applications of the selected node on another node. The "Call for help" button can be found on the node edit dialog. See the screenshot in Figure 8.

3.3.9 Viewing logs

The Admin Panel provides several filters for log messages. These can be found on the bottom panel of the log tab. The filters include one for the Logger of the message, one for the log level and numerous display options. See screenshot in Figure 9.

3.3.10 Add conveyor module

In order to add a conveyor module to FiLeCC it is necessary to establish a network connection between the conveyor module and an existing node in the FiLeCC cloud. Then FiLeCC has to be informed about the new conveyor module. This can be done by using the admin panel. The tab "BoardCommunication" lists all conveyor modules which are currently integrated into FiLeCC. See screenshot in Figure 2.

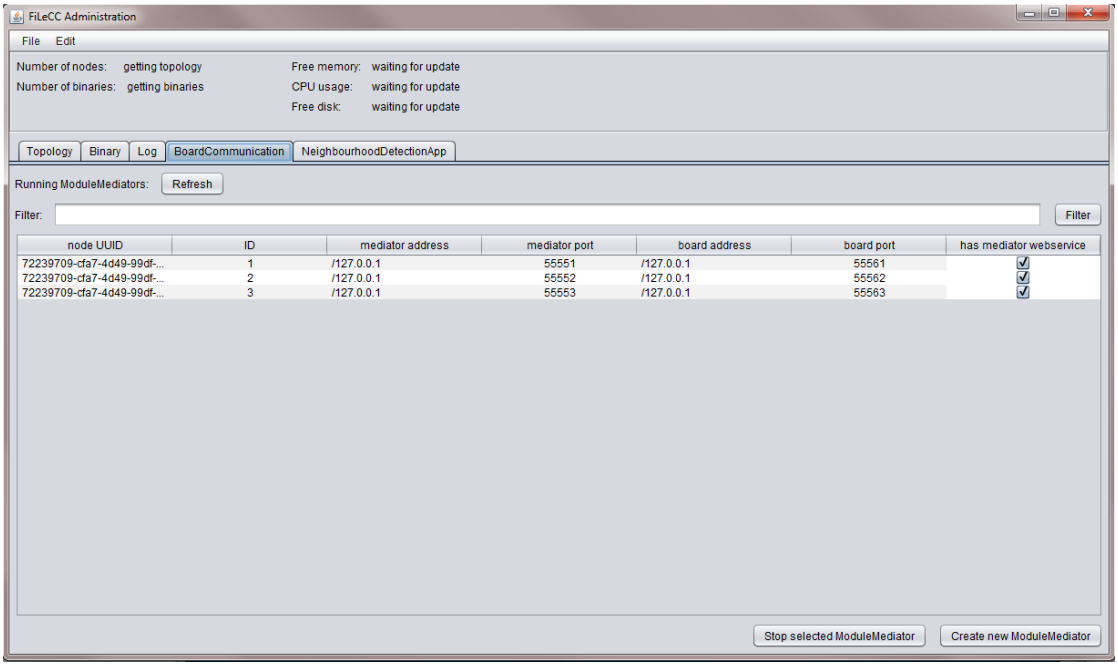


Figure 2: Admin Panel: Tab BoardCommunication

To inform FiLeCC about the new conveyor module, a new `ModuleMediator` has to be created. The button "Create new ModuleMediator" offers a configuration dialog (see Figure 3) for creating a new `ModuleMediator` which is the FiLeCC component for managing the communication with conveyor modules.

To create a new `ModuleMediator` the UUID of the target node, i.e. the node being connected to the new conveyor module, the IP address and port of the conveyor module and the IP address and port which the `ModuleMediator` will use to listen for incoming requests have to be specified. Optionally, it is possible to specify wether or not the `ModuleMediator` is available as Web

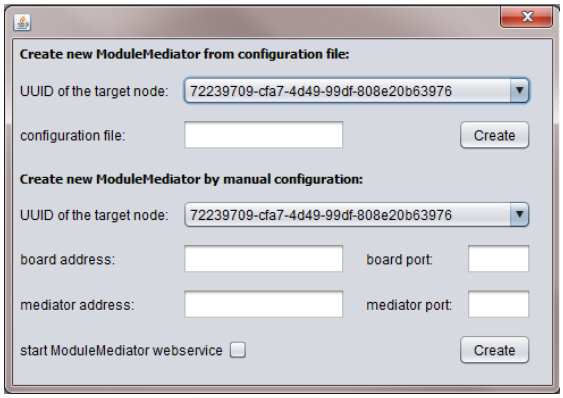


Figure 3: Admin Panel: `ModuleMediator` configuration dialog

service. These parameters can be specified manually or by entering the name of a configuration file containing these parameters. The configuration dialog offers both alternatives. Note that the configuration file has to be part of the FiLeCC jar file of the target node. Configuration files have to be placed into the "mediator_configs" folder of the jar file.

3.3.11 Remove conveyor module

To remove a conveyor module, FiLeCC has to be informed about it, i.e. stopping the corresponding `ModuleMediator`. The tab "BoardCommunication" lists all currently running `ModuleMediators`. See screenshot in Figure 2. After selecting the `ModuleMediator` corresponding to the conveyor module which should be removed and pressing the button "Stop selected ModuleMediator" the module mediator is stopped. After stopping the `ModuleMediator` the conveyor module can be disconnected from the node.

3.3.12 Display topology

The topology is displayed in the tab "Topology". See screenshot in Figure 4.

If the `TopologyApp` is not started, the topology only consists of the currently running nodes instead of the conveyor modules. If the `TopologyApp` is running, the conveyor modules are displayed in the topology and neighbourhood relationships are represented by edges between the conveyor modules. To display a correct topology the `TopologyApp` needs a `NeighbourhoodDetectionService` for each `ModuleMediator`, i.e. for each conveyor module. Conveyor modules without a running `NeighbourhoodDetectionService` will not be displayed in the topology. Starting a new `NeighbourhoodDetectionService` can be done by using the admin panel. The tab "NeighbourhoodDetectionApp" lists all `NeighbourhoodDetectionServices` currently running. See screenshot in Figure 5.

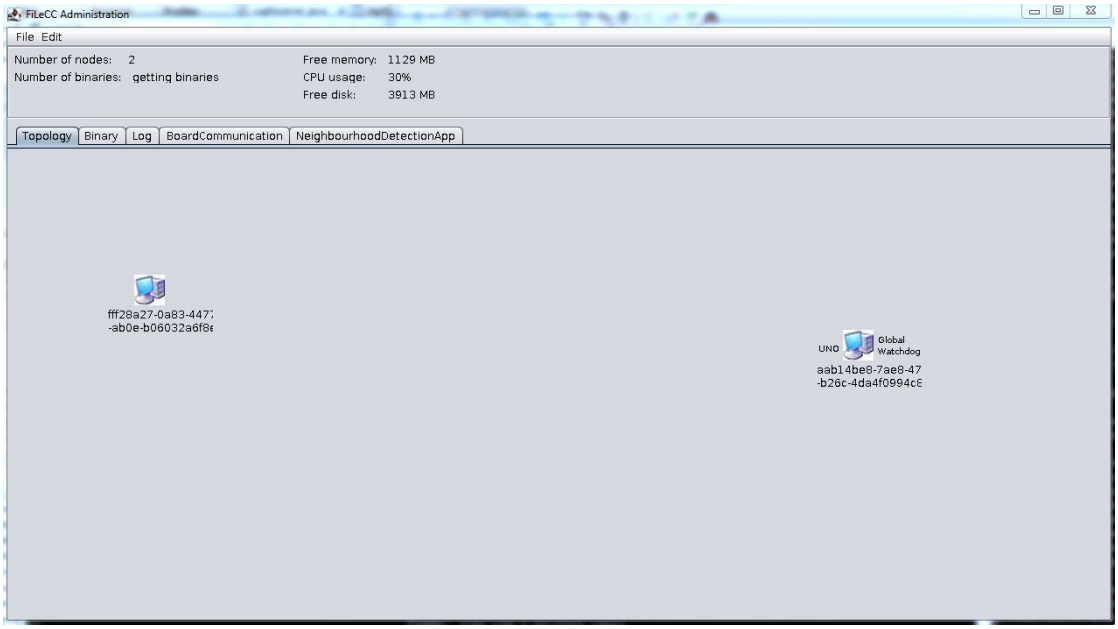


Figure 4: Admin Panel showing the topology.

To add a new `NeighbourhoodDetectionService` press the button "Create new NeighbourhoodDetectionService" and enter the requiered parameters in the pop-up dialog. See screenshot in Figure 6.

The UUID of the target node, i.e. the node running the `ModuleMediator` which manages the communication with the conveyor module, the mediator's address, its port and the URI of the conveyor module have to be specified. These parameters can either be entered manually or specified by a configuration file (which must be included in the FiLeCC jar file of the target node in the "serviceProperties" folder). The configuration dialog offers both alternatives. Remember to stop the corresponding `NeighbourhoodDetectionService` when removing a conveyor module.

3.4 Basic Configuration

While working with FiLeCC, it is possible to change technical details of the system by using a configuration file. The file 'filecc.properties' is included in the 'filecc.jar' and will be read by the system at startup. It is possible to configure each node differently, however it is not recommended to have major differences between the nodes. Especially the behavior of global components should be the same on all nodes. The following section includes an overview of the parameters that can be altered (the bold name is the key used in the property file).

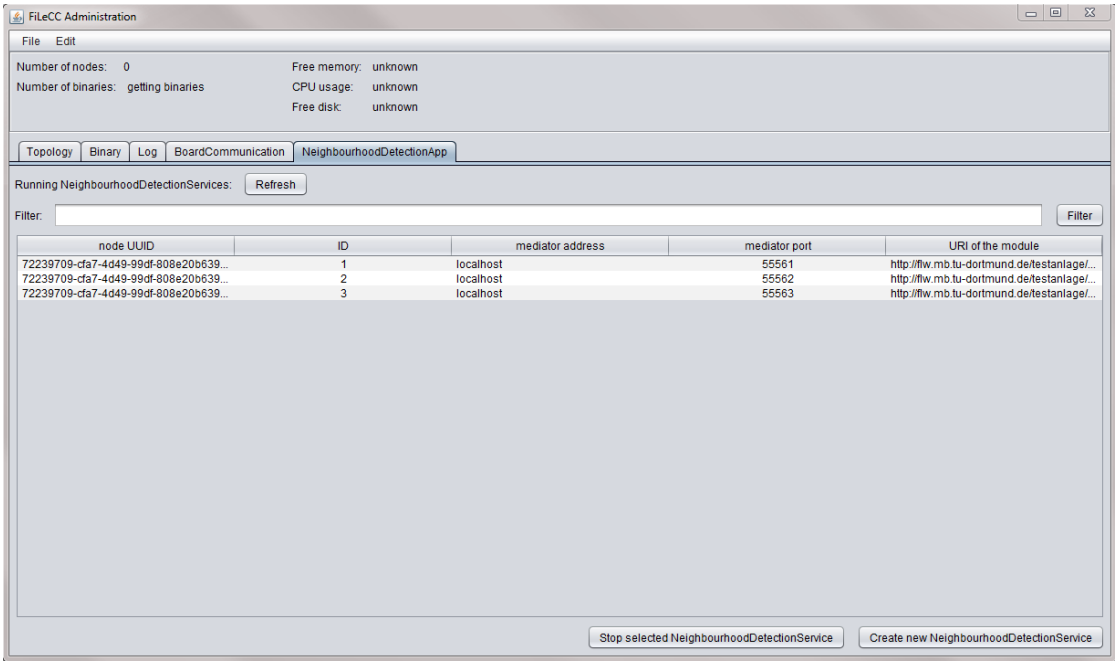


Figure 5: Admin Panel: Tab "NeighbourhoodDetectionApp"

HAZELCAST LOADDB MAP NAME is a unique name that can be used in order to identify the database, which stores the load data of the cloud nodes (short: LoadDB). This parameter must be the same on all nodes, otherwise there will be two or more different databases. Expected type: String.

HAZELCAST MIN RAM FOR START defines the minimum the amount of free RAM [MB] being required for the node to start a new instance of the Hazelcast LoadDB. Expected type: Integer.

HAZELCAST BACKUP COUNT defines how many times the load data is replicated within the cloud to avoid data loss. It is recommended to enlarge this number with the size of the cloud, since netsplits might affect more nodes than in smaller clouds. It is important to keep this parameter consistent on all nodes, otherwise the node with the largest memory creates more backups, while other nodes try to scale the number of backups down again. Expected type: Integer.

HAZELCAST CLIENTSTART CONNECTIONCHECK DEFAULTTIMEOUT is a timeout value in milliseconds used by the LoadDB clients. If this value is exceeded the connection attempt to the LoadDB is refused and therefore no database will be available. At system start this causes the system to start a new database. Expected type: Integer.

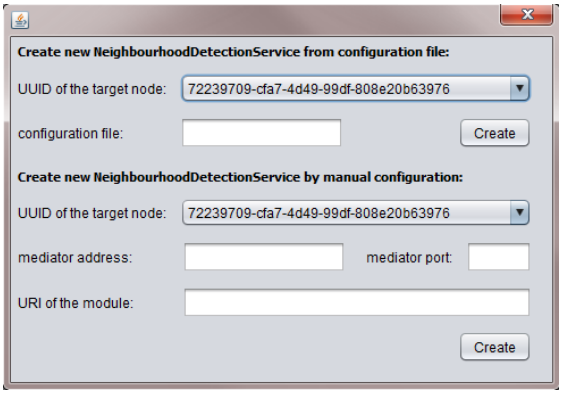


Figure 6: Admin Panel: `NeighbourhoodDetectionService` configuration dialog

HAZELCAST CLIENTSTART CONNECTIONCHECK INTERVALL defines time intervals in milliseconds, during which the connection to the load database will be attempted. Combined with the timeout a total number of connection attempts is specified. Expected type: Integer.

UUID FILE WINDOWS and UNIX is the directory in which the text file containing the UUID, used as a unique identifier for the node, is stored. This way the UUID of a node does not change. Expected type: String.

SDRM PATH WELCOME WIN and LINUX are the paths to the directories that are used for incoming files (using a Windows or Linux operating system). This folder is observed by the `SDRM`'s `WelcomeMonitor`. Expected type: String.

SDRM PATH STORAGE WIN and LINUX are the paths to the directories being used for deployed files. Expected type: String.

SDRM PATH DUMPSTER WIN and LINUX are the paths to the directories that are used for files that could not be processed by the `SDRM`. Expected type: String.

SDRM PATH TEMP WIN and LINUX are the paths to the directories that are used for temporary files. Expected type: String.

SDRM PATH BINARIES WIN and LINUX are the paths to the directories that are used in order to extract packed binaries, in case the application is not a jar-file. Expected type: String.

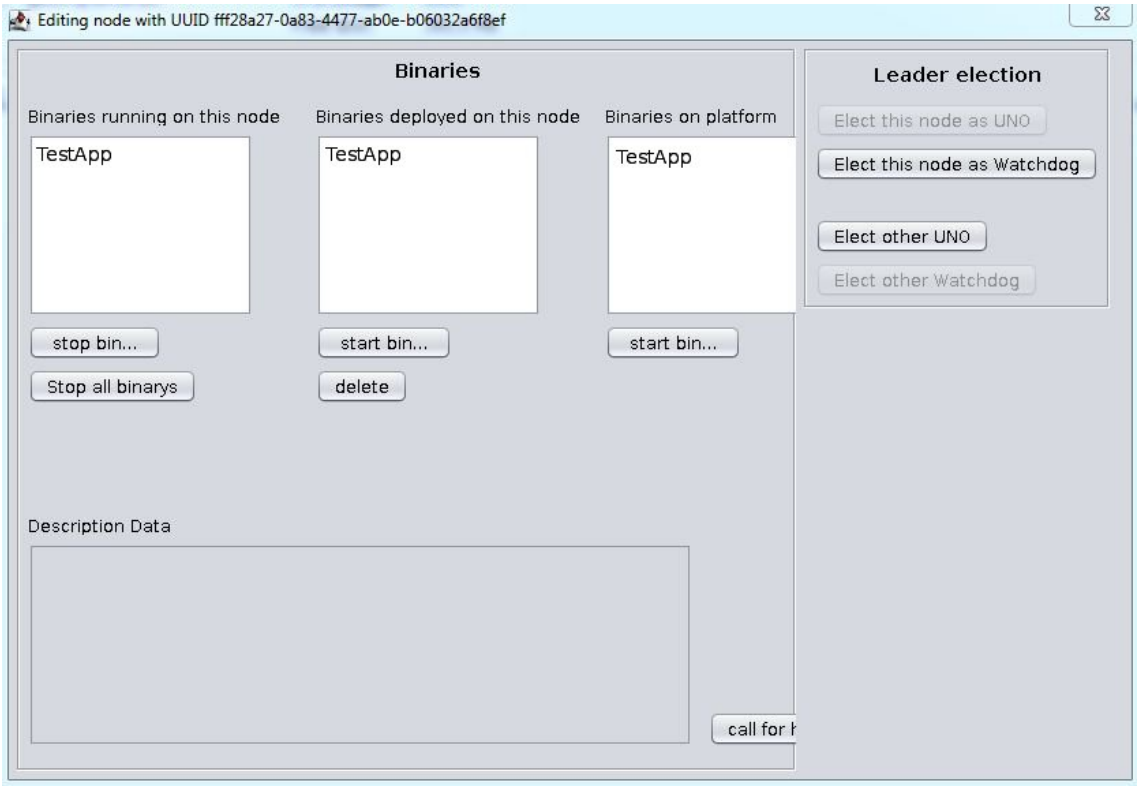


Figure 7: Admin Panel dialog for editing a node.

DESCRIPTION FILENAME is the filename of a FiLeCC 'xml-description-file' that has to be stored in the root folder of external programs that are run in FiLeCC. Expected type: String.

MONITORLOOP INTERVAL specifies how often the `MonitorLoop` checking the welcome folder is checked (in milliseconds). Expected type: Integer.

SDRM RANDOM DEPLOYMENT FACTOR defines a percentage value on how many nodes software that is installed in FiLeCC will be replicated to avoid any kind of data loss and make it accessible. Expected type: Double.

SDRM KEEP FREEDISKSPACE is a minimum value in megabytes of hard disk space that will not be used by the `SDRM` when deploying files. This way it is assured that the deployment does not use all the hard disk space. Expected type: Integer.

WATCHDOG GLOBAL WATCHDOG PORT is the port used by the `GlobalWatchdog` to contact the `LocalWatchdogs` to detect malfunctioning nodes. Expected type: Integer.

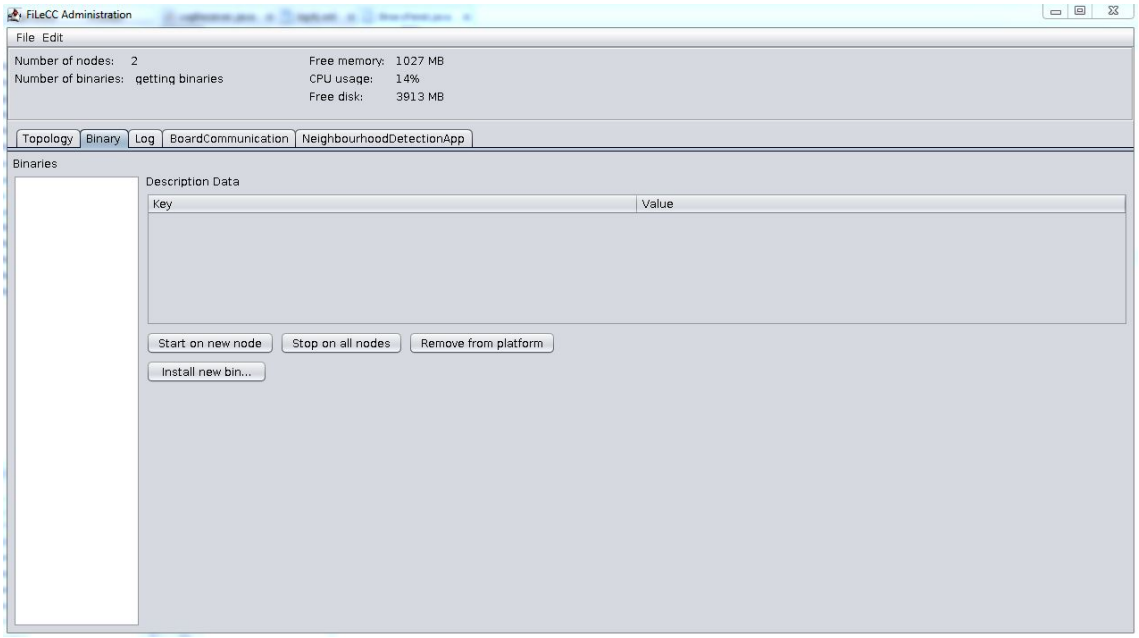


Figure 8: Admin Panel showing the binary information.

WATCHDOG LOCAL WATCHDOG PACKET INTERVAL specifies a time interval in which the `LocalWatchdogs` send UDP packages to the `GlobalWatchdog` to confirm that they are still running. Expected type: Integer.

WATCHDOG LOCAL WATCHDOG TIMEOUT specifies the timespan in milliseconds during which the `LocalWatchdog` must successfully submit one UDP package to the `GlobalWatchdog` before being marked as 'dead'. Expected type: Integer.

LOCAL WATCHDOG WEBSERVICE PORT is the port at which the `LocalWatchdog` hosts his Web services. Expected type: Integer.

WATCHDOG SERVICE PROBE INTERVAL specifies the time interval in milliseconds between sending two probe messages. Used in combination with the number of maximum missing probe answers to determine whether a device has to be marked as 'dead'. Expected type: Integer.

WATCHDOG MAX MISSING PROBE ANSWERS defines the number of missing probe answers before a device is marked as 'dead'. Expected type: Integer.

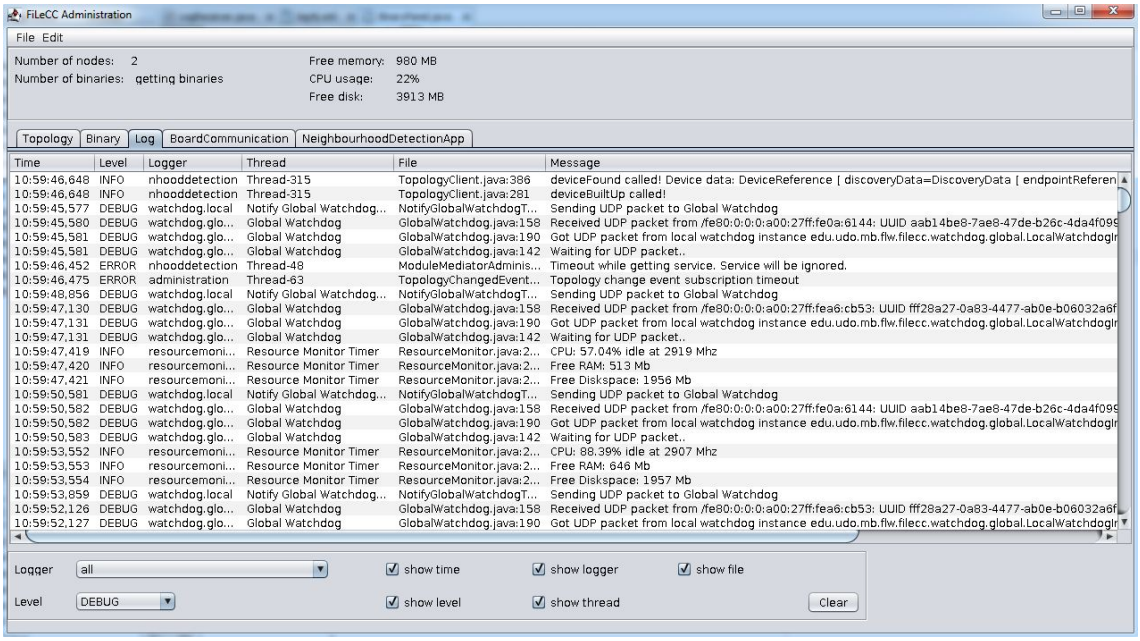


Figure 9: Admin Panel showing the log.

RESOURCEMONITOR DEFAULT UPDATETIME specifies a time interval (in seconds) in which the `ResourceMonitor` checks the node's load and therefore, determines whether or not the node is overloaded. Expected type: Integer.

RESOURCEMONITOR MIN CPU PERC FREE and MIN RAM FREE defines minimum values for the CPU idle time in percent and the free RAM in megabytes. If those requirements are not met, the node is possibly overloaded. Expected type: Double.

RESOURCEMONITOR ITERATIONS BEFORE HELP specifies a consecutive number of times the node has to be detected as overloaded, before it contacts the `UNO` and asks for help. Expected type: Integer.

3.5 User database configuration

In addition to the internal FiLeCC database configuration, it is possible to configure the external developers database. This database is provided for external developers who have custom storage requirements. Its functionality and guidelines for its use are provided in the accompanying Developer's Manual. The configuration parameters are saved in the 'gg_conf_user.xml' file. By default, a sample configuration with one preconfigured cache is provided. In order to extend the functionality, this file needs to be adjusted accordingly.

3.6 log4j – XML-Konfiguration

FileCC uses logging mechanisms to store status data. To ensure a consistent logging method for all services, the FiLeCC application provides a special logger, it is called `filecc.extern`. The configuration of this logger can be found in the log4j.xml configuration file which is deployed with the whole application. To use the logger, the method `Logger.getLogger("filecc.extern")` has to be called. To log messages the method `Logger.getLogger("filecc.extern").error("message")` has to be called.

References

[Sys12] System, The GNU O.: *GNU Lesser General Public License v3.0 - GNU Project - Free Software Foundation (FSF)*. online, 2012. – http://www.gnu.org/licenses/lgpl.html

ANHANG C

Developer's Manual

Developer's Manual

Field Level Computation Cloud

PG 557, Department of Computer Science, TU Dortmund

Martin Bring, Iryna Denysenko, Katharina Diekmann, Tim Düllmann, René Grzeszick, Katrin Holterbork, Sebastian Homann, Henning Meister, Stanislav Nikolov, Marco Pfahlberg, Dirk Schalge, Nils Schmidt

SUPERVISING TUTORS: Sascha Feldhorst and Christian Mosblech

winter semester 2011/2012
DRAFT, JUNE 1, 2012

Contents

1 Introduction

The purpose of this manual is to help developers improve and extend, add new features and write new apps for FiLeCC software in the future.

Until now many new cloud-based applications have been developed. This manual covers all information and necessary knowledge which is needed by the developer to extend the software.

1.1 Contents of this Book

This manual is divided in to five chapters. The first chapter of the book introduces this manual and the manuals contents, with a quick look on the glossary, where the technical terms used in the manual are explained.

In the second chapter the general concept, the purpose, and further the layout of FiLeCC are explained by reference to diagrams of the internal structure.

The main issue in the third chapter is packaging and the manifest file of FiLeCC needed for installation of the software.

In the fourth chapter there is a quick reference on each of the FiLeCC modules.

Finally the fifth chapter refers to the Licence of FiLeCC and marketing purposes.

1.2 Glossary

The following glossary contains definitions of technical terms used in this manual.

Cloud A Cloud is an abstract IT-infrastructure that provides services in the form of computing resources or capacity to end users. However, the end user does not needs to know of the actual architecture.

PaaS – Platform as a service, Provides Computing resources.

IaaS – Infrastructure as a Service, Provides storage capacity.

DPWS Abbreviation for *Devices Profile for Web Services*. DPWS specifies a standard that allows Web services to be used on embedded systems with limited resources.

FiLeCC Abbreviation for *Field Level Computing Cloud*. FiLeCC refers to the part of the software that is developed and implemented by the Project Group 557.

GridGain GridGain is a Java-based framework for computer clusters that allows distributed computing and executing of Java programs using MapReduce-Algorithmus.

IPC Abbreviation for *Industrial PC*. Industrial PCs are PC-based computing platforms designed for the use in an industrial environment and protected against environmental influences and electromagnetic interference.

Node A node is a unit that provides computational capacity within the network, e.g. an Industrial PC.

Pandaboard The Pandaboard is a small, cheap, and power-saving unit based on the Texas Instruments OMAP4430 processor. It is equipped with Dual-Core 1 GHz ARM Cortex-A9 MPCore CPU and PowerVR SGX540 GPU, with 1GB DDR2-SDRAM.

Service-Oriented Architecture A *Service-Oriented Architecture* (SOA) is an abstract concept for software architecture. It describes the collaboration between service providers and service consumers to provide a higher level of interoperability.

2 Introduction to the Development

This chapter deals with a general description of FiLeCC. First the purpose of FiLeCC is explained in section 2.1 and in section 2.2 the main components of FiLeCC are introduced.

2.1 What is FiLeCC?

FiLeCC is a computer program that can host and manage applications in a cloud environment. Every computer connected to the network and running FiLeCC automatically joins the cloud. A request to start an application can be made to any of the clouds nodes. FiLeCC supports running two different kinds of applications: a jar-File that will simply be executed or a specially crafted jar-File, which implements an Interface from FiLeCC-Package. In the latter case, the application will be executed in the same Java Virtual Maschine (JVM) as the FiLeCC application. This allows optimized memory management and stricter runtime control. More details are described in chapter 3 and in section 4.2. Applications that can be executed by FiLeCC are called *Apps* in this manual.

2.2 Structure of FiLeCC

Every computer takes part as one node in the cloud environment. They are able to communicate via Web services and therefor offer an interface for DPWS. Web services that are linked to one hardware resource are called *local*, e.g. for controlling attached actors. Web services with a variable end point are called *flexible*, because they have a flexible IP-Address. Flexible Web services always have to be available to every node in the environment, even if the net is split into different parts. To guarantee the availability of flexible Web services, a Watchdog System consisting of one global and multiple local components constantly monitors all nodes in the environment. On each node runs a Local Watchdog and checks whether or not all local processes are still being executed. At regular intervals, it informs the Global Watchdog that it is still alive. When the Global Watchdog does not receive messages from a Local Watchdog anymore, it is an evidence that the corresponding node is overloaded or even has crashed.

In these cases, the Global Watchdog has two options: If the whole node fell out, he sends a message to the system component UNO which then replaces the missing node by replicating all missing Apps onto other active nodes. If on the other hand only the Local Watchdog fell out, the Global Watchdog informs the *Software Deployment and Runtime Management* (SDRM) component to restart the Local Watchdog on the corresponding node.

To avoid the overload of particular nodes, it is necessary to distribute all offered Web services among the hardware resources according to the current processor load on each hardware resource. Therefore, the load must be monitored at any time. For this purpose, every node has a *Local Resource Monitor*. This software component logs the processor load of a node and, in case of overload, advises the UNO to transfer Web services of the current node to a node with lower processor load.

Both UNO and Global Watchdog are each started on a single node. They change their host node only if it is overloaded. In this case, the new nodes hosting the UNO and the Watchdog are determined by a *Leader Election.*

All components are described in detail in section 4.

3 Packaging

To use an App on FiLeCC pack it as a jar-file, if it is a Java application, or as zip-file, if it is another executable file. Within these file a Manifest-File has to exist, which matches the XML schema described in section 3.1. The Manifest-File contains parameters which are needed to handle the App in FiLeCC. One example for such a parameter is the unique identifier, which identifies the App in FiLeCC. The Manifest-File has to be added to the root folder of the App and has to be named the special name, being spezified in the configuration of FiLeCC. It can be set by the administrator by configuring the default value in the configuration file of FiLeCC (see administrator manual for further informations). The default name of the Manifest-File is *filecc.xml*.

3.1 Manifest-File

A file matching the XML schema shown in listing 1 has to be added to each App you want to install to FiLeCC. An example description file is provided in listing 2.

The XML-File is required to contain the elements *id* and *version*. The *id* has to be unique among all deployed Apps in FiLeCC. Otherwise it is treated as another version of an already deployed *App* with the same identifier (see section 4.4). The *version* element is necessary to handle updates. FiLeCC checks whether the version of an installed App is higher or lower then the new one. The App only will be updated if the new version is higher then the old one, otherwise the new App will be ignored.

Furthermore a set of optional parameters can be added to this file. These are *cmdline_arguments*, *executable_filename*, *muticast_address*, *keep_local* and *main_class*.

The parameter *cmdline_arguments* is used if the App does not implement any interfaces of FiLeCC and is started as a new process on a node. The commandline arguments for this operating-system-call can be given in this parameter.

If the App is not packaged into a Java-jar-File, it must be packaged as a zip-File and FiLeCC needs the information which file has to be executed. This information must be given in the *executable_filename* parameter.

The *multicast address* parameter is used, if the App implements a *Leader-Election-Interface* from FiLeCC and stores the needed address for this purpose.

If the App implements an interface of the *AppServer* as explained in section 4.2, the class which implements this interface must be given in the parameter *main_class*.

keep_local is a boolean field that is assumed *false* per default. It is used to determine if the App is only supposed to be deployed and run on a single node either prepackaged with that node or manually installed. In that case the value must be *true* and the *SDRM* skips the deployment mechanisms for this App.

Listing 1: XML schema of the Manifest-File

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://filecc.flw.mb.tu-dortmund.de"
    xmlns="http://filecc.flw.mb.tu-dortmund.de">
    <xs:element name="fileccDescription">
        <xs:complexType >
            <xs:sequence>
                <xs:element name="id" type="xs:string"/>
                <xs:element name="version" type="xs:string"/>
                <xs:element name="cmdline_arguments" type="xs:string" default="" ↩
                    minOccurs="0"/>
                <xs:element name="multicast_address" type="xs:string" default="" ↩
                    minOccurs="0"/>
                <xs:element name="executable_filename" type="xs:string" default="" ↩
                    minOccurs="0"/>
```

```
                <xs:element name="keep_local" type="xs:boolean" default="false" ↩
                    minOccurs="0"/>
                <xs:element name="main_class" type="xs:string" default="" minOccurs="↩
                    0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Listing 2: Example Manifest-File

```
<?xml version="1.0" encoding="UTF-8"?>

<ns1:fileccDescription
        xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
        xsi:schemaLocation='http://filecc.flw.mb.tu-dortmund.de'
        xmlns:ns1='http://filecc.flw.mb.tu-dortmund.de'>
    <id>MyUniqueWebservice</id>
    <version>1.0-r1</version>
    <executable_filename></executable_filename>
</ns1:fileccDescription>
```

4 Single Modules

4.1 log4j – XML-Konfiguration

FileCC uses logging mechanisms to store status data. To ensure a consistent logging method for all services, the FiLeCC application provides a special logger, it is called `filecc.extern`. The configuration of this logger can be found in the log4j.xml configuration file which is deployed with the whole application. To use the logger, the method `Logger.getLogger("filecc.extern")` has to be called. To log messages the method `Logger.getLogger("filecc.extern").error("message")` has to be called.

4.2 AppServer

The AppServer is the main component of FiLeCC that loads and starts Apps implementing the `FiLeCCApplication` or the `MonitoredFiLeCCApplication` interface. These Apps will be executed in the same JVM as the FiLeCC program itself. After loading an App it can be started by the SDRM (see section 4.4). A new thread will be created that executes the `start()` method of the `FiLeCCApplication` interface and the state of the App will be set to `running`. Ordering the SDRM to stop the App will create a new thread that calls the `stop()` method. Inside the `stop()` method the program has to make sure that all dependent threads (including the thread that executes `start()`) will be terminated.

Implementing `MonitoredFiLeCCApplication` instead of `FiLeCCApplication` gives the developer the possibility to implement the special method `alive()`, that will be periodically

called by the Watchdog component of FiLeCC. If the method returns `false`, the App is considered 'dead' and is restarted (if the state of the App was `running`). The most straightforward implementation is to check the Apps' thread state and, if it is `Thread.State.TERMINATED`, return `false`. Note that the method `alive()` is called within a seperate Thread, so calling `Thread.currentThread()` will NOT return the thread executing `start()`. However, saving a reference to the thread in the `start()` method as an object attribute allows the `alive()` method to access that information.

Each App has its own execution context, which means that classes loaded and used by other Apps and FiLeCC itself cannot be accessed. Standard Java Classes (normally residing under the `java` and `javax` package) are loaded by the System Class Loader and shared between all applications and FiLeCC. This may not be critical, because these classes don't contain any global attributes that can be manipulated by a loaded App. Loading classes of the package `edu.udo.mb.flw.filecc` is completely forbidden within a loaded App.

4.3 DB-Interface

Apps that require a database can utilize a GridGain `Grid` called UserGrid. The configuration of this grid is provided in the 'gg_conf_user.xml' file. The file is a GridGain configuration file, whose functionality is described in [Iva12]. For purposes of security and stability, only parts of the configuration are taken into account. In particular, the Cache Configuration is taken 'as is', such as the number and the types of available caches. Additionally, the performance of the database can be tweaked by the following parameters: 'executorService', 'systemExecutorService' and 'peerClassLoadingExecutorService'. Through these parameters, the number of threads in a given thread pool can be set. In this way, the desired balance between database performance and resource utilization can be achieved.

A change in the configuration is a multistep process. First, the desired configuration file needs to be created and tested locally. Then, the final configuration file is to be provided to the Administrator. The Administrator then proceeds by overwriting the existing configuration files with the new ones. As a last step, the method `reconfigureUserGrid()` in `GridGainNodeApp` needs to be called programmatically, which stops the running UserGrid and starts it with the desired configuration. The last step is performed by the developer, who calls the method from within their software, so as to ensure that the latest configuration is in effect.

A reference to the running `Grid` can be obtained from the method `getUserGrid()` in `edu.udo.mb.flw.fileccapp.UserDatabaseFactory`. The exposed functionality depends on the active configuration and is not limited by FiLeCC. The applications can use all types of GridGain data structures for example `GridCache`.

4.4 SDRM

The *Software Deployment and Runtime Management* component (SDRM) manages the installation, deinstallation and update of Apps (see subsections 4.4.1 and 4.4.2). Furthermore it is used

to start or stop Apps (see subsection 4.4.3).

4.4.1 Add and remove Apps to and from FiLeCC

To use Apps at the FiLeCC system they need to be installed. There are two possibilities to install Apps. The first option is to use the *InstallWebService* operation belonging to the *AdministrationService* of each *SDRMDevice*. A use of this operation can be found in the Admin Panel (see administrator manual for further informations). The second option is to copy the file directly into the welcome directory of a node, which is configured in the global FiLeCC configuration (see administration manual).

To remove an App from FiLeCC you have to call the *DeinstallWebService* operation belonging to the *AdministrationService* of each *SDRMDevice* (also available in the Admin Panel). The App will be stopped if it is running and its executables removed from every node in FiLeCC.

4.4.2 Update Apps in FiLeCC

Updating Apps works similar to installing new ones. The only difference is, that the version number in the accompanying Manifest-File needs to be higher the currently installed. The FiLeCC system will notice and handle the update automatically.

4.4.3 Start and stop Apps in FiLeCC

To start or stop an App in FiLeCC, the *StartStopService* of the *SDRMDevice* is used. The *startService*-Operation requires the unique identifier of an App as a parameter and will start this App on its node. The identifier is declared in the Manifest-File of the App (see section 3.1). To stop an App on one node the *stopService* operation needs to be called with the unique identifier.

Both starting and stopping can be done manually from the Admin Panel.

4.5 Client API

The FiLeCC framework is constructed to utilize the ressources of huge numbers of seperate devices with varying performance capabilities. These devices tend to be embedded systems and will often provide lower computing performance than modern computers do. The power of a cloud based on FiLeCC relies on the sheer number of available devices which renders the problem of assigning a task to a specific device non-trivial. In FiLeCC, tasks are represented by Apps, which in turn are based on webservices so the problem is essentially reduced to distributing all available webservices equally among all nodes. Some of these Apps may only be used occasionally while others are constantly in use, possibly from various clients at the same time. To accomodate for such load discrepancies, the FiLeCC-middleware automatically replicates heavily used Apps to spread the load of these tasks over several nodes. In some cases with volatile workloads, the

active locations of Apps as well as the number of Apps of the same kind may change rapidly according to the load distribution system.

One purpose of a cloud framework is to conceal the implementation details of the underlying system. To that end, a simple client library is provided, which acts as a mediator between a client and the dynamic App landscape described above. The full API specification can be seen in the class diagramm in Figure 1. A client may simply issue a number of search criteria and is returned a single service representation which hides all load balancing details behind an easy to use facade. The following sections describe briefly the basic concepts of this client api. How to search Apps is explained in subsection 4.5.1 and subsection 4.5.2 describes, how to use them once found.

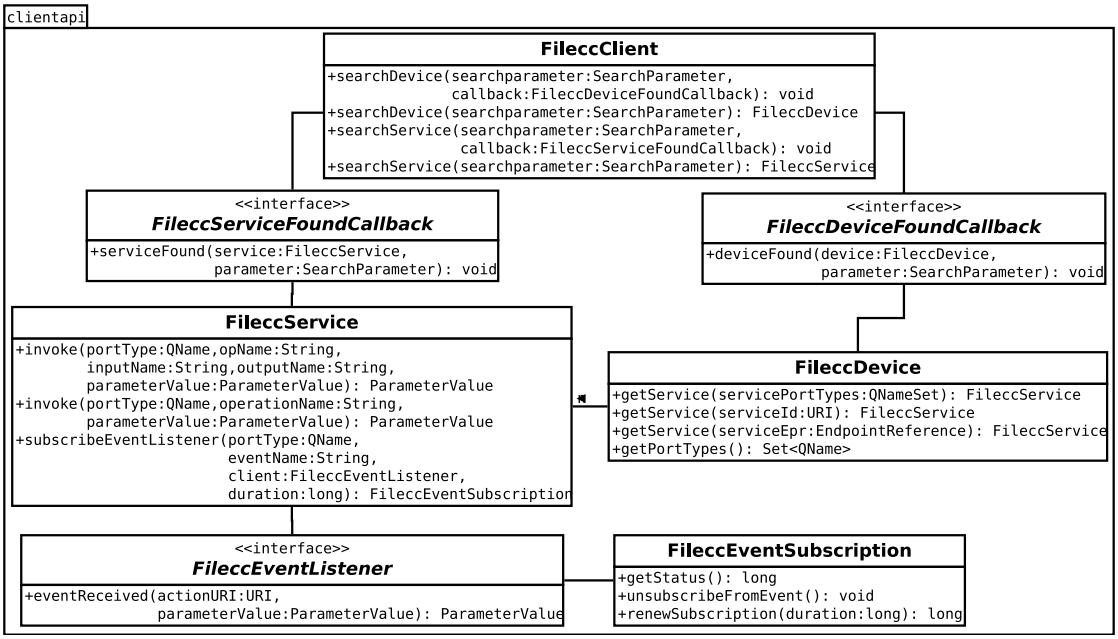


Figure 1: Client-API classes

4.5.1 Finding a service

The first step in using an App inside FiLeCC is the lookup. The central class for this task is the `FileccClient` class, which should only be instantiated once per application to maximize the utilization of internal caching mechanisms of the client-API. See the `FileccClient` class in the class diagramm in Figure 1 for details. Looking up devices and services is conceptually equivalent, so the following descriptions are restricted to services.

To initiate a search for an App, you have to specify a `SearchParameter` object defining the kind of App to search for. This process is described in detail in [fDW11].

There are two ways to start an App lookup, synchronous and asynchronous. The asynchronous lookup method expects a callback interface as a parameter and does not block, so the client may

work while waiting for the lookup to complete. The synchronized lookup on the other hand returns immediately as well, this time with a representation of an App. The difference being, that this representation may not be fully initialized yet. The initialization is completed by a background task while the service lookup is still in progress. Any call to this service will be rejected as long as no matching service has been found.

4.5.2 Using a service

After a service lookup has been issued, the client has received a `FileccService` object either directly via a `FileccClient` object or asynchronously using a `FileccServiceFoundCallback` interface (see Figure 1). Either way, this new `FileccService` object will act as a mediator between the client and any number of services matching the search criteria used to create this object. All invokations of operations through this service facade will be spread equally among all known Apps implementing this operation using an internal cache of Apps. To call an operation use the `FileccService` objects invoke-method specifying the operation either by portType and operation name or by its full qualifier. According to the dpws specification, an Operation is uniquily identified by a quadruple consisting of its port type, operation name, input name and output name. The input parameters of the operation are specified with a `ParameterValue` object from the jmeds-api [fDW11]. Similary the output value of any operation is returned as another `ParameterValue` object. Lastly, any DPWS compatible App may offer a number of event sources to which a client may subscribe. This can be accomplished with a single call to the `subscribeEventListener` method of a `FileccService` providing the port type and name of the event as well as a `FileccEventListener` interface, which is called whenever an event is received.

5 Licenses and Commercialisation

5.1 Licensing

Licence: *GNU Lesser General Public License* (LGPL), Version 3, 29th June 2007. LGPL is a license for free software and was published by the Free Software Foundation (FSF). A software under the LGPL can be used for any purpose. It can be changed, modified and republished. For details see ([Sys12]).

Special thanks to Chair of Materials Handling and Warehousing at TU Dortmund.
Webpage: ([Sad12]).

References

[fDW11] Web Services for Devices (WS4D). Ws4d - jmeds api (java multi edition dpws stack), accessed 11.12.2011. `http://ws4d-javame.sourceforge.net/api/`.

[Iva12] Nikita Ivanov. *Real Time Big Data Processing with GridGain*, 2012. http://www.gridgain.com/book/book.html.

[Sad12] Dr.-Ing. Volker Sadowsky. Lehrstuhl für Förder- und Lagerwesen. online, 2012. http://www.flw.mb.tu-dortmund.de.

[Sys12] The GNU Operating System. GNU Lesser General Public License v3.0 - GNU Project - Free Software Foundation (FSF). online, 2012. http://www.gnu.org/licenses/lgpl.html.

ANHANG D

Quelltext Testprogramme

D.1 MemoryTest

```
1  package pg557_testapp;
2
3  import java.util.LinkedList;
4  import java.util.List;
5
6  public class MemoryTest {
7
8      public void run() {
9          List<String> list = new LinkedList<String>();
10
11         for (int i = 0; i < 100; i++) {
12             for (int j = 0; j < 2000000; j++) {
13                 String s = "↩
                       sdafkjws09trjsiodfjiosjdfsdafkjws09trjsiodfjiosjdfsdafkjws09trjsiodfjiosj
                       ";
14                 list.add(s);
15             }
16
17             for (int j = 0; j < 2000000; j++) {
18                 list.remove(0);
19             }
20         }
21     }
22
23     public static void main(String[] args) {
24         long startTime = System.currentTimeMillis();
25
26         MemoryTest test = new MemoryTest();
```

```
27          test.run();
28
29          System.out.println("done in " + (System.currentTimeMillis() - ↩
                startTime));
30      }
31  }
```

D.2 PrimeTest

```
1   package pg557_testapp;
2
3   public class PrimeTest {
4
5       public static boolean isPrime(int number) {
6           if (number < 2) {
7               return false;
8           } else if (number == 2) {
9               return true;
10          } else {
11              if (number % 2 == 0) {
12                  return false;
13              }
14
15              for (int i = 3; i <= Math.sqrt(number); i = i + 2) {
16                  if (number % i == 0) {
17                      return false;
18                  }
19              }
20              return true;
21          }
22      }
23
24      public static void main(String[] args) {
25          if (args.length == 0) {
26              System.out.println("usage: <maxNumber> - find prime number ↩
                    between 2 and <maxNumber>");
27              return;
28          }
29
30          long startTime = System.currentTimeMillis();
31          int primes = 0;
32          for (int z = 1; z <= Integer.parseInt(args[0]); z++) {
33              if (isPrime(z)) {
34                  primes++;
35              }
36          }
37          System.out.println(primes + " prime numbers found (2-" + args[0] + ↩
                ") in " + (System.currentTimeMillis() - startTime) + "ms.");
38      }
39  }
```

D.3 CompressionTest

```
1  package pg557_testapp;
2
3  import java.io.File;
4  import java.io.FileInputStream;
5  import java.io.FileNotFoundException;
6  import java.io.FileOutputStream;
7  import java.io.IOException;
8  import java.io.InputStream;
9  import java.io.OutputStream;
10 import java.util.logging.Level;
11 import java.util.logging.Logger;
12 import java.util.zip.GZIPInputStream;
13 import java.util.zip.GZIPOutputStream;
14
15 public class CompressionTest {
16
17     public void run(String fileName) {
18         long startTime = System.currentTimeMillis();
19
20         try {
21             OutputStream fos = new FileOutputStream(fileName + ".gz");
22             OutputStream gzipOut = new GZIPOutputStream(fos);
23
24             InputStream fis = new FileInputStream(fileName);
25             int nextChar;
26             while ((nextChar = fis.read()) != -1) {
27                 gzipOut.write(nextChar);
28             }
29
30             gzipOut.close();
31             fos.close();
32         } catch (FileNotFoundException ex) {
33             Logger.getLogger(CompressionTest.class.getName()).log(Level.↩
                   SEVERE, null, ex);
34         } catch (IOException ex) {
35             Logger.getLogger(CompressionTest.class.getName()).log(Level.↩
                   SEVERE, null, ex);
36         }
37
38
39         try {
40             InputStream fis = new FileInputStream(fileName + ".gz");
41             InputStream gzipin = new GZIPInputStream(fis);
42
43             InputStream fis2 = new FileInputStream(fileName);
44
45             int nextChar;
46             while ((nextChar = gzipin.read()) != -1) {
47                 if (nextChar != fis2.read()) {
48                     System.out.println("error");
49                 }
50             }
51
52             fis2.close();
53             gzipin.close();
```

```
54              fis.close();
55
56              new File(fileName + ".gz").delete();
57          } catch (FileNotFoundException ex) {
58              Logger.getLogger(CompressionTest.class.getName()).log(Level.↩
                    SEVERE, null, ex);
59          } catch (IOException ex) {
60              Logger.getLogger(CompressionTest.class.getName()).log(Level.↩
                    SEVERE, null, ex);
61          }
62
63          System.out.println("done in " + (System.currentTimeMillis() - ↩
                startTime));
64      }
65
66      public static void main(String[] args) {
67          if (args.length == 0) {
68              System.out.println("usage: <filename>");
69              return;
70          }
71
72          CompressionTest test = new CompressionTest();
73          test.run(args[0]);
74      }
75  }
```

D.4 DijkstraTest

```
1   package pg557_testapp;
2
3   import edu.uci.ics.jung.algorithms.shortestpath.DijkstraDistance;
4   import edu.uci.ics.jung.algorithms.shortestpath.DijkstraShortestPath;
5   import edu.uci.ics.jung.graph.Graph;
6   import edu.uci.ics.jung.graph.SparseMultigraph;
7   import java.util.Random;
8
9   public class DijkstraTest {
10
11      public void run(int xNodes, int yNodes, int dijstraCount) {
12          Graph<String, String> graph = new SparseMultigraph<String, String↩
                >();
13          for (int x = 0; x < xNodes; x++) {
14              for (int y = 0; y < yNodes; y++) {
15                  graph.addVertex("V" + x + "." + y);
16              }
17          }
18
19          for (int x = 0; x < xNodes; x++) {
20              for (int y = 0; y < yNodes; y++) {
21                  if (x < 9) {
22                      graph.addEdge("E" + x + "." + y + "->" + (x + 1) + "." ↩
                            + y, "V" + x + "." + y, "V" + (x + 1) + "." + y);
23                  }
24                  if (x > 0) {
```

```
                    graph.addEdge("E" + x + "." + y + "->" + (x - 1) + "." + y, "V" + x + "." + y, "V" + (x - 1) + "." + y);
                }

                if (y < 9) {
                    graph.addEdge("E" + x + "." + y + "->" + x + "." + (y + 1), "V" + x + "." + y, "V" + x + "." + (y + 1));
                }
                if (y > 0) {
                    graph.addEdge("E" + x + "." + y + "->" + x + "." + (y - 1), "V" + x + "." + y, "V" + x + "." + (y - 1));
                }
            }
        }

        DijkstraShortestPath<String, String> dijkstra = new DijkstraShortestPath(graph);
        Random r = new Random();

        for (int i=0; i<dijstraCount; i++) {
            dijkstra.getPath("V" + (Math.abs(r.nextInt())%xNodes) + "." + (Math.abs(r.nextInt())%yNodes), "V" + (Math.abs(r.nextInt())%xNodes) + "." + (Math.abs(r.nextInt())%yNodes));
        }
    }

    public static void main(String[] args) {

        if (args.length < 3) {
            System.out.println("usage: <xNodes> <yNodes> <dijstra calculations>");
            return;
        }

        long startTime = System.currentTimeMillis();

        DijkstraTest test = new DijkstraTest();
        test.run(Integer.parseInt(args[0]), Integer.parseInt(args[1]), Integer.parseInt(args[2]));

        System.out.println("time: " + (System.currentTimeMillis() - startTime) + "ms");
    }
}
```

D.5 Whetstone

```

public class Whetstone {

    static long begin_time,
            end_time,
            total_time;
    /*
```

```
8       * Whetstone benchmark in C. This program is a translation of the
9       * original Algol version in "A Synthetic Benchmark" by H.J. Curnow
10      * and B.A. Wichman in Computer Journal, Vol 19 #1, February 1976.
11      Convert into java and automatic iterations by Tarquin Mills from C, by anon from Unix Hensa
12      * Used to test compiler optimization and floating point performance.
13      *
14      * Compile by: cc -O -s -o whet whet.c
15      * or: cc -O -DPOUT -s -o whet whet.c
16      * if output is desired.
17      */
18      static int ITERATIONS; /* ITERATIONS/10 = Millions Whetstone instructions */
19
20      static int numberOfCycles;
21      static int cycleNo;
22      static double x1, x2, x3, x4, x, y, z[] = new double[1], t, t1, t2;
23      static double e1[] = new double[4];
24      static int i, j, k, l, n1, n2, n3, n4, n6, n7, n8, n9, n10, n11;
25
26      public static double mainCalc() { /* initialize constants */
27          /*System.out.println("Start");*/
28          t = 0.499975;
29          t1 = 0.50025;
30          t2 = 2.0;
31          /* set values of module weights */
32          n1 = 0 * ITERATIONS;
33          n2 = 12 * ITERATIONS;
34          n3 = 14 * ITERATIONS;
35          n4 = 345 * ITERATIONS;
36          n6 = 210 * ITERATIONS;
37          n7 = 32 * ITERATIONS;
38          n8 = 899 * ITERATIONS;
39          n9 = 616 * ITERATIONS;
40          n10 = 0 * ITERATIONS;
41          n11 = 93 * ITERATIONS;
42          begin_time = System.currentTimeMillis();
43          for (cycleNo = 1; cycleNo <= numberOfCycles; cycleNo++) {
44              /* MODULE 1: simple identifiers */
45              x1 = 1.0;
46              x2 = x3 = x4 = -1.0;
47              for (i = 1; i <= n1; i += 1) {
48                  x1 = (x1 + x2 + x3 - x4) * t;
49                  x2 = (x1 + x2 - x3 + x4) * t; // correction: x2 = ( x1 + x2 - x3 - x4 ) * t;
50                  x3 = (x1 - x2 + x3 + x4) * t; // correction: x3 = ( x1 - x2 + x3 + x4 ) * t;
51                  x4 = (-x1 + x2 + x3 + x4) * t;
52              }
53              // if (cycleNo==numberOfCycles) pout(n1, n1, n1, x1, x2, x3, x4);
54
55              /* MODULE 2: array elements */
56              e1[0] = 1.0;
57              e1[1] = e1[2] = e1[3] = -1.0;
58              for (i = 1; i <= n2; i += 1) {
59                  e1[0] = (e1[0] + e1[1] + e1[2] - e1[3]) * t;
60                  e1[1] = (e1[0] + e1[1] - e1[2] + e1[3]) * t;
61                  e1[2] = (e1[0] - e1[1] + e1[2] + e1[3]) * t;
```

```
62                  e1[3] = (-e1[0] + e1[1] + e1[2] + e1[3]) * t;
63              }
64              // if (cycleNo==numberOfCycles) pout(n2, n3, n2, e1[0], e1[1], ↩
                    e1[2], e1[3]);
65              /* MODULE 3: array as parameter */
66              for (i = 1; i <= n3; i += 1) {
67                  pa(e1);
68              }
69              // if (cycleNo==numberOfCycles) pout(n3, n2, n2, e1[0], e1[1], ↩
                    e1[2], e1[3]);
70              /* MODULE 4: conditional jumps */
71              j = 1;
72              for (i = 1; i <= n4; i += 1) {
73                  if (j == 1) {
74                      j = 2;
75                  } else {
76                      j = 3;
77                  }
78                  if (j > 2) {
79                      j = 0;
80                  } else {
81                      j = 1;
82                  }
83                  if (j < 1) {
84                      j = 1;
85                  } else {
86                      j = 0;
87                  }
88              }
89              // if (cycleNo==numberOfCycles) pout(n4, j, j, x1, x2, x3, x4);
90              /* MODULE 5: omitted */
91              /* MODULE 6: integer arithmetic */
92              j = 1;
93              k = 2;
94              l = 3;
95              for (i = 1; i <= n6; i += 1) {
96                  j = j * (k - j) * (l - k);
97                  k = l * k - (l - j) * k;
98                  l = (l - k) * (k + j);
99                  e1[l - 2] = j + k + l; /* C arrays are zero based */
100                 e1[k - 2] = j * k * l;
101             }
102             // if (cycleNo==numberOfCycles) pout(n6, j, k, e1[0], e1[1], e1↩
                    [2], e1[3]);
103             /* MODULE 7: trig. functions */
104             x = y = 0.5;
105             for (i = 1; i <= n7; i += 1) {
106                 x = t * Math.atan(t2 * Math.sin(x) * Math.cos(x) / (Math.↩
                        cos(x + y) + Math.cos(x - y) - 1.0));
107                 y = t * Math.atan(t2 * Math.sin(y) * Math.cos(y) / (Math.↩
                        cos(x + y) + Math.cos(x - y) - 1.0));
108             }
109             // if (cycleNo==numberOfCycles) pout(n7, j, k, x, x, y, y);
110
111             /* MODULE 8: procedure calls */
112             x = y = z[0] = 1.0;
113             for (i = 1; i <= n8; i += 1) {
114                 p3(x, y, z);
115             }
```

```
116             // if (cycleNo==numberOfCycles) pout(n8, j, k, x, y, z[0], z↩
                    [0]);
117             /* MODULE9: array references */
118             j = 0;
119             k = 1;
120             l = 2;
121             e1[0] = 1.0;
122             e1[1] = 2.0;
123             e1[2] = 3.0;
124             for (i = 1; i <= n9; i++) {
125                 p0();
126             }
127             // if (cycleNo==numberOfCycles) pout(n9, j, k, e1[0], e1[1], e1↩
                    [2], e1[3]);
128             /* MODULE10: integer arithmetic */
129             j = 2;
130             k = 3;
131             for (i = 1; i <= n10; i += 1) {
132                 j = j + k;
133                 k = j + k;
134                 j = k - j;
135                 k = k - j - j;
136             }
137             // if (cycleNo==numberOfCycles) pout(n10, j, k, x1, x2, x3, x4)↩
                    ;
138             /* MODULE11: standard functions */
139             x = 0.75;
140             for (i = 1; i <= n11; i += 1) {
141                 x = Math.sqrt(Math.exp(Math.log(x) / t1));
142             }
143             // if (cycleNo==numberOfCycles) pout(n11, j, k, x, x, x, x);
144         } /* for */
145         end_time = System.currentTimeMillis();
146         System.out.println(" (time for " + numberOfCycles + " cycles): "
147                 + (end_time - begin_time) + " millisec.");
148         return (end_time - begin_time);
149     } /* Main */
150 
151 
152     public static void pa(double e[]) {
153         int j;
154         j = 0;
155         do {
156             e[0] = (e[0] + e[1] + e[2] - e[3]) * t;
157             e[1] = (e[0] + e[1] - e[2] + e[3]) * t;
158             e[2] = (e[0] - e[1] + e[2] + e[3]) * t;
159             e[3] = (-e[0] + e[1] + e[2] + e[3]) / t2;
160             j += 1;
161         } while (j < 6);
162     }
163 
164     public static void p3(double x, double y, double z[]) {
165         x = t * (x + y);
166         y = t * (x + y);
167         z[0] = (x + y) / t2;
168     }
169 
170     public static void p0() {
171         e1[j] = e1[k];
```

```
172         e1[k] = e1[l];
173         e1[l] = e1[j];
174     }
175     // public static void pout(int n, int j, int k, double x1, double x2, ↩
            double x3, double x4)
176     // {
177     // System.out.println(n+"\t"+j+"\t"+k+"\t"+x1+"\t"+x2+"\t"+x3+"\t"+x4);
178     // }
179
180     public static void main(String[] args) {
181         System.out.println("WHETSTONE - using 64-bit floating-point data");
182         ITERATIONS = 100; /* ITERATIONS/10 = Millions Whetstone ↩
                instructions */
183         numberOfCycles = 100;
184         int numberOfRuns = 10;
185         float elapsedTime = 0;
186         float meanTime = 0;
187         float rating = 0;
188         float meanRating = 0;
189         int intRating = 0;
190         for (int runNumber = 1; runNumber <= numberOfRuns; runNumber++) {
191             System.out.print(runNumber + ". Test");
192             // Call the Whetstone benchmark procedure
193             // compute elapsed time
194             elapsedTime = (float) (mainCalc() / 1000);
195             // sum time in milliseconds per cycle
196             meanTime = meanTime + (elapsedTime * 1000 / numberOfCycles);
197             // Calculate the Whetstone rating based on the time for
198             // the numbers of cycles just executed
199             rating = (1000 * numberOfCycles) / elapsedTime;
200             // Sum Whetstone rating
201             meanRating = meanRating + rating;
202             intRating = (int) rating;
203             // Reset no_of_cycles for the next run using ten cycles more
204             numberOfCycles += 10;
205         }
206         meanTime = meanTime / numberOfRuns;
207         meanRating = meanRating / numberOfRuns;
208         intRating = (int) meanRating;
209         System.out.println("Number of Runs " + numberOfRuns);
210         System.out.println("Average time per cycle " + meanTime + " ↩
                millisec.");
211         System.out.println("Average Whetstone Rating " + intRating + " ↩
                KWIPS");
212     }
213 } /* benchmarks */
```

D.6 Dhrystone

```
1 /*
2  * Dr. Reinhold P. Weicker. Dhrystone benchmark: Rationale for
3  * version 2 and measurement rules. SIGPLAN Notices 23,8 (Aug.
4  * 1988), [49-62], 1988.
5  */
```

```
6
7  public class Dhrystone extends GlobalVariables {
8
9      static long numberOfRuns = 10;
10     static long numberOfLoops = 100000;
11
12     public static double execute() {
13         int Int_Loc_1,
14                 Int_Loc_2,
15                 Int_Loc_3;
16         int[] Int_Loc_3_Ref = new int[1];
17         int[] Int_Loc_1_Ref = new int[1];
18         char Char_Index;
19         int[] Enum_Loc = new int[1];
20         String String_Loc_1,
21                 String_Loc_2;
22         long begin_time,
23                 end_time,
24                 total_time;
25         int Run_Index,
26                 Meas_Index;
27         Next_Record_Glob = Second_Record;
28         Record_Glob = First_Record;
29         Record_Glob.Record_Comp = Next_Record_Glob;
30         Record_Glob.Discr = Ident_1;
31         Record_Glob.Enum_Comp = Ident_3;
32         Record_Glob.Int_Comp = 40;
33         Record_Glob.String_Comp = "DHRYSTONE PROGRAM, SOME STRING";
34         String_Loc_1 = "DHRYSTONE PROGRAM, 1'ST STRING";
35         begin_time = System.currentTimeMillis();
36         for (Run_Index = 1; Run_Index <= numberOfLoops; ++Run_Index) {
37             Proc_5();
38             Proc_4();
39             Int_Loc_1 = 2;
40             Int_Loc_2 = 3;
41             String_Loc_2 = "DHRYSTONE PROGRAM, 2'ND STRING";
42             Enum_Loc[0] = Ident_2;
43             Bool_Glob = !Func_2(String_Loc_1, String_Loc_2);
44             while (Int_Loc_1 < Int_Loc_2) {
45                 Int_Loc_3_Ref[0] = 5 * Int_Loc_1 - Int_Loc_2;
46                 Proc_7(Int_Loc_1, Int_Loc_2, Int_Loc_3_Ref);
47                 Int_Loc_1 += 1;
48             }
49             Int_Loc_3 = Int_Loc_3_Ref[0];
50             Proc_8(Array_Glob_1, Array_Glob_2, Int_Loc_1, Int_Loc_3);
51             Proc_1(Record_Glob);
52             for (Char_Index = 'A'; Char_Index <= Char_Glob_2; ++Char_Index)↩
                    {
53                 if (Enum_Loc[0] == Func_1(Char_Index, 'C')) {
54                     Proc_6(Ident_1, Enum_Loc);
55                 }
56             }
57             Int_Loc_3 = Int_Loc_2 * Int_Loc_1;
58             Int_Loc_2 = Int_Loc_3 / Int_Loc_1;
59             Int_Loc_2 = 7 * (Int_Loc_3 - Int_Loc_2) - Int_Loc_1;
60             Int_Loc_1_Ref[0] = Int_Loc_1;
61             Proc_2(Int_Loc_1_Ref);
62             Int_Loc_1 = Int_Loc_1_Ref[0];
63         }
```

```
64          end_time = System.currentTimeMillis();
65          total_time = end_time - begin_time;
66          System.out.println(" (time for " + numberOfLoops + "): " + ↩
                total_time + " millisec.");
67          return (total_time);
68      }
69
70      static void Proc_1(Record_Type Pointer_Par_Val) {
71          Record_Type Next_Record = Pointer_Par_Val.Record_Comp;
72          Pointer_Par_Val.Record_Comp = Record_Glob;
73          Pointer_Par_Val.Int_Comp = 5;
74          Next_Record.Int_Comp = Pointer_Par_Val.Int_Comp;
75          Next_Record.Record_Comp = Pointer_Par_Val.Record_Comp;
76          Proc_3(Next_Record.Record_Comp);
77          if (Next_Record.Discr == Ident_1) {
78              Next_Record.Int_Comp = 6;
79              Int_Ref[0] = Next_Record.Enum_Comp;
80              Proc_6(Pointer_Par_Val.Enum_Comp, Int_Ref);
81              Next_Record.Enum_Comp = Int_Ref[0];
82              Next_Record.Record_Comp = Record_Glob.Record_Comp;
83              Int_Ref[0] = Next_Record.Int_Comp;
84              Proc_7(Next_Record.Int_Comp, 10, Int_Ref);
85              Next_Record.Int_Comp = Int_Ref[0];
86          } else {
87              Pointer_Par_Val = Pointer_Par_Val.Record_Comp;
88          }
89      }
90
91      static void Proc_2(int Int_Par_Ref[]) {
92          int Int_Loc;
93          int Enum_Loc;
94          Int_Loc = Int_Par_Ref[0] + 10;
95          Enum_Loc = 0;
96          do {
97              if (Char_Glob_1 == 'A') {
98                  Int_Loc -= 1;
99                  Int_Par_Ref[0] = Int_Loc - Int_Glob;
100                 Enum_Loc = Ident_1;
101             }
102         } while (Enum_Loc != Ident_1);
103     }
104
105     static void Proc_3(Record_Type Pointer_Par_Ref) {
106         if (Record_Glob != null) {
107             Pointer_Par_Ref = Record_Glob.Record_Comp;
108         } else {
109             Int_Glob = 100;
110         }
111         Int_Comp_Ref[0] = Record_Glob.Int_Comp;
112         Proc_7(10, Int_Glob, Int_Comp_Ref);
113         Record_Glob.Int_Comp = Int_Comp_Ref[0];
114     }
115
116     static void Proc_4() {
117         boolean Bool_Loc;
118         Bool_Loc = Char_Glob_1 == 'A';
119         Bool_Loc = Bool_Loc || Bool_Glob;
120         Char_Glob_2 = 'B';
121     }
```

```
122
123     static void Proc_5() {
124         Char_Glob_1 = 'A';
125         Bool_Glob = false;
126     }
127
128     static void Proc_6(int Enum_Par_Val, int Enum_Par_Ref[]) {
129         Enum_Par_Ref[0] = Enum_Par_Val;
130         if (!Func_3(Enum_Par_Val)) {
131             Enum_Par_Ref[0] = Ident_4;
132         }
133         switch (Enum_Par_Val) {
134             case Ident_1:
135                 Enum_Par_Ref[0] = Ident_1;
136                 break;
137             case Ident_2:
138                 if (Int_Glob > 100) {
139                     Enum_Par_Ref[0] = Ident_1;
140                 } else {
141                     Enum_Par_Ref[0] = Ident_4;
142                 }
143                 break;
144             case Ident_3:
145                 Enum_Par_Ref[0] = Ident_2;
146                 break;
147             case Ident_4:
148                 break;
149             case Ident_5:
150                 Enum_Par_Ref[0] = Ident_3;
151                 break;
152         }
153     }
154
155     static void Proc_7(int Int_Par_Val1, int Int_Par_Val2, int Int_Par_Ref↩
        []) {
156         int Int_Loc;
157         Int_Loc = Int_Par_Val1 + 2;
158         Int_Par_Ref[0] = Int_Par_Val2 + Int_Loc;
159     }
160
161     static void Proc_8(int[] Array_Par_1_Ref, int[][] Array_Par_2_Ref, int ↩
        Int_Par_Val_1, int Int_Par_Val_2) {
162         int Int_Index,
163                 Int_Loc;
164         Int_Loc = Int_Par_Val_1 + 5;
165         Array_Par_1_Ref[Int_Loc] = Int_Par_Val_2;
166         Array_Par_1_Ref[Int_Loc + 1] = Array_Par_1_Ref[Int_Loc];
167         Array_Par_1_Ref[Int_Loc + 30] = Int_Loc;
168         for (Int_Index = Int_Loc; Int_Index <= Int_Loc + 1; ++Int_Index) {
169             Array_Par_2_Ref[Int_Loc][Int_Index] = Int_Loc;
170         }
171         Array_Par_2_Ref[Int_Loc][Int_Loc - 1] += 1;
172         Array_Par_2_Ref[Int_Loc + 20][Int_Loc] = Array_Par_1_Ref[Int_Loc];
173         Int_Glob = 5;
174     }
175
176     static int Func_1(char Char_Par_1_Val, char Char_Par_2_Val) {
177         char Char_Loc_1,
178                 Char_Loc_2;
```

```
179         Char_Loc_1 = Char_Par_1_Val;
180         Char_Loc_2 = Char_Loc_1;
181         if (Char_Loc_2 != Char_Par_2_Val) {
182             return Ident_1;
183         } else {
184             return Ident_2;
185         }
186     }
187 
188     static boolean Func_2(String String_Par_1_Ref, String String_Par_2_Ref)↩
         {
189         int Int_Loc;
190         char Char_Loc = '\0';
191         Int_Loc = 2;
192         while (Int_Loc <= 2) {
193             if (Func_1(String_Par_1_Ref.charAt(Int_Loc), String_Par_2_Ref.↩
                 charAt(Int_Loc + 1)) == Ident_1) {
194                 Char_Loc = 'A';
195                 Int_Loc += 1;
196             }
197         }
198         if (Char_Loc >= 'W' && Char_Loc < 'Z') {
199             Int_Loc = 7;
200         }
201         if (Char_Loc == 'X') {
202             return true;
203         } else {
204             if (String_Par_1_Ref.compareTo(String_Par_2_Ref) > 0) {
205                 Int_Loc += 7;
206                 return true;
207             } else {
208                 return false;
209             }
210         }
211     }
212 
213     static boolean Func_3(int Enum_Par_Val) {
214         int Enum_Loc;
215         Enum_Loc = Enum_Par_Val;
216         if (Enum_Loc == Ident_3) {
217             return true;
218         } else {
219             return false;
220         }
221     }
222 
223     public static void main(String argv[]) {
224         //Msg.out = System.err;
225         int i;
226         double mean_time = 0;
227         for (i = 1; i <= numberOfRuns; i++) {
228             System.out.print(i + ". Test");
229             mean_time = mean_time + execute();
230         }
231         System.out.println("\nAverage Time over " + numberOfRuns + " runs: ↩
             "
232                 + (mean_time / numberOfRuns) + " millisec.");
233     }
234 }
```

```
235
236 class DhrystoneConstants {
237
238     public static final int Ident_1 = 0;
239     public static final int Ident_2 = 1;
240     public static final int Ident_3 = 2;
241     public static final int Ident_4 = 3;
242     public static final int Ident_5 = 4;
243 }
244
245 class GlobalVariables extends DhrystoneConstants {
246
247     static Record_Type Record_Glob,
248             Next_Record_Glob;
249     static int Int_Glob;
250     static boolean Bool_Glob;
251     static char Char_Glob_1,
252             Char_Glob_2;
253     static int[] Array_Glob_1 = new int[128];
254     static int[][] Array_Glob_2 = new int[128][128];
255     static Record_Type First_Record = new Record_Type(),
256             Second_Record = new Record_Type();
257     static int[] Int_Comp_Ref = new int[1];
258     static int[] Int_Ref = new int[1];
259 }
260
261 class Record_Type {
262
263     Record_Type Record_Comp;
264     int Discr;
265     int Enum_Comp;
266     int Int_Comp;
267     String String_Comp;
268     int Enum_Comp_2;
269     String String_Comp_2;
270     char Char_Comp_1;
271     char Char_Comp_2;
272 }
```