UNIVERSITÄT DORTMUND

Projektgruppe 475

Management kooperierender Web Services

Roasted Kit

Realtime Service-Orientated Architecture Enthusiastically Developed

Stand Mai 2006

Betreuer Prof. Dr. H. Krumm, Prof. Dr. P. Herrmann , A. Pohl

Projektmitglieder Bastian Brill, Müjdat Cagliyan, Sascha Feldhorst, Martin Fiedler, Jörn Gaeb, Michael Heinemann, Töresin Karakoyun, Jan Konieczny, Tobias Mayer, Holger Niehoff, Tobias Peper, Lars Rüsing

Universität Dortmund
Fachbereich Informatik
LS 4 - Rechnernetze und vert. Systeme
Prof. Dr. H. Krumm

Universität Dortmund
Fachbereich Maschinenbau
Lehrstuhl für Förder- und Lagerwesen
Prof. Dr. M. ten Hompel

Autorenverzeichnis

Kapitel	Autor	Überarbeitet von
0 - Titelseite	T. Mayer	-
1 - Einleitung[1]	T. Mayer	S. Feldhorst
2 - Projektbeschreibung	L. Rüsing, J. Koniezcny	B. Brill
3 - Web Services	J. Gaeb	B. Brill, S. Feldhorst
4 - Anlage des Lehrstuhls FLW[1]	L. Rüsing	S. Feldhorst
5 - Eingesetzte Software & Werkzeuge[2]	M. Cagliyan	B. Brill
6 - Anforderungen an den Prototypen[1]	J. Koniezcny	S.Feldhorst
7 - Anlagen-Modell	S. Feldhorst	S. Feldhorst
8 - Architektur-Übersicht[1]	T. Mayer	S.Feldhorst
9 - Einzelne Subkomponenten	-	-
9.1 - UPnP[1]	M. Cagliyan	B. Brill
9.2 - Eventing	T. Peper	T. Mayer
9.3- StatusControl	B. Brill	T. Mayer
9.5 - ServiceControl	B. Brill	T. Mayer
9.4 - TypeLibrary[2]	J. Gaeb	T. Mayer
9.6 - CommonLog	J. Gaeb	T. Mayer
9.7 - Monitoring[1]	J. Koniezcny	T. Mayer
9.8 - Supervisor	T. Mayer	B. Brill
9.9 - OrderManagement	B. Brill	M. Fiedler
9.10 - PacketManagement	H. Niehoff	M. Fiedler
9.11 - SystemManagement[2]	B. Brill	M. Fiedler
9.12 - Interpolation	T. Peper	M. Fiedler
9.13 - DeviceManagement[2]	L. Rüsing	M. Fiedler
10 - Testphase, Installation & Erprobung	J. Koniezcny	B. Brill
11 - Schluss	T. Mayer	B. Brill
A - Anhang A	L. Rüsing	-
B - Anhang B[3]	*je nach Kapitel*	T. Mayer
C - Anhang C	S.Feldhorst	-

[1] Dieses Kapitel wurde durch den Überarbeiter komplett neu geschrieben.
[2] Dieses Kapitel wurde durch den Überarbeiter teilweise neu geschrieben.
[3] Alle detaillierten Schnittstellenbeschreibungen wurden durch den Autor des jeweiligen Kapitels verfasst. Der für Anhang B angegebene Überarbeiter hat diese detaillierte Schnittstellenbeschreibung in den Anhang B verschoben und für jedes Kapitel (9.1 bis 9.13) eine Beschreibung in textueller Form erstellt.

Inhaltsverzeichnis

1 Einleitung

Schon seit Jahren nimmt das Interesse der Wissenschaft und Wirtschaft an verteilten Systemen kontinuierlich zu, und obgleich die Informatik bereits zahlreiche interessante und effiziente Ergebnisse hevorgebracht hat, finden diese eher selten interdisziplinäre Anwendung. Aus diesem Grund nahm sich die Projektgruppe 475 zum Ziel, einen Werkzeugkasten zu erschaffen, der den Entwicklern von verteilten Systemen zahlreiche effiziente Verwaltungsbausteine zur Verfügung stellt. Da ein Werkzeug ohne Anwendungsdomäne schwer zu entwickeln ist, widmete sich die Projektgruppe dezentralen Steuerungen für stetige Stückgutförderanlagen - einer äußerst interessanten und aktuellen Problemstellung aus der Intralogistik[1]. Inspiriert wurde diese Problemstellung durch den Lehrstuhl für Förder- und Lagerwesen der Universität Dortmund, welcher der Projektgruppe als Kooperationspartner zur Seite stand.

1.1 Die Problemstellung und ihre Motivation

Komplexe Materialflusssysteme werden trotz der Fortschritte in der Automatisierungs- und Steuerungstechnik heute, von wenigen Prototypenentwicklungen einmal abgesehen, mit zentralen SPS[2]-Steuerungen bedient und bleiben nahezu unverändert über einen längeren Zeitraum in Betrieb [6]. Dadurch sind die meisten klassischen Steuerungen starre Systeme mit einem begrenzten Maß an Flexibilität. Änderungen erfordern dabei primär die Überarbeitung und Neuinstallation des oft komplexen Steuerungsprogramms, verbunden mit dem vorübergehenden Stop des Systems [6]. Nach [7] wird diese Problemstellung durch die folgenden Faktoren erschwert:

- Keine Anlage gleicht der anderen.
- Großer Entwicklungsaufwand.
- Hohe Inbetriebnahmekosten.
- Kleine Änderungen haben große Auswirkungen auf das Gesamtsystem.
- Die Steuerungslogik bleibt unverändert über den ganzen Lebenszyklus (engl.:*Lifecycle*) der Anlage.

Nach [7] werden klassische Steuerungssysteme bzw. SPS nach dem IEC Standard 1131 in einer Assembler-ähnlichen Programmiersprache entwickelt. Dies kann in Anbetracht der mittlerweile zur Verfügung stehenden Hochsprachen und Abstraktionskonzepte nicht mehr als zeitgemäß bezeichnet werden. Schließlich entwickelt man heutzutage auch keine komplexe Office-Suite mehr in Assembler, sondern verwendet Hochsprachen wie z.B. Java oder C++.

Möchte man den Betrieb von Materialflusssystemen optimieren, so ist es notwendig, durch kurzfriste Modifikationen die Steuerung an variierende Systemanforderungen anzupassen (siehe [6]). Diese Anpassungen sollten idealerweise im laufenden Betrieb möglich sein.
Variierende Systemanforderungen können sich dabei durch vielerlei Faktoren ergeben, wie z.B.

[1] Def. Intralogistik: Die Intralogistik umfasst die Organisation, Steuerung, Durchführung und Optimierung des innerbetrieblichen Materialflusses, der Informationsströme sowie des Warenumschlags in Industrie, Handel und öffentlichen Einrichtungen.

[2] Def. SPS: Eine Speicherprogrammierbare Steuerung (SPS, engl.: *Programmable Logic Controller*, PLC) ist eine elektronische Baugruppe, die in der Automatisierungstechnik für Steuerungs- und Regelungsaufgaben eingesetzt wird.

- Schwankungen in der Auftragslage
- Schwankungen in den Tagesspektren
- Notwendigkeit von austauschbaren Materialflußstrategien

Schafft man es diesen Anforderungen gerecht zu werden, ergeben sich daraus zahlreiche vor allem wirtschaftliche Vorteile gegenüber den klassischen Lösungen, wie beispielsweise

- Erhöhung der Verfügbarkeit des Systems
- Erhöhung der Skalierbarkeit und Gesamtleistung des Systems
- Erhöhung der Flexiblität des Systems
- Senkung von Entwicklungs- und Wartungskosten
- Geringerer Verkabelungsaufwand (als bei bisherigen Lösungen)
- Konfigurationsanpassungen zur Laufzeit möglich

Deshalb steht die Entwicklung von flexibleren und skalierbareren Materialflusssystemen schon seit längerem auf den Wunschzetteln von vielen Wirtschaftsunternehmen, was dieser Problemstellung einen sehr praxisorientierten Rahmen verleiht.

Besonders aufgrund der stetig zunehmenden Rechenleistung auch im Bereich von eingebetteten Systemen und der Entwicklung/Evaluierung der RFID-Technologie, bieten sich nunmehr ganz neue Möglichkeiten, welche die Aufteilung von Steuerungsaufgaben in einem dezentral organisierten System enorm erleichtern. Deshalb steht nun das nötige Rüstzeug bereit, um einen Paradigmenwechsel in der Steuerungstechnik herbeizuführen. Zumindest sollte dies im Zuge der Projektgruppe prototypisch nachgewiesen werden.

1.2 Zielsetzung

Die eigentliche Zielsetzung dieses Projekts kann man sich am besten an der klassischen Steuerungspyramide vergegenwärtigen (siehe Abb. 1.1). Wie man sieht, läßt sich diese vor allem dadurch charakterisieren, dass auf den einzelnen Steuerungsebenen verschiedenartige Kommunikationsverfahren verwendet werden, und dass die Regelungskompentenz von der Leitebene bis zur Feldebene stark abnimmt. So steckt in den einzelnen SPS-Steuerungen für die Sensoren und Aktoren der Feldebene nur recht wenig Logik, wohingegen die Leitebene über eine globale Sicht verfügt. Von diesem Standpunkt aus kann sie alle nötigen Steuerungsentscheidungen ohne Weiteres treffen allerdings mit der Konsequenz, dass ein Ausfall oder Umbau in der Leitebene fatale Folgen für das gesamte System hat.
Es besteht bereits in der klassischen Steuerungspyramide und damit auch in klassischen Materialflusssystemen ein gewisses Maß an Verteilung, was man vor allem an der Ethernet-basierenden Kommunikation in Leit- und Zellebenen ablesen kann. Allerdings ist diese Verteilung noch nicht ausgeschöpft und hat besonders in die prozessnahen Bereichen noch keinen Einzug gehalten. Aus diesem Grund beschäftigt sich der Lehrstuhl für Förder- und Lagerwesen bereits seit längerem mit dieser Problemstellung und hat in der Vergangenheit einen ersten Prototyp einer prozessnahen dezentralen Steuerung für eine eigens dafür umgebaute Testanlage entwickelt. Ausgehend von dieser Arbeit wollte die Projektgruppe 475 ein Framework erstellen, dass die Entwicklung von dezentralen Steuerungen erleichtern bzw. unterstützen soll. Besonderes Augenmerk wurde im Zuge dessen auf Komponenten gelegt, die häufig auftretende Verwaltungsprobleme (wie z.B. Topologie- und Routenerkundung) lösen. Außerdem wurde darauf geachtet, die Steuerungspyramide abzuflachen und die Leitebene nur noch für die Zustandsvisualisierung und Statuswechsel von Komponenten zu nutzen.

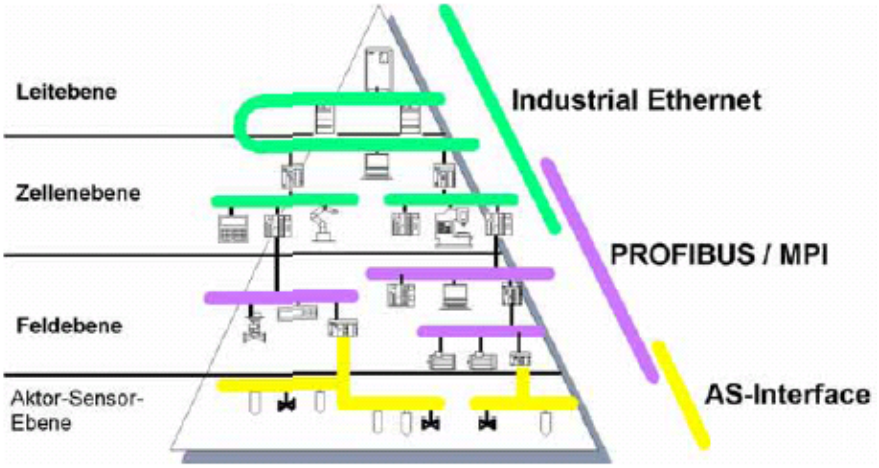

Abbildung 1.1: Schematische Darstellung der klassischen Steuerungspyramide

1.3 Kapitelübersicht

Im Folgenden werden die Kapitel kurz vorgestellt, um dem Leser einen Überlick über den Endbericht machen zu geben.
Kapitel 2 beschreibt zunächst das Projekt an sich, also Ziele, Organisation und Zeitplanung der Projektgruppe. Kapitel 3 gibt eine allgemeine Einführung in das zentrale Werkzeug: Web Services. Die Projektgruppe 475 hatte das Glück, dank einer Kooperation mit dem Lehrstuhl für Förder- und Lagerwesen eine reale Förderanlage steuern zu dürfen; diese wird in Kapitel 4 genauer betrachtet und dort insbesondere auf die verbaute Hardware eingegangen. Die Informationen über benutzte Software und Werkzeuge werden in Kapitel 5 vermittelt. Kapitel 6 präsentiert noch weitere generelle Informationen: Anforderungen an Förderanlagen-Steuerungen - genauer gesagt die Erläuterung von funktionalen und nicht-funktionalen Anforderungen. Es wird dabei auch schon speziell auf die Arbeit der Projektgruppe 475 eingegangen. Kapitel 7 beschreibt das Anlagenmodell und bietet eine mathematische Definition für Förderanlagen, -strecken und -knoten. Aufbauend auf dieses Anlagenmodell stellt Kapitel 8 die erarbeitete Systemarchitektur vor. Diese soll zuerst einen Gesamtüberblick über die Steuerung bieten, bevor im Kapitel 9 auf die einzelnen Subkomponenten eingegangen wird, so dass die Steuerung dann komplett beschrieben wird. In Kapitel 10 wird dann die Installation und Testphase direkt an der Anlage des Lehrstuhls FLW vorgestellt. Abschließend bietet Kapitel 11 dann ein Fazit der Projektgruppe 475, also erreichte Ziele, weitere Perspektiven und die persönlichen Sichten der Beteiligten.

2 Projektbeschreibung

2.1 Projektgruppen an der Universität Dortmund

Projektgruppen (kurz: PG's) sind fester Bestandteil der Diplomprüfungsordnung für Informatiker. Eine PG ist eine Lehrveranstaltung des Fachbereichs Informatik an der Universität Dortmund. Das Ziel einer PG ist es, Studenten der Informatik einen Einblick in Berufspraxis und Teamarbeit zu vermitteln. Angesetzt ist ein Zeitrahmen von 2 Semestern, in dem ein Projekt von 8-12 Studierenden bearbeitet wird. Eine Projektgruppe ist wie folgt aufgebaut:
Zuerst sollen Seminare und Praktika dafür sorgen, dass alle Teilnehmer das gleiche Grundlagenwissen besitzen. Danach wird innerhalb der PG der strukturierte Lösungsweg festgelegt. Die in den PG-Sitzungen getroffenen Entscheidungen und Planungen werden in Form von Protokollen festgehalten.
Die Projektgruppe endet mit der Erstellung eines Abschlussberichts und einer Präsentation am Fachbereich. Dabei sollen die Betreuer darauf achten, dass die sachlichen und zeitlichen Planungen der PG mit den PG-Zielen übereinstimmen.

2.2 Projektbeschreibung der PG475

2.2.1 Aufgabe und Minimalziel der PG475

Die Aufgabe der PG 475 war die Erstellung eines Frameworks, mit desssen Hilfe dezentrale und modulare Steuerungen für stetige Stückgutförderanlagen entwickelt werden können. Grundlage dafür war eine dienstorientierte Architektur, in der jede Komponente ihre Funktionen dem System per Web Service zur Verfügung stellt.
Da zur Zeit Steuerungen für Förderanlagen statisch und zentralisiert organisiert sind, verfolgt die PG 475 einen neuen Ansatz, der die Entwicklung und den Betrieb von Materialfluß enorm verbessern könnte: Angesichts schnell wechselnder Anforderungen an Materialflußsysteme und ihre Steuerungen sind die jetzigen SPS-Programme (*Controller*-Programme) nicht mehr als zeitgemäß anzusehen. An dieser Stelle hat die Projektgruppe 475 angesetzt und ein Framework erstellt, das es ermöglicht, in einer gängigen Hochsprache (Java/C++) und mit einer diensteorientierten Architektur, Förderanlagensteuerungen zu entwickeln, die den Anforderungen der heutigen Zeit gewachsen sind. Das Minimalziel dabei war die Entwicklung einer prototypischen dezentralen auf Web Services basierenden Steuerung.

2.2.2 Anforderungen an den Prototypen

Der Prototyp muss folgende Eigenschaften haben:

- Dezentralität
- Modularität
- Autonomität
- Adaptivität
- Fehlertoleranz
- Proaktivität

Dezentralität

Der Prototyp soll möglichst dezentral gesteuert sein, d.h. es soll innerhalb des Systems so wenig wie möglich zentral entschieden werden. Im Idealfall organisieren sich die Komponenten in einem Peer-to-Peer System selbst. Durch die Verteilung der Steuerung können einige Probleme auftreten, die bei einer zentralen Steuerung leichter zu beheben wären bzw. gar nicht vorhanden sind. Dazu gehören

- Wegewahl bei Weichen
- Partnersuche (Finden der angeschlossenen Förderelemente - direkte Nachfolger)
- Kollisionsverhinderung/Priorisierung bei Zusammenführungen

Modularität

Das Konzept des Prototypen soll möglichst modular sein, so dass man einzelne Komponenten (insbesondere die anlagenspezifischen) einfach austauschen kann ohne die anderen ändern zu müssen.

Autonomität

Der Prototyp soll nach dem Einschalten automatisch in einen betriebsbereiten Zustand übergehen. Dazu sind folgende Schritte nötig:

- Alle Netzwerkteilnehmer sowie deren Dienste identifizieren sich
- Jeder Netzwerkteilnehmer meldet sich bei den bereits im Netz befindlichen Teilnehmern an
- Erkundung der Topologie
- Funktionale Einheiten untereinander und den Förderelementen zuordnen
- Berechnung von eigenen Routingtabellen an Weichen

Jedes Förderelement „kennt“ nur sich selbst und seinen eigenen Standort. Durch Absprachen mit den anderen Förderelementen wird es möglich, dass alle Förderelemente zusammen genauso funktionieren, wie bei einer klassischen Steuerung in Form einer SPS, d.h. sie transportieren Pakete von A nach B, ohne im Vorfeld zu wissen, welche Topologie die Förderanlage hat.

Adaptivität

Auf Veränderungen der Topologie beispielsweise durch Hinzufügen eines Förderelements soll der Prototyp zur Laufzeit reagieren und die Veränderungen in seine zukünftigen Steuerentscheidungen mit einbeziehen. Folgende Punkte müssen vom System dazu berücksichtig werden:

- Erkennen von Topologie-Änderungen: Aktualisierung der Informationen über vorhandene Netzwerkteilnehmer und deren Dienste.
- Aktualisierung der Topologie und ihrer Visualisierung.
- Erneute Berechnung aller Routen
- Hinzukommen von Förderelementen:
  Zuordnung funktionaler Einheiten zu den Förderelementen.
- Wegfallen von Förderelementen:
  Deaktivierung aller nun unbrauchbarer Förderstrecken.

Fehlertoleranz

Der Ausfall von Hard- und Software soll erkannt und behandelt werden.

1. Vorgehen bei Ausfall von Software Komponenten:
   - Verteilung der Arbeit auf den Ersatzknoten.
   - Neuinitialisierung der Komponente (falls möglich).
2. Vorgehen bei Ausfall von Hardware Komponenten:
   - Fall 1: IPC erkennt defekte Hardware. Fehlermeldung an zuständige Komponente senden. Weitere Fehlerbehandlung erfolgt durch die zuständige Komponente.
   - Fall 2: Der Ausfall eines IPCs muss von der zuständigen Steuerungskomponente bemerkt werden.

Proaktivität

Es sollen, sofern möglich, dezentrale Entscheidungen im Vorfeld getroffen werden. Das bedeutet, dass die Topologie und auch die Routen berechnet werden, bevor die Pakete da sind.

2.3 Organisation und zeitlicher Ablauf

2.3.1 1. Semester

Seminare

In den Seminaren wurden Referate an die Teilnehmer verteilt, die inhaltlich zu der Projektgruppe passten. Jeder musste zu seinem Thema eine zehnseitige Ausarbeitung schreiben. Dazu kam noch eine Präsentation. Dies diente der Einarbeitung in die Themen Web-Services, Logistik, Systembeschreibung, Middleware-Plattformen und verteilte Algorithmen.

Infrastruktur

Es wurde ein Team Infrastruktur gebildet, dass sich um eine Arbeitsumgebung gekümmert hat. Es wurde ein PG-Server aufgesetzt, der ein Wiki sowie ein Subversion-Repository zur Versionsverwaltung zur Verfügung stellte. Jeder Teilnehmer hatte auf diesen Server Zugriff und konnte Wikiseiten erstellen und Dateien up- und downloaden.

Praktika

In den Praktika stellten sich die PG-Mitglieder untereinander Aufgaben, um sich so in verschiedenste Themengebiete einzuarbeiten. Bei Bedarf wurde eine kleine Präsentation vorbereitet, um den anderen PG-Teilnehmern das Themengebiet und die Aufgaben zu erläutern. Folgende Themen wurden hier behandelt: Eclipse, Subversion, Latex, Java und UML, TLA/cTLA, Web Services und Nutzung des PG-Servers.

Teambildung und Arbeit in den Teams

Nach den Praktika wurden die Teilnehmer in verschiedene Teams eingeteilt. So gab es ein Team „Hardware “. Diese Gruppe setzte sich mit der Förderanlage und allen damit verbundenen Problemen und Aufgaben auseinander. Weiterhin gab es ein Team „Dezentrale Steuerungslogik “, das sich mit der Steuerung und deren Umsetzung beschäftigt hat. Außerdem wurde ein Team „Simulation “gebildet, das sich um eine Emulierung der Anlage gekümmert hat, um so auch ohne eine reale Anlage arbeiten zu können. Zuletzt gab es ein Team „Monitoring “, dass eine grafische

Ansicht, sowohl in 2D zur Log- und Eventausgabe als auch in 3D entwickelte. Die Ergebnisse wurden im Wiki festgehalten, um jedem eine Möglichkeit zu bieten, sich über die Aktivitäten in den Gruppen zu informieren. Die Phase endete am Anfang des 2. Semesters mit der Erstellung der Spezifikationen der einzelnen Komponenten, welche als Implementierungsvorlagen dienten.

2.3.2 2. Semester

Walkthrough

Um zu überprüfen, ob die einzelnen Komponenten aufeinander abgestimmt waren, wurden ein Initialisierungswalkthrough und ein Walkthrough bei laufendem System durchgeführt und im Wiki festgehalten. Anhand dieser Walkthroughs konnten viele Unklarheiten geklärt und die jeweiligen Spezifikationen angepasst werden.

Implementierung und Testphase

In der Implementierungsphase wurde weitestgehend in Zweier-Teams gearbeitet. Die Programmierung hielt sich an die Spezifikationen, die in den vorherigen Phasen erstellt wurden. Jedem Team wurden verschiedene Aufgabengebiete zugeordnet, die dem ausgearbeiteten Zeitplan entsprechend abgearbeitet werden sollten. Nach der Implementierung einer Komponente wurde diese mit Hilfe von JUnit auf Funktionsfähigkeit getestet. Im Anschluss daran begann der Integrationstest, an dem bis zum Ende der Projektgruppe immer weitere fertiggestellte Komponenten teilnahmen.

Installationsphase

In dieser Phase wurde das gesamte System am Lehrstuhl für Förder- und Lagerwesen auf die IPCs und auf mehreren Laptops installiert. Auf den IPCs wurde das System auf den Flashkarten installiert. Nach der Installation wurde die Steuerung auf der realen Anlage mit echten Paketen getestet.

Endbericht

Für den Endbericht wurde eine Gliederung erstellt, so dass sich der Endbericht aus 11 Kapiteln zusammensetzt. Das Kapitel 9 über die einzelnen Komponenten wurde noch in mehrere Subkapitel unterteilt. Dadurch konnten die einzelnen Abschnitte auf die Teilnehmer verteilt werden, so dass jeder die ihm zugeordneten Teile schreiben konnte.

Präsentationen

Es fanden zwei Abschlusspräsentationen statt, eine am Lehrstuhl für Förder- und Lagerwesen, bei der auch die Steuerung an der realen Anlage vorgeführt wurde. Die zweite Präsentation fand am Lehrstuhl 4 des Fachbereichs Informatik statt. Bei dieser Vorführung wurde die unabhängig von der realen Anlage arbeitende Emulation gezeigt.

3 Web Services

Web Services sind webbasierte Anwendungen, die offene XML-Standards und Transportprotokolle zum Datenaustausch mit Clients benutzen. Durch die Verwendung von verbreiteten, akzeptierten Standards ist ein interoperabler Austausch möglich. Für die Beschreibung der Schnittstellen wurde die Web Service Definition Language (WSDL) spezifiziert. Diese ermöglicht es, die Parameter und Typen zu beschreiben, die für die Benutzung der Anwendung notwendig sind.
Web Services sind mit dem Ziel entwickelt worden, eine automatisierte Kommunikation über das Internet zu ermöglichen. Dabei orientieren sie sich an der *Service Oriented Architecture (SOA)*. Durch lose Kopplung einfacher Dienste lassen sich komplexe Dienste generieren, die durch die Aneinanderreihung von eigenständigen Serivces entstehen. Um den eigenen Dienst öffentlich zu machen, kann man diesen mit UDDI (*Universal Description, Discovery and Integration*) beschreiben und ihn in einen Verzeichnisdienst eintragen. Der Eintrag besteht aus einer Dienstbeschreibung und den Funktionen, die dieser Service anbietet. Dadurch kann jeder Client oder Konsument erkennen, ob diese Funktion für ihn interessant ist und ob er die Funktionalität in seinem Projekt nutzen möchte. Als gutes Beispiel gilt der Google Web Service, der es jedem ermöglicht, per Web Service-Nutzung in dem Index von Google zu suchen. Die Implementierung des Eingabefeldes für die Suchbegriffe im eigenen Projekt bestimmt dann der Programmierer selbst.
Für die Datenübertragung vom Client zum Serviceanbieter wird SOAP verwendet, das aus dem von Microsoft erfundenen XML RPC entstanden ist. Die SOAP Spezifikation wurde diesmal nicht mehr von einer großen, dominierenden Firma erarbeitet, sondern mit Hilfe von IBM bis zur Version 1.2 weiter entwickelt und anschließend im Jahr 2000 dem W3C zur Standardisierung vorgelegt. Eine SOAP Nachricht besteht aus drei Teilen: dem *SOAP Envelope* (Umschlag), in dem *SOAP Header* und *SOAP Body* stecken. Die SOAP Nachricht selbst wird dann mit Hilfe des Transportprotokolls TCP übertragen. Web Service-Nachrichten bestehen aus reinem XML. Binäre Dateien können nur kodiert in die eigentliche SOAP Nachricht eingebunden werden (z.B. mit Base64). Möchte man mehrere große Anhänge mitschicken, kann das (de-)kodieren zu aufwändig werden. Hier bieten sich dann Nachrichten-Anhänge an. Die Architektur einer SOAP Nachricht wird durch das SOAP Attachment Feature erweitert, so dass SOAP Envelope und Attachment in einen gemeinsamen Container (*Compound Message*) gepackt werden.

BPEL4WS BPEL steht für *Business Process Execution Language* und definiert eine Beschreibungssprache für die Komposition von unterschiedlichen Web Services zu einem Geschäftsprozess. Mit Hilfe von BPEL werden Web Services so beschrieben, dass sie von anderen Web Services aufgerufen und in ihre Geschäftsprozesse integriert werden können oder zu neuen Diensten zusammengesetzt werden. Mit BPEL wird der gesamte Prozessablauf festgelegt, aber auch, welche Eingaben ein Web Service benötigt, was mit den Ausgaben passiert oder welcher Teil von welchem Dienst zur Verfügung gestellt wird.

WS-Coordination Die Web Service Coordination Spezifikation beschreibt ein erweiterbares Framework, das die Definition von Protokollen für die Teilnahme an verteilten Aktivitäten erlaubt. Das Framework unterstützt verschiedene Koordinationstypen, wie z.B. atomare und Business-Transaktionen. Die Hauptaufgabe besteht jedoch in der Koordination der Web Services untereinander sowie der Registrierung für die Teilnahme an dem Prozeß. Der Austausch erfolgt über einen sogenannten *Coordination Context*, der Attribute besitzt, die sich während einer Sitzung verän-

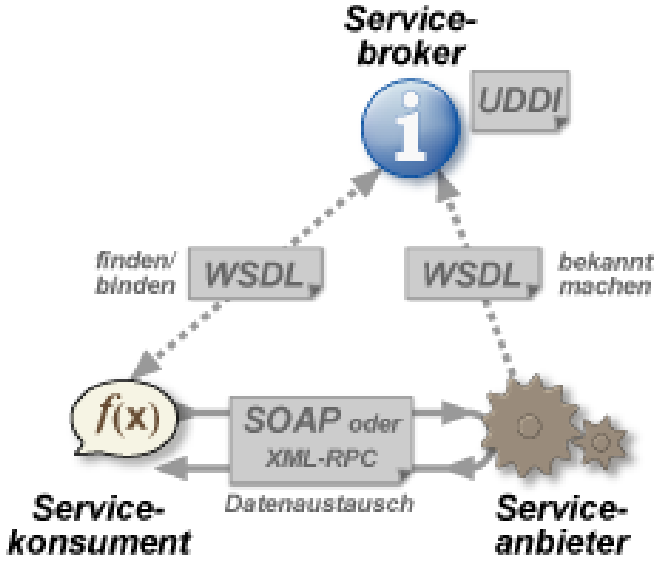

Abbildung 3.1: Kommunikation mit Web Services

dern. Dieser Coordination Context bildet mit dem Registration-Service und Activation-Service die drei wesentlichen Elemente von WS-Coordination.

WS-Eventing WS-Eventing unterstützt asynchrone Verbindungen, so dass man sich für ein Event registrieren kann und anschließend die Verbindung trennt. Tritt ein Event auf, wird eine Verbindung zu dem Interessenten aufgebaut und eine Aktion ausgeführt.

WS-Security WS-Security ermöglicht es die Integrität und Vertraulichkeit von Nachrichten sicherzustellen. Um dies zu erreichen, kann auf eine Vielzahl von Sicherheitssystemen(z.B. PKI, SSL) zurückgegriffen werden, die die SOAP Nachricht verschlüsseln oder den SOAP-Umschlag signieren. Eine grössere Sicherheit läßt sich jedoch nur in Kombinationen mit höher angesiedelten programmspezifischen Protokollen erreichen.

WS-Transaction

- **WS-Atomic Transaction (WS-AT):** WS-AT unterstützt das ACID (Atomic, Consistent, Isolation, Durable) Prinzip, so dass entweder alle oder keine Operationen ausgeführt werden. Tritt ein Fehler bei der Ausführung einer Anweisung auf, müssen alle vorherigen Anweisungen zurückgenommen werden. Anwendung findet dieses Protokoll hauptsächlich bei lokalen, kurzlebigen Transaktionen.
- **WS-BusinessActivity (WS-BA):** WS-BA wurde für den Einsatz von verteilten Geschäftsprozessen entwickelt, welche sehr lange dauern können. Aus diesem Grund darf die Anwendung nicht über den ganzen Zeitraum blockiert bleiben. WS-BA sieht für diese langlebigen Prozesso nun vor, dass sogenannte “Activity Scopes” definiert werden, in der die Ressourcen erst allokiert werden, wenn sie gebraucht werden. Geschriebene Daten werden nicht mehr zurückgenommen, sondern nur bei einem Konflikt durch “Compensation Tasks” korrigiert.

4 Anlage des Lehrstuhls für Förder- und Lagerwesen

In diesem Kapitel wird die Zusammensetzung der Förderanlage beschrieben, die der Projektgruppe vom Lehrstuhl für Förder- und Lagerwesen (kurz: *FLW*) als Teststellung zur Verfügung gestellt wurde.

Der Lehrstuhl für Förder- und Lagerwesen beschäftigt sich mit Themen aus der Intralogistik wie Materialflussplanung, Materialflusssteuerung und Materialflusstechnik. Im Zuge des Projekts stand das FLW als Ansprechpartner bei logistischen Fragen zur Verfügung und hat somit die Kundenseite des Projekts ausgefüllt.

4.1 Förderanlage

Da die Förderanlage des FLW bereits für die Umsetzung einer dezentralen Steuerung im Zuge des Projekts "*Realtime Logistics::Dezentrale Steuerung*[1]" eingesetzt wurde, war ihr Aufbau bereits für den dezentralen Einsatz ausgelegt. So waren schon mehrere Industrie PCs (kurz: *IPCs*) oder Knotenrechner an der Anlage verbaut, die jeweils gesonderte Teilbereiche der Anlage steuern. Steuern bedeutet in diesem Fall, dass die steuerbaren Entitäten eines Teilbereichs an den Knotenrechner über einen speziellen Bus (hier: *K-Bus*) angeschlossen wurden.
In der derzeitigen Ausbaustufe besteht die Förderanlage aus 36 Fördersegmenten auf zwei Ebenen mit einer Gesamtlänge über 120m. Die durchschnittliche Fördergeschwindigkeit beträgt ca. 1 m/sec., was einen Materialfluß von ca. 15 Transporteinheiten/min ermöglicht. Auf der Anlage sind etwa 50 Antriebe, sowie ca. 60 Sensoren verbaut, welche an insgesamt sieben Knotenrechner angeschlossen sind.

4.1.1 Modularisierung

Um der Zielsetzung der strikten Modularisierung der Steuerung (siehe Kapitel 2.1) gerecht zu werden, muss die Anlage selbst bereits modular aufgebaut sein. So unterteilt sich die Anlage des FLW in 36 Fördersegmente, die man prinzipiell alle autonom mit gesonderten Knotenrechnern betreiben könnte.
In diesem Zusammenhang wurde die folgenden Terminologie festgelegt, welche in den folgenden Abschnitten genutzt wird:

Definition: Förderelement Ein Förderelement ist der Zusammenschluß von Fördertechnik und Aktoren bzw. Sensoren. In diesem Sinne entspricht ein Förderelement also einem fördertechnischen Modul. Somit kann eine Förderanlage als eine Komposition von Förderelementen verstanden werden. Wie man in Abbildung 4.1 sieht, teilt sich die untere Ebene der Förderanlage in 13 Förderelemente (U1 - U13) auf.

Definition: Förderdevice In den folgenden Abschnitten wird für Aktoren und Sensoren des Öfteren der Begriff Förderdevice verwendet (eine genauere Definition befindet sich in Kapitel 7).

[1] Realtime Logistics::Dezentrale Steuerung ist ein Projekt des Lehrstuhls für Förder- und Lagerwesen der Universität Dortmund (siehe www.realtime-logistics.de)

4.1.2 Typen von Fördererelementen

In diesem Abschnitt werden die einzelnen Förderelemente anhand von charakterisierenden Eigenschaften typisiert. Diese Typisierung war nötig, um innerhalb des Roasted Kit die einzelnen Förderelemente adäquat steuern zu können. Dabei wurden die folgenden Typen von Fördererelementen festgelegt:

- Einfaches Förderelement
- Puffer
- Zusammenführung
- Weiche
- Wareneingang bzw. Warenausgang

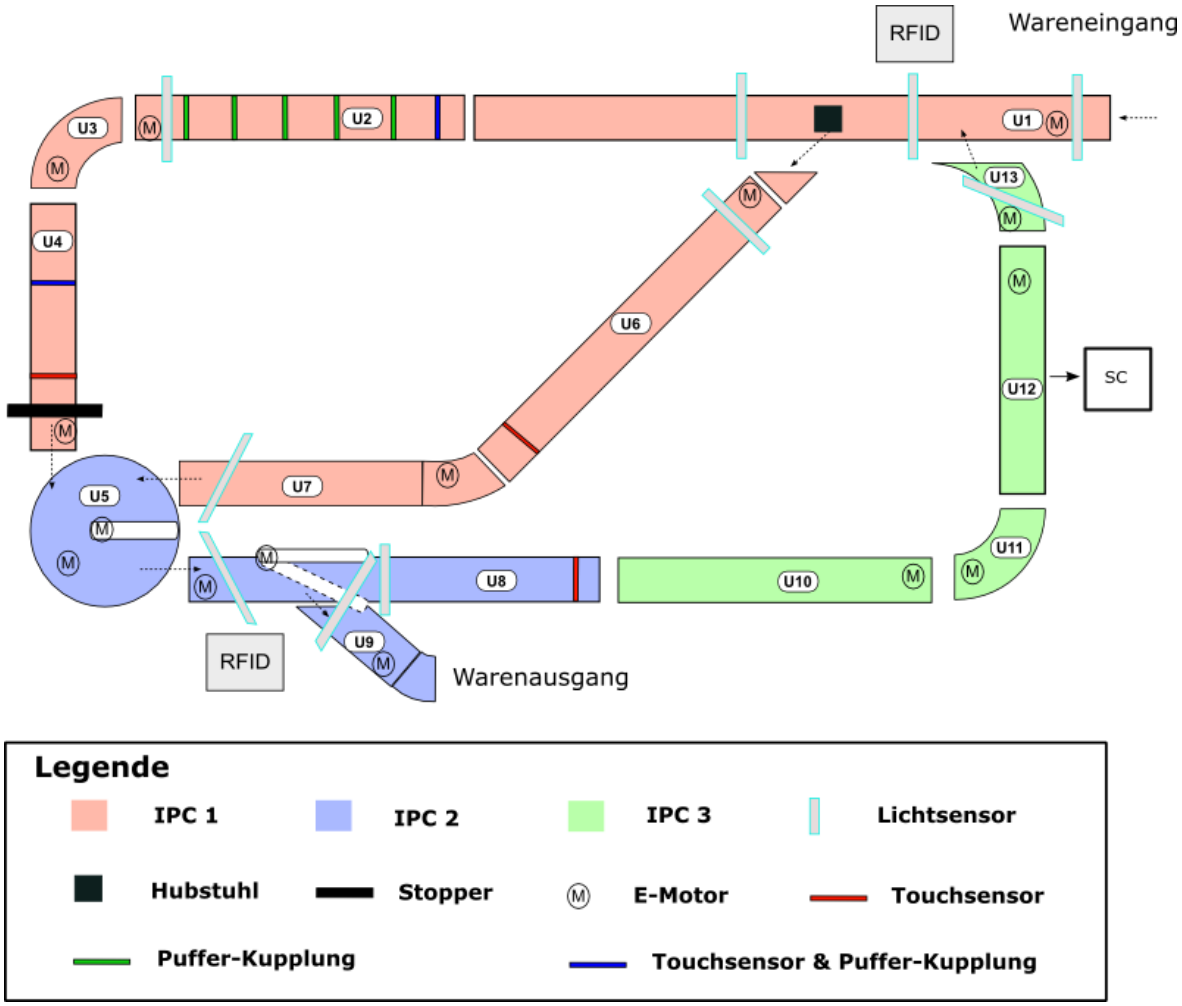

Abbildung 4.1: Die Förderanlage des Lehrstuhls FLW

4.2 Verwendete Förderelemente und ihre Knotenrechner

Aufgrund der Komplexität der Aufgabenstellung hat sich die Projektgruppe dazu entschieden nur die untere Ebene der Förderanlage (des FLW) in die Steuerung miteinzubeziehen. Deshalb

wurde das Förderelement U9 als Warenausgang definiert. Die Topologie der unteren Ebene kann der Abbildung 4.1 entnommen werden.
Gesteuert werden die Sensoren und Aktoren der unteren Ebene von drei Knotenrechnern/IPCs. Die verwendeten Knotenrechner basieren auf einer PC-Architektur in hutschienenmontierbaren Gehäusen, werden direkt in den Schaltschränken montiert und verfügen weder über ein Display, noch über eine Tastatur. Der Datenaustausch, die Konfiguration und die Bedienung finden ausschließlich über die Ethernet-Schnittstelle statt (vgl. [7]).
Es folgt eine Beschreibung der einzelnen Knotenrechner und der von ihnen gesteuerten Förderelemente in Fließrichtung beginnend am Wareneingang:

4.2.1 Knotenrechner UC1

Der Knotenrechner UC1 steuert die Förderelemente U1, U2, U3, U4, U6 und U7 und hat damit von allen Knotenrechnern der unteren Ebene die meisten Förderelemente in seiner Obhut (siehe Abb. 4.2). Erreichbar ist UC1 über die IP-Addresse 192.168.11.30.

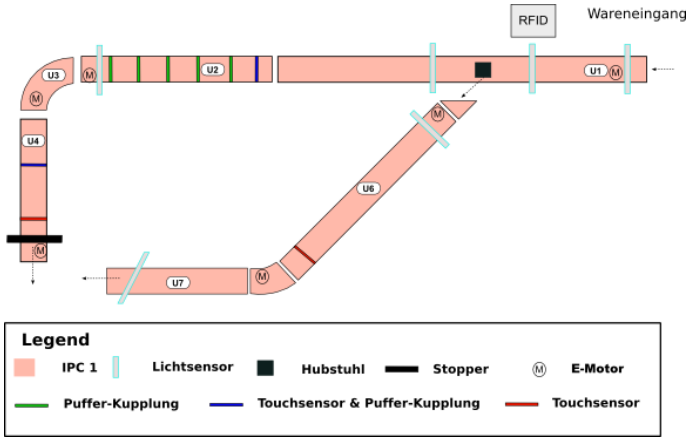

Abbildung 4.2: Von IPC1 gesteuerte Förderelemente

4.2.2 Knotenrechner UC2

Der Knotenrechner UC2 steuert die Förderelemente U5, U8, U9 und U14 wobei U14 im Zuge des Prototypen nicht mit in die Steuerung integriert wurde (siehe Abb. 4.3). Erreichbar ist UC2 über die IP-Addresse 192.168.11.31.

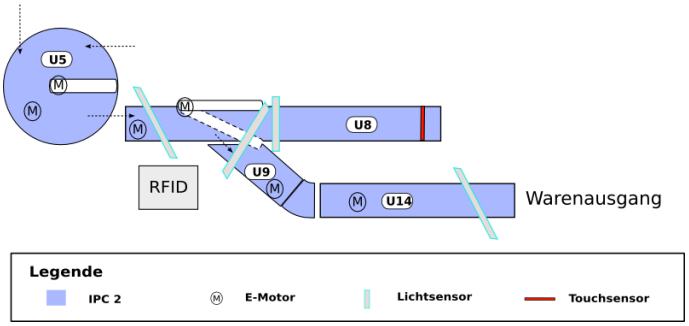

Abbildung 4.3: Von IPC2 gesteuerte Förderelemente

4.2.3 Knotenrechner UC3

Der Knotenrechner UC3 steuert die Förderelemente U10 bis U13 (siehe Abb. 4.4). Erreichbar ist UC3 über die IP-Addresse 192.168.11.32.

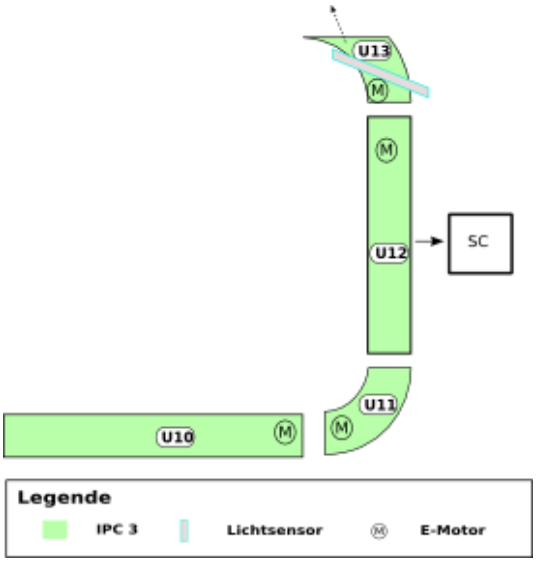

Abbildung 4.4: Von IPC3 gesteuerte Förderelemente

4.3 Die RFID-Anlage

Der Materialfluß auf der Anlage wird mit Hilfe einer RFID-Anlage gesteuert (siehe Abb. 4.5). Es gibt zwei RFID-Antennen, die sich an den Förderelementen U1 und U8 befinden. Diese werden von einer Zentraleinheit gesteuert. An den Paketen sind batteriebetriebene RFID-Tags angebracht, welche von den Antennen ausgelesen werden, sobald sie sich im Scan-Radius befinden. Auf den RFID-Tags befindet sich in der derzeitigen Ausbaustufe des Roasted Kit nur eine eindeutige Identifikationsnummer. Dadurch ist es möglich den virtuellen und realen Materialfluß zu synchronisieren. Die Steuergeräte der RFID-Anlage stehen über Ethernet verkabelt mit den IPCs in Verbindung. Das Steuergerät kann über die IP-Adresse 192.168.11.40 angesprochen werden.

Abbildung 4.5: Komponenten der RFID-Anlage

5 Eingesetzte Software und Werkzeuge

Für die Entwicklung des Roasted Kit wurden viele verschiedene Bibliotheken und Werkzeuge eingesetzt. Es sollte, im Gegensatz zu den meisten konventionellen Steuerungen, auf Hochsprachen wie Java und C++ zurückgegriffen werden. Dadurch soll eine Modularisierung des Systems deutlich erleichtert werden, um so später die gewünschte Flexibilität zu erreichen. Aufgrund der Dezentralität der einzelnen Komponenten des Prototypen ist es für eine flexible Steuerung notwendig, dass sich diese Komponenten während der Systeminitialisierung oder zur Laufzeit „finden“. Hierzu wird im Roasted Kit UPnP genutzt. Da Web Services und die ihnen zugrunde liegende *Service Oriented Architecture (SOA)* ein vielversprechender Ansatz für die Konzeption von komplexen Systemen sind, hat die Projektgruppe sich dazu entschlossen, innerhalb des Prototypen auf Web Services zurückzugreifen. Die eigentliche Steuerung und Logik des Systems wurde in Java, Version 1.5, implementiert. Als Entwicklungsumgebung wurde Eclipse benutzt. Die Steuerung der Hardware der Förderanlage wurde in C++ implementiert. Die C++ Implementierung wurde in KDevelop 3.2.3, einer Entwicklungsumgebung unter Linux, realisiert.

5.1 Eclipse und die genutzten Plugins

Als Entwicklungsumgebung (IDE) für RoastedKit wurde Eclipse (Abb. 5.1) in Version 3.0.0 bis 3.1.2 benutzt. Eclipse war erst unter Leitung von IBM, nach Freigabe der Quellcodes ist es ein OpenSource-Projekt geworden, um dessen Verwaltung und Entwicklung sich die Eclipse-Foundation kümmert.

Eclipse ist ein Framework zur Integration verschiedenster Anwendungen. Die wichtigsten Plugins, welche die PG nutzte, sind:

- SWT: Grafik-Bibliothek, zur Erstellung der Orderform und der SWT-Control
- Visual Editor: Plugin zur Erstellung von Benutzeroberflächen mit SWT.
- Subclipse: Zur Versionsverwaltung des Quellcodes
- Omondo: Zur Erstellung von UML-Diagrammen
- JUnit: Zum Komponententest
- Web Standards Tool (WST): Quellcode-Editor für “Internet-Sprachen“ wie HTML, JavaScript etc.

Das Web Standards Tool (WST), benutzt in der Version 0.7-1.0, ist eines der Unterprojekte des Eclipse Web Tool Platform (WTP) Projekts. Hier bietet es unter anderem Quelltexteditoren für HTML, JavaScript, SQL, CSS u.a. Ferner inbegriffen sind ein Assistent zur Erstellung von Web Services und Werkzeuge zum Testen der Web Service-Interoperabilität. Das „Web Standards Tool“ ist ein Plugin, das die Entwicklung von Web-Anwendungen in Eclipse untestützt. Während unseres Projekts wurde der komfortable XML-Schema/WSDL-Editor eingesetzt, der in Abb. 5.3 zu sehen ist.

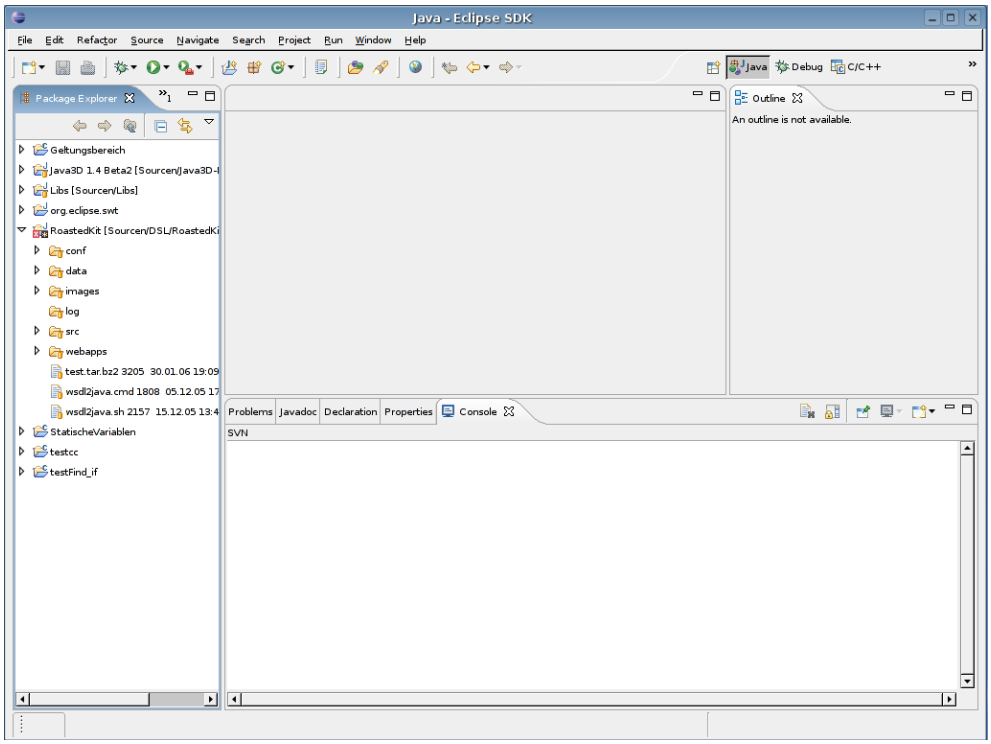

Abbildung 5.1: Eclipse in der Version 3.1.1

5.2 Subversion

Für die Versionsverwaltung aller Dokumente, Abbildungen und Quellcodes der Projektgruppe wurde die Open-Source-Software Subversion ab 1.0 verwendet. Ziel war es, alle erstellten Dateien an einer zentralen Stelle abzulegen, um sicherzustellen, dass diese stets für alle verfügbar waren.

5.3 Trac - Projektmanagement

Alle wichtigen Informationen über die Arbeit der Projektgruppe wurden im Trac-Project Management (Version 0.8.1) festgehalten. Trac ist ein erweitertes Wiki, ähnlich wie die freie Enzyklopädie Wikipedia. Für die Projektgruppe diente es vornehmlich als Entwicklungswerkzeug und Projektmanagement-Software. Trac unterstützt ein *Wiki-Markup*, d.h. es hat seine eigene Wiki-Syntax, um die Texte zu formatieren oder Tabellen anzulegen. Das Wiki-Markup hat auch die Eigenschaft, HTML-Syntax zu erkennen. Man kann Links auf Dateien, Webseiten oder andere Wikiseiten setzen. Die Änderungen einer Wiki Seite und auch am gesamten Quelltext können verfolgt werden und man kann gegebenenfalls auch ältere Versionen wiederherstellen. Trac beinhaltet ein Ticketing System, mit dem man Aufgaben und Bugs sowohl zeitlich als auch personell genau definieren kann. Trac bietet eine Schnittstelle zu Subversion und zeigt in einer Reportansicht alle Änderungen, sowohl im Wiki als auch im Quelltext und sonstigen Dateien der Entwickler an. Die Rechte des Wikis können über Trac Permission individuell und passwortgeschützt angepasst werden.

5.4 Universal Plug and Play (UPnP)

Universal Plug and Play (UPnP) basiert auf einer Reihe von standardisierten Netzwerkprotokollen und Datenformaten. Es dient zur herstellerübergreifenden Ansteuerung von Geräten (Stereo-

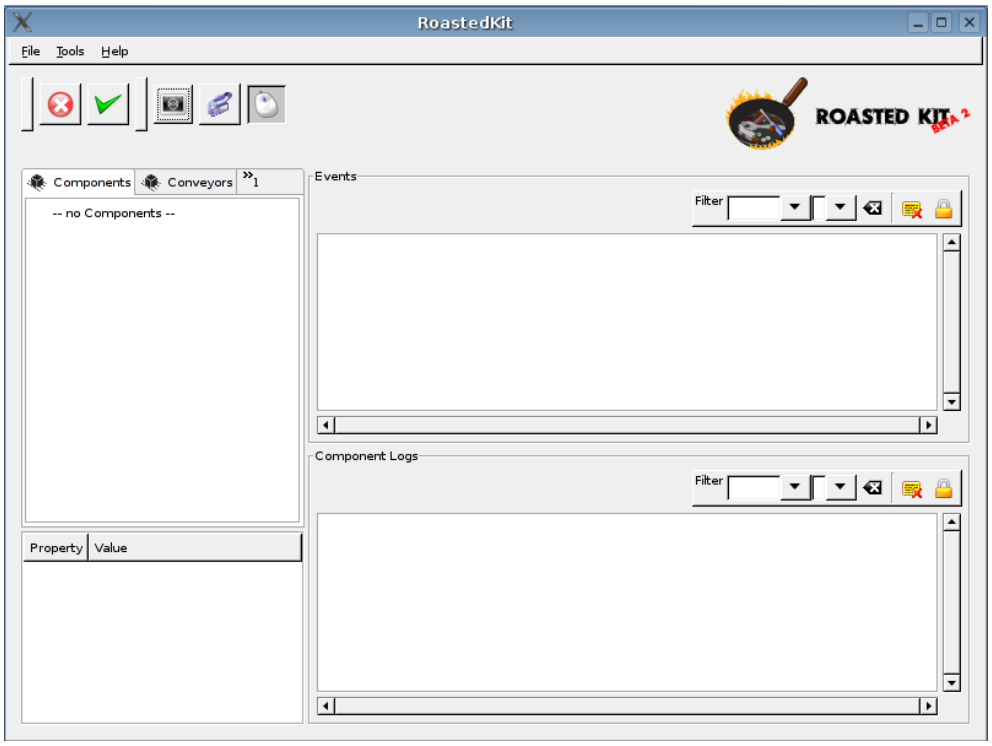

Abbildung 5.2: GUI RoastedKit, erstellt mit SWT

anlagen, Router, Drucker, Haussteuerungen) über ein IP-basierendes Netzwerk.UPnP zeichnet sich insbesondere durch folgende Merkmale aus:

- Ein Kontrollpunkt kann andere Geräte ohne Interaktion des Benutzers finden
- Alle physikalischen Medien, die IP-Kommunikation unterstützen, können verwendet werden, z. B. Ethernet, Funk (Bluetooth, Wireless LAN), FireWire (IEEE 1394)
- Es werden standardisierte Technologien wie IP, UDP, Multicast, TCP, HTTP, XML, SOAP, etc. verwendet
- Ein UPnP-Gerät oder -Kontrollpunkt kann auf jedem IP-fähigen Betriebssystem und mit den verschiedensten Programmiersprachen (hauptsächlich C, C++ und Java) realisiert werden.
- UPnP bietet Möglichkeiten für herstellerspezifische Erweiterungen.

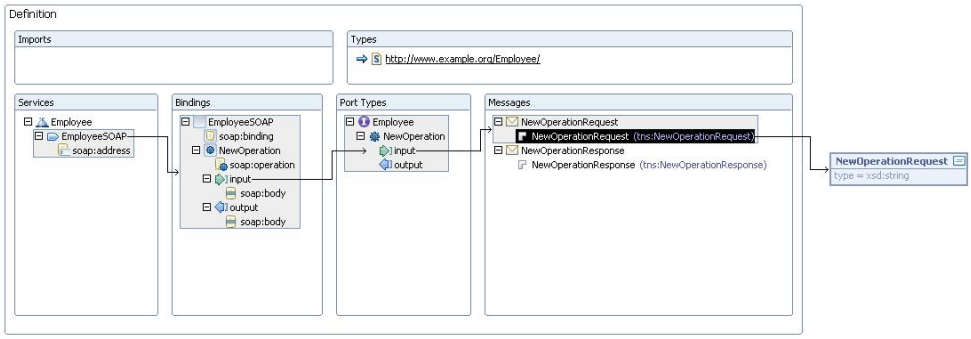

Abbildung 5.3: Der WSDL-Editor von Eclipse

Die Schritte des sogenannten UPnP-Kontrollflusses werden im folgenden detailliert beschrieben.

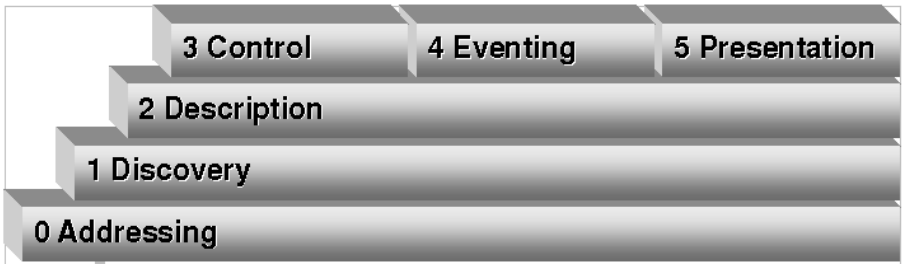

Abbildung 5.4: Der UPnP Kontrollfluß

5.4.1 Adressierung (Addressing)

Da die Basis von UPnP ein IP-Netzwerk ist, muss ein Gerät oder Kontrollpunkt zuerst über eine gültige IP-Adresse verfügen. Dies kann nach dem UPnP-Standard einerseits via DHCP erfolgen, oder via AUTO-IP.

5.4.2 Lokalisierung (Discovery)

Sobald ein UPnP-Gerät über eine IP-Adresse verfügt, muss es seine Existenz im Netzwerk an die Kontrollpunkte melden. Dies erfolgt via UDP über die Multicast-Adresse 239.255.255.250:1900 auf der Basis des SSDP-Protokolls. Ebenso können Kontrollpunkte nach UPnP-Geräten im Netzwerk suchen. In beiden Fällen enthält die *Discovery Message* nur die wichtigsten Angaben ber das Gerät und seine Dienste, wie z. B. den Gerätenamen, Gerätetyp und eine URL zur genauen Beschreibung des Gerätes. Das Device sendet nun periodisch erneute *alive*-Nachrichten. Sollte vor Ablauf der Gültigkeitsdauer keine Erneuerung des SSDP-Nachrichten vollzogen werden, wird davon ausgegangen, dass sich das Gerät nicht mehr (aktiv) im Netzwerk befindet.

5.4.3 Beschreibung (*Description*)

Nachdem ein Kontrollpunkt ein Gerät gefunden hat, holt er sich per HTTP über TCP/IP die Beschreibung des Gerätes von der URL, welche ihm bei der Lokalisierung mitgeteilt wurde. Diese stellt das Gerät in Form eines XML-Dokumentes zur Verfügung. Die Beschreibung beinhaltet Informationen über den Hersteller, die Seriennummer, URL-Adressen für die Steuerung, Ereignisse und die Präsentation. Für jeden Service, den ein Gerät anbietet, werden Kommandos und Aktionen sowie Datentypen und Datenbereiche spezifiziert. Die Beschreibung beinhaltet neben den Diensten, die es anbietet, auch alle eingebetteten Geräte mit deren Diensten. Eine Beschreibung (*Description*) besteht aus einer XML-Datei, die wie folgt aussehen kann:

Listing 5.1: Beispiel einer *Description*

```
1  <?xml version="1.0"?>
2  <root xmlns="urn:schemas-upnp-org:device-1-0">
3    <specVersion>
4      <major>1</major>
5      <minor>0</minor>
6    </specVersion>
7    <URLBase>base URL for all relative URLs</URLBase>
8    <device>
9      <deviceType>
10         urn:schemas-upnp-org:device:deviceType:v
```

```
11      </deviceType>
12      <friendlyName>short user-friendly title</friendlyName>
13      <manufacturer>manufacturer name</manufacturer>
14      <manufacturerURL>URL to manufacturer site</manufacturerURL>
15      <modelDescription>long user-friendly title</modelDescription>
16      <modelName>model name</modelName>
17      <modelNumber>model number</modelNumber>
18      <modelURL>URL to model site</modelURL>
19      <serialNumber>manufacturer's serial number</serialNumber>
20      <UDN>uuid:UUID</UDN>
21      <UPC>Universal Product Code</UPC>
22      <iconList>
23        <icon>
24          <mimetype>image/format</mimetype>
25          <width>horizontal pixels</width>
26          <height>vertical pixels</height>
27          <depth>color depth</depth>
28          <url>URL to icon</url>
29        </icon>
30        XML to declare other icons, if any, go here
31      </iconList>
32      <serviceList>
33        <service>
34          <serviceType>
35                  urn:schemas-upnp-org:service:serviceType:v
36          </serviceType>
37          <serviceId>urn:upnp-org:serviceId:serviceID</serviceId>
38          <SCPDURL>URL to service description</SCPDURL>
39          <controlURL>URL for control</controlURL>
40          <eventSubURL>URL for eventing</eventSubURL>
41        </service>
42        Declarations for other services defined by a UPnP Forum
43        working committee (if any) go here
44      </serviceList>
45      <deviceList>
46        Description of embedded devices defined by a UPnP Forum
47        working committee (if any) go here
48        Description of embedded devices added by UPnP vendor
49        (if any) go here
50      </deviceList>
51      <presentationURL>URL for presentation</presentationURL>
52    </device>
53  </root>
```

Dabei legt die UPnP-Spezifikation für die einzelnen Elemente die folgende Semantik fest:

Elementename	**Verwendung**	**Beschreibung**
root	Erforderlich	
– URLBase	Optional	Definiert eine Basis-URL für spätere relativ angegebene URLs
– device	Erforderlich	

—- deviceType	Erforderlich	UPnP Devicetyp. Bei nicht-Standard Devices - wie in unserem Fall - setzt er sich wie folgt zusammen: urn:DomainName:device:DeviceType:Version (DeviceType max. 64 Zeichen). Der DomainName muss gemäß RFC2141 gebildet werden. Das heisst insbesondere, dass Punkte im Domainnamen durch Bindestriche ersetzt werden. Also: www.uni-dortmund.de -> www-uni-dortmund-de
—- friendlyName	Erforderlich	Name des Gerätes (für den Benutzer, max. 63 Zeichen)
—- manufacturer	Erforderlich	Name des Herstellers (max. 63 Zeichen)
—- manufacturerURL	Optional	URL des Herstellers
—- modelDescription	Empfohlen	Lange Beschreibung (für den Benutzer, max. 127 Zeichen)
—- modelName	Erforderlich	Modellname (max. 31 Zeichen)
—- modelNumber	Empfohlen	Modellnummer (max. 31 Zeichen)
—- serialNumber	Empfohlen	Seriennummer (max. 63 Zeichen)
—- UDN	Erforderlich	Eindeutiger Gerätename. Setzt sich zusammen aus: uuid:UUID, wobei UUID vom Hersteller bestimmt wird.
—- UPC	Optional	Universeller Produkt-Code / EAN-Code
—- iconList	Optional	Kann genutzt werden um Icons fürs Device zu hinterlegen.
—- serviceList	Optional	Enthält eine Liste von Diensten, die vom Gerät angeboten werden.
—— service	Optional	Leitet einen Dienst ein.
——— serviceType	Erforderlich	UPnP Service Type. Darf nicht die Raute (#) enthalten! Setzt sich bei nicht-standard Diensten so zusammen: urn:DomainName:service:ServiceType:Version (ServiceType max. 64 Zeichen). Der DomainName muss gemäß RFC2141 gebildet werden.
——— serviceID	Erforderlich	ID für den Dienst. Muss innerhalb eines Devices eindeutig sein. Syntax: urn:DomainName:serviceID:ServiceID (!ServiceID max. 64 Zeichen). Der DomainName muss gemäß RFC2141 gebildet werden.
——— SCPDURL	Erforderlich	URL zur Dienstbeschreibungdatei (WSDL). Kann relativ zur URLBase sein.
——— controlURL	Erforderlich	URL zum Service-Endpunkt. Kann relativ zur URLBase sein.
——— eventSubURL	Erforderlich	URL für das Eventing. Kann relativ zur URLBase sein.
—- deviceList	Optional	Eingebettete Devices
—— device	Optional	Für jedes Eingebette Device folgt hier ein Abschnitt. Für die Unterelemente gilt das gleiche wie für die device-Element direkt unter root.

—- presentationURL	Empfohlen	URL zur Darstellung des Gerätes für den Benutzer

5.4.4 Steuerung (*Control*)

Anhand der Informationen, die der Kontrollpunkt aus dem Beschreibungsdokument des Gerätes erhalten hat, kann er nun SOAP-Mitteilungen an die Steuerungs-URL des Gerätes schicken, um dieses zu steuern.

5.4.5 Ereignismeldungen (*EventNotification*)

Damit ein Kontrollpunkt nicht dauernd den Zustand eines Gerätes bzw. dessen Statusvariablen abfragen muss (enthalten im Beschreibungsdokument des Gerätes), nutzt UPnP die XML-basierte General Event Notification Architecture (GENA). Mit GENA können Kontrollpunkte Informationen zum Gerätestatus abonnieren; somit werden diese bei jeder Änderung einer Statusvariablen automatisch informiert. Dazu werden Ereignisnachrichten verschickt, die den Zustand der abonnierten Variablen enthält, die sich geändert haben.

5.4.6 Präsentation (*Presentation*)

Die Präsentation ist eine Alternative zur Steuerung und den Ereignismeldungen. Über die *presentationURL*, welche bei der Beschreibung (Description) bekanntgegeben wird, kann mittels Webbrowser auf das Gerät zugegriffen werden. Dies gibt dem Hersteller die Möglichkeit, neben dem standardisierten Zugriff via UPnP eine alternative Benutzeroberfläche zur Verfügung zu stellen.

Im Projekt Roasted Kit wurde die UPnP Implementierung von CyberLink (Version 1.7) benutzt, da sie sowohl für Java als auch C++ verfügbar ist und darüberhinaus kontinuierlich weiterentwickelt wird. In Kapitel 9.1 wird genauer erläutert, auf welche Art und Weise UPnP im Projekt Roasted Kit benutzt wurde. Nachdem unter C++ diverse Probleme mit CyberLink for C++ 1.7.1 auftraten, wurde hier auf das Linux SDK for UPnP Devices 1.2.1 (libupnp) gewechselt.

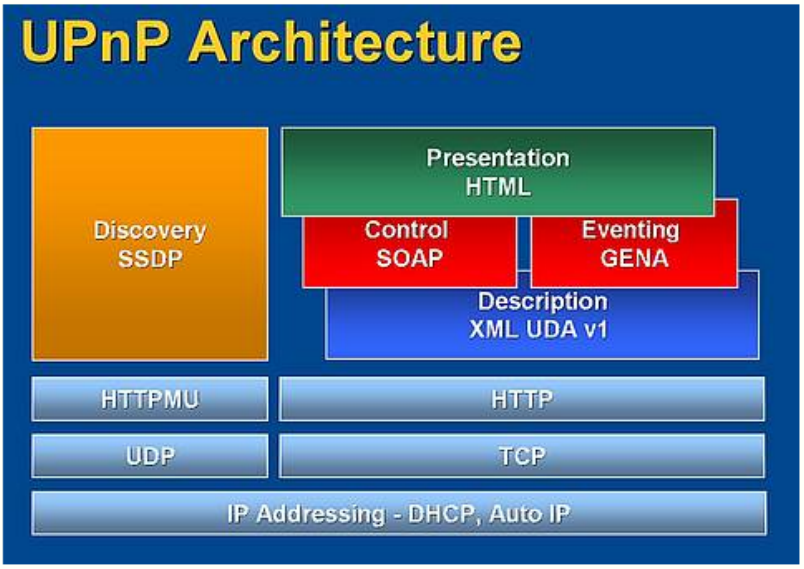

Abbildung 5.5: Die Architektur von „Universal Plug and Play “.

5.5 RTAI (Real Time Application Interface)

Auf den IPCs kommt Linux mit RTAI-3.3 Erweiterung von DIAPM zum Einsatz. Dieses Linux ist speziell auf den IPC zugeschnitten und bietet Unterstützung für alle speziellen Hardwarekomponenten des IPCs (Abb. 5.6). RTAI erweitert den Linuxkernel um Echtzeitfunktionen. Genauer gesagt handelt es sich beim RTAI um einen kleinen Echtzeitkern, den Linux in einem eigenen Task ablaufen lässt. Linux ist dabei der *idletask*, der nur dann aufgerufen wird, wenn kein anderer Echtzeit-*Task* lauffähig ist. Interrupts werden vom RTAI abgefangen und in eine Warteschlange gelenkt. Existiert keine Realtime-Routine, so wird der Interrupt an den Standard-Linuxkernel weitergereicht. Die entsprechende Routine wird dann ausgeführt, um den Interrupt zu behandeln. Unter RTAI werden Realtime-Programme als Module implementiert, was ihre Funktionalität beschränkt, weswegen werden mit RTAI nur die zeitkritischen Teile des Programms implementiert, alles andere wird im Userspace durchgeführt.

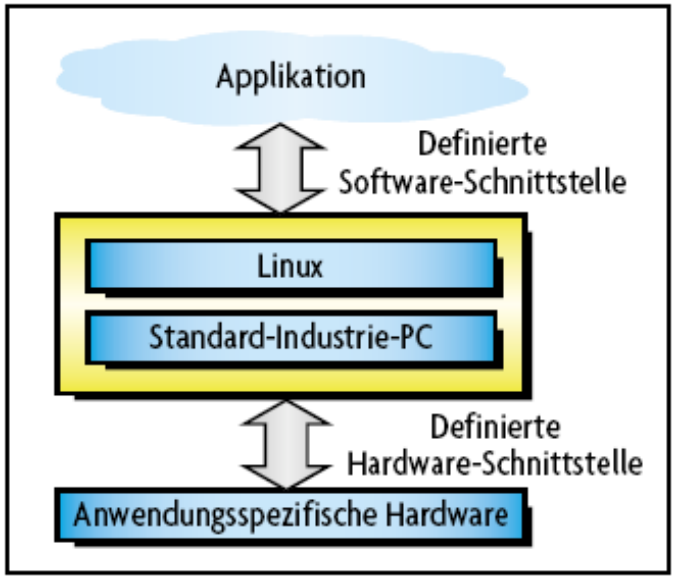

Abbildung 5.6: Einsatz von Linux mit RTAI Erweiterung auf dem IPC

5.6 Linux-BSP (Board Support Packages)

Ein Board Support Package (kurz: BSP) ist eine Hardware-Abstraktionsschicht (HAL), die einem Betriebssystem eine hardwareunabhängige Plattform zur Verfügung stellt. In der Regel besteht ein BSP aus einem Satz Bibliotheksfunktionen, welche meistens vom Hersteller des Betriebssystems für eine bestimmte Prozessorarchitektur zur Verfügung gestellt werden und an die jeweilige Hardware angepasst und erweitert werden müssen. In Kapitel 9.12.1 wird die Architektur näher erläutert.
Das Linux-BSP 2.4.32 (Beta) bietet eine RTOS-fähige Laufzeitumgebung mit K-Bus Unterstützung an. In unserem Fall setzen wir ein BSP für den Thinkio IPC ein, das von der Firma Kontron zusammengestellt worden ist und normalerweise zusammen mit ELinOS benutzt wird. Laut Aussagen der Firma Kontron wurde dieses BSP in enger Zusammenarbeit mit der Firma SysGo entwickelt von der auch das ELinOS Entwicklungstoolkit stammt. Dabei wurde auf einige Bibliotheksfunktionen der nicht freien ELinOS-Bibliothek zurückgegriffen. Dies hat zur Folge, dass man ohne das kommerzielle ELinOS das Kontron-BSP nicht benutzen kann. Dieser Umstand ist natürlich nicht im Interesse von Kontron, da für ihre Kunden zusätzlich Kosten im beträchtlichen Umfang entstehen. Deshalb entwickelt Kontron momentan eine GPL-Version ihres

BSPs. Im Zuge unseres Projekts wurde uns von der Firma Kontron erlaubt, die unveröffentlichte Beta-Version des GPL-BSP zu nutzen. Dafür möchten wir uns an dieser Stelle bedanken.

5.7 Die gSOAP Library

GSOAP 2.7.6 ist eine Bibliothek für den Gebrauch und die Entwicklung von Web Services in C und C++. Das Paket enthält einen *stub and skeleton compiler*, der automatisch die nötigen Routinen zur SOAP-Übertragung erzeugt. Die Library ist sehr schnell und der Speicherbedarf von Web Service-Clients und Servern ist gering(häufig weniger als 150 KBytes).
Um mit gSOAP in C++ zu entwickeln, benötigt man «soapcpp2»und «wsdl2h». Das Programm «wsdl2h» ist ein WSDL Parser, welcher die WSDL in eine gSOAP Header-Datei konvertiert. Das Programm «soapcpp2» kompiliert daraus Stubs und Skeletons. Zur Entwicklung eines Web Service mit gSOAP sollte man die angebotenen Dienst in WSDL spezifizieren. Dazu nutzen die Projektgruppe den WSDL-Editor von Eclipse.
Aus dieser WSDL-Datei kann man nun von gSOAP «wsdl2h»eine gSOAP-Header Datei erzeugen. In dieser Header Datei werden alle in der Schnittstelle verwendeten Typen und Funktionen definiert. Darüberhinaus enthält diese Datei noch spezielle Kommentare, die später vom gSOAP Skeleton-Compiler (soapcpp2) ausgelesen werden. Dazu gehören beispielsweise Namensräume (Abb. 5.7). Aus der gSOAP-Header Datei werden vom gSoap Skekelon-Compiler die Source- und

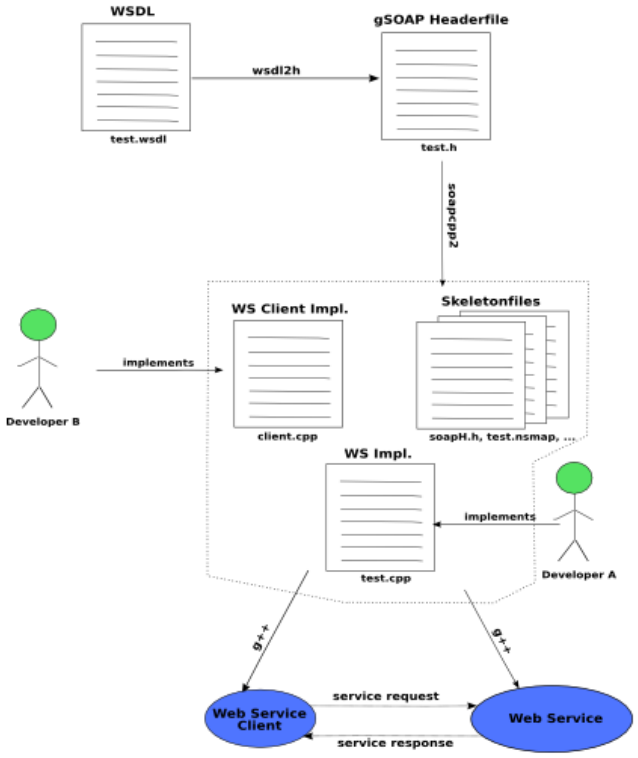

Abbildung 5.7: Anschauliche Architektur von gSOAP

Headerdateien der Skeletons generiert. Die Skeletons enthalten einen Teil der SOAP-Engine, d.h. die Logik, welche das Marshalling und Unmarshalling der Funktionsaufrufe in XML und die Umsetzung in SOAP-Nachrichten übernimmt. Der Rest der SOAP-Engine von gSOAP befindet sich in einer Bibliothek. Der gSoap Skeleton-Compiler erzeugt also ein Quellcode-Gerüst, das später in den Web Service miteinkompiliert wird und bereits einen Großteil der Arbeit für den Entwick-

ler erledigt. Der Entwickler muss lediglich den Code, der den Dienst bzw. den Client ausmacht, implementieren. Abschließend wird aus den generierten und selbst erzeugten Quellcodes der Web Service bzw. Client mit Hilfe eines gewöhnlichen Compilers erzeugt. Da beide Seiten anhand der gleichen WSDL-Datei generiert worden sind, ist die Interoperabilität von Client und Web Service sichergestellt.

5.8 Apache Axis und Jetty

Apache Axis (Apache Extensible Interaction System) ist eine servlet-basierende SOAP Engine für Java, (siehe Kapitel 3), mit der Web Services einfach und einheitlich konstruiert werden können. Roasted Kit nutzte die Version 1.4 "nightly build" von Axis. Für das *Parsen* der XML-Dokumente zeichnet sich Axis vor allem durch seine schnelle Verarbeitungsgeschwindigkeit aus. Zusätzlich werden auch Hilfsprogramme angeboten, die aus einer vorhandenen WSDL Definition ein Source Code Skeleton erzeugen. Aber auch die andere Richtung ist möglich. Ein zusätzliches Feature ist das TCP/IP Monitoring zum Beobachten der übermittelten SOAP-Nachrichten zu Debugzwecken.
Axis wird innerhalb von Jetty 5.1.4 betrieben, der als Servlet Container dient. Jetty ermöglicht ein (Un-)Deployen[1] von Web Services zur Laufzeit. (siehe Kapitel ServiceControl, 9.5) Seine geringe Größe erlaubt es, es leicht in andere Software zu integrieren.

5.9 Apache Derby

Derby ist eine Open-Source-Datenbank API für Java. Derby basiert auf den Standards Java, JDBC und SQL und unterstützt das Client/Server Modell.Eingesetzt wird Derby im Roasted Kit zur Datenhaltung des zentralen LoggingServers, Commonlog Server(siehe dazu Kapitel 9.6).

5.10 Apache Log4J

Mit Log4J bietet die Apache-Group ein Framework zur Ausgabe von Log-Informationen, wie Debugnachrichten oder Warnungen, an. Log4J erlaubt es, die Ausgaben gezielt zu unterdrücken, auf bestimmte Art und Weise zu formatieren oder aber das Ausgabemedium zu ändern. Hierfür sind keine Änderungen am Quellcode des Projekts nötig. Es reicht aus, eine entsprechende Konfigurationsdatei anzupassen. Als Ausgabe für die Log-Informationen kann die Konsole, eine Datei oder eine Datenbank (z.B. die des CommonLog) genutzt werden. Log4J bietet fünf hierarchische LogLevel an: Fatal, Error, Warn, Info, Debug. Hierdurch kann man entscheiden, ab welcher Stufe man Nachrichten ausgeben möchte. Damit die Ausgaben im CommonLog ausgegeben werden, mußte zuerst ein Web Service eingerichtet werden, der die Nachrichten ausgibt. Apache Log4J wird in der Version 1.2.8 benutzt.

5.11 Apache Xerces

Xerces ist ebenfalls ein Projekt der Apache-Group. Xerces kann XML-Dokumente parsen und generieren und wird für Java und C++ zur Verfügung gestellt. In der Java Entwicklung wurde Xerces 2.5.0, bei der Implementierung in C++ Xerces-C++ 2.7.0 verwendet.

[1] Ab- oder Anmelden von Web Services

6 Anforderungen an den Prototypen

Nachdem die vorangegangenen Kapitel zunächst einen Überblick über das Projekt Roasted Kit, die eingesetzten Technologien und die verwendete Förderanlage geboten haben, sollen in diesem Kapitel die Anforderungen an den Prototypen genauer spezifiziert werden. Dabei werden ähnlich wie in einem Pflichtenheft funktionale und nicht funktionale Anforderungen an den Prototypen unterschieden.

6.1 Funktionale Anforderungen

Unter funktionalen Anforderungen sind alle Anforderungen zusammengefasst, die sich direkt auf den Funktionsumfang der Software bzw. der Steuerung beziehen. Die funktionalen Anforderungen unterteilen sich nochmals in Anforderungen, welche die Außenwelt an die Steuerungen hat, und in Anforderungen, welche die einzelnen Komponenten des System untereinander haben.

6.1.1 Außenansicht des Prototypen (Use Cases)

Ein Anwendungsfall (engl. *use case*) bezeichnet laut [11] die Interaktion zwischen Akteuren (in diesem Fall ein umgebendes WMS oder ein Endbenutzer) und dem betrachteten System. Die Anwendungsfälle, unter deren Prämisse der Prototyp des Roasted Kit entwickelt wurde, sind

- **Aufträge verwalten:** Ein Auftrag ist in diesem Kontext als eine Menge von Beförderungs-Jobs zu verstehen, welche das System durchführen soll. Zu jedem Job gehört ein durch eine ID identifiziertes Paket zusammen mit einer Liste von Zwischenstationen (englisch: *Intermediates*), die das Paket nacheinander auf seiner Reise durch das System anfahren soll.
  - **Aufträge registrieren:** Der Akteur gibt über eine fest definierte Schnittstelle Aufträge ins System ein. Für die Komposition von Aufträgen muss dem menschlichen Akteur eine graphische Benutzerschnittstelle zur Verfügung stehen. Die Registrierung erfolgt stets bevor die Pakete in die Anlage eingespeist werden.
  - **Aufträge verfolgen:** Während der Laufzeit des Prototypen müssen die Zustände der im System befindlichen Aufträge von außen nachvollziehbar sein.
- **Pakete ins System einspeisen:** Der Akteur kann zu einem beliebigen Zeitpunkt ein Paket in das System einspeisen, in dem er es auf dem Wareneingang der Förderanlage platziert.
- **System visualisieren:** Der Akteur fragt den aktuellen Systemzustand über eine graphische Benutzeroberfläche ab.
  - **Fördertechnik visualisieren:** Die Komposition und das Aussehen der eingesetzten Fördertechnik muss das System für den Akteur graphisch dargestellen.
  - **Zustand der Steuerung visualisieren:** Das System muss für den Akteur den aktuellen Zustand seiner Komponenten zugänglich machen und auch Möglichkeiten bieten diese Zustände anzupassen.
  - **Technische Zusammensetzung des Systems abfragen:** Der Akteur kann die Details der Fördertechnik vom System erfragen.

- **System steuern:** Der Akteur kann über eine festdefinierte Schnittstelle Steuerungsoperationen innerhalb des Prototypen auslösen. Für einen menschlichen Akteuer sollten diese Steuerungsoperationen über eine graphische Benutzeroberfläche angeboten werden.
  - **System starten:** Der Akteur kann die vorhandene Fördertechnik bzw. die dazugehörenden Steuerungskomponenten über eine festdefinierte Schnittstelle in Betrieb nehmen.
  - **System stoppen/pausieren:** Der Akteur kann die vorhandene Fördertechnik bzw. die dazugehörenden Steuerungskomponenten über eine festdefinierte Schnittstelle stoppen bzw. pausieren.

6.1.2 Innenansicht des Prototypen

Innerhalb des Prototypen, d.h. zwischen den einzelnen Komponenten, haben sich die folgenden Anforderungen herauskristallisiert:

- Für jedes vorhandene Förderelement muss das System on demand (d.h. zur Laufzeit) die benötigten Steuerungskomponenten erstellen und zuweisen.
- Systemkomponenten, die Abhängigkeiten zu anderen Komponenten haben, dürfen erst den Betrieb aufnehmen, wenn sie ihre Kooperationspartner gefunden haben.
- Die Topologieerkundung und Routenbestimmung des Systems muss zur Laufzeit erfolgen, d.h. die Teilnehmer dieser Erkundung werden dynamisch ermittelt.
- Die Ansteuerung des technischen Prozesses muss von einer Komponente gekapselt werden und ähnlich wie bei einem Betriebssystem den anderen über eine festdefinierte Schnittstelle zur Verfügung gestellt werden.
- Um den virtuellen und realen Materialfluß zu synchronisieren, muss die Fördertechnik Sensoren zur Identifizierung (wie z.B. RFID) bereitstellen.

6.2 Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen beziehen sich auf einzuhaltende Gesetze bzw. Normen, Sicherheitsanforderungen sowie Anforderung bzgl. der Plattformabhängigkeit. Bei der Entwicklung des Roasted Kit wurden die folgenden nichtfunktionalen Anforderungen berücksichtigt:

- Das System muss die Sicherheit des technischen Prozesses gewährleisten und beim Auslösen des Notschalters den Betrieb unverzüglich einstellen.
- Die vom System angebotenen graphischen Benutzeroberflächen müssen möglichst intuitiv bedienbar sein.
- Es muss gewährleistet werden, dass jedes in die Anlage eingespeiste Paket nach einer bestimmten Zeit das System (vorzugsweise) über den Warenausgang verlässt.
- Die System-Schnittstellen müssen plattformunabhängig entworfen werden, so dass eine Implementierung auf einer anderen Plattform ohne Weiteres möglich ist.
- Das System muss in seiner Funktion von der zugrundeliegenden Fördertechnik abstrahiert werden, so dass eine Portierung/Erweiterung der Fördertechnik jeder Zeit möglich ist.
- Das System darf nur in Betrieb genommen werden, wenn sichergestellt ist, dass es funktionsfähig ist.

7 Anlagen-Modell

In diesem Kapitel wird ein Modell vorgestellt, mit dessen Hilfe man Materialflusssysteme formalisieren kann. Es wurde im Vorfeld der Systemkonzeption des Roasted Kit entworfen, um die Problemstellung besser greifbar zu machen. Dabei wurde auch die Terminologie verwendet, die bereits in Kapitel 4 vorgestellt worden ist. Dieses Modell ist natürlich nur ein einfacher Ansatz, ein Materialflußsystem zu formalisieren. Die Formalisierung erleichterte jedoch viele spätere Designentscheidungen und half an einigen Stellen ein besseres Gefühl für die Problemstellung zu bekommen.

7.1 Förder-Devices und Förder-Technik

Förder-Devices und -Technik bilden die Atome des Förderanlagen-Modells, das hier entwickelt werden soll. Diese Unterscheidung ist nötig, da man bzgl. einer dezentralen Steuerung unterscheiden muss, ob man eine Hardware-Komponente direkt ansprechen/steuern kann oder nicht. Deshalb unterscheidet man auf unterster Ebene die Förder-Devices und Förder-Technik.

7.1.1 Förder-Devices

Ein Förder-Device ist eine Hardwarekomponente, die mit Hilfe von Software/Treibern **direkt** angesprochen/gesteuert werden kann.

Man unterscheidet zwei Typen von Förder-Devices:

- *aktives* Förder-Device, wie z.B. Motor, Stopper, Weiche *(Aktor)*
- *passives* Förder-Device, wie z.B. Touchsensor/Lichtschranke *(Sensor)*

7.1.2 Förder-Technik

Unter Förder-Technik versteht man die Hardware, die **nicht direkt** mit Hilfe von Software/-Treibern angesprochen/gesteuert werden kann, wie z.B. Rollen, Rutschen usw.

7.2 Förderelemente

Ein Förderelement (*kurz: FE*) ist ein Zusammenschluss von Förder-Devices und Förder-Technik. Es bildet den kleinsten identifizierbaren Abschnitt einer Förderanlage. Jedes Förderelement und seine Funktionen soll durch eine Instanz des DeviceManagements (*kurz: DM*) zur Verfügung gestellt werden. Durch den Aufruf des DMs eines Förderelemets kann die dezentrale Steuerungslogik Steuerungsentscheidungen umsetzen (siehe Kapitel 9.13).

7.3 Förderanlage

Da eine Förderanlage im Grunde ähnlich wie ein Netzwerk oder ein Straßensystem aufgebaut ist, kann man sie mit Hilfe eines Graphen modellieren.

Def: Förderanlage Eine Förderanlage ist ein gerichteter, zusammenhängender Graph $G = (V, E)$, wobei

$$
\begin{aligned}
&V = F \cup Q \cup S && \text{[Knoten des Graphen]}\\
&F = \{f_1, \cdots, f_n\} && \text{[Förderelemente]}\\
&Q = \{q_i \mid i \in N \wedge in(q_i) = \emptyset\} && \text{[Wareneingänge]}\\
&S = \{s_i \mid i \in N \wedge out(s_i) = \emptyset\} && \text{[Warenausgänge]}\\
&E \supseteq (F \times F) \cup (Q \times F) \cup (F \times S) && \text{[Kanten des Graphen]}
\end{aligned}
$$

Hilfsdefinitionen Um die obige Definition zu vervollständigen, benötigt man die folgenden Hilfsdefinitionen:

$$
\begin{aligned}
&\forall v \in V: \quad in(v) := \{(u, v) \mid (u, v) \in E\} && \text{[eingehende Kanten]}\\
&\forall v \in V: \quad out(v) := \{(v, w) \mid (v, w) \in E\} && \text{[ausgehende Kanten]}
\end{aligned}
$$

Mit Hilfe dieses mathematischen Modells ist es nun möglich, konkrete Förderanlagen in eine Graphendarstellung zu überführen. Dies wird am folgenden Beispiel verdeutlicht:

Beispiel Die folgende Abb. 7.1 zeigt einen vereinfachten Aufbau der unteren Etage der Förderanlage vom FLW, und wie man sieht, sind in dieser Abbildung bereits die einzelnen Förderelemente $f_1, \cdots, f_7$ namentlich gekennzeichnet.

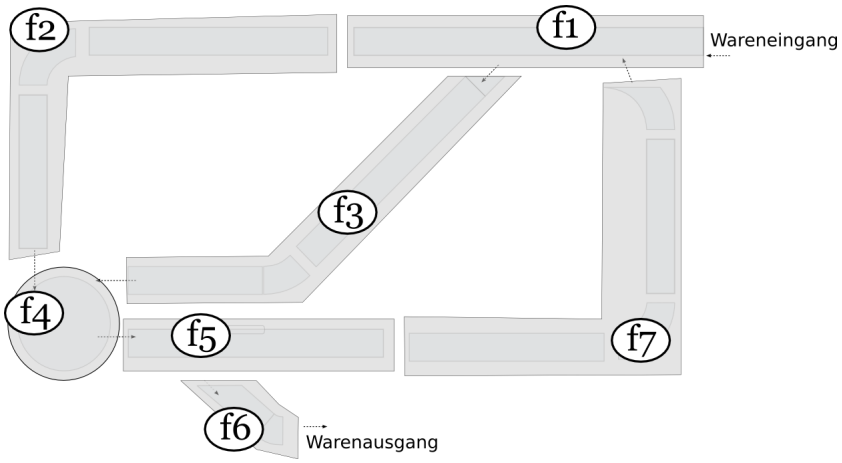

Abbildung 7.1: Vereinfachte Darstellung der unteren Etage der Förderanlage des FLW

Unter Berücksichtigung der oben genannten Definition kann man nun die einzelnen Förderelemente und deren Assoziationen in den folgenden Graphen umsetzen:

$G = (V, E)$, wobei
$V = \{f_1, \cdots, f_7\} \cup \{s, q\}$
$E = \{(f_1, f_2), (f_1, f_3), (f_2, f_4), (f_3, f_4), (f_4, f_5), (f_5, f_6), (f_5, f_7), (f_7, f_1)\} \cup \{(q, f_1), (f_6, s)\}$

In der folgenden Abb. 7.2 wird gezeigt, wie man den oben genannten Graphen visualisieren könnte.

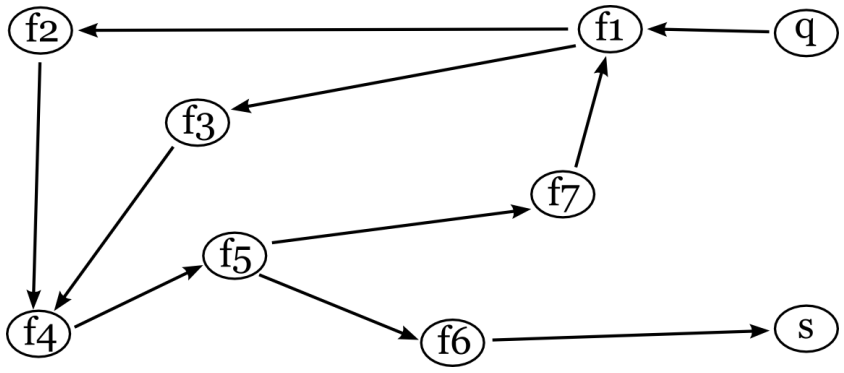

Abbildung 7.2: Graphen-Darstellung der unteren Etage der Förderanlage des FLW

7.4 Entscheidungsknoten und Förderstrecken

Obwohl die oben genannten Definitionen ausreichen, um eine Förderanlage als Graph darzustellen, ist es für die dezentrale Steuerung hilfreich, Förderelemente auf sinnvolle Art und Weise logisch zusammenzufassen. Jedem Förderelement wird eine Instanz des SystemManagement zugeteilt, Förderelementen mit Weichen wird zusätzlich noch eine Instanz des PacketManagements zugeteilt. Wareneingänge bzw. -ausgänge werden darüber hinaus noch mit einer Instanz des OrderManagements versehen. Diese Einteilung erfordert eine Erweiterung des oben definierten Graphenmodells.

7.4.1 Entscheidungsknoten

Ein Entscheidungsknoten ist ein spezieller Knoten mit folgenden Eigenschaften:
$f^* \in F : \mid out(f^*) \mid > 1 \ \vee$
$\exists (q_i, f^*) \in E, \ wobei \ q_i \in Q \vee$
$\exists (f^*, s_i) \in E, \ wobei \ s_i \in S$

8 Architektur-Übersicht

Dieses Kapitel soll dem Leser einen Einblick in die Architektur des Roasted Kit bieten und die dahinterstehenden Konzepte sowie deren Motivation aufzeigen.

8.1 Exkurs: Framework

Bevor auf die Architektur des Roasted Kit eingangen wird, soll zunächst in einem kleinen Exkurs grundlegendes Wissen über Frameworks im allgemeinen vermittelt werden. Wer sich in diesem Gebiet schon sicher fühlt, kann diesen Abschnitt ruhigen Gewissens überspringen.
Der Begriff Framework steht für Gerüst oder Skelett und wird in der Software-Technik für spezielle Softwarebibliotheken verwendet. Frameworks zeichnen sich dadurch aus, dass sie im Gegensatz zu einer einfachen Bibliothek dem Entwickler nicht nur wiederverwendbare Komponenten, sondern auch eine Architektur zur Verfügung stellen. Die Architektur einer mit Hilfe eines Frameworks entwickelten Anwendung wird also bereits durch das Framework vorgegeben. Dies kann den Designaufwand enorm reduzieren und damit zu einer erheblichen Zeitersparnis führen.
Neben der Architektur definiert ein Framework i.d.R auch den Kontrollfluß innerhalb einer Anwendung und legt dessen Schnittstellen fest. Dies führt zu einer Umkehrung der Kontrolle, d.h. der Entwickler implementiert Code, welchen er dann beim Framework registriert und der dann vom Framework ausgeführt wird (vgl. [11]). Ein Beispiel dafür bilden die später vorgestellten Steuerungsmodule (siehe Abschnitt 8.3.2)
Laut [11] haben Frameworks in den meisten Fällen eine konkrete Anwendungsdomäne und sind deshalb meist domänenspezifisch oder auf einen bestimmten Anwendungstyp beschränkt. Beispiele sind Frameworks für graphische Editoren, Buchhaltungssysteme oder elektronische Warenhäuser im World Wide Web.

8.2 Verfolgte Ziele

Während der Entwicklung des Roasted Kit wurden die folgenden Ziele verfolgt (vgl. Kapitel 1):

- Abflachung der klassischen Steuerungspyramide
- Konfiguration statt Programmierung
- Verlagerung von Steuerungskompetenzen aus der Leitebene in die unteren Ebenen
- Bereitstellung von Bausteinen für häufig auftretende Verwaltungsprobleme

Natürlich konnte im Vorfeld nicht für alle während der Entwicklung eines dezentral gesteuerten Materialflusssystems auftretenden Verwaltungsprobleme Bausteine bestimmt und entworfen werden. Die folgenden Verwaltungsprobleme konnten identifiziert werden:

- Auffinden von Partnern.
- Dynamische Bestimmung der Topologie.
- Berechnung von Wegen innerhalb der berechneten Topologie.
- Auftragsverwaltung in einem dezentralen Umfeld.
- Propagation von Ereignissen innerhalb der Steuerung.

8.3 Architektur

Wie bereits des Öfteren erwähnt, basiert das Roasted Kit auf einer serviceorientierten Architektur und stellt dem Entwickler sowohl Whitebox-[1] als auch Blackbox-Komponenten[2] zur Verfügung.

Grundsätzlich unterteilt sich die Architektur des Roasted Kit in

- Generische Komponenten
- Anlagenspezifische Informationen

8.3.1 Generische Komponenten und ihre Dienste

Die generischen Komponenten des Roasted Kit zeichnen sich dadurch aus, dass sie so entworfen und implementiert worden sind, dass sie für ein breites Spektrum von Materialflusssystemen verwendet werden können. Dies wurde durch die Komposition bzw. Formulierung der Komponentenfunktionen als Dienste und durch den Entwurf von festdefinierten Diensteschnittstellen möglich. Durch die Schnittstellen wird aus einer generischen Komponente eine Blackbox, deren Implementierungsdetails dem Dienstnehmer verborgen bleiben. Dadurch wird die Komponente zu einem wiederverwendbaren, spezialisierten Systembaustein.
Das Roasted Kit enthält die folgenden generische Komponenten, welche sich in Form eines Stacks anordnen lassen:

- *Monitoring*
- *OrderManagement*
- *PacketManagement*
- *SystemManagement*
- *DeviceManagement*

An den Stellen, wo eine Anpassung einer Komponente an ihr Anwendungsumfeld von Nöten war, wurde versucht, dies mit Hilfe von Konfigurationsdaten zu vollziehen (siehe Kapitel 8.3.2). Beispiele hierfür finden sich vor allem im Device- und SystemManagement, wo es einer Anpassung an die zugrundeliegende Fördertechnik bedarf.

Im Folgenden werden die einzelnen generischen Komponenten und ihre Aufgaben innerhalb einer mit dem Roasted Kit entwickelten Steuerung kurz erläutert, wobei für Details auf Kapitel 9 und dessen Unterkapitel verwiesen wird:

Monitoring Das Monitoring dient vornehmlich der Darstellung des interpolierten Systemzustands und Visualisierung der Anlage. Da diese Dienste hauptsächlich für menschliche Nutzer interessant sind, wurde das Monitoring in eine graphische Benutzeroberfläche integriert und dient somit auch als Benutzerschnittstelle. Das Monitoring kann deshalb auch als Werkzeug der Leitebene verstanden werden, jedoch ohne die Kapselung von Steuerungslogik.

[1] Eine Whitebox ist in der Software-Technik meist eine abstrakte Klasse, deren Funktionen durch den Entwickler implementiert werden muß.
[2] Eine Blackbox dagegen ist eine fertige Komponente, die ihre Funktionen über eine festdefinierte Schnittstelle anbietet.

OrderManagement Das OrderManagement (kurz: *OM*) ist für die Verwaltung von Aufträgen im dezentralen Umfeld zuständig. Seine Dienste bestehen also maßgeblich in der Annahme, Zerteilung und Weiterleitung von Aufträgen innerhalb der Steuerung. Die dafür definierten Schnittstellen werden im Prototypen von der sogenannte OrderForm - einer graphischen Oberfläche zur Komposition von Aufträgen - genutzt. Auf diese Weise kann ein Benutzer Aufträge ins System eingeben.

PacketManagement Das PacketManagement (kurz: *PM*) beschäftigt sich mit der Pflege von Routingtabellen und der Job- bzw. Paketverwaltung, wobei hierunter vor allem die Wahl des richtigen Weges für die einzelnen Pakete fällt. Zwischen den im System befindlichen PMs wird ein Routing-Algorithmus ausgeführt, der auch im Internet genutzt wird.

SystemManagement Das SystemManagement (kurz: *SM*) bietet Dienste zur Aufbau der Topologie, Interpolation von Paketpositionen zwischen Sensorereignissen und zum Transport von Paketen über einzelne Förderelemente. Des Weiteren stellt das SM eine Laufzeitumgebung für die Steuerungsmodule zur Verfügung (siehe Kapitel 8.3.2).

DeviceManagement Das DeviceManagement (kurz: *DM*) dient der Ansteuerung der Fördertechnik und Propagation von Ereignissen. Das DM kapselt die zu steuernde Fördertechnik und bietet ihre Dienste über einen Hardware-Abstraction-Layer[3] (kurz: *HAL*) an.

8.3.2 Anlagenspezifische Informationen

Die Anlagenspezifische Informationen können im Grunde als Konfigurationsdaten verstanden werden, welche die generischen Komponenten für ihren Einsatz in einer konkreten Steuerung anpassen. Sie unterteilen sich in

- *Steuerungsmodule* (Java)
- *Typeninformationen* (XML)
- *Förderelement-Deskriptoren* (XML)

Steuerungsmodule Da es sich beim Roasted Kit nicht um eine generische Steuerung, sondern ein Framework zur Entwicklung von Steuerungen handelt, muss der Entwickler die eigentliche Steuerungslogik für die einzelnen Förderelemente dem Framework zur Verfügung stellen. Dafür wurde eine spezielle Whiteboxkomponente - das Steuerungsmodul - entworfen (in Form einer abstrakte Java-Klasse), welche der Entwickler mit Leben füllen muss. Die benötigten Steuerungsmodule lädt das Roasted Kit bzw. das SystemManagement on demand. In den Steuerungsmodulen befinden sich hauptsächlich Routinen zur Initialisierung bzw. Deinitialisierung und zur Ereignisbehandlungen.

Typeninformationen Typeninformationen oder Typendefinitionen sind Datensätze, welche statische Informationen wie Hersteller, Abmaße oder die Fördergeschwindigkeit der eingesetzten Fördertechnik beinhalten. Das Format dieser Datensätze ist dabei so flexibel gehalten, dass damit Informationen einfach als Key-Value Paare abgelegt werden können. Die Typeninformationen werden über die sogenannten Typenbibliothek (engl: *TypeLibrary*) dem System zur Verfügung gestellt (siehe 8.4). Abgelegt werden diese Datensätze in XML-Dateien.

[3] Ein HAL ist eine architektonische Ebene größerer Computerprogramme, die dafür sorgt, dass andere Software-Komponenten nicht auf die Spezifikationen der Hardware Rücksicht nehmen müssen. (vgl. [11])

Förderelement-Deskriptoren Ein Förderelement-Deskriptor (engl. *ConveyingElement Descriptor*, kurz *CE Descriptor*) ist eine XML-basierende Konfigurationsdatei und beschreibt die Komposition eines Förderelements. Ein CE Descriptor legt also fest, aus welchen steuerbaren Entitäten ein Förderer besteht, also insbesondere welche Aktoren bzw. Sensoren an einem Förderelement verbaut sind. Für die einzelnen Entitäten werden Typenschlüssel angegeben, welche auf die passende Typendefinitionen verweisen. Des Weiteren enthält ein CE Descriptor auch die Position des Förderelements im Anlagenraum in Form von drei Messpunkten. Durch Nullpunkt und zwei Orientierungspunkte lässt sich die Lage des Förderelements exakt bestimmen.

8.3.3 Topologie einer Roasted Kit Steuerung

Wie man der Abbildung 8.1 entnehmen kann, läuft eine Roasted Kit Steuerung nicht nur auf den in Kapitel 4 beschriebenen IPCs oder Knotenrechner, sondern benötigt zusätzlich sogenannte Steuerungsknoten. Diese Unterscheidung war vor allem aus Performanzgründen notwendig, bedingt durch die Ressourcenknappheit auf den Knotenrechnern. Da jede Kommunikationsbeziehung zwischen Komponenten in Form von Web Services realisiert wurde, ist das Verhältnis zwischen Steuerknoten und Förderelementen variabel, so dass man ein konfigurierbares Maß der Verteilung erzielen kann. Dadurch ist es letztendlich auch denkbar, dass die nächsten, leistungsstärkeren Generationen von Knotenrechnern der gesamten Steuerungslogik als Laufzeitumgebung dienen können.

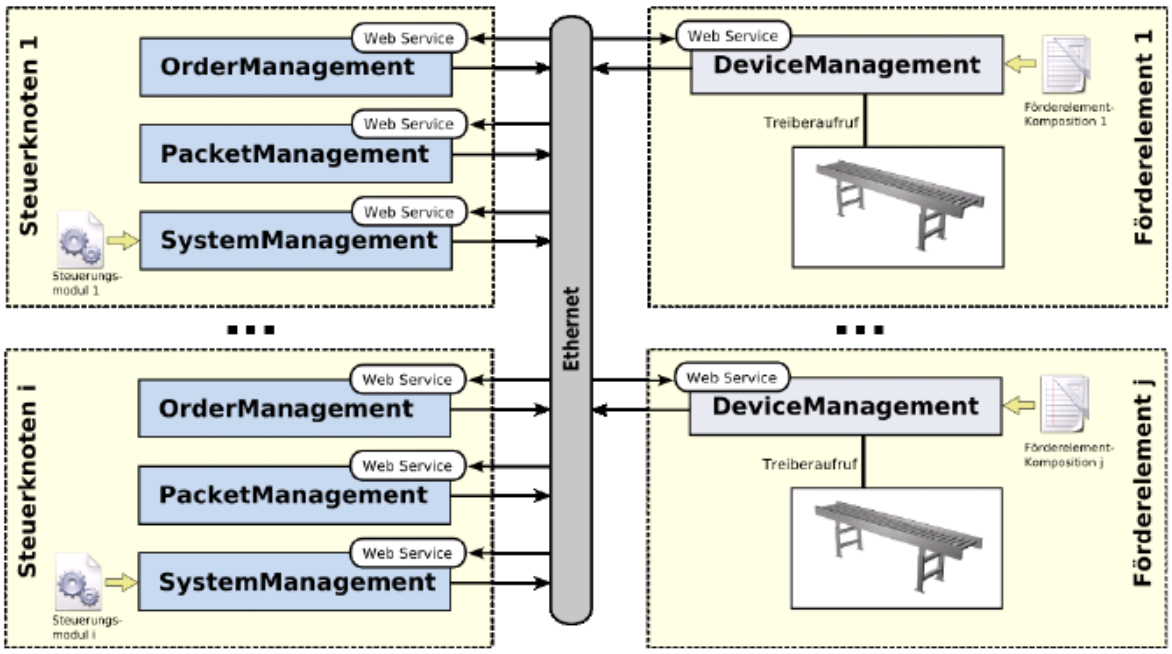

Abbildung 8.1: Topologie einer Roasted Kit Steuerung

8.3.4 Addressierung von Komponenten

Wie man in Abb. 8.1 sieht, bestehen innerhalb des Roasted Kit eine Vielzahl von dedizierten Kommunikationsbeziehungen zwischen generischen Komponenten. Dies erforderte ein eindeutige Adressierung der Komponenteninstanzen innerhalb einer mit dem Roasted Kit entwickelten Steuerung. Dafür wurde ein eigenes Adressierungsschema entworfen - die sogenannte Komponentenadresse (engl: *ComponentAddress*).

Eine Komponentenadresse setzt sich aus Koordinate und Typ zusammen. Der Typ definiert die Art der generischen Komponente (z.B. PacketManagement: PM) und die Koordinate beschreibt die Position dieser Komponente im Raum. Bei Softwarekomponenten ist dies die Position des entsprechenden Rechners, bei Förderelementen logischerweise die Position des Förderelements. Die ComponentAddress des SystemManagements muss zusätzlich zur Position auch noch die Koordinaten der vorhandenen Ein- bzw. Ausgänge des gesteuerten Förderelements enthalten. Diese Koordinaten werden für das Auffinden der Nachbarn benötigt, siehe Kapitel 9.11.2 auf Seite 80. Der Aufbau einer *ComponentAddress* sieht wie folgt aus:

Aufbau:

ComponentAddress = ComponentTyp(x,y,z);[[Port];[Port];...]

ComponentTyp: DM, SM, PM, OM oder OIS

Port: Eingang-/Ausgangskoordinaten des Elements in der Form *I(u,v,w)* bzw. *O(u,v,w)*

x,y,z: die Position im Anlagenraum

Beispiele:

SystemManagement mit einem Eingang und zwei Ausgängen
SM(0.123,1.23,5.21);I(1.34,0,2.3);O(2.82,0.1,1.123);O(4.82,1.1,1.123)

PacketManagement an gleichem Förderelement wie das SystemManagement
PM(0.123,1.23,5.21)

8.3.5 Komponentengerüst

In diesem Abschnitt sollen dem Leser die Atome der Komponenten-Architektur des Roasted Kit vorgestellt werden. Dies ist hauptsächlich für Leser interessant, die sich für die Implementierungsdetails des Roasted Kit interessieren, so dass dieser Abschnitt auch gegebenenfalls übersprungen werden kann.

Die generischen Komponenten müssen an vielen Stellen die gleichen Probleme lösen, wie z.B. sich innerhalb des Netzwerkes bekannt zu machen. Es liegt also auf der Hand, dass Subkomponenten, welche diese gemeinsamen Probleme lösen, universell einsetzbar sein sollten und demzufolge also nicht für jede generische Komponente eigens entwickelt werden müssen. Um diese Subkomponenten objektorientiert greifbar zu machen, wurden sie in einzelne Klassen gekapselt und diese Klassen in passenden Kompositionsstufen zusammengefasst. Will heißen, dass für die generischen Komponenten eine Hierarchie von Basisklassen definiert worden ist. Diese Vererbungshierarchie zusammen mit den dazugehörenden Subkomponenten ist in der Abbildung 8.2 zu sehen und umfasst die folgenden Klassen:

- *Component*
- *SimpleComponent*
- *BaseComponent*

Component Die Klasse *Component* dient als Basis aller Komponenten und stellt Methoden zur Verfügung, um Komponenten und ihre Subkomponenten in einer baumartigen Struktur abzulegen. Dabei orientiert sich die Klasse Component eng an dem *Composition Design Pattern* (vgl. [4]), d.h. innerhalb dieser baumartigen Struktur werden nur Objekte vom Typ Component abgegelegt. Deshalb verwaltet jede Komponente eine Referenz auf ihre Elterkomponente, sowie eine Liste von Referenzen auf ihre Kinder.

SimpleComponent Die Klasse *SimpleComponent* erweitert die Klasse *Component* um die folgenden Subkomponenten:

- *ServiceControl*
- *UPnPControl*
- *Log-Appender*

Außerdem hat jede *SimpleComponent* eine *ComponentAddress*, welche mit Hilfe der Subkomponente UPnPControl in eine Web Service URL aufgelöst werden kann. Weiteres zu den hier genannten Subkomponenten befindet sich im Kapitel 8.3.6.

BaseComponent Die Klasse *BaseComponent* bildet die unterteste Ebene des Roasted Kit Komponentengerüstes und erweitert die die Klasse *SimpleComponent* um die folgenden Subkomponenten:

- *EventControl*
- *StatusControl*

Alle in Kapitel vorgestellten 8.3.1 generischen Komponenten sind vom Typ BaseComponent, wie beispielsweise SM, PM und OM.

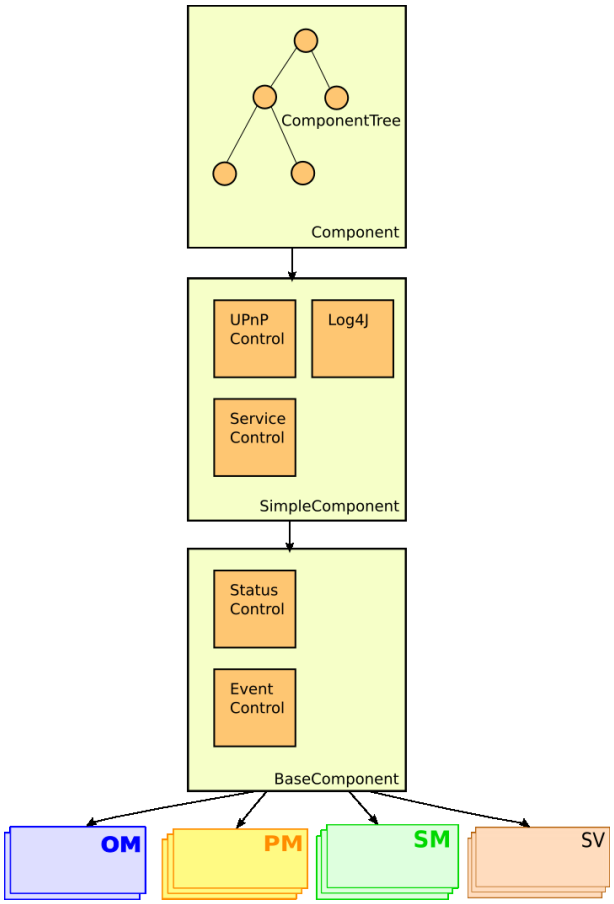

Abbildung 8.2: Die Vererbungshierarchie der Basiskomponenten

8.3.6 Wichtige Subkomponenten

Wie oben bereits erwähnt, gibt es in der Architektur des Roasted Kit verschiedene Funktionalitäten, welche in mehreren Komponenten benötigt werden. Es folgt eine kurze Auflistung der wichtigsten Subkomponenten und ihrer Funktionen bzw. Dienste:

- *EventControl*
- *StatusControl*
- *UPnPControl*

EventControl Die Subkomponente (bzw. Klasse) *EventControl* kapselt die Eventbroker[4]-Funktionalität des Roasted Kit Eventsystems, so dass diese einfach in jede Komponente integriert werden kann. Eine detaillierte Beschreibung des EventSystems ist in Kapitel 9.2 zu finden.

StatusControl Die Subkomponente (bzw. Klasse) *StatusControl* verwaltet den Zustand einer Komponente und regelt den externen Zugriff auf diesen. Welche Zustände eine Komponente unterstützt wird in Kapitel 9.3 genauer erläutert.

UPnPControl Mit Hilfe der Subkomponente (bzw. Klasse) *UPnPControl* kann sich eine Komponente innerhalb des Netzwerkes sichtbar machen und selbst andere Komponente finden. Mit Hilfe der UPnP-Technologie (siehe Kapitel 5.4) wird die *ComponentAddress* und die *Web Service URL* per SSDP-Broadcast mitgeteilt. Auf diese Weise kann *UPnPControl* die *ComponentenAddress* einer Komponente in ihre *Web Service URL* auflösen. Details zu der Subkomponente *UPnPControl* sind im Kapitel 9.1 zu finden.

8.4 Weitere Komponenten der Steuerung

Neben den oben vorgestellten Komponenten umfasst das Roasted Kit noch drei weitere Komponenten, welche für die Entwicklung und den Betrieb einer Steuerung unabdingbar sind. Zum einem den Supervisor, der während der Systeminitialisierung dafür sorgt, dass die benötigten Steuerungskomponenten instanziiert werden und zum anderen das CommonLog sowie die TypeLibrary, zwei globale Komponenten[5] des Roasted Kit.

Supervisor Wie oben bereits erwähnt, handelt es sich beim Supervisor um eine Komponente, welche während der Initialisierung einer Roasted Kit Steuerung dafür sorgt, dass die benötigten Komponenten wie SM, PM und OM entsprechend der Komposition der Fördertechnik erzeugt werden. So wird im Zuge der Systeminitialisierung auf jedem Steuerungsknoten ein Supervisor gestartet, welche die vorhandenen Förderelemente (repräsentiert durch *DeviceManagements*, siehe Kapitel 9.13) untereinander aufteilen. Um diese Aufteilung zu koordinieren wird ein temporärer Server - der Primärsupervisor - gewählt. Details zum Supervisor sind im Kapitel 9.8 beschrieben.

CommonLog Das CommonLog stellt innerhalb des Roasted Kit eine zentrale Sammelstelle für Log-Nachrichten zur Verfügung. Dabei werden die Log-Nachrichten über eine Web Service Schnittstelle entgegengenommen und in einer Datenbank abgelegt.

[4] Im Handel ist ein Broker jemand der zwischen Käufer und Verkäufer vermittelt. In diesem Kontext also zwischen Ereignisquelle und Ergeignisinteressenten.

[5] Globale Komponenten stehen innerhalb einer Steuerung nur einmal zur Verfügung.

TypeLibrary Die TypeLibrary stellt innerhalb einer Roasted Kit Steuerung die Typeninformationen über eine Web Service Schnittstelle zur Verfügung. Auf diese Weise wird vermieden, dass die Datensätze redundant vorhanden sein müssen. Außerdem erhält man auf diese Weise eine transparente Datenhaltung, so dass man falls notwendig die Typeninformationen auch in einer Datenbank anstatt in XML-Dateien unterbringen könnte.

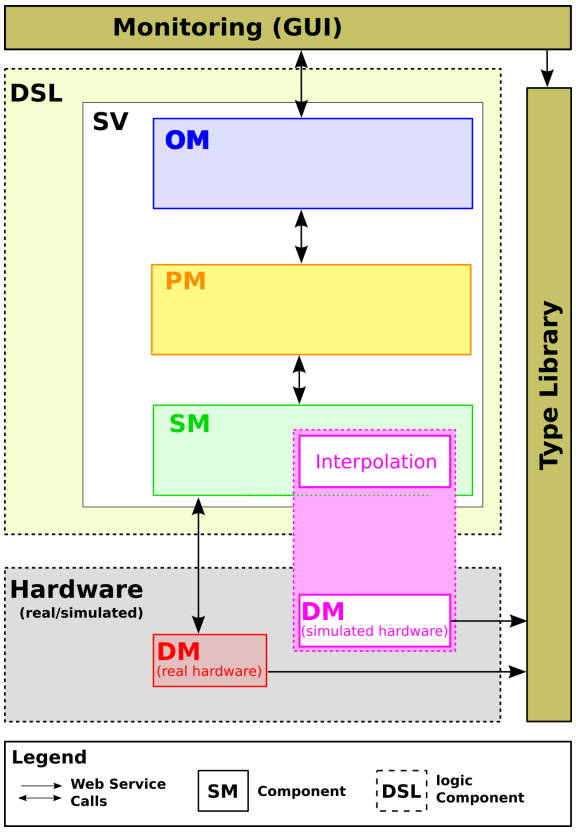

Abbildung 8.3: Schematische Darstellung der Steuerungsarchitektur

8.5 Verwendete BusinessObjects

Zuletzt werden in diesem Abschnitt die wichtigsten BusinessObjects innerhalb des Roasted Kit vorgestellt. Aus dem englischen Wikipedia geht folgende Definition für BusinessObjects hervor:

Business objects are objects in a computer program that abstract the entities in the domain that the program is written to represent. For example, an order entry program needs to work with concepts such as orders, line items, invoices and so on. Each of these may be represented by a business object. Good business objects will encapsulate all of the data and behavior associated with the entity that it represents. For example, an order object will have the sole responsibility for loading an order from a database, exposing or modifying any data associated with that order (i.e. order number, the order's customer account), and saving the order back to the database. Business objects don't necessarily need to represent objects in an actual business (though they often do). They can represent any object related to the domain in which a developer is creating business logic for. The term is used to distinguish between the objects a developer is creating or using related to the domain and all the other types of object he or she may be working with such as user interface widgets and database objects such as tables or rows.

Innerhalb des Roasted Kit werden die folgende BusinessObjects eingesetzt:

8.5.1 ComponentAddress

Alle im System vorhandenen Basiskomponenten benötigen eine *ComponentAddress*. Diese stellt die steuerungsinterne Netzwerkadresse dar (vgl. 8.3.4). Eine *ComponentAddress* enthält die folgenden Felder:

- Kompomententyp: Typ der Komponente, wie z.B. SM oder DM.
- Koordinate: Gibt an wo sich die Komponente im Raum befindet.
- Liste der In-Ports: Enthält die Koordinaten aller Eingänge eines Förderers (nur SM).
- Liste der Out-Ports: Enthält die Koordinaten aller Ausgänge eines Förderers (nur SM).

8.5.2 Order

Eine *Order* ist ein Auftrag, bestehend aus mindestens einem Job, der i.A. durch einen Sachbearbeiter über die Benutzerschnittstelle eingegeben wird. Da Schnittstellen definiert sind, könnten auch externe Systeme (z.B. Warehouse Management Systeme (WMS)) Aufträge anlegen. Eine *Order* beinhaltet folgende Informationen:

- OrderID: Eine eindeutige ID zur Bestimmung des Auftrags
- Status: Der Fortschritt des Gesamtauftrags, zuerst als (new/in progress/done), später als prozentuale Angabe
- Jobliste: Eine Liste von Job-Objekten, mit den jeweiligen Zielen der Pakete

8.5.3 Job

Ein *Job* repräsentiert den Zustellungsauftrag für ein einziges Paket im Rahmen einer *Order*. Dementsprechend beinhaltet ein Job folgende Informationen:

- JobID: Die eindeutige ID eines einzelnen Jobs
- PaketID: Die ID des zum Job gehörigen Pakets
- OrderID: Die ID der Order, zu dem der Job gehört
- Status: Der Status des Jobs (mögliche Werte: new/in progress/done)
- Exit: Der Ausgang, zu dem das Paket geroutet wird
- Intermediates: Die Liste der Zwischenstationen, die vor Erreichen des Warenausgangs angesteuert werden müssen

8.5.4 Packet

Packets sind die virtuellen Abbildungen realer Pakete. Somit vereinen diese virtuellen Pakete neben den Informationen der realen Pendants auch die implizit durch die Physik gegebenen Eigenschaften:

- PaketID: Die eindeutige ID eines einzelnen Pakets
- Höhe: Höhe des Pakets
- Breite: Breite des Pakets

- Tiefe: Tiefe des Pakets
- Gewicht: Gewicht des Pakets
- Orientierung: Orientierung des Pakets. Koordinaten eines Vektors, der in die aktuelle Fahrtrichtung zeigt
- Position: Mittelpunkt des Pakets

Hinweis: Es sei an dieser Stelle erwähnt, dass diese Liste von BusinessObjects nur die wichtigsten umfasst und nicht vollständig ist.

9 Einzelne Subkomponenten

9.1 UPnPControl

Die UPnPControl ist für die Bereiche *Discovery* und *Description* der *UPnP Device Architecture 1.0* zuständig. Die allgemeine UPnP-Architektur wurde bereits in Kapitel 5 beschrieben, in diesem Kapitel wird nur auf die konkrete Implementierung im Roasted Kit eingegangen.
Um sich sowohl selbst zu melden als auch andere Steuerungskomponenten zu finden, implementiert jede UPnP-Control-Klasse sowohl ein *UPnP-Device* als auch einen *UPnP-Control-Point*. Die Aufgabe der UPnPControl ist das Auffinden der einzelnen Förderelemente und Steuerungskomponenten, wie z.B. SystemManagement, PacketManagement, OrderManagement, Monitoring, Type Library u.a. Hierzu wird der *Discovery*-Teil der Cyberlink-Implementierung genutzt. Die UPnPControl verwaltet alle gefundenen Teilnehmer mit ihrer ComponentAddress in einer Liste. Sie bietet nun Methoden an, um sich entweder alle Komponenten liefern zu lassen oder nur die von einem bestimmten Typ (*OM*,*PM*,*SM*). Eine weitere wichtige Funktion ist das Auflösen einer ComponentAddress in die jeweilige Base-Url, die von jeder Komponente genutzt werden kann, um das korrekte Ziel für seine Web Service Aufrufe zu finden. Grundsätzlich melden sich alle UPnP-Teilnehmer in regelmäßigen Abständen (ca. 20 Sekunden), wodurch es UPnPControl möglich ist, festzustellen ob sich eine Komponente nicht kontrolliert abgemeldet hat (z.B. durch ein versehentlich entferntes Ethernet-Kabel).
Die UPnPControl bietet eine wichtige Grundlage für das gesamte System, da ohne sie keine ComponentAddress der jeweiligen URL des Web Services zugeordnet werden kann und somit keine Web Service Aufrufe stattfinden können, wenn nicht die URL anderweitig in Erfahrung gebracht wird.

9.1.1 Schnittstellenbeschreibung

UPnPControl (wie auch die ServiceControl) ist im Vergleich zu den anderen Komponenten ein Spezialfall, denn es bietet nach außen hin keine Schnittstellen an, sondern stellt nur lokal einen Dienst zur Verfügung. Dieser Dienst bietet der UPnPControl beinhaltenden Komponente die Teilnahme am Steuerungsnetzwerk mit Hilfe von Universal Plug and Play (UPnP). Dabei werden keine Steuerungsaufgaben o.ä. übernommen, sondern Basisfunktionen bereitgestellt um die Teilnahme überhaupt zu ermöglichen. Dazu kann man innerhalb des angebotenen Dienstes wiederum drei Subdienste erkennen:

An-/Abmeldung am Steuerungsnetzwerk Dieser Subdienst hat die Aufgabe allen im Steuerungsnetzwerk vorhandenen Komponenten bekannt zu machen, dass eine UPnPControl implementierende Komponente nun neu im Netzwerk vorhandenen ist (Anmeldung) bzw. nicht mehr erreichbar sein wird (Abmeldung), z.B. aufgrund einer expliziten Deinitialisierung.

Informationsabfrage eines Netzwerkteilnehmers Um eine Kommunikation mit anderen Komponenten zu ermöglichen, also deren Dienste zu nutzen, benötigt man diverse Informationen des entsprechenden Netzwerkteilnehmers wie z.B. die Netzwerkadresse (*BaseUrl*). Die Möglichkeit einer Abfrage wird durch diesen Subdienst abgedeckt.

Aktualisierung der Liste aller Netzwerkteilnehmer Wie oben bereits erwähnt kann man explizit nach den im Netzwerk vorhandenen Teilnehmern suchen um die intern verwaltete Teilneh-

merliste zu aktualisieren.

Eine detaillierte Auflistung der UPnPControl-Schnittstelle befindet sich im Anhang B.1 auf Seite 120.

9.2 Eventing

Die Kommunikation innerhalb einer Roasted Kit Komponente und auch zwischen verschiedenen Roasted Kit Komponenten geschied hauptsächlich in Form von Events. Das Eventing ist für die Verarbeitung dieses Nachrichtenverkehrs verantwortlich.

9.2.1 Architektur

Für das Eventing gibt es in jeder Komponente, die von BaseComponent abgeleitet ist, eine Subkomponente EventControl, die die Verarbeitung von Events in dieser Komponente übernimmt. Um auch anderen, externen, Komponenten ein Subscriben und Senden von Events zu ermöglichen, gibt es zu jeder EventControl auch eine Web Service Komponente mit dem Namen EventService, der die benötigten Methoden als Web Service zur Verfügung stellt.
Zwischen den Komponenten und auch innerhalb von Komponenten erfolgt die Kommunikation mit Events.
Ein Event ist dabei eine Klasse, das aus folgenden Komponenten besteht:

- **Topic**: Thema dieses Events. Die Topics sind hierarchisch aufgebaut. Die erste Ebene dieser Hierarchie kann dabei *info*, *warn* oder *error* sein.
- **Timestamp**: Absendezeitpunkt der Nachricht.
- **SenderId**: ComponentAddress des Absenders der Nachricht.
- **EventData-Array**: In dem Feld *EventData* können Zusatzinformationen (wie z.B. die ID eines ausgelösten Sensors) für ein Event mitgesendet werden. Je nach Topic dieses Events können hier Objekte unterschiedlichen Datentyps übertragen werden.

Generell ist es so, daß sich (Sub-)Komponenten per *subscribe()* Aufruf in der EventControl für Events zu bestimmten Topics registrieren können, und sich für diese alle auf einmal per *unsubscribe()* auch wieder abmelden können. Dabei hat man die Möglichkeit das Symbol * als Wildcard zu benutzen und auch Topics anzugeben, zu denen man keine Events erhalten möchte (z.B.: möchte INFO.*, außer INFO.PACKETDETECTED). Man muss generell zwischen 3 Typen von Subscriptions unterscheiden:
(Anmerkung: Im Folgenden bedeutet *extern* "aus der EventControl einer anderen Komponenten" und *intern* "aus einer Subkomponente der eigenen BaseComponent")

- von extern kommende (Un-)Subscriptions
- von intern kommende (Un-)Subscriptions, die nach extern addressiert sind
- von intern kommende (Un-)Subscriptions, die sich für interne Events registrieren

Dazu werden die Subscriptions in drei seperaten, aber gleichen Datenstrukturen gehalten, anhand derer man später entscheidet, welche (Sub-)Komponenten informiert werden müssen:

- Subscriptions von externen Komponenten *(remoteSubscriptions)*
- Subscriptions von eigenen Subkomponenten für externe Events *(delegateSubscriptions)*
- Subscriptions von eigenen Subkomponenten für interne Events *(localSubscriptions)*

Subkomponenten können nun per *fire()* ein Event werfen. Die EventControl überprüft dabei die *remoteSubscriptions* und die *localSubscriptions* auf Subscriptions für das geworfene Event. Dabei wird mit Hilfe zweier Empfängerlisten, worin die ComponentAddresses, bzw. Referenzen auf die Subkomponenten abgelegt sind darauf geachtet, daß ein Event nicht zweimal an die gleiche Komponente gesendet wird. Dazu werden die Subscriptions in delegateSubscriptions und localSubscriptions der Reihe nach wie folgt bearbeitet:

- Nehme einen Eintrag und überprüfe ob das Topic zu dem gegebenen Event unter den explizit ausgegrenzten Topics (*Excludes*) zu finden ist. Wenn JA weiter zum nächsten Eintrag. Wenn NEIN weiter.

- Überprüfe ob das Topic zu dem gegebenen Event unter den gebuchten Topics zu finden ist. Wenn NEIN weiter zum nächsten Eintrag. Wenn JA weiter.

- Überprüfe in der Empfängerliste, ob zu der in der Subscription angegebenen ComponentAddress (*Callback*) (bzw. bei localSubscriptions ein EventHandling Objekt) schon ein Eintrag vorliegt. Wenn JA weiter zum nächsten Eintrag (in der Subscription Datenstruktur). Wenn NEIN weiter.

- Trage in der entsprechenden Empfängerliste die ComponentAddress bzw. die Referenz auf die Subkomponenten ein. Rufe notify() bei der aktuellen (Sub-)Komponente auf. Weiter zum nächsten Eintrag.

Sind alle Einträge durchlaufen kann die Emfängerliste gelöscht werden, da sie nur dazu dient festzuhalten ob eine Komponente das aktuelle Event schon erhalten hat. Per *notify()* eingehende

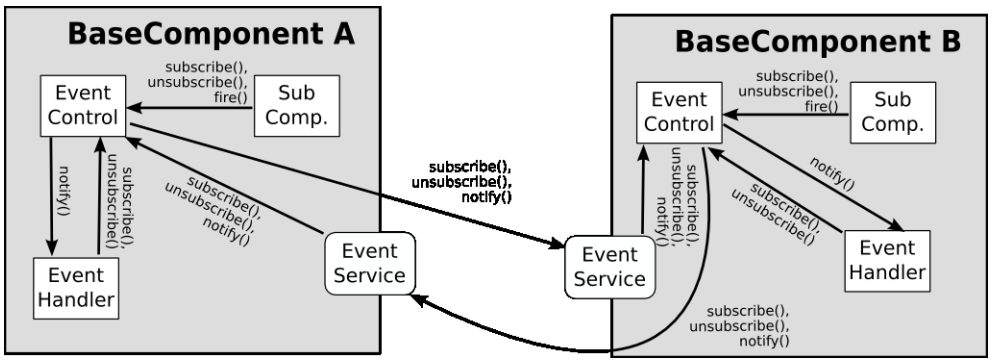

Abbildung 9.1: Schematische Darstellung der Architektur und Kommunikationsbeziehungen

Events werden anhand der delegateSubscriptions an die eigenen Subkomponenten weitergeleitet, die sich dafür registriert haben. Dazu wird in der entsprechenden Subkomponente die Methode *notify()* aufgerufen, welche durch das implementierte EventHandling Interface vorgeschrieben ist.

9.2.2 Schnittstellenbeschreibung

EventService

Der einzige durch die EventControl angebotene Web Service ist der EventService. Dieser dient als Nachrichtendienst und bietet die Möglichkeit, Ereignis-Informationen im System zu propagieren und ein Ereignis somit allen Interessenten zu übermitteln. Der EventService bietet lediglich folgende drei Subdienste an:

Anmeldung für die Benachrichtigung von Ereignissen Jede beliebige Komponente, die selbst ein Eventing beinhaltet, kann sich für Ereignisse bei einer anderen Komponente (die auch ein Eventing beinhaltet) registrieren. Bei Auftreten eines Ereignisses werden somit nur diejenigen Komponenten (genauer: deren EventServices) benachrichtigt, die für das entsprechende Ereignis registriert sind.

Abmeldung für die Benachrichtigung von Ereignissen Das Gegenstück zur Anmeldung ist logischerweise die Abmeldung von Ereignisbenachrichtungen. Komponenten können sich so für bestimmte Ereignisse deregistrieren, so dass sie nicht mehr benachrichtigt werden. Dieser Subdienst wird im Allgemeinen nur genutzt, wenn eine Komponente gewollt abgeschaltet (deinitialisiert) wird.

Mitteilung von Ereignis-Benachrichtigungen Bei Auftreten eines Ereignisse werden allen angemeldeten (registrierten) Komponenten eine Benachrichtigung zugesendet. Diese kann neben der Ereignis-Bezeichnung auch zusätzliche Informationen wie z.B. eine *ComponentAddress* o.ä. enthalten.

Eine detaillierte Schnittstellenbeschreibung ist im Anhang B.2 auf Seite 120 zu finden.

9.3 StatusControl

StatusControl ist eine sehr einfache Komponente, die lediglich für eine übergeordnete Basiskomponente eine Statusinformation verwaltet sowie einen Web Service anbietet um den Status auszulesen bzw. zu setzen. Da in jeder Basiskomponente zu jedem Zeitpunkt eine StatusControl aktiv ist, kann somit stets der aktuelle Zustand erfragt und in angemessener Weise darauf reagiert werden.

9.3.1 Architektur

Die StatusControl verwaltet eine einzelne Integer-Variable. Diese repräsentiert den aktuellen Status einer Komponente. Jeder Komponentenstatus unterteilt sich i.d.R. in drei Stufen bzw. Werte. Der erste Wert eines Zustands ist der Wert, der aktiv von einer anderen Komponente gesetzt wird. Wurde diese Zustandsänderung akzeptiert, wird der Wert um einen Punkt erhöht. Wurde schliesslich der Zustand erreicht, wird der Wert ein weiteres Mal erhöht. Direkt gesetzt wird also jeweils der durch 10 teilbare Wert, die anderen werden nur abgefragt. Es werden folgende Stati definiert:
Die Komponente ist fertig erstellt, aber noch nicht initialisiert:

CREATED = 12 (Ende des Konstruktors)

Die Komponente wird gestoppt, alle Motoren werden sofort angehalten, alle Services außer StatusService pausiert:

STOP = 20
STOPPING = 21
STOPPED = 22

Die Komponente befindet sich in der Initialisierungsphase und startet gerade alle Subkomponenten:

INITIALIZE = 30
INITIALIZING = 31
INITIALIZED = 32

Die Komponente pausiert, existierende Aufgaben werden abgearbeitet, aber keine neuen angenommen:

PAUSE = 40
PAUSING = 41
PAUSED = 42

Die Komponente wird gestartet, nachdem sie im Zustand INITIALIZED, PAUSED oder STOPPED war:

START = 50
STARTING = 51
STARTED = 52

Die Komponente wird vollständig, aber kontrolliert, vernichtet (deinitialisiert):

KILL = 60
KILLING = 61

Auf jede Statusänderung hin wird ein Event STATUS_CHANGED geworfen, welches den neuen Status der Komponente enthält. Jede Basiskomponente muss die Methoden *initialize()*, *pause()*, *stop()*, *start()* und *kill()* implementieren und dort dafür sorgen, dass die für den jeweiligen Zustand oben beschriebenen Aktionen tatsächlich geschehen. Der Supervisor sorgt beim Erstellen der Komponente dafür, dass der Status auf den passenden Wert gesetzt wird. Daraufhin wird in

der jeweiligen Komponente die entsprechende Methode (initialize(), pause(), stop(), start() oder kill()) aufgerufen. Insbesondere die Fördermodule reagieren auf den Status ihrer Vorgänger und Nachfolger, um ihren eigenen Status auf INITIALIZED zu setzen.

9.3.2 Schnittstellenbeschreibung

StatusService

Aufgrund der geringen Komplexität der StatusControl ist auch deren einziger angebotener Web Service - der StatusService - recht trivial: es werden lediglich zwei Subdienste angeboten. Der eine dient dazu den Status der Komponente auszulesen, der andere um ihn zu setzen. Durch das Setzen von Statuswerten kann eine Komponente z.B. dazu veranlasst werden sich zu pausieren bzw. zu deinitialisieren.

Eine detaillierte Schnittstellenbeschreibung wird im Anhang B.3 auf Seite 121 geboten.

9.4 TypeLibrary

Die TypeLibrary gehört zu den globalen Komponenten innerhalb einer Roasted Kit Steuerung und hat die Aufgabe die vorhandenen typenspezifischen Datensätze den anderen Komponenten zur Verfügung zu stellen. Diese Datensätze umfassen Informationen über die Fördertechnik und Förder-Devices (Aktoren/Sensoren).
Die Bibliothek ist aus dem Bedürfnis heraus entstanden, die komplette Beschreibung der Fördertechnik nicht immer mitschicken zu müssen, sondern dies auf eine ID zu beschränken. Die Eigenschaften des Förderelements werden dann über einen Web Service Aufruf an der TypeLibrary unter Verwendung der ID abgefragt. Ein weiterer Vorteil der zentralen Speicherung ist die Möglichkeit neue Datensätze auf einfache Weise hinzufügen zu können, so dass Komponenten wie Interpolation und SystemManagement sofort auf diese zugreifen können.
Die Datenspeicherung wurde in XML realisiert, da die Anzahl der Datensätze sehr gering ist und der Betrieb keine gesonderte Software erfordert. Bei der Entscheidung wurde berücksichtigt, dass falls im Betrieb die Anzahl der Typen signifikant höher ist als bei der vorhandenen Versuchsanlage, ein Wechsel zu einer SQL-Datenbank möglich ist. Da die TypeLibrary über einen Web Service abgefragt wird, macht es für die Clients keinen Unterschied, in welcher Form die Daten abgelegt wurden.
Die Informationen, welche die TypeLibrary verwaltet, wurden zwar in XML abgelegt, jedoch nicht in einer einzigen Datei. Stattdessen wurde für jeden Typ von Fördertechnik und Geräten eine eigene XML-Datei erstellt. Typenbeschreibungen (XML-Dateien) gibt es somit nicht nur für Aktoren und Sensoren, sondern auch für die Fördertechnik selbst. An dieser Stelle soll noch einmal betont werden, dass die Typenbeschreibungen nicht pro Gerät, sondern pro Geräte-Typ angelegt wurden. Gibt es z.B. insgesamt 26 Lichtschranken, aber nur drei verschiedene Typen, so müssen lediglich drei Dateien angelegt werden.
Der grundsätzliche Aufbau einer Typenbeschreibung ist relativ simpel. Ein Gerät - unabhängig ob Fördertechnik oder Förder-Device - wird beschrieben durch mehrere Schlüssel-Wert-Tupel. Ein Schlüssel-Wert-Tupel spezifiziert eine bestimmte Information des Gerätes wie z.B. den Hersteller genauer und besteht aus den drei Feldern *Key*, *Value* und *Mime-Type*. Der *Key* bezeichnet den Schlüssel des Datensatzes, also um welche Information es sich handelt. In dem Feld *Value* wird der zu dem Schlüssel gehörende Wert abgelegt und der *Mime-Type* gibt die Art dieses Wertes an. Folgende Tabelle 9.1 beschreibt einen Förderer in tabellarischer Form: Natürlich dient diese tabellarische Darstellung nur zur einfachen Veranschaulichung der typenspezifischen Informationen. Zur maschinellen Verarbeitung wird wie bereits erwähnt das XML-Format verwendet, wobei ein Schlüssel-Wert-Tupel durch einen sogenannten TLEntry-Tag repräsentiert wird. Der Dateiname der XML-Datei gibt die eindeutige Typen-ID an, so dass dann auf diese Typenbeschreibung referenziert werden kann. Folgendes Listing zeigt eine Typenbeschreibung im XML-Format:

Key	*Value*	*Mime-Type*
Bezeichnung	Förderer AK293567 V1.2z	text/plain
Hersteller	Wago	text/plain
LxBxH	15.23x1.50x2.30	text/plain
3D-Object	Binärdaten in Base64 kodiert	base64/pdf
Sim-Data		text/xml
I/O-Ports	i(-0.5;0.3;0);o(10.52;0.3;0);o(7.3;2.1;0)	text/plain
Datenblatt	Binärdaten in Base64 kodiert	base64/pdf

Tabelle 9.1: Tabellarische Typenbeschreibung eines Förderers

Listing 9.1: Beispiel einer Typdefinition

```
(Hinweis: "_" Bedeutet einen Zeilenumbruch)

<?xml version="1.0" encoding="UTF-8"?>
<tns:TypeDefinition _
xmlns:cemns="http://conveyingElementModel.objects.base._
roastedkit.org/" _
xmlns:tns="http://objects.base.roastedkit.org/TLEntry" _
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" _
xsi:schemaLocation="http://objects.base.roastedkit.org/TLEntry _
../../webapps/wsdl/TLEntry.xsd http://conveyingElementModel._
objects.base.roastedkit.org/ConveyingElementModel.xsd ">
  <TLEntry>
    <Key>Typ</Key>
    <Value xsi:type="tns:StringValue">
      <className>org.roastedkit.base.objects.TLEntry.StringValue
      </className>
      <String>Motor</String>
    </Value>
    <MIMEType>text/plain</MIMEType>
  </TLEntry>
  <TLEntry>
    <Key>Hersteller</Key>
    <Value xsi:type="tns:StringValue">
      <className>org.roastedkit.base.objects.TLEntry.StringValue
      </className>
      <String>SEW-Eurodrive</String>
    </Value>
    <MIMEType>text/plain</MIMEType>
  </TLEntry>
  <TLEntry>
    <Key>Bezeichnung</Key>
    <Value xsi:type="tns:StringValue">
      <className>org.roastedkit.base.objects.TLEntry.StringValue
      </className>
      <String>SF31 DT71C-2</String>
    </Value>
    <MIMEType>text/plain</MIMEType>
  </TLEntry>
</tns:TypeDefinition>
```

9.4.1 Schnittstellenbeschreibung

TypeLibraryService

Die TypeLibrary stellt mit dem TypeLibraryService einen Dienst zur Verfügung, um die strukturiert abgelegten Informationen der TypeLibrary abzufragen. Auch hier kann man wieder zwischen verschiedenen Subdiensten unterscheiden (mit „Entität“ ist im Folgenden TypeLibrary-Objekt gemeint, das ein Gerät repräsentiert, genauer gesagt eine Sammlung von *key-value*-Tupeln):

Abfrage von Informationen einer Entität Möchte man Informationen zu einer bestimmten Entität abfragen, so kann man diesen Subdienst nutzen. Dabei kann man den Umfang der abzufragenden Informationen (also der Schlüssel) angeben, so dass nur ausgewählte Informationen übertragen werden.

Abfrage von Informationen nach Kategorie Mit diesem Dienst können alle Entitäten angefordert werden, die zu einer bestimmten Kategorie gehören.

Anfordern einer Schlüsselbeschreibung Hierdurch kann eine textuelle Beschreibung angefordert werden, die angibt was ein Schlüsselwert bedeutet bzw. für was dieser genutzt wird.

Eine detaillierte Beschreibung der angebotenen Dienste befindet sich im Anhang B.4 auf Seite 122.

9.5 ServiceControl

Die ServiceControl ermöglicht die einfache Bereitstellung (engl.: *Deployment*) von Web Services zur Laufzeit. Dafür wird der HTTP-Server und Servlet-Container Jetty mit der Servlet SOAP-Engine Axis verwendet. Da in jeder Komponente des Systems Web Services auftauchen, ist die ServiceControl Teil der SimpleComponent und somit Teil jeder Komponente. Durch diese hier zu Verfügung gestellten Methoden muss sich kein Web Service-Programmierer um Details der Bereitstellung kümmern, er ruft nur die ServiceControl auf, welche diese Aufgaben automatisch und zur Laufzeit erledigt.

9.5.1 Architektur

Die Web Services können durch die ServiceControl zur Laufzeit *deployed* und *undeployed* werden. Dazu muss lediglich für jeden Service die entsprechende (un-)deploy.wsdd Datei vorhanden sein. Die ServiceControl sorgt dafür, dass Axis den Service nun *deployed* und richtet den Jetty WebServer, in dem Axis läuft, entsprechend ein. Möglich sind hier z.B. verschiedene Timeout-Einstellungen für nicht-funktionierende Web Service Aufrufe. Da die ServiceControl selbst dazu dient Web Services bereitszustellen, bietet sie selbst keine Web Services an und stellt nur eine lokale Programmierschnittstelle zur Verfügung auf die hier nicht weiter eingegangen wird.
Hinweis: Die UPnPControl sollte erst nach der ServiceControl initialisiert werden, ansonsten könnten externe Komponenten schon versuchen, auf die noch nicht fertig initialisierte ServiceControl zuzugreifen.

9.5.2 Schnittstellenbeschreibung

Da die ServiceControl genutzt wird um Dienste bereitzustellen, kann vorher logischerweise kein Web Service nach außen hin angeboten werden.
Diese Komponente bildet ebenso wie UPnPControl einen Spezialfall, da sie selbst keine Dienste nach außen hin anbietet, sondern nur komponentenintern fungiert. Wie bereits erwähnt, übernimmt sie die Bereitstellung (engl.: *Deployment*) von Web Services mit Hilfe der Software Jetty und Axis (siehe dazu Kapitel 5).

Eine detaillierte Auflistung der ServiceControl-Schnittstelle befindet sich im Anhang B.5 auf Seite 122.

9.6 CommonLog

Das CommonLog dient als zentrale Sammelstelle für Log-Nachrichten und ist eine optionale Komponente im System. Durch das zentrale Sammeln der Log-Nachrichten soll das Verstehen der Vorgänge in diesem verteilten System vereinfacht werden, was insbesondere das Auffinden/-Beheben von Fehlern (*debugging*) optimiert.
Das CommonLogs ist als Erweiterung zu Log4J gestaltet, da es bereits die Möglichkeit bietet sogenannte „Appender“ für den eigenen Einsatzzweck zu implementieren. Mit Hilfe der Appender-Funktionalität können dann Log-Nachrichten (zusätzlich zum lokalen Logging) über einen Web Service an das CommonLog weitergeleitet werden. Dazu wird auf Clientseite (den Basiskomponenten) ein WebserviceAppender implementiert, der die Nachrichten über einen Web Service Aufruf zum CommonLog Server überträgt. Die Implementierung umfasst weniger als 10 Zeilen, in der die URL für den Log-Server dynamisch gesetzt und der Aufbau der Verbindung vorbereitet wird. Da der Appender ein Teil des Log4J Systems ist, lässt sich die Schwelle, ab wann geloggt werden soll (Threshold) über eine übliche *.properties*-Datei konfigurieren.

Listing 9.2: Beispiel einer Logausgabe

```
[Feb 06 14:45:31] DEBUG [PM(14.3,-0.09,0.0)]
   (StatusControl.java:100) - Event STATUS_CHANGED fired, value: 52
[Feb 06 14:38:02] INFO  [SV(18.0,18.0,18.0)]
   (ComponentAllocationControl.java:131) - Revisionnumber: 42
```

Der Threshold entspricht der Logging-Hierarchie DEBUG < INFO < WARN < ERROR < FATAL. Daraus ist zu sehen, dass wenn als Log-Level INFO angegeben ist, INFO-Nachrichten und Nachrichten höherer Stufen an den Web Service Appender weitergegeben werden, DEBUG Nachrichten jedoch nicht. Man hat jedoch die Möglichkeit, in der *.Properties*-Datei einen ConsoleAppender mit dem Treshold DEBUG zu definieren, der dann ab dem DEBUG-Level die Nachrichten auf der Konsole ausgibt.
Die Übertragung der Nachrichten wird über einen Web Service Aufruf bei dem CommonLogService realisiert. Da das CommonLog eine globale Komponente im *Roasted Kit* ist, muss es von SimpleComponent erben, um sich über UPnP bekannt machen zu können. Sobald ein CommonLog gefunden wurde, wird die Clientseite des Loggers konfiguriert, so dass die Log-Nachrichten auch an diesen Appender weitergeleitet werden können. In der Praxis stellte sich heraus, dass der Web Service Aufruf für eine einzelne Nachricht zu lange dauert. Aus diesem Grund werden die Nachrichten, die während der Übertragung anfallen, zwischengespeichert, um sie dann alle gesammelt zu übertragen.
Auf Serverseite werden die Daten in einer SQL-Datenbank abgespeichert. Wir haben uns für die Verwendung von Derby (Teil des Apache DB Projekts) entschieden, die vollständig in Java implementiert ist und keine weitere Installation benötigt. Der CommonLog Server kann dadurch auf jedem Rechner innerhalb kurzer Zeit gestartet werden. Zu Beginn wird überprüft, ob die Datenbank vorhanden ist und bei Bedarf angelegt.
Eine weitere Funktion dieses Services ist die Informationsbereitstellung der Log-Nachrichten für den Benutzer durch das Monitoring. In periodischen Abständen oder bei Bedarf werden die Daten vom CommonLog-Server abgerufen und eine Auflistung der Nachrichten in der GUI angezeigt.
Der Vorteil eines zentralen Logging-Systems macht sich im realen Betrieb bemerkbar. Alle Ausgaben laufen zentral zu einer Workstation, von der man direkt den korrekten Ablauf der Software überprüfen kann.
Der CommonLog Web Service umfasst drei Methoden (siehe Abb. 9.2) die zum Speichern und Abrufen der Daten benutzt werden können. Alle 3 Methoden erfordern die Übertragung eines Log-Objektes, das einen Zeitstempel der Nachricht, das Loglevel, den Nachrichtentext, sowie die ComponentAddress des Senders enthält. Um eine Nachricht zu speichern, wird über den Web Service die Methode *log* mit einem Array von Log-Objekten als Parameter aufgerufen. Alle

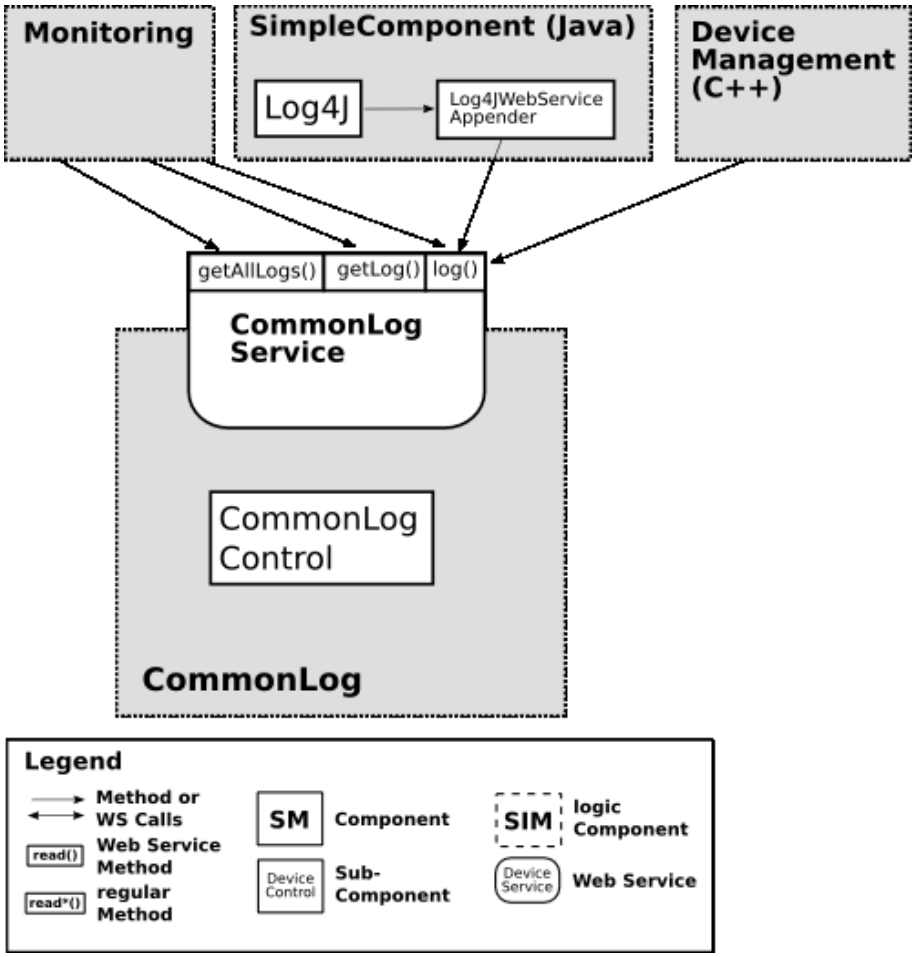

Abbildung 9.2: Schematische Darstellung des CommonLogs

Objekte werden dann auf Serverseite in der Datenbank gespeichert. Zum Abrufen der Informationen kann das Monitoring *getAllLogs(startTime, stopTime, minLogLevel, maxLogLevel)* oder *getLog(component, startTime, stopTime, minLogLevel, maxLogLevel)* aufrufen.

9.6.1 Schnittstellenbeschreibung

CommonLogService

Die Subkomponente CommonLog ist - ähnlich wie die TypeLibrary - nicht sehr komplex und bietet lediglich einen Dienst an: den CommonLogService. Dieser repräsentiert die Schnittstelle des CommonLog nach außen und kümmert sich um die Bereitstellung und Entgegennahme von Log-Nachrichten. Dazu bietet der CommonLogService zwei Subdienste:

Speichern von Log-Nachrichten Um eine oder mehrere Log-Nachrichten (gleichzeitig) zum CommonLog zwecks Speicherung zu übertragen, ist dieser Subdienst zuständig. Um die Log-Nachrichten zu speichern, nimmt der CommonLogService die entsprechenden Nachrichten entgegen und fügt sie lediglich an die interne Log-Liste an; die Komplexität dieses Subdienstes ist somit sehr gering.

Abfrage von Log-Nachrichten Dieser Subdienst bietet die Möglichkeit des Abrufens von Log-Nachrichten in Abhängigkeit von Log-Nachricht-Eigenschaften. Dies realisiert einen Nachrichten-Filter, so dass die abgerufenen Log-Nachrichten selbst bestimmt werden können. So wird bei-

spielsweise die Abfrage aller Log-Nachrichten einer Komponente eines bestimmten Zeitintervalls ermöglicht, deren Log-Level *warn* oder höher ist.

Eine detaillierte Beschreibung des CommonLogService ist im Anhang B.6 auf Seite 123 zu finden.

9.7 Monitoring

Das Monitoring und seine Subkomponenten sind dafür zuständig, dem Benutzer eine graphische Schnittstelle zum System zu bieten. In dieser Schnittstelle wird dem Benutzer nicht nur die Möglichkeit geboten, mit dem System zu interagieren, es werden auch wichtige System- und Zustandsinformationen vermittelt.
Insbesondere wird die reale Anlage emuliert (siehe Kapitel Interpolation, 9.12) und als animierte (dreidimensionale) virtuelle Anlage präsentiert. Dadurch kann der Benutzer nicht nur die Zustände und den internen Ablauf der Steuerung nachvollziehen, sondern direkt mit dem realen Steuerungsverlauf vergleichen.

9.7.1 Architektur

Das Monitoring ist eine komplexe Komponente und unterteilt sich daher wiederum in Subkomponenten um die Komplexität beherrschbar zu machen; siehe dazu Abb. 9.3 . Diese Subkomponenten wurden in eigene und fremde unterteilt. Eigene Subkomponenten werden als diejenigen bezeichnet, die sich ausschließlich um Monitoring-spezifische Aufgaben kümmern, wohingegen fremde Subkomponenten eher unterstützend wirken und somit nur generelle Funktionen wie Nachrichtenvermittlung (*Eventing*) übernehmen. Im Folgenden werden alle Subkomponenten des Monitorings aufgelistet, wobei die eigenen Subkomponenten im nachfolgenden Abschnitt (9.7.2) genauer dargestellt werden. Auf eine detaillierte Beschreibung der fremden Subkomponenten wird an dieser Stellt verzichtet, da sie in den entsprechenden Kapiteln schon dargelegt wurden (Referenzen dazu siehe Aufzählung der fremden Subkomponenten).

Eigene Subkomponenten:

MonitoringControl MonitoringControl dient als Verwaltungskomponente innerhalb des Monitoring und kümmert sich primär um das Bereitstellen der durch SWTControl und Java3DControl benötigten Informationen.

SWTControl Diese Komponente dient als die Schnittstelle zum Benutzer. Hier kann die gesamte Anlage (reale oder virtuelle Förderanlage) gestartet oder gestoppt werden. Des Weiteren können Aufträge in das System eingegeben werden.

Java3DControl Als zweite Benutzerschnittstelle dient die Java3DControl. In einem zweiten Fenster wird die vorhandene Anlage (sofern eine reale vorhanden ist) inkl. diverser Zustandsinformation wie z.B. Position eines Paketes oder ausgelöste Lichtschranken virtuell abgebildet.

MonitoringInterpolation Mit Hilfer dieser Komponente wird ein Teil der Emulation von Fördertechnik in das Monitoring ausgelagert; es hat die Aufgabe eine Animation eines Paketes über die Förderelemente zu erschaffen.

Interne Typenbibliothek Zusätzlich zur normalen Typenbibliothek dient diese interne nur der Performance, also dem Zwischenspeichern (engl.: *Caching*) von Information aus der eigentlichen Typenbibliothek.

Fremde Subkomponenten:

EventControl Verwaltet die Propagation von Ereignis-Nachrichten und bietet dazu den Web Service *EventService* an; siehe Kapitel 9.2.

UPnPControl Macht sich mit *Universal Plug and Play (UPnP)* im Netzwerk bekannt; siehe Kapitel 9.1.

StatusControl Verwaltet den Status für die entsprechende Komponente (hier für das Monitoring) und bietet dazu den Web Service StatusService an; siehe Kapitel 9.3.

ServiceControl Dient der Bereitstellung (engl.: *Deployment*) von Web Services; siehe Kapitel 9.5.

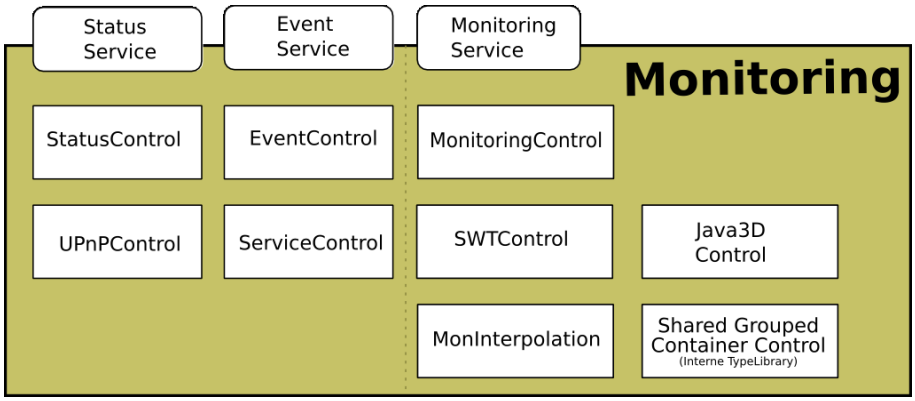

Abbildung 9.3: Schematische Darstellung des Monitorings

9.7.2 Subkomponenten

SWTControl

Das SWTControl ist die eigentliche Schnittstelle zwischen dem Benutzer und dem Monitoring, daher vermittelt sie primär Systeminformationen wie Ereignis-Nachrichten, vorhandene Komponenten, Auftragsinformationen und gefundene Förderelemente inkl. derer Aktoren/Sensoren sowie Typenbibliotheks-Informationen. Des Weiteren bietet die SWTControl dem Benutzer die Möglichkeit mit dem System zu interagieren, d.h. die gesamte (reale oder virtuelle) Anlage ein- bzw. auszuschalten oder die Kameraposition für die durch Java3DControl erzeugte virtuelle Anlage festzulegen. Abbildung 9.4 zeigt die von SWTControl
geschaffene Benutzerschnittstelle im laufenden Betrieb.
Wie man erkennen kann, teilt sich die graphische Benutzerschnittstelle grundsätzlich in drei Bereiche: die Menüleiste, eine Werkzeugleiste und der Bereich zur Vermittlung von Systeminformationen.
In der Menüleiste wird dem Benutzer u.a. die Möglichkeit geboten, virtuelle Pakete (gedacht für den emulierten Betrieb) in das System einzuspeisen oder einfach nur die Benutzerschnittstelle kontrolliert zu beenden. Die Menüleiste ist sehr simpel gehalten, da die wichtigsten Funktionen direkt zugänglich gemacht wurden.
Im zweiten Bereich - der Werkzeugleiste - kann der Benutzer mit dem System interagieren wobei nur der linke Knopfbereich mit dem eigentlichen System kommuniziert - er dient zum Ein-/Ausschalten der Förderanlage. Die beiden rechten Knopfbereiche übernehmen Steuerungsfunktionen bzgl. der 3D-Visualisierung von Java3DControl. Der zweite (mittlere) Knopfbereich übernimmt dabei allgemeine Einstellungen zur Visualisierung (von links: De-/Aktivierung der 3D-Visualisierung, manuelle Steuerung de-/aktivieren, zu Startposition springen, Heranzoomen), der rechte bietet fünf verschiedene Kamerapositionen zwischen denen der Benutzer wählen kann. Der dritte Bereich unterhalb der Werkzeugleiste, der gleichzeitig die größte Fläche einnimmt, dient der Vermittlung von Systeminformationen und teilt sich wiederum in vier Bereiche. Die gesamte rechte Seite zeigt auftretende systemweite Ereignis-Nachrichten (oben) bzw. Komponentenspezifische (unten). Die linke Seite ist jedoch etwas komplexer. So verdeutlicht der obere Bereich dem Benutzer mit Hilfe von Reitern die im System befindlichen Komponenten, Aufträge und

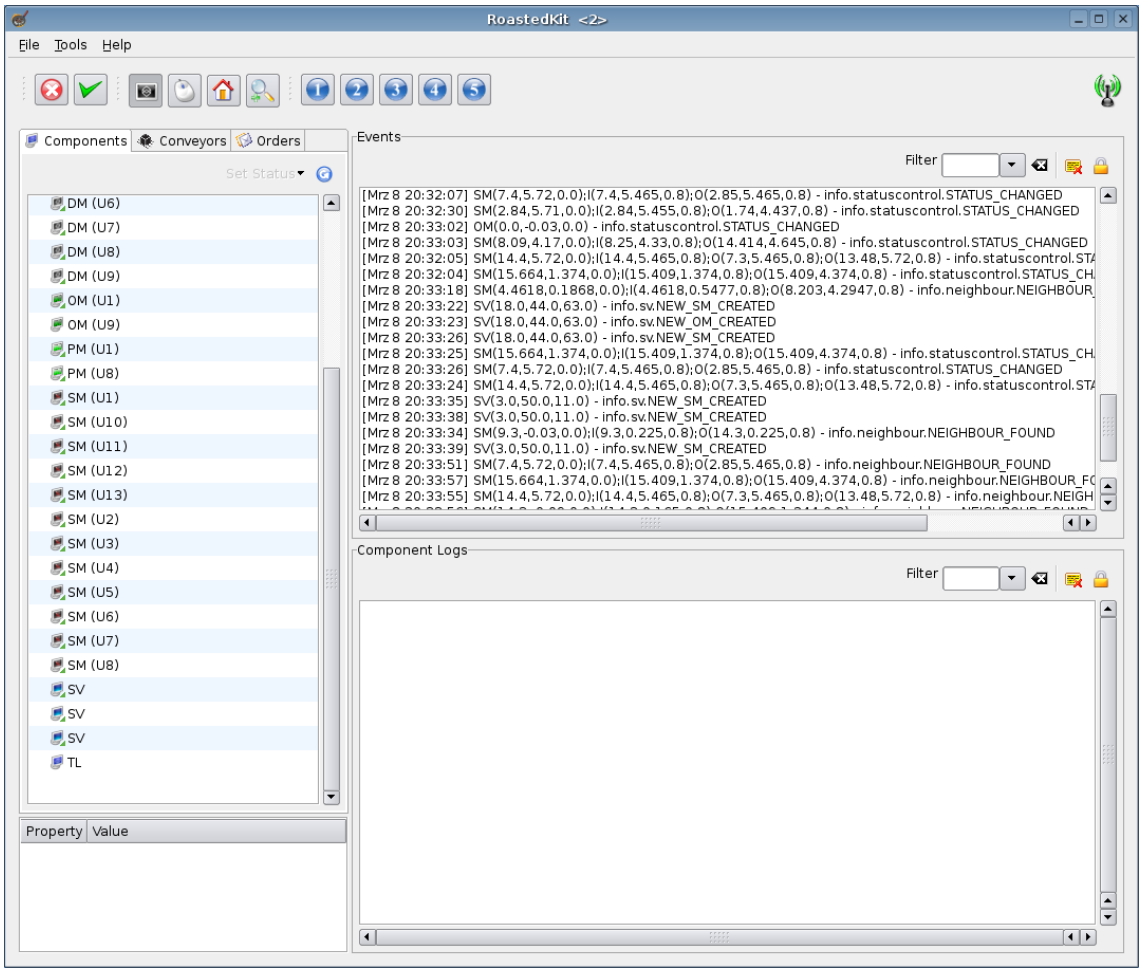

Abbildung 9.4: Die graphische Benutzerschnittstelle SWTControl in Aktion

gefundene Komponenten (inkl. ihrer Geräte wie Aktoren und Sensoren). Man hat hier weiterhin die Möglichkeit, den Status einzelner Komponenten zu setzen oder an ausgewählte Fördertechnik heranzuzoomen. Der untere Bereich vermittelt entsprechende Zusatzinformationen. Die Ausprägung dieser Zusatzinformationen richtet sich nach den ausgewählten Entitäten. Bei Komponenten werden beispielsweise Systeminformationen, wie der aktuelle Status oder die Komponentenadresse angezeigt. Wohingegen bei der Fördertechnik die Informationen aus der Typenbibliothek und bei Aufträgen beispielweise die Auftrags-ID oder die festgelegten Zwischenstationen dargestellt werden.

Java3DControl

Die Java3DControl hat die Aufgabe der Visualisierung der gesamten Versuchsanlage inkl. der Zustände von Sensoren, Weichen und Paketen. Sie bildet Systemzustände in einer Weise ab, die für den menschlichen Benutzer sehr verständlich ist. Da die gesamte Versuchsanlage des Lehrstuhls FLW am Computer (maßstabsgetreu) modelliert wurde und somit digital verfügbar ist (siehe dazu Abb. 9.5), zeigt die Java3DControl eine virtuelle Versuchsanlage, die der realen äußerst ähnlich ist. Weiterhin wurden Paketbewegungen animiert, so dass erst durch diese Animation eine virtuelle Abbildung einer realen Förderanlage entstand. Dadurch kann der Benutzer jederzeit den Systemzustand der Steuerung (virtuelle Anlage) mit der realen Situation vergleichen und evtentuelle Fehlersituation sehr schnell entdecken.
Weiterhin hat man durch die SWTControl u.a. die Möglichkeit, die Kameraposition zu wechseln,

was dem Benutzer Kamerafahrten aber auch ein Heranzoomen an die Sensoren, Aktoren sowie den gesamten Förderelementen ermöglicht.

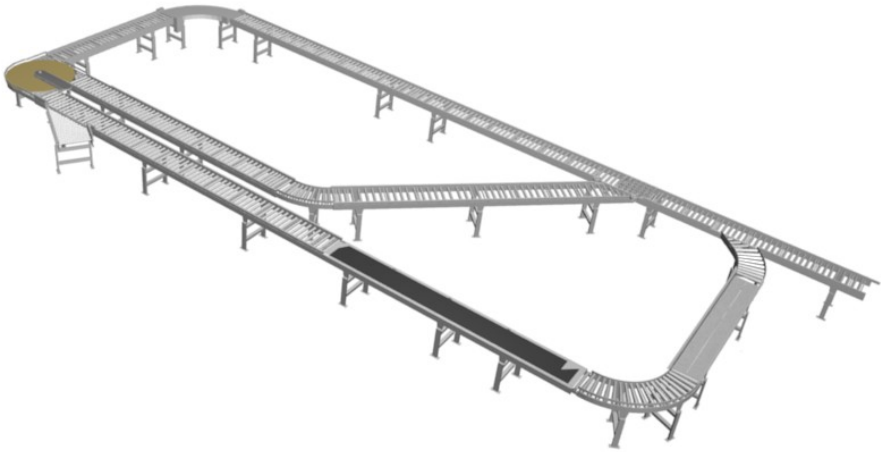

Abbildung 9.5: Gerenderte Darstellung der Förderanlage des Lehrstuhls FLW

MonitoringControl

Die MonitoringControl stellt die wichtigste Subkomponente im Monitoring dar und ist für interne Verwaltungsaufgaben innerhalb des Monitorings zuständig. Neben der De-/Initiliasierung benötigter Komponenten verwaltet sie hauptsächlich die von SWTControl und Java3DControl benötigten Daten. Dazu ist sie die einzige Komponente des Monitorings, die Web Services anderer Komponenten nutzt um diese Daten zu beschaffen. Sie ist somit die Komponente, die alle anderen „verzahnt“ um ein funktionsfähiges Monitoring bereitzustellen. Weiterhin stellt sie selbst den MonitoringService zur Verfügung, um anderen Komponenten eine Kommunikationsschnittstelle zu bieten.

Interne Typenbibliothek

Die interne Typenbibliothek wurde nur aus Performance-Gründen geschaffen und dient zur Zwischenspeicherung (engl.: *caching*) von Informationen aus der Typenbibliothek. Zur Visualisierung benötigt das Monitoring die in der Typenbibliothek gespeicherten (in Base64 kodierten) 3D-Modelle. Gerade die 3D-Modelle sind jedoch verhältnismäßig groß, was ein Laden zu Laufzeit erschwert, da lediglich eine Netzwerkinfrastruktur mit 100MBit pro Sekunde zur Verfügung stand und die Verarbeitung des Datenstroms sehr rechenintensiv ist. Um unnötigen Netzwerkverkehr zu vermeiden, werden die 3D-Modelle in der internen Typenbibliothek zwischengespeichert, so dass erneute Übertragung des entsprechenden Modells nicht mehr erforderlich ist.

MonitoringInterpolation

Die Komponente MonitoringInterpolation ist dafür zuständig jeweils ein Förderelement auf Basis der Geschwindigkeitsangaben aus der Typenbibliothek zu emulieren. Dabei geht es ausschließlich

darum, ein Paket über ein Förderlement zu animieren, so dass es für den Benutzer aussieht als würde das Paket von dem Förderlement transportiert. Da die eigentliche Emulation/Simulation von der Interpolation innerhalb des SystemManagements durchgeführt wird, wurde MonitoringInterpolation geschaffen um nicht jede berechnete Positionsänderung eines Paketes über das Netzwerk übertragen zu müssen. So werden ausschließlich Übergänge zwischen Förderlementen dem Monitoring mitgeteilt, das dann das entsprechende Förderelement „lokal“ emuliert. Dadurch wird die Gesamteffizienz erhöht, da der benötigte Netzwerkverkehr vermindert wurde, was in einer höheren maximalen Bildwiederholungsrate resultiert, deren Grenze lediglich durch die Rechner-Hardware und nicht zusätzlich die Netzwerkinfrastruktur bestimmt wird.

9.7.3 Schnittstellenbeschreibung

Das Monitoring ist eine optionale Komponente und dient im Wesentlichen als Benutzerschnittstelle und kümmert sich dabei um die Visualisierung der vorhandenen Förderanlage. Zur Beschaffung der darzustellenden Informationen werden hauptsächlich die Dienste der anderen Komponenten genutzt, so dass lediglich ein Dienst angeboten wird: der MonitoringService.

MonitoringService

Dieser Dienst ist sehr trivial und dient nur dazu die Paketpositionen der vorhandenen Pakete neu zu setzen. Bei der Dienstnutzung wird eine Paketliste an den MonitoringService übermittelt, der diese an die MonitoringControl weiterreicht. Diese Subkomponente nutzt dann Informationen der Paketliste um die Visualisierung der in der Liste angegebenen Pakete zu aktualisieren.

Die detaillierte Beschreibung dieses Dienstes ist in Anhang B.7 auf Seite 124 dargestellt.

9.8 Supervisor

Der Supervisor (SV) als eine globale Basiskomponente ist ein wichtiges Verwaltungsorgan für die dezentrale Steuerung. Die Hauptaufgaben des Supervisors sind die Identifizierung der Förderelemente (repräsentiert durch DeviceManagments) und die Erstellung der für diese Förderlemente benötigten Systemkomponenten (SystemManagement, PacketManagement und OrderManagement). Somit ist der Supervisor den Systemkomponenten logisch übergeordnet (siehe Abb. 9.6). Der Supervisor muss dynamisch auf Topologie-Änderungen reagieren können und nach Neuhinzukommen oder Wegfall von Förderelementen die Systemkomponenten entsprechend entfernen oder erstellen. Um diese Aufgaben zu erfüllen, verwaltet jeder Supervisor eine sogenannte Collection, die als zentrale Komponente des Supervisors gilt. Die Collection ist eine Liste, die die Zuweisungen der Förderelemente - genauer gesagt deren Systemkomponenten - zu dem entsprechenden Supervisor, aber auch Zusatzinformationen wie Stati, beinhaltet. Um die an verschiedenen Supervisorn befindlichen Collections konsistent zu halten, hat nur ein sog. Primärsupervisor (PSV) Schreibrechte für die Collection. Dieser Primärsupervisor wird von allen Supervisorn gewählt, so dass evtl. Schreibanfragen für die Collection an den Primärsupervisor gestellt werden.

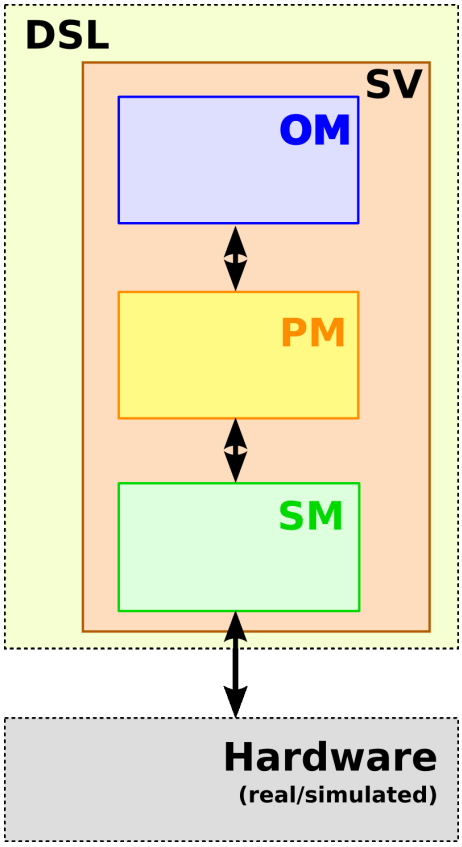

Abbildung 9.6: Supervisor als globale Basiskomponente

Anforderungen Im Detail sind folgende Anforderungen an den Supervisor vorhanden:

- Mindestens ein Supervisor (SV) muss zu jeder Zeit aktiv sein.
- Exakt ein PSV muss zu jeder Zeit aktiv und allen anderen SVs bekannt sein; ist kein PSV vorhanden, werden Wahlen abgehalten.
- Schreibzugriffe in die Collection sind nur vom PSV gestattet.

- Jedes zu steuernde Förderelement muss exakt einem Supervisor zugewiesen sein. Die Anzahl der Zuordnungen sollte nach einer Leistungs-/Peformance-Bewertung der Rechner eines Supervisors stattfinden.
- Alle Systemkomponenten eines Förderelements dürfen nur genau einem Supervisor zugewiesen sein.
- Jeder Supervisor muss eine aktuelle Collection aller vorhandenen Förderelemente besitzen.
- Der für ein Förderlement zuständige Supervisor muss die notwendigen Managementkomponenten initialisieren.

Voraussetzungen Der Supervisor hat folgende Voraussetzungen:

- Die describe()-Methode, die als Web Service vom *DeviceManagement* angeboten wird
- *CommonLog* für Standard-Lognachrichten
- *UPnPControl* zwecks Auffinden anderer Rechner/Supervisor sowie um Informationen über Soft-/Hardware-Ausfälle zu erhalten

9.8.1 Architektur

Notwendigkeit eines Supervisors

Das gesamte Steuerungskonzept basiert auf einer verteilten Architektur - der Rechnerebene und der Steuerebene. Auf der Rechnerebene befinden sich die IndustriePCs, welche fest den Förderelementen (aufgrund der Verkabelung) zugeordnet sind. Die dort vorhandenen DeviceManagments nehmen nur Steuerbefehle von der Steuerebene (z.B. „Motor 5 einschalten“) per Web Service entgegen und führen diese aus. Wer diese Steuerbefehle schickt, ist dabei für die DeviceManagements unwichtig. Somit besteht zwischen der Rechnerebene und der eigentlichen Steuerebene nur eine lose Kopplung und eine Zuweisung der zu steuernden Förderelemente zu den Rechnern der Steuerebene ist notwendig. Ein Ziel des Steuerungskonzeptes ist die dynamische Zuweisung der Förderelemente zu Steuerrechnern während der Laufzeit. Daher wird eine Instanz benötigt, die diese Zuweisung durchführt. Dies ist die Aufgabe des Supervisors.

Architektur

Der Supervisor ist eine komplexe Einheit der Steuerung und wurde in mehrere Subkomponenten unterteilt (siehe Abb. 9.7). Als primäre Subkomponenten gelten davon diejenigen, die ausschließlich genutzt werden um Supervisor-spezifische Aufgaben durchzuführen; dies sind PerformanceControl, VotingControl, CollectionControl und ComponentAllocationControl. Sekundäre Subkomponenten gehören nicht nur speziell zum Supervisor, sondern bieten generelle Basisfunktionen. Dazu zählen EventControl, UPnPControl, ServiceControl, CommonLog und StatusControl. Um die Aufgabe der Zuweisung von Förderelementen zu Supervisorn und die Erschaffung der benötigten Systemkomponenten zu erfüllen, sind viele Probleme zu bewältigen. Da die gesamte Steuerebene dynamisch ist, also Steuerrechner hinzukommen und wegfallen können, muss die Art der Entscheidungsfindung geklärt werden. In der Informatik gibt es für solche verteilte Systeme zwei Möglichkeiten - den zentralisierten und den komplett verteilten Ansatz. Beim zentralisierten Ansatz wird ein temporärer Server gewählt, der alle Entscheidungen trifft bzw. bestätigt. Im komplett verteilten Ansatz muss für die Entscheidungsfindung jeder Kommunikationspartner von jedem anderen eine Bestätigung einholen. In der hier vorgestellten Steuerung wird der zentralisierte Ansatz genutzt, d.h. es wird eine Art Server - der Primärsupervisor - gewählt. Seine Aufgabe ist nicht direkt die eigentliche Entscheidungsfindung, sondern eher die Bestätigung von Zuordnungsanfragen, aber auch die Erhaltung konsistenter Daten. Nur der Primärsupervisor hat

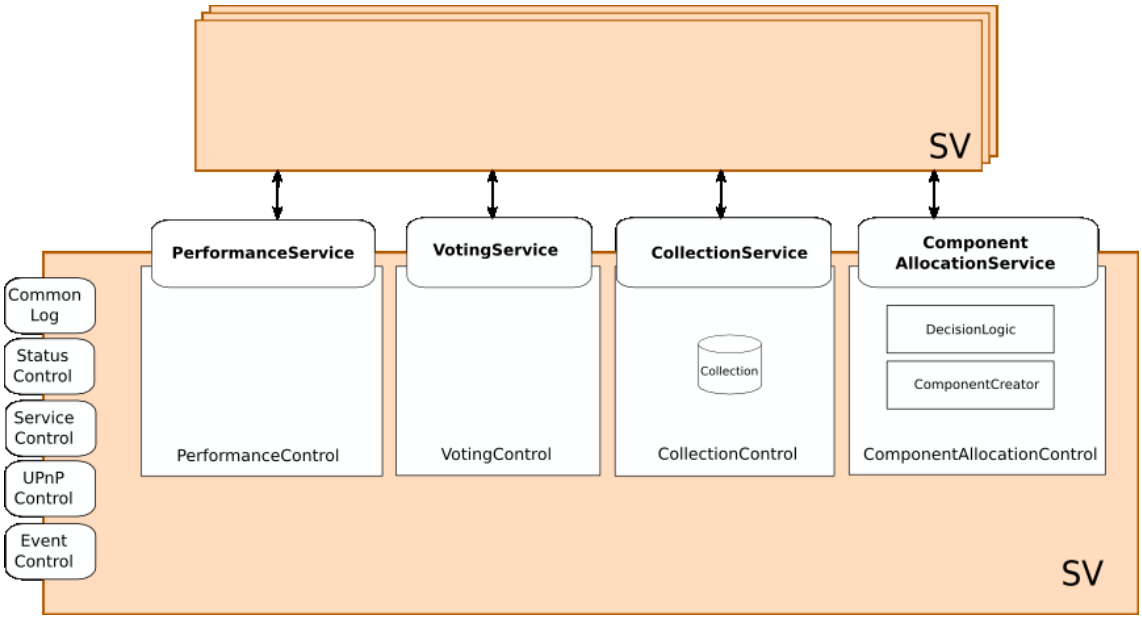

Abbildung 9.7: Schematische Darstellung des Supervisors

Schreibzugriff auf die Collection, also die Datenstruktur, die die Zuordnungen der Förderelemente (repräsentiert durch DeviceManagements) zu den Supervisorn enthält. Möchten normale Supervisor eine Collection-Änderung durchführen, so wird eine Anfrage an den Primärsupervisor gestellt. Änderungen an der Collection müssen also verbreitet werden, damit die Datenhaltung konsistent bleibt.
Die Wahl des Primärsupervisor wird durch die Subkomponente VotingControl durchgeführt. Im Allgemeinen sollte der Primärsupervisor derjenige Supervisor mit der besten Performance sein, also der schnellste Rechner mit der besten Netzwerkanbindung. Die Identifizierung eines solchen "schnellsten Rechners" kann mit Hilfe der PerformanceControl durchgeführt werden, da diese Subkomponente zuständig für die Berechnung eines solchen Performance-Wertes ist. Dieser Wert dient auch als Grundlage eines sogenannten *Utilization*-Wertes, der angibt, wieviele Förderelemente insgesamt übernommen werden können. Desweiteren kümmert sich die ComponentAllocationControl um die Auswahl eines noch nicht zugeordneten Förderelements, die Zuordnungsanfragen an den Supervisor sowie die eigentliche Erzeugung der für die Steuerung benötigten Systemkomponenten. Alle Zuordnungen werden in der Collection abgelegt, die von der CollectionControl verwaltet wird. Diese primären Subkkomponenten des Supervisors werden im Folgenden noch genauer vorgestellt; für eine Beschreibung der oben aufgeführten sekundären Subkomponenten wird auf die anderen Unterkapitel des Kapitels 9 verwiesen.

9.8.2 Subkomponenten

PerformanceControl

Die PerformanceControl ermittelt und verwaltet den Performance-Wert des Supervisors. Dieser Wert gibt die Performance des Rechners eines Supervisors im Bezug zur gesamten Steuerung an. Bei der Berechnung wird zum einen die Geschwindigkeit des Supervisor-Rechners berücksichtigt, zum anderen die Anbindung an das Netzwerk (duch einen Dummy-Web Service Aufruf). Der Performance-Wert wird zudem dazu genutzt, den sog. Utilization-Wert zu berechnen (siehe ComponentAllocationControl).

VotingControl

Die Subkomponente VotingControl hat die Aufgabe Wahlen durchzuführen, um einen Primärsupervisor zu bestimmen. VotingControl hat keine weiteren Subkomponenten; zentraler Bestandteil ist hier der Wahl-Algorithmus. Der durch die VotingControl gewählte Server (Primärsupervisor) muss zu jeder Zeit vorhanden sein, da nur dieser Schreibrechte auf die Collection hat. Somit wird bei Wegfall des Primärsupervisors direkt ein neuer gewählt, damit diese zentrale Verwaltungskomponente - und somit Schreibzugriff für die Collection - möglichst schnell wieder verfügbar ist. Bei der Initialisierung wird allerdings nicht der erste (fertig initialisierte) Supervisor eine Wahl anstoßen und (da er noch alleine ist) sich selber wählen. Stattdessen wird ein Zeitraum von 45 Sekunden abgewartet, bis alle Supervisor initialisiert wurden und dann eine Wahl initiiert.

CollectionControl

Die CollectionControl gilt in Zusammenhang mit der ComponentAllocationControl als das Herzstück des Supervisors. In der CollectionControl wird in erster Linie die Collection verwaltet (siehe unten), in der die Zuordnung der Förderelemente (genauer: die Systemkomponenten der Förderelemente) zu einem Supervisor gespeichert wird. Als Schnittstelle nach außen wird der CollectionService angeboten.

Collection Bei der Collection handelt sich um eine Datensammlung, die die Zuteilung der Systemkomponenten zu den Supervisorn enthält. Schreibzugriff hat dabei nur der durch die VotingControl gewählte Primärsupervisor, alle anderen Supervisor haben nur Lesezugriff; für evtl. Schreibzugriffe müssen sie per Web Service Aufruf eine Anfrage an den Primärsupervisor stellen. Somit wird als Replikationsalgorithmus ein Primary-Copy-Verfahren angewendet. Bei einer Schreib-Anfrage an den Primärsupervisor entscheidet dieser dann, ob die Anfrage abgewiesen oder gestattet wird. Wird sie erlaubt, wird sie in einer sog. *DirtyQueue* gespeichert, die dann von einem weiteren Thread abgearbeitet wird. Dieser führt die entsprechenden Änderungen selber durch und "verteilt" diese an alle vorhandenen Supervisor, um alle Collections konsistent zu halten. Um zwischen den einzelnen Collection-Versionen unterscheiden zu können, wird eine Revisionsnummer verwendet, die bei jeder Änderung um eins erhöht wird. Auch diese darf nur vom Primärsupervisor geändert werden.
Intern wird für die Collection eine Hashmap zum Ablegen der einzelnen Einträge verwendet. Stellt man die Collection als einfache Tabelle dar, so hat jeder Datensatz folgenden Aufbau:

Systemkomponente	Supervisor	Jobliste	Status

- Systemkomponente : *ComponentAddress*

  Mit Systemkomponente sind die Komponenten gemeint, die zur Steuerung eines Förderelementes notwendig sind. Dies sind also SystemManagement, PacketManagement und OrderManagement. Diese ComponentAddress der Systemkomponente ist gleichzeitig der Primärschlüssel dieser Datenstruktur, doppelte Einträge werden somit vermieden.

- Supervisor : *ComponentAddress*

  In diesem Feld wird der Supervisor in Form einer ComponentAddress angegeben, der der Systemkomponente zugewiesen wurde. Ein für Systemkomponenten zuständiger Supervisor hat die Aufgabe diese zu erzeugen, zu deinitialisieren und bei Ausfall neu zu initialisieren.

- Jobliste : *Integer-Array*

Die Jobliste wird nur für temporär vorhandene PacketManagements verwendet; nur dann wird sie mit richtigen Daten gefüllt, ansonsten ist sie "null". In dieser Liste werden die JobIDs der Pakete gespeichert, die noch das Förderelement von dem als Systemkomponente angegebenen PacketManagement passieren wollen. Ist die Jobliste leer, so ist das PacketManagement überflüssig geworden, wird deinitialisiert und der Eintrag aus der Collection entfernt. Durch die Verwendung eines Integer-Arrays ist nicht nur bekannt wieviele Pakete noch das entsprechende PM passieren müssen (bevor es entfernt wird), sondern auch deren JobIDs. Dadurch können diverse Fehlerfälle, wie z.B. das "Verlieren" eines Pakets oder falsches Routing behandelt werden.

- Status : *Integer*

  Der Status gibt den aktuellen Zustand einer Komponente an. So kann man daraus erkennen, ob eine Komponente u.a. fertig initialisiert oder pausiert wurde. Desweiteren wird der Status-Wert u.a. dazu genutzt um Re-/Deinitialisierungen anzustoßen. Die benutzten Statuswerte sind in StatusControl (siehe Kapitel 9.3) beschrieben.

ComponentAllocationControl

Die ComponentAllocationControl ist für zwei wichtige Aufgaben zuständig: Zum einen werden von ihr "noch freie" Förderelemente den vorhandenen Supervisorn zugewiesen, zum anderen erzeugt sie die für die Steuerung benötigten Systemkomponenten (SM, PM, OM) sobald eine Zuweisung eines Förderelementes zu einem Supervisor erfolgt ist. Die faire Verteilung der zu steuernden Förderelemente über alle vorhandenen Supervisor wird über einen sog. Utilization-Wert geregelt. Dieser Wert wird mit Hilfe des Performance-Wertes berechnet und beschreibt, wieviele Förderelemente der entsprechende Supervisor übernehmen kann, ohne dass eine Überlastung des Rechners droht. Um diese Aufgaben zu erfüllen, teilt sich die ComponenentAllocationControl in zwei Subkomponenten, die DecisionLogic und den ComponentCreator.
Die DecisionLogic soll Förderelemente einem geeigneten Supervisor zuteilen. Im Allgemeinen werden noch nicht vergebene, also "freie" Förderelemente immer dem eigenen Supervisor zugewiesen, ist dieser aber ausgelastet, also der Utilization-Wert erreicht, so werden auch Zuweisungen zu fremden Supervisorn durchgeführt.
Um zu identifizieren, welche Förderelemente noch frei und somit nicht zugewiesen sind, wird eine weitere Liste verwaltet, die sich in der CollectionControl befindet. Wurden ein Förderelement und ein Ziel-Supervisor ausgewählt, so wird für diese Zuweisung beim Primärsupervisor eine Bestätigung eingeholt. Bei einer Bestätigung trägt der Primärsupervisor die Änderung (also die Zuweisung) in seine lokale Collection ein, erhöht die Revisionsnummer und verteilt die Collection-Änderung an alle Supervisor. Existiert für die Komponente noch kein Collection-Eintrag, so wird dies gleichzeitig als Aufforderung zur Komponentenerzeugung betrachtet. Jetzt wird die Subkomponente ComponenentCreator aktiv. Diese kümmmert sich um die Erschaffung der benötigten Systemkomponenten. Als erstes erfragt der ComponentCreator beim DeviceManagement die zur Steuerung benötigten Systemkomponenten. Dabei wird ein Integer-Wert zurückgeliefert; die möglichen Werte sind 1 (nur SystemManagement), 2 (SystemManagement und PacketManagement) und 3 (SystemManagement, PacketManagement und OrderManagement). Dadurch sind die benötigten Informationen vorhanden, so dass daraufhin die entsprechenden Systemkomponenten jeweils als eigener Thread erzeugt werden können.

9.8.3 Schnittstellenbeschreibung

Der Supervisor ist vergleichsweise sehr komplex und bietet vier verschiedene Dienste an:

- PerformanceService
- VotingService

- CollectionService
- ComponenentAllocationService

PerformanceService

Im jedem Supervisor wird wie oben erwähnt, ein Performance-Wert verwaltet. Um diesen zu ermitteln, sowie um ihn duch externe Komponenten abfragen zu können ist der PerformanceService zuständig.

VotingService

Der VotingService spielt eine wesentliche Rolle bei der Ermittlung des Primärsupervisors, da über diesen die Wahlen abgehalten werden. Die zwei wesentlichen Subdienste sind hier das Initiieren einer Wahl sowie das Abfragen eines bisher gewählten Primärsupervisors.

CollectionService

Dieser Dienst dient zur Synchronisierung der Collection des Primärsupervisors mit denen aller anderen Supervisor. Hier kann man zwischen folgenden zwei Subdiensten unterscheiden:

Anfrage auf eine Collection-Änderung Möchte ein Supervisor eine Änderung an der Collection durchführen, so nutzt er diesen Dienst am Primärsupervisor. Anschließend entscheidet der PSV, ob die angeforderte Änderung an der Collection durchgeführt wird.

Durchführung einer Collection-Änderung Dieser Subdienst wird ausschließlich von dem Primärsupervisor genutzt um eine Änderung der Collection (genauer: eines Collection-Eintrags) zu verbreiten.

ComponenentAllocationService

Mit Hilfe des ComponenentAllocationService kann man temporäre PacketManagements anfordern, die dann für die jeweils angegeben Förderelemente erzeugt werden und als Intermediate - also eine Zwischenstation - dienen. Des Weiteren bietet dieser Dienst die Möglichkeit der Auslastungsabfrage des Steuerrechners.

Eine detaillierte Beschreibung aller angebotenen Dienste befindet sich im Anhang B.8 ab Seite 125.

9.9 OrderManagement

Das OrderManagement verwaltet die im System vorhandenen Aufträge. Außerdem bietet es die Schnittstelle zum Benutzer. Ein Benutzer kann hier sowohl ein Mensch sein, der per Hand Aufträge eingibt, oder ein Warehouse Management System, welches den vom OrderManagement angebotenen Web Service OrderService nutzt und die Aufträge ins System einspeist. Das OrderManagement nimmt diese Aufträge an, zerlegt sie in Jobs und leitet sie an das PacketManagement, das auf dem gleichen Knoten läuft, weiter. Außerdem wird der jeweilige Status der Aufträge verwaltet.
Eine OrderManagement-Komponente ist auf jedem Wareneingang und jedem Warenausgang vorhanden, den Weg der Pakete zwischen Ein- und Ausgängen kennt es jedoch nicht. Wie alle Basiskomponenten enthält das OrderManagement die zuvor beschriebenen Subkomponenten EventControl, StatusControl und ServiceControl.

9.9.1 Architektur

Wie in Abb. 9.8 dargestellt, ist das OrderManagement aus folgenden Subkomponenten aufgebaut:

OrderRegistration
Diese Subkomponente bildet die eigentliche Verwaltungsinstanz der Aufträge vor allem, weil alle über den OrderService eingehenden Aufträge zunächst bei ihr verarbeitet werden.

OrderDecomposition
Diese Subkomponente zerlegt die eingehenden Aufträge in ihre Jobs und sendet diese Jobs per Web Service Aufruf an das PacketManagement auf demselben Knoten weiter.

OrderEventHandler
Diese Subkomponente verarbeitet eingehende Events des Eventing-Systems (siehe Kapitel 9.2).

9.9.2 Subkomponenten

In diesem Abschnitt wird genauer auf die Aufgaben und Funktionen der einzelnen Subkomponenten eingegangen. Behandelt werden nur die OrderManagement-spezifischen Subkomponenten, wobei auf die allgemeinen Subkomponenten jeweils in einem Extrakapitel eingegangen wird.

OrderRegistration

Die Schnittstelle des OrderManagements, der Web Service OrderService, bietet die im Folgenden beschriebenen Methoden an. Diese werden von den regulären Methoden mit Leben gefüllt. Die Subkomponente OrderRegistration stellt also quasi die interne *Controller*-Klasse des OrderService dar.

addOrder Hiermit kann ein neues Orderobjekt ins System eingegeben werden

deleteOrder Hiermit kann ein Orderobjekt gelöscht werden

getOrderList Diese Funktion liefert die aktuelle OrderList zurück

Verwaltung von Aufträgen Nachdem eine Order durch die OrderForm (siehe Kapitel 9.9.3) oder einen anderen Dienstnehmer hinzugefügt wurde, wird diese in der OrderList gespeichert. Die Verwaltung der OrderList findet in der OrderRegistration statt.

Eine Order kann folgende Stati durchlaufen:

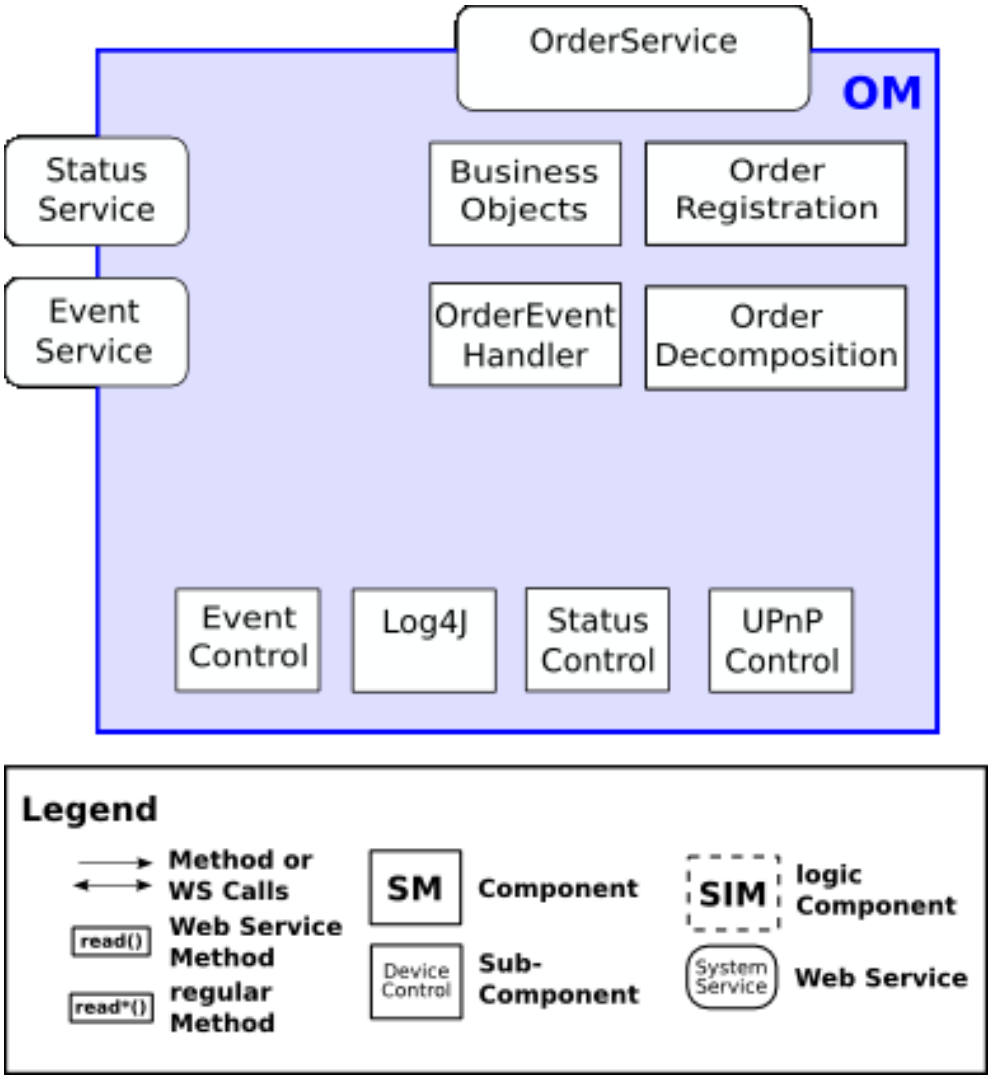

Abbildung 9.8: Schematische Darstellung des OrderManagements

- *new*, wenn die Order erstmalig in das System eingegeben wurde. Der Status der Order wird nun auf *registered* gestellt und an die anderen OrderManagements weitergeleitet. Außerdem werden alle Intermediates einer Order an den Supervisor weitergeleitet, indem bei diesem allocateIntermediate() aufgerufen wird. Der Supervisor erstellt aus dieser Information bei Bedarf flüchtige PacketManagements.
- *registered*, wenn es von einem anderen OrderManagement kommt. Bei dem Status *registered* muss addOrder() nicht erneut aufgerufen werden.
- *in progress* mit Prozentangabe, wenn Jobs teilweise bereits am Ziel sind.

Das neue Orderobjekt wird an die Subkomponente OrderDecomposition weitergereicht und dort weiterbehandelt. Erreicht ein Paket sein Ziel, wird vom PacketManagement der Status dieses Jobs auf *done* gesetzt, das OrderManagement erhält das Event JOB_DONE und sorgt dafür, dass der Gesamtstatus der Order angepasst wird, nachdem dieses Event an alle OrderManagements geschickt wurde.
Über den OrderService kann mit Hilfe der Methode *getOrderList()* zusätzlich die aktuelle Liste anderer OrderManagements geholt werden, um sie mit der eigenen zu synchronisieren. Hat das letzte Paket einer Order sein Ziel erreicht, wird die Order gelöscht.

Wurde ein Auftrag zur OrderList hinzugefügt oder entfernt, wird ein entsprechendes Event geworfen, damit das Monitoring die aktuelle OrderList mitbekommt und darstellen kann.

OrderDecomposition

Diese Subkomponente sorgt für die Zerlegung von Aufträgen in Jobs und registriert diese im PacketManagement. Die OrderDecomposition bekommt per Aufruf der Methode *splitOrder()* der eigenen OrderRegistration eine Order übermittelt. Sie zerlegt diese Order nun in einzelne Jobs, und leitet diese per Web Service (*pushJob()*) an das eigene PacketManagement weiter. Anschließend wird der Auftragsstatus auf *in progress* gesetzt.

OrderEventHandler

Diese Subkomponente registriert sich bei der EventControl seines OrderManagement für alle auftretenden Events und Fehler; dazu implementiert sie das Interface *EventHandling*.

9.9.3 OrderForm

Die OrderForm bildet eine grafische Oberfläche für den OrderService - die Schnittstelle nach Außen. Mit ihr ist es möglich Aufträge in das System einzuspeisen, sie ist als eigenständige SWT-Java-Applikation programmiert. Die OrderForm kann separat an jedem beliebigen Rechner gestartet werden, auch mehrere gleichzeitig sind möglich. Sie sucht sich automatisch ein OrderManagement, an das es dann die eingegebenen Daten schickt. Der Benutzer gibt in der OrderForm alle benötigten Daten einer Order ein, und diese wird im System mit den entsprechenden ID's gespeichert.
In die OrderForm kann der Benutzer folgende Daten eingeben:

- Anzahl der zum Auftrag gehörenden Pakete.
- Für jedes Paket die Paket-ID aus einer Liste der real existierenden Paket-IDs (also die IDs der RFID-Tags der Pakete).
- Für jedes Paket den Wareneingang.
- Für jedes Paket den Warenausgang aus einer Liste der im System gestarteten Ausgänge.
- Für jedes Pakete optional Intermediates aus einer Liste der im System gestarteten Förderelemente.

Die Listen, aus denen man Eingang, Ausgang und Intermediates wählen kann, werden mit Hilfe der von der UPnPControl ermittelten Daten gefüllt. Aus den eingegebenen Daten wird eine *Order* erstellt und diese mit der Methode addOrder() des OrderService ins System eingespeist. Der Status des Auftrags wird hierbei auf *new* gesetzt.

9.9.4 Schnittstellenbeschreibung

OrderService

Der OrderService ist der einzige durch das OrderManagement angebotene Dienst und bietet eine Auftragsverwaltung für das Steuerungssystem an. Somit kann man zwischen folgenden drei Subdiensten unterscheiden:

Eingabe von Aufträgen Dieser Subdienst ist der wichtigste des gesamten OrderService, denn durch ihn legt der Benutzer (entweder der Mensch durch eine grafische Oberfläche oder ein umliegendes Warehouse Management System) Aufträge bzgl. Paketzustellungen im System an. Des Weiteren wird dieser Subdienst zur Synchronisierung der Datenbestände aller OrderManagements genutzt.

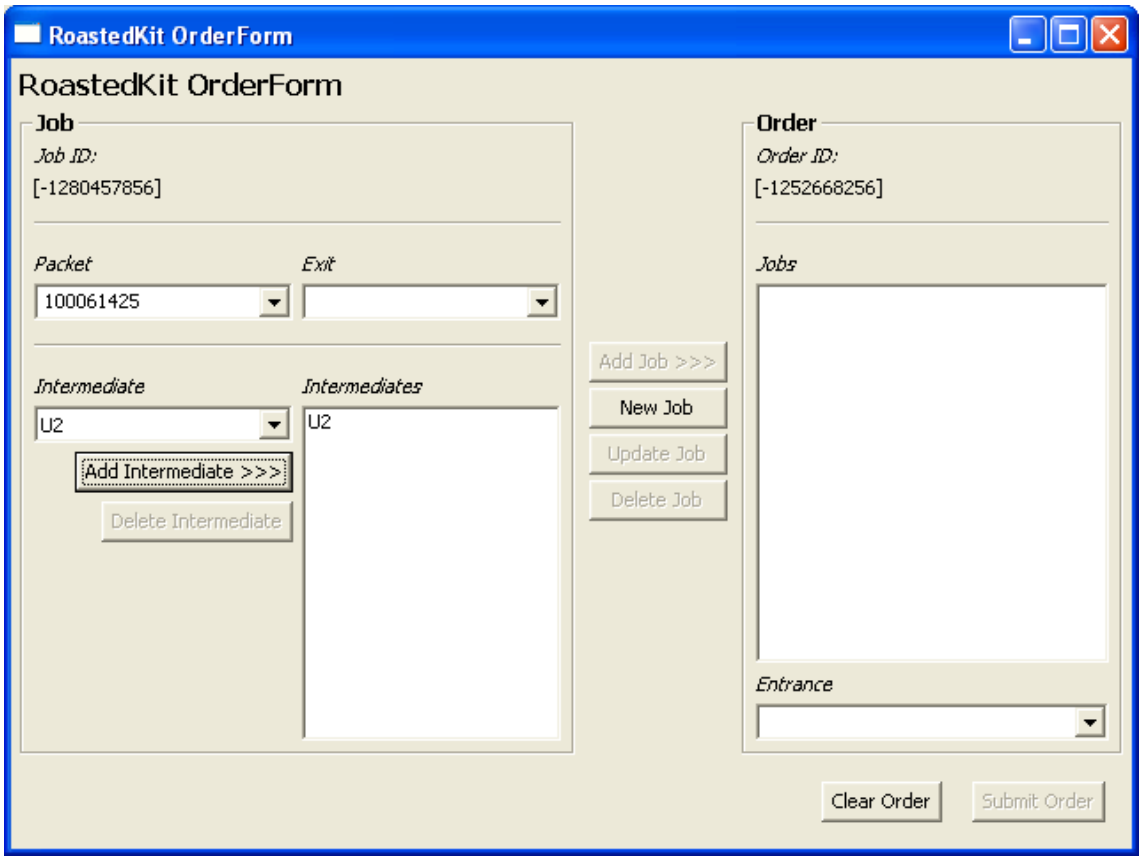

Abbildung 9.9: Benutzeroberfläche „OrderForm“

Entfernen von Aufträgen Primär wird das Entfernen von Aufträgen genutzt um (wie bei der Eingabe) die Datenbestände aller OrderManagements konsistent zu halten. Änderungen in der Auftragsliste wie beispielsweise das Entfernen eines abgearbeiteten Auftrags werden hiermit den anderen OrderManagements mitgeteilt. Weiterhin kann der Benutzer diese Schnittstelle nutzen um manuell Aufträge zu entfernen.

Abfragen von Auftragsinformationen Dieser Subdienst ist wichtig für die grafische Benutzeroberfläche und bietet die Abfrage von auftragsspezifischen Informationen, so dass diese dem Benutzer in der GUI mitgeteilt werden können.

Die detaillierte Beschreibung dieses Dienstes ist in Anhang B.9 auf Seite 130 dargestellt.

9.10 PacketManagement

Die zentrale Aufgabe des PacketManagement (PM) besteht darin, die vom OrderManagement (OM) generierten Jobs anzunehmen und auszuführen. Ausführen bedeutet hier: Das zum Job gehörende Paket über die angegebenen Intermediates zum Ausgang zu leiten, und zwar auf dem bestmöglichen Weg. Konkret bedeutet dies, an einem Entscheidungsknoten dem SystemManagement (SM) mitzuteilen, an welchen Ausgang es ein bestimmtes Paket zu befördern hat.
Um diese Aufgabe zu erfüllen bedarf es zusätzlicher Verfahren, die es ermöglichen, Wege zu finden und dann zu entscheiden, welcher der Beste zu einem bestimmten Ziel ist. Diese Verfahren werden ebenfalls im PacketManagement bereitgestellt. Es ist auf jedem Entscheidungsknoten und zusätzlich auf Waren Ein- und Ausgängen vorhanden. Ebenfalls wird es bei Bedarf auf Förderelementen, die als Intermediates angefahren werden sollen gestartet. Das ist notwendig, damit dieser Knoten in den Routing Tabellen der anderen PacketManagements auftaucht.

9.10.1 Architektur

Das PacketManagement ist eine von BaseComponent abgeleitete Komponente. Dadurch erhält es auch deren Subkomponenten UPnPControl, ServiceControl, EventControl und StatusControl. Diese sind in Abb. 9.10 auf der linken Seite zu sehen. Passend dazu gibt es den Event- und den StatusService. Auf der rechten Seite in Abb. 9.10 sieht man die "eigenen" Subkomponenten des PacketManagement: das Jobmanagement, das Routing und das Scouting. Dort ist auch die Web Service Schnittstelle "PacketService" zu finden. Damit das PacketManagement arbeiten kann, muss für den selben Knoten bereits ein SystemManagement laufen.

9.10.2 Subkomponenten

In diesem Abschnitt wird genauer auf die Aufgaben und Funktionen der einzelnen Subkomponenten eingegangen. Behandelt werden nur die PacketManagement-spezifischen Subkomponenten.

JobManagement

Das JobManagement kümmert sich darum, dass Jobs angenommen und weitergeleitet werden. Dies geschieht vornehmlich durch den Aufruf der Web Service Methode *pushJob()*, welchen das JobManagement bearbeitet. Alle Methodenaufrufe, die über die Grenzen des JobManagements hinausgehen sind der Übersicht halber in Abb. 9.11 dargestellt. Die Hauptaufgabe des JobManagements ist den besten Weg für das im Job enthaltene Paket zu ermitteln und diese Entscheidung dem SystemManagement mitzuteilen. Dazu wird die Methode *deliverPacket()* am SystemService aufgerufen. Bevor jedoch der Weg für einen Job herausgesucht werden kann, muss geprüft werden, ob der aktuelle Knoten in der *Intermediate*-Liste des Jobs an erster Stelle steht. Ist das der Fall, wird der Eintrag gelöscht. Nun wird getestet, ob es noch Intermediates gibt, oder ob schon das Ziel angefahren werden soll. Ist das geklärt, kann mit der Methode *getRoutingTable()* im Routing die aktuelle Routing Tabelle geholt werden, und dort nachgeschlagen werden, wohin a) das SystemManagement das Paket weiterschicken muss, und b) an welches PacketManagement der Job weitergesendet werden muss. Das Weitersenden und Archivieren der Jobs in der History sind abhängig von Events (welche während der Initialisierungsphase im SM gebucht werden). Empfängt das JobManagement das Event PACKET_DETECTED, wird der passende Job an das nächste JobManagement gesendet. Archiviert wird der Job, sobald das Event PACKET_DONE empfangen wird.
Damit das JobManagement arbeiten kann, muss von der Routing Subkomponente eine aktuelle Routing Tabelle bereitgestellt werden. Des Weiteren muss das SystemManagement desselben Knotens den SystemService zur Verfügung stellen, damit die Methode deliverPacket benutzt werden kann. Als Letztes muss von der ScoutingMaster-Subkomponente eine Liste der direkten Nachfolger bezogen werden können (getSuccerssors).

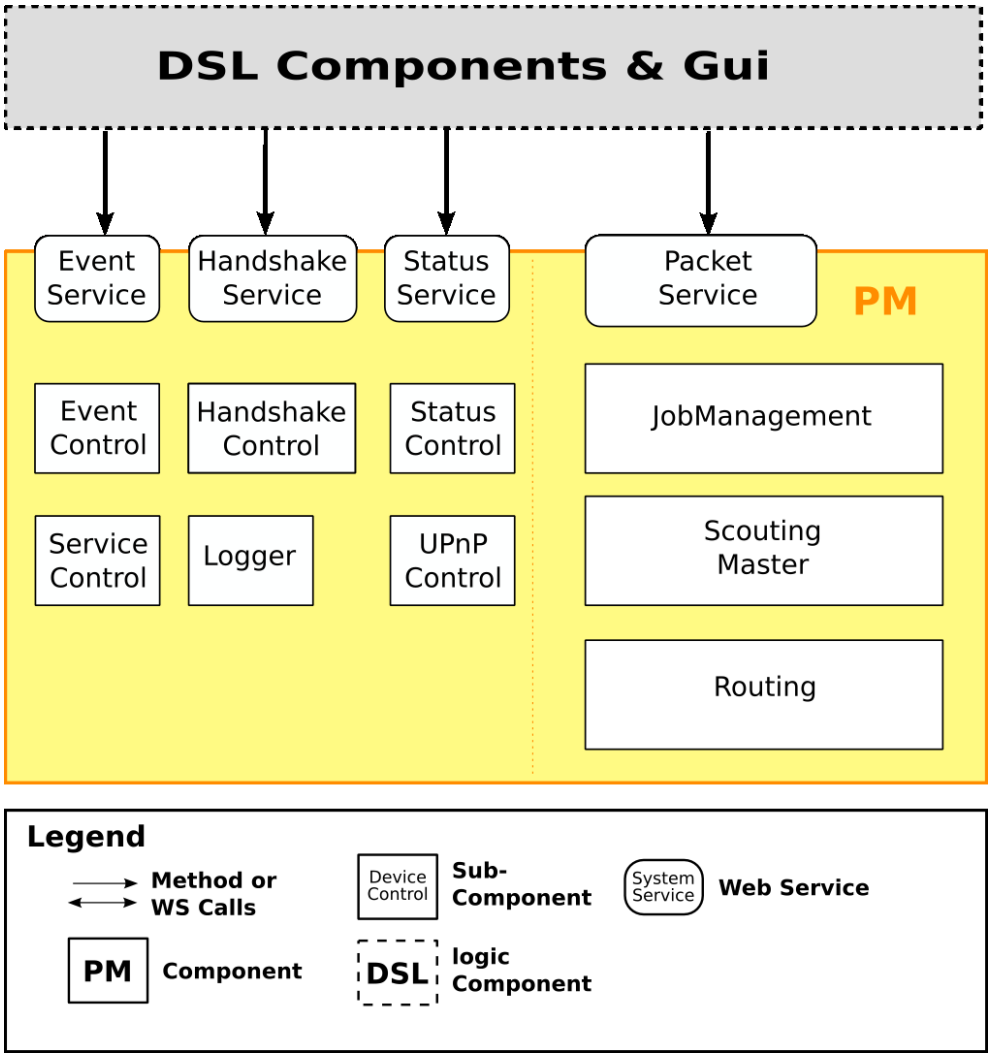

Abbildung 9.10: Schematische Darstellung des PacketManagements

Routing

Das Routing dient dazu, kürzeste Wege in der Förderanlage zu finden und diese bereit zu stellen. Dazu müssen Informationen erstellt, verbreitet, gesammelt und ausgewertet werden. Damit andere Komponenten möglichst einfach Informationen abfragen können, wird von der Routing Komponente eine Routing Tabelle gepflegt, aus der der kürzeste Weg zu einem anderen Packet-Management abgefragt werden kann.
Das Routing benutzt ein besonderes Business Object RoutingMessage, welche die folgenden Felder enthält:

- ***ComponentAddress*** *source*: Die Adresse des Quell PMs.
- ***long*** *sequencenumber*: Eine Sequenznummer, die zur Bestimmung der Aktualität einer Routing Information genutzt wird.
- ***byte*** *ttl*: Die Time To Live. Begrenzt die Zeit, die eine RoutingMessage weitergeschickt wird.
- ***Successor-Array*** *successors*: Die Liste der Nachfolger und der Abstand zu diesen.

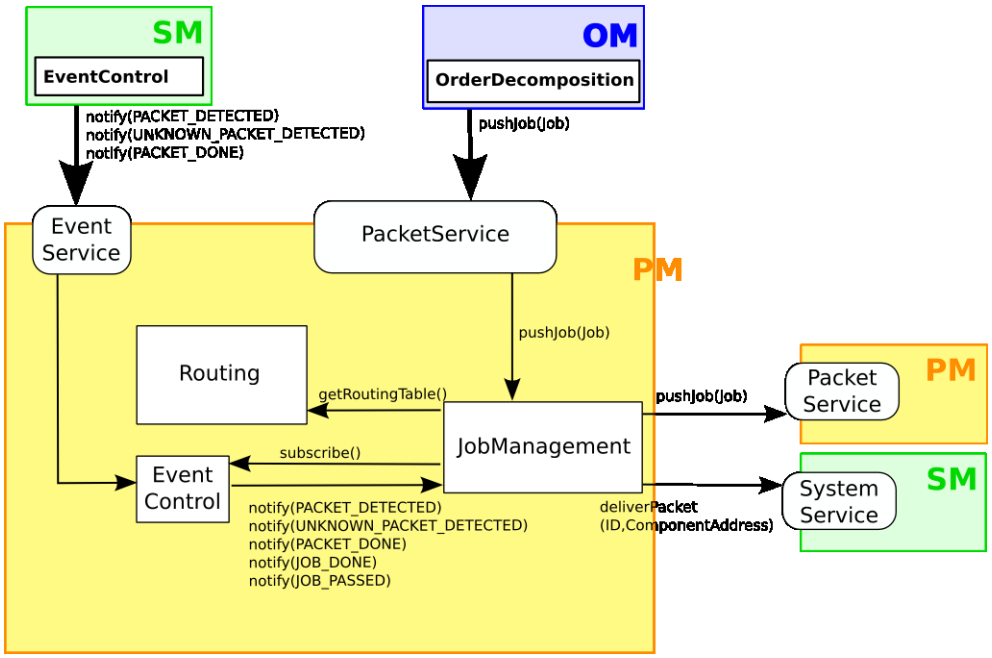

Abbildung 9.11: Kommunikationsbeziehungen des JobManagements

Wie das Routing genau funktioniert, soll die folgende Kurzbeschreibung der wichtigsten Methoden skizzieren. Zur besseren Übersicht sind die Methodenaufrufe, die über die Grenzen des Routing hinausgehen, in Abb. 9.12 dargestellt.

shareRoutingInformation(RoutingMessage) Diese Methode wird vom PacketService durchgereicht und bearbeitet RoutingMessages, die von anderen PacketManagements empfangen werden. Dazu wird geprüft, ob von dem Quell-PM (Inhalt des Felds *source*) bereits eine RoutingMessage vorliegt. Ist das der Fall, so wird geprüft, ob die Sequenznumber der empfangenen RoutingMessage größer als die der schon vorhandenen ist ($Sequenznr_{neu}$ $Sequenznr_{alt}$), in diesem Fall wird die RoutingMessage gespeichert, anderenfalls ($Sequenznr_{neu} \leq Sequenznr_{alt}$) verworfen. Liegt zu dem Quell-PM der RoutingMessage noch kein Eintrag vor, wird die RoutingMessage ebenfalls gespeichert. Die TTL wird nun um 1 decrementiert und falls der Wert noch größer 0 ist, wird die RoutingMessage per *shareRoutingInformation()* an alle bekannten Nachfolger weitergesendet.

update() In dieser Methode werden per Aufruf von *getSuccessorlist()*, bzw. *getPredecessorlist()* im ScoutingMaster die direkt benachbarten PacketManagements geholt. Mit der Successorlist wird eine RoutingMessage erstellt. Diese wird an alle PacketManagements aus der Predecessorlist per shareRoutingInformation(PacketService) versendet. Abschließend wird noch die Sequenznummer, welche im Routing verwaltet wird um 1 erhöht, damit die nächste RoutingMessage nicht die gleiche Sequenznummer verwendet.

makeSpanningTree() Diese Methode führt den Dijkstra-Algorithmus auf den gespeicherten RoutingMessages aus. Die RoutingMessages repräsentieren in ihrer Gesamtheit die Topologie der Anlage als Graph. Sie werden so gespeichert, dass sie als Adjazenzliste vorliegen. Nach Ende des Algorithmus enthält ein Distanzfeld die Abstände aller Knoten zum Startknoten s. In dem Vorgängerfeld wird ein Spannbaum, mit allen von s ausgehenden, minimalen Wegen, in Form

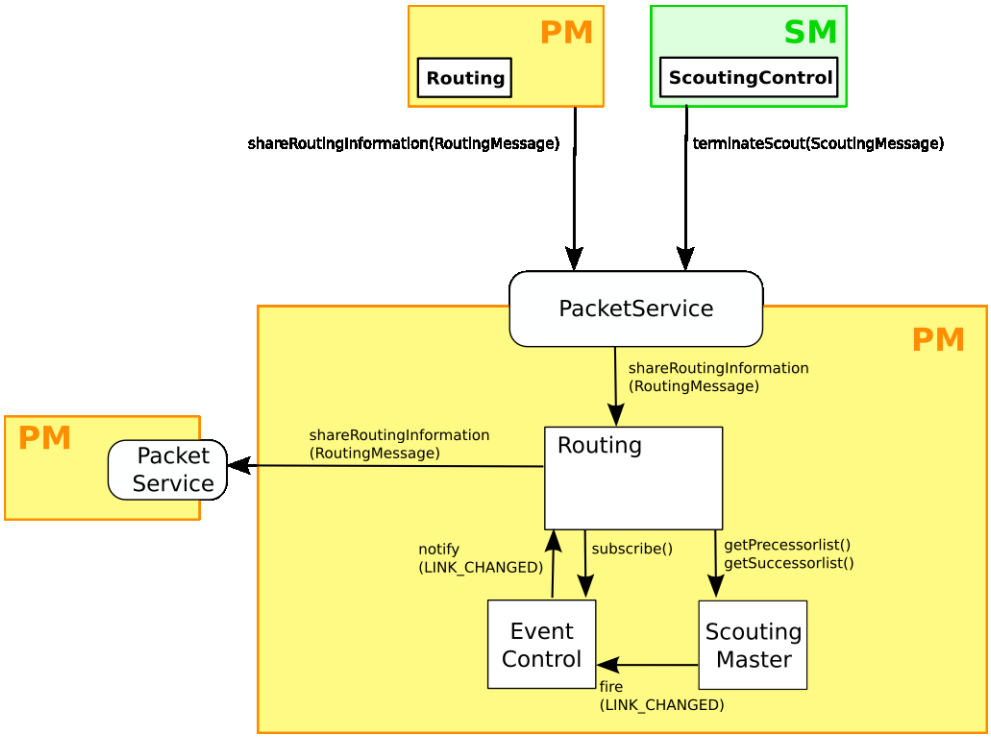

Abbildung 9.12: Kommunikationsbeziehungen des Routing

eines In-Tree abgelegt. Im Folgenden wird nun kurz auf die Berechnung eines Spannbaums mit Hilfe des Dijkstra-Algorithmus, wie in [5] erklärt, eingegangen.

Gegeben: Die Knotenmenge $V = 1, ..., n$, eine Kostenmatrix mit den Einträgen $c(i, j) \geq 0$ für $1 \leq i, j \leq n$, welche die Kosten der Kante von i nach j angeben. Außerdem ist ein ausgezeichneter Startknoten s gegeben.
Aufgabe: Berechne für alle Knoten i die Kosten des billigsten Weges von s nach i. Diese Kosten seien mit $d(i)$ bezeichnet.

Listing 9.3: Pseudo-Codeblock des Dijkstra-Algorithmus

```
1  A:={s}. For all j in V set d*(j):=c(s,j).
2  while A != {1, ..., n} do
3  begin
4      choose vertex i* = i from V \ A with min{ d*(i) }
5      A:= union over A and {i*}.
6      for all vertexes j not in A do
7      begin
8          d*(j):= min{d*(j),d*(i*) + c(i*,j)}.
9      end.
10 end.
```

calculateRoutingTable() Hierbei wird nun mit Hilfe des "Vorgänger"-Baums für jeden Knoten ermittelt, welches der nächste Hop, also das erste PacketManagement auf dem kürzesten Weg, ist. Zu diesem wird dann das erste SystemManagement bestimmt. Dies benötigt man für Aufrufe im eigenen SystemManagement. Diese Informationen werden zusammen mit der Metrik und dem Knoten, für den man das alles berechnet hat, in die Routing Tabelle geschrieben. Damit das Routing arbeiten kann, muss von der ScoutingMaster-Subkomponente die Liste der direkten Nachbarn abrufbar sein.

ScoutingMaster

Der ScoutingMaster kümmert sich zusammen mit ScoutingControl (Subkomponente des System-Managements), um die Erkundung der (PacketManagement-)Topologie innerhalb des Prototyps. Dies ist zum einen nötig, um die Kooperationsbeziehungen zwischen den vorhandenen Packet-Managements zu bestimmen und zum anderen, um die nötigen Topologieinformationen für das Routing zu bestimmen.
Der ScoutingMaster benutzt ein besonderes Business Object ScoutingMessage, welches die folgenden Felder enthält:

- ***ComponentAddress*** *source*: Die eigene Adresse des Quell-PMs, zu der die ScoutingMessage zurückgeschickt werden muss.
- ***ComponentAddress*** *firstSM*: die Adresse des ersten SMs auf dem Weg. [1]
- ***int*** *metric*: initial = 0; Wird auf jedem Knoten erhöht (steht für die durchschnittliche Durchlaufzeit in Sekunden).
- ***ComponentAddress*** *terminationPM*: initial = NULL; die Adresse desjenigen PMs, welches die Nachricht zur Source zurückschickt.

Abbildung 9.13 zeigt eine Übersicht der über die Grenzen des ScoutingMaster hinausgehenden Methodenaufrufe. Einmal gestartet, sendet ein ScoutingMaster daher alle 30 Sekunden eine Scou-

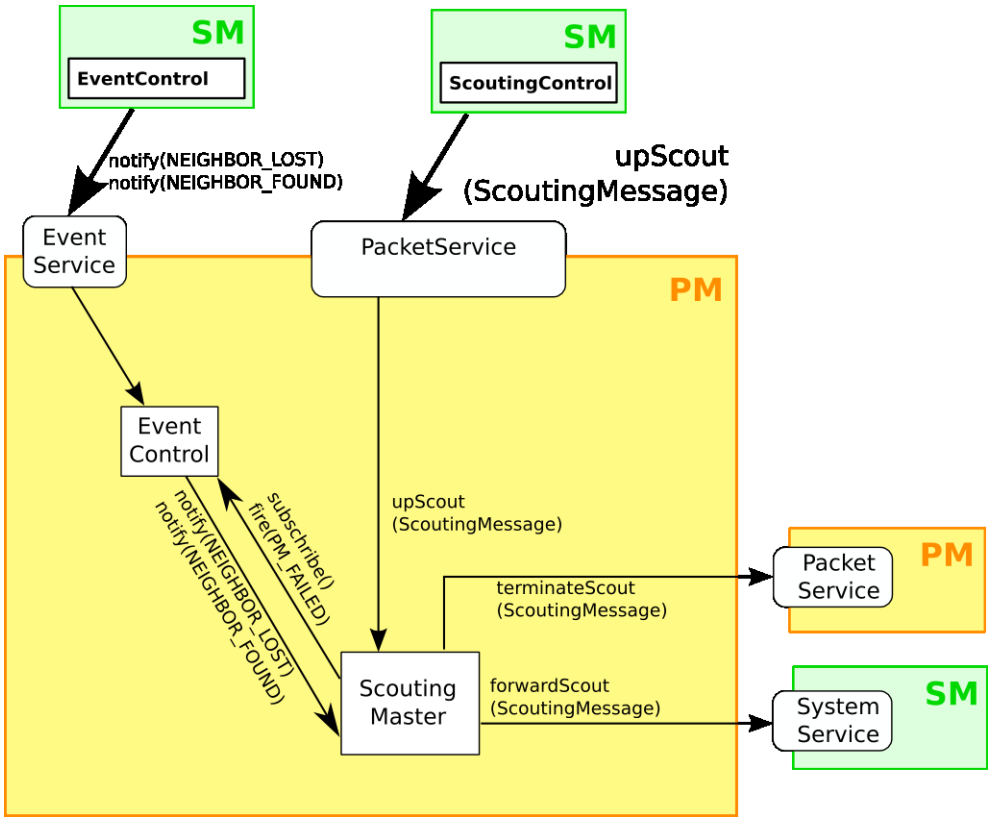

Abbildung 9.13: Kommunikationsbeziehungen des ScoutingMaster

tingMessage an alle Nachfolger, welche die NeigbourControl (siehe Kapitel 9.11.2) identifiziert

[1] Das ist sehr wichtig, da es mehrere Ausgänge am aktuellen Knoten gibt (sonst gäbe es initial kein PM) und man für spätere Steueraktionen das dann erforschte PM einem SM zuordnen muss, welches das eigene SM kennt

hat. Die Nachfolger werden über den Web Service Aufruf *getSuccessors()* im SystemManagement desselben Knotens geholt. Die ScoutingMessage wird dann per *startScout()* Aufruf im SM gestartet. Hat eine Scouting Nachricht einen Knoten erreicht, auf dem ein PacketManagement vorhanden ist, so wird vom SystemManagement die Methode *upScout()* aufgerufen werden. Das PM, welches die ScoutingMessage erhalten hat, wird das PacketManagement, welches in dem Feld *source* eingetragen ist, in seiner Liste der Vorgänger speichern. Als letztes wird in der ScoutingMessage in das Feld *terminationPM* der aktuelle Knoten eingetragen und dann die ScoutingMessage an das Quell-PM per *terminateScout()* zurückgeschickt. Wenn eine ScoutingMessage nun per *terminateScout()* an seinen Ursprung zurückkehrt, wird die ganze ScoutingMessage in die SuccessorList aufgenommen.
ScoutingMessages werden auch gesendet, wenn das Event NEIGHBOUR_LOST oder NEIGHBOUR_FOUND erhalten wurde, da sich durch Änderungen der Nachbarschaftsbeziehungen neue Vorgänger/Nachfolger ergeben können.
Damit der ScoutingMaster arbeiten kann, müssen vom SystemManagement desselben Knotens die Web Service Methoden *getSuccessors()* und *startScout()* erreichbar sein.

9.10.3 Schnittstellenbeschreibung

PacketService

Auch im PacketManagement wird nur ein Dienst - der PacketService - angeboten und kümmert sich um verschiedene Aufgaben für den Bereich der Paketzustellung. Daher lässt sich dieser Dienst wiederum in drei Bereiche an Subdiensten unterteilen:

Paketzustellung Dieser Subdienst dient dazu, Pakete vom OrderManagement entgegenzunehmen und zum entsprechenden Warenausgang (über eventuelle *Intermediates*) zu befördern. Für die Paketzustellung nutzt das PacketManagement wiederum Dienste des SystemManagement.

Topologieerkundung Die Topologieerkundung erfolgt mit Hilfe des oben genannten Scouting-Algorithmus', der wiederum diesen Subdienst nutzt. Diese Informationen über die lokale Umgebung werden dann allen anderen PacketManagements mitgeteilt.

Berechnung der Routing-Tabelle Mit Hilfe dieses Dienstes können Routing-Informationen ausgetauscht bzw. verbreitet werden. Erst dadurch kann überhaupt ein Graph für Wegeberechnungen erstellt werden.

Eine detaillierte Schnittstellenbeschreibung ist im Anhang B.10 auf Seite 131 zu finden.

9.11 SystemManagement

Das SystemManagement (SM) stellt die Schnittstelle zur realen bzw. emulierten Hardware dar, eine Instanz eines SystemManagements ist mit einem einzigen Förderelement verknüpft.
Eine der Hauptaufgaben des SystemManagements ist es miteinander verbundene Förderelemente zu finden. Erst durch bekannte Nachbarn, also SystemManagaments, die benachbarte Förderelemente steuern, ist es möglich auf einer höheren Ebene die gesamte Topologie aufzubauen und über diese z.B. Routingentscheidungen zu treffen.
Ferner bietet das SystemManagement die Laufzeitumgebung für die Steuermodule, welche erst zur Laufzeit geladen werden, was eine Erweiterung um neue Förderelemente vereinfacht. Steuermodule können die Devices (wie Motoren, Weichen, Sensoren) eines Förderelements abfragen bzw. steuern und behandeln Events (z.B. „Lichtschranke 1 ausgelöst!“) von diesen.
Des Weiteren enthält das SystemManagement die Interpolation, welche es ermöglicht, ein Förderelement und darauf befindliche Devices und Pakete zu emulieren (genaueres siehe Kapitel 9.12 auf Seite 85). Dies erlaubt es, auch komplett emulierte Förderanlagen - sei es zu Testzwecken oder Evaluierung neuer Förderelemente - zu erstellen.

9.11.1 Architektur

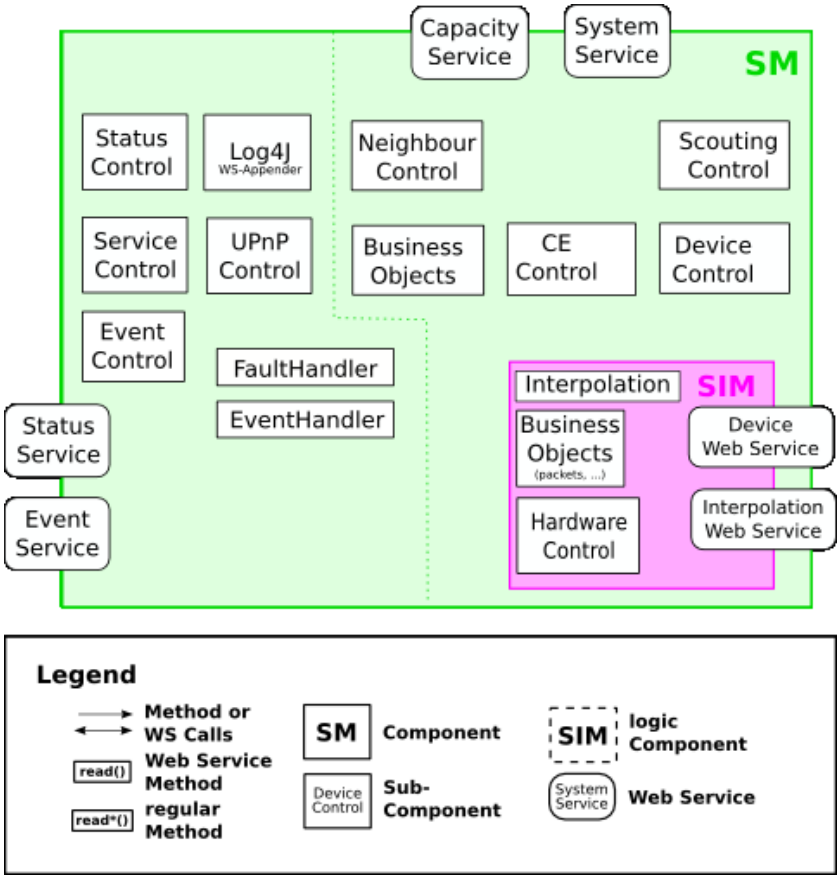

Abbildung 9.14: Schematische Darstellung des SystemManagements

Die in der Architektur zu erkennenden Subkomponenten des SystemManagements sind folgende:

- *ScoutingControl*: Diese Subkomponente sorgt für das Scouting durch die Förderanlage. Aus den gesammelten Scoutinginformationen erstellt das PacketManagement dann seine Routingtabellen.

- *NeighbourControl*: Diese Subkomponente berechnet, anhand der eigenen Koordinaten und der eingehenden UPnP-Meldungen, die Nachbarn des Förderelements.
- *CE-Control*: Diese Subkomponente sorgt dafür, dass das richtige Steuermodul des Elements zum richtigen Zeitpunkt geladen und gestartet wird. Es existiert für jedes Förderelement ein Steuermodul, welches ganz konkret auf die Anforderungen dieses Moduls zugeschnitten ist, denn jedes Modul unterscheidet sich durch die Anordnung und Anzahl der auf ihm vorhandenen Devices.
- *DeviceControl*: Diese Subkomponente übernimmt die eigentliche Ansteuerung der Hardware. Sie wandelt die Aufrufe aus den Steuermodulen in *read/write*-Befehle um und leitet diese an das DeviceManagement weiter. (näheres siehe Kapitel 9.13)
- *CapacityControl*: Diese Subkomponente ermöglicht es, die Kapazitäten der einzelnen Elemente bzw. der Trackparts (Trackparts sind einzelne Abschnitte auf einem Förderelement) zu setzen. Im Prototyp beschränken wir uns auf eine Kapazität von 1 pro Element. Es können also niemals mehrere Pakete gleichzeitig auf einem Element vorhanden sein.
- *Interpolation*: Die Interpolation berechnet aufgrund, der zu Verfügung stehenden Daten und Events, die Positionen der Pakete und stellt diese bereit. Wegen der Komplexität dieser Subkomponente ist diese Beschreibung in das Kapitel 9.12 ausgelagert.

9.11.2 Subkomponenten

NeighbourControl

Diese Komponente findet die direkten Nachbar-SystemManagements und überwacht Änderungen der Nachbarschaftsbeziehungen, wie etwa Wegfall und Hinzukommen neuer Nachbarn. Um Nachbarn benennen und unterscheiden zu können, wird eine eindeutige Adressierung benötigt, welche die *ComponentAddress* realisiert (siehe Kapitel 8.5).

Globaler Anlagenraum und lokaler Förderelementraum Grundsätzlich unterscheiden wir zwei von uns definierte Räume. Der *Anlagenraum* beschreibt den dreidimensionalen Raum, in dem alle Förderelemente an unterschiedlichen Positionen aufgestellt sind. Dieser umfasst die gesamte Förderanlage in einem gemeinsamen Koordinatensystem, Abstände zwischen zwei Förderelementen können mit Hilfe der Vektorrechnung bestimmt werden.

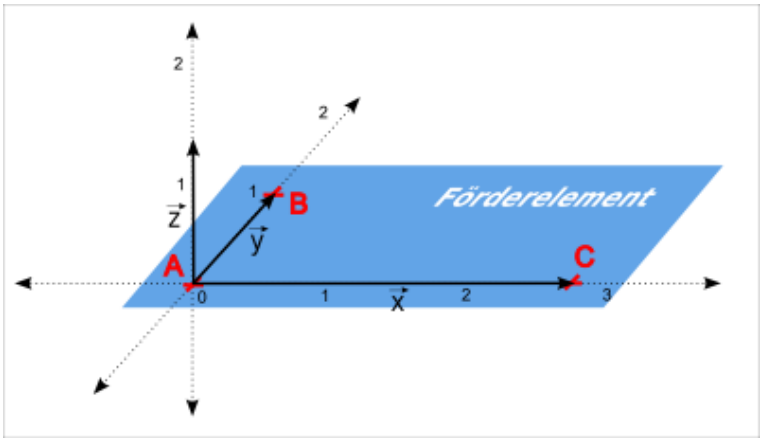

Abbildung 9.15: Aufgespannter Förderelementraum über drei Messpunkte

Der *Förderelementraum* hingegen dient dazu, die Ein- bzw. Ausgangskoordinaten des Förderelements im Anlagenraum zu berechnen. Jedes Förderelement hat durch seinen Deskriptor (siehe

Kapitel 8) drei Messpunkte gegeben, welche die Lage im Anlagenraum festlegen. Der Förderelementraum wird wie in Abb. 9.15 über die drei Messpunkte aufgespannt und so entsteht ein eigenes Koordinatensystem pro Förderelement. Die (normierten) Basisvektoren des Koordinatensystems berechnen sich wie folgt

$$\overrightarrow{x} = \frac{\overrightarrow{C} - \overrightarrow{A}}{|\overrightarrow{C} - \overrightarrow{A}|}$$

$$\overrightarrow{y} = \frac{\overrightarrow{B} - \overrightarrow{A}}{|\overrightarrow{B} - \overrightarrow{A}|}$$

$$\overrightarrow{z} = \frac{\overrightarrow{x} \times \overrightarrow{y}}{|\overrightarrow{x} \times \overrightarrow{y}|}$$

wobei $\overrightarrow{A}$, $\overrightarrow{B}$, $\overrightarrow{C}$ die Messpunkte des Förderelements im Anlagenraum sind.

Berechnen der Ein-/Ausgänge des Förderelements Die relativen Ein- und Ausgangskoordianten werden für jedes Förderelement in der Typenbibliothek abgelegt; sie beschreiben Ein-/Ausgangspunkte im Förderelementraum. Die NeighbourControl berechnet diese nun für den Anlagenraum. Als Beispiel in Abb. 9.16 sei ein Ausgangspunkt P gesucht, der mit Hilfe der folgenden Formel berechnet werden kann

$$\overrightarrow{P} = \overrightarrow{A} + r * \overrightarrow{x} + s * \overrightarrow{y} + t * \overrightarrow{z}$$

$\overrightarrow{A}$: Messpunkt (Nullpunkt) des Förderelements
$\overrightarrow{x}$, $\overrightarrow{y}$, $\overrightarrow{z}$: die oben berechneten Basisvektoren
r, s, t: relative Ein- und Ausgangskoordianten für Punkt P aus der Typenbibliothek

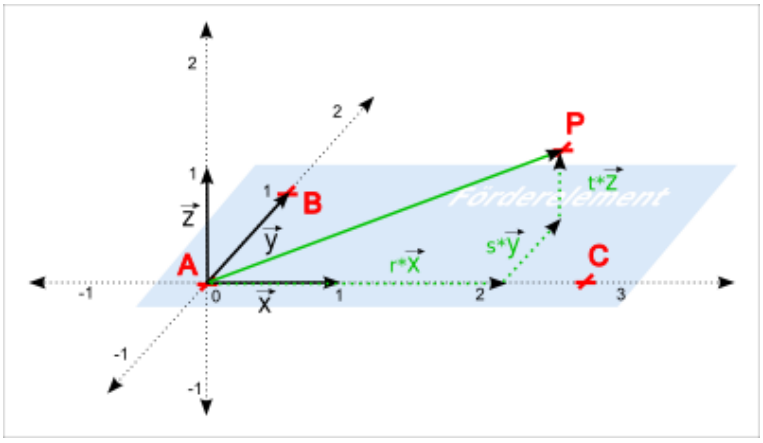

Abbildung 9.16: Berechnung des Ausgangspunktes P

Nachdem während der Initialisierung die Koordinaten aller Ein- bzw. Ausgänge für den Anlagenraum berechnet wurden, setzt die NeighbourControl diese in der *ComponentAddress* des SystemManagements. Erst jetzt kann das SystemManagement fertig initialisiert werden und per UPnP sichtbar geschaltet werden.

Berechnen der Nachbarn Wird ein anderes SystemManagement per UPnP gefunden, prüft die NeighbourControl anhand der in der ComponentAddress übermittelten Ein- bzw. Ausgangskoordinaten über Kreuz, ob diese zu den eigenen Ein- bzw. Ausgängen passen. Hierbei wird der Abstand der beiden Punkte bestimmt. Ist dieser nicht größer als 10cm wird das gefundene SystemManagement als Nachbar anerkannt. Wird ein neuer Nachbar gefunden, erzeugt die

NeighbourControl ein Event, das die ComponentAddress des gefundenen Nachbarn enthält und benachrichtigt die lokale EventControl, die dieses wiederum weiterleitet.
Die Listen der Nachfolger und der Vorgänger jedes Elements werden in der NeighbourControl verwaltet und anderen Elementen zu Verfügung gestellt, die diese Daten benötigen, beispielsweise dem Scouting.

CapacityControl

Diese Komponente soll es ermöglichen, eine Kapazität für ein Förderelement zu buchen. Für jedes Paket, das auf das nächste Förderelement geschickt werden soll, muss erst eine Kapazitätseinheit gebucht werden. Diese Kapazität wird intern von der Gesamtstrecke auf einen TrackPart (Streckenabschnitt eines Förderelements, siehe Interpolation) umgelegt. So bietet die Außensicht nur die Buchung einer Einheit, während im Inneren genau überprüft wird, auf welchem Abschnitt des Förderelements der geforderte Platz frei sein muss.
Die genauen Kapazitäten der einzelnen TrackParts werden im jeweiligen *CEModule* bestimmt und während deren Initialisierungsphase gesetzt. Im Prototyp beträgt diese Kapazität genau 1 pro Förderelement.
An dieser Stelle ist besonders das Konzept des *Buchens* zu erwähnen. Bei einer Buchung wird geprüft, ob auf dem Trackpart, welcher der buchenden Komponente am nächsten liegt, noch Platz zur Verfügung steht. Ist dies der Fall, darf das buchende Paket direkt einfahren. Verlässt das Paket den Trackpart wieder, sorgt die CapacityControl dafür, dass die aktuelle Kapazität des momentanen Trackparts wieder um einen Punkt gesenkt und auf dem nächsten erhöht wird. Dies wiederholt sich, bis das Paket das Förderelement verlassen hat und der gleiche Ablauf auf dem nächsten Förderelement und dessen CapacityControl fortgesetzt wird.
Ist die Buchung nicht erfolgreich gewesen, muss das Paket auf dem Förderband davor warten. Sobald nun die Kapazität wieder da ist, wird ein RELEASE_PACKET Event mit der PaketID des Pakets geschickt, das nun einfahren kann. Für diese Events registriert sich jedes Modul bei der CapacityControl.

DeviceControl

Die DeviceControl stellt der CEControl eine einfache Schnittstelle zur Verfügung, um einzelne Devices steuern zu können. Sie bietet vereinfachte Methoden an, die es z.B. erlauben einen Motor anzuschalten. Intern setzt sie einen Aufruf in die im DeviceService definierten States um und leitet sie an die HardwareControl (Subkomponente der Interpolation) weiter. Die DeviceControl muß nur initial describe() an der HardwareControl der Interpolation aufrufen. Dadurch erhält sie die wirklich vorhandenen Devices auf dem aktuellen Förderelement.

Steuermodule & CE-Control

Die CEControl lädt eine von CEModule abgeleitete Klasse zur Laufzeit. In dieser abgeleiteten Klasse findet sich die eigentliche Steuerung eines bestimmten Förderelements wieder; hier wird genau bestimmt, welche Steueraktionen für die jeweiligen Events durchgeführt werden müssen. Jedes Förderelement besitzt sein eigenes CE-Modul. Die Module nutzen die eigene DeviceControl, um die eigentliche Hardwaresteuerung durchzuführen.
Hier geschieht die eigentliche Steuerung des Elements. Sie reagiert auf hereinkommende Events. Kommt ein Event eines RFID-Scanners, der an jeder Weiche vorhanden ist, wird anhand der in diesem Event vorhandenen Paketinformationen das vom Routing bestimmte Ziel abgerufen und bei Bedarf die Weiche gestellt. Des Weiteren wird auf jedem Förderelement auf Sensorevents reagiert, entweder auf "reale" von Lichtschranken oder auf Events virtueller Lichtschranken, falls keine geeigneten realen vorhanden sind. Nach dem Event der Lichtschranke, kurz vor Ende des Elements wird die Methode *bookCapacity()* der CapacityControl des nächsten Elements aufgerufen. Kommt hier ein *true* zurück, fährt das Paket weiter und verlässt dieses Modul. Kommt

hingegen ein *false*, wird das aktuelle Förderelement angehalten und das Paket muss auf das Event RELEASE_PACKET warten, bis der Motor wieder startet. Eine Ausnahme bietet hier das Modul U4, welches hier nicht ausgeschaltet wird, sondern seinen Stopper auslöst und auf diese Art die Pakete anhält An jeder Zusammenführung, also in der Anlage an U1 und an U5, wird eine Vorfahrtsstrecke definiert, in unserem Fall U1 und U7. Kommt von diesen Elementen ein Paket, müssen die Pakete auf den anderen Elementen, U13 bzw. U4, warten, damit es nicht zu Kollisionen kommt. Die CE-Control hat nun die Aufgabe, dafür zu sorgen, dass die Module nur zu den "richtigen" Zeitpunkten auf Events reagieren. Insbesondere werden die Module nur dann gestartet, wenn die jeweiligen Nachfolger auch im System vorhanden sind und durch die NeighbourControl gefunden werden. Der allererste Impuls eines Anlagen- bzw. Steuerung-Starts muss vom Nutzer kommen, der dazu im grafischen Frontend den Startbknopf drückt. Darauf startet Element U9, welches den Start von U8 veranlasst; sind alle Elemente bereit, starten rückwärts durch das System alle anderen Förderelemente.

ScoutingControl

Die ScoutingControl sorgt für das Scouting durch die gesamte Förderanlage. Die ScoutingControl nimmt ScoutingMessages vom PacketManagement entgegen und fügt zu dieser Nachricht die eigene Metrik hinzu. Die Metrik entspricht in diesem Fall der durch Messungen ermittelten Durchschnittsgeschwindigkeit der einzelnen Förderelemente. Diese ScoutingMessage wird nun an das nächste SystemManagement oder an das eigene PacketManagement (falls vorhanden) weitergeleitet. Im Prototypen existieren standardmäßig vier verschiedene Routen, die die Metriken 2 (U8 → U9), 27 (U8 → U1), 61 (U1-U8 via U6) und 84 (U1-U8 via U2) haben. Die ScoutingMessages laufen jeweils genau von einem PM zum nächsten. Weitere Details hierzu stehen in Kapitel 9.10, PacketManagement.

Schema Der ScoutingMaster holt sich als erstes vom eigenen SM die Nachfolger SMs. Nun wird für jedes der Nachfolger SMs eine ScoutingMessage erzeugt.

1. Das PM von "A" erstellt eine Scoutnachricht mit "1" als erstem SM und sendet diese ans eigene SM (ScoutingControl).
2. Das SM von "A" wertet als erstes SM "1" aus und schickt die Nachricht an dieses weiter.
3. Das SM von "1" erhöht die Metrik um 2 und schickt die Nachricht an "2" weiter.
4. Das SM von "2" erhöht die Metrik um 1 und schickt die Nachricht an "3" weiter.
5. Das SM von "3" erhöht die Metrik um 4 und schickt die Nachricht an "B" weiter.
6. Das SM von "B" erhöht die Metrik um 6 und da es ein PM hat reicht es die Nachricht an dieses hoch.
7. Das PM von "B" speichert "A" als Vorgänger ab und schickt ihm die Nachricht zurück!

Analog funktioniert der Fall mit 4 als erstem SM. Damit haben wir 2 Dinge erledigt. Zum einen kennt "A" nun alle seine Nachfolger PMs und die Entfernung zu diesen. Zweitens kennen jeweils "B" und "C" "A" als einen ihrer Vorgänger.

9.11.3 Schnittstellenbeschreibung

Das SystemManagement ist eine speziellere Komponente im System. Es dient als Schnittstellenkomponente für den Steuerrechner zum IndustriePC (an den die Fördertechnik angeschlossen ist) und ist somit diejenige Komponente, die die eigentlichen Steuerbefehle im Form von Web Service Aufrufen verschickt. Für die Erledigung dieser Aufgabe bietet das SystemManagement zwei Dienste an:

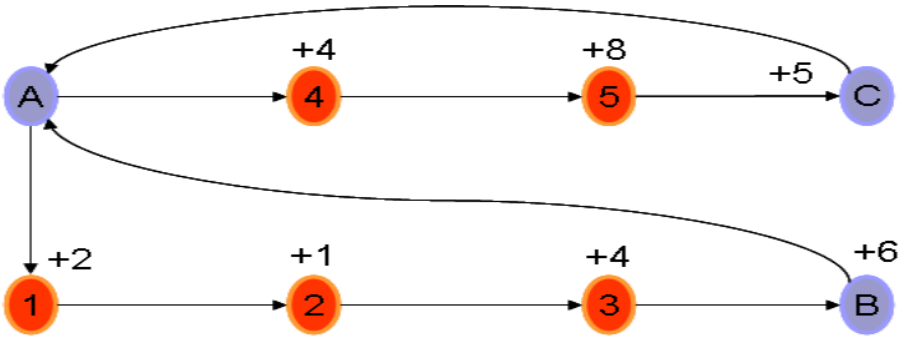

Abbildung 9.17: Graphische Darstellung des Scouting-Algorithmus

SystemService

Der SystemService kann in zwei Subdienste unterteilt werden:

Paketzustellung Dieser Subdienst kümmert sich um die wirkliche Paketzustellung zu dem entsprechenden Ziel bzw. entsprechenden Nachfolge-Förderelement. Genutzt wird diese Dienstleistung vom PacketManagement und erbracht durch die Umsetzung der Zustellungsentscheidung in Steuerbefehle (also Web Service Aufrufe) für das zuständige DeviceManagement.

Topologieerkundung Dieser Subdienst ist gekoppelt an den Topologieerkundungs-Dienst des PacketManagements und wird von diesem auch genutzt. Ziel der Beiden ist die Erkundung der Wege zu anderen PacketManagements und deren Metrik. Dies wird mit Hilfe des Scouting-Algorithmus' durchgeführt (siehe Kapitel 9.10).

CapacityService

Der CapacityService ist zuständig für die Kapazitätskontrolle bzgl. vorhandener Pakete auf den Förderelementen. Um die aktuelle Kapazität (Anzahl an Paketen) eines Förderelementes abzufragen bzw. Kapazität zu buchen, kann man also den CapacityService nutzen.

Eine detaillierte Schnittstellenbeschreibung befindet sich im Anhang B.11 auf Seite 132.

9.12 Interpolation

Die Interpolation ist eine Subkomponente des SystemManagement und ist dafür zuständig, zu jedem Zeitpunkt die Position der Pakete auf einem Förderelement zu errechnen. Außerdem gibt es hier eine Unterstützung für virtuelle Devices, d.h. man kann virtuelle Lichtschranken zum Förderelement hinzufügen. Falls ein Paket nun in eines dieser virtuellen Devices hinein- oder hinausfährt, wird ein entsprechendes Event von der Interpolation geworfen.

9.12.1 Anforderungen und Voraussetzungen

Die Interpolation muss die Paketpositionen auf den Förderelementen berechnen und dabei virtuelle Devices unterstützen. Die Interpolation soll sich an die reale Hardware anpassen, also die Geschwindigkeiten der realen Förderelemente mit der eigenen abgleichen. Ferner soll sie Pakete zur nächsten Interpolation übergeben, wenn diese die Förderelementgrenze überschreiten. Darüber hinaus wird eine DeviceService Schnittstelle zur Verfügung gestellt, welche nur von anderen Subkomponenten des lokalen SystemManagement genutzt wird.

9.12.2 Architektur

Das SystemManagement spricht das DeviceManagement über die Interpolation an. Diese unterscheidet als einzige Komponente zwischen dem DeviceManagement und der DeviceEmulation.
Die Interpolation implementiert den DeviceService. Allerdings wird dieser hier sowohl als Web Service als auch als Java-Schnittstelle implementiert. Da das SystemManagement alle Zugriffe auf die Hardware über die Interpolation abwickelt, bekommt die Interpolation jeden Befehl mit, der an die Förderelemente gesendet wird. Die Interpolation registriert sich auch bei der EventControl seines lokalen DeviceManagements, um so alle Rückmeldungen von der Hardware mitzubekommen.
Für jedes (virtuelle) Paket wird im Betrieb alle 50ms die aktuelle Position errechnet. Um auftretende Ungenauigkeiten der Interpolation auszugleichen, wird die Interpolation mit den Ereignissen von der realen Hardware synchronisiert. Wenn ein Paket in der Interpolation in einem Neuberechnungsschritt eine Lichtschranke unterbrechen würde, dieses Event aber noch nicht von dem DeviceManagement eingetroffen ist, wird das betroffene Paket in der Interpolation angehalten. Es wird auf das Event der echten Hardware gewartet. Wenn dieses eintrifft, wird der Fehlerfaktor ermittelt und die Geschwindigkeit dieser Teilstrecke des Förderelementes in der Interpolation angepasst. Sollte das Ereignis nicht eintreffen bis das nächste Paket an diesem Punkt ist, wird das vermutlich verloren gegangene virtuelle Paket entfernt. Das SystemManagement kann so genannte virtuelle Devices definieren. Das sind Punkte im Raum des Förderelements, die sich wie Lichtschranken verhalten, d.h. dass das SystemManagement benachrichtigt wird, wenn ein Paket diese Position erreicht hat bzw. sie verlässt. Dies wurde im RoastedKit an den Förderelementen genutzt, die keine eigenen echten Lichtschranken zu Verfügung stellten. Entsprechend werden Events an die EventControl des SystemManagements geschickt.
Für Ihre Arbeit braucht die Interpolation auch Informationen aus der TypeLibrary. Diese werden als Typ *sim_data* bei den Förderelementen gespeichert. Dabei wird ein Förderelement wie z.B. das in Abb. 9.18 (links) in Teilabschnitte - in sog. TrackParts (siehe Abb. 9.18, rechts) - unterteilt. Außerdem sind in der Abbildung auch Stopper und Lichtschranken dargestellt. Ein TrackPart ist ein Streckenabschnitt - z.B. von einem Eingang bis zu einer Weiche oder von der Weiche bis zu einem Ausgang. Wichtig hierbei ist, dass ein Streckenabschnitt nie über eine Weiche oder einen Zusammenführer hinausgehen kann. Sind weder Weiche noch Zusammenführer vorhanden, kann das ganze Förderelement durch einen Streckenabschnitt dargestellt werden. Ein TrackPart besteht aus 2 Punkten, dem Start- und dem Endpunkt des TrackParts. Dabei müssen die Punkte in Laufrichtung des Streckenabschnittes angegeben werden. Jeder Streckenabschnitt besitzt innerhalb des Förderelementes eine eindeutige ID.

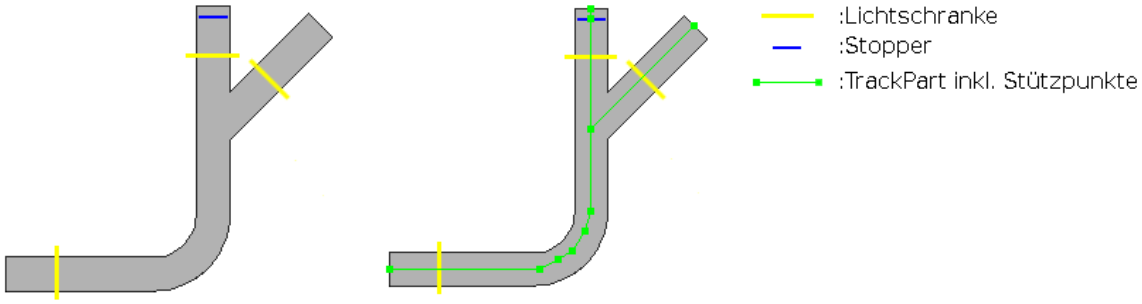

Abbildung 9.18: Förderelement ohne (links) und mit (rechts) TrackParts

Sollte es Merger oder Weichen geben, gibt es zusätzlich zu den TrackParts TrackSwitches unter sim_data. Diese Elemente führen die TrackPart-IDs zur Erkennung der angeschlossenen TrackParts. Ein TrackSwitch hat entweder mehrere Inports oder mehrere Outports.
Für die Lichtschranken müssen jeweils zwei Koordinaten angegeben werden. Einmal der Lichtaustrittpunkt und die zweite Koordinate für den Reflektor auf der anderen Seite des Förderbandes.
Für einen RFID-Scanner gibt man den Mittelpunkt seiner Antenne sowie die maximale Scanentfernung an. Hierbei wird davon ausgegangen, dass die Antenne in allen Richtungen gleich weit scannen kann. Dies entspricht - z.B. nach hinten - nicht der Realität, aber für den wichtigen Bereich auf dem Förderband sollte diese Näherung ausreichen.

9.12.3 Schnittstellenbeschreibung

InterpolationService

Die Interpolation ist Teil des SystemManagements, bietet aber selber auch einen Dienst an, den InterpolationService. Dieser bietet die Nutzung emulierter Fördertechnik für virtuelle Pakete. So kann man dem InterpolationService ein virtuelles Paket übergeben, der es daraufhin mit Hilfe der emulierten Fördertechnik innerhalb der Interpolation zum virtuellen Warenausgang befördert.
Eine detaillierte Schnittstellenbeschreibung des InterpolationService befindet sich im Anhang B.12 auf Seite 134.

9.13 DeviceManagement

Das DeviceManagement (kurz: *DM*) und seine Subkomponenten sind dafür zuständig, dass die von der dezentralen Steuerungslogik getroffenen Entscheidungen auf den zu steuernden technischen Prozess (Aktoren/Sensoren) umgesetzt werden können. Sie laufen daher auf den IPCs der Förderanlage. Pro Förderelement wird ein DeviceManagement erzeugt.
Da die Palette von Hardware echter Förderanlagen weit gefächert ist, bieten das DeviceManagement und seine Subkomponenten einen *Hardware Abstraction Layer* (kurz: HAL) an. In diesem werden die abstrakten Funktionalitäten der Förderdevices nach außen einheitlich angeboten. Intern erfolgt dann die für jeden Devicetyp nötige Umsetzung. Das DeviceManagement bildet also die unterteste Ebene einer Roasted Kit Steuerung.

9.13.1 Aufgaben und Voraussetzungen

Das DeviceManagement soll einen Hardware Abstraction Layer bereitstellen. Darüber hinaus sollen die einzelnen Devices über die Methoden *read()* und *write()* des DeviceService angesteuert werden, bzw. mit der Methode *describe()* beschrieben werden können. Auf Basis dieser Methoden werden also Steuerungsentscheidungen auf unterster Ebene umgesetzt. Weiterhin sollen die angeschlossenen Sensoren überwacht werden.
Um diese Aufgaben zu bewältigen benötigt das DM eine Anbindung an das Eventsystem des Roasted Kits. Ebenfalls muss es über die Subkomponenten UPnP- und StatusControl verfügen. Außerdem sollte das Betriebssystem echtzeitfähig sein und eine Treiberunterstützung für die angeschlossenen Devices bieten.

9.13.2 Architektur

Es sei bemerkt, dass das DeviceManagement und seine Subkomponenten in C++ entwickelt wurden und deshalb neben dem eigentlichen DeviceManagement gesonderte Implementierungen des Eventings, der Status- und der UPnPControl benötigt wurden.

Laufzeitumgebung

Das IPC-RTOS der Firma Komtron ist ein speziell auf den IPC angepasstes, echtzeitfähiges Linux Betriebssystem. Speziell angepasst heißt in diesem Fall, dass ein spezielles, vom Hersteller zur Verfügung gestelltes Board Support Package verwendet wird, das es unter anderem ermöglicht, mit Hilfe von Kernel-Modulen auf den K-Bus (siehe Anhang A) und damit auf die angeschlossene Hardware zuzugreifen.

SOAP-Engine

Im Gegensatz zu den anderen globalen Basiskomponenten (wie SystemManagement oder PacketManagement) läuft das DeviceManagement auf den IPCs. Deshalb ist es nicht ohne weiteres möglich, Axis als SOAP-Engine zu verwenden, da die Ressourcen auf den IPCs sehr begrenzt sind und auch die Möglichkeiten des verwendeten Betriebssystems äußerst eingeschränkt sind. Aus diesen Gründen setzt das DeviceManagement auf gSOAP als SOAP-Engine. gSOAP basiert auf C/C++ und ist äußerst performant.

9.13.3 OIS-U Daemon

Der OIS-U Daemon bildet die Schnittstelle zwischen der RFID-Anlage und dem RoastedKit Framework. Er wurde notwendig, da sich die CU.30 Zentraleinheit, an der die RFID-Antennen angeschlossen sind, nur von einer Instanz zur gleichen Zeit ansprechen lässt. Sie liefert über eine

geöffnete TCP-Verbindung anhand eines zurückgegebenen ASCII-Texts Ereignisse aller angeschlossenen Antennen, welche mit einer gesonderten Quittierung einzeln bestätigt werden müssen. Da im unteren Teil der Anlage aber schon zwei RFID-Antennen vorhanden sind, kommt es hier zu Konflikten, wenn z.B. zwei Sensortreiber gleichzeitig versuchen würden, sich mit der Zentraleinheit zu verbinden. Es wird aber nur eine gleichzeitige Verbindung erlaubt.
Die CU.30 Zentraleinheit wurde von uns so konfiguriert, dass sie selbsständig über die geöffnete TCP Verbindung *DataTagEvents*, also neu im Abstrahlwinkel der Antenne gescannte RFID-Tags, mit der ID des jeweiligen Tags sendet. Die Zentraleinheit bietet noch viele weitere Konfigurationsmöglichkeiten, hier sei auf das Installationshandbuch ([8]) verwiesen.

Aufgabe Die Aufgabe des OIS-U Daemon ist es, mit der CU.30 Zentraleinheit, an der die RFID-Antennen angeschlossen sind, zu kommunizieren und empfangene Events weiterzuleiten. Daher implementiert er einen Teil des Kommunikationsprotokolls aus [9].
Der Daemon bietet über den EventService (siehe Kapitel 9.2) die Möglichkeit für (Sub-)Komponenten des Frameworks Antennen-Events zu abonnieren. Speziell ist dies natürlich für die Sensorentreiber des SensorManagements gedacht, welchen hiermit eine einfache Möglichkeit geboten wird diese zu erhalten. (Sub-)Komponenten können Events der gewünschten Antennen einzelnen abonnieren.
Beim Lesen eines neuen RFID-Tags wird ein Event generiert, das die ID des Tags enthält, und mit der entsprechenden Topic (z.B. ANTENNA_1_SCANNED) an die Abonnenten verschickt.

Architektur Der OIS-U Daemon ist eine eigenständige Applikation, also keine Subkomponente des DeviceManagements. Er ist allerdings wie das Devicemanagement in C++ implementiert und enthält die von der BaseComponent geerbten Subkomponenten EventControl, StatusControl und UPnPControl. Nachdem die Verbindung zur Zentraleinheit aufgebaut wurde, wird der OIS-U Daemon per UPnP sichtbar geschaltet und die Treiberinstanzen können sich für die jeweiligen RFID-Antennen abonnieren.

9.13.4 Subkomponenten

Das DeviceManagement enthält die gleichen Subkomponenten, wie eine BaseComponent, d.h.

- EventControl
- StatusControl
- UPnPControl

Das DeviceManagement erbt diese Subkomponenten, da es eine von BaseComponent abgeleitete Komponente ist. Des Weiteren verfügt eine DeviceManagement Instanz über die folgenden „eigenen“ Subkomponenten:

- DeviceStateMapper
- SensorManagement
- LowLevelControlHandler

DeviceStateMapper

Der DeviceStateMapper setzt innerhalb des DeviceManagement den Hardware Abstraction Layer um. Dadurch bleibt den darüberliegenden Steuerungskomponenten (wie der Interpolation oder dem SystemManagement) der genaue Aufbau der Förderhardware, wie Motoren oder Lichtschranken verborgen. Lediglich der abstrakte Zustand (engl.: *State*) der Förderdevices ist nach

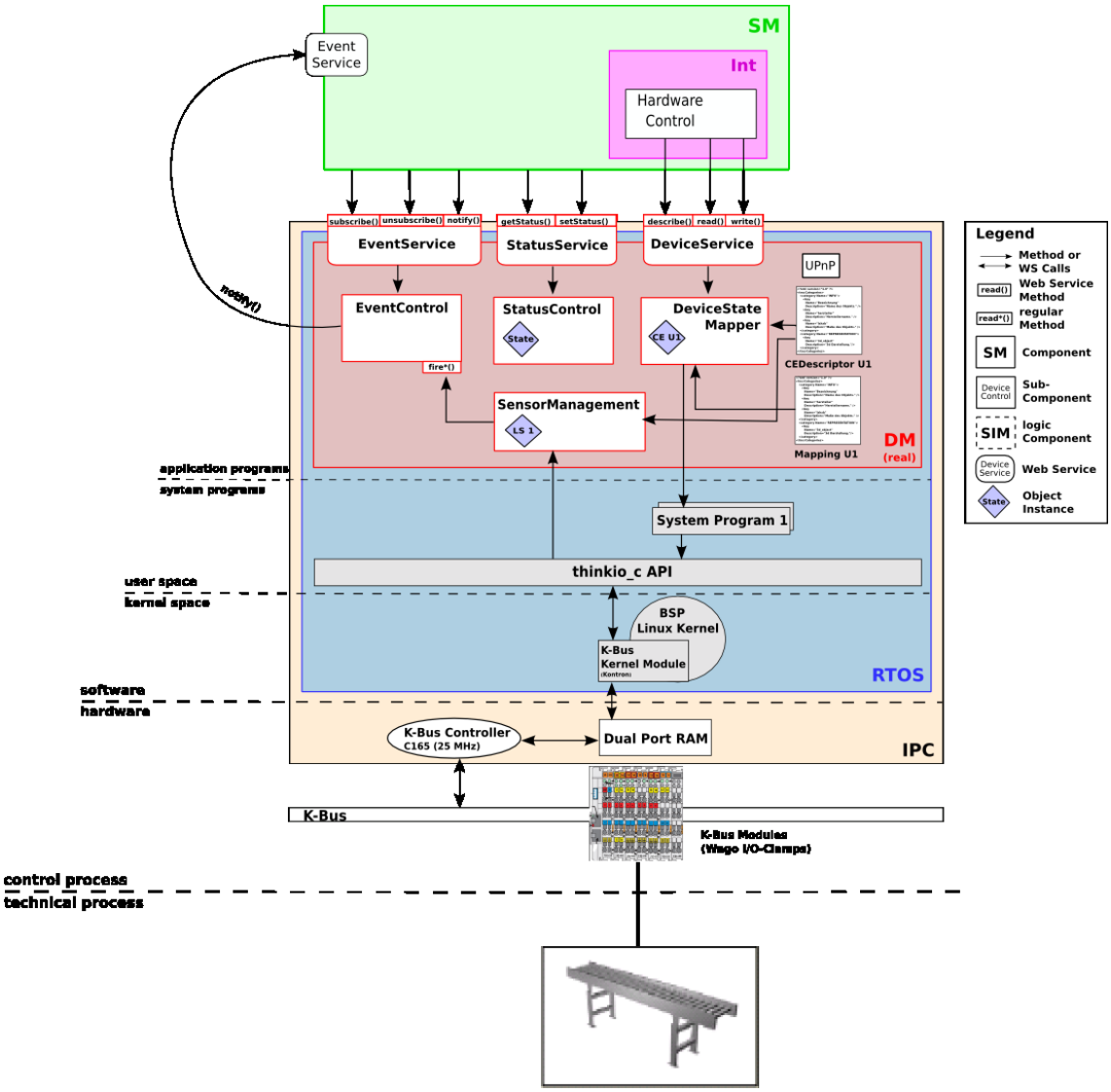

Abbildung 9.19: Schematische Darstellung des DeviceManagements auf einem IPC

oben sichtbar. Der Aufbau dieser abstrakten Zustände richtet sich nach der Art des Förderdevices. Der DeviceStateMapper bildet also im Grunde die Logik hinter den Methoden *read()* und *write()* der DeviceService-Schnittstelle. Dazu muss er die über die DeviceService-Schnittstelle transportierten Statusobjekte umsetzen/mappen. Auf der einen Seite müssen alle eintreffenden Statusobjekte (virtuelle Zustände) auf den zu steuernden technischen Prozess umgesetzt werden, und auf der anderen Seite müssen die realen Stati des technischen Prozesses interpretiert und in virtuelle überführt werden.

Anforderungen und Voraussetzungen des DeviceStateMappers Der DeviceStateMapper wurde unter folgenden Anforderungen entwickelt:

- Umsetzung des HAL innerhalb des DeviceManagements.
- Fehlnutzung verhindern, z.B. das Schreiben einer Lichtschranke.
- Aufwandsarme Portierbarkeit auf andere Fördertechnik.

Dazu braucht der DeviceStateMapper:

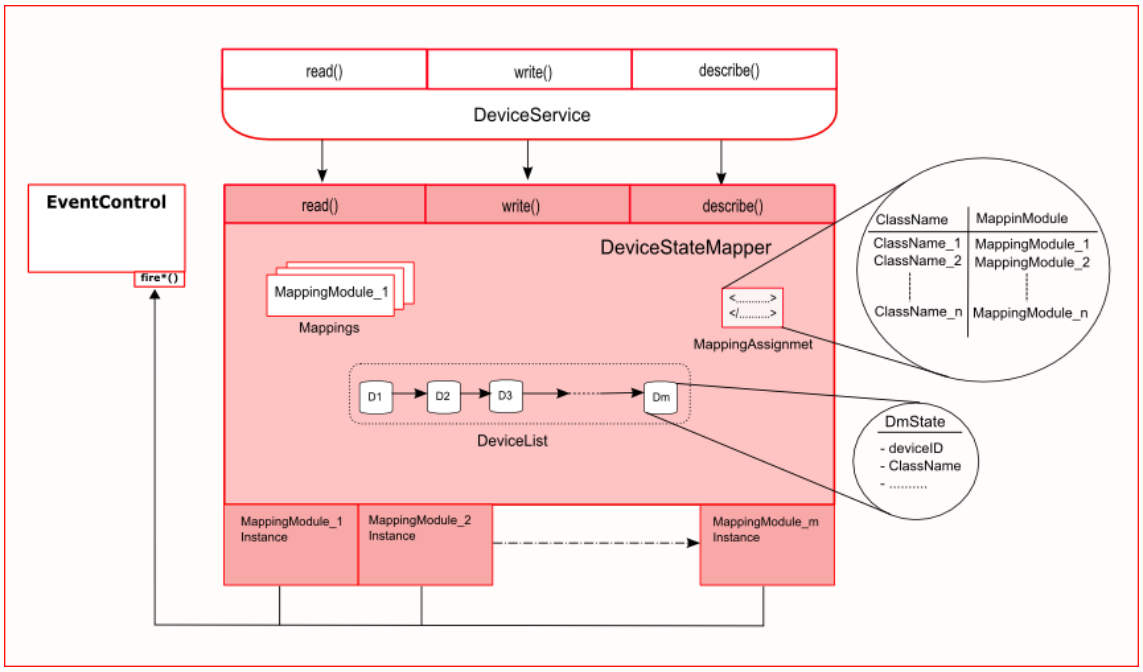

Abbildung 9.20: Schematische Darstellung des DeviceStateMapper

- Eine Liste der vorhandenen Förderdevice.
- Konfigurationsdaten, wie die Zuordnung zwischen Geräteklassen und Mappingmodulen.

Architektur des DeviceStateMappers Um den DeviceStateMapper möglichst "unabhängig" von der verwendeten Förderhardware zu machen, wurde seine Architektur flexibel gehalten. Deshalb werden die Zustände der einzelnen Devices nicht direkt vom DeviceStateMapper überführt, sondern in eigens dafür vorgesehenen Mappingmodulen, die dynamisch zur Laufzeit geladen werden können. Jedes Mappingmodul kümmert sich um eine eigene Geräteklasse. Momentan werden die folgenden Geräteklassen unterstützt:

- ***DigitalActor***: Alle Aktoren, die man lediglich ein- bzw. ausschalten kann.
- ***AnalogActor***: Alle Aktoren, deren Betriebszustand bzw. Geschwindigkeit reguliert werden kann.
- ***SwitchedActor***: Alle Aktoren, die mehrere symbolische Betriebszustände haben, wie z.B. Weichen.
- ***DigitalSensor***: Alle Sensoren, die als Sensorwerte lediglich 0 oder 1 zurückliefern, wie z.B. Lichtschranken.
- ***AnalogSensor***: Alle Sensoren, die komplexe Sensorwerte zurückliefern, wie z.B. RFID-Scanner.

Um eine neue Geräteklasse hinzuzufügen, die während der Konzeption nicht bedacht worden ist, muss lediglich ein neues Mappingmodul geschrieben werden und dies dem DeviceStateMapper bekannt gemacht werden.

Aufbau von Mappingmodulen Bei einem Mappingmodul handelt es sich um eine Klasse, die vom DeviceStateMapper zur Laufzeit nachgeladen werden kann. Um auch die Mappingmodule so flexibel wie möglich zu gestalten, werden die für das Mapping nötigen Treiberaufrufe ausgelagert. D.h. im Zuge des Mapping wird eine externe Applikation aufgerufen, das die nötigen Treiberaufrufe für das Mappingmodul durchführt. Auf diese Weise kann ein Mappingmodul einfach durch den Austausch dieser Applikation auf neue Förderdevices (gleichen Typs) angepasst werden. Allerdings muss das Mappingmodul dafür konfigurierbar gemacht werden.
Dazu müssen zumindest die folgende Information ohne Neukompilation konfigurierbar sein: Die Zuordnungen zwischen

- Device-IDs
- Zustandsausprägungen
- Aufrufe von externen Applikationen, die diese Zustandsausprägung herstellen bzw. einlesen.

Zusammengefasst muss ein Mappingmodul aus den folgenden Komponenten bestehen:

- *Objectcode des Moduls* (als shared library kompiliert, z.B. libMappingModule.so)
- *Konfigurationsdaten* (angepasst auf die Devices auf den jeweiligen IPCs)
- *externe Applikationen/Treiber* (zum Umsetzen der Zustände)

SensorManagement

Das SensorManagement verwaltet die Überwachung der einzelnen Sensoren (z.B. Lichtschranke, RFID-Anlage). Der Status der Lichtschranken wird in periodischen Abständen überwacht (engl.: *polling*), da diese keine Events verschicken können. Die RFID-Anlage hingegen ist in der Lage Events zu senden.
Außerdem beinhaltet das SensorManagement eine Zuordnungstabelle (engl.: *DriverAssignment*), welche die Type-IDs dem jeweiligen Treiber zuweist. Weiterhin verwaltet das SensorManagement die eigentlichen Treiber, und erzeugt die passenden Treiberinstanzen für die jeweiligen Sensoren. Jede Treiberinstanz verfügt über eine XML-Datei, welche die genaue Adresse des zu überwachenden Sensors kennt. Dies ist zum einen die KBUS-Addresse bei Lichtschranken und zum anderen die IP-Addresse des OIS-U Daemon, der Events der RFID-Sensoren übermittelt (siehe Kapitel 9.13.3).

Anforderungen und Voraussetzungen des SensorManagements Innerhalb des DeviceManagements dient der DeviceStateMapper der Erkennung von Sensorzustandswechsel durch periodische Überwachung oder per Benachrichtigung. Um adäquat funktionieren zu können, braucht der DeviceStateMapper eine Liste der vorhandenen Sensoren sowie eine Zuordnungstabelle der Typ-ID zum jeweiligen Treiber. Diese Konfigurationsdaten müssen durch das DeviceManagement zu Beginn der Initialisierung zur Verfügung gestellt werden.

Architektur des SensorManagements Das SensorManagement besteht aus eine Liste von Sensoren, einer Treiberzuordnung (engl.: *DriverAssignment*) und konkreten Treiber (siehe Abb. 9.21). Während der Initialisierung erhält das SensorManagement von ßeinem"DeviceManagement alle benötigten Informationen.
Alle SystemManagements, die sich über Sensoren-Zustände und Events der Devices benachrichtigen lassen wollen, registrieren sich per Web Service an der EventControl des jeweiligen DeviceManagements. Die Sensor-Treiber überprüfen (an den bekannten Hardware-Adressen) in periodischen Abständen von 10ms die angeschlossenen Sensoren auf Zustandsänderung. Die Information, welche Adresse angesprochen werden soll, holt sich der Treiber durch eine interne

XML-Datei, welche die Zuordnung enthält. Wird eine Änderung erkannt, z.B. das Auslösen eines Sensors durch ein Packet, so wird feuert das SensorManagement über die Subkomponente EventControl ein Ereignis, welches dann an alle registrierten Komponenten gesendet wird.

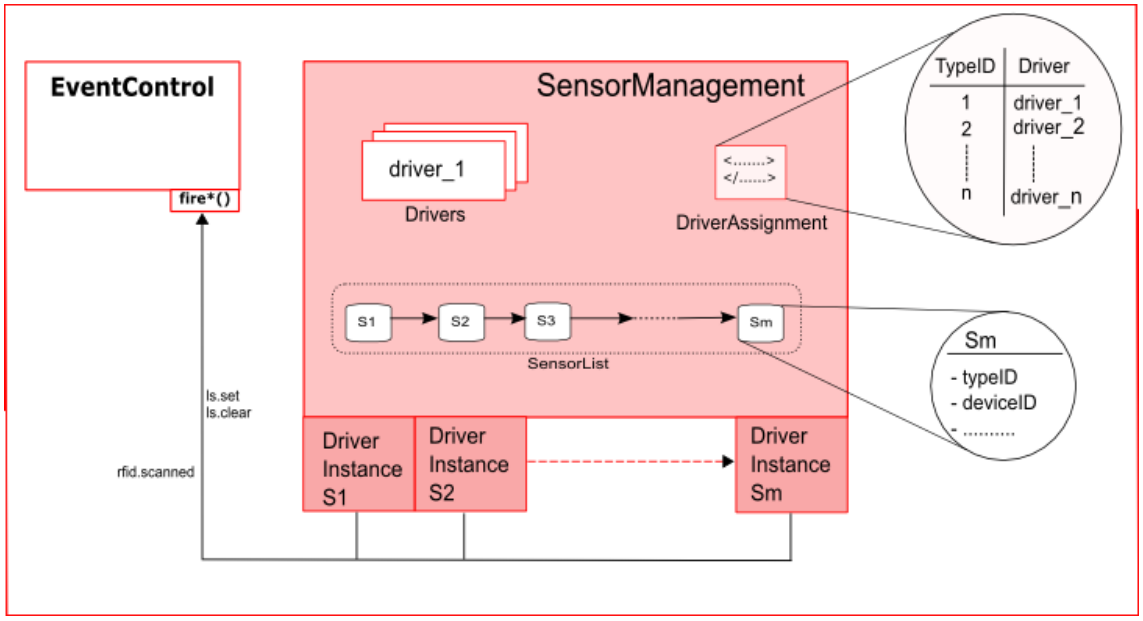

Abbildung 9.21: Schematische Darstellung des SensorManagements

LowLevelControlHandler

Innerhalb des LowLevelControlHandlers werden Events behandelt, deren Verarbeitung keine Verzögerung erlaubt. Dies ist z.B. der Fall, wenn Sensoren ausgelöst werden, die eine kritische Fehlersituation anzeigen (wie z.B. ein Kettenbruchsensor). Der Wunsch, spezielle Events direkt auf unterster Ebene zu behandeln, hat vor allem sicherheitstechnische Gründe.

Sonstige Subkomponenten

UpnPControl, EventControl und StatusControl sind analog zur BaseComponent entwickelt. Sie sind nur nicht in Java, sondern wie der Rest des DeviceManagement in C++ implementiert.

9.13.5 Konfigurationsdaten

Folgende Konfigurationsdaten werden benutzt:

- *CEDescriptor*: Beschreibung des zu steuernden Förderelements (ConveyingElement)
- *UPnPDescriptor*: Siehe Kapitel 5.4.4
- *TypeMapping*: Zuordnung zwischen Devicetyp und MappingModule
- *MappingModule Konfigurationsdateien*: Zuordnung zwischen Device und Treiberaufruf

Alle Konfigurationsdateien sind direkt auf den IPCs abgelegte XML Dateien. Jedem DeviceManagement werden initial die entsprechenden Dateien übergeben. Das Format der Konfigurationsdateien wird von einer entsprechenden XML-Schema-Datei definiert und kann gegen diese validiert werden.

9.13.6 Initialisierung

Jede Instanz des DeviceManagments kümmert sich genau um ein ConveyingElement, d.h. dass auf einem IPC stets mehrere Instanzen des DeviceManagements laufen. Initial bzw. beim Starten der IPCs muss es also eine Komponente geben, die für jedes angeschlossene Förderelement ein DeviceManagement erzeugt. Der Einfachheit halber geschieht dies durch ein Linux-*Initscript*, das fest in den Bootvorgang des verwendeten RTOS integriert wurde. Die Aufgabe dieses Skriptes ist es, für jeden vorhandenen CEDescriptor ein DeviceManagement zu erzeugen und diesem den Namen des CEDescriptors zu übergeben. Das DeviceManagement übernimmt dann das Parsing des CEDescriptors und erzeugt die nötigen Subkomponenten.
Schrittweiser Ablauf der Initialisierung:

1. Einschalten der IPCs über den Hauptschalter am Schaltschrank ES1.
2. Hochfahren des RTOS von den internen Flashkarten der IPCs.
3. Start des Initscripts.
   - Überprüfung des CEDescriptor Verzeichnisses,
   - Start von DeviceManagement Instanzen (eine für jeden Descriptor),
4. Start der einzelnen DeviceManagments. Dazu sind jeweils die folgenden Schritte notwendig:
   - Parsing und Validierung des CEDescriptors.
   - Erzeugen des CE Objekts.
   - Start des DeviceStateMapper und des DeviceService (Übergabe: Liste der vorhandenen Devices).
   - Start von EventControl und des EventService.
   - Start von StatusControl und des StatusService.
   - Erzeugen des SensorManagement (Übergabe: Referenz auf EventControl und Liste der vorhandenen Sensoren)
   - Bekanntmachung des Device-, Event- und Status-Services per UPnP

9.13.7 Schnittstellenbeschreibung

DeviceService

Das DeviceManagement repräsentiert innerhalb des Steuerungssystems jeweils ein Förderelement, also die Fördertechnik, Aktoren und Sensoren. Daher bietet der durch ihn angebotene DeviceService als Dienstleistung die Steuerung des Förderelements. Dazu kann man in drei Subdienste unterteilen:

Gerätebeschreibung abrufen Mit Hilfe dieses Subdienstes kann man eine ausführliche Gerätebeschreibung anfordern. Daraufhin sind die Geräte inkl. ihrer IDs bekannt, so dass Zustandsänderungen (z.B. Motor einschalten) durchgeführt werden können.

Gerätezustand lesen Will man den aktuellen Zustand eines Gerätes auslesen, so kann man diesen Subdienst nutzen; dies wird insbesondere für Sensoren gemacht.

Gerätezustand schreiben Dieser Subdienst bietet das Schreiben von Gerätezuständen, so dass Zustandsänderungen durchgeführt und somit z.B. ein Motor eingeschaltet werden kann.

Eine detaillierte Schnittstellenbeschreibung befindet sich im Anhang B.13 auf Seite 135.

10 Testphase, Installation & Erprobung

In der Testphase wurde die von der Projektgruppe entwickelte Software getestet. Die Tests waren in verschiedene Phasen eingeteilt. Zuerst wurde die Funktionalität der einzelnen Klassen getestet, diese dann zu ihren Steuerkomponenten zusammengefasst und integriert getestet. Anschließend wurde die gesamte Förderanlage mit Hilfe der Interpolation (siehe Kapitel 9.12) simuliert und virtuelle Pakete hindurchgeschleust. Die letzte Testphase bestand in der Integration der DSL-Komponenten mit der realen Hardware und den Testfahrten realer Pakete.

10.1 Testszenarien

Die programmierten Klassen wurden mit JUnit auf ihre Funktionalität getestet. Zu jeder Klasse wurde ein Test geschrieben, der anhand vorgegebener Eingaben überprüft, ob die zurückgelieferten Ergebnisse mit den erwarteten Werten übereinstimmen. Dazu wurden gezielt Dummy-Objekte erstellt, diese mit Daten gefüllt und anschließend geprüft, ob diese Objekte auf die erwartete Weise im System genutzt werden.
Im Anschluss an die Klassentests wurden die Komponenten und ihr Zusammenspiel getestet. Dazu wurde eine vernetzte Testumgebung aufgebaut, in der auch die Kommunikation über UPnP und Web Services getestet wurde. So wurde überprüft, ob sich die einzelnen Komponenten per UPnP gegenseitig finden und ob sie sich korrekt initialisieren. Die nächsten Tests bestanden im Erstellen eines virtuellen Paketes, das durch die Anlage geschickt wurde. Auf diese Weise konnten die Abläufe im System getestet werden. In dieser Phase wurden die meisten Fehler gefunden und behoben. Hierbei wurde auch das Monitoring getestet, welches durch die grafische Anzeige des Pakets direkt ekennen ließ, wenn ein Paket irgendwo s“festhing“ und nicht dahin fuhr, wo es hinfahren sollte.
Die letzten Tests waren das Starten der Software an der realen Anlage. Die Software wurde in Teilen auf den IPCs und auf verschiedenen Laptops installiert. Daraufhin fanden sich ohne Probleme die separaten C++- und Java-Teile. Auf einem Rechner wurde zur Kontrolle das simultan mitlaufende Monitoring gestartet. Danach wurden die einzelnen Komponenten initialisiert und die Software ein erneutes Mal auf ihre Funktion geprüft, diesmal mit einem realen Paket.
Nach Abschluss der letzten Testphase konnte die Anlage in einen Zustand gebracht werden, in dem die Funktionsfähigkeit des Prototypen gezeigt werden konnte. Welche Probleme sich bei den Tests ergaben und welche davon noch nicht gelöst wurden, wird in den folgenden Abschnitten detailliert dargestellt.

10.2 Axis

Zu Anfang gab es Kompatibilitätsprobleme zwischen den SOAP-Engines Axis(Java) und gSOAP(C++). Gelöst wurde das Problem, indem das SOAP Protokoll auf „RPC literal“ umgestellt wurde. Außerdem musste die neueste Entwicklerversion von Axis benutzt werden, da mit den anderen Versionen Vererbungshierarchien nicht korrekt übertragen wurden. Als ein weiteres Problem stellte sich die Performance von Web Services heraus, die mit Hilfe von Axis unter MS Windows realisiert wurden. Hier benötigte ein einzelner Web Service Aufruf bis zu 5 Sekunden, auf Linux-Systemen hingegen lediglich ca. 8 Millisekunden. Als Konsequenz wurden bei den weiteren Tests keine Windowsrechner mehr eingesetzt.

10.3 Leistung

Zur problemlosen Steuerung der Anlage wurden mindestens 2 leistungsfähige Notebooks mit je 1 Supervisor benötigt. Durch Hinzunahme weiterer Rechner ließ sich die Last problemlos verteilen. Es gab allerdings Faktoren, welche die Skalierbarkeit einschränken. Der Flaschenhals unserer Steuerungssoftware war die Interpolation, da jedes Paket, das sich in der Förderanlage befand, einen gesonderten Interpolationsthread brauchte. Diese Threads verbrauchten viel Systemressourcen, d.h. die Anzahl der Pakete, die sich gleichzeitig im System befinden können, war allein durch die Interpolationsthreads begrenzt.
Für die Visualisierung der Anlage in Java3D benötigten wir einen leistungsstarken Rechner mit einer entsprechend starken Grafikkarte. Durch die Kamerafahrten gelangten wir trotzdem an die Leistungsgrenzen unserer Hardware. Hier gibt es sicher noch viel Raum für Optimierungen.
Die Web Service Aufrufe waren zumindest in den Testläufen innerhalb des Systems schnell genug, um die nötigen Steuerentscheidungen rechtzeitig zu treffen. Allerdings wurden nie zwei Pakete direkt hintereinander in die Anlage geschickt. Wieviel Pakete zu einem bestimmten Zeitpunkt in dem System sein können, müsste mit weiteren Tests herausgefunden werden.
Bei UPnP könnte noch ein Problem mit der Netzwerklast entstehen, da UPnP über Broadcastnachrichten läuft, die regelmäßig an alle anderen Komponenten übertragen werden. So sorgt jede zusätzliche Komponente innerhalb eines IP-Netzwerkes für zusätzliche Netzlast. In den Testläufen konnte noch nicht endgültig getestet werden, wie viele Komponenten ein 100 MBit LAN unterstützt, es sollten theoretisch aber ausreichend viele auch für große Förderanlagen sein. Aufgrund der hohen CPU-Auslastung auf den IPCs musste kurz vor Schluss noch die UPnPImplementierung unter C++ gewechselt werden. Auch die Java-Implementierung von CyberLink musste mehrmals angepasst werden, um unseren Anforderungen zu genügen.

10.4 Flexibilität

Prinzipiell sollte die Steuerungssoftware so flexibel sein, dass jede Komponente sauber gestartet, pausiert und gestoppt werden kann. Dieses funktionierte in der Testphase einwandfrei, mit Ausnahme des PacketManagements. Hier liegt das Problem an den sich gegenseitig aufrufenden Web Service Aufrufen. Sobald ein PM beendet wird, steht es noch in den Routingtabellen der anderen PM's, auch bekommt das eigene SM nicht mit, dass es nun kein PM mehr hat, und versucht weiterhin, die Scouts nach „oben“ zu schicken. Aus diesem Grund war es nur möglich, flüchtige Intermediates zu erstellen, nicht jedoch, diese nach Beendigung des Jobs wieder zu entfernen. Wenn der Supervisor feststellte, dass eine Komponente verlorengegangen war, war er in der Lage, diese neu zu erstellen. Aufgrund der in der Testphase gemachten Erfahrungen können wir feststellen, dass der Ausfall von SystemManagement, PacketManagement und OrderManagement grundsätzlich zu beheben ist. Fällt ein Supervisor aus, werden alle von diesem verwalteten Komponenten von den anderen Supervisorn übernommen und, falls es sich um den PSV handelte, auch ein neuer PSV gewählt. Probleme gab es, wenn Komponenten mit der gleichen ComponentAddress auftauchten, oder wenn (z.B. wenn nur ein einziges PM existierte und sich dieses über das Scouting selbst als Nachfolge-PM fand) eine Komponente die eigene ComponentAddress auflösen sollte. Dieses Problem wurde durch eine Anpassung der UPnP-Control gelöst.

10.5 Aufsetzen des Systems und Handhabung

Um Roasted Kit zu nutzen, müssen zuerst die DeviceManagements auf den IPCs gestartet werden. Zusätzlich startet man die Typelibrary auf einem beliebigen Rechner, ebenso das Monitoring. Auf einer weiteren beliebigen Anzahl von Rechnern startet man nun jeweils einen Supervisor.

Hierbei muss darauf geachtet werden, dass in der Konfigurationsdatei der Utilizationwert hoch genug ist, um für alle gewünschten Förderelemente Steuerkomponenten zu erstellen. (siehe Kapitel 9.8. Nun werden automatisch alle Komponenten initialisiert und sollten im Monitoring korrekt angezeigt werden. Hier kann man nun durch Drücken des Startknopfs die Anlage starten, es werden dann alle Fördermodule, von hinten beginnend, gestartet. Der nächste Schritt besteht im Starten der Orderform auf einem beliebigen Rechner, hier gibt man die gewünschten Aufträge mit ihren Jobs ein, mit der jeweils zum Job gehörigen PaketID, Wareneingang, Warenausgang und optional Intermediates. Anschließend stellt man das Paket in den Wareneingang und sieht es - hoffentlich - zum gewünschten Ausgang fahren. Parallel dazu kann man im Fenster des Monitoring die Events sehen. Optional kann man noch auf einem Rechner das CommonLog starten. Hier sieht man dann an einer zentralen Stelle alle Logausgaben aller verteilten Rechner.

11 Schluss

11.1 Fazit

11.1.1 Erreichte Ziele

Das Minimalziel der PG 475 war die Entwicklung einer prototypischen, dezentralen, auf Web Services basierenden Steuerung für Förderanlagen. Da mit der hier vorgestellen Förderanlagensteuerung Roasted Kit ein solches System geschaffen wurde, ist das Minimalziel erreicht worden. Während der Projektgruppen-Zeit sind diverse unerwartete Probleme aufgetreten, die dann in Zeitproblemen bzgl. der Zielerreichung resultierten. Dadurch konnte leider das erweiterte Ziel - die Identifizierung von Problemfeldern und Überführung in ein Framework - nicht vollständig erreicht werden. Da am Ende der PG leider die Zeit fehlte, um die entwickelten Bausteine zu einem Framework zusammenzufassen. Dafür wären wohl noch ein weiterer Monat Arbeit von Nöten gewesen.

Anforderungen an die Steuerung

Dezentralität Aufgrund der flexiblen Architektur ist die Anforderung eines dezentralen System mehr als erfüllt. Das System kann sowohl serverorientiert als auch als komplett verteiltes System agieren. Desweiteren kommunizieren die einzelnen Subkomponenten per Web Service untereinander (auch lokal); dies bedeutet, dass über einen komplett verteilten Ansatz hinaus auch die Subkomponenten eines Steuerrechners rechnerunabhängig aktiv sind und theoretisch auf andere Rechner übertragen werden könnten. Der Grad der Dezentralisierung wird dadurch enorm gesteigert.

Modularität Die hier vorgestellte Steuerung ist äußerst modular aufgebaut. Durch die Unterteilung in wichtige Systemkomponenten und Definition von Schnittstellen, sowie einheitlicher "Inter-Komponenten-Kommunikation", basierend auf Web Services, konnte auch diese Anforderung erfolgreich erfüllt werden.

Autonomität Die Anforderung der Autonomität wurde erfüllt, indem die Initialisierungsphase komplett automatisiert wurde. Dies geschah durch generische Systemkomponenten; ausschließlich *CEModule* (Java-Modul) sowie *CEDescriptoren* (XML-Datei) sind förderanlagenspezifisch und müssen generiert sowie an die einzelnen *DeviceManagements* bzw. *SystemManagements* verteilt werden. Mit Hilfe dieser zwei Dateien wandelt sich die vorher generische Steuerung in eine, die speziell für die entsprechende Anlage angepasst wurde. Dadurch werden alle Daten zur Verfügung gestellt, die eine automatische Systeminitialisierung ermöglichen.

Adaptivität Die Anpassung an sich ändernde Topologie war von Anfang an einer der wichtigsten Punkte, auf die während der Entwicklung geachtet wurde. Bereits die Initialisierungsphase basiert auf dem sequentiellen Hinzufügen von DeviceManagements zu der Steuerung. Ein Abschluss dieser Phase wird gewissermaßen nie erreicht, da jederzeit Komponenten hinzugefügt oder entfernt werden dürfen. Die Technologie *Universal Plug and Play (UPnP)* macht der Steuerung dazu Änderungen bei Kommunikationsteilnehmern bekannt. Dadurch wurde also auch dieser Punkt vom System voll erfüllt.

Fehlertoleranz Da jederzeit Komponenten (unabhängig ob Software- oder Hardwarekomponenten) ausfallen dürfen (siehe Adaptivität), ist die Anforderung der Fehlertoleranz implizit erfüllt. Der Ausfall von Komponenten wird per Eventsystem allen Teilnehmern mitgeteilt, so dass auf Fehlersituationen reagiert werden kann. Die Supervisor erstellen ausgefallene Komponenten bei Bedarf neu.

Proaktivität Da viele Informationen, insbesondere zeitkritische Steuerungsinformationen wie z.B. die Wahl des vorhandenen Weges (durch Routing-Subkomponenten), vorab berechnet werden, ist ein proaktives System gegeben.

11.1.2 Weitere Erkenntnisse

Neben der oben genannten Erfüllung des Minimalziels sowie der Anforderungen wurden weitere technologische Erkenntnisse gewonnen.

Qualitativ hochwertige Arbeit An dieser Stelle soll noch einmal etwas erwähnt werden, was eigentlich obligatorisch sein sollte, aber dennoch oft nicht durchgeführt wird: die Ablieferung qualitativ hochwertiger Arbeit.
Zu oft passiert es, dass ein PG-Mitglied schlichtweg "keine Lust auf Arbeit" hat, so dass über die Qualität der Arbeit nicht nachgedacht wird. "Hauptsache fertig" ist dann das Motto. Diese "gesparte Zeit" muss später leider wieder durch "Mehrarbeit", meist anderer PG-Mitglieder, ausgeglichen werden. Zusätzlich überwiegt im Allgemeinen diese "Mehrarbeit" die vorher "gesparte Zeit", was in einem erhöhten Zeitbedarf bzw. in einer Kostensteigerung (in wirtschaftlich arbeitenden Unternehmen) resultiert. Bei der Durchführung von Softwareprojekten kann qualitativ schlechte Arbeit sogar dazu führen, dass ganze Konzeptänderungen durchgeführt werden müssen, da aufgrund schlechten Quellcodes sowie Zeitdruck das geplante Konzept nur unter großem Aufwand eingehalten werden kann. Dies kann zu Ergebnissen führen, die nur noch Teile der eigentlichen Zielstellung erfüllen oder diese ineffizient erfüllen.
Die Lösung für dieses Problem ist trivial: es sollte immer darauf geachtet werden, qualitativ hochwertige Arbeit abzuliefern. Dies bedeutet, die *Qualität* als primäres Ziel anzusehen und nicht die *Zeit*, jedoch unter Berücksichtigung des Faktors Zeit als sekundäres Ziel. Dadurch erreicht man repräsentative Ergebnisse und doppelte Arbeit wird vermieden.

UPnP Die Technologie *Universal Plug and Play (UPnP)* wirkt anfangs überaus vielversprechend. Geboten wird (angeblich) automatisches Auffinden von Netzwerkteilnehmern, und von evtl. angebotenen Diensten, die softwaretechnisch genutzt werden können und dies alles in einem IP-basierenden Netzwerk mit einer automatischen IP-Adressenvergabe (AutoIP/DHCP) - soweit die Theorie. Die Realität sieht jedoch viel problematischer aus. Während der Arbeit der PG475 mit UPnP stellte sich heraus, dass derzeit keine stabil arbeitende UPnP-Bibliothek für Java vorhanden ist. Die genutzte UPnP-Implementierung (Cyberlink von Cybergarage) zeigte höchst unerwartetes, teilweise nicht-deterministisches Verhalten, so dass viele Stunden damit verbracht wurden, Fehler in *externer Software* zu beheben. Das in C++ geschriebene *DeviceManagement* der hier vorgestellen Steuerung wechselte sogar während der Implementierungs-/Testphase von *Cyberlink* hin zu der UPnP-Implementierung von Intel (siehe Kapitel 9.1).
Soll UPnP stabil und effizient genutzt werden, ist also entweder vorerst eine Recherche notwendig, um eine stabile Implementierung zu finden, oder es wird eine eigene Bibliothek entwickelt. Diese könnte komplett neu geschrieben sein oder natürlich eine vorhandene erweitern, anpassen oder korrigieren. Desweiteren hat man natürlich die Möglichkeit der Nutzung von Alternativen wie zum Beispiel *ZeroConf*. Diese Technologie bietet die gleichen Möglichkeiten wie UPnP, also automatische IP-Adressenvergabe sowie Auffinden von Diensten im Netzwerk. Die Zuverlässigkeit der jeweiligen Implementierung müsste aber wie bei UPnP vorab überprüft werden, um unnötigen Zeit-/Kostenaufwand zu vermeiden.

Java Die Nutzung der Programmiersprache Java für zeitkritische Anwendungen, wie die hier vorgestelle Steuerung, ist eher kritisch zu betrachten. Java als Hochsprache für Anwendungsentwicklung ist nicht dafür ausgerichtet, zeit- oder performancekritische Aufgaben zu bewältigen. Seit der Einführung von Java wurde dessen Geschwindigkeit zwar enorm verbessert, kommt aber leider immer noch nicht an C++ heran. Des Weiteren sind die Möglichkeiten der Optimierung bei Java im Vergleich zu C++ doch eher als gering anzusehen. Dies lässt sich am Beispiel der Speicherverwaltung sehr gut darstellen. Java hat auf der einen Seite durch den *GarbageCollector* eine komfortable Art entwickelt, dem Programmierer Arbeit bzgl. Speicherverwaltung abzunehmen und somit Fehlerquellen zu vermeiden. Auf der anderen Seite entsteht dadurch ein Kontrollverlust, da es in Java schwieriger (als bei C++) ist, den exakten Zeitpunkt zu bestimmen, an dem Speicher freigegeben werden soll. Genauer gesagt, kann man nur den *GarbageCollector* manuell aufrufen, die Routinen des *GarbageCollectors* sind jedoch nicht beeinflussbar.
Während der Arbeit an der *Interpolation* (siehe Kapitel 9.12) ist aufgefallen, dass das scheinbar kleinste durch Java identifizierbare Zeitintervall ca. 50ms beträgt. Diese ist natürlich für Anwendungen mit "Echt-Zeit-Anforderungen", also anwendungsspezifischen "Zeitschranken", nicht ausreichend und könnte Java als eingesetzte Programmiersprache ausschließen. Eine Hilfe könnte allerdings eine Echt-Zeit-Erweiterung für Java bieten, die während der Projektgruppen-Phase publik gemacht wurde.
Abschließend lässt sich zusammenfassen, dass die Entwicklung zeitkritischer Anwendungen mit Java möglich ist, jedoch benötigt der Faktor Performance besondere Aufmerksamkeit. Dies gilt nicht nur für die Architektur der entsprechenden Software, sondern auch für Programmierstil und eingesetzte Technologien.

Web Services Ähnlich wie bei UPnP oder Java, sind auch für Web Services vorab Recherchen zu empfehlen, da sich die Implementierungen stark unterscheiden können. Insbesondere die Performance, also die Geschwindigkeit des *Parsings* der per HTTP versendeten XML-Nachrichten, kann sehr variieren. So ist zum Beispiel das Problem enstanden, dass Web Service Aufrufe extrem verlangsamt wurden (bis zu 5 Sekunden pro Aufruf), sobald sich ein Rechner mit dem Microsoft Windows Betriebssystem als Kommunikationsteilnehmer im Steuerungverbund befand. Nur wenn auschließlich Linux benutzt wurde, wurden Web Service Aufrufe effizient abgearbeitet (ca. 8 Milisekunden, also ca. um den Faktor 625 schneller). Effizient arbeitende Web Service Bibliotheken sind zwar vorhanden, sowohl für Java als auch für C++, diese unterscheiden sich aber zum Teil stark in der Benutzbarkeit, Stabilität sowie dem Funktionsumfang.

Nebenläufigkeit & Threads Eine weitere Erkenntnis, die jedes Mitglied der Projektgruppe 475 erlangt hat, ist die Nutzung von Threads in Verbindung mit Web Services. Da in der hier vorgestellten Steuerung jeder Web Service Aufruf als eigener Thread realisiert wird und die gesamte Kommunikation auf Web Services basiert, werden äußerst viele Web Service Aufrufe durchgeführt und somit Threads generiert. Neben der Kommunikation an sich, müssen weitere Threads/Prozesse aktiv sein, die steuerungsinterne (Verwaltungs-)Aufgaben durchführen.
Jede Dienstnutzung resultiert also neben der "eigentlichen Arbeit" (*XML-Parsing* etc.) noch in Betriebssystemoperationen (Thread-Erzeugung /-vernichtung), die weiterhin Zeit und Rechenleistung benötigen. Eine "intelligente" Nutzung von Diensten ist also ein wichtiger Punkt, der beim Entwurf der Architektur berücksichtigt werden muss, um das System leistungsfähig zu machen.

Supervisor Wie schon in Kapitel 9.8 erwähnt, dient der Supervisor dazu, die für die Steuerung benötigten Komponenten zu erzeugen. Da mehrere Supervisor und somit mehrere Steuerrechner vorhanden sein können bzw. sollen, ist eine Absprache unter diesen notwendig, um Dateninkonsistenz zu vermeiden.
In der vorhandenen Steuerung wird ein temporärer Server - der *Primärsupervisor* - gewählt, der als eine Art Server agiert. Entscheidungen werden nur vom *Primärsupervisor* durchgeführt bzw.

bestätigt und danach jedem Supervisor bekannt gemacht. Während der Testphase hat sich jedoch herausgestellt, dass dieser Ansatz diverse Nachteile hat. So wird bei der Initialisierungsphase der erste Supervisor sich selbst als Primärsupervisor wählen, da er zu dem Zeitpunkt noch der einzige Supervisor ist. Kommen weitere Supervisor hinzu, was normalerweise der Fall ist, stellt sich zum Beispiel die Frage, ob jemals ein neuer Primärsupervisor gewählt werden soll, sofern der Steuerrechner des Primärsupervisors nicht ausfällt.
Eine andere Realisierungsmöglichkeit ist ein komplett verteilter Ansatz, bei dem es keinen (temporären) Server gibt. Für die Entscheidungsfindung fragt jeder Supervisor bei jedem anderen an und erbittet eine Bestätigung. Stimmen alle zu, so ist die Entscheidung bestätigt. Der Vorteil ist hier zwar die Flexibilität, der Nachteil jedoch die Anzahl der Web Service Aufrufe, die dann für n Supervisor bei $n^2 - n$ pro Entscheidung liegt.
Man hat weiterhin die Möglichkeit, die DSL-Ebene in die Hardware-Steuerungs-Ebene zu verschieben (siehe Kapitel 8), also die Supervisor und die entsprechenden Systemkomponenten auf den IndustriePCs (IPCs) starten. Dadurch würden viele Web Service Aufrufe lokal ablaufen, was schließlich die Performance erhöht. Ob die Steuerung überhaupt auf den IPCs lauffähig ist, müsste jedoch getestet werden.
Man sieht also, das hier vorgestelle Architekturkonzept bietet viel Flexibilität, jedoch auch viele Optimierungsmöglichkeiten. Hier ist also ein *Trade-Off* zwischen Flexibilität und Performance zu erkennen. Somit sind noch weitere Untersuchungen nötig, um das beste Flexibilität-Performance-Verhältnis herauszufinden.

Simulation Am Ende der Projektgruppen-Zeit konnte man erkennen, wie wichtig bzw. hilfreich eine Simulation gerade in Verbindung mit einer Visualisierung der Anlage ist. Eventuelle "Denkfehler" (z.B. die Nutzung falscher Routen) kann man schnell erkennen, indem man die internen Stati dank der Visualisierung direkt auf Korrektheit prüfen kann.
Durch eine Simulation hat man natürlich auch erst die Möglichkeit, nicht vorhandene, also virtuelle, Systeme zu testen. Hier müsste man lediglich passende XML-Dateien erstellen und die Steuerungsmodule hierfür schreiben. Erst dadurch werden umfangreiche System- und Ausführungstests möglich, da man beliebige Systeme mit nur sehr geringem Kosten-/Zeitaufwand testen kann. Leistungsgrenzen, Skalierbarkeit und Anpassungsfähigkeit lassen sich so problemlos für beliebige Systemtopologien empirisch verifizieren.

3D-Visualisierung Eine 3D-Visualisierung bietet neben der grafischen Erweiterung der Simulation noch zwei Eigenschaften, die für die PG475 bzw. alle Softwareprojekte wichtig sind. Im Allgemeinen bleibt die eigentliche Arbeit bei Softwareprojekten dem Betrachter/Benutzer verborgen. Obwohl beispielsweise jahrelange Arbeit in eine Software investiert wurde, könnte lediglich eine Zeile mit dem Ergebnis ausgegeben werden. Die geleistete Arbeit durch die Programmierer ist für den Benutzer zunächst nicht sichtbar. Dementsprechend uninteressant bleibt die Software. Eine gute Software ist nutzlos, wenn man sie nicht "gut verkaufen kann" und sie dadurch nicht eingesetzt wird.
Bei der PG475 macht die 3D-Visualisierung dem Betrachter sichtbar, dass es wirklich eine Steuerung gibt, die auf einer virtuellen Anlage virtuelle Pakete transportiert. Die 3D-Visualisierung kann somit als eine Art "Aushängeschild" dienen, um (mit der Software nicht vertrauten) Dritten die geleistete Arbeit auf einfache, aber dennoch ansprechende sowie Interesse weckende Art und Weise zu präsentieren. Somit dient die 3D-Visualisierung dem Zweck, das Ergebnis einer Gruppe von Informatikern, vor allem fachfremden Personen näher zu bringen.
Ein weiterer Punkt, den man während der Projektgruppen-Zeit beobachten konnte, war die Wirkung der 3D-Oberfläche auf die Projektmitglieder. Die Mitarbeit an einem Projekt, das man stolz anderen präsentieren kann, produzierte einen Motivationsschub für alle Beteiligten. Oft hörte man etwas wie "hab ich meinen Freunden gezeigt" oder "sieht super aus" von den Projektmitgliedern. Dieser Motivationsschub führte nicht nur zu einer Erhöhung der Arbeitsleistung

bzw. -bereitschaft, sondern verstärkte auch soziale Beziehungen in Form der Zusammenarbeit und Zusammenhalt der Gruppe. Desweiteren wurde das Interesse vieler Projektmitglieder an der 3D-Modellierung geweckt, so dass die außeruniversitäre Weiterbildung im Bereich der 3D-Modellierung gefördert wurde.

11.2 Perspektiven

Die Arbeit der PG475 bietet viele Perpektiven für weitere Projektgruppen bzw. andere Teams. So könnten wesentlich mehr verteilte Algorithmen implementiert werden, um das Management der Web Services zusätzlich zu unterstützen bzw. zu vereinfachen. Ein einheitlicher Takt würde beispielsweise helfen, die nebenläufigen Prozesse synchron zu halten und deren zeitliche Abfolge zu ordnen. Ein Schnappschuss-Algorithmus könnte zur Statusüberwachung/Monitoring dienen, indem die Stati für u.a. Schlüsselzeitpunkte festgehalten und verglichen würden. Ein solcher Algorithmus nimmt aber auch eine wichtige Rolle während der Enwicklung ein. So wird das Beheben von Programmierfehlern (*debugging*) erleichtert, da man für ein verteiltes System in einer übersichtlichen Form Status- und somit Laufzeitinformationen zur Verfügung hat.
Der Supervisor bietet für die hier vorgestellte Steuerung äußerst viele Verbesserungsmöglichkeiten. Als zentrale Komponente wirken sich dort durchgeführten Änderungen stark auf das Steuerungsverhalten aus. So können schon kleinere Änderungen die Gesamtperformance erhöhen. Die Wahl eines Primärsupervisors könnte man (statt des hier verwendeten zentralen Ansatzes) komplett verteilt durchführen, was die Flexibilität der Wahlen steigern würde. Desweiteren lassen sich dann die Performance-Werte mit der hier vorgestellten Version vergleichen um die bessere Alternative hinsichtlich verschiedener Aspekte abschätzen zu können. Ein wichtiger Punkt der Verbesserung des Supervisors ist die Zuständigkeitsübertragung von Komponenten/Förderlementen. Dies hätte enorme Vorteile hinsichtlich der Flexibilität sowie Ausfallsicherheit. Ist eine Zuständigkeitsübertragung möglich, so könnte eine faire Zuordnung (also eine Zuordnung in Abhängigkeit der Rechnerperformance) der Förderlemente zu den Rechnern der Steuerungsebene durchgeführt werden. Desweiteren ist erst dann die Erstellung einer hochdynamischen Steuerungsebene möglich. Das bedeutet, dass jede Möglichkeit der Steuerungstopologie erreicht werden kann, von einer zentralen Steuerung (insgesamt ein Steuerrechner) bis hin zu einem komplett verteilten Ansatz (ein Steuerrechner pro Förderlement). Ausfälle von Steuerrechnern könnten dann kompensiert werden, indem die Zuständigkeit übertragen wird. Die Steuerung an sich versagt erst dann, wenn komplett alle Steuerrechner ausgefallen sind. Diese Beispiele waren nur mögliche Perspektiven und sollen verdeutlichen, dass die Arbeit der PG475 noch viel Potential für weitere Arbeiten bietet.

11.3 Persönliche Sichten der Beteiligten

Projektmitglied 1

Rückblickend auf die PG lassen sich verschiedene positive als auch negative Aspekte entdecken. Positiv sind sicherlich das Gesamtkonzept, eine praktische Teamarbeit und die relativ große Eigenverantwortung hinsichtlich der Themenrichtung. Man lernt einmal etwas Praxisnahes, was später im Arbeitsleben nützlich sein kann. Negativ war die unterschiedliche Arbeitsaufteilung unter uns Studenten, die zu unterschiedlichen Arbeitszeiten und Anteilen am Gesamtprojekt führte. Hier fehlten wohl auch der Druck und das Gefühl, dass die Möglichkeit besteht, den Schein nicht zu erhalten. Die Seminarthemen hätten besser zum Thema der PG passen müssen, damit man eine wirkliche Basis direkt zu Beginn der PG erhält. Das ganz konkrete Thema der PG war einige Zeit unsicher, aufgrund der teilweise schwierig zu organisierenden Kooperation der vielen verschiedenen Mitarbeiter am FWL, an der Uni, an Gambit u.a. war die Planung schwierig. Das Thema an sich war jedoch recht interessant, das Ergebnis direkt auf einer realen Förderanlage

zu sehen, motivierend. Insgesamt positiv waren auf jeden Fall die Gruppe und die persönlichen Kontakte innerhalb dieser und es wird sicherlich noch das ein oder andere Dachterrassengrillen geben.

Projektmitglied 2

Positiv an dieser Projektgruppe war die hohe Eigenverantwortung der gesamten Gruppe. So wurde nicht von den Betreuern vorgegeben, wer was zu tun hatte, sondern es wurde die gesamte Arbeit in der Gruppe intern aufgeteilt. Gut war weiterhin, dass sich durch die PG neue Themen praxisnah erschließen konnten und nicht, wie sonst an der Universität üblich, eher theoretisch abgehandelt wurden. Negativ war jedoch die schlechte Planung der gesamten Projektgruppe. So hatten die Seminarthemen wenig mit dem Projekt zu tun und waren größtenteils für die weitere Arbeit in der PG unwichtig und nicht brauchbar. Weiterhin wusste anfangs keiner, was das Ziel dieser PG sein sollte. Außerdem gab es eine ungleichmäßige Arbeitsaufteilung, die sich durch die gesamte PG-Zeit zog. Zusammenfassend kann man sagen, dass alle PG-Teilnehmer viel gelernt haben und mir die PG viel Spaß gemacht hat. Man hat gelernt, selbstständig zu arbeiten und ein komplexes System von Grund auf zu entwickeln. Allerdings kamen durch die ungleichmäßige Arbeitsaufteilung einige Spannungen auf.

Projektmitglied 3

Die PG hinterließ bei mir einen gemischten Eindruck. Auf der einen Seite war es spannend, ein Projekt mit Web Services zu machen, auf der anderen auch eine Herausforderung, die Teilnehmer mit ihren unterschiedlichen Qualifikationen zu organisieren. Dabei muss man leider zugeben, dass Letzteres vielleicht nicht ganz so gut glückte. Was ich ein wenig vermisst habe, war die Kontrolle und der Druck von einer Autoritätsperson, die auch die PG-Leitung hätte sein können. So war es dann auch kein Wunder, dass die Arbeit nicht von allen im gleichen Umfang geleistet wurde. Es sind jedoch auch positive Dinge zu nennen, wie z.B. der Praxisbezug, den es im Studium viel zu selten gibt. Eine Sache, die mich positiv überrascht hat, war der „Spaßfaktor“ beim gemeinsamen Programmieren der Komponenten ebenso wie das Erkennen von Problemen und Erarbeiten von Lösungen zur Fertigstellung des ROASTED KIT.

Projektmitglied 4

Als ich von der PG gelesen hatte, dass es sich um Webbasierte Technologien handelt, war ich sehr gespannt und freute mich auf die erste Sitzung. Der erste Vortrag war erfolgreichen abgeschlossen und wir lernten uns mit den anderen PG Teilnehmern kennen. Jeder bekam ein spezielles Teilgebiet und wir teilten uns in Gruppen auf, wobei ich mir nur gedacht hatte, hoffentlich werden ich nicht das ganze Jahr das gleiche machen. Auch der Gedanke das der Professor unser Betreuer war hat mich zuerst eingeschüchtert, weil ich Professoren kennen gelernt hatte die streng und immer verärgert waren. Erfreulicherweise war das in der PG erspart geblieben. Der Professor und sein Betreuer waren sehr nett und haben uns von vorneherein klar gemacht, welche Position sie einnahmen und zwar die Betreuung und die Beratung der PG. Nach meiner Meinung war dieses sehr gelungen. Es hat viel Spaß gemacht mit dem Professor und seinen Betreuern (jeweils ein Betreuer war für ein halbes Jahr da) zusammen zu arbeiten. Den Freiraum den sie uns gegeben hatten, war sehr hilfreich für das selbstständige Arbeiten. Dadurch hatte man die Möglichkeit bekommen, Lösungsansätze selbst zu entwickeln. Ein Nachteil hatte aber die Selbstständigkeit und zwar, dass man gegen die Zeit arbeitete, weil so eine Erfahrung einfach mehr Zeit in Anspruch nimmt. Da wir auch noch keine Erfahrung von einem großen Projekt hatten, hatten wir unserer Seits viele Fragen und Probleme, die wir mit unsere Betreuern und dem Professor gelöst haben. Anfangs war die PG interessant, aber nach einem halben Jahr hat sich das Projekt nur in die Länge gezogen. Wir hatte ein schwieriges und komplexes Projekt das wir zu bewältigen

versuchten. Für uns hieß es auch, die vorlesungsfreie Zeit durch zuarbeiten. Da war es auch selbstverständlich das ein paar Leute in den Urlaub gegangen sind. Es störte mich, dass es in der PG kein Zeitmanagement gab, wann gearbeitet und an welchen Tagen freigenommen werden dürfte.
Es hieß immer es müsste durchgearbeitet werden. Und wenn man sich doch frei oder Urlaub nehmen wollte, kam die Kritik sofort oder wenige Monate später. Nach meiner Meinung, sollte in der vorlesungsfreie Zeit nicht gearbeitet werden. Die Leute, die arbeiten möchten, sollten das auch auf einer freiwilligen Basis tun. Einzige Voraussetzung dafür ist aber, dass das Projekt nicht so groß gestaltet werden darf. Bei unserem Projekt hatte man sehr viel zu tun, so dass wir für freie Tage im Prinzip keine Zeit hatten. Dieses machte sich auch an der Stimmung am Schluss der PG bemerkbar. Bei einigen Leuten bemerkte man, dass die Motivation und die Interesse fertig zu werden verschwand. Das Projekt fand einfach kein Ende. Es lag auch nicht direkt bei uns, sonder auch an den Werkzeugen und Softwaretools die wir benutzten. Sie waren an manchen stellen für den Einsatz von Web Service so unstabil, das wir an machen Tagen verzweifelten. Die Teamarbeit in den Gruppen war sehr positiv. Man könnte viel von anderen Teilnehmern lernen[1]. Auch wenn das nicht immer so geklappt hat wie man sich das vorgestellt hatte. Aber ich glaube, dass jeder von jedem lernen konnte. Alle Teilnehmer waren Hilfsbereit und Engagieren sich viel, auch um kleine Disparitäten auszugleichen. Unsere Wiki-Seite hat viel dazu beigetragen, Wissen auf einfache Weise zu vermitteln. Manchmal kam es auch zu irrelevanten Diskussionen die uns manchmal Zeit und Nerven gekostet haben, aber auch aus vielen Diskussionen konnte man Positives entnehmen[2]. Ein Abschließendes Fazit: Das Projekt war sehr Interessant und ich konnte viele Erfahrungen sammeln. Mit den Teilnehmern habe ich gerne zusammengearbeitet. Es war eine angenehme Projektgruppe, wo ich auch viel Spaß hatte. Zurück blicke ich auf eine schöne, stressige und lehrhafte Zeit und ich hoffe, dass wir uns nicht so schnell aus den Augen verlieren.

Projektmitglied 5

Bei der PG475 ist zunächst mal anzumerken, dass die Anforderungen so schwammig formuliert waren, dass bei einer kleinen internen Umfrage zu Beginn der PG keine wusste, was genau entwickelt werden soll - mit Ausnahme von Tobias M., der am Lehrstuhl FLW arbeitet. Desweiteren ist mir negativ aufgefallen, dass nicht mal die Hälfte der Mitglieder der PG gut programmieren können - das hätte ich eigentlich von jedem erwartet, der im Informatik-Studium kurz vor dem Abschluss steht. Durch diesen Umstand haben sich dann leider auch sehr unterschiedliche Engagements für die Arbeit in der PG ergeben, so dass einige sehr viel gemacht haben und andere extrem wenig. Auch dass die Projektleitung gruppenintern war, fand ich nicht besonders gelungen. Der Projektleiter sollte normalerweise eine höhere Stellung einnehmen als die anderen, um auch entsprechend den anderen Aufgaben zuteilen zu können. Bei uns war es aber so, dass sich wohl einige gedacht haben "lass den mal Reden" und es dann zu keinen Konsequenzen gekommen ist. Meine Vorschläge für zukünftige PGs wären deshalb:

- Klarere Beschreibung der Ziele der PG.
- Benoteter Schein, der auch mit in die Diplomnote mit eingeht, um die Arbeitsleistung einiger zu steigern.
- Projektleiter von außerhalb der PG (z.B. Wissenschaftlicher Mitarbeiter), der dann auch überblicken kann was jeder geleistet hat und dann Noten verteilen kann.

Man muss aber auch sagen, dass mir die PG durchaus Spaß gemacht hat. Ich mag es, solche Herausforderungen gestellt zu bekommen und diese dann im Team zu lösen. Allerdings macht

[1] "Wenn man etwas nicht weiß, so kann man fragen; wenn man etwas nicht kann, so kann man es lernen." - Lü Bu We
[2] "Diskussion ist ein Austausch von Wissen. Ein Streit ein Austausch von Ignoranz." - Robert Quillen

es mehr Spaß, wenn die Aufgabe gleichmäßig verteilt werden kann und nicht ein Großteil von einigen wenigen geschultert werden muss.

Projektmitglied 6

Manchmal muss man über Brücken gehen, um Erfahrungen zu sammeln, die es nicht auf dieser Seite der Brücke zu entdecken gibt. Die erste große praktische Aufgabe seit langem im Studium in einem zusammengetrommelten Team, die erste größere Herausforderung, die die Geduld der Einzelnen gefordert und des öfteren auch strapaziert hat. Die PG war ein Platz des Lernens. Gelernt wurde nicht nur irgendeine Programmiersprache, irgendeine neue Technologie, sondern insbesondere auch der Umgang und die Zusammenarbeit in einem Team, einem Team voller Facetten, voller unterschiedlicher Menschen. Da unser Team weitesgehend selbstständig gearbeitet hat, waren wir keinem äußeren Druck unterworfen, so machten wir viele Fehler, aus denen man bekanntlich viel mehr lernt. Wir sind einige Male gefallen, aber jeder Fall hat erheblich dazu beigetragen, professioneller zu arbeiten. Manche mögen meinen, dass die Art Selbstbestimmung, die uns zugemutet wurde, uns nicht gut getan hat. Es war eine interessante Zeit.

Projektmitglied 7

An der Projektgruppe gefiel mir, dass die Projektgruppen-Mitglieder viel in Eigenverantwortung arbeiten konnten. Wir haben dadurch Teamfähigkeit und freies Arbeiten gelernt, was sicherlich ein gutes Training für das spätere Berufsleben ist, und dadurch die Möglichkeit gibt, Lösungsansätze selbst entwickeln zu können.
Innerhalb unseres sehr unterschiedlichen Teams wurden uns allerdings auch die Probleme aufgezeigt, die entstehen können, wenn so viel verschiedene Personen an einem Projekt arbeiten. Da die Seminarthemen wenig mit dem Projekt zu tun hatten und das genaue Ziel der PG und der Weg dahin nicht exakt vorgegeben waren, tauchten verschiedenste Probleme auf, die im Rahmen der PG bewältigt werden mussten. Dies war oftmals mühsam, aber wir lernten dadurch wie wahrscheinlich in der Praxis auch aus unseren eigenen Fehlern.
Da wir überwiegend selbständig arbeiten konnten, bisher aber keinerlei Erfahrung mit einem so großen Projekt hatten, und auch die Leiter des Projektes vorher noch nie ein derartiges Projekt betreut hatten, tauchten im zweiten Teil der Projektarbeit terminliche Probleme auf. Fazit: Diese zum Ende der PG auftauchende Problematik wäre durch eine bessere Zeitplanung und gleichmäßigere Arbeitsaufteilung zu lösen.
Insgesamt war es interessant mitzuerleben, wie aus anfänglichen Plänen unser Projekt Formen bis hin zur Fertigstellung angenommen hat. In der Projektgruppe habe ich die Erfahrung gemacht, dass trotz Problemen die Arbeit Spaß gemacht hat und ich auch viel von den anderen PG-Teilnehmern lernen konnte.

Projektmitglied 8

Meine Meinung zur PG ist sehr gemischt. Einerseits denke ich das ich viel lernen konnte, z.B. wie man ein komplexes Software Projekt angeht oder wie man effektiv im Team zusammen arbeiten kann. Dass wir fast alles, was wir getan haben, völlig frei entscheiden konnten, hat schon dazu beigetragen ,das man lernt, sich selbst zu organisieren. Aber andererseits fand ich die Vorbereitung der PG sehr schlecht. Der Kontakt zum Lehrstuhl FLW war extrem schwierig und hat uns genau so wie die schwammige Zielsetzung und die schlechte Praktikumsphase nur unnötig aufgehalten. Die Leute fand ich eigentlich alle sehr nett, und das Arbeitsklima war fast immer gut. Allerdings trifft das auf die letzten 2 Wochen überhaupt nicht zu, da alle sich gegenseitig nur noch Schuldzuweisungen gemacht haben.

Projektmitglied 9

Grundsätzlich fand ich die Arbeitseinstellung von so manchem Projektmitglied nicht korrekt. In keinem Projekt ist die Arbeitsaufteilung ausgeglichen, und es kann ja auch nicht jeder alles machen. Dennoch hätten sich manche etwas mehr anstrengen können und sollen, um andere zu entlasten. Eine solche "ungerechte" Arbeitsaufteilung förderte stark Zwietracht innerhalb der Projektgruppe, sorgte also für starke soziale Spannungen. An dieser Stelle sei noch mal der Vorschlag benoteter Scheine von Projektmitglied 5 angesprochen. Dadurch würde die Mehrarbeit der entsprechenden Projektmitglieder belohnt, indem die Diplomnote verbessert wird.
Obwohl die Arbeit in einer so großen Gruppe doch schwierig war, bin ich gluecklich und ziemlich stolz, eine so schwierige und umfassende Projektgruppe erfolgreich beendet zu haben. Desweiteren möchte ich hier noch erwähnen, dass es eine einmalige Chance war, durch das Ansteuern einer realen Förderanlage eine interdisziplinäre Aufgabe zu bearbeiten. Der Kooperation mit dem Lehrstuhl für Förder- und Lagerwesen bin ich sehr dankbar. Bisher gab es noch keine auf Web Service basierende Materialflusssysteme; daher freue ich mich auch einmal die angewandte Forschung erleben zu dürfen, auch wenn die Qualität nicht dir von "professionellen" Forschungsgruppen ist - aber wir sind ja auch noch Studenten.
Mit der Arbeit der Betreuer bin ich im Nachhinein sehr zufrieden. Wir waren die meiste Zeit auf uns allein gestellt, so dass wir viele Dinge selbst erlernen mussten/konnten. Dies resultierte leider oft in Diskussionen bzw. Mehrarbeit und somit in persönlichen Ärger. Zu diesen Zeitpunkten wünschte ich mir oft, dass sie unsere Grenzen aufzeigten und uns nicht wirklich alles selbst erlenen ließen. Dennoch war es gerade diese Art des Lernens - auch wenn es teilweise Mehrarbeit bedeutete - die genau die Erfahrung brachte, die man nicht so schnell vergisst und auch später noch anwenden kann. Zudem fühlten wir uns durch die von den Betreuern gebotene Freiheit nicht als irgendeine "billige Arbeitskraft", sondern wirklich als Team, das selbständig einem Ziel zuarbeitet.
Alles in allem war es eine anstrengende aber lohnenswerte Zeit und ich bin stolz, Teilnehmer der Projektgruppe 475 gewesen zu sein.

Projektmitglied 10

Im Laufe des vergangenen Jahres hatten wir die Möglichkeit, viele Dinge zu lernen, die im Laufe unseres späteren Berufslebens äußerst hilfreich sein werden. Um sicherzustellen, dass ich/wir sie nicht vergessen, habe ich versucht, sie im Folgenden festzuhalten. Die meisten Lerneffekte ergaben sich dabei aus den vielzähligen Fehlern, die wir im Laufe der Projektgruppe begangen haben. Da ich mit Sicherheit bei der wahrlosen Beschreibung dieser Lerneffekte den Überblick verloren hätte, habe ich sie thematisch geordnet und in einzelne Abschnitte aufgeteilt.

- **Organisatorisches**
  Zunächst einmal mussten wir lernen, dass ein Team von 12 gleichberechtigten Entwicklern äußerst schwer zu organisieren ist. Diesem Problem konnten wir nur durch die Festlegung einer Hierarchie begegnen. Deshalb wurden Teams gebildet und Team-Sprecher sowie Projektleiter basis-demokratisch gewählt. Weiterhin hat sich im Laufe des Jahres gezeigt, dass der Posten eines Projektleiters ein Fulltime-Job ist, dessen Ausübung rückblickend keinerlei innerprojektliche Aktivitäten erlaubt. Da uns die genaue Definition des Projektzieles selbst überlassen wurde, konnten wir außerdem lernen, wie schwierig es sein kann, den Umfang eines größeren Projekts sinnvoll und realistisch abzustecken. Nichts schlug dabei so fehl, wie die Einschätzung der eigenen Arbeitsleistung. Dies führte natürlich dazu, dass wir uns hoffnungslos übernommen haben.

- **Fachliches**
  Was unsere fachlichen Kenntnisse angeht, so konnten wir uns in vielerlei Hinsicht austoben. So wollten wir initial einen Werkzeugkasten für die Entwicklung von verteilten Systemen

entwickeln. Um eine konkrete Problemstellung zu haben, spezialisierten wir uns dabei auf die Erstellung eines Werkzeugkastens für die Entwicklung von dezentralen Steuerungen für Materialflusssysteme. Da sich dieses Themengebiet als äußerst komplex erwies, mußten wir des Öfteren den fachlichen Horizont unseres Projektes einschränken. Dabei hat mich persönlich am meisten begeistert, endlich mal die Möglichkeit zu haben, theoretische Konzepte aus der Informatik an einer praktischen Problemstellung anwenden zu können. Währenddessen hat sich vor allem gezeigt, dass man in der ein oder anderen Vorlesung den Stoff zu unrecht für zu theoretisch befunden und ausgeblendet hat.

- **Menschliches**
  Allein mit Punkten zu diesem Thema hätte ich problemlos mehrere Seiten füllen können. Deshalb versuche ich mich hier auf die wichtigsten Erkenntnisse zu beschränken. Was mich im Laufe der PG am meisten beeindruckt hat, war die Hingabe, mit der einige PG-Mitglieder sich unserem Projekt gewidmet haben. Sie haben meiner Meinung nach erkannt, was für eine unglaubliche Chance sich uns geboten hat. Schließlich erhält nicht jede PG die Möglichkeit interdisziplinär zu arbeiten und darüberhinaus durch die Kooperation mit einem Logistiklehrstuhl (FLW) die Gelegenheit eine echte Föderanlage für ihr Projekt zu nutzen. Darüber hinaus haben sich diese PG-Mitglieder besonders dadurch hervorgetan, dass sie für die anderen stets ein offenes Ohr hatten und auch in ihrer Freizeit die liegengebliebene Arbeit erledigt haben. Diesen Teamgeist und diese Kollegialität kann ich im Nachhinein nur als vorbildlich bezeichnen. Persönlich möchte ich mich dabei besonders bei Martin und Michael bedanken, ohne deren unermüdlichen Einsatz unser Projekt mit Sicherheit gescheitert wäre.

Abschließend kann ich nur sagen, dass es ein interessantes, lehrreiches und „manchmal“ auch anstrengendes Projekt war, das ich nicht mehr missen möchte. Es hat mir aufgezeigt, wie viel es noch zu lernen gibt und dabei geholfen, ein besseres Gefühl für professionelle Arbeit zu bekommen. Ich bedanke mich bei allen Beteiligten für die Zusammenarbeit und wünsche allen viel Glück für ihr weiteres Leben.

Projektmitglied 11

Hätte man für die in die Projektgruppe investierte Arbeit Geld bekommen, dann wäre ich vermutlich einer Derjenigen, die sich nun mehr als Andere leisten könnten. Ich habe das letzte Jahr als sehr stressig und arbeitsintensiv empfunden. Nichtsdestotrotz habe ich auch viel Erfahrung gesammelt, die mir für die zukünftige Arbeit in zusammengewürfelten Teams weiterhelfen wird.

Zum einen ist für eine erfolgreiche Planung eines Projekts das Wissen über die Fähigkeiten der einzelnen Teammitglieder zwingend nötig, um Ziele nicht zu hoch anzusetzen. Hier haben wir uns wirklich überschätzt, vor allem die Arbeitseinstellung einzelner Teammitglieder war der entscheidene Punkt warum wir nicht alle Ziele erreichen konnten.

Denn schon zu Beginn kristallisierten sich mehrere Teilnehmer-Typen heraus; natürlich gab es auch im Verlauf der Projektgruppe Verschiebungen in der Zuordnung der Teilnehmer zu den Typen:

- Teamplayer
  Er ist bereit Neues zu lernen, sich selbstständig in neue Themen einzuarbeiten und im Team Probleme zu lösen. Er kommentiert seinen Quelltext, so daß auch andere (und er selbst) ihn noch nachher verstehen können.

- Einzelgänger
  Er nimmt alt Bewährtes, ändert Dinge ohne Absprache, kommentiert seinen Quelltext nicht (ausreichend), welcher dadurch für Andere (und ihn) später nur schwer verständlich ist.

- Mitläufer
  Er arbeitet nur nach Auftrag und erfüllt ihn meistens nicht ausreichend. Quelltext hat er wenig geschrieben oder dieser musste von Anderen neugeschrieben werden.

Ferner sollte es wirklich einen festen Projektleiter geben, der sich ausschließlich um die Organisation und die Zielsetzung des Projekts kümmert. Das von uns angewandte Open-Source-Prinzip - jeder macht irgendwas wo er gerade Lust zu hat - führte vor allem in der Endphase zu vielen Problemen, es gab keine klare Richtung in die es gehen sollte. Die Planung war in der Endphase nicht mehr möglich, da die im ersten Semester verlorengegangene Zeit fast nicht mehr aufzuholen war und die Projektleitung ihre Mehrfachrolle (Leitung, Design, Implementierung, Social Engineering etc.) natürlich nicht mehr voll erfüllen konnte.

Vergleichbar mit einem Kutscher der den Karren, in dem seine Pferde sitzen, selbst zieht, kam mir die Projektgruppe oft vor. Die Motivation der Gruppe etwas für das Projekt zu tun war unter aller Sau! Jeder setzt seine Prioritäten selbstverständlich anders, aber was einige Teilnehmer im Verlauf des Jahres (nicht) geleistet haben, ist einfach traurig. Danken will ich auf jeden Fall Sascha und Michael für ihren teilweise übermenschlichen Einsatz, hier sieht man daß wo ein Wille ist auch ein Weg ist - nicht Jedermanns Muttersprache war C++ bzw. Java.

Als Abschluß bleibt noch zu sagen, daß ich einen benoteten Schein auch lieber gehabt hätte oder aber man hätte Prof. Krumms Vorschlag am Anfang des zweiten Semesters, also von ihm vergebene Aufgaben, ernster nehmen sollen. Wir hätten sicherlich viel mehr erreicht wenn mehr Druck dagewesen wäre, sei es eben durch eine schlechte Note oder aber Nichtvergabe des Scheins.

Projektmitglied 12

Nach langer Überlegung bin ich der Meinung, Worte sind wie Schall und Rauch. Deswegen sollte sich jeder seine eigene Meinung bilden, egal ob sie nun am Ende positiv oder negativ ist.

A Aufbau der Schaltschränke

In diesem Kapitel werden die Schaltschränke beschrieben, sowie die darin verwendeten Klemmen.

A.1 IPC1

Die Schaltschränke, die an IPC1 hängen, sind US1, ES1 und US2. Diese, sowie UC1, der Schaltschrank von IPC1 selbst, werden nun detailliert beschrieben.

A.1.1 UC1

Datenblatt Betriebsspannung: 24 V DC
MAX. Anschlussleistung: 7.5 VA
Steuerspannung: 24 V DC
Es gibt eine Klemme mit 9 Anschlüssen (1-8 und PE), die wie folgt benutzt werden: Von Außen kommen Leitungen 1 bis 6 von ES1. 1 ist Dauerspannung 24V und geht parallel durch 2 Sicherungen, um dann zum einen den IPC selbst mit Strom zu versorgen, aber auch um zu US1 weitergeleitet zu werden. Anschluss 5 dient zur Kontrolle, ob die Steuerspannung eingeschaltet ist und geht in Eingang E1 des ersten KBus Moduls vom Typ 750-410, also einer digitalen Eingangsklemme mit 2 Kanälen. Alle anderen Klemmenbelegungen werden nur weitergeleitet. Das zweite K-Bus Modul, welches vom Typ 750-627 ist, dient zur Weiterleitung der Signale an die weiteren Schränke. Dazu hat das Modul einen RJ-45 Anschluss. UC2 und UC3 sind analog zu UC1 aufgebaut. Der IPC mit angeschlossenen Klemmen ist in Abb. A.1 dargestellt.

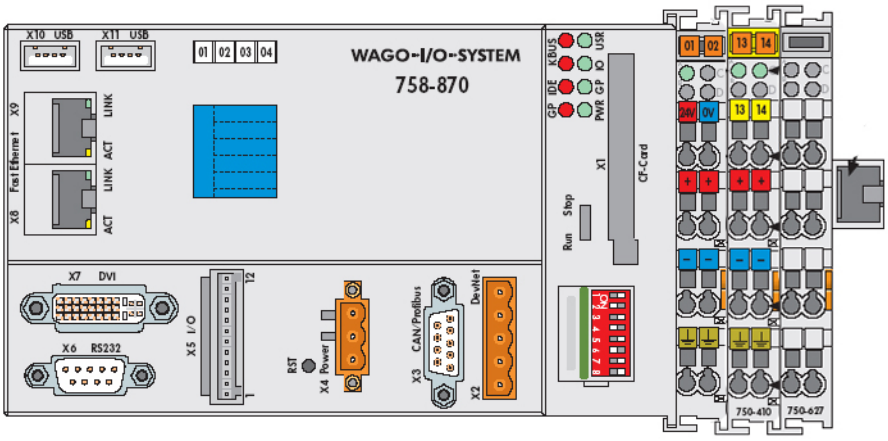

Abbildung A.1: Schematische Darstellung des IPC1

A.1.2 US1

Datenblatt Betriebsspannung: 400V
Frequenz: 50Hz
MAX. Anschlussleistung: 7.5kVA
Steuerspannung: 24VDC

Dieser Schaltschrank hat 2 verschiedene Spannungsversorgungen. Zum einen gibt es eine 400V Klemme, die den Schrank mit Betriebsspannung versorgt, zum anderen eine 24V Klemme, um den Schrank mit Steuerspannung zu versorgen. Die 400V werden benötigt, um die Motoren mit Betriebsspannung zu versorgen. Von den 3 Phasen wird dazu pro Motor jede Phase durch einen Motorschutzschalter und ein Schütz der Firma Siemens geführt, das von Wago I/O Modulen gesteuert werden. Als erstes wird die Spannungsversorgung für den Motor des Rollenförderers U1 durch den Motorschutzschalter vom Typ 3VU13, dann durch das Schütz vom Typ 3TF20 geleitet. Wenn nun das Schütz anzieht, dann wird der Motor mit Strom versorgt. Analog zu dem Motor von U1 wird der Motor von U6 und der Motor von U7 versorgt. Die 24V Eingangsklemme ist wie folgt belegt:

1 Dauerspannung (wird nur durchgeschleift, in diesem Schaltschrank aber nicht genutzt)

2 Not-Aus Spannung

3 Dauerspannung intern

4 Dauerspannung extern

5 Signal Steuerung Ein (wird ebenfalls hier nicht genutzt, jedoch durchgeschleift)

6 Bezugspotential M

7 PE

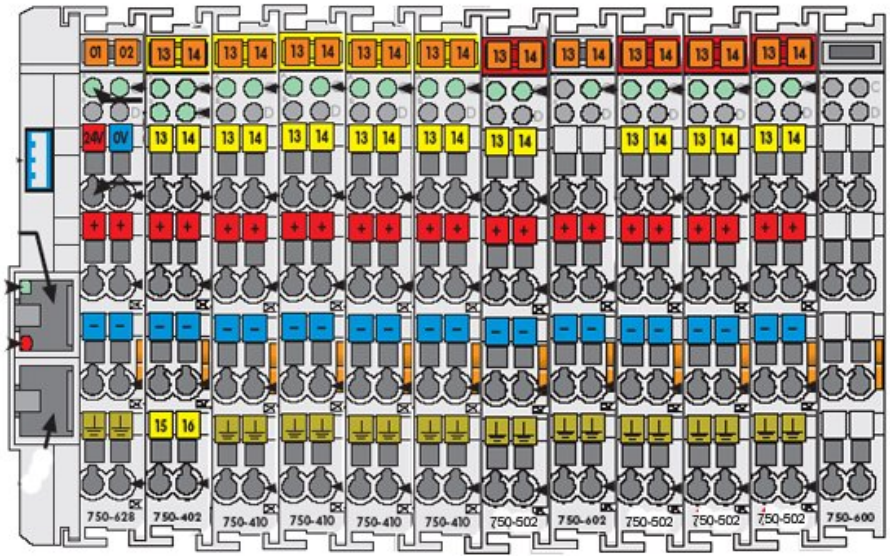

Abbildung A.2: Schematische Darstellung des Schaltschranks US1

Leitung 3 und 4 versorgen das erste K-Bus Modul vom Typ 750-628. Die beiden RJ-45 Anschlüsse an dem Koppler dienen zur Kommunikation mit dem WAGO I/O Modul im Schrank UC1 (IN) und ES1 (OUT). Die an dieses Modul angeschlossene digitale 4 Kanal Eingangsklemme vom Typ 750-402 kontrolliert, ob die 400V Spannungsversorgung an dem Motor von U1(E1), U6(E2) und U3(E3) betriebsbereit ist. Der vierte Eingang ist zur Reserve.
Die nächsten 4 Klemmen sind vom Typ 750-410 und daher digitale 2 Kanal Eingangsklemmen. Diese versorgen die Sensoren (Lichtschranken und Initiatoren) mit Strom und bekommen die Ergebnisse der Prüfungen zurück (siehe Abb. A.4). Sie sind in folgender Reihenfolge belegt:

1. Modul: Die Lichtschranke an U1 am Wareneingang (E2) sowie am Ende von U1 (E1) werden von hier überprüft.

2. Modul: Die Ergebnisse vom Initiator des Rollenteppichs oben (E1) und unten (E2) werden hier empfangen.

3. Modul: Von hieraus werden die Lichtschranke an U6 am Anfang (E1) und der Initiator an U6 am Ende (E2) überprüft.

4. Modul: Hier ist die Lichtschranke an U7 (E1) und die an U1 vor dem Abzweig (E2) angeschlossen.

Das nächste Modul ist eine digitale 2 Kanal Ausgangsklemme vom Typ 750-502 und dient zur Steuerung der Sammelstörungsleuchte (A1) und bietet eine Reserve (A2). Nun folgt eine Klemme vom Typ 750-602 zur Potentialeinspeisung, die von Klemme 2 der 24V Eingangsklemme gespeist wird.. Bevor die Endklemme vom Typ 750-600 die WAGO I/O Modulreihe abschließt, befinden sich noch 3 digitale 2 Kanal Ausgangsklemmen in der Reihe. Sie dienen zum Einschalten der Motoren (siehe Abb. A.3), oder genauer dazu, das Schütz zu schalten. Sie sind wie folgt aufgeteilt:

1. Modul: Hier können U1 (A1) und U6 (A2) eingeschaltet werden.

2. Modul: Hier kann U7 eingeschaltet werden (A1). A2 dient zur Reserve.

3. Modul: Hier kann der Rollenteppich (Hubstuhl) angehoben werden (A1). A2 ist Reserve. Die Abb. A.2 zeigt den Aufbau der gerade beschriebenen Klemmen.

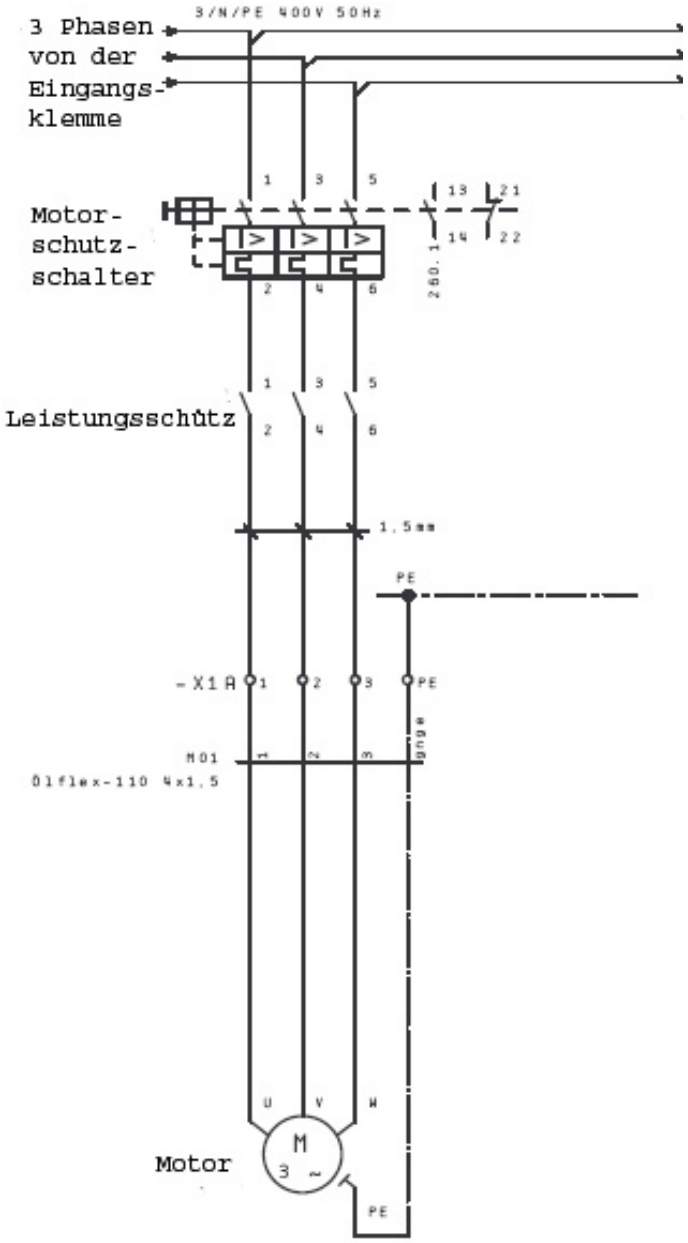

Abbildung A.3: Schaltskizze einer Motoransteuerung durch eine digitale Ausgangsklemme

A.1.3 Schaltschrank ES1

Datenblatt Betriebsspannung: 400V
Frequenz: 50Hz
MAX. Anschlussleistung: 24kVA
Steuerspannung: 24VDC
Es gibt eine Eingangsklemme, die den Schrank mit 3-Phasen Wechselstrom versorgt. Weiterhin gibt es eine 400V DC Ausgangsklemme mit 21 Ausgängen (1, 2, 3, N, PE, 4, 5, 6, N, PE, 7, 8, 9, N, PE, 10, 11, 12, N, PE und 13), eine Klemme für die 24 V Spannungsversorgung (1, 2, 3, 4, 5, 6, PE, 7, 8, 9, 10, 11, 12, PE, 13), eine für das Not-Aus (1, 2, 3, 4, 5, 6, 7, 8, 9, 10) und eine zur Spannungsversorgung mit 230V (1, 2). Der 3-Phasen Wechselstrom wird nach der

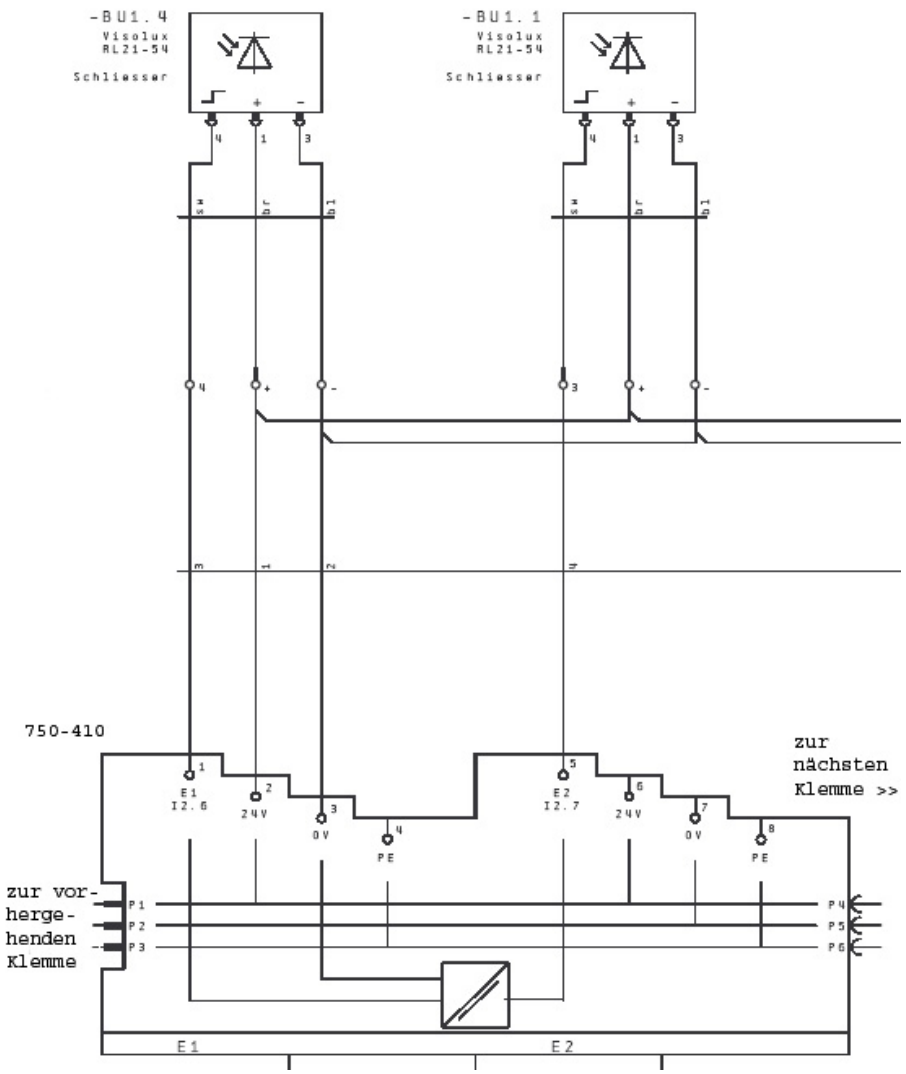

Abbildung A.4: Schaltskizze zum Auslesen der Zustands einer Lichtschranke oder eines Initiators durch eine digitale Eingangsklemme

Einspeisung über einen Leistungsschalter mit Drehgriff geregelt. Hierüber wird die komplette Stromversorgung der Anlage geregelt. Um zu kontrollieren, ob alle 3 Phasen vorhanden sind, gibt es an der Tür des Schaltschranks 3 grüne Kontrollleuchten. Von dieser Leitung werden nun mehrere Leitungen parallel geschaltet:

1. Über einen 63 A Leistungsschalter wird die Spannungsversorgung für die obere Ebene geschaltet.

2. Über einen 35 A Leistungsschalter wird die Spannungsversorgung für die untere Ebene geschaltet.

3. Weiterhin gibt es noch eine Reserve, die mit einem 15 A Leistungsschalter gesichert ist und eine die mit einem 10 A Leistungsschalter gesichert ist.

4. Von der ersten Phase wird die Schaltschrankbeleuchtung und die Anfahrhupe betrieben. Es ist ein Sicherungsschalter dazwischengeschaltet.

5. Von der 3. Phase werden 3 Steckdosen betrieben. die ersten 2 werden über eine 4 A Sicherung gesichert, die dritte einzeln über eine 16 A Sicherung.

6. Zum Schluss werden alle Phasen in einen durch einen 3 A Leistungsschalter gesicherten Spannungswandler geführt, der aus den 400 V Drehstrom 24 V Gleichstrom macht.

Die 24 V Leitung, die aus dem Spannungswandler kommt, wird nun wie folgt aufgeteilt:

1. Über eine 2 A Sicherung ist die Spannungsversorgung intern gesichert,

2. Über eine weitere die Spannungsversorgung E/A.

3. Dann geht die Spannungsversorgung für die Not-Aus Schaltung, die über einen Leistungsschalter gesichert ist, von hier ab, sowie

4. Über jeweils 2 Sicherungsschalter der Firma Moeller vom Typ PKZMO -10 wird die Spannungsversorgung für den 24V Bus der oberen Ebene und

5. der unteren Ebene geschaltet.

6. Eine letzte, über einen Sicherungsschalter wie in 4 und 5 gesicherte Leitung dient als Reserve.

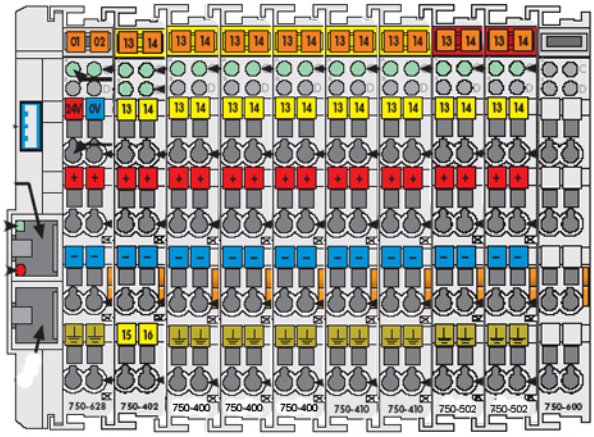

Abbildung A.5: Schematische Darstellung des Versorgungsschranks ES1

Eine grüne Kontrollleuchte, die von der Spannungsversorgung intern abgeht, zeigt an, ob die Spannungsversorgung intern vorhanden ist. Die Spannungsversorgung intern sowie die Spannungsversorgung E/A dienen zur Spannungsversorgung des ersten K-Bus Moduls im Schaltschrank vom Typ 750-628. Diese Kopplerklemme hat 2 belegte RJ-45 Anschlüsse, um vom Schrank US1 Signale zu empfangen oder zu US2 weiterzuleiten. Rechts neben dieser Klemme befindet sich eine digitale 4 Kanal-Eingangsklemme vom Typ 750-402. Die 4 Eingänge sind wie folgt belegt:

E1: Überprüfung der 400V Spannungsversorgung des Leistungsbusses der oberen Ebene

E2: Überprüfung der 400V Spannungsversorgung des Leistungsbusses der unteren Ebene

E3+E4: Überprüfung der 400V Reservespannungsversorgung

Die nächste Klemme ist eine digitale 2 Kanal Eingangsklemme vom Typ 750-400. E1 kontrolliert die Spannungsversorgung der Not Aus Schaltung und E2 die Reserve. Die nächsten beiden Klemmen sind vom selben Typ und dient zum Kontrollieren, ob die 24 V Spannung für die Not-Aus Schaltung an der oberen Ebene (E1) und die 24V Dauerspannung für die untere Ebene anliegen (E2), sowie die 24V für die Not-Aus Spannung der unteren Ebene (E1) und die Reserve vorhanden ist. Als nächstes dient eine digitale 2 Kanal Eingangsklemme vom Typ 750-410 zum Kontrollieren, ob das Not-Aus gedrückt wurde (E1) und ob der Not-Aus Schlagtaster am Schaltschrank selbst gedrückt wurde (E2). Eine weitere vom selben Typ kontrolliert, ob das Not-Aus am Schalter des Schranks selbst bestätigt wurde (E1). E2 ist Reserve. Vor der Endklemme vom Typ 750-600 sind 2 digitale Ausgangsklemmen vom Typ 750-502. Die erste steuert die Sammelstörungsleuchte und legt 24 V auf die Störungsleitung. Die zweite steuert die Not-Aus Quittierung sowie die Hupe, die als Anfahrwarnung dient. Der Aufbau der K-Bus Klemmen ist in Abb. A.5 dargestellt. Die Spannungsversorgung für die Not-Aus Schaltung durchläuft zuerst eine Schützsicherheitskombination vom Typ Siemens 3TK2804-0BB4. Von dort aus sind die 9 Not-Aus Schalter in Reihe geschaltet.

A.1.4 US2

Datenblatt

Betriebsspannung: 400V
Frequenz: 50Hz
MAX. Anschlussleistung: 7.5kVA
Steuerspannung: 24VDC
In diesem Schaltschrank (siehe Abb. A.6) gibt es eine 400V Einspeisungsklemme (1, 2, 3, M, PE), ein 24V Eingangsklemme (1, 2, 3, 4, 5, 6, PE) sowie eine 400V Klemme für die Abgänge (1, 2, 3, PE, 4, 5, 6, PE, 7, 8, 9, PE, 10, 11, 12, PE). Die 400V Einspeisungsklemme wird zuerst weitergeleitet zu den anderen Schaltschränken, dient jedoch auch zur Spannungsversorgung der Motoren. Analog zu US1 werden die Motoren von den Förderelementen U2, U3 und U4 durch einen Motorschutzschalter vom Typ 3VU13 gesichert und von einem Schütz vom Typ 3TF20 geschaltet. Die 24V Eingangsklemme ist genauso belegt wie in US1. 3 und 4 versorgen wieder das erste WAGO I/O Modul vom Typ 750-627 mit Spannung. Angesteuert wird dieses Modul von dem RJ45 Eingang, der an ES1 angeschlossen ist. Der RJ45 Ausgang ist nicht belegt, da dieser der letzte von UC1 gesteuerte Schaltschrank ist. Nach diesem Modul folgt eine digitale 4 Kanal Eingangsklemme, um zu überprüfen, ob die 400V Versorgungsspannung an den Motoren von U2(E1), U3(E2)und U4(E3) vorhanden ist. E4 ist ein Reserveeingang. Als nächstes folgen 2 digitale 2 Kanal Eingangsklemmen vom Typ 750-410. Sie dienen zur Überprüfung der Initiatoren auf U2 (1. Modul, E1), auf U4 am Anfang (1. Modul, E2) und auf U4 vor der Sperre(2. Modul, E1). Am 2. Modul dient E2 als Lichtschrankeneingang der Lichtschranke an U2. Die folgende, digitale 2 Kanal Ausgangsklemme vom Typ 750-502 steuert die Sammelstörungsleuchte (A1) und hat A2 als Reserveausgang. Die nächste Klemme vom Typ 750-602 dient der Potentialeinspeisung und wird gespeist mit der Not-Aus Spannung. Daran gekoppelt sind 3 digitale 2 Kanal Ausgangsklemmen vom Typ 750-502. Analog zu US1 werden die Motoren wie folgt angeschlossen:

1. Modul: Hier wird der Motor von U2(A1) und U3(A2) eingeschaltet

2. Modul: Der Motor von U4 wird eingeschaltet (A1). A2 ist ein Reserveausgang.

3. Modul: Das Ventil zum Aktivieren der Rollen am Touchsensor von U2 wird deaktiviert (A1) und das Ventil für die Sperre an U4 wird von hier gesteuert.

Zuletzt ist eine Endklemme vom Typ 750-600 angebaut.

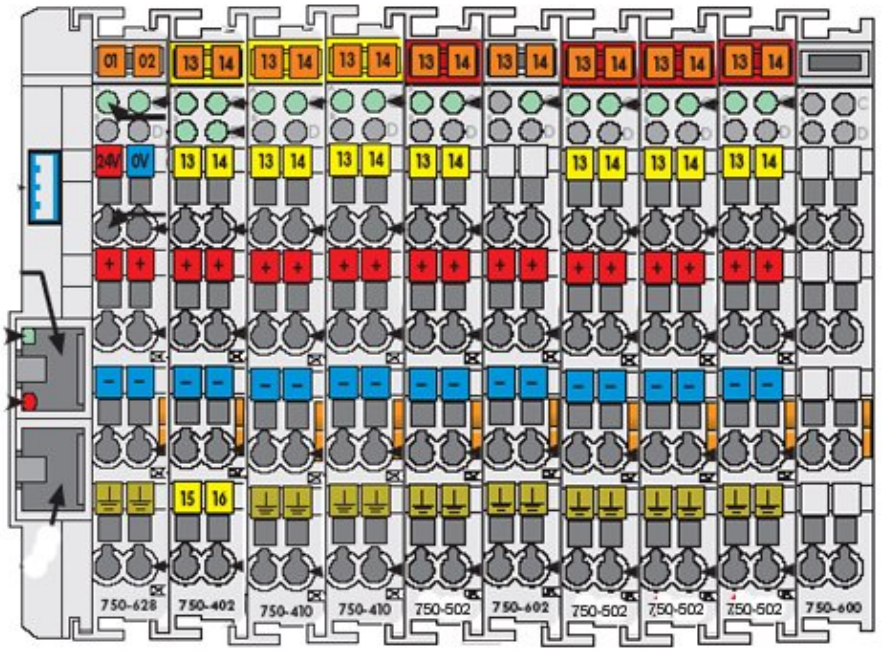

Abbildung A.6: Schematische Darstellung des Schaltschranks US2

A.2 IPC2

Die Schaltschränke, die an IPC2 hängen, sind US3 und US4. Diese, sowie UC2, der Schaltschrank von IPC2 selbst, werden nun detailliert beschrieben

A.2.1 UC2

Dieser Schrank ist genauso aufgebaut wie IPC1(UC1).

A.2.2 US3

Datenblatt Betriebsspannung: 400V
Frequenz: 50Hz
MAX. Anschlussleistung: 7.5kVA
Steuerspannung: 24VDC
Es gibt eine 400V Einspeiseklemme (1, 2, 3, M, PE), eine 400V Ausgangsklemme (1, 2, 3, PE, 4, 5, 6, PE, 7, 8, 9, PE, 10, 11, 12, PE) und eine 24V Spannungsversorgungsklemme (1, 2, 3, 4, 5, 6, PE). Die 400V kommen von US2 und werden weitergeleitet zum nächsten Schaltschrank. Sie werden benötigt um die Motoren zu steuern. Von diesem Schaltschrank werden 2 Motoren geschaltet, die sich beide an dem Drehteller U5 befinden. Der erste, der analog zu den bisherigen Motoren angesteuert wird, treibt den Abweiser an U5 an. Der andere dient für den Antrieb des Drehtellers selbst. Dazu werden die 3 Phasen für den Drehstrom nach dem Motorschutzschalter und dem Leistungsschütz in einen Phasenwandler vom Typ Siemens Micromaster 420 geführt, der von WAGO I/O Modulen gesteuert wird. So kann die Geschwindigkeit des Drehtellers verändert werden. Der Micromaster hat dazu verschiedene Ein- und Ausgänge. Ein digitaler Eingang dient zum Starten des Drehtellers. 2 analoge Eingänge dienen zum Angeben der Geschwindigkeit. Dazu wird eine Spannung zwischen 0 und 10V von einem WAGO I/O Modul aus dem Schaltschrank (s.u.) eingespeist. Zuletzt werden noch 2 Digitale Ausgänge zum WAGO I/O geführt.
Neben der Motorversorgung gibt es noch WAGO I/O Module in diesem Schaltschrank. Das erste

Modul ist vom Typ 750-628 und daher eine Kopplerklemme. Die 2 RJ-45 Anschlüsse werden genutzt um von UC2 Signale zu bekommen und nach US4 zu senden. Die Anschlüsse 3 und 4 aus der 24 V Eingangsklemme (Spannungsversorgung intern und extern) werden genutzt, um die Module mit Spannung zu versorgen und ebenfalls in das erste Modul geführt. Das nächste Modul ist eine digitale 4 Kanal Eingangsklemme vom Typ 750-402 und prüft, ob die 400V Spannungsversorgung am Abweiser des Drehtellers betriebsbereit ist(E1), ob eine Störung im Micromaster vorliegt (E2), ob die 400V Spanungsversorgung am Drehteller (oder genauer am Motorschutzschalter des Drehtellers) anliegt (E3) und bietet eine Reserve (E4). Die Schaltung des Drehtellers wird in Abb. A.7 dargestellt. Die nächste Klemme ist eine digitale 2 Kanal Ausgangsklemme, steuert die Sammelstörungsleuchte (A1) und kann den Drehsteller starten lassen (A2). Die nächste Klemme vom Typ 750-602 dient zur Potentialeinspeisung. Dazu wird die Notausspannung in die Klemme geführt. Als nächstes ist eine digitale 2 Kanal Ausgangsklemme, die den Abweiser am Drehteller einschaltet (A1) und das Leistungsschütz am Drehteller schaltet. Die letzte Klemme vor der Endklemme vom Typ 750-600 ist eine analoge 2 Kanal Ausgangsklemme vom Typ 750-550. Sie legt eine Spannung an den Micromaster an, der damit die Geschwindigkeit des Drehtellers regelt.

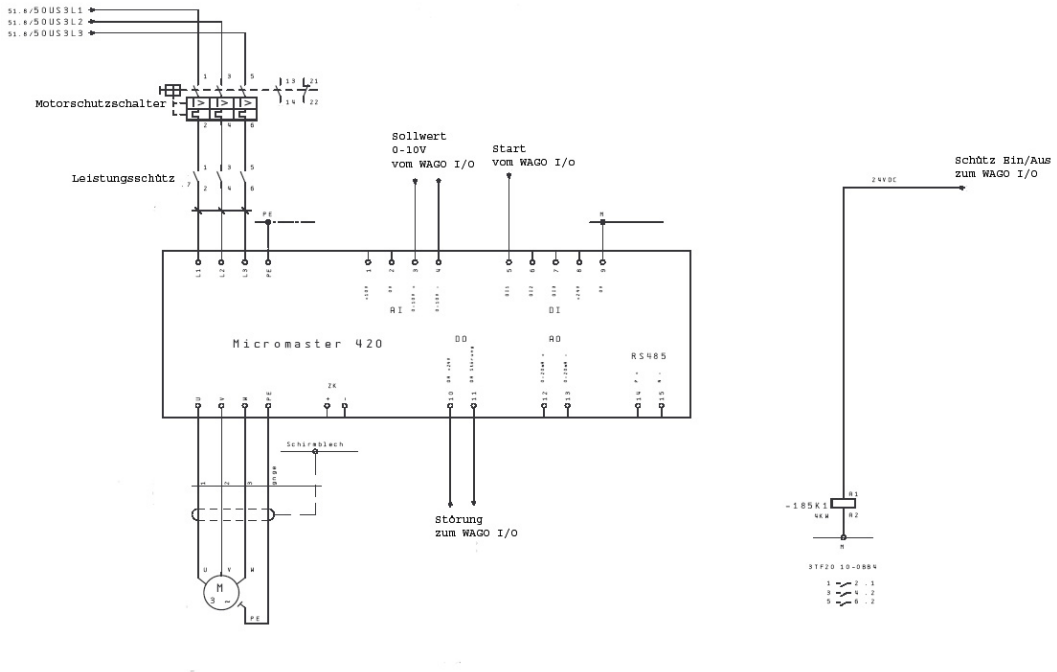

Abbildung A.7: Schaltskizze des Drehtellers (Micromaster)

A.2.3 US4

Datenblatt Betriebsspannung: 400V
Frequenz: 50Hz
MAX. Anschlussleistung: 7.5kVA
Steuerspannung: 24VDC
Auch dieser Schrank besitzt eine 400V Einspeiseklemme (1, 2, 3, N, PE), eine 400V Ausgangsklemme (1, 2, 3, PE, 4, 5, 6, PE, 7, 8, 9, PE, 10, 11, 12, PE) und eine Klemme für die 24V Spannungsversorgung. Die eingespeisten 400V werden zum nächsten Schaltschrank weitergeleitet, aber auch benutzt, um die folgenden Motoren auf die bereits bekannte Art und Weise mit Spannung zu versorgen: Rollenförderer U8, Abweiser U8, Rollenförderer U9 und der Gurtförderer

U14 nach oben (für uns nicht relevant), aber auch zum nächsten Schaltschrank weitergeleitet. Die 24V Eingangsklemme wird teilweise genutzt (Not-Aus Spannung, Dauerspannung intern und extern), ansonsten aber weitergeleitet. Die nun folgende Beschreibung der K-Bus Module wird

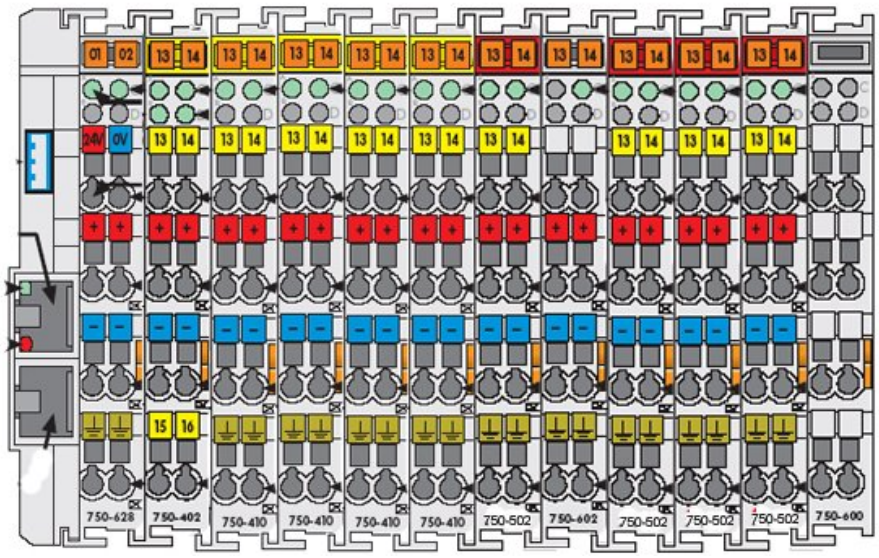

Abbildung A.8: Schematische Darstellung des Schaltschranks US4

in Abb. A.8 dargestellt. Dauerspannung intern und extern werden in das erste K-Bus Modul vom Typ 750-628 gespeist. Der RJ-45-In Anschluss ist belegt und versorgt diese Modulkette mit Informationen vom Schrank US3 und somit vom IPC aus dem Schrank UC2. Das zweite WAGO I/O Modul ist eine digitale 4 Kanal Eingangsklemme vom Typ 750-402. Sie kann überprüfen, ob die 400V Spannungsversorgung an U8 (E1), die 400V Spannungsversorgung am Abweiser von U8 (E2), die Spannungsversorgung am Rollenförderer U9 (E3) und die Spannungsversorgung an U14 (E4) anliegt. Als nächstes folgen 4 digitale 2 Kanal Eingangsklemmen vom Typ 750-410. Sie überwachen folgende Sensoren:

- 1. Modul E1: Lichtschranke an U14 am Ende
- 1. Modul E2: Lichtschranke zur Überwachung der Schwenkerabbereichs
- 2. Modul E1: Initiator Abweiser eingeschwenkt
- 2. Modul E2: Initiator Abweiser ausgeschwenkt
- 3. Modul E1: Lichtschranke am Anfang von U8
- 3. Modul E2: Initiator am Ende von U8
- 4. Modul E1: Lichtschranke nach Abzweig an U8
- 4. Modul E2: Reserve

Als nächstes folgt eine digitale 2 Kanal Ausgangsklemme vom Typ 750-502, die eigentlich nur dazu dient, eine Sammelstörung per Lichtsignal zu verkünden (E1). Die nächste Klemme ist eine Potentialeinspeisungsklemme vom Typ 750-602, die von der Not-Aus Spannung gespeist wird. Vor der Endklemme vom Typ 750-600 befinden sich 3 digitale 2 Kanal Ausgangsklemme, deren Ausgänge wie folgt belegt sind:

- 1. Klemme A1: Rollenförderer U8 ein

- 1. Klemme A2: Abweiser an U8 ein
- 2. Klemme A1: Rollenförderer U9 ein
- 2. Klemme A2: Auffahrt U14 ein
- 3. Klemme A1: Ventil für das Ausschwenken des Abweisers
- 3. Klemme A2: Ventil für das Einschwenken des Abweisers

A.3 IPC3

An IPC3 hängt nur US5.

A.3.1 UC3

Der Schaltschrank IPC3(UC3) ist genauso aufgebaut, wie IPC1(UC1).

A.3.2 US5

Datenblatt Betriebsspannung: 400V
Frequenz: 50Hz
MAX. Anschlussleistung: 7.5kVA
Steuerspannung: 24VDC
Der letzte Schaltschrank hat 3 verschiedene Klemmen. Eine 400V Eingangsklemme (1, 2, 3, N, PE), eine 400V Ausgangsklemme (1, 2, 3, PE, 4, 5, 6, PE, 7, 8, 9, PE, 10, 11, 12, PE) und eine 24V Klemme (1, 2, 3, 4, 5, 6, PE).
Die Spannung aus der 400V Eingangsklemme wird verwendet, um die Motoren des Gurtförderers U10, der Rollenbahnkurve U11, des Gliederbahnförderers U12 und der Rollenbahnkurve U13 zu versorgen. Dazu wird immer ein Motorschutzschalter vom Typ 3TF20 vor ein Leistungsschütz vom Typ 3VU13 der Firma Siemens geschaltet (zur genaueren Betrachtung s. US1).
Die 24V Klemme hat in der Schaltung in diesem Schrank die letzte Station erreicht. Lediglich die Dauerspannung extern und das Bezugspotential M werden zum RFID Steuermodul geleitet. Verwendet werden die Leitungen 2,3,4,6 und PE. Leitung 3 und 4 wird benutzt, um das erste WAGO I/O Modul vom Typ 750-628 mit Spannung zu versorgen. Die Modulkette wird gesteuert von UC3 und daher bekommt das erste Modul die Signale über den RJ-45-Eingang. Darauf folgt eine digitale 4 Kanal Eingangsklemme vom Typ 750-402, die die Leistungsschütze für die Motoren am Gurtbahnförderer U10 (E1), an der Rollenbahnkurve U11, am Gliederbahnförderer U12 und an der Rollenbahnkurve U13 kontrolliert, ob sie auch geschaltet haben. Als nächstes folgt eine digitale 2 Kanal Eingangsklemme vom Typ 750-410, die die Lichtschranke an U13 überprüft. Die nächste Klemme, ein digitale 2 Kanal Ausgangsklemme vom Typ 750-502, steuert eine Störungssammelleuchte (A1) und hat eine Reserve (A2). Nach der Potentialeinspeisungsklemme vom Typ 750-602, die mit der 2. Leitung aus der 24V Klemme gespeist wird, folgen 2 digitale 2-Kanal-Ausgangsklemmen, die die Leistungsschutze der Motoren in folgender Belegung schalten sollen:

- 1. Klemme A1: Gurtbahnförderer U10 ein
- 1. Klemme A2: Rollenbahnkurve U11 ein
- 2. Klemme A1: Gliederbahnförderer U12 ein
- 2. Klemme A2: Rollenbahnkurve U13 ein

Zuletzt befindet sich eine Endklemme vom Typ 750-600 in der Modulkette.

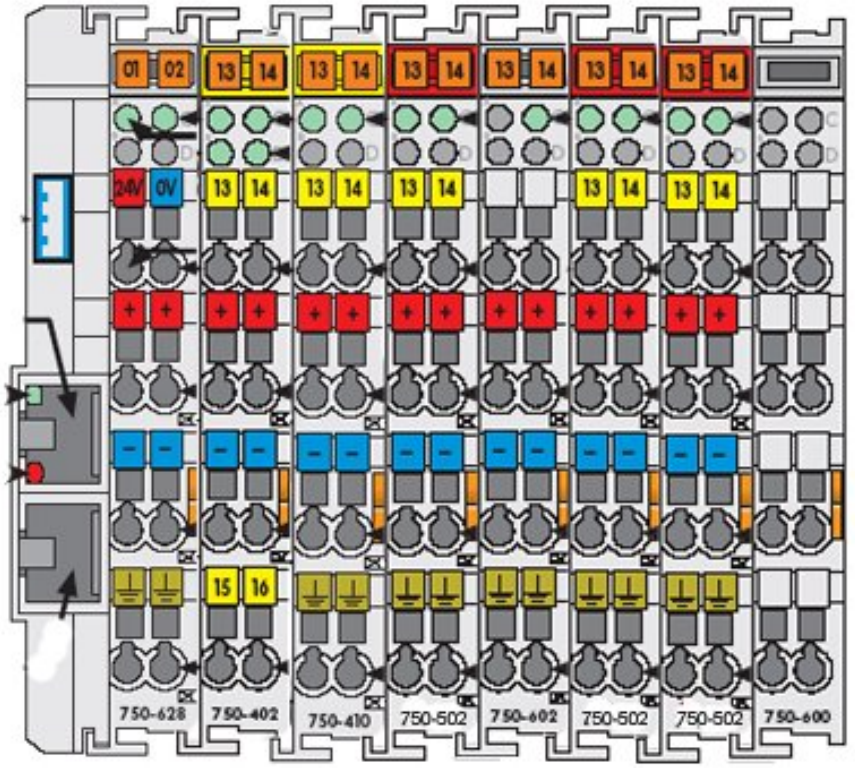

Abbildung A.9: Schematische Darstellung des Schaltschranks US5

B Schnittstellenbeschreibung

B.1 UPnPControl

Wie bereits erwähnt, bietet UPnPControl nur lokal Schnittstellen an. Das Bereitstellen von Web Services entfällt also, da die im Folgenden aufgelisteten Methoden direkt von der UPnPControl implementierenden Komponente aufgerufen werden können.

Methode: start Der Control-Point und das Device werden gestartet. Von diesem Moment an wird die eigene Komponente im Netzwerk bekannt gemacht.

Methode: getComponents Die Methode *getComponents(String componentType)* gibt ein Array von ComponentAddress-Objekten für jede im Netzwerk vorhandene Komponente zurück.

Methode: getBaseURL Die Methode *getBaseURL(ComponentAddress componentAddress)* gibt die URL des Web Services einer durch die ComponentAddress spezifizierten Komponente zurück.

Methode: startNewSearch Leitet eine erneute Suche nach allen im Netzwerk vorhandenen Komponenten ein.

Methode: terminate Stoppt Device und Control-Point und schickt ein *byebye* an die anderen Komponenten(UPnP Abmeldung). Nun ist die Komponente nicht mehr aufzufinden.

B.2 EventService

Der EventService wird von der EventControl (siehe Kapitel 9.2) angeboten und bietet insgesamt drei Methoden:

- subscribe
- unsubscribe
- notify
- fire

Methode: subscribe

Wird an die EventControl durchgereicht und bearbeitet von der EventControl einer anderen Komponenten kommende Registrierungen.

Methodensignatur(en):

subscribe(***String*[] topics, *String*[] excludes, *ComponentAddress*** **callback):*int***

Parameter:

topics (*String*[]): Vector/Array von Topics, die die registrierende Komponente zugesendet bekommen möchte.

excludes (*String*[]): Vector/Array von Topics, die die registrierende Komponente NICHT zugesendet bekommen möchte.

callback (*ComponentAddress*): Die ComponentAddress der Komponente an die die Events geschickt werden sollen.

Rückgabe:

int die SubcriptionId, welche man benötigt um sich wieder abzumelden.

Methode: unsubscribe

Wird an die EventControl durchgereicht und bearbeitet von der EventControl einer anderen Komponenten kommende Deregistrierungen.

Methodensignatur(en):

unsubscribe(*int* subscriptionId):*boolean*

Parameter:

subscriptionId (*int*): ID der Subscription, die aufgehoben werden soll.

Rückgabe:

true falls die Deregistrierung erfolgreich war.

false falls die Deregistrierung fehlgeschlagen ist.

Methode: notify

Wird an die EventControl durchgereicht um Events die von anderen Komponenten kommen zu verarbeiten.

Methodensignatur(en):

notify(*Event* event):*boolean*

Parameter: -

event (Event): Event, das weitergeleitet werden soll.

Rückgabe:

true falls das Event erfolgreich angenommen wurde.

false falls das Event nicht erfolgreich angenommen wurde.

B.3 StatusService

Die StatusControl bietet nur zwei verschiedene Methoden per Web Service an:
getStatus(): liefert den aktuellen Zustand einer Komponente zurück
setStatus(): verändert den Status einer Komponente

Methode: getStatus

Die Methode liefert den aktuellen Status einer Komponente.

Methodensignatur(en):

getStatus(*void*): *int*

Parameter: -

Rückgabe:

int liefert den Status-Wert zurück

Methode: setStatus

Die Methode setzt den aktuellen Status einer Komponente.

Methodensignatur(en):

setStatus(*int* status): *boolean*

Parameter: -

status (*int*): der zu setzende Status

Rückgabe:

true falls Status gesetzt wurde

false falls Status nicht gesetzt werden konnte

B.4 TypeLibraryService

Der TypeLibraryService bietet 4 Methoden an, die vom SystemManagement, der Interpolation sowie dem Monitoring aufgerufen werden:

getDescription(*String*[] keys):*TLEntry* []: Liefert eine textuelle Beschreibung zurück, was ein Schlüsselwert bedeutet bzw. für was es gebraucht wird.

getCategory(*int* typeId, *String* category): *TLEntry* []: liefert alle TLEntry Elemente zurück, die zu der gesuchten Kategorie gehören. Dazu sucht die Methode intern erst die passenden Schlüssel heraus und ruft dann getValues auf.

getValues(int typeId, *String*[] keys): *TLEntry* []: gibt alle Einträge zu einem Typ zurück, der sich in dem "keys" Array befindet.

getAllValues(*int* type_id): *TLEntry* []: liefert alle Einträge zu einem Typ zurück.

B.5 ServiceControl

Da diese Schnittstellen genutzt wird um Dienste bereitzustellen, kann vorher logischerweise kein Dienst angeboten werden. Daher beschreibt die im Folgenden aufgelistete Schnittstelle nur die lokal der ServiceControl-implementierenden Komponente angebotenen Methoden.

Methode: deployService

Methodensignatur(en):

deployService(*String* servicename, *Object* instance): *boolean*

Parameter:

servicename (*String*): Teil des Namens der *WSDD (Web Service Deployment Descriptor)* Datei des Services, der deployed werden soll. Alle WSDD Dateien sollten die Form "<servicename>_deploy.wsdd" haben.

instance (*Object*): Instanz der Klasse, die deployt werden soll, ist also ein Objekt der entsprechenden *ServiceName-BindingImpl* des Services.

Rückgabe:

true falls Service deployed werden konnte

false falls Fehler aufgetreten

Methode: undeployService

Methodensignatur(en):

undeployService(*String* servicename): *boolean*

Parameter:

servicename (*String*): Namen des Services der undeployed werden soll

Rückgabe:

true falls Service undeployed werden konnte

false falls Fehler aufgetreten

Desweiteren existieren die Methoden *startServer* und *stopServer* zur Kontrolle des Webservers Jetty. Über die Methode *configureJetty* kann dieser so eingestellt werden, dass Axis entsprechend angepasst wird und z.B. der Axis SoapMonitor gestartet wird. Dies läuft alles für den Anwender unbemerkt ab.

B.6 CommonLogService

Methode: log

Um eine Nachricht zu speichern wird die Methode log mit einem Array von Log Objekten aufgerufen, die dann in der Datenbank gespeichert werden.

Methodensignatur(en):

log(*Log*[] msg): *boolean*

Parameter:

msg (*Log*[]): Array von Log Objekten

Rückgabe:

true: falls erfolgreich geloggt wurde.

false: falls beim loggen Fehler aufgetreten sind.

Methode: getLog

Zum Abrufen der Log-Nachrichten kann das Monitoring getLog benutzen. Der Zeitpunkt und das Loglevel kann durch die Parameter bestimmt werden.

Methodensignatur(en):

getLog(*ComponentAddress* component, *DateTime* startTime, *DateTime* stopTime, *int* minLoglevel, *int* maxLogleve): *boolean*

Parameter:

component (*ComponentAddress*): Es werden nur Nachrichten zu der ComponentAddress zurück geliefert.

startTime (*DateTime*): Es werden nur Nachrichten die älter als startTime sind zur zurück geliefert.

stopTime (*DateTime*): Es werden nur Nachrichten die jünger als stopTime sind zur zurück geliefert.

minLoglevel (*int*): Priorität der Nachrichten ist mindestens so hoch wie minLoglevel

maxLogleve (*int*): Priorität der Nachrichten ist höchstens so hoch wie maxLoglevel

Rückgabe:

Log [] Array der Nachrichten die zu den Kriterien passen.

Methode: getAllLogs

GetAllLogs wird dazu benutzt, alle Nachrichten, die zwischen einem übergebenen Loglevel und Zeitpunkt liegen, abzurufen.

Methodensignatur(en):

getLog(*DateTime* startTime, *DateTime* stopTime, *int* minLoglevel, *int* maxLogleve): *boolean*

Parameter:

startTime (*DateTime*): Es werden nur Nachrichten die älter als startTime sind zurückgeliefert.

stopTime (*DateTime*): Es werden nur Nachrichten die jünger als stopTime sind zurückgeliefert.

minLoglevel (*int*): Priorität der Nachrichten ist mindestens so hoch wie minLoglevel

maxLogleve (*int*): Priorität der Nachrichten ist höchstens so hoch wie maxLoglevel

Rückgabe:

Log [] Array der Nachrichten die zu den Kriterien passen.

B.7 MonitoringService

MonitoringService wird von der MonitoringControl zur Verfügung gestellt und implementiert die Methode setNewPacketList(). Die Methode nimmt eine PacketList entgegen, in der die einzelnen Paketpositionen enthalten sind, die an die interne MonitoringControl weitergeleitet werden. Eine PacketList ist eine Liste von Paketen, in der für jedes Paket Informationen wie ID und Positionen zur Verfügung stellt.

Methode: SetNewPacketList

Methodensignatur(en):

SetNewPacketList (*PacketList* pl, *ComponetAdress* ca): *boolean*

Parameter:

pl (*PacketList*): Eine Liste von Paketen wird übergeben.

ca (*ComponentAddress*): Die Adresse der Komponente, die die neuen Pakete gesendet hat.

Rückgabe:

true: falls die PacketList erfolgreich übermittelt wurde.

false: falls die PacketList nicht erfolgreich übermittelt wurde.

B.8 Supervisor

Der Supervisor als eine sehr komplexe Komponente bietet mehrere Dienste an:

- PerformanceService
- VotingService
- CollectionService
- ComponenentAllocationService

Im Folgenden werden die von den Subkomponenten des Supervisors angebotenen Web-Services detailliert dargestellt

B.8.1 PerformanceService

Methode: testService Eine Testmethode, die nur dazu dient, die Zeit eines Web Service Methodenaufrufs zu ermitteln, d.h. innerhalb der Methode wird nichts berechnet, sondern direkt *true* zurückgegeben.

Methodensignatur(en)

testService(): *boolean*

Parameter: -

Rückgabe:

true Falls der Service erfolgreich aufgerufen wurde (immer der Fall).

false Falls der Serviceaufruf erfolglos blieb (wird hier nicht passieren, dient nur der Vollständigkeit halber).

Methode: getPerformance Mit getPerformance() kann der Performance-Wert eines SVs erfragt werden. Dieser Wert gibt die Performance eines SVs wieder und dient zur Verteilung der Systemkomponenten an die SVs. AllocationControl nutzt diesen Wert beispielsweise zur Berechnung wieviele Komponenten überhaupt übernommen werden können.

Methodensignatur(en)

getPerformance(): *float*

Parameter: -

Rückgabe:

float Der lokal errechnete Performance Wert.

B.8.2 VotingService

Methode: voteSV VoteSV dient dazu, einem anderen SV eine Stimme zu geben. Dies bedeutet automatisch, dass sich der „Stimme abgebende“ als möglichen PSV ausschließt. Den genauen Voting-Algorithmus befindet sich im Abschnitt VotingControl des Kapitels 9.8.2.

Methodensignatur(en):

voteSV(*ComponentAddress* sourceSV): *boolean*

Parameter:

sourceSV (*ComponentAddress*): CA des SV, der mitteilt, dass er selbst nicht mehr zur Wahl des Primary-SV kandidiert

Rückgabe (dient nur zum Debuggen, nicht benötigt):

true Stimme bestätigt

false Stimme nicht gezählt

Methode: initVoting Hiermit wird ein neues Voting initiiert. Diese Methode sollte bei allen SVs aufgerufen werden um eine Wahl durchzuführen.

Methodensignatur(en):

initVoting() : *boolean*

Parameter: -

Rückgabe:

true Wahlen bestätigt

false Wahlen nicht bestätigt

Methode: notifyPSV Der PSV stellt sich vor.

Methodensignatur(en):

notifyPSV(*ComponentAddress* psvCA): *boolean*

Parameter:

psvCA (*ComponentAddress*): CA des PSV

Rückgabe:

true PSV-Wahl angenommen, und PSV Bekanntgabe

false PSV-Wahl abgelehnt(falls einer die Wahl nicht akzeptiert, werden Neuwahlen verkündet)

Methode: getCurrentPSV Gibt den die ComponentAddress des Primärsupervisors zurück zurück; ist kein Primärsupervisor vorhanden so ist "null" der Rückgabewert.

Methodensignatur(en):

getCurrentPSV() : *ComponentAddress*

Parameter: -

Rückgabe:

ComponentAddress des PSV, falls *null*: PSV nicht bekannt bzw. nicht vorhanden

B.8.3 CollectionService

Methode: getCollection Anfordern der gesamten Collection, zum Beispiel nach dem Neustart eines SVs oder nach dem Ausfall des Primärsupervisors, um Inkonsistenzen zu erkennen und zu beseitigen.

Methodensignatur(en):

getCollection(): *CollectionObject*

Parameter: -

Rückgabe:

CollectionObject : Die komplette Collection in einem Objekt.

Methode: updateCollection updateCollection() verändert oder erzeugt einen Datensatz in der Collection. Sofern schon ein Eintrag besteht, werden grundsätzlich alle Einträge durch die übergebenen überschrieben. Dabei repräsentiert die ComponentAddress der Komponenten den Primärschlüssel, so dass niemals doppelte Einträge entstehen können. updateCollection() (also ein Schreibzugriff auf die Collection) wird immer nur durch den Primärsupervisor bei den SVs aufgerufen. Möchten Supervisor Einträge verändern oder erstellen, so rufen sie requestCollectionUpdate() beim Primärsupervisor auf.
Hinweis: Bei Neuerzeugung eines Datensatzes wird zusätzlich noch intern die modifiyCEListEntry()-Methode aufgerufen um ein CE als zugewiesen zu markieren.

Methodensignatur(en):

updateCollection(*ComponentAdress* component , *ComponentAdress* sv, *int*[] jobIDs, *int* status, *int* revisionNr):*boolean*

Parameter:

component (*ComponentAddress*): Die Komponente, deren Datensatz neu hinzugefügt oder verändert werden soll

sv (*ComponentAddress*): Spezifiziert den für diese Komponente zuständigen SV

jobIDs (*int*[]): Eine Liste mit JobIDs; hat den Wert *null* falls es eine permanente Komponente ist.

status (*int*): Der neue Statuswert für diese Komponente

revisionNr (*int*): Die neue Revisionsnummer für die Collection

Rückgabe:

true : Das Update war erfolgreich.

false : Das Update konnte nicht durchgeführt werden.

Methode: requestCollectionUpdate Mit requestCollectionUpdate() kann ein SV eine Anfrage auf Veränderung/Erzeugung eines Datensatzes der Collection an den PSV stellen. Diese Methode dient gleichzeitig als Reservierung einer Komponente, denn es kann passieren, dass mehrere SVs gleichzeitig eine Komponente übernehmen wollen. Der "Gewinner" muss erst einmal allen anderen SVs per updateCollection()-Aufruf mitgeteilt werden, was Zeit kostet. Daher dient requestCollectionUpdate() auch als eine Art Reservierung, so dass die Zuordnung einer Komponente zu einem SV lokal beim PSV gespeichert wird (Collection-Eintrag), bis diese Zuweisung offiziell und somit durch updateCollection()-Aufruf allen SVs bekannt ist. Bis zum Aufruf von updateCollection() gilt diese Zuordnung nur als lokal beim PSV reserviert und alle weiteren Reservierungsanfragen für diese Komponente werden vom PSV mit *false* beantwortet, so dass eine andere Komponente (zwecks Steuerung/Übername) von der jeweiligen DecisionLogic ausgewählt wird.

Methodensignatur(en):

requestCollectionUpdate(*ComponentAdress* component , *ComponentAdress* sv): *int*

Parameter:

component (*ComponentAddress*): Spezifiziert die Komponente, deren Datensatz verändert oder neu hinzugefügt werden soll.

sv (*ComponentAddress*): Spezifiziert den für diese Komponente zuständigen SV.

Rückgabe:

int : Falls das Update angenommen wurde, wird eine Revisionsnummer zurückgegeben, ansonsten -1

requestCollectionUpdate(*ComponentAdress* component , *ComponentAdress* sv, *int* status): *int*

Parameter:

component (*ComponentAddress*): Spezifiziert die Komponente, deren Eintrag verändert oder neu hinzugefügt werden soll.

sv (*ComponentAddress*): Spezifiziert den für diese Komponente zuständigen SV.

status (*int*): Der Status, der für die angegebene Komponente gesetzt werden soll.

Rückgabe:

int : Falls das Update angenommen wurde, wird eine Revisionsnummer zurückgegeben, ansonsten -1.

Methode: addJobs Mit dieser Methode kann man Jobs bei einem flüchtigen PM hinzufügen. Dies dient hauptsächlich als Zähler (permanente PMs haben den Wert null), hat aber den Vorteil über das Wissen der Job-IDs (bei Fehlerfällen können sie hier identifiziert und entfernt werden). Aufgerufen wird addJobs ausschließlich von der ComponentAllocationControl, genauer gesagt wenn per *Orderform Intermediates* angegeben werden. Diese werden pro Order der ComponentAllocationControl mitgeteilt, die neben einer eventuellen Neuverteilung der Komponenten auch per AddJobs die Collection aktualisiert.

Methodensignatur(en):

addJobs(***ComponentAdress*** **component ,** ***int***[] **newJobs):** ***boolean***

Parameter:

component (*ComponentAddress*): Spezifiziert das "flüchtige" PM, deren Job-ID-Liste bearbeitet werden soll.

newJobs (*int*[]): Array der Job-IDs, die hinzugefügt werden sollen.

Rückgabe:

true: Das Hinzufügen von Job(s) war erfolgreich.

false: PM nicht vorhanden oder ist ein permanentes PM; es muss ein neuer Eintrag per requestCollectionUpdate()-Aufruf erzeugt werden.

Methode: removeJob Es wird ein Job eines flüchtigen PM entfernt. Dies geschieht durch die CollectionControl, die sich für dass Event JOB_PASSED anmeldet und in dem Fall den entsprechenden Job aus der Collection entfernt.

Methodensignatur(en):

removeJob(***ComponentAdress*** **component ,** ***int*** **theJob):** ***boolean***

Parameter:

component (ComponentAddress): Spezifiziert das flüchtige PM, dessen Job-ID-Liste bearbeitet werden soll.

theJob (*int*): Job-ID, die aus der Collection gelöscht werden soll.

Rückgabe:

true: Entfernen des Jobs war erfolgreich; es wird auf ein updateCollection()-Aufruf des PSV gewartet.

false: PM nicht vorhanden oder ist ein permanentes PM.

Methode: getRevision Diese Methode gibt lediglich die Revisionsnummer der Collection zurück. Diese Methode dient hauptsächlich der Vollständigkeit und zu Debugzwecken, da die Revisionsnummer sowieso bei jedem Collection-Update mitgeliefert wird. Diese Methode kann natürlich an jedem SV ausgeführt werden, am sinnigsten ist aber natürlich der Aufruf am PSV, da dieser die aktuellste Collection und somit die aktuellste Revisionsnummer hat.

Methodensignatur(en):

getRevision() : *int*

Parameter: -

Rückgabe:

integer die Revisionsnumer

B.8.4 ComponenentAllocationService

Methode: allocateIntermediate Übergabe eines Intermediates und ein Array mit der Job Liste.

Methodensignatur(en):

allocateIntermediate(*ComponentAddress* IntermediateAddress, *JOB_ID*[] Job_ID_Array): *int*

Parameter: -

IntermediateAddress (*ComponentAddress*[]): ComponentAddress Intermediate

Job_ID_Array (*JOB_ID*[]): Job_IDs der Intermediates

Rückgabe:

int die Revisionsnumer

Methode: getUtilization Rückgabe der Auslastung eines SVs, wieviele Komponenten wurden schon initialisiert, und können noch initialisiert werden.

Methodensignatur(en):

getUtilization():*int*[]

Parameter: -

Rückgabe:

int[] Auslastung des Steuerknotenrechners; int[0]: maximal erzeugbare Components, int[1] erzeugte Components

B.9 OrderService

Das OrderManagement bietet nach außen nur Schnittstellen durch den OrderService an (siehe Abb. B.1), der im Folgenden dargestellt wird.

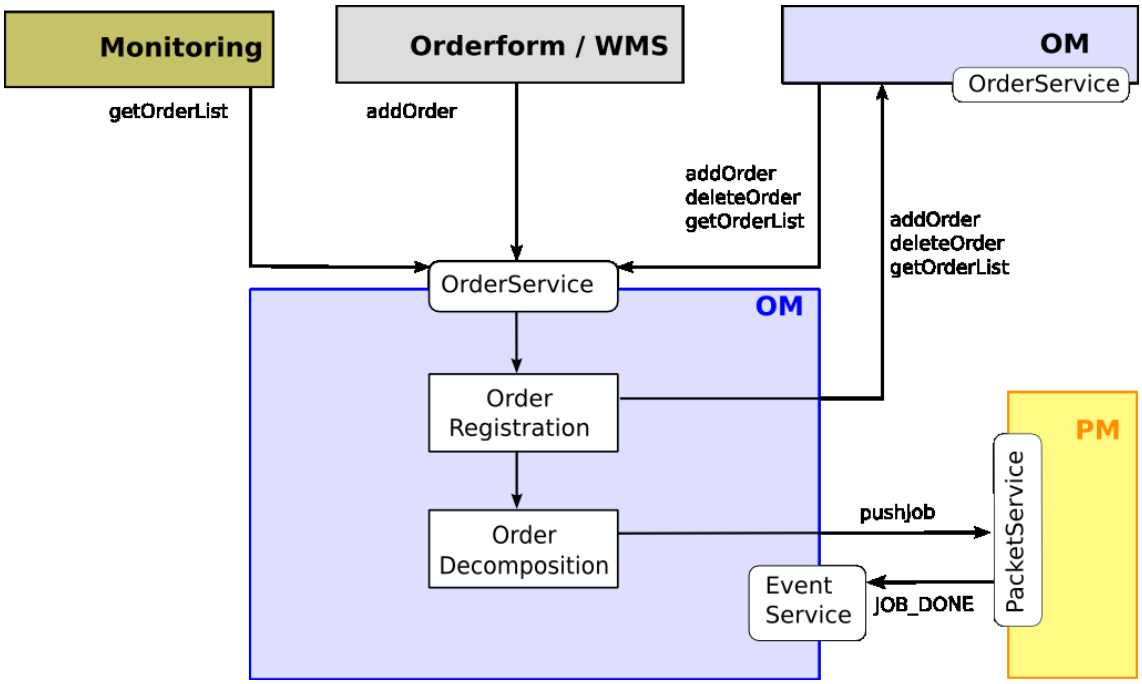

Abbildung B.1: Durch den OrderService angebotene Schnittstellen des OrderManagements

Methode: addOrder

Diese Methode fügt ein Orderobjekt der OrderList der im System vorhandenen Aufträge hinzu. Ist der Status dieser Order *new*, muss der Auftrag mit neuem Status *registered* an andere OrderManagements weitergeleitet werden.

Methode: deleteOrder

Diese Methode erwartet die ID eines Auftrags und löscht diesen aus dem System. Diese Methode wird ausgeführt, sobald ein Auftrag beendet ist, also alle seine zugehörigen Jobs das Ziel erreicht haben. Das OrderManagement, an dem der Auftrag endet, also das letzte JOBDONE-Event eingegangen ist, ruft diesen Web Service bei allen anderen OrderManagements auf.

Methode: getOrderList

Diese Methode stellt die Liste aller im System befindlichen Aufträge bereit. Dies wird zum einen vom Monitoring genutzt, zum anderen zur Synchronisation der Auftragsliste zwischen den OrderManagements. Kommt ein OrderManagement neu ins System, holt es sich selbständig über diese Methode die aktuelle OrderList eines anderen OrderManagements. So erhält auch ein neues OrderManagement sofort die vollständige Liste aller Order.

B.10 PacketService

Der PacketService stellt die Gesamtheit aller Web Service Methoden dar, die vom PacketManagement angeboten werden. Diese Schnittstelle wird vom PacketManagement implementiert und leitet die Aufrufe bei Bedarf an die Subkomponenten weiter. Genutzt werden die Methoden des PacketService vom OrderManagement, SystemManagement und anderen PacketManagements. Im folgenden Abschnitt werden die Methoden definiert, die in dieser Schnittstelle angeboten

werden. Für einen besseren Überblick zeigt Abb. B.2 alle Web Service Aufrufe, die vom Packet-Management gesendet oder empfangen werden.

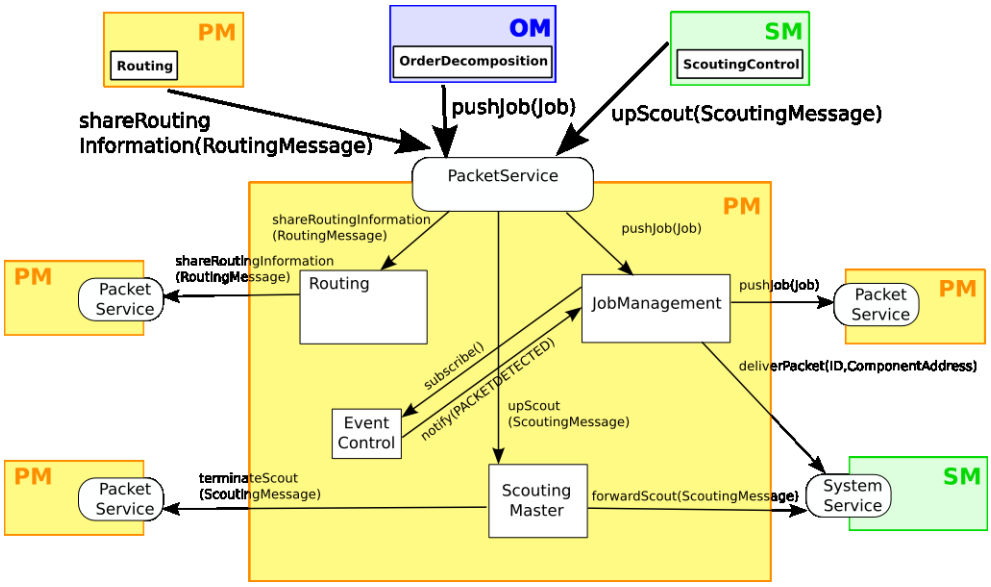

Abbildung B.2: Web Service Übersicht des PacketManagements

Methode: pushJob

Nimmt Jobs an, damit diese verarbeitet werden können. Der Job wird innerhalb des PacketManagement an die Subkomponente JobManagement weitergeleitet und von dieser bearbeitet.

Methode: upScout

Wird von der ScoutingControl aufgerufen, wenn diese erkannt hat, dass ein PacketManagement auf demselben Knoten existiert. Wird innerhalb des PacketManagement an die Subkomponente ScoutingMaster weitergeleitet und von dieser bearbeitet.

Methode: terminateScout

Schickt die Scouting Nachricht zu ihren Ursprung zurück und beendet so einen Scouting Durchlauf. Wird innerhalb des PacketManagement an die Subkomponente ScoutingMaster weitergeleitet und von dieser bearbeitet.

Methode: shareRoutingInformation

Dient dem Verbreiten von Routing Informationen in der Anlage. Wird innerhalb des PacketManagement an die Subkomponente Routing weitergeleitet und von dieser bearbeitet.

B.11 SystemManagement

B.11.1 SystemService

Methode: deliverPacket

Das PacketManagement übergibt dem SystemManagement mit deliverPacket() eine PaketID und die ComponentAddress eines folgenden SystemManagements, zu dem das Paket weitergeleitet

werden soll. Ein Aufruf wird an die Methode setExitForPacket() der Subkomponente CEControl weitergereicht.

Methodensignatur(en):

deliverPacket(***int*** **packet_id,** ***ComponentAddress*** **address):** ***boolean***

Parameter:

packet_id (*int*): ID des Pakets (RFID-Code)

address (*ComponentAddress*): Die ComponentAddress des SMs, an das das Paket weitergeleitet werden soll

Rückgabe:

true falls address gültiges Nachfolge-SM beschreibt

false falls address kein gültiges Nachfolge-SM beschreibt

Methode: forwardScout

Diese Methode soll eine Scoutnachricht weiterleiten und die Metrik des eigenen SMs hinzufügen. Ein Aufruf wird an die Methode forwardScout() der Subkomponente ScoutingControl weitergereicht.

Methodensignatur(en):

forwardScout(***ScoutingMessage*** **msg):** ***boolean***

Parameter:

msg (*ScoutingMessage*): beschreibt die Scoutnachricht, sie enthält das nächste zu erreichende SM

Rückgabe:

true falls address gültiges Nachfolge-SM beschreibt

false falls address kein gültiges Nachfolge-SM beschreibt

Methode: startScout

Genau wie forwardScout(), nur wird hierbei die Metrik anders berechnet.

Methode: getSuccessors

Diese Methode liefert die Nachfolge-SMs, genauer die ComponentAddress der jeweiligen Nachfolger. Ein Aufruf wird an die Methode getSuccessors() der Subkomponente NeighbourControl weitergeleitet.

Methodensignatur(en):

getSuccessors(): ***ComponentAddress***[] **successors**

Parameter: -

Rückgabe:

ComponentAddress[]: Liste aller Nachfolge-SMs

B.11.2 CapacityService

Methode: getCapacity

Diese Methode liefert die maximale Gesamtkapazität der Komponente zurück. Sie ergibt sich aus der Summe der Kapazitäten aller Trackparts. Dieser Aufruf wird an die CapacityControl weitergeleitet.

Methodensignatur(en):

getCapacity(void): ***int***

Parameter: -

Rückgabe:

int maximale Kapazität der Komponente

Methode: bookCapacity

Diese Methode soll einmalig eine Kapazitätseinheit für eine Komponente buchen. Dieser Aufruf wird an die CapacityControl weitergeleitet. Ist die Kapazität wieder frei, nachdem sie beim Aufruf erschöpft war, wird ein Event RELEASE_PACKET geworfen. Dazu muss sich die buchende Komponente für dieses Event registrieren, was in der Initialisierung jedes Elements geschieht.

Methodensignatur(en):

bookCapacity(***ComponentAddress*** **ca ,** ***int*** **packet_id):** ***boolean***

Parameter:

ca *ComponentAddress* die Adresse der Komponente, welche die Kapazität buchen will

packet_id die PaketID des Pakets, für das Kapazität gebucht werden soll

Rückgabe:

true Buchung erfolgreich

false Buchung fehlgeschlagen

B.12 InterpolationService

Methode: addVirtualDevice

Über diese Methode kann das SystemManagement virtuelle Devices anlegen. Ein virtuelles Device ist einfach ein imaginärer Punkt im Raum. Ein Event wird beim Eintritt und Austritt dieses fixen Punktes von einem fahrenden Paket ausgelöst.

Methodensignatur(en):

addVirtualDevice(***int*** **ID,** ***Coordinate*** **position):** ***boolean***

Parameter:

ID (*int*): Die ID dieses virtuellen Devices.

position (*Coordinate*): Position, an der das virtuelle Device angelegt werden soll.

Rückgabe:

true Anlegen war erfolgreich

false Anlegen ist fehlgeschlagen

Methode: getPacketList

Über diese Methode kann das SystemManagement oder das Monitoring eine Liste aller im System vorhandenen Pakete zurückgeliefert bekommen.

Methodensignatur(en):

getPacketList(): *Packet*[]

Parameter: -

Rückgabe:

Packet[] beinhaltet eine Sequenz vom Packet-Elementen die auf dem Förderelement vorhanden sind.

Methode: passPacket

Mit dieser Methode wird ein virtuelles Paket von einer Interpolation an die andere übergeben.

Methodensignatur(en):

passPacket(*Packet* packet, *WayTimeSimCalculation* avgSpeedCalc): *boolean*

Parameter:

packet (*Packet*): Paket, dass an die nächste Interpolation übergeben werden soll.

avgSpeedCalc (*WayTimeSimCalculation*): Mit diesem Parameter kann die Durchschnittsgeschwindigkeit von Förderelementen bestimmt werden, die keinen oder nur einen Sensor haben.

Rückgabe:

true Übergabe erfolgreich

false Übergabe fehlgeschlagen

B.13 DeviceService

Die Methode describe liefert eine Repräsentation des ConveyingElements, dessen Ansteuerung der jeweilige DeviceStateMapper anbietet. Dieser Aufruf wird beispielsweise initial von jedem SystemManagement an dem zugeteilten DeviceStateMapper aufgerufen, um das zu steuernde Föderelement zu erkundschaften.

Methode: describe Liefert eine komplette Beschreibung der angeschlossenen Hardware zurück.

Methodensignatur(en):

describe(): ConveyingElement

Parameter: -

Rückgabe:

ConveyingElement, falls die DWSI-Instanz/sim. Hardware mit einer gültigen Instanz initialisiert wurde, *null*, falls ein Fehler aufgetreten ist.

Methode: read Die Methode *read* liest den aktuellen Zustands des *Devices* mit der angegebenen ID aus und liefert diesen zurück.

Methodensignatur(en):

read(*int* id): *State*

Parameter:

id (*int*): ID des Devices, dessen Zustand gelesen werden soll.

Rückgabe:

State falls ein Device mit der angegebenen ID gefunden wurde und der Zustand erfolgreich gelesen wurde. *null* falls die Id unbekannt ist oder der Zustand nicht gelesen werden konnte.

Methode: write Die Methode *write* schreibt einen neuen Zustand in das Device mit der angegebenen ID und liefert zurück, ob diese Operation erfolgreich war.

Methodensignatur(en):

write(*int* id, *State* newState): *boolean*

Parameter:

id (*int*): ID des Devices, dessen Zustand geschrieben werden soll.

newState (*State*): Der neue Zustand des Devices.

Rückgabe:

true falls der Zustand erfolgreich geschrieben wurde

false falls während der Schreiboperation ein Fehler aufgetreten ist

C Übersicht der Förderlemente

An dieser Stelle soll ein kurzer Überblick über die verwendeten Förderelemente gegeben werden. Dabei werden die Forderelemente entsprechend ihrer Zugehörigkeit vorgestellt.

C.1 Knotenrechner UC1

An dem Knotenrechner UC1 sind die Förderelemente U1, U2, U3, U4, U6 und U7 angeschlossen.

Förderelement U1

- **Typ:** Rollenförderer
- **Logik:** Wareneingang mit Zusammenführung und Weiche.
- **Antrieb:** SEW-Eurodrive SF31 DT71D-4
- **Sensorik:**
  - *Lichtschranke 1:* Visolux RL 21-54-1447
  - *Lichtschranke 2:* Sick Optex WL 260
  - *Lichtschranke 3:* Visolux RL 21-54-1447
  - *RFID-Antenne:* Baumer Ident AN.37 für OIS-U CU.33
- **Weiche:** Hubstuhl
- **Nachfolger:** U2, U6

Förderelement U2

- **Typ:** Rollenstauförderer
- **Logik:** Puffer.
- **Antrieb:** SEW-Eurodrive SF31 D71D-4
- **Sensorik:**
  - *Touchsensor:* Hersteller unbekannt
  - *Lichtschranke:* Visolux RL 21-54-1447
- **Weiche:** -
- **Nachfolger:** U3

Förderelement U3

- **Typ:** 90°-Gurtkurve
- **Logik:** Einfacher Förderer.
- **Antrieb:** SEW-Eurodrive SF32 DT71D-4
- **Sensorik:** -
- **Weiche:** -
- **Nachfolger:** U4

Förderelement U4

- **Typ:** Rollenstauförderer
- **Logik:** Puffer mit einem Stopper.
- **Antrieb:** SEW-Eurodrive SF31 D71C-2
- **Sensorik:**
  - *Touchsensor 1:* Hersteller unbekannt
  - *Touchsensor 2:* Hersteller unbekannt
- **Weiche:** -
- **Nachfolger:** U5

Förderelement U6

- **Typ:** Rollenförderer
- **Logik:** Einfacher Förderer.
- **Antrieb:** SEW-Eurodrive SF31 DT71D-4
- **Sensorik:**
  - *Lichtschranke:* Sick Optex WL 260
  - *Touchsensor:* Hersteller unbekannt
- **Weiche:** -
- **Nachfolger:** U7

Förderelement U7

- **Typ:** Rollenförderer mit einem 45°-Kurvensegment.
- **Logik:** Einfacher Förderer.
- **Antrieb:** SEW-Eurodrive SF31 DT71C-2
- **Sensorik:**
  - *Lichtschranke:* Visolux RL 21-54-1447
- **Weiche:** -
- **Nachfolger:** U5

C.2 Knotenrechner UC2

An dem Knotenrechner UC2 sind die Förderelemente U5, U8 und U9 angeschlossen.

Förderelement U5

- **Typ:** Drehteller
- **Logik:** Zusammenführung.
- **Antrieb:** SEW-Eurodrive SF31 DT71C-2
- **Sensorik:** -
- **Weiche:** -
- **Nachfolger:** U8

Förderelement U8

- **Typ:** Rollenförderer
- **Logik:** Weiche
- **Antrieb:** SEW-Eurodrive SF31 DT71D-4
- **Sensorik:**
  - *Lichtschranke 1:* Visolux RL 21-54-1447
  - *Lichtschranke 2:* Visolux RL 21-54-1447
  - *RFID-Antenne:* Baumer Ident AN.37 für OIS-U CU.33
  - *Touchsensor:* Hersteller unbekannt
- **Weiche:** Bandabweiser
- **Nachfolger:** U9, U10

Förderelement U9

- **Typ:** Rollenförderer mit einem 45°-Kurvensegment.
- **Logik:** Warenausgang
- **Antrieb:** SEW-Eurodrive R40 DT80K-4
- **Sensorik:**
  - *Lichtschranke:* Sick Optex WL 12
- **Weiche:** -
- **Nachfolger:** -

C.3 Knotenrechner UC3

An dem Knotenrechner UC3 sind die Förderelemente U10, U11, U12 und U13 angeschlossen.

Förderelement U10

- **Typ:** Gurtbandförderer.
- **Logik:** Einfacher Förderer.
- **Antrieb:** SEW-Eurodrive R40 DT80K-4
- **Sensorik:** -
- **Weiche:** -
- **Nachfolger:** U11

Förderelement U11

- **Typ:** Rollenförderer mit 90°-Rollenbahnkurve.
- **Logik:** Einfacher Förderer.
- **Antrieb:** SEW-Eurodrive SF31 DT71C-2
- **Sensorik:** -
- **Weiche:** -
- **Nachfolger:** U12

Förderelement U12

- **Typ:** Gliederbahnförderer.
- **Logik:** Einfacher Förderer.
- **Antrieb:** SEW-Eurodrive KA46T DT80N-4
- **Sensorik:** -
- **Weiche:** -
- **Nachfolger:** U13

Förderelement U13

- **Typ:** Rollenförderer mit 90°-Rollenbahnkurve.
- **Logik:** Einfacher Förderer.
- **Antrieb:** SEW-Eurodrive KA46T DT80N-4
- **Sensorik:**
  - Lichtschranke: Visolux RL 21-54-1447
- **Weiche:** -
- **Nachfolger:** U1

Literaturverzeichnis

[1] Silberschatz, Galvin, Gagne: *Operating System Concepts with Java, Sixth Edition*, John Wiley & Sons Inc. (2004)

[2] Flanagen, D.: *Java in a Nutshell, Deutsche Ausgabe für Java 1.2 und 1.3*, Köln, O'Reilly (2000)

[3] Oualline, S.: *Practical C++ Programming.* O'Reilly (1995)

[4] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: *Design Patterns.* Addison-Wesley (1995)

[5] Wegener, I.: *Datenstrukturen.* Vorlesungsskript, Universität Dortmund (2001/2002)

[6] ten Hompel, M.; Sondhof, U.; Libert, S.: *Vorzge dezentraler autonomer Steuerungssysteme.* In: F+H 11/2004

[7] ten Hompel, M.; Sondhof, U.; Libert, S.:*Dezentrale Steuerung für Materialflußysteme - am Beispiel einer Stückgutförder- und Sortieranlage.* In: SPS/IPC/Drives Kongressband 2005

[8] Baumer Ident CU.30 Installationshandbuch, Reg. Nr.: UM.0701.DE; http://www.baumerident.com

[9] Baumer Ident Kommunikationsprotokoll, Reg. Nr.: UM.0730.DE, Best.-Nr.: 146788; http://www.baumerident.com

[10] Sun Developer Network (2006). http://java.sun.com/webservices/

[11] Wikipedia (2006) - Die freie Enzyklopädie. http://www.wikpedia.de

Abbildungsverzeichnis

Index