Realisierung eines geordneten Mehrzugbetriebs auf einer Modellbahnanlage

Diplomarbeit von
Jochen Koberstein

Kiel, den 21. Januar 2003

am Institut für Informatik und Praktische Mathematik
der Christian-Albrechts-Universität zu Kiel

Inhaltsverzeichnis

1 Einleitung **1**

2 Beschreibung der Modellbahnanlage **5**
2.1 Inner und Outer Circle 6
2.2 Bahnhöfe 7
2.3 Der Paß 7

3 Netzmodelle der Anlage **9**
3.1 Unidirektionale Strecken 9
3.2 Weichenbereiche 10
3.3 Eingleisige Paßstrecke 10
3.4 Paßstrecke mit Ausweichgleis 12

4 Hardware **15**
4.1 Sensoren und Aktoren 16
4.2 Interbus 16
4.2.1 Aufbau eines Interbus-Slaves 17
4.2.2 Datenübertragung 18
4.2.3 CRC-Sequenz 19
4.2.4 Ansteuerung des UART Master 19
4.2.5 Aufbau an der Modellbahnanlage 20
4.2.6 Übertragungszeiten 21
4.3 Fehlerquellen 23

5 Software **25**
5.1 Aufgabenstellung 25
5.2 Threads 26
5.2.1 Konzept 26
5.2.2 Implementierung 26
5.3 Objektrepräsentationen 27
5.3.1 Topologie der Modellbahnanlage 27
5.3.2 Darstellung eines Zuges 29
5.3.3 Darstellung eines Fahrplans 31
5.4 Fehlermeldung 31
5.5 Steuerung der Hardware 32
5.5.1 Serielle Schnittstelle 33
5.5.2 Interbus-Steuerung 33
5.5.3 Softwareschnittstelle 34
5.5.3.1 Adressierung von Sensoren und Aktoren 34
5.5.3.2 Schleife zur Datenübertragung 35
5.5.3.3 Auslesen von Reedkontaktmeldungen 35
5.5.3.4 Ansteuerung von Aktoren 36

5.5.3.5 Auslesen des Anlagenzustandes 36
5.5.4 Hardwarenahe Maßnahmen zur Zugsteuerung 36
5.5.5 Kommunikation mit anderen Threads 37
5.6 Implementierung von Petrinetzen 37
5.7 Steuerung des Fahrbetriebs 40
5.7.1 Erstellung von Fahrplänen 41
5.7.2 Reservierung von Gleisabschnitten 41
5.7.2.1 Validierung der Reedkontaktzählerstände 43
5.7.2.2 Auflösen von Konfliktsituationen 44
5.7.2.3 Prüfung der Blockbelegung 46
5.8 Grafische Oberfläche 47
5.8.1 Entwicklung und Implementierung 47
5.8.2 Kontroll- und Eingabebereiche 48
5.8.3 Eingriffe in den laufenden Fahrbetrieb 49

6 Zusammenfassung **51**

A Lichtsignale **53**

B Daten verschiedener Feldbussysteme **55**

C Illustrationen **57**

Abbildungsverzeichnis

2.1 Streckenlayout der Modellbahnanlage 5

3.1 Netzausschnitt einer unidirektionalen Strecke 9
3.2 Netzmodell eines Weichenbereichs 10
3.3 Netzmodell einer bidirektionalen Strecke 10
3.4 Netzmodell mit Deadlockvermeidung und Fairneßregelung 11
3.5 Erweiterung des Modells um eine Fairneßdurchbrechung 11
3.6 Modell der Paßstrecke mit Ausweichgleis 12
3.7 Deadlockfreies Modell der Paßstrecke mit Ausweichgleis 13

4.1 Schematischer Vergleich zentrale Verdrahtung ↔ Bussystem 15
4.2 Funktionsprinzip eines Reedkontaktes 16
4.3 Topologie des Interbus, wie er an der Modellbahnanlage zum Einsatz kommt. 17
4.4 Der Interbus arbeitet ähnlich einem räumlich verteilten Schieberegister. 17
4.5 Schematischer Aufbau eines Interbus-Slaves 18
4.6 Kommunikation zwischen Workstation und Master bei einem Schreib-/Lesezyklus. 20
4.7 Ausgabedaten für eine Platine mit zwei Slaves. 20
4.8 Eingabedaten für eine Platine mit zwei Slaves. 20
4.9 Schematischer Aufbau der Hardware zur Steuerung der Modellbahnanlage. 21
4.10 Übertragung eines Zeichens über die serielle Schnittstelle und den Interbus 21
4.11 Auslösen eines Reedkontaktes 24

5.1 Gliederung der Software und die Abhängigkeit der Teile voneinander. 25
5.2 Beispiel einer `block_status` Struktur 29
5.3 Hardwarebezogene Klassen in UML Darstellung 32
5.4 Schichtenmodell zur Datenübertragung 32
5.5 Schleife des Hardware-Threads 35
5.6 Der Puffer zur Speicherung von Meldungen über Reedkontaktbetätigungen ist als verkettete Liste implementiert. 36
5.7 Schleife des Hardware-Threads mit Zugriffen auf gemeinsam genutzte Datenstrukturen 37
5.8 UML-Klassenstruktur der Petrinetzimplementierung 38
5.9 Hauptschleife zur Zugsteuerung 40
5.10 Auswertung des Zugstatus 42
5.11 Reduziertes Netzmodell der Paßstrecke mit Ausweichgleis 43
5.12 Auswertung der Blockanforderungsliste 46
5.13 Anzeige zur Überwachung des Fahrplanfortschritts 48

C.1 Übersicht der Modellbahnanlage 57
C.2 Platine mit zwei Interbus-Slaves 58
C.3 Reedkontaktpaar 58
C.4 Fahrplaneingabe der grafischen Oberfläche 59
C.5 Eingabe der Zugparameter 59
C.6 Eingabemaske zur Rückführung von Zügen 60
C.7 Optionen zur Steuerung und Protokollierung 60

Erklärung

Hiermit erkläre ich, daß ich diese Diplomarbeit ausschließlich in Zusammenarbeit mit Oliver Schmitz und nur unter Gebrauch der angegebenen Hilfsmittel angefertigt habe.

Das Kapitel 3 (Netzmodelle) und Abschnitt 5.8 (Grafische Benutzeroberfläche) wurden dabei von Oliver Schmitz und das Kapitel 4 (Hardware) und Abschnitt 5.5 (Steuerung der Hardware) von mir in eigenständiger Arbeit erstellt.

Kiel, den 25. September 2003

Kapitel 1

Einleitung

Wenn man Systeme wie Behörden, Produktionsprozesse, Verkehrssysteme, Kommunikationseinrichtungen und Betriebssysteme betrachtet, fallen einem Gemeinsamkeiten auf: In ihnen laufen unterscheidbare Prozesse ab, die Ressourcen, wie z. B. Anträge oder Rohstoffe benötigen, oder bei denen Bedingungen, wie das Freiwerden einer Maschine, erfüllt sein müssen. Diese Ressourcen sind im allgemeinen nur begrenzt verfügbar. Die beteiligten Prozesse können unabhängig voneinander oder nebenläufig (concurrent) ablaufen. Allerdings können Prozesse durch das Beanspruchen gleicher Ressourcen in zufällige wechselseitige Abhängigkeiten geraten. In solchen arbeitsteiligen Systemen muß der Ablauf der Prozesse bezüglich ihrer Ressourcenanforderung durch Kommunikation mit einer Ressourcenverwaltung koordiniert werden, damit die gestellten Ziele erreicht werden können. Die Art des Systems ist irrelevant für die zu entwickelnden Lösungsverfahren.

Damit ein System korrekt arbeiten kann, müssen Sicherheits- und Lebendigkeitsbedingungen erfüllt werden. Zur Einhaltung der Sicherheitsbedingungen muß gewährleistet sein, daß das System von einem legitimen Zustand nur in einen ebenfalls legitimen Folgezustand übergeht, um Chaos oder Stillstand des Systems zu vermeiden. Dazu müssen systemspezifische Invarianten eingehalten werden.

Beanspruchen mehrere Prozesse eine Ressource, die nur exklusiv vergeben werden darf, muß der Zugriff durch eine globale Instanz koordiniert werden. Beginnt ein Prozeß die Nutzung einer Ressource, tritt er in eine kritische Region ein, in der er nicht terminieren darf. Dies ist eine Sicherheitsbedingung, da die Ressource in diesem Fall dauerhaft für andere Prozesse blockiert bliebe und das gesamte System so zum Stillstand kommen könnte. Diese Situation bezeichnet man als Verklemmung (deadlock). Verläßt ein Prozeß eine kritische Region, ist die entsprechende Ressource wieder freizugeben.

Die Lebendigkeit (liveness) eines Systems wird dadurch charakterisiert, daß Ereignisse irgendwann oder sogar unmittelbar stattfinden, sofern sie konzessioniert sind. Die Nutzung einer Ressource durch einen Prozeß findet also statt, sofern die Ressource verfügbar ist.

Wird ein Prozeß auf unbestimmte Zeit durch andere blockiert, spricht man vom Aushungern (starvation). Das ist z. B. der Fall, wenn eine Ressource von mehreren Prozessen beansprucht, aber nur einem oder einem Teil der Prozesse wiederholt zugeteilt wird. Dies muß bei der Vergabe der Ressourcen berücksichtigt werden.

Um Lösungen für die oben erwähnten Probleme zu entwickeln und zu testen, wurde am Lehrstuhl für Rechnerorganisation eine Modellbahnanlage aufgebaut. Am Beispiel der Modellbahn lassen sich stellvertretend für alle anderen arbeitsteiligen Systeme durch ein geeignet gewähltes Streckenlayout alle organisatorischen Probleme modellieren. Dazu sind auf der Anlage drei untereinander verbundene Ring-

strecken vorhanden, von denen zwei unidirektional und eine bidirektional befahren werden. Jeder Ring beinhaltet einen Bahnhof. In den Bahnhöfen befinden sich die Start- und Zielpositionen der Züge. Das Layout der Anlage wurde so gewählt, daß auf ihr die Gesamtheit der Probleme arbeitsteiliger Systeme auftreten.

Die Züge sind in diesem System die Prozesse, die die Gleise als Ressourcen beanspruchen. Um eine nebenläufige Ausführung der Prozesse bzw. einen Fahrbetrieb mit mehreren Zügen zu ermöglichen, sind die Ringe in Gleisabschnitte unterteilt. Aus Sicherheitsgründen darf jeweils nur ein Gleisabschnitt von einem Zug befahren werden. Die Belegung dieser Blöcke unterliegt dabei keinen Einschränkungen für einzelne Züge, d. h. jeder Zug darf jeden Block befahren.

Jeder Zug erhält vor Beginn des Fahrbetriebs einen individuellen Fahrplan, der in Form einer einfach verketteten Liste die Reihenfolge der zu befahrenden Gleisabschnitte vorgibt. Diese Listen müssen eine zusammenhängende Strecke auf der Anlage beschreiben und einen identischen Start- und Zielblock enthalten. Ansonsten sind die Fahrpläne der Züge beliebig und voneinander unabhängig. Die Abarbeitung eines Fahrplans geschieht inkrementell, sodaß jeweils nur möglichst wenige Gleisabschnitte der zu befahrenden Strecke von einem Zug beansprucht werden. Dadurch wird die nebenläufige Abarbeitung der Fahrpläne möglich.

Da die Blöcke nur im wechselseitigem Ausschluß zu vergeben sind, treten während des Fahrbetriebs Abhängigkeiten zwischen den Zügen auf, die von einer globalen Instanz zu verwalten sind. Für einen Zug muß, bevor er einen neuen Block befährt, überprüft werden, ob dieser noch nicht durch einen anderen Zug belegt ist. Ist der Block frei, kann der Zug unmittelbar einfahren; andernfalls darf er seinen aktuellen Block solange nicht verlassen, bis der nächste Block wieder freigegeben wurde. An den Weichen treten Konflikte auf, sobald zwei oder mehr Züge einen Nachfolgeblock beanspruchen. Diese Konflikte sind geordnet aufzulösen, z. B. durch eine Zufallsentscheidung oder durch die Vergabe von Prioritäten. Diese Methoden verhindern allerdings nicht, daß einzelne Züge benachteiligt werden. Um dies zu vermeiden, können z. B. dynamische Prioritätsverfahren (Aging) genutzt werden.

Auf den unidirektionalen Ringstrecken ist insbesondere die wechselseitige Zuteilung der Gleisabschnitte zu realisieren. Verklemmungen können innerhalb dieser Ringe nur auftreten, wenn ein oder mehrere Züge die Abarbeitung ihrer Fahrpläne beenden und so einzelne Gleisabschnitte nicht wieder freigegeben werden.

Der bidirektional befahrbare Ring erfordert zusätzliche Koordination. Es dürfen z. B. zwei Züge nicht gegenläufig einfahren, da sie sich so gegenüberstehen und den weiteren Ablauf blockieren würden. Um die Strecke besser auszulasten, sollten aber mehrere Züge in einer Richtung nachfolgen können. Dabei darf keine Fahrtrichtung bevorzugt werden, d. h. nach einer festgelegten Anzahl von Zügen ist die Richtung umzukehren. Hat ein Zug keinen Konkurrenten bzgl. eines zu befahrenden, freien Streckenabschnitts, ist ihm unmittelbarer Zugang zu gewähren. Die Hinzunahme einer Ausweichstrecke in der Mitte des eingleisigen Rings ermöglicht weitere Optimierungen des Zugbetriebs, führt jedoch zu deutlich komplexeren Bedingungen bzgl. der Belegung von Gleisabschnitten.

Verklemmungen können auf mehrere Arten auf der Anlage auftreten. Sie sind zum einen struktureller Natur, wie bei der bidirektionalen Strecke, in die gegenläufig eingefahren werden kann. Zum anderen ergeben sie sich aus dem Ablauf, so können mehrere Züge die Bahnhöfe und die Übergangsbereiche der einzelnen Ringe so blockieren, daß kein Betrieb mehr möglich ist. Solche Verklemmungen müssen zuvor erkannt und durch entsprechende Strategien vermieden werden.

Im Rahmen dieser Arbeit wird eine Lösung der dargestellten Probleme exemplarisch für die Modellbahnanlage entwickelt. Dazu ist ein Steuerungsprogramm zu implementieren, das den Zugbetrieb koordiniert. Aufgrund der Komplexität ist es notwendig, ein von der Hardware unabhängiges Modell zu entwickeln, das die

strukturellen Eigenschaften der Anlage berücksichtigt und die daraus resultierenden organisatorischen Probleme löst. Diese Modellierung geschieht in Form von Petrinetzen. Das vorhandene Streckenlayout läßt sich dabei direkt in ein Netzmodell umwandeln, das um Mechanismen zur Deadlockvermeidung und Gewährleistung der Fairneß erweitert wird.

Die Ansteuerung der räumlich verteilten Hardware wird mit Hilfe eines Feldbusses realisiert, der über eine ebenfalls zu entwickelnde Hardwarebibliothek angesteuert wird. Die Programmierung des gesamten Projektes wird objektorientiert in der Sprache C++ durchgeführt. Das Programm ermöglicht schließlich einen geregelten Mehrzugbetrieb, dessen Parameter mit Hilfe einer grafischen Benutzeroberfläche angepaßt werden können.

In dem nachfolgenden Kapitel 2 wird der Aufbau der Anlage erläutert. Darauf werden in Kapitel 3 die Netzmodelle der Anlage beschrieben. Kapitel 4 gibt eine Übersicht über die verwendete Hardware, das Kapitel 5 enthält Details zur Implementierung. Abgeschlossen wird die Arbeit in Kapitel 6 mit einer Zusammenfassung.

Kapitel 2

Beschreibung der Modellbahnanlage

Die Modellbahnanlage erstreckt sich auf einer Fläche von 18 m^2 und verfügt über ein Gesamtstreckenlänge von rund 100 m, die in 47 elektrisch voneinander getrennte Gleisabschnitte von 0.5 bis 3 m Länge unterteilt ist. Diese werden im Folgenden als Blöcke oder Gleisabschnitte bezeichnet. Die Fahrstrecke der Züge wird mit Hilfe von 28 Weichen bestimmt. Die nachfolgende Abbildung 2.1 zeigt das Streckenlayout.

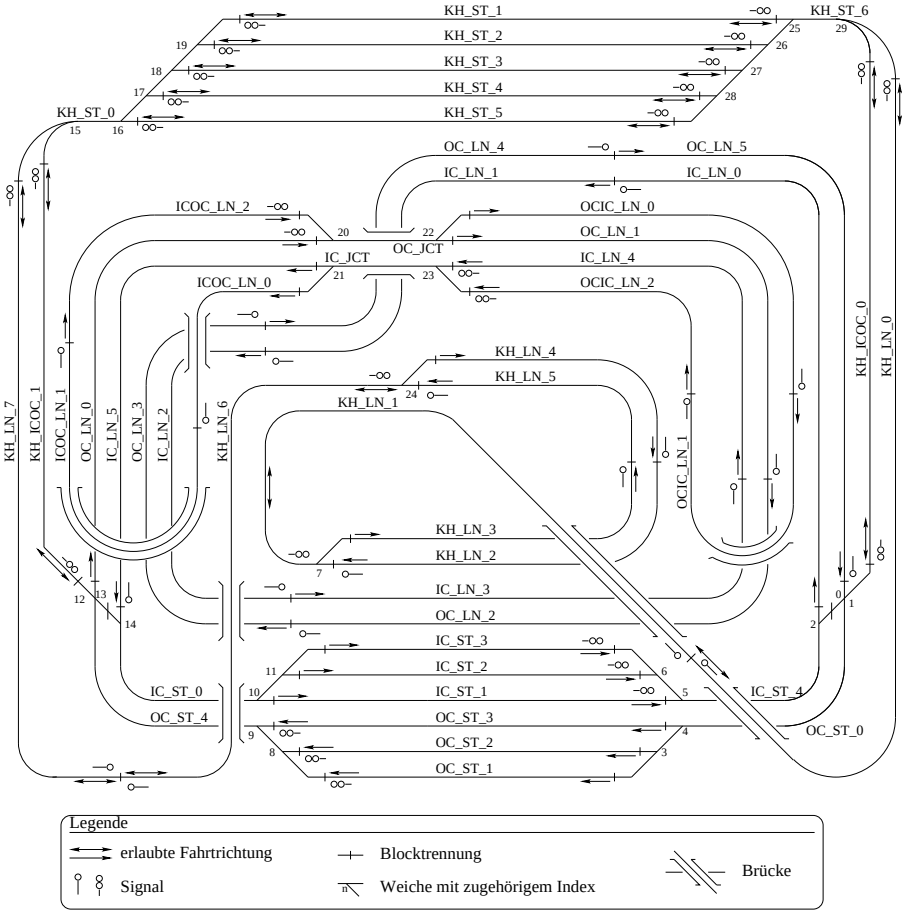


Abbildung 2.1: Streckenlayout der Modellbahnanlage

Die Anlage besteht aus drei Ringstrecken, die jeweils einen Bahnhof besitzen und miteinander verbunden sind. Blöcke, deren Länge ausreicht, um einen Zug vollständig aufzunehmen, verfügen über Reedkontaktpaare, die durch jeweils an Lokomotive und letztem Waggon befestigten Magnete betätigt werden. Diese Gleisabschnitte besitzen zur visuellen Kontrolle ein Lichtsignal und werden im Folgenden als Bahnhofs- und Streckenblöcke bezeichnet.

Die Bezeichnungen der Gleisabschnitte setzen sich aus drei Teilen zusammen:

- Ringstrecke
- Bahnhofs- oder Streckenblock
- laufende Nummer

Die Richtungsangaben werden im Folgenden mit “cw“ für clockwise und “ccw“ für counterclockwise abgekürzt.
Für den Zugverkehr gelten folgende Bedingungen:

- Jeder Zug besitzt einen individuellen Fahrplan. Dieser besteht aus einer einfach verketteten Liste, die die Reihenfolge der zu befahrenden Gleisabschnitte eindeutig festlegt. Dabei entspricht die Ausgangs- der Zielposition.
- Da die Züge keinerlei Identifikationsmerkmale besitzen, muß eine Positionsbestimmung anhand der Ausgangsposition und der betätigten Reedkontakte erfolgen. Die Zuteilung der Blöcke ist dann durch ein Steuerprogramm zu koordinieren.
- Ein Zug darf nur halten, wenn er sich vollständig in einem Strecken- oder Bahnhofsblock befindet. Es sind nur die in Abbildung 2.1 dargestellten Fahrtrichtungen erlaubt.
- Jeder Block kann durch jeden Zug beansprucht werden.
- Das Befahren eines Blocks ist irreversibel, d. h. ein in einen Block eingefahrener Zug darf diesen nur in Vorwärtsrichtung wieder verlassen.
- Es ist ein sicherer und lebendiger Betrieb zu gewährleisten. Das bedeutet, daß Blöcke nur in wechselseitigem Ausschluß zu befahren sind; auftretende Konfliktsituationen sind geordnet aufzulösen; der Betrieb ist verklemmungsfrei zu gestalten.

2.1 Inner und Outer Circle

Der Inner und Outer Circle sind eingleisige Ringstrecken, die parallel und in Form einer in sich verschlungenen Acht verlaufen. Beide Strecken werden ausschließlich unidirektional genutzt, wobei die Fahrtrichtungen gegenläufig zueinander sind. Ein Richtungswechsel ist nur über die Wendeschleifen möglich, die beide Ringe miteinander verbinden. Jeder Ring beinhaltet einen Bahnhof mit jeweils drei und eine Fahrstrecke mit jeweils sechs Streckenabschnitten.

Da ein Gleisabschnitt nur von jeweils einem Zug befahren werden kann, muß vor der Einfahrt sichergestellt werden, daß dieser durch keinen anderen Zug belegt ist. Ist das der Fall, muß die Weiterfahrt solange verhindert werden, bis der vorausfahrende Zug jenen Block freigegeben hat. Konflikte treten an den Weichenbereichen der Ringübergänge auf.

2.2 Bahnhöfe

Ausgangs- und Zielpositionen der Züge liegen immer auf Bahnhofsblöcken. In den Bahnhofsbereichen treten Konflikte auf: Stehen mindestens zwei Züge in gleicher Richtung mit nicht leerem Fahrplan in einem Bahnhof, beanspruchen diese den selben Nachfolgeblock. Da ein Block nur exklusiv vergeben werden kann, muß dieser Konflikt geeignet aufgelöst werden.

Deadlocks treten auf, wenn das Nachfolgegleis für einen Zug dauerhaft belegt ist. Dies ist z. B. dann der Fall, wenn mindestens sieben Züge in der gleichen Richtung einen der Ringe befahren, und für einen dieser Züge der Fahrplan abgearbeitet und das Heimatgleis noch belegt ist.

2.3 Der Paß

Der sogenannte Paß ist die bidirektional zu befahrende Ringstrecke, die über zwei Streckenblöcke an Inner und Outer Circle angebunden ist. Weiterhin ist ein Bahnhof mit fünf Gleisen vorhanden, der von jedem Zug durchfahren werden muß, um in die Paßstrecke zu gelangen. Diese beinhaltet drei Sektionen, die beiden äußeren mit jeweils zwei Blöcken; die mittlere Sektion ist zweigleisig ausgeführt. Dadurch kann der Paß entweder als eingleisiger Ring oder alternativ bidirektional mit Ausweichgleis genutzt werden. Der Bahnhof kann in beide Richtungen befahren werden, wobei die Richtung von der Einfahrt aus dem Inner oder Outer Circle abhängt.

Neben den für unidirektional befahrene Strecken geltenden Sicherheitsbedingungen muß das Auftreten weiterer Deadlocks vermieden werden: Wenn zwei Züge gegenläufig in die Paßstrecke einfahren, führt dies bei eingleisig genutzter Paßstrecke zwangsläufig zu einer Verklemmung, die den Verkehr über den Paß blockiert.
Wird jeweils nur eine Fahrtrichtung zugelassen, können zur besseren Auslastung weitere Züge nachfolgen. Es darf aber keine Richtung monopolisiert werden, andernfalls ist es möglich, daß ein in Gegenrichtung auf Einfahrt wartender Zug beliebig lange benachteiligt wird. Um einen solchen Fall zu verhindern, muß eine Fairneßregelung eingesetzt werden, die die maximale Anzahl an einfahrenden Zügen pro Richtung bis zu einem Richtungswechsel festlegt. Einem Zug, dem aufgrund der Fairneßregelung die Einfahrt in den Paß verweigert wird, der in der Gegenrichtung aber keinen Konkurrenten hat, ist der unmittelbare Zugang zum Paß zu gewähren.

Kapitel 3

Netzmodelle der Anlage

Die Verteilung der auf der Anlage befindlichen Züge sowie die Positionierung der Weichen können eine Vielzahl von Zuständen annehmen. Die Aufteilung in verteilte Komponenten hat zur Folge, daß man Ereignisse und deren Auswirkungen unabhängig voneinander betrachten kann. Das Weglassen einer zeitlichen Achse, auf der die Ereignisse geordnet werden, wurde in der Dissertation von Petri [18] vorgestellt. Die nach ihm benannten Netze erlauben die Modellierung von Zuständen und deren Übergänge.

Derartige Modelle lassen sich aus der Struktur der Modellbahnanlage unabhängig von Systemeigenschaften wie z. B. der Fahrgeschwindigkeit der Züge gewinnen und fließen unmittelbar in die Entwicklung des Steuerprogramms mit ein.

Auf die Einführung in die Theorie der Petrinetze wird hier verzichtet, man findet sie unter anderem in [19]. Die in den Abschnitten 3.1 bis 3.3 erläuterten Netze wurden im Rahmen der Vorlesung "System-Modellierung mit Petrinetzen" [16] vorgestellt. Die im Abschnitt 3.4 beschriebenen Modelle sind Erweiterungen dieser Netze.

3.1 Unidirektionale Strecken

Die Netzmodelle für die Inner und Outer Circle Strecken lassen sich direkt aus dem Layout der Anlage ableiten. Abbildung 3.1 zeigt einen Streckenabschnitt mit drei Blöcken.

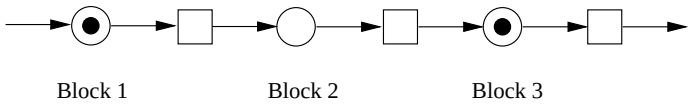


Abbildung 3.1: Netzausschnitt einer unidirektionalen Strecke

Zu jedem Block gehören eine Stelle und Transitionen, die die Ein- und Ausfahrt eines Blocks modellieren. Die Token repräsentieren Züge; ein Token auf einer Stelle zeigt an, daß der Block belegt ist, andernfalls ist er frei. Durch die Beschränkung von einem Token pro Stelle ist sichergestellt, daß Blöcke nur jeweils von einem Zug genutzt werden können.

In der Abbildung kann der Zug aus Block 1 in den zweiten Block einfahren, muß dann aber solange angehalten werden, bis der andere Zug Block 3 verlassen hat.

3.2 Weichenbereiche

In den Bahnhofsbereichen und den Übergangsstellen der Ringe treffen mehrere Blöcke in Weichenbereichen aufeinander, die dann einen oder mehrere Nachfolgeblöcke besitzen. Das entsprechende Netzmodell ist in der folgenden Abbildung 3.2 zu sehen.

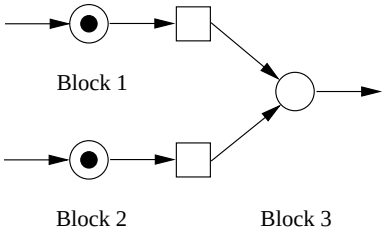


Abbildung 3.2: Netzmodell eines Weichenbereichs

In der Abbildung 3.2 stehen die in Block 1 und Block 2 befindlichen Züge in Konflikt um den nachfolgenden freien Block 3. Sobald der Konflikt zu Gunsten eines Zuges aufgelöst wird, kann dieser in den Block einfahren. Der unterlegene Zug kann dann erst bei Freiwerden des Blocks 3 nachfolgen, oder muß sich ggf. einer neu aufgetretenen Konfliktsituation stellen.

3.3 Eingleisige Paßstrecke

Die eingleisige Paßstrecke besteht aus sechs Blöcken, die in beide Richtungen befahren werden können. Die Strecke mündet an beiden Enden in den Paßbahnhof. Abbildung 3.3 stellt das Netzmodell dar.

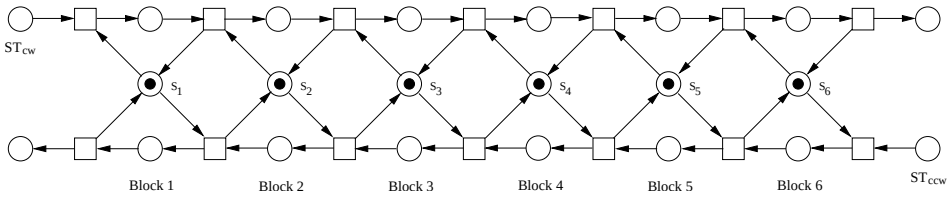


Abbildung 3.3: Netzmodell einer bidirektionalen Strecke

Zu jedem Block gehören drei Stellen, wobei die obere und untere angeben, ob ein Zug im oder gegen den Uhrzeigersinn den Block befährt. Die mit s_i gekennzeichneten Stellen zeigen an, ob der jeweilige Block frei ist. Zusätzlich gibt es zu jeder Richtung Transitionen, die die Ein- und Ausfahrt eines Blocks modellieren. Die Stellen ST_{cw} und ST_{ccw} stehen für den Bahnhof, in dem Züge in beiden Richtungen auf die Paßeinfahrt warten können.

Das Modell garantiert aber keinen lebendigen Betrieb. Es können von jeder Seite Züge in den Paß einfahren, so daß dieser blockiert wird. Um das zu verhindern, müssen zwei weitere Stellen V_{cw} und V_{ccw} (vgl. Abbildung 3.4) eingeführt werden, die initial mit sechs Token belegt werden. Ein Zug, der im Uhrzeigersinn den Paß befahren soll, muß von der Stelle V_{cw} ein Token und von V_{ccw} sechs Token abziehen können. Dadurch, daß nach Verlassen des ersten Blocks die sechs Token wieder zurückgelegt werden, können weitere Züge nachfolgen. Durch dieses Netz ist gewährleistet, daß die Einfahrt in den Paß nur in einer Richtung möglich ist.

Ein Problem, das bei diesem Modell auftreten kann, ist die Bevorzugung einer Fahrtrichtung. So können im Paßbahnhof wartende Züge durch andauernden

Gegenverkehr ausgehungert werden. Um eine Monopolisierung einer Fahrtrichtung zu vermeiden, müssen zwei weitere Stellen F_{cw} und F_{ccw} dem Modell hinzugefügt werden, die initial mit n Token belegt werden. Die Summe dieser Token legt die maximale Anzahl von Zügen in einer Richtung fest. Für einen Zug, der im Uhrzeigersinn den Paß befahren soll, wird ein Token von F_{cw} abgezogen und eines zu F_{ccw} hinzugefügt. Kann von der Stelle F_{cw} kein Token mehr abgezogen werden, können Züge in Gegenrichtung den Paß befahren. Gleiches gilt für die Gegenrichtung.

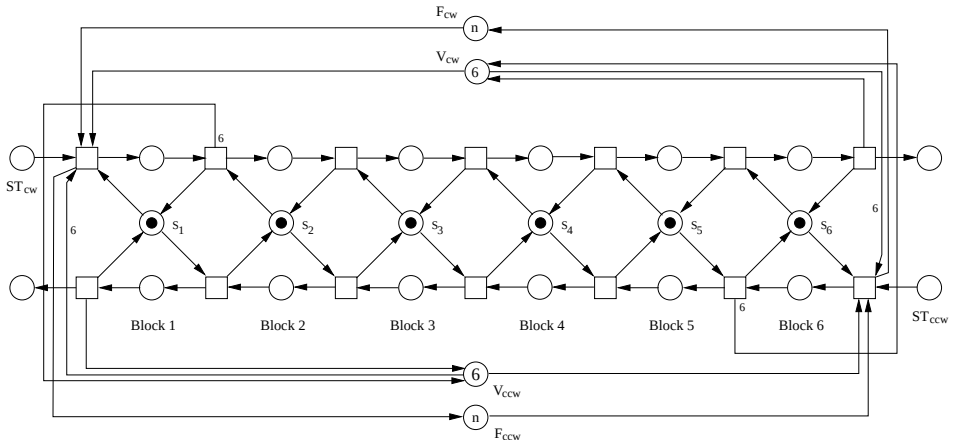


Abbildung 3.4: Netzmodell mit Deadlockvermeidung und Fairneßregelung

Durch das Einführen dieser Fairneßregelung ergibt sich aber ein weiteres Problem. Falls z. B. ein Zug in Richtung "cw" im Bahnhof wartet, dabei die Stelle F_{cw} kein Token enthält und kein Zug in Gegenrichtung den Paß befahren soll, ist keine Einfahrt in den Paß möglich. In diesem Fall ist die Fairneßregelung zu durchbrechen und dem Zug die Einfahrt zu gewähren. Dazu muß ein Token auf die Stelle F_{cw} gelegt werden. Die folgende Abbildung 3.5 zeigt die Erweiterung des Netzmodells aus Abbildung 3.4.

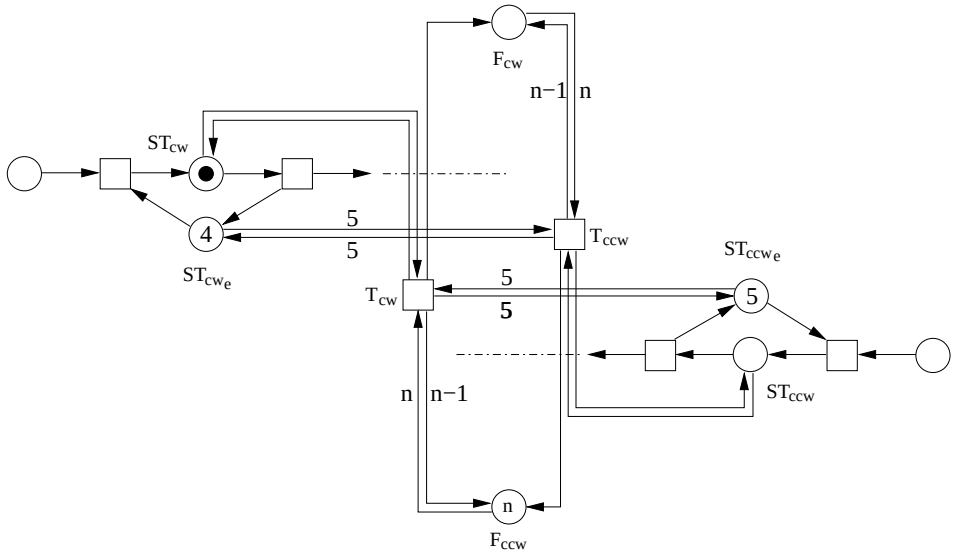


Abbildung 3.5: Erweiterung des Modells um eine Fairneßdurchbrechung

Dabei werden zu den Bahnhöfen ST_{cw} und ST_{ccw} zwei Stellen ST_{cw_e} und ST_{ccw_e} hinzugefügt, die die Zahl der freien Gleise in den Bahnhöfen repräsentieren. Mit

den zwei Transitionen T_{cw} und T_{ccw} wird überprüft, ob kein Zug in Gegenrichtung im Bahnhof wartet und die Fairneßregelung erschöpft ist, d. h., ob von den Stellen ST_{ccw_e} und ST_{cw_e} sowie den Fairneßstellen F_{ccw} und F_{cw} jeweils die volle Tokenanzahl abgezogen werden kann. Zurückgelegt werden auf die Stellen ST_{ccw_e} und ST_{cw_e} die volle Anzahl, auf die Stellen F_{ccw} und F_{cw} jeweils n-1 Token. Die Transition T_{cw} legt ein Token auf F_{cw}, analog dazu T_{ccw}.

In der Situation aus Abbildung 3.5 steht ein Zug "cw" im Bahnhof (ein Token auf ST_{cw}), kein Zug in Gegenrichtung (keine Token auf ST_{ccw}), und die aktuell mögliche Fahrtrichtung ist "ccw", d.h. F_{cw} ist leer. Mit Hilfe der Transition T_{cw} wird überprüft, ob von ST_{ccw_e} und F_{ccw} die volle Tokenanzahl abgezogen werden kann und ein Token in ST_{cw} vorhanden ist, d.h. die Bedingungen zum Durchbrechen der Fairneßregelung erfüllt sind. In diesem Fall werden die Token an ST_{ccw_e} und ST_{cw} wieder zurückgegeben, die Stelle F_{cw} erhält ein und F_{ccw} die restlichen Token. Dadurch kann der Zug in den Paß einfahren.

3.4 Paßstrecke mit Ausweichgleis

Bei der Paßstrecke mit Ausweichgleis werden die sechs Blöcke in drei Sektionen aufgeteilt (vgl. Abbildung 3.6). Dabei umfaßt jede Sektion zwei Blöcke, wobei die beiden äußeren Sektionen eingleisig und bidirektional befahrbar sind und die mittlere Sektion die Ausweichgleise beinhaltet, so daß für jede Richtung zwei Blöcke unidirektional zu befahren sind.

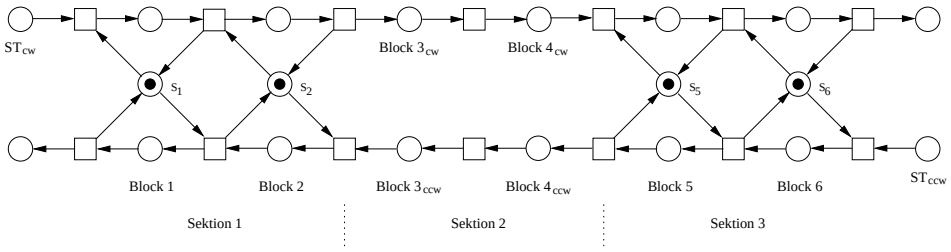


Abbildung 3.6: Modell der Paßstrecke mit Ausweichgleis

Damit ist, im Gegensatz zum eingleisigen Modell, ein verklemmungsfreier Betrieb bei gegenläufiger Einfahrt von Zügen in den Paß möglich. Aber auch in diesem Modell ist die Einfahrt nicht beliebig erlaubt, um einen regulären Ablauf zu gewährleisten. So können z. B. mindestens drei Züge pro Richtung in den Paß einfahren, aber damit ist für keinen der Züge eine Ausfahrt aus dem Paß möglich. Deshalb ist die Anzahl der Züge in einer Richtung auf zwei zu beschränken, falls in Gegenrichtung bereits mehr als zwei Züge eingefahren sind. Damit ist gewährleistet, daß immer ausreichend Ausweichgleise zur Verfügung stehen. Die dazu nötigen Erweiterungen des Modells sind in Abbildung 3.7 dargestellt, wobei zur besseren Übersicht die Stellen s_1, s_2, s_5 und s_6 weggelassen wurden.

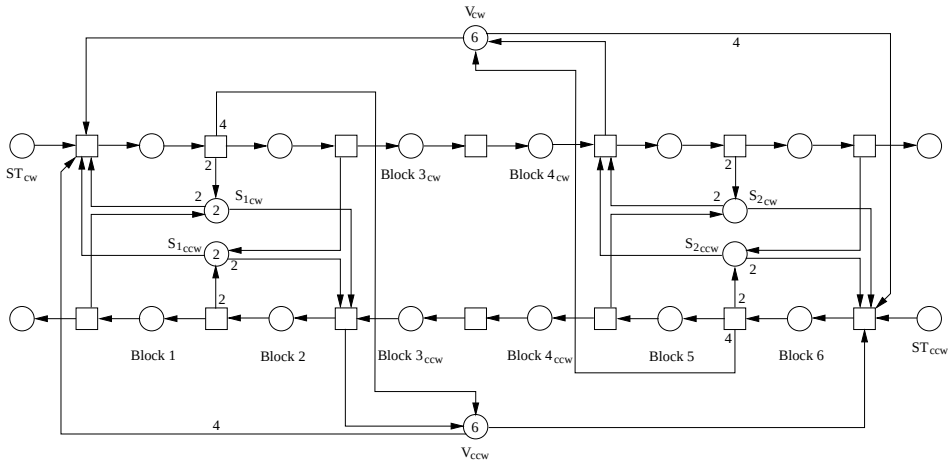


Abbildung 3.7: Deadlockfreies Modell der Paßstrecke mit Ausweichgleis

Die beiden Stellen V_{cw} und V_{ccw} werden initial mit sechs Token belegt. Die Einfahrt eines Zuges z. B. in Richtung "ccw" kann erfolgen, wenn von der Stelle V_{ccw} ein Token und von V_{cw} vier Token abgezogen werden können. Beim Verlassen des ersten Blocks werden wieder vier Token auf V_{cw} zurückgelegt, damit weitere Züge in der Richtung nachfolgen können. Damit mehr als vier Züge in einer Richtung fahren können, sofern kein Zug in Gegenrichtung wartet, muß beim Verlassen der mittleren Sektion ein Token auf die Stelle V_{ccw} zurückgelegt werden.

Bei den eingleisigen Sektionen muß wiederum verhindert werden, daß Züge gegenläufig in die Sektionen einfahren. Der Zugang zu einer Sektion wird über zwei Stellen kontrolliert, die beide initial mit zwei Token belegt sind. Ein Zug kann z. B. von ST_{cw} nur in die Sektion 1 einfahren, wenn von der Stelle $S_{1_{cw}}$ zwei und $S_{1_{ccw}}$ ein Token abgezogen werden kann. Damit ist für einen Zug in Gegenrichtung keine Einfahrt mehr möglich. Ein weiterer Zug kann nachfolgen, da beim Verlassen des ersten Blocks die zwei Token auf die Stelle $S_{1_{cw}}$ zurückgelegt werden. Beim Verlassen der Sektion wird ein Token auf die Stelle $S_{1_{ccw}}$ zurückgelegt, so daß gegebenenfalls ein Zug in Gegenrichtung einfahren kann.

In Bezug auf die Fairneßregelung und deren Durchbrechen gelten die gleichen Mechanismen, die in dem eingleisigen Netzmodell vorgestellt wurden. Dem Modell aus Abbildung 3.7 müssen somit noch analog die Stellen F_{cw} und F_{ccw} sowie die Transitionen T_{cw} und T_{ccw} hinzugefügt werden.

Kapitel 4

Hardware

Eine zentrale Steuerung, die einen geordneten Zugbetrieb ermöglichen soll, benötigt Daten über Zugbewegungen auf der Anlage und die Möglichkeit, Aktoren der Anlage beeinflussen zu können. Da die Steuerung durch eine Workstation realisiert werden soll, ist eine geeignete Anbindung der Sensoren und Aktoren der Modellbahnanlage notwendig.

Die Sensoren und Aktoren sind über die gesamte Fläche der Anlage verteilt, was bei Nutzung eines zentralen E/A-Knotens[1] zu einem beträchtlichen Verdrahtungsaufwand führt. Durch die Nutzung eines Feldbusses läßt sich dieser stark reduzieren.

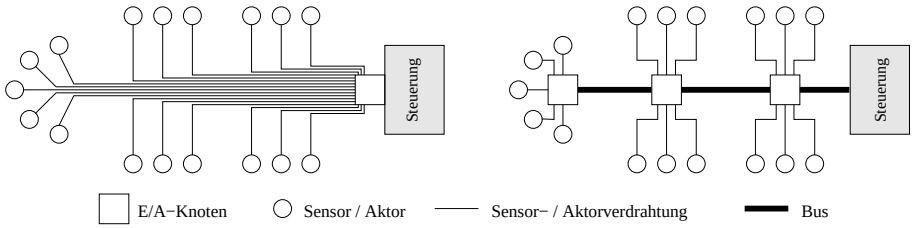


Abbildung 4.1: Schematischer Vergleich zentrale Verdrahtung ↔ Bussystem

Ein solches System besteht aus mehreren räumlich getrennten E/A-Knoten, die über einen Bus verbunden sind. Die Knoten untergliedern sich in aktive Teilnehmer (Master) und passive Teilnehmer (Slaves). Der oder die Master initiieren und kontrollieren die Kommunikation über den Bus und verfügen über eine Schnittstelle zu einem übergeordneten System, wie einer SPS[2] oder einer Workstation. Die Slaves hingegen können nur Daten empfangen oder nach Aufforderung senden und besitzen Ein-/Ausgänge, die mit Sensoren oder Aktoren verbunden sind.

In der industriellen Anwendung finden sich eine Reihe verschiedener Feldbustypen, die für unterschiedliche Einsatzbereiche konzipiert wurden (vgl. Anhang B). Neben dem reduzierten Verdrahtungsaufwand zeichnen sich diese Systeme durch

[1] Eingabe-/Ausgabe-Knoten
[2] Speicher-Programmierbare-Steuerung

- hohe Störsicherheit (verbreitete Feldbussysteme nutzen Alphabete mit Hammingabständen [3] zwischen 2 und 6),
- modulare Erweiterbarkeit,
- große Reichweite und
- Standardisierung

aus.

4.1 Sensoren und Aktoren

Die Zugbewegungen werden mit Hilfe von unter den Zügen befestigten Magneten und in den Gleisen eingelassenen Reedkontakten bestimmt. Jeder Bahnhofs- oder Streckenblock besitzt an beiden Enden im Abstand von ca. 15 cm zur Blocktrennung ein Reedkontaktpaar. Ein Zug ist jeweils unter der Lok und dem letzten Waggon mit einen Magneten ausgestattet. Beim Betätigen eines Reedkontaktes findet keine direkte Identifizierung des Zuges statt.

Die auf der Anlage anzusteuernden Aktoren sind elektronische Bauteile zur Steuerung der Fahrspannungen, Weichen und Lichtsignale[4]. Im Gegensatz zur Steuerung der Fahrspannungen und Weichen beeinflussen die Lichtsignale nicht den Fahrbetrieb, sondern dienen nur der visuellen Kontrolle.

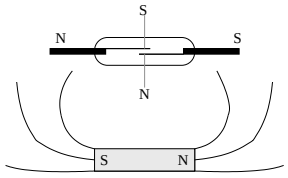


Abbildung 4.2: Reedkontakt: Das Magnetfeld des Dauermagneten setzt sich in dem Material innerhalb des Glaskolbens fort, die magnetische Kraft überwindet die Federkraft und führt so zum Schließen des Kontakts.

4.2 Interbus

Zum sicheren Betrieb der Modellbahnanlage ist die Einhaltung zeitlicher Schranken erforderlich. Deshalb kommt an der Modellbahnanlage mit dem Interbus (vgl. [1, 23, 12]) ein synchrones System zum Einsatz, für das die Dauer der Lese- und Schreibzyklen für eine feste Anzahl von Slaves konstant ist.
Der Interbus basiert auf einem aktiven Ring, d. h. jeder Teilnehmer im Ring sendet empfangene Daten aktiv an seinen Nachfolger weiter. Wie in Abbildung 4.3 dargestellt, gibt es genau einen Master und bis zu 512 Slaves, von denen der 'letzte' den Ring schließt. Abschlußwiderstände zur Vermeidung von Störungen (Reflektionen an Kabelenden) sind so überflüssig. Durch diesen Aufbau können Übertragungswege von mehreren Kilometern überbrückt und auftretende Fehler lokalisiert werden. Außerdem gibt es die Möglichkeit, weitere Ringe anzukoppeln, von der in diesem Projekt allerdings kein Gebrauch gemacht wird.

An der Modellbahn kommt ein Interbus UART[5] Master Protokoll-Chip zum Einsatz, der den Betrieb eines Interbussystems über eine serielle RS232 Schnittstelle (vgl. [22, 10]) ermöglicht. Um diesen Protokoll-Chip so kostengünstig wie möglich zu halten, wurden nur essentielle Funktionen in Hardware implementiert, Initialisierungsroutinen, die Aufbereitung der Nutzdaten zur Übertragung über den Interbus

[3] siehe [5]
[4] siehe Anhang A
[5] **U**niversal **A**synchronous **R**eceiver/**T**ransmitter

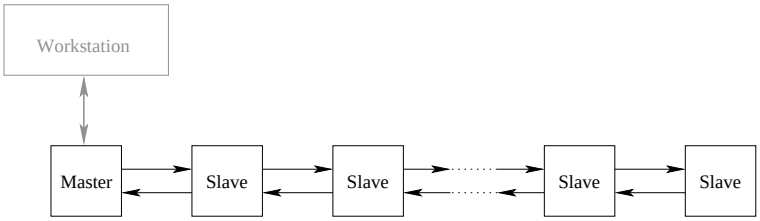


Abbildung 4.3: Topologie des Interbus, wie er an der Modellbahnanlage zum Einsatz kommt.

und Fehlerbehandlungsroutinen müssen deshalb in der ansteuernden Software realisiert werden. Dazu sind nähere Kenntnisse der Technik des Interbus notwendig.

4.2.1 Aufbau eines Interbus-Slaves

Der Interbus arbeitet ähnlich einem räumlich verteilten Schieberegister mit dem Master als zentrale Stelle, bei der Daten hinein- und herausgeschoben werden (siehe Abbildung 4.4). Dies ermöglicht eine Vollduplex-Übertragung und einen einfachen Aufbau der Busschnittstellen der Teilnehmer, da keine Adreßverwaltung, Kollisionserkennung o. ä. notwendig ist. Die Adressierung der Teilnehmer geschieht durch entsprechende Anordnung der zu übertragenden Daten.

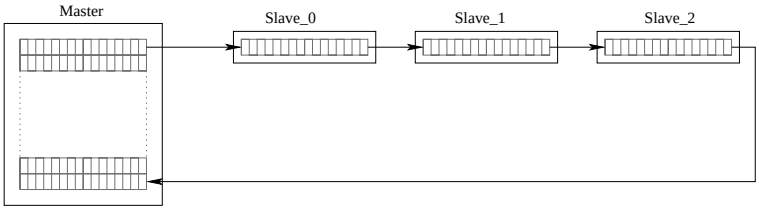


Abbildung 4.4: Der Interbus arbeitet ähnlich einem räumlich verteilten Schieberegister.

Ein Interbus-Slave besitzt vier Schieberegister, Logik zur Durchführung eines CRC[6] und Protokoll-Logik, die die Schnittstelle zum Bus bereitstellt.
Zur Übertragung der Nutzdaten kommen das Ein- und das Ausgangsregister zum Einsatz. Ihre Breite ist identisch und hängt von der Anzahl der Ein- und Ausgänge ab. Informationen darüber und zum Typ eines Slaves lassen sich aus dem Identifikationsregister auslesen, Steuerungsdaten werden hingegen in das Steuerregister geschrieben. Beide haben eine feste Breite von 16 Bit.

[6] Cyclic Redundancy Check, leicht in Hardware zu implementierende Fehlerüberprüfung

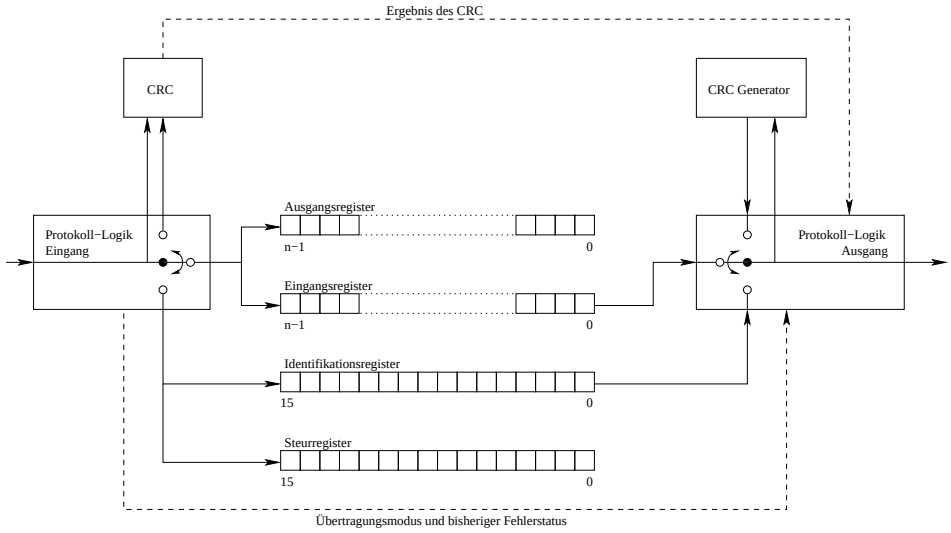


Abbildung 4.5: Schematischer Aufbau eines Interbus-Slaves

4.2.2 Datenübertragung

Das Interbus-Protokoll definiert Identifikations- und Datenzyklen. Beiden folgt immer eine CRC-Sequenz, die zum einen der Erkennung von Übertragungsfehlern dient und zum anderen den Slaves das Ende eines Zyklus signalisiert. Dieses führt zur Übernahme der Daten aus Steuer- und Eingangsregister und zur Aktualisierung der Daten im Identifikations- und Ausgangsregister. Jeder Zyklus wird vom Master initiiert.

Für den Identifikationszyklus werden das Identifikations- und das Steuerregister der Slaves genutzt. Der Master schiebt ein eindeutiges Loopback-Wort gefolgt von Steuerdaten in die Steuerregister der Slaves. Währenddessen empfängt der Master die aus den Identifikationsregistern herausgeschobenen Daten. Wird das Loopback-Wort wieder empfangen, ist der Zyklus vollständig und der Master kann die CRC-Sequenz einleiten. Die so gesammelten Daten sind zur Durchführung eines Datenzyklus notwendig, da sie Informationen über alle Teilnehmer am Bus enthalten.

Während eines Datenzyklus werden Nutzdaten die angeschlossene Peripherie betreffend übertragen. Es kommen dazu das Ein- und Ausgangsregister zum Einsatz. Ähnlich dem Identifikationszyklus schreibt der Master ein Loopback-Wort, gefolgt von den Ausgabedaten auf den Bus. Die Länge des Ausgabedatensatzes wurde zuvor mit einem Identifikationszyklus bestimmt, sodaß das Loopback-Wort nun genutzt werden kann, um die Anzahl der Teilnehmer am Bus zu überprüfen. Tritt dabei ein Fehler auf, muß der Master einen erneuten Identifikationszyklus einleiten und die gelesenen Daten verwerfen.

Die Übertragung der Daten zwischen den Slaves geschieht mit Hilfe von Status- und Datentelegrammen. Ein Datentelegramm besteht aus einem Header von 4 Bit, in denen ein Start-Bit, der aktuelle Fehlerstatus und Typ der Übertragung enthalten sind, 8 Bit Nutzdaten und einem Stop-Bit. Ein Statustelegramm ist identisch mit einem Datentelegramm, beinhaltet jedoch keine Nutzdaten und hat so nur eine Länge von 5 Bit. Diese Status- oder auch Idle-Telegramme werden vom Master gesendet, wenn keine Nutzdaten zu übertragen sind. Dadurch kann ein Slave jederzeit den Master über einen aufgetretenen Fehler informieren. Außerdem signalisiert das Ausbleiben von Telegrammen die Inaktivität des Masters oder die Unterbrechung

des Busses. In diesem Fall führt jeder betroffene Teilnehmer nach 25 ms einen internen Reset durch und überführt die an ihm angeschlossene Peripherie so in einen neutralen Zustand.

Die Headerinformationen der Telegramme werden von der Protokoll-Logik eines Slaves ausgewertet und zur internen Lenkung der Nutzdaten eingesetzt. Zur Generierung eines neuen Telegrammheaders am Ausgang eines Slaves wird der eintreffende Header und ggf. das Ergebnis des CRC herangezogen.
Da der Bus nicht über eine zusätzliche Leitung zur Übertragung eines Taktsignals verfügt, besitzt jeder Teilnehmer einen internen Taktgenerator. Diese werden mit Hilfe der Start- und Stop-Bits der übertragenen Telegramme synchronisiert. Da weiterhin jedes übertragene Bit in der Mitte seiner Übertragungszeit abgetastet wird, kann jeder Slave erst nach 1,5 Bitübertragungszeiten ein Telegramm an seinen Nachfolger weitersenden.

4.2.3 CRC-Sequenz

Während der Übertragung von Nutzdaten, generiert jeder Teilnehmer am Interbus für die ein- und ausgehenden Daten jeweils eine CRC-Prüfsumme mit einer Länge von 16 Bit. Wurden alle Nutzdaten übertragen, wird vom Master eine CRC-Sequenz initiiert. Dabei schickt jeder Teilnehmer seinem Nachfolger die Prüfsumme, über die von ihm verschickten Daten. Der Nachfolger kann diese dann mit seiner selbst generierten Prüfsumme der eingegangenen Daten vergleichen. Stimmen diese nicht überein, ist ein Fehler aufgetreten, der an alle anderen Teilnehmer propagiert wird. Anderenfalls werden die empfangenen Daten an die Peripherie weitergegeben und neue Daten von der Peripherie in das Eingangsregister eingelesen.

4.2.4 Ansteuerung des UART Master

Da der genutzte UART Master Protokoll-Chip nur Teile der Aufgaben eines Interbus-Masters übernimmt, muß der verbleibende Teil in Software realisiert werden. Diese Software wird auf der Workstation ausgeführt. Es bedarf deshalb einer Anbindung zwischen Workstation und Master, die über die serielle Schnittstelle RS232 hergestellt wird.

Jeder Kommunikationszyklus wird dabei von Seiten der Workstation initiiert und besteht immer aus einem an den Master gesendeten Befehl, gefolgt von einer Antwort des Masters. Sowohl Befehle, als auch Antworten werden in Form von den in Abbildung 4.6 dargestellten Summenrahmen übermittelt. Ein Befehls-Summenrahmen besteht aus folgenden Teilen:

1. Befehlscode (2 Byte), immer vorhanden
2. Parameterlänge (2 Byte, falls Nutzdaten folgen, sonst nicht vorhanden), wenn für den Befehl zusätzliche Daten notwendig sind, wird hier die Länge der folgenden Nutzdaten angegeben
3. Nutzdaten (Länge variabel, nicht immer vorhanden), beinhaltet z. B. Daten, die auf den Interbus geschrieben werden sollen oder Parameter zur Konfiguration des Masters

Ein Antwort-Summenrahmen des Masters beinhaltet:

1. Bestätigungscode für den Befehl (2 Byte), immer vorhanden
2. Nutzdaten (Länge variabel, ist anhand des gesendeten Befehls festgelegt)

3. Fehlercode (1 Byte), immer vorhanden

Es ist die Aufgabe der zu entwickelnden Steuerungssoftware, diese Summenrahmen zu generieren bzw. auszuwerten. Weiterhin muß dafür Sorge getragen werden, daß bestimmte Befehlsfolgen eingehalten werden, also z. B. nach einem Schreib-/ Lese-Zyklus immer ein CRC-Zyklus durchgeführt wird.

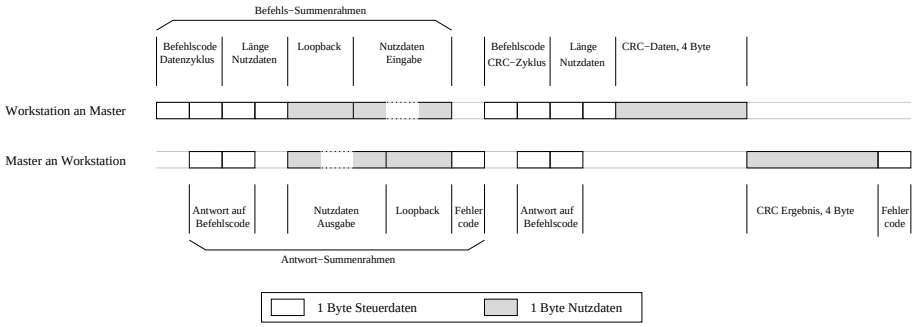


Abbildung 4.6: Kommunikation zwischen Workstation und Master bei einem Schreib-/Lesezyklus. (Verzögerungen durch Master oder Workstation wurden außer Acht gelassen.)

Die serielle Übertragung zwischen Master und Workstation geschieht dabei mit 1 Start-Bit, 8 Daten-Bits, geradem Parity-Bit und 2 Stop-Bits (vgl. [10]). Zur Übertragung eines Bytes Nutzdaten werden also zusätzlich 4 Bit Steuerinformationen benötigt. Es werden keine Handshaking-Leitungen genutzt.

4.2.5 Aufbau an der Modellbahnanlage

Für die Anwendung an der Modellbahnanlage wurde am Lehrstuhl ein Platinenlayout entwickelt, das zwei Interbus-Slaves auf einer Platine zusammenfaßt. Dabei besitzt der erste Slave jeweils 16 digitale Ausgänge, der zweite Slave je 8 digitale Ein- und Ausgänge. Die Gesamtlänge der Schieberegister einer solchen Platine beträgt also 24 Bit. Diese reicht aus, um über die zusätzlich auf der Platine integrierte Peripherie zwei Gleisabschnitte anzusteuern. Die Bedeutung der einzelnen Bits zeigen die Abbildungen 4.7 und 4.8.

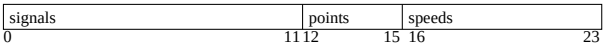


Abbildung 4.7: Ausgabedaten für eine Platine mit zwei Slaves.

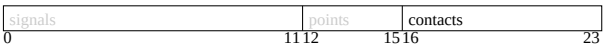


Abbildung 4.8: Eingabedaten für eine Platine mit zwei Slaves.

Bei insgesamt 47 Gleisabschnitten kommen so 24 Platinen zum Einsatz. Es ergibt sich also eine Gesamtnutzdatenlänge von 72 Byte. Den schematischen Aufbau der Hardware zeigt Abbildung 4.9.

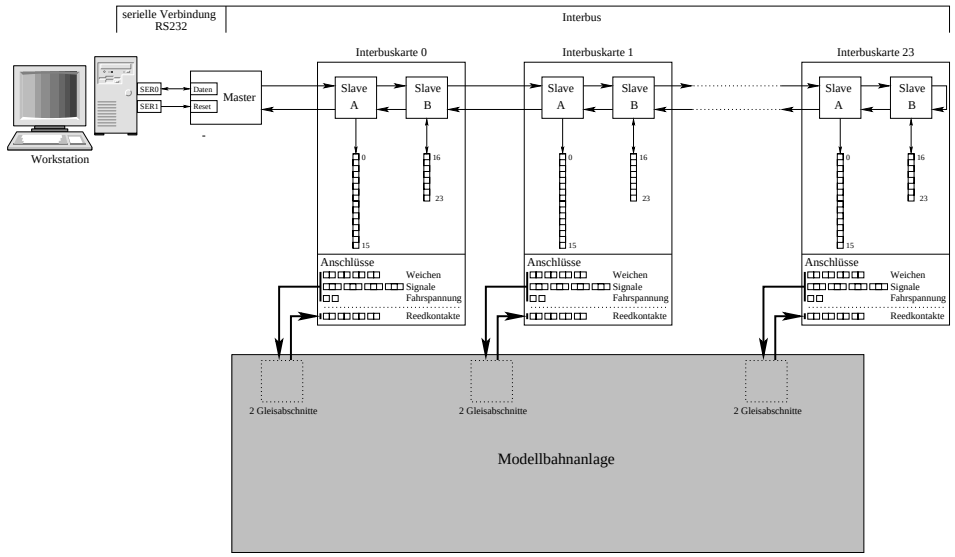


Abbildung 4.9: Schematischer Aufbau der Hardware zur Steuerung der Modellbahnanlage.

4.2.6 Übertragungszeiten

Die Dauer eines Schreib-/Lesezyklus setzt sich zusammen aus der Dauer der Bitübertragung, der Verzögerung durch Master und Slaves, der Verzögerung durch das Übertragungsmedium und der Laufzeit der ansteuernden Software.

$$T_{E/A\_Zyklus} = t_{Daten} + t_{Teilnehmer\_delay} + t_{Bus\_delay} + t_{Software}$$

Typischerweise wird der Interbus mit einer Datenrate von 500 $\frac{kBit}{s}$ betrieben. Die serielle Schnittstelle zwischen Master und Workstation wird hingegen mit einer deutlich niedrigeren Datenrate von z.B. 76,8 $\frac{kBit}{s}$ oder 153,6 $\frac{kBit}{s}$ betrieben. Die dadurch entstehenden Übertragungslücken auf dem Interbus werden vom Master mit Idle-Telegrammen aufgefüllt (siehe Abb. 4.10). Für die Bestimmung der Bitübertragungszeit ist also die Datenrate der UARTs maßgebend.

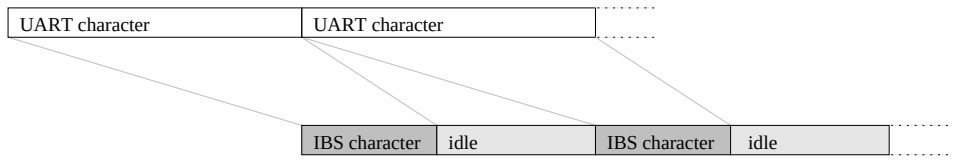


Abbildung 4.10: Übertragung eines Zeichens über die serielle Schnittstelle und den Interbus

Wie aus Abschnitt 4.2.4 und insbesondere der Abbildung 4.6 zu entnehmen ist, antwortet der Master auf einen Befehl bereits während die Workstation noch sendet. Dadurch sind in einem vollständigen Schreib-/Lesezyklus neben den Nutzdaten nur 20 Byte Steuerdaten zu berücksichtigen. Mit den in Abschnitt 4.2.4 genannten

Parametern der seriellen Schnittstelle ergibt sich als Dauer der Bitübertragung:

$$\begin{aligned} t_{Daten} &= (20+n) \cdot 12 t_{Bit\_UART} \\ \text{n} &: \text{Länge der Nutzdaten in Byte} \\ t_{Bit\_UART} &: \text{Dauer der Übertragung von einem Bit} \\ & \quad \text{über die serielle Schnittstelle} \end{aligned}$$

Die Verzögerung des UART Masters gibt der Hersteller mit 100 μs an. Da einer Datenübertragungssequenz eine CRC-Sequenz folgt, summiert sich dies auf 200 μs. Jeder Slave gibt, wie unter 4.2.2 beschrieben, ein Telegramm erst nach 1,5 Bitübertragungszeiten des Interbus an seinen Nachfolger weiter. Die resultierende Verzögerung hängt also von der Anzahl der Slaves ab. Für die durch die Teilnehmer verursachte Verzögerung ergibt sich

$$\begin{aligned} t_{Teilnehmer\_delay} &= 200\mu s + 2 \cdot 1,5 \cdot t_{Bit\_IBS} \\ t_{Bit\_IBS} &: \text{Dauer der Übertragung von einem Bit} \\ & \quad \text{über den Interbus} \end{aligned}$$

Die Signalverzögerung des Übertragungsmediums Kupfer liegt etwa bei 16 $\frac{\mu s}{km}$. Bei einer Busausdehnung von nur etwa 50 m folgt für die Anwendung an der Modellbahnanlage daraus eine Verzögerung von 0,8 μs, die vernachlässigt werden kann.

Die Dauer eines vollständigen Schreib-/Lese-Zyklus berechnet sich dann wie folgt:

$$\begin{aligned} T_{E/A\_Zyklus} &= t_{Daten} + t_{Teilnehmer\_delay} + t_{Bus\_delay} + t_{Software} \\ &= (20+n) \cdot 12 t_{Bit\_UART} + 200\mu s + 2 \cdot 1,5 \cdot t_{Bit\_IBS} + \\ &\quad l_{Bus} \cdot 16 \cdot 10^{-3} \frac{\mu s}{m} + t_{Software} \\ l_{Bus} &: \text{Länge des Busses in m} \end{aligned}$$

Für den Betrieb an der Modellbahnanlage mit 48 Slaves, einer Nutzdatenlänge von 72 Byte, einer Interbusdatenrate von 500 $\frac{kBit}{s}$ und einer UART Datenrate von 76,8 $\frac{kBit}{s}$ bzw. 153,6 $\frac{kBit}{s}$ ergibt sich rechnerisch

$$\begin{aligned} T_{E/A\_Zyklus\_76,8} &\approx 14,9ms + t_{Software} \\ T_{E/A\_Zyklus\_153,6} &\approx 7,7ms + t_{Software} \end{aligned}$$

Den weitaus größten Anteil daran hat die Dauer der seriellen Übertragung zwischen Workstation und Interbus Master mit ca. 14,4 ms bzw. 7,2 ms.

Bei einer durchschnittlichen Geschwindigkeit der Züge von 25 $\frac{cm}{s}$ legt ein Zug während dieser Zykluszeiten ohne Softwarelaufzeit 3,7 mm bzw. 1,9 mm zurück. Setzt man eine maximal zurückzulegende Strecke von 1 cm zwischen zwei Schreib-/Lese-Aktionen voraus, verbleiben für die Softwarelaufzeit 25,1 ms bzw. 32,3 ms.

4.3 Fehlerquellen

Tabelle 4.1 zeigt Meßergebnisse, die neben den Interbuslaufzeiten auch die Softwarelaufzeit beinhalten. Zum Einsatz kamen eine Sun SPARC Station 5 mit einer auf 110 MHz getakteten CPU und 128 MB Hauptspeicher und eine Sun UltraSPARC 10, die mit einer 440 MHz CPU und 256 MB Hauptspeicher ausgestattet war. Dabei wurde die serielle Schnittstelle der SPARC Station 5 mit 76,8 $\frac{kBit}{s}$ und die der UltraSPARC mit 153,6 $\frac{kBit}{s}$ betrieben. Die Angabe "Last" bedeutet in diesem Zusammenhang, daß während des Betriebes ein Benutzer am Rechner arbeitet, also z. B. einen Text editiert oder Internetseiten betrachtet. Dieses ist u. U. problematisch, da jede Tastatureingabe oder Mausbewegung einen Interrupt erzeugt, der die Programmabarbeitung unterbricht und so zu Verzögerungen führen kann. Diese Art der Last läßt sich kaum identisch reproduzieren, verdeutlicht aber die Abhängigkeit von Umgebungsparametern.

	SPARC Station 5 keine sonst. Last	SPARC Station 5 zusätzl. Last	UltraSPARC 10 keine sonst. Last	UltraSPARC 10 zusätzl. Last
minimal	80 ms	80 ms	9 ms	9 ms
durchschn.	85 ms	100 ms	9 ms	9 ms
maximal	14 0ms	300 ms	30 ms	230 ms

Tabelle 4.1: Gemessene Zykluszeiten des Interbus beim Einsatz verschiedener Workstations über eine Betriebsdauer von ca. 15 Minuten.

Das Variieren der Laufzeiten auch ohne Last ist auf das eingesetzte Multitasking Betriebssystem zurückzuführen. Dieses unterbricht die Ausführung des steuernden Programms je nach Bedarf anderer Prozesse und führt so zu nicht fest berechenbaren Verzögerungen. Die nahe beieinander liegenden Meßwerte der Minimal- und Durchschnittswerte deuten aber darauf hin, daß es sich bei den Maximalwerten nur um "Ausreißer" handelt, die verhältnismäßig selten auftreten.

Tabelle 4.1 zeigt, daß beim Einsatz der weniger leistungsfähigen SPARC Station 5 die Vorgabe einer maximal zurückzulegenden Strecke von 1 cm zwischen zwei Schreib-/Lese-Aktionen aufgrund der erhöhten Reaktionszeiten nicht eingehalten wird. Dies kann z. B. dazu führen, daß Fahrspannungen nicht rechtzeitig reduziert werden und Züge nicht wie gewünscht zum Stehen kommen.

Außerdem ist die Hardware der Interbus-Slaves so ausgelegt, daß jeweils die bei einem CRC-Zyklus an den Eingängen liegenden Zustände in das Ausgangsregister übernommen werden, um beim nächsten Datenzyklus versendet zu werden. Deshalb werden die oft nur wenige Millisekunden dauernden Kontaktbetätigungen mit Hilfe monostabiler FlipFlops für ca. 150 ms gehalten. Wird jedoch auch während dieser Zeitspanne der Zustand der Eingänge nicht in das Register übernommen, können Kontaktmeldungen verloren gehen. Deshalb ist es wichtig, Schreib-/Lesezykluszeiten von deutlich weniger als 150 ms zu erreichen.

Trotz sorgfältigen Aufbaus der Hardware ist es in einem komplexen System wie der Modellbahnanlage nicht zu vermeiden, daß auch Hardwarefehler im laufenden Betrieb auftreten. Diese können sowohl durch Fehlverhalten einzelner Bauteile, als auch durch äußere Einflüsse auftreten.

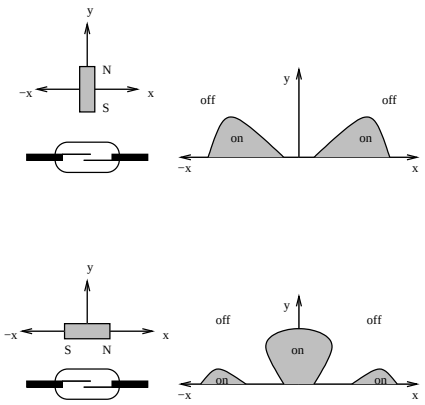


Abbildung 4.11: Das Auslösen eines Reedkontaktes hängt von der Lage und Position des Dauermagneten ab.

So stellte sich im Laufe der Entwicklung heraus, daß auch durch die doppelte Auslegung der Reedkontakte keine hinreichende Sicherheit der Kontaktmeldungen zu erreichen ist. Es werden sowohl Kontakte beim Überfahren nicht betätigt, als auch nicht vorhandene Betätigungen gemeldet. Letzteres ließ sich durch Messungen auf Weichenbetätigungen zurückführen. Die hierbei auftretenden Schaltströme sind offenbar groß genug, um Spannungsspitzen in die Meßleitungen der Reedkontakte zu induzieren, die von der auswertenden Elektronik als Betätigung des entsprechenden Kontakts interpretiert werden. Trotz einer entsprechenden Verlegung der Leitungen ist dies weiterhin nicht auszuschließen. Ein weiteres Problem stellt die Schaltcharakteristik von Reedkontakten dar. Wie in Abbildung 4.11 zu erkennen, löst ein Reedkontakt beim Vorbeistreichen eines Dauermagneten mehrfach aus. Dies kann trotz der eingesetzten FlipFlops bei langsamer Fahrt eines Zuges zu einer doppelten Kontaktmeldung führen. Weiterhin sind mechanische Probleme, besonders im Hinblick auf die Züge, nicht auszuschließen.

Diese Fehlerquellen müssen bei der Erstellung der Software berücksichtigt und entsprechende Reaktionen vorgesehen werden.

Kapitel 5

Software

5.1 Aufgabenstellung

Die im Rahmen dieser Arbeit erstellte Software implementiert folgende Funktionen:

- Ansteuerung der in Kapitel 4 vorgestellten Hardware:
  Es ist notwendig, über die serielle Schnittstelle der Workstation den Interbus UART Master anzusteuern und Methoden zur Initialisierung, Steuerung und Datenübertragung zu entwickeln.
- Steuerung des Zugbetriebs unter den in Kapitel 3 vorgestellten Gesichtspunkten:
  Es muß eine Steuerung entwickelt werden, die Fahrpläne der Züge verwaltet und abarbeitet. Dies muß zu einem verklemmungsfreien und fairen Zugbetrieb führen, der mit Hilfe der Software zur Hardwareansteuerung umgesetzt wird.
- Bereitstellung einer Benutzerschnittstelle:
  Dem Benutzer soll es möglich sein, Fahrpläne zu erstellen, Parameter des Zugbetriebs zu beeinflussen und während der Laufzeit den Fortschritt und Fehlermeldungen zu überwachen. Dazu wird auf die Software zur Zugsteuerung und Hardwareansteuerung zurückgegriffen.

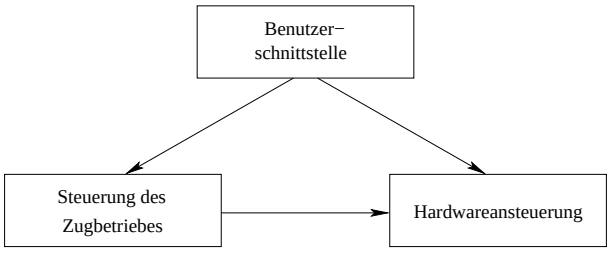


Abbildung 5.1: Gliederung der Software und die Abhängigkeit der Teile voneinander.

Zur Ansteuerung der Hardware bietet sich die Programmiersprache C an (vgl. [20]), da es eine große Anzahl von Bibliotheken gibt, die den Zugriff auf die Hardware ermöglichen. Bei dem Umfang des zu realisierenden Projektes ist es notwendig, eine Methodik zur Gliederung und Modularisierung des Quellcodes zu nutzen. Eine Modularisierung ist insbesondere auch nützlich, um mit mehreren Entwicklern an einem Projekt arbeiten zu können. Aus diesen Gründen wurde C++ als eine objektorientierte Sprache gewählt (vgl. [21, 13]), die zudem auch von Werkzeugen wie "automake" (vgl. [25]) und "autoconf" (vgl. [24]) unterstützt wird.

Zur Versionskontrolle kommt CVS[1] (vgl. [11, 17]) zum Einsatz. Dieses System verwaltet den Sourcecode mehrerer Entwickler, führt diesen zu einem Projektabbild zusammen und ermöglicht den Zugriff auf frühere Versionen der verwalteten Dateien.

5.2 Threads

5.2.1 Konzept

Die unter 5.1 beschriebenen Aufgaben müssen alle während der Laufzeit wahrgenommen werden. Deshalb bietet sich die Aufteilung auf mehrere Prozesse an. Dies führt zu einer zumindest scheinbar nebenläufigen Ausführung, erfordert aber auf Grund der Abhängigkeit der Aufgaben untereinander ein Kommunikationskonzept zur Synchronisation der Prozesse (vgl. [2, 15, 3]).

Bei der Betrachtung der zur Ausführung der Prozesse notwendigen Daten wird deutlich, daß es diesbezüglich eine Reihe von Überschneidungen gibt. So muß z. B. der Prozeß zur Realisierung der Benutzerschnittstelle auf Fahrpläne und andere Zugparameter zugreifen können, die aber auch vom Prozeß zur Steuerung des Zugbetriebs genutzt werden. Deshalb kommen in dieser Arbeit sog. Threads (vgl. [7]) zum Einsatz. Dies sind Prozesse, die während ihrer Ausführung auf den gleichen Speicherbereich zugreifen wie ihr Vaterprozeß. Damit ist die gemeinsame Nutzung von Datenstrukturen möglich, es werden aber Mechanismen benötigt, um diese konsistent zu halten.

In Anlehnung an die in 5.1 dargestellte Gliederung der Software werden vier Threads implementiert:

1. **Hardware-Thread:** Dieser Thread realisiert das Senden zur bzw. das Empfangen der Daten von der Anlage.
2. **Steuerungs-Thread:** Dieser Thread überwacht und beeinflußt die Zugbewegungen. Bei der Initialisierung dieses Threads muß der Hardware Thread bereits laufen.
3. **Feedback-Thread:** Dieser Thread stellt alle Informationen zum Zustand der Anlage dem Benutzer zur Verfügung und aktualisiert diese fortwährend.
4. **Main-Thread:** Dies ist der Vater-Thread aller obigen Threads. Er nimmt während der Programmlaufzeit Benutzereingaben entgegen und führt entsprechende Anpassungen von Datenstrukturen durch.

5.2.2 Implementierung

Für das genutzte Solaris Betriebssystem stehen eine Reihe verschiedener Threadbibliotheken zur Verfügung, die grundsätzlich ähnliche Lösungsansätze haben. In diesem Projekt kommen zwei verschiedene dieser Bibliotheken zum Einsatz, die jeweils POSIX[2](vgl. [7]) und Qt Threads (vgl. [8]) implementieren. Die POSIX-Threads werden vorwiegend in C und C++ Programmen angewendet. Sie kommen in diesem Projekt im Bereich der Hardwareansteuerung und Steuerung

[1] Concurrent Version System

[2] Portable Operating System Interface, die POSIX-Standards wurden vom Portable Applications Standards Committee (PASC, vgl. [9]), einer Vereinigung der IEEE Computer Society, entwickelt.

des Zugbetriebs zum Einsatz. Die Hinzunahme der Qt Threads ist wegen der Nutzung der gleichnamigen Bibliothek zur Erstellung einer grafischen Benutzeroberfläche notwendig, um z. B. das Entgegennehmen von Benutzereingaben und die Aktualisierung des auf der Oberfläche dargestellten Datenbestandes in zwei eigenständigen Threads zu ermöglichen. Beide Standards verfügen über einen Mutex Mechanismus, mit dessen Hilfe der wechselseitige Zugriff mehrerer Threads auf die selben Datenstrukturen realisiert werden kann.

5.3 Objektrepräsentationen

Um den Zugbetrieb in dem Steuerprogramm nachhalten zu können, werden Objekte benötigt, die den Zustand der Anlage beschreiben. Zur Repräsentation der Topologie des Streckenlayouts ist eine Beschreibung der Gleisabschnitte, deren Verknüpfungen untereinander und der dabei ggf. genutzten Weichen erforderlich.

Außerdem ist die Darstellung der Züge mit zugehörigen Fahrplänen und weiteren Parametern notwendig. Letztere beziehen sich auf die Abarbeitung der Fahrpläne und werden während der Laufzeit angepaßt.

5.3.1 Topologie der Modellbahnanlage

Die das Streckenlayout enthaltende Struktur wird beim Programmstart angelegt. Dafür wird eine Konfigurationsdatei verwendet, die alle benötigten Daten enthält. So ist es möglich, eine Anpassung an ein geändertes Streckenlayout durchzuführen, ohne Veränderungen im Quellcode vorzunehmen. Das kann vor allem dann nützlich sein, wenn die Verdrahtung der Anlage und damit z. B. die Nummerierung der Weichen geändert wird.

Die so generierte Datenstruktur besteht aus drei Typen von Unterstrukturen,

- der `point_link` Struktur,
- der `block_link` Struktur und
- der `block_status` Struktur,

die im Folgenden erläutert werden. Die `block_status` Struktur beschreibt dabei mit Hilfe der `point_link` und der `block_link` Strukturen einen Gleisabschnitt und dessen Integration in das Streckennetz.

Die `point_link` Struktur besteht lediglich aus zwei Komponenten,

- einer Weichennummer und
- der zugehörigen Stellung.

Sie beschreibt die Stellung einer Weiche, die das Erreichen eines Nachfolgeblocks ermöglicht.

Die `block_link` Struktur stellt die Verbindung zu einem Folgegleis dar. Sie enthält damit alle nötigen Parameter, die für das Einfahren eines Zuges aus einem Block in einen Folgeblock nötig sind. Dazu gehören

- die Nummer des Nachfolgeblocks,
- die mögliche Einfahrtsrichtung (cw oder ccw),

- die Fahrtrichtung auf dem Nachfolgeblock selbst (cw oder ccw),
- die Nummern von ggf. zu befahrenden Weichenblöcken (maximal zwei),
- die Fahrtrichtung in diesen Weichenblöcken (cw oder ccw) und
- ein Array von `point_link` Strukturen, das die benötigten Weichenstellungen beschreibt (maximal fünf).

Anzumerken ist, daß eine Fahrtrichtungsangabe auf den Weichenblöcken nicht notwendig ist, da eine Fahrtrichtungsänderung bei dem vorliegenden Streckenlayout nur innerhalb der Wendeschleifen stattfindet, und damit immer unabhängig von Weichenblöcken ist. Bei der Implementierung haben sich diese Parameter jedoch als nützlich erwiesen.

Eingebunden wird diese Struktur in die Beschreibung eines Gleisabschnitts mit Hilfe der `block_status` Struktur. Diese enthält als Parameter

- die Nummer des Blocks,
- den Namen des Blocks,
- die Gruppenzugehörigkeit,
- die mögliche Fahrtrichtung (cw, ccw oder beides),
- den Geschwindigkeits-Offset,
- die Nachfolgegleise und
- Angaben über den/die verwendeten Signaltyp(en).

Die Nummern der Blöcke werden eindeutig und aufsteigend vergeben. Die Gruppenzugehörigkeit eines Blocks sagt aus, ob er zu einem Bahnhof – und wenn ja zu welchem – gehört, oder ob es sich um ein Streckengleis handelt. Diese Kennung ist für die Suche eines freien Bahnhofsgleises notwendig, für den Fall, daß das ursprünglich im Fahrplan enthaltene Bahnhofsgleis belegt ist.

Der Geschwindigkeits-Offset bietet die Möglichkeit, auf Strecken mit Gefälle oder Steigung oder auf Blöcken, deren Reedkontaktabstand nur geringfügig größer als eine Zuglänge ist, die Geschwindigkeit der Züge anzupassen.
Weiterhin sind in der `block_status` Struktur als Parameter

- der Belegungszustand und
- die aktuelle Geschwindigkeit

enthalten. Diese werden während des Fahrbetriebs genutzt, um Gleise für bestimmte Züge zu reservieren und deren aktuell gefahrene Geschwindigkeit abzubilden.

Die nachfolgende Abbildung 5.2 zeigt exemplarisch die Zustandsbeschreibung eines Blocks. Die Nachfolgegleise werden dargestellt durch ein Array von `block_link` Strukturen. Diese enthalten die zur Einfahrt eines Zuges benötigten Daten in einen Nachfolgeblock. Das sind beim gewählten Streckenlayout maximal acht: Die Gleise `KH_ICOC_0` und `KH_ICOC_1` sind bidirektional zu befahren und haben so als Nachfolgegleise die Blöcke des Outer- bzw. Inner-Circle und des Paßbahnhofs.

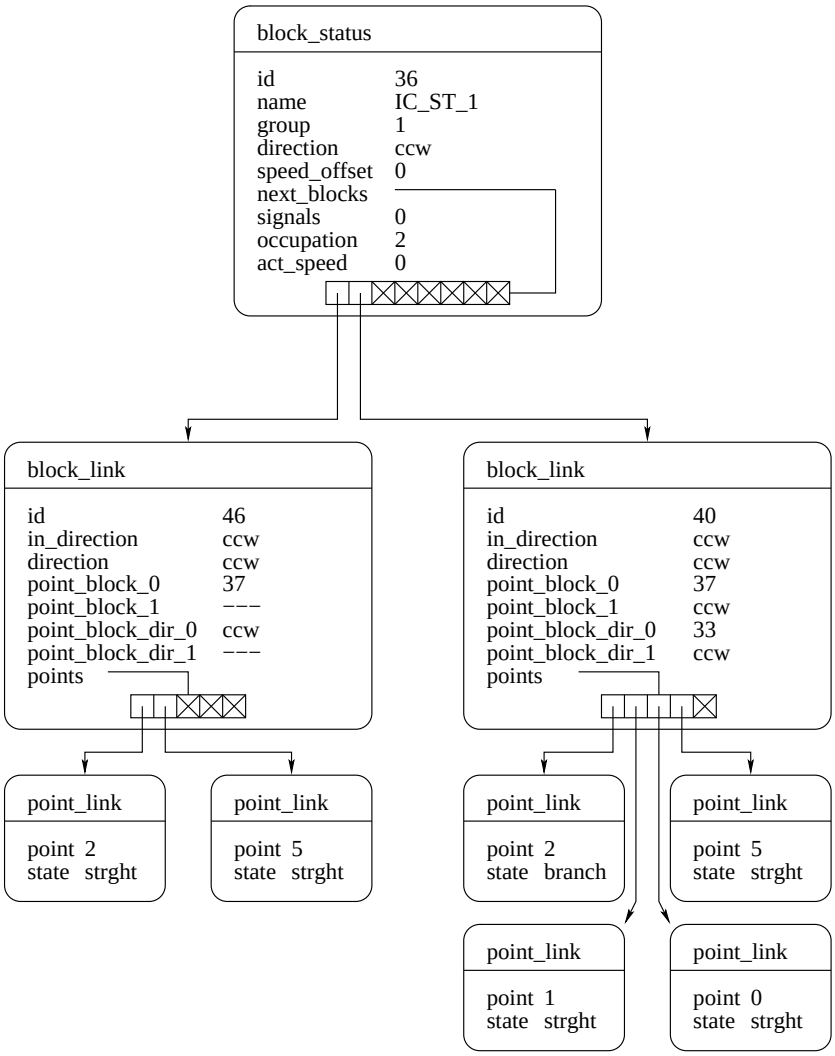


Abbildung 5.2: Beispiel einer `block_status` Struktur

Das gesamte Streckennetz wird dann durch ein Array von insgesamt 47 `block_status` Strukturen abgebildet. Diese sind entsprechend den Blocknummern aufsteigend angeordnet. Dadurch wird es möglich, anhand einer Blocknummer direkt die entsprechende Beschreibung innerhalb des Arrays auszuwählen.

5.3.2 Darstellung eines Zuges

Die zur Darstellung eines Zuges genutzte Struktur enthält folgende Einträge:

- die Zugnummer,
- das Heimatgleis,
- die Richtung, mit der das Heimatgleis verlassen wird,
- die Mindest- und Maximalgeschwindigkeit,
- der aktuelle Fahrplan und
- die Anzahl der insgesamt zu befahrenden Blöcke.

Da die Züge in dem Steuerprogramm über ihre Zugnummer identifiziert werden, und das Nachhalten der Zugpositionen ausgehend von den Heimatbahnhöfen erfolgt, sind diese eindeutig zu wählen. Die Mindest- und Maximalgeschwindigkeiten legen die Fahrstufen beim Heranfahren an ein Haltesignal bzw. bei freier Fahrt fest und können während des Betriebs angepaßt werden. Der Fahrplan enthält eine Liste der abzufahrenden Blöcke. Die Gesamtzahl aller zu befahrenden Blöcke wird für die Berechnung der Fortschrittsanzeige benötigt.

Weiterhin gibt es eine Reihe von Parametern, die zur Laufzeit angepaßt werden und damit den aktuellen Zustand des Zuges widergeben. Dazu gehören

- die Priorität,
- die Anzahl der noch zu befahrenden Blöcke,
- Zähler, die die Reedkontaktmeldungen aufnehmen,
- ein Flag, das neu eingetroffene Reedkontaktmeldungen anzeigt,
- eine Zeitmarke, die beim Eintreffen einer Reedkontaktmeldung auf die aktuelle Systemzeit gesetzt wird und
- ein Status Flag, das den Zugbetriebsstatus `RUNNING`, `WAITING`, `STOPPED` oder `READY` enthält.

Die Priorität eines Zuges wird zur Auflösung von Konfliktsituationen herangezogen. Die Anzahl der noch zu befahrenden Blöcke wird benötigt, um bei nahezu beendetem Fahrplan die Strecke bis zum Heimatgleis freizuhalten. Die Streckenführung ist so gestaltet, daß ein Zug maximal zwei mit Reedkontakten versehene Gleisabschnitte belegen kann. Für den aktuellen Block und den Nachfolgeblock werden somit zwei Zähler benötigt, die die jeweilige Anzahl der Kontaktmeldungen aufnehmen. Jede Kontaktmeldung wird mit einem Zeitstempel versehen. Tritt bei einem fahrenden Zug innerhalb eines festgelegten Intervalls keine neue Kontaktmeldung mehr auf, kann davon ausgegangen werden, daß ein schwerwiegender Fehler aufgetreten und der Betrieb aus Sicherheitsgründen umgehend einzustellen ist.

Zur Veranschaulichung sei folgendes Beispiel in Tabelle 5.1 aufgezeigt, in dem ein Zug auf dem Gleis OC_ST_1 vor dem Signal steht und in den Nachfolgeblock OC_LN_0 ausfährt. Die Abkürzung "Rkz" steht für Reedkontaktzähler.

Rkz aktuell	Rkz Folgeblock	aktueller Block	Folgeblock	Aktion
3	0	OC_ST_1	OC_LN_0	Zug fährt aus dem Gleis aus
3	1	OC_ST_1	OC_LN_0	Zug überfährt ersten Kontakt von OC_LN_0 mit der Lok
4	1	OC_ST_1	OC_LN_0	Zug überfährt letzten Kontakt von OC_ST_1 mit dem Hänger
4	2	OC_ST_1	OC_LN_0	Zug überfährt ersten Kontakt von OC_LN_0 mit dem Hänger, OC_ST_1 wird freigegeben, aktueller und nächster Block angepaßt
2	0	OC_LN_0	OC_LN_1	Überprüfung, ob nächster Block reserviert werden kann, evtl. Geschwindigkeitsreduzierung
3	0	OC_LN_0	OC_LN_1	Halt vor dem Signal oder normales Überfahren

Tabelle 5.1: Beispiel zur Belegung der Reedkontaktzähler

5.3.3 Darstellung eines Fahrplans

In der Fahrplanstruktur sind folgende Komponenten enthalten

- eine Blocknummer,
- die auf dem Block gefahrene Richtung und
- ein Zeiger auf ein weiteres Fahrplanelement.

Eine einfach verkettete Liste aus mehreren Strukturelementen bildet einen Fahrplan für einen Zug. Dabei werden der Reihe nach alle zu befahrenden Blöcke angegeben, zusätzlich die Fahrtrichtung auf jedem Block. Die Richtung wird im Fall eines Benutzerabbruchs genutzt, um für jeden Zug, ausgehend von seiner aktuellen Position, einen neuen Fahrplan bis zu seinem Heimatbahnhof zu bestimmen.

Während des Ablaufs wird beim Verlassen eines Blocks das aktuelle Kopfelement gelöscht. Ein Zug hat seine Ausgangsposition im fehlerfreien Betrieb wieder erreicht, wenn kein weiteres Element mehr in der Liste vorhanden ist.

5.4 Fehlermeldung

Tritt während der Ausführung einer Methode ein Fehler auf, so wird dieser der aufrufenden Methode mit Hilfe des Rückgabewertes mitgeteilt. Diese Rückgabewerte sind Fehlercodes, die global definiert sind und Aufschluß über die Fehlerursache geben. Dieses Konzept beruht auf der Annahme, daß, auch bei tiefer Verschachtelung der Methodenaufrufe, zumindest eine der aufrufenden Methoden geeignete Maßnahmen zur Fehlerbehandlung einleiten kann.

5.5 Steuerung der Hardware

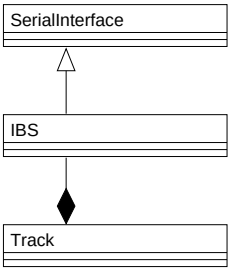


Abbildung 5.3: Darstellung der hardwarebezogenen Klassen als UML[4]Diagramm

Die Ansteuerung der Hardware untergliedert sich in drei Ebenen, die jeweils aufeinander aufbauen. Die unterste Ebene bildet die Steuerung der seriellen Schnittstelle zur Übertragung von Daten zwischen Interbus Master und Workstation. Darauf baut die Ebene der Interbussteuerung auf, die grundlegende Verfahren zur Initialisierung des Interbus und Übertragung von Daten an die Slaves realisiert. Die oberste Ebene stellt Methoden bereit, die die Ansteuerung einzelner Weichen, Gleise und Signale ermöglichen und für die Erfassung der Reedkontaktmeldungen sorgen. Jede dieser Ebenen wird in einer Klasse implementiert, deren Namen und Beziehungen in Abbildung 5.3 dargestellt sind. Damit ergibt sich das in Abbildung 5.4 dargestellte Schichtenmodell, bei dem jede Schicht jeweils einen Dienstleister für die darüberliegende Schicht darstellt, bzw. jede Schicht ein Dienstnehmer der darunterliegenden Schicht ist. Schichten, die im Modell auf der selben Ebene liegen, kommunizieren über die ihnen untergeordneten Dienstleister miteinander. Dies wird durch den eingezeichneten Datenübertragungspfad widergegeben. Es ist anzumerken, daß jede Kommunikation von Seiten der Steuerung initiiert wird.

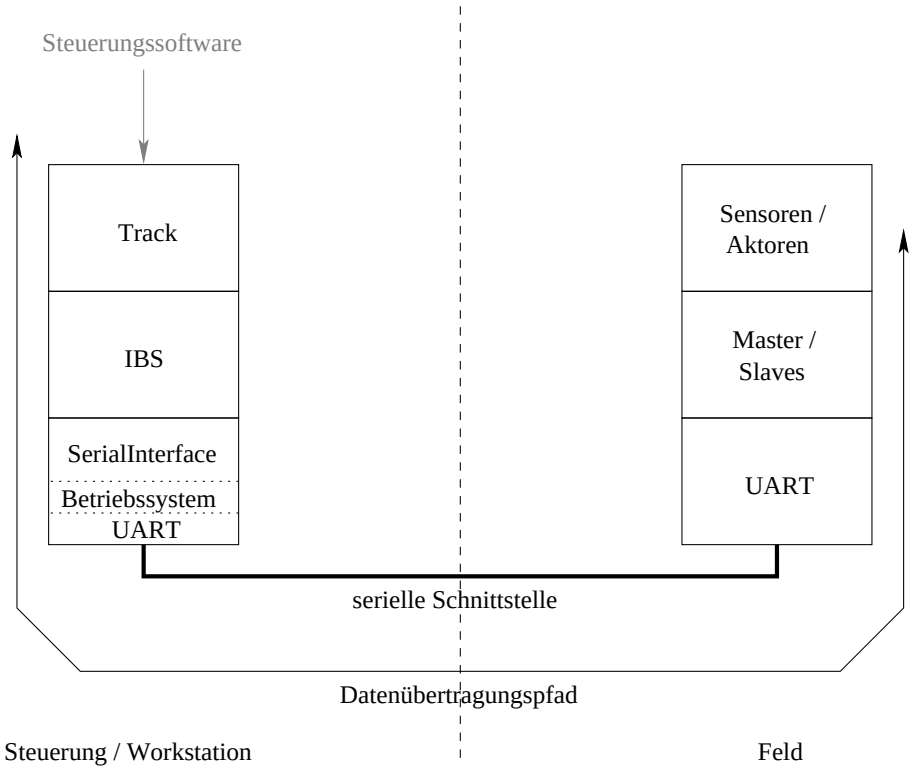


Abbildung 5.4: Schichtenmodell zur Datenübertragung

[4]Unified Modelling Language, vgl. [14]

5.5.1 Serielle Schnittstelle

Die Klasse `SerialInterface` kapselt das Öffnen und Schließen der Schnittstelle, Setzen der Parameter, sowie Schreib- und Lesezugriffe auf die Schnittstelle. Es wird dabei nicht direkt auf die Hardware, sondern auf den Treiber des Betriebssystems zugegriffen. Die Initialisierung des Schnittstellentreibers geschieht mit den Parametern

- Lesen und Schreiben
- kein Handshaking
- Lesen einzelner Bytes
- nicht blockierend.

Das Handshaking ist zu deaktivieren, da der Schnittstellentreiber des Betriebssystems die nicht vorhandenen Handshaking-Leitungen mißinterpretieren kann. Dies führt zu einem unerwarteten Verhalten bzw. zu Fehlermeldungen.

Der Schnittstellentreiber des Betriebssystems verfügt über zwei FIFO-Puffer, in dem empfangene und zu sendende Daten zwischengespeichert werden. Das Schreiben von Daten entspricht also einem Kopieren der Daten in den Schreibpuffer, dessen Inhalt, gesteuert vom Betriebssystem, nach und nach auf die Hardwareschnittstelle geschrieben wird. Auf der Hardwareschnittstelle empfangene Daten werden vom Schnittstellentreiber im Lesepuffer zwischengespeichert, bis sie durch einen lesenden Zugriff auf den Schnittstellentreiber abgerufen werden. Dadurch ist es möglich, einen Summenrahmen vollständig zu schreiben, bevor die vom Master zurückgesendeten Daten gelesen werden. Weiterhin ist der Schnittstellentreiber so zu konfigurieren, daß auch einzelne Bytes gelesen werden können, da die Länge der Summenrahmen variiert.

Für den Fall, daß aus irgendwelchen Gründen die vom Master zurückgelieferten oder empfangenen Daten unvollständig sind, ist es notwendig, den Zyklus abzubrechen. Dies geschieht mit Hilfe eines Timeout Mechanismus auf Seiten der Workstation. Damit dieser greifen kann, darf das Lesen von Daten nicht blockieren, falls keine weiteren Daten mehr empfangen werden.

5.5.2 Interbus-Steuerung

Die Klasse `IBS` implementiert den Datentransfer zwischen Interbus und Software auf der Workstation. Es werden Methoden zur Initialisierung, zum Rücksetzen des Interbus, zum Senden und Empfangen von Datensätzen bereitgestellt. Alle diese Methoden erstellen dazu Summenrahmen nach dem in Abschnitt 4.2.4 dargestellten Schema, die mit Hilfe der Klasse `SerialInterface` an den Master gesendet werden. Die Initialisierung des Interbus erfolgt in folgenden Schritten:

- Öffnen der seriellen Schnittstelle mit einer Übertragungsrate von 1200 Bit/s. Das ist die Standardeinstellung des Masters und und ermöglicht bereits eine uneingeschränkte Kommunikation. Die geringe Übertragungsrate resultiert aber in langen Übertragungszyklen, die die mögliche Reaktionszeit des Systems zu groß werden lassen.
- Übertragung eines Teilers zur Bestimmung der gewünschten Betriebsgeschwindigkeit. Der Master verfügt über einen Quarz, der mit 16 MHz schwingt. Der Teiler gibt an, mit welchem Bruchteil dieser Frequenz der UART betrieben werden soll. Die Übertragungsrate auf Seiten des Masters ist damit nahezu frei wählbar. Die serielle Schnittstelle der Workstation läßt sich hingegen nur mit wenigen standardisierten Übertragungsraten betreiben.

- Schließen und erneutes Öffnen der seriellen Schnittstelle mit der neu eingestellten Übertragungsrate.
- Suche der angeschlossenen Slaves: Dies geschieht mit Hilfe der im Kapitel 4 vorgestellten Identifikationszyklen. Da das zwischen Master und Workstation genutzte Summenrahmenprotokoll immer die Angabe einer Nutzdatenlänge erfordert, wird der Nutzdatensatz so groß gewählt, daß er für die Daten aller maximal zu erwartenden Slaves ausreichend Platz bietet. Überschüssige Bytes werden vom Master mit Nullen belegt. Die ausgelesenen Daten enthalten Informationen zu den angeschlossenen Slaves.

Ist die Initialisierung abgeschlossen, können Datenzyklen durchgeführt werden. Dazu bietet die Klasse `IBS` eine Methode, die als Parameter zwei Zeiger auf Bytearrays übernimmt. Der Inhalt des ersten Bytearrays wird als Nutzdatensatz an die Interbus-Slaves gesendet. Nach jedem Datenzyklus wird eine CRC-Sequenz durchgeführt, die empfangenen Fehlercodes und das CRC-Ergebnis werden ausgewertet. Treten keine Fehler auf, werden die empfangenen Nutzdaten im zweiten Bytearray abgelegt, ansonsten wird ein entsprechender Fehlercode an die aufrufende Methode zurückgegeben.

Treten Übertragungsfehler auf der seriellen Schnittstelle zwischen Master und Workstation auf, bedeutet dies meist den Verlust einer unbekannten Anzahl von Bytes, was dazu führt, daß der aktuelle Zustand des Masters nicht mehr bekannt ist. Deshalb schafft in diesem Falle nur ein vollständiger Reset des Masters und damit auch aller Slaves und deren Ausgänge Abhilfe. Die Fahrspannungen aller Gleisabschnitte werden dabei auf Null gesetzt und alle Weichen auf gerade geschaltet. Beim Auslaufen der Züge auftretende Reedkontaktbetätigungen werden nicht erfaßt, außerdem sind Entgleisungen in Weichenbereichen möglich. Es ist deshalb nicht sinnvoll, den unterbrochenen Zugbetrieb nach einer erneuten Initialisierung des Interbus unmittelbar fortzuführen.

5.5.3 Softwareschnittstelle

Die Klasse `Track` bildet die dritte Ebene der Hardwareansteuerung und stellt in Bezug auf das Schichtenmodell in Abbildung 5.4 den Dienstleister für die Steuerungssoftware dar. Es wird, neben Methoden zur Ansteuerung der Aktoren und zum Auslesen der Sensordaten, auch die in Abschnitt 5.3.1 beschriebene Datenstruktur zur Beschreibung der Topologie der Anlage implementiert. Ziel ist das weitgehende Verbergen der zur Datenübertragung genutzten Hardware (serielle Schnittstelle und Interbus), sodaß der Aufruf einer Methode der Klasse `Track` ohne Kenntnisse über diese Hardware erfolgen kann.

5.5.3.1 Adressierung von Sensoren und Aktoren

Die Adressierung von Gleisabschnitten und Weichen erfolgt über fortlaufende Identifikationsnummern, die zwischen 0 und 47 bzw. 0 und 27 liegen. Da Signale und Reedkontakte in jedem Fall eindeutig einem Gleis zuzuordnen sind, werden diese mit der Gleisnummer und einem Index von 0 bis 1 bzw. 0 bis 3 adressiert. Diese Indizes orientieren sich aus Gründen der einfacheren Umsetzung in Bitadressen an der Reihenfolge der entsprechenden Bits im Nutzdatensatz. Dies gilt analog für die Nummerierung der Gleisabschnitte.

Bei der Nummerierung der Weichen ist diese Umrechnung nicht sinnvoll, da nicht zu jedem Gleis zwei Weichen gehören und die jeweilige Zuordnung nicht eindeutig ist. Die entsprechenden Bits sind also im Nutzdatensatz in unregelmäßigen

Abständen angeordnet. Deshalb kommt ein Integerarray zum Einsatz, das an Position i die Adresse der Weiche mit der Nummer i enthält (wobei $0 \leq i \leq 27$).

5.5.3.2 Schleife zur Datenübertragung

Da die Interbus-Slaves passive Komponenten sind, ist es notwendig, das Sammeln der Daten über Reedkontaktbetätigungen in einem Polling-Verfahren zu implementieren, d. h. es müssen permanent Datenübertragungszyklen durchgeführt und die ausgelesenen Daten ausgewertet werden. Wie in Kapitel 4 beschrieben, werden bei einem Datenübertragungszyklus sowohl Daten geschrieben, als auch gelesen. Der Hardware-Thread führt die in Abbildung 5.5 dargestellte Schleife aus, wobei das Ausgabe-Bytearray fortlaufend als Nutzdatensatz an den Interbus-Master gesendet und das Eingabe-Bytearray mit den eingelesenen Daten aktualisiert wird.

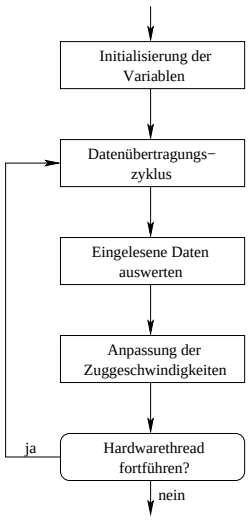


Abbildung 5.5: Schleife des Hardware-Threads

5.5.3.3 Auslesen von Reedkontaktmeldungen

Tritt eine Reedkontaktbetätigung auf, so muß eine entsprechende Nachricht zwischengespeichert werden, bis der Steuerungsthread diese abgefragt hat. Weiterhin sollte das in Abschnitt 4.3 beschriebene mehrfache Auslösen eines Reedkontaktes beim einmaligen Überfahren oder die Betätigung beider Kontakte eines Paares nur zu einer Meldung ausgewertet werden. Dazu wird eine variable Zeitspanne von etwa einer Sekunde festgelegt, innerhalb derer eine erneute Betätigung ignoriert wird.

Die Betätigung eines Reedkontaktes läßt sich durch den Vergleich neu eingelesener Nutzdaten mit denen des vorangegangenen Zyklus feststellen. Zuvor wird auf beide Datensätze eine Maske angewendet, die alle nicht zur Meldung einer Reedkontaktbetätigung genutzten Bits ausblendet. Durch eine geeignete Maske lassen sich so auch Meldungen über Reedkontaktbetätigungen von Gleisabschnitten ohne Reedkontakte vermeiden. Als Reedkontaktbetätigungen werden nur Veränderungen eines Bits von 0 nach 1 ausgewertet.

Zur Zwischenspeicherung der entsprechenden Meldungen kommt die in Abbildung 5.6 dargestellte verkettete Liste zum Einsatz. In ihr werden Meldungen über Reedkontaktbetätigungen, bezogen auf Reedkontaktpaare, zusammen mit einer Zeitmarke abgelegt. Es werden zwei Zeiger auf Listenelemente nachgeführt. Der erste zeigt auf das erste Element der Liste oder den Wert NULL, falls die Liste leer ist. Der zweite zeigt auf das als nächstes auszulesende Listenelement oder hat den Wert NULL, falls es kein solches Element gibt. Werden neue Listenelemente hinzugefügt, so geschieht dies immer am Ende der Liste.

Wird eine neue Kontaktbetätigung festgestellt, sind zunächst nicht mehr benötigte Listeneinträge zu entfernen. Dies sind Elemente, die bereits von der Steuerungssoftware ausgelesen wurden, d. h. in der Liste vor dem Zeiger auf das als nächstes auszulesende Element liegen und bei denen die Differenz zwischen der eingetragenen und der aktuellen Zeitmarke größer als das festgelegte Intervall ist.

Ist die Liste bereinigt, wird sie nach einem Eintrag durchsucht, der sich bzgl. des Reedkontaktpaares mit der aktuellen Reedkontaktbetätigung deckt. Ist die Suche erfolgreich, wird die neue Betätigung verworfen, ansonsten wird am Ende der Liste ein entsprechender neuer Eintrag angehängt.

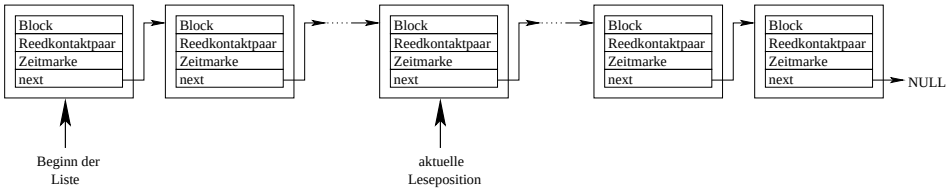


Abbildung 5.6: Der Puffer zur Speicherung von Meldungen über Reedkontaktbetätigungen ist als verkettete Liste implementiert.

Die Methode `GetContact()` liefert eine Meldung über die Betätigung eines Reedkontaktpaares zurück und setzt den zweiten Zeiger auf das nachfolgende Listenelement, falls die Liste nicht ausgelesene Elemente enthält. Ist dies nicht der Fall, hat der zweite Listenzeiger den Wert NULL, und `GetContact()` liefert einen entsprechenden Fehlercode zurück.

5.5.3.4 Ansteuerung von Aktoren

Die Ansteuerung der Aktoren erfolgt durch eine entsprechende Anpassung des Ausgabe-Bytearrays. Dies geschieht durch Aufruf der Methoden `SetSpeed`, `SetSignal` und `SetPoint`. Im nächsten Datenübertragungszyklus werden diese Daten automatisch an die Interbus-Slaves übertragen.

5.5.3.5 Auslesen des Anlagenzustandes

Aktuelle Parameter zum Zustand der Anlage können mit Hilfe der Methoden `GetSpeed`, `GetContact`, `GetNumberOfContacts`, `GetSignal` und `GetPoint` ausgelesen werden. Dabei wird auf die oben beschriebenen Datenstrukturen zurückgegriffen.

5.5.4 Hardwarenahe Maßnahmen zur Zugsteuerung

Da die auf der Anlage verwendeten Lokomotiven unterschiedliche Laufeigenschaften besitzen, ist es erforderlich, auf speziellen Blöcken die Geschwindigkeiten anzupassen. Zusätzlich wird beim Anfahren eines Zuges die Geschwindigkeit langsam erhöht. Diese Anpassungen werden in der Klasse `Track` durchgeführt.

Um ein langsames Anfahren zu realisieren, wird ein Array genutzt, in das für jeden von einem Zug belegten Block eine Soll- und aktuelle Geschwindigkeit sowie eine Zeitmarke eingetragen wird. Soll ein Zug nach einem Halt anfahren, wird von dem Steuerprogramm die Sollgeschwindigkeit des Zuges in der dem Block entsprechenden Arrayposition mit einer Zeitmarke eingetragen. In dem in Abbildung 5.5 dargestellten Zyklus wird im letzten Schritt für alle Blöcke die Differenz zwischen der eingetragenen Zeitmarke und aktueller Zeit berechnet. Bei Überschreitung eines festgelegten Intervalls wird die Geschwindigkeit auf dem Block um eine Stufe inkrementiert. Diese Anpassung wird so oft durchgeführt, bis die aktuelle Geschwindigkeit der Sollgeschwindigkeit des Zuges entspricht.

Für einige Gleisabschnitte ist eine gesonderte Anpassung der Geschwindigkeit notwendig. Dieses betrifft den Verbindungsblock zwischen der Paß- und Inner Circle-Station, der nur geringfügig länger ist, als die auf der Anlage befindlichen Züge.

Damit hier ein Halten der Züge trotzdem möglich ist, wird die maximale Geschwindigkeit auf eine mittlere Fahrstufe begrenzt.

Muß ein Zug auf einem ansteigenden Gleisabschnitt langsam an ein Signal heranfahren, ist es möglich, daß die für den Zug angegebene Minimalgeschwindigkeit nicht zum Überwinden der Steigung ausreicht. Der Zug bleibt dann frühzeitig in diesem Block stehen und erreicht das Lichtsignal nicht. Dies kann u. U. sogar zum Abbruch des Fahrbetriebes führen, da der Zug laut seinem Status noch fährt, aber für eine lange Zeitspanne keine Reedkontaktmeldungen mehr produziert werden (vgl. 5.3.2). Aus diesem Grund wird auf den betroffenen Blöcken die Minimalgeschwindigkeiten erhöht.

5.5.5 Kommunikation mit anderen Threads

Abbildung 5.7 zeigt die Datenstrukturen, die von mindestens zwei verschiedenen Threads genutzt werden. Alle dargestellten Datenstrukturen werden dabei durch die Klasse `Track` instantiiert und als private Variablen deklariert. Deshalb kann ausschließlich der ebenfalls in der Klasse `Track` implementierte Hardware-Thread direkt zugreifen. Alle anderen Threads nutzen die auf der rechten Seite der Abbildung dargestellten öffentlichen Methoden. Durch den Einsatz des Mutex-Mechanismus in der Klasse `Track` kann so gewährleistet werden, daß diese Zugriffe immer im gegenseitigen Ausschluß stattfinden.

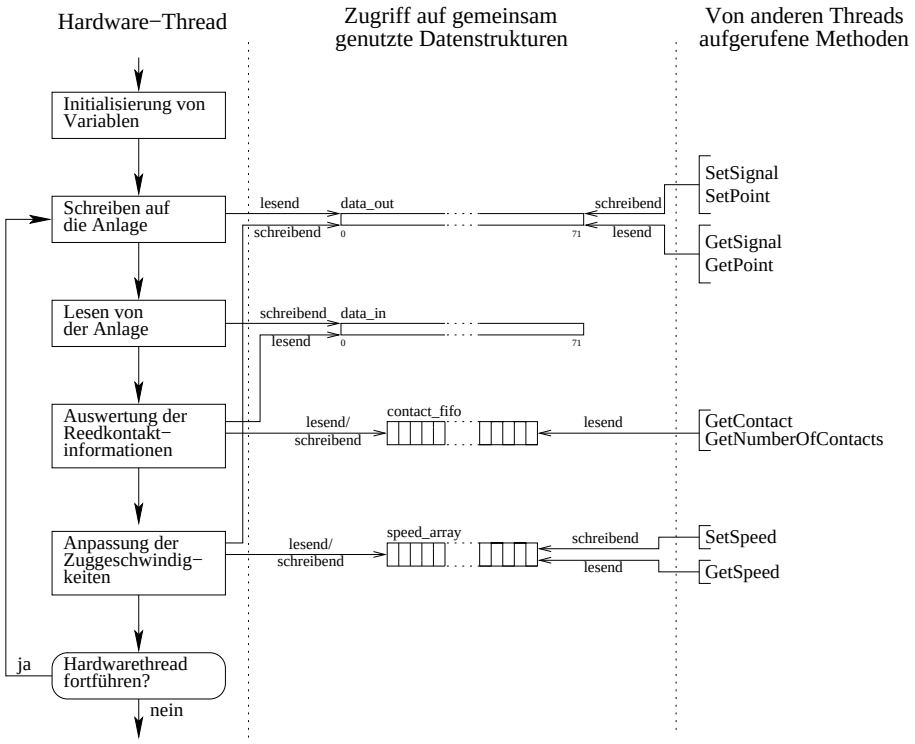


Abbildung 5.7: Schleife des Hardware-Threads mit Zugriffen auf gemeinsam genutzte Datenstrukturen

5.6 Implementierung von Petrinetzen

Um die in Kapitel 3 vorgestellten Netzmodelle in das Steuerprogramm zu integrieren, wird die im Folgenden erläuterte Klassenstruktur verwendet. Diese

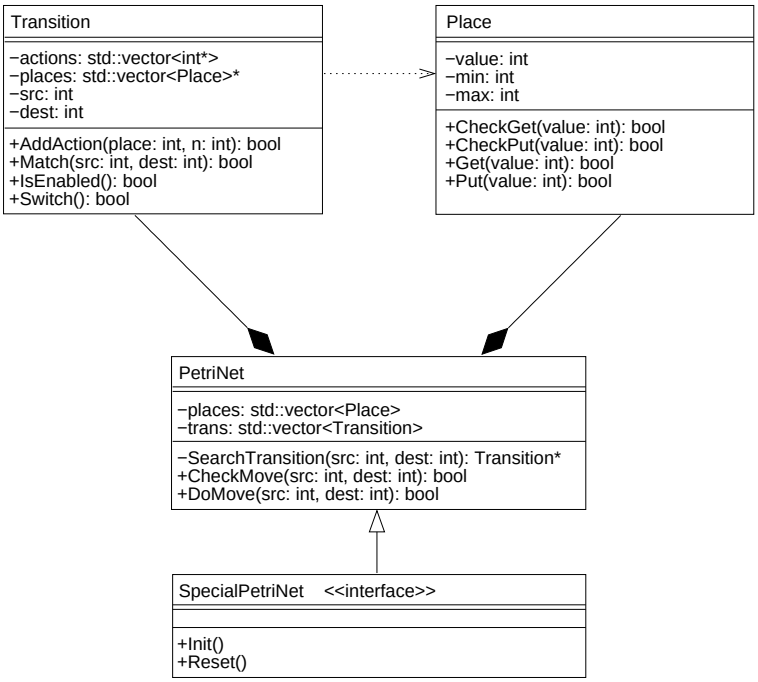


Abbildung 5.8: UML-Klassenstruktur der Petrinetzimplementierung

Struktur erlaubt die Darstellung von Stellen und Transitionen und somit die Repräsentation eines Netzes im Steuerprogramm. Die Abbildung 5.8 liefert einen Überblick über die Klassenstruktur.

Fährt ein Zug von einem Gleisabschnitt in den nächsten, so entspricht dies dem Schalten einer Transition in dem in Kapitel 3 dargestellten Netzmodell. Mit Hilfe der in Klasse `SpecialPetriNet` und `PetriNet` implementierten Methoden ist es möglich, entsprechende Transitionen auf ihre Aktivierung zu prüfen. Ist das Schalten der Transition möglich, kann der Zug in den nächsten Block einfahren. In diesem Fall muß auch die Transition der Petrinetzimplementierung schalten, um die Belegung der Stellen nachzuführen.

Eine Instanz der Klasse `Place` stellt eine Stelle des Netzes dar, der mit Hilfe der Methodenaufrufe `Get(n)` und `Put(n)` Token entzogen oder hinzugefügt werden können. Wird ein negativer Wert für `n` angegeben, entspricht `Put(-n)` dem Aufruf `Get(n)` und umgekehrt. Der Rückgabewert der Methoden ist dabei `true`, falls die Operation erfolgreich war, ansonsten `false`. Dies ist auch dann der Fall, wenn die Stelle mit zu wenigen oder zu vielen Token belegt ist. Es wird dann keine Änderung der Tokenbelegung durchgeführt.

Identisch verhalten sich die Methoden `CheckPut` und `CheckGet` mit dem Unterschied, daß diese grundsätzlich nicht die Tokenbelegung der Stelle ändern. Sie stellen nur Anfragen dar, ob die entsprechende `Get` oder `Put` Operation erfolgreich durchführbar ist. Die Methoden sind notwendig, da vor dem Schalten einer Transition zunächst alle betroffenen Stellen überprüft werden müssen, bevor Token abgezogen bzw. hinzugefügt werden.

Eine Instanz der Klasse `Transition` stellt eine einzelne Transition innerhalb des Netzes dar. Sie besitzt einen Zeiger auf die Liste aller im Netz befindlichen Stellen und eine Liste von Aktionen, die beim Schalten dieser Transition stattfinden. Eine einzelne Aktion ist durch zwei Integerwerte kodiert. Der erste gibt den Index der

Stelle an, von oder zu der Token transferiert werden sollen. Der zweite Wert gibt die Anzahl n an, wobei ein negativer Wert das Entfernen und ein positiver Wert das Hinzufügen von n Token signalisiert. Die Transition ist somit aktiviert, wenn alle Aktionen durchgeführt werden können.

Die privaten Membervariablen `src` und `dest` bilden eine eindeutige Identifikation der Transition und werden mit Blocknummern belegt. D. h., wenn eine Transition das Bewegen eines Zuges von einem Gleisabschnitt `A` in den nachfolgenden Gleisabschnitt `B` darstellt, ist `src = A` und `dest = B`. Dadurch ist es möglich, Zugbewegungen zu bearbeiten, eine Zuordnung zu Transitionsnamen o. ä. entfällt.

Zur Darstellung einer Transition ist es notwendig, einen Zeiger auf die Liste der Stellen des Netzes und eine ID zu setzen. Deshalb verfügt diese Klasse nur über einen parameterbehafteten Konstruktor. Weiterhin müssen mit Hilfe der Methode `AddAction` Aktionen definiert werden.

Mit der Methode `Match` kann die ID der Transition überprüft werden. Darauf kann ggf. mit der Methode `IsEnabled` getestet werden, ob die Transition aktiviert ist, um diese dann mit der Methode `Switch` zu schalten.

Die Klasse `PetriNet` bildet ein vollständiges Petrinetz ab und besitzt ein Liste aller Stellen und Transitionen. Die Methoden `CheckMove` und `DoMove` suchen mit Hilfe der Methode `SearchTransition` nach einer zu den Parametern passenden Transition und liefern `true` zurück, falls die Transition aktiviert ist, sonst `false`.

Wie im Folgenden deutlich wird, ist es nicht sinnvoll das vollständige Petrinetz der Modellbahnanlage abzubilden, stattdessen werden insbesondere Blockbelegungen gesondert dargestellt. Deshalb liefern die Methoden `CheckMove` und `DoMove` grundsätzlich `true` zurück, falls keine passende Transition im vorhandenen Netz gefunden wird.

5.7 Steuerung des Fahrbetriebs

Der Benutzer kann durch die Angabe der abzufahrenden Bahnhöfe jedem Zug einen Fahrplan zuweisen. Diese Angaben müssen vor dem Ablauf in eine Blockliste umgewandelt werden, die dann für jeden Zug abgearbeitet wird. Diese Abarbeitung der Fahrpläne erfolgt innerhalb einer Schleife, die erst terminiert, wenn alle Züge ihren Fahrplan beendet haben. Abbildung 5.9 verdeutlicht die dabei durchgeführten Schritte.

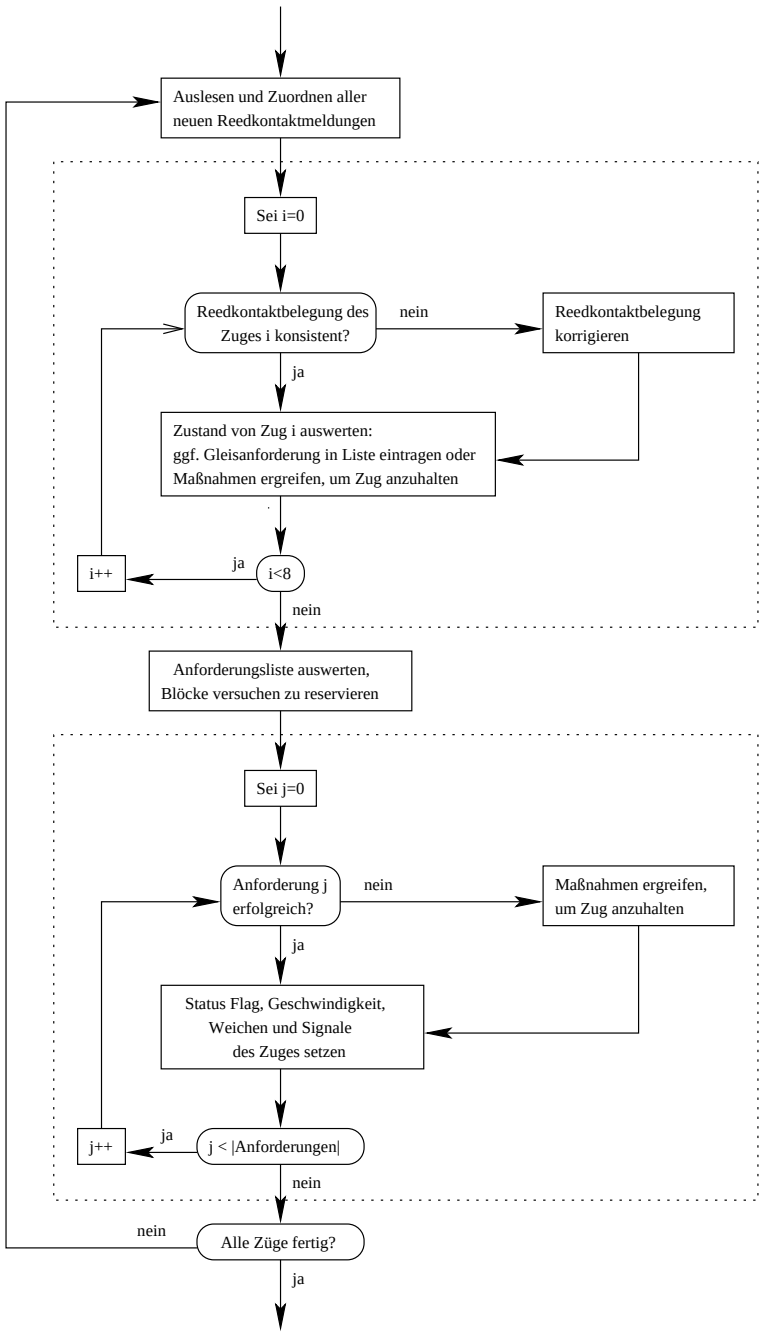


Abbildung 5.9: Hauptschleife zur Zugsteuerung

Zuerst werden die von dem Hardware-Thread gelieferten Reedkontakte abgefragt und anhand der Gleisnummer den Zügen zugeordnet. Danach werden in einem ersten Abschnitt für alle Züge i die Reedkontaktzählerstände auf Fehler überprüft und gegebenenfalls korrigiert, anschließend werden die Blockanforderungen der Züge in eine Liste eingetragen. Dabei müssen nicht alle Züge in der Liste vorhanden sein, so treten z. B. für einen Zug mit abgearbeitetem Fahrplan keine weiteren Anforderungen mehr auf. Sind alle Anforderungen eingetragen, wird die Liste ausgewertet und für jeden Zug j aus der Liste die Einfahrtsgenehmigung erteilt oder verweigert. In einem zweiten Abschnitt werden anhand der ausgewerteten Anforderungsliste u. a. die Geschwindigkeiten der Züge angepaßt.

5.7.1 Erstellung von Fahrplänen

Der Benutzer kann durch die Angabe einer beliebigen Kombination der drei Kürzel 'I'/'i', 'O'/'o' und 'K'/'k' für die Inner-Circle-, Outer-Circle- und Paß-Station für jeden Zug einen Fahrplan festlegen. Für eine Verwendung durch das Steuerprogramm müssen diese Zeichenketten in die beschriebenen verketteten Listen mit Blocknummern umgewandelt werden (vgl. 5.3.3).

Dazu werden zunächst alle Zeichen ungleich der oben genannten aus den Zeichenketten entfernt, um eine fehlerhafte Generierung der Fahrpläne zu vermeiden. Das erste Element einer Liste bildet der für jeden Zug bekannte Heimatblock mit entsprechender Fahrtrichtung. Die Abfolge der Blöcke zwischen den einzelnen Bahnhöfen ist bekannt, sodaß diese Blockfolgen in die Liste entsprechend der angegebenen Bahnhöfen eingehängt werden. Entspricht der letzte eingegebene Bahnhof auch dem Heimatbahnhof, muß nur der letzte Block auf das Heimatgleis korrigiert werden. In den anderen Fällen wird die Strecke bis zum Heimatblock ergänzt.

Bei den im Paßbahnhof beheimateten Zügen ist zusätzlich darauf zu achten, daß die Richtung der Züge am Ende ihrer Fahrpläne der beim Start entspricht. Dazu wird gegebenenfalls der Fahrplan um eine Durchfahrt von Inner-Circle- oder Outer-Circle-Station vor dem Paßbahnhof ergänzt, um einen abschließenden Richtungswechsel zu erzwingen.

5.7.2 Reservierung von Gleisabschnitten

Da die Blöcke jeweils nur von einem Zug befahren werden, ist es notwendig zu prüfen, ob für einen Zug die Einfahrt in einen Nachfolgeblock erlaubt ist. Sollte dieser belegt sein, muß die Einfahrt verhindert werden. In den Bahnhöfen kann alternativ ein Ausweichgleis befahren werden. Die Einfahrt eines Zuges in einen Block hängt aber nicht nur von dessen Belegung ab. So kann ein Nachfolgeblock im Paß zwar frei sein, eine Einfahrt würde aber zu einem Deadlock führen. Auch ist eine Überprüfung abhängig von dem aktuellen Zustand eines Zuges. Die nachfolgende Abbildung 5.10 zeigt die Notwendigkeit einer Blockreservierung in Abhängigkeit der Zugposition. Diese wird, wie schon erläutert, über die Reedkontaktmeldungen festgestellt, deren Zähler in der Abbildung 5.10 als Tupel dargestellt wird: (< aktueller Block>,<Nachfolgeblock>).

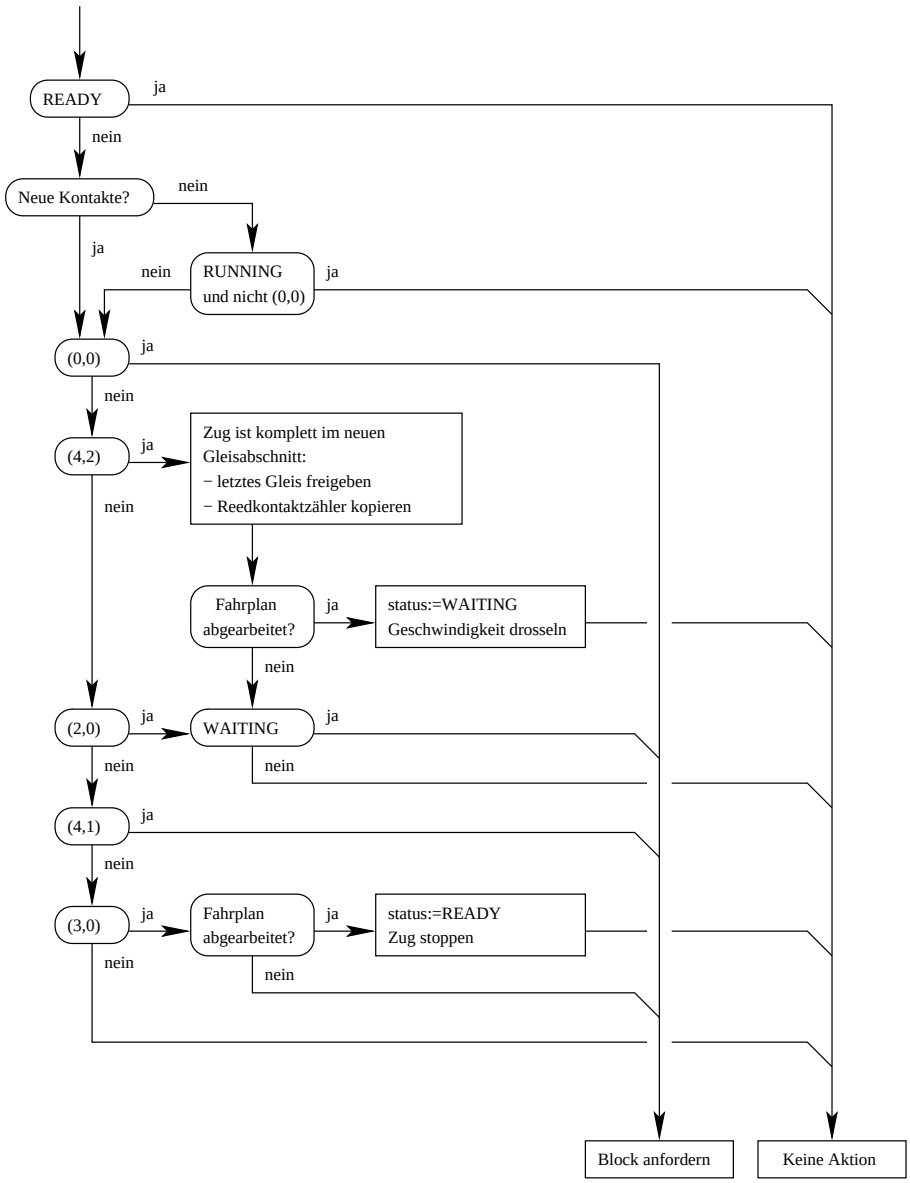


Abbildung 5.10: Auswertung des Zugstatus

Dabei kann z. B. bei der Einfahrt eines Zuges in einen Block (Zählerstand (4,1)) überprüft werden, ob der Nachfolgeblock auch frei ist, um eine direkte Weiterfahrt ohne Geschwindigkeitsänderung zu erlauben. Die Zuteilung eines Blocks darf aber nicht sofort durchgeführt werden, da sonst Züge mit höherer Priorität bei einem Reservierungsversuch keine Einfahrtsgenehmigung erhalten. Darum werden alle Anforderungen in die erwähnte Liste eingetragen, die eine Zuteilung der Blöcke nach Prioritäten erlaubt.

Die Reservierung eines Blocks im Paß wird zusätzlich über die in Abschnitt 5.6 vorgestellte Petrinetzklasse realisiert. Dazu werden für den Betrieb als eingleisige Paßstrecke und als Paßstrecke mit Ausweichgleis zwei Netzmodelle implementiert. Abbildung 5.11 zeigt als Beispiel das Paßmodell mit Ausweichgleis. Die schwarz gekennzeichneten Komponenten werden durch die Petrinetzklasse abgebildet, die Belegung der einzelnen Gleise (graue Komponenten) wird aus Gründen der Übersichtlichkeit in der Klasse `Track` realisiert.

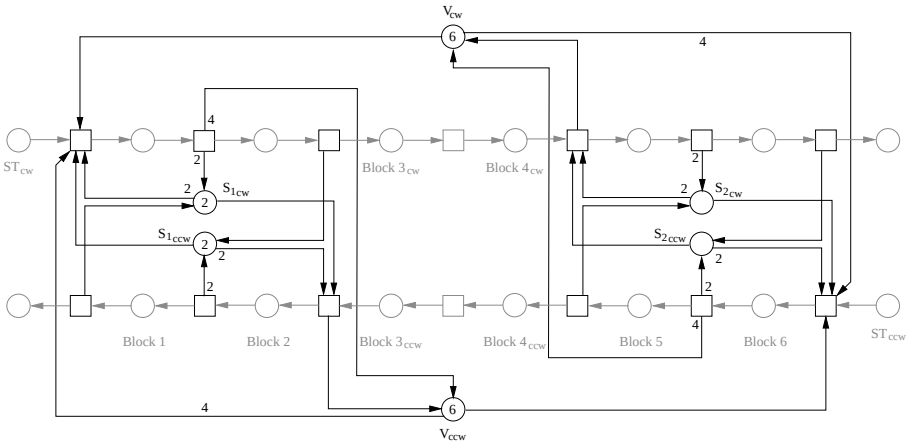


Abbildung 5.11: Reduziertes Netzmodell der Paßstrecke mit Ausweichgleis

Im Folgenden wird der Reservierungsmechanismus vorgestellt, der die Einfahrt eines Zuges in einen Block überprüft.

5.7.2.1 Validierung der Reedkontaktzählerstände

Einige Fehler innerhalb der Reedkontaktmeldungen können nur unter Kenntnis der Zugzustände erkannt und teilweise auch korrigiert werden. Da diese Informationen nur innerhalb des Steuerungprogramms zur Verfügung stehen, müssen hier weiterführende Maßnahmen zur Fehlerbehandlung getroffen werden. Wird ein Fehler innerhalb der Reedkontaktkonfiguration eines Zuges erkannt, muß entschieden werden, ob dieser korrigiert werden kann oder aus Sicherheitsgründen der Betrieb eingestellt werden muß.

Diese Überprüfung und ggf. Korrektur der Reedkontaktstati wird zu Beginn eines jeden in Abbildung 5.9 dargestellten Zyklus für jeden Zug durchgeführt, bevor Entscheidungen über Gleisreservierungen, -freigaben, Geschwindigkeitsänderungen, usw. getroffen werden.

Grundsätzlich ist festzuhalten, daß sowohl Kontaktmeldungen ausbleiben, als auch nicht aufgetretene Betätigungen gemeldet werden können (vgl. Kapitel 4). Deshalb kann es im Fehlerfall notwendig sein, sowohl Kontaktmeldungen der aktuellen Konfiguration hinzuzufügen, als auch zu verwerfen. Das führt unter Umständen zu nicht eindeutig aufzulösenden Situationen. Man betrachte beispielsweise einen Zug, der von Block A in den Block B fährt und dabei den Reedkontaktstatus (3,2)[5] hat. Dann wurden 3 Kontaktmeldungen in Block A und zwei im Block B empfangen, was offensichtlich fehlerhaft ist. Es gibt zwei Möglichkeiten, wie es dazu gekommen sein kann:

1. Es wurde ein Kontakt in Block A nicht gemeldet und der Zug befindet sich nun komplett in Block B.

2. Es wurde in Block B ein Kontakt zu viel gemeldet und der Zug befindet sich noch zwischen Block A und B.

Es wird deutlich, daß es an dieser Stelle hilfreich sein kann, auch zu betrachten, welche der Kontakte in den Blöcken betätigt worden sind. So gibt es eine Reihe von Kombinationen, die zu dem Status (3,2) geführt haben können. Exemplarisch seien drei Möglichkeiten genannt, die jeweils verschiedenen Fehlerkategorien angehören:

[5] Ein 2-Tupel (x,y) deutet die zusammengefaßten Reedkontaktzählerstände an, wobei x der Zähler des vorangehenden und y der des nachfolgenden Blockes ist.

(1,2,2,0)[6] Man sieht leicht ein, daß hier offensichtlich eine Kontaktmeldung des ersten Kontaktpaares im ersten Block verlorengegangen ist und sich der Zug nun bereits vollständig im zweiten Block befindet. Dementsprechend wird dieser Zustand in (2,2,2,0) korrigiert.

(2,1,1,1) Bei diesem Zustand liegt es nahe, daß im zweiten Block der zweite Kontakt fälschlich als betätigt gemeldet worden ist. Ansonsten müßten zwei Kontakte verloren gegangen sein. Also wird dieser Zustand in (2,1,1,0) korrigiert werden.

(0,3,0,2) Dieser Reedkontaktstatus kann nur durch eine Reihe von Fehlern zustande gekommen sein. In einem solchen Fall ist der Betrieb aus Sicherheitsgründen sofort einzustellen.

Wie man an diesen Beispielen sieht, ist eine Bewertung der möglichen Fehler notwendig. Wenn man etwa annimmt, daß zuviel gemeldete Kontaktbetätigungen unwahrscheinlicher gegenüber verlorengegangenen sind, würde man im zweiten Beispiel u. U. anders entscheiden.

Grundsätzlich kann man davon ausgehen, daß eine überflüssige Kontaktmeldung aufgetreten ist, falls einer der Zähler größer als 2 wird. Diese Zählerstände werden grundsätzlich auf 2 korrigiert. Dadurch läßt sich eine Matrix aufstellen, die alle möglichen Kombinationen (insgesamt 81), die ggf. korrigierten Zählerstände und jeweils ein Flag für den Typ des aufgetretenen Fehlers enthält. Der Fehlertyp kodiert dabei die Zustände

kein Fehler: Die Kontaktzähler bleiben unverändert.

korrigierbarer Fehler: Die Kontaktzähler werden mit den Korrekturwerten aus der Matrix überschrieben.

nicht korrigierbarer Fehler: Der Zugbetrieb wird aus Sicherheitsgründen sofort eingestellt.

Die Matrix ermöglicht so, unter der Voraussetzung, daß jeder Zählerstand größer zwei auf zwei zurücksetzt wird, eine individuelle Reaktion auf jeden möglichen Zählerstand.

Als weitere Sicherheitsmaßnahme besitzt jeder Zug eine Zeitmarke, in der die letzte Reedkontaktbetätigung des Zuges vermerkt wird. Bewegt sich ein Zug auf der Anlage, kann man davon ausgehen, daß er innerhalb eines Zeitintervalls von z. B. 60 Sekunden wieder einen Reedkontakt auslöst. Ist das nicht der Fall, wird ebenfalls der Zugbetrieb sofort eingestellt, da der Zug dann beispielsweise entgleist sein oder einen Magneten verloren haben könnte.

5.7.2.2 Auflösen von Konfliktsituationen

Die auf der Modellbahnanlage auftretenden Konflikte werden mit Hilfe von Prioritäten aufgelöst. Jeder Zug besitzt eine Priorität, die der Benutzer zuweisen kann oder automatisch generiert wird. Konkurrieren mehrere Züge bezüglich der Anforderung an einen Gleisabschnitt, ist es mit Hilfe der Prioritäten möglich, diesen Konflikt gezielt aufzulösen. Besteht ein Konflikt zwischen Zügen mit gleicher Priorität, ist eine Zufallsentscheidung zu treffen.

[6] Bei einem 4-Tupel (a,b,c,d) stehen die Werte für die einzelnen Reedkontaktzähler entsprechend der Fahrtrichtung. D. h. a,b sind die Zähler des ersten und c,d die Zähler des zweiten Blocks und a,c jeweils die Zähler des ersten Kontaktpaares innerhalb des Blocks.

Es lassen sich nun verschiedene Strategien entwerfen, Prioritäten zu ermitteln, die sich aus verschiedenen Zielsetzungen ergeben. Drei in diesem Zusammenhang naheliegende Methoden sind implementiert:

1. **Statische Prioritäten** werden vom Benutzer vor oder während der Laufzeit des Programms festgelegt. Dieses ist immer dann sinnvoll, wenn Züge aus nicht anhand der dem Programm zur Verfügung stehenden Informationen kategorisiert werden sollen. Ein typisches Beispiel ist die Bevorzugung von Personenzügen gegenüber Güterzügen. Es ist in diesem Modus auch möglich, während der Abarbeitung eines Fahrplans Prioritäten neu zu vergeben, um z. B. die Abarbeitung der Fahrpläne einzelner Züge zu beschleunigen.

2. Das Prinzip des **Aging** ist eine Erweiterung der statischen Prioritäten, die vermeidet, daß Züge mit niedriger Priorität von Zügen mit hoher Priorität ausgehungert werden. Dies ist z. B. dann denkbar, wenn ein Zug A mit niedriger Priorität mit mehreren Zügen höherer Priorität in einem Kreis der Anlage verkehrt. Wenn ausreichend Züge höherer Priorität vorhanden sind, wird sich neben Zug A auch immer ein konkurrierender Zug mit höherer Priorität im Bahnhof befinden und A wird warten, bis die anderen Züge den Kreis verlassen oder Ihren Fahrplan beendet haben.
   Um diese Situation zu vermeiden wird die Priorität des Zuges A bei jeder zu seinen Ungunsten entschiedenen Konfliktsituation inkrementiert. So ist gewährleistet, daß er nach einer endlichen Zahl von Konfliktsituationen eine ebenso hohe oder höhere Priorität besitzt wie seine Konkurrenten. Gewinnt er eine Konfliktsituation, wird seine Priorität auf den ursprünglichen Wert zurückgesetzt.

3. Die dritte Methode arbeitet **fahrplanorientiert**. Sie zielt darauf ab, daß alle Züge Ihre Fahrpläne in etwa dem gleichen Zeitraum beenden. Dazu werden die Prioritäten aufsteigend entsprechend der Länge des verbleibenden Fahrplans vergeben, d.h. der Zug mit dem längsten verbleibenden Fahrplan bekommt die höchste Priorität. Dadurch wird er in Konfliktsituationen bevorzugt und verliert weniger Zeit als andere mit dem Warten auf die Freigabe von belegten Nachfolgegleisen. Dadurch gleichen sich im Idealfall die Längen der verbleibenden Fahrpläne immer mehr an.

In der Anforderungsliste stehen alle Züge, die einen weiteren Block beanspruchen. Die nachfolgende Abbildung 5.12 stellt die Auswertung der Liste dar.

In dieser Liste werden zuerst die eingetragenen Züge absteigend nach Prioritäten sortiert. Konkurrieren mehrere Züge um einen Block wird die Reihenfolge per Zufallsentscheidung ermittelt. Die sortierte Liste kann nun der Reihe nach abgearbeitet werden. Für jeden Zug in der Liste wird versucht, den gewünschten Block zu reservieren. Ist eine Reservierung erfolgreich, ist sichergestellt, daß der Zug mit der höchsten Priorität einfahren kann, da die Reservierungen evtl. konfligierender Züge fehlschlagen.

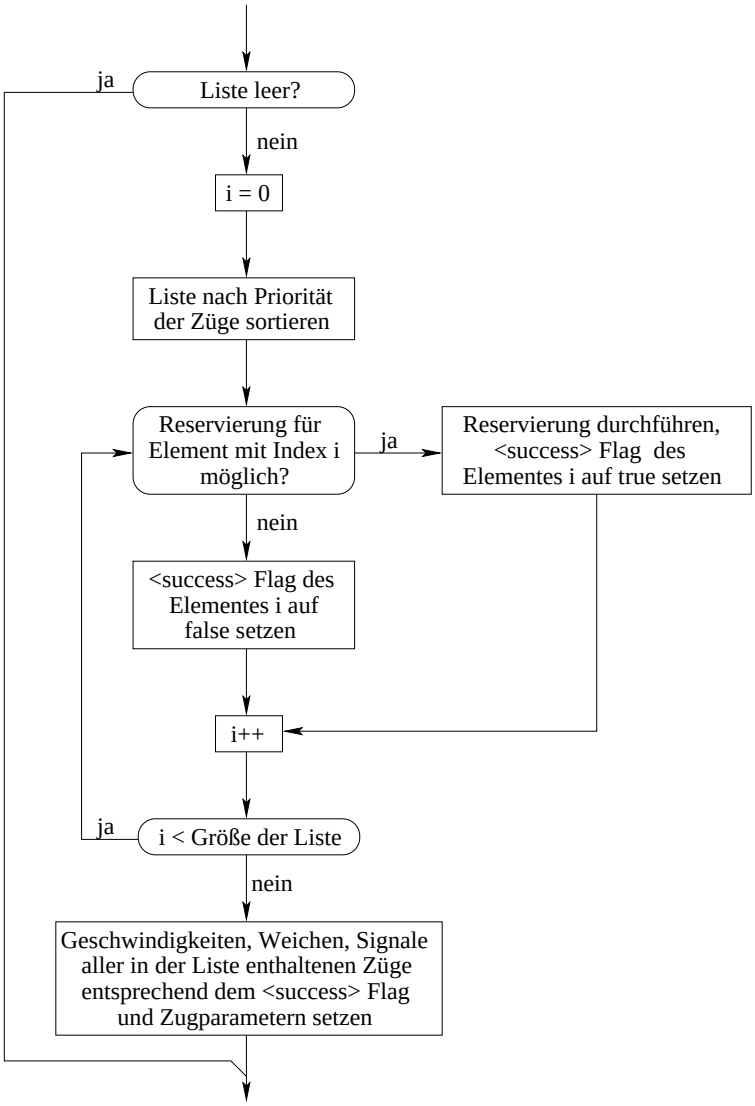


Abbildung 5.12: Auswertung der Blockanforderungsliste

5.7.2.3 Prüfung der Blockbelegung

Bei der Auswertung der oben beschriebenen Anforderungsliste wird der Reihe nach versucht, die angeforderten Blöcke fúr die entsprechenden Züge zu reservieren. Falls der Block belegt ist, besteht im Falle eines Bahnhofsgleises die Möglichkeit über ein evtl. freies Gleis des gleichen Bahnhofs auszuweichen. Dazu sind die Blöcke in die Gruppen Streckengleis und Bahnhofsgleis aufgeteilt.Bei einem Bahnhofsgleis werden alle Blöcke des Bahnhofs überprüft. Ist einer frei, wird in dem Fahrplan der ursprünglich eingetragene Block durch den freien ersetzt. Besteht keine Ausweichmöglichkeit, muß der Zug warten.

Ist der referenzierte Block noch nicht belegt, müssen für Gleisabschnitte im Paß die in den Netzmodellen vorgestellten Überprüfungen vorgenommen werden. Diese werden mit Hilfe der in Abschnitt 5.6 vorgestellten Petrinetzklasse durchgeführt. Sind die Bedingungen für die Reservierung eines Nachfolgeblocks erfüllt, wird getestet, ob vor dem gewünschten Block noch Weichenblöcke zu durchfahren und ob diese verfügbar sind. Sind auch diese frei, wird der Gleisabschnitt und evtl. vorhan-

denen Weichenblöcke für den Zug reserviert.

Eine weitere Überprüfung muß vorgenommen werden, wenn der Fahrplan eines Zuges kurz vor dem Ende steht. Da ein Zug auf seinem Heimatgleis den Fahrbetrieb beenden soll, darf dieser nur den vorletzten Bahnhof verlassen, wenn die Strecke bis zu seinem Heimatgleis reserviert werden kann. Andernfalls kann ein Deadlock auftreten, wenn der auf dem Heimatgleis stehende zweite Zug den Block nicht mehr verläßt.

Ist die Liste ausgewertet, werden für alle Züge anhand erfolgreicher oder fehlgeschlagener Reservierungen die Geschwindigkeiten, Weichen und Signale gesetzt.

5.8 Grafische Oberfläche

Der Benutzer muß für die einzelnen Züge die Fahrpläne erstellen können und die Möglichkeit besitzen, die Zugparameter, wie z. B. Prioritäten, zu verändern. Eine textbasierte Eingabe bietet nur wenig Komfort und erlaubt zusätzliche Anzeigen nur z. B mit Hilfe der Bibliothek "curses", die unter anderem Funktionen zur Textpositionierung bereitstellt. Diese Bibliothek ist aber nicht thread-sicher. Darum bot sich die Entwicklung einer grafischen Oberfläche an, die dem Benutzer die notwendigen Eingaben erleichtert und zusätzlich die Rückmeldungen der Anlage auf dem Bildschirm visualisiert. Als Basis dafür wurden die Qt-Bibliotheken der Firma Trolltech gewählt. Diese C++ Bibliotheken ermöglichen eine einfache Entwicklung einer GUI für Windows- und UNIX-basierte Plattformen. Qt wurde 1996 vorgestellt, ist objektorientiert und enthält eine Vielzahl von Komponenten, die für eigene Programme verwendet werden können.

5.8.1 Entwicklung und Implementierung

Zur Hilfe bei der visuellen Gestaltung einer Oberfläche gibt es den Qt-Designer. Dabei handelt es sich um einen Editor, mit dessen Hilfe man Elemente wie Eingabefelder oder Schaltflächen in seinem Programmfenster wie gewünscht anordnen kann. Dabei lassen sich jeder Komponente noch eine Vielzahl an Eigenschaften zuordnen, so kann z. B. die Länge eines Eingabefeldes festgelegt oder die Eingabe auf bestimmte Zeichen beschränkt werden. Eine wichtige Eigenschaft ist die Zuweisung eines passenden Namens, damit bei mehreren Schaltflächen im Quelltext eine korrekte Zuordnung gefunden werden kann. Sollte eine Komponente nicht zur Verfügung stehen, gibt es die Möglichkeit eigene zu entwerfen und diese mit in die Oberfläche zu integrieren.

Ist der visuelle Entwurf abgeschlossen, müssen Vorbereitungen getroffen werden, damit auf Benutzeraktionen reagiert werden kann. Qt beinhaltet ein Signal/Slot-Konzept, im Gegensatz zu anderen Entwicklungsumgebungen. Dort werden Funktionszeiger als Reaktionen auf Benutzeraktionen verwendet, in Qt allerdings besitzen Klassen Slots und Signale. Signale werden von Objekten freigesetzt, etwa falls ein Benutzer eine Schaltfläche aktiviert. Dabei erhält das sendende Objekt keinerlei Rückmeldung, ob das Signal empfangen wurde oder nicht. Als Empfänger dienen Memberfunktionen, die bei entsprechendem Signal ausgeführt werden. Da die verwendeten Objekte so nicht miteinander kommunizieren, können mehrere Signale an einen Slot angeknüpft werden und umgekehrt. Als abschließende Aktion müssen somit alle Signale entsendenden Objekte, wie Schaltflächen, Checkboxen usw. mit Slots verknüpft werden.

Um den vorhandenen Sourcecode an die Oberfläche anzubinden, müssen von dieser zuerst die erforderlichen Header- und Implementierungsdateien erzeugt werden. Dazu wird der Qt User Interface Compiler verwendet, der aus der Oberflächendatei die benötigten C++ Header- und Implementierungsdateien erzeugt. Da das

Signal/Slot-Konzept eine Erweiterung von Qt ist, muß der Meta Object Compiler verwendet werden, der die von der grafischen Oberfläche erzeugten C++ Dateien einliest und die unter anderem für den Signal und Slot-Mechanismus benötigten Erweiterungen in den Quellcode einfügt. Da bei jeder Änderung der Oberfläche der Meta Object Compiler die Dateien neu erzeugt werden, muß für die eigenen Implementierungen der Slot-Methoden eine eigene Klasse verwendet werden, die von der vom Compiler generierten Klasse abgeleitet wird. In den automatisch generierten Dateien werden die Slot-Methoden als virtuelle Methoden deklariert, dadurch können dann durch Überladungsmechanismen die selbst geschriebenen Methoden verwendet werden. In diesen Methoden werden dann die Funktionen aufgerufen, die die Anlage steuern.

5.8.2 Kontroll- und Eingabebereiche

Die Oberfläche bietet auf fünf Reitern Einstellungs- und Kontrollmöglichkeiten für den Zugbetrieb. Zum einen existiert eine Eingabemöglichkeit für Fahrpläne, in der jedem Zug, ausgehend von dem Heimatblock, eine Folge der Bahnhöfe 'I', 'O' und 'K' zugewiesen werden kann. Ein Zähler gibt dabei an, wie oft der Fahrplan wiederholt werden soll. Diese Wiederholung wird während der Ausführung nicht mit anderen Zügen synchronisiert.

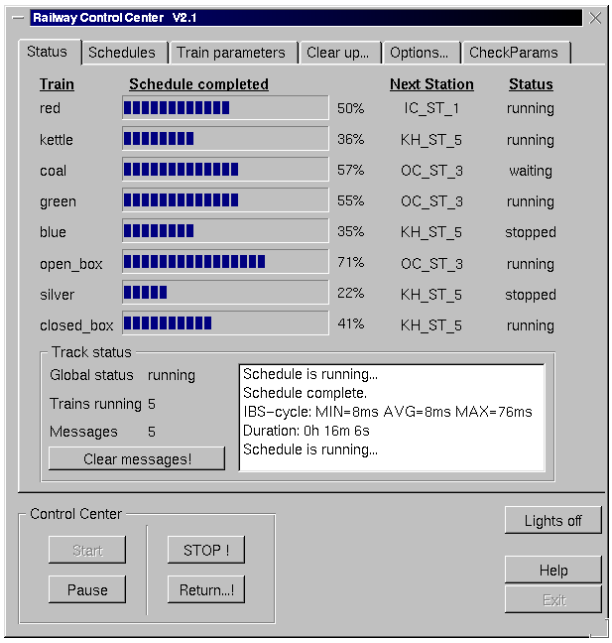


Abbildung 5.13: Anzeige zur Überwachung des Fahrplanfortschritts

Eine Speichern- und Laden-Option erlaubt das Weiter- und Wiederverwenden von Fahrplänen. Ein weiterer Reiter legt die Zugcharakteristika fest, hier können den Zügen Prioritäten sowie die Minimal- und Maximalgeschwindigkeit zugewiesen werden. Für die Festlegung der Prioritäten gibt es drei Auswahlmöglichkeiten, die in 5.7.2.2 erläutert wurden. In dem Reiter für den Gesamtablauf läßt sich durch Wahl der Netzmodelle der Paß als eingleisige Strecke oder als Strecke mit Ausweichgleis festlegen, im weiteren können Dateien mitgeschrieben werden, in denen Status- sowie Fehlermeldungen protokollieren und somit eine spätere Auswertung und Fehlersuche erleichtern. Ebenfalls wird hier über die Angabe der Anzahl der Fairneßtoken das Richtungswechselverhalten festgelegt. Sollten Züge ihre Fahrpläne

nicht beendet haben, läßt sich durch Eingabe der aktuelle Position und Richtung der falsch positionierten Züge der Rückführungsmodus aktivieren, der die betroffenen Züge wieder in ihre Ausgangspositionen fährt. Die Abbildung 5.13 zeigt die Anzeige zur Überwachung des Fahrplanfortschritts, die für jeden Zug prozentual den bisher abgearbeiteten Fahrplan und seinen nächsten Bahnhof angibt. Während der Abarbeitung kann man den Fahrplan pausieren oder abbrechen. Weitere Abbildungen zur Oberfläche sind im Anhang zu finden.

Da auf der Oberfläche laufend der aktuelle Zustand dargestellt werden soll, aber ebenso noch Benutzereingaben verwaltet werden müssen, wird ein weiterer Thread verwendet, der die von Qt gelieferten Methoden verwendet. Sobald die Abarbeitung eines Fahrplans beginnt, wird dieser Thread gestartet, um die Anzeigen zu aktualisieren. Dementsprechend wird er beendet, sobald der Fahrplan abgearbeitet ist. Bevor eine Änderung der angezeigten Werte vorzunehmen ist, wird ein Mutex bzgl. der grafischen Oberfläche gesetzt, der den Zugriff für andere Threads sperrt. Wurden alle Änderungen durchgeführt, wird der Mutex wieder aufgehoben.

5.8.3 Eingriffe in den laufenden Fahrbetrieb

Drei Optionen zum direkten Eingriff in die Zugsteuerung stehen dem Benutzer zur Verfügung. Zum einen kann während des Betriebs der Fahrplan pausiert werden. Dazu werden durch das Steuerprogramm alle Reservierungsversuche der Züge unterbunden, sodaß diese am Ende der aktuell befahrenen Blöcke zum Stehen kommen. Beim Aufheben der Pause können Reservierungen wieder vorgenommen werden und der Zugbetrieb wird fortgesetzt.

Zum anderen kann der Fahrplan abgebrochen werden, dabei sollen die Züge auf kürzestem Wege ihre Heimatgleise aufsuchen. In den Fahrplänen der Züge werden die jeweils nächsten Bahnhofsgleise gesucht und alle darauf folgenden Elemente der Liste gelöscht. Aus Sicherheitsgründen wird diese Suche erst ab dem dritten Block vorgenommen. An die gekürzten Fahrpläne wird dann anhand der bekannten Verbindungen zwischen Bahnhöfen der Weg zum jeweiligen Heimatgleis angehängt. Dabei ist für Züge mit Heimatgleis im Paßbahnhof die Richtung der Einfahrt zu beachten und ggf. eine Verbindung zum Outer- oder Inner-Circle-Bahnhof einzufügen.

Falls ein Fahrplan nicht korrekt beendet wurde und die Züge verteilt auf der Anlage stehen, können sie mit Hilfe einer Rückführungsmethode auf die Ausgangsposition bewegt werden. Dazu werden, analog zum Abbruch eines Fahrplans, ausgehend von den aktuellen Positionen der Züge deren Wege zum nächsten Bahnhof bestimmt, von dort aus wird die restliche Strecke bis zum Heimatgleis berechnet.

Kapitel 6

Zusammenfassung

Das bekannte Modell der "Fünf dinierenden Philosophen" (vgl. [4]) zeigt eine Problemstellung auf, die in vielen praxisbezogenen Systemen in ähnlicher Form zu lösen ist – so auch auf der vorgestellten Modellbahnanlage. Das Layout der Anlage ist so gewählt, daß auf Ihr alle typischen organisatorischen Probleme arbeitsteiliger Systeme auftreten. Aufgrund der Komplexität der Aufgabenstellung ist es notwendig, geeignete Werkzeuge zur Modellierung des Systems zu nutzen. Petrinetze erweisen sich dafür als handliches und wirksames Hilfsmittel, da das grundlegende Netz direkt aus dem Layout der Anlage gewonnen werden kann. Das so erhaltene Netz läßt sich erweitern, um einen verklemmungsfreien und fairen Betrieb zu gewährleisten. Dies betrifft in besonderem Maße die Paßstrecke. Nach erfolgreicher Simulation am Modell, lassen sich die Ergebnisse problemlos in Quellcode umsetzen.

Neben der rein in Software implementierten Lösung der organisatorischen Probleme, ist die Ansteuerung externer Hardware ein Bestandteil dieser Arbeit. Die Nutzung externer elektronischer und mechanischer Baugruppen führt zu einem sehr breiten Spektrum möglicher Fehler, die zu erkennen und geeignet zu behandeln sind. Dazu gehören z. B. die unsichere Betätigung der Reedkontakte oder Übertragungsfehler auf der seriellen Schnittstelle. Die Fehlersuche wird zusätzlich durch das oft nicht reproduzierbare Verhalten der Hardware erschwert, das u. a. zu unterschiedlichen Abläufen eines identischen Fahrplans führt. Deshalb ist eine umfangreiche Fehlerbehandlung erforderlich.

Der Einsatz des Interbus erweist sich als vorteilhaft. Nach anfänglichen Problemen während der Inbetriebnahme zeigt sich ein hohes Maß an Stabilität. Im Vergleich zu einem früheren Aufbau mit zentraler Verdrahtung ist der Interbus weniger fehleranfällig und wartungsintensiv.

Der Umfang der Arbeit und die Beteiligung von zwei Entwicklern macht eine ausgereifte Organisation und Verwaltung des Quellcodes notwendig. Dazu gehört eine klare Untergliederung des Codes, ohne die Fehlersuche und Erweiterung sehr erschwert werden. Die Modularisierung spiegelt dabei Teilbereiche der Aufgabenstellung wider, so wurde z. B. eine in sich abgeschlossene Bibliothek erstellt, die die Ansteuerung der Hardware realisiert, und ihrerseits wiederum in Klassen unterteilt ist. Um die Lesbarkeit des Quellcodes zu fördern, stellen sich u. a. einheitliche Richtlinien zur Benennung von Variablen und Methoden, sowie eine ausführliche Dokumentation als nützlich heraus.

Anstatt einer manuellen Versionsverwaltung kommt das CVS zum Einsatz. Auch wenn dieses Werkzeug eine gewisse Einarbeitung erfordert, lohnt sich der Einsatz, da viele häufig vollzogene Abläufe dadurch vereinfacht werden. Hinzu kommt eine Reduzierung der mit diesen Arbeitsschritten verbundenen Fehlern.

Weiterhin sind “autoconf”, “configure” und “automake” hilfreiche Mittel zur Erstellung eines ausführbaren Programms. Sie vereinfachen sowohl die Nutzung von Bibliotheken, als auch die Migration in andere Systemumgebungen.

Das erstellte Steuerprogramm löst die gestellten organisatorischen Probleme und ermöglicht einen geordneten Mehrzugbetrieb auf der Modellbahnanlage. Die grafische Oberfläche erlaubt dem Benutzer dabei eine komfortable Interaktion mit dem System.

Im Bereich der Benutzerschnittstelle sind vielfältige Verbesserungen wie z. B. Netzwerkanbindung und detailiertere Statistiken bzgl. des Ablaufs denkbar. Weitaus interessanter sind allerdings Erweiterungen im organisatorischen Bereich: So wäre es z. B. möglich, im Paß die Ausweichgleise für beide Richtungen freizugeben. Dazu wäre ein neues Petrinetzmodell zu entwerfen, das in das bestehende Steuerprogramm eingebunden werden könnte.

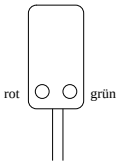
Das Hinzufügen weiterer Züge würde ebenfalls zusätzliche Maßnahmen zur Deadlockvermeidung erforderlich machen, da dann in einem oder mehreren Bahnhöfen kein freies Durchfahrtsgleis mehr zur Verfügung stünde. Wenn also alle Züge, die ihr Heimatgleis in einem solchen Bahnhof haben, ihren Fahrplan beendet haben, wäre es ohne zusätzliche Maßnahmen keinem Zug mehr möglich, diesen Bahnhof zu durchfahren.

Weiterhin könnten die Fahrpläne erweitert werden, indem den Zügen eine Ankunftszeit und Aufenthaltsdauer in den Bahnhöfen zugewiesen werden. Dieses würde neue Prioritätsregelungen auf Basis des zeitlichen Ablaufes erlauben, um Zügen mit Verspätungen die Möglichkeit zu geben, diese wieder aufzuholen.

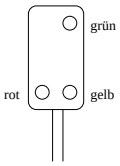
Anhang A

Lichtsignale

Auf der Modellbahnanlage werden zwei Lichtsignaltypen genutzt:



Lichtsignal mit den Farben rot und grün.



Lichtsignal mit den Farben rot, grün und gelb.

Beide Typen zeigen für einen vor dem Signal befindlichen Zug den Status des nachfolgenden Gleisabschnitts an. In Anlehnung an die bei der Deutsche Bahn AG herrschende Praxis, gibt es bei der Signalisierung folgende Farbkombinationen:

rot	Einfahrt in den folgenden Gleisabschnitt ist nicht gestattet
grün	Einfahrt in den folgenden Gleisabschnitt ist ohne Einschränkungen gestattet
grün + gelb	Einfahrt in den folgenden Gleisabschnitt ist nur mit langsamer Fahrt gestattet

Die Signalisierungen “rot” und “grün” sind bzgl. ihrer Bedeutung offensichtlich. Signale mit der zusätzlichen Farbe “gelb” kommen zum Einsatz, falls der nachfolgende Block mindestens eine Weiche enthält. Ist diese so gestellt, daß der betroffene Zug von der Strecke abzweigt, ist die Geschwindigkeit zu drosseln. Dieses wird mit der Farbe “gelb” signalisiert.

Anhang B

Daten verschiedener Feldbussysteme

Bussysteme kommen in verschiedenen Ebenen der Steuerungstechnik zum Einsatz. Der in dieser Arbeit genutzte Interbus gehört zu den Feldbussen. Dies sind Bussysteme, die die Verbindung von Sensoren und Aktoren auf dem sog. Produktionsfeld und einer übergeordneten Steuerungseinheit herstellen. Eine solche Steuerungseinheit kann dabei wiederum über Bussysteme der sog. Systembusebene an eine weitere übergeordnete Steuerungseinheit gekoppelt sein. Weitere solcher Steuerungsebenen sind dabei möglich. Durch diese Technik läßt sich eine klare Gliederung und Kapselung einzelner Bereiche einer gesamten Anlage erreichen. Außerdem ermöglicht dies räumlich sehr weit ausgedehnte Netze. Ein Beispiel für einen solchen Aufbau findet sich in [6].

Die in Tabelle B.1 genannten Bussysteme gehören alle zu den Feldbussen (vgl. [23]).

	Interbus	CAN[1]	ProFiBus[2]
Topologie	aktiver Ring	linearer Bus oder Stern	linearer Bus mit Token-Passing
Organisation	Single-Master/Slave	Multi-Master/Slave	Multi-Master/Slave
Bruttodatenrate	500 kBit/s oder 2 MBit/s	bis 1MBit/s	bis 1,5 MBit/s
Nettodatenrate	54%	bis zu 50 %	bis zu 80%
Zeitverhalten	synchron	asynchron	asynchron
Medien	Twisted-Pair, LWL[3], Infrarot	Twisted-Pair, LWL	Twisted-Pair, LWL

Tabelle B.1: Eckdaten häufig genutzter Feldbussysteme

Das Erreichen der Nettodatenrate garantiert dabei nur der Interbus. CAN und ProFiBus sind diesbezüglich abhängig vom Datenaufkommen und anderen Parametern. Dies begründet sich auch in der asynchronen Technik dieser Systeme. Aus diesem Grunde kommt der Interbus im Gegensatz zu CAN und ProFiBus bevorzugt im Produktionsfeld zum Einsatz. CAN und ProFiBus werden hingegen auch in übergeordneten Steuerungsebenen eingesetzt, da hier mehrere Master eingesetzt werden können.

[1] Controller Area Network
[2] Process Field Bus
[3] Licht-Wellen-Leiter

Das Spektrum der verschiedenen Feldbussysteme ist mittlerweile sehr groß und bietet sehr spezielle wie auch allgemein einsetzbare Lösungen. Auch die bereits bestehenden Feldbussysteme werden dabei noch weiterentwickelt. So gibt es z.B. für den Interbus mittlerweile noch eine Reihe von Ansätzen, weitere abzweigende Busse einzuhängen oder sog. "Intelligente Teilnehmer" zu nutzen, die mit einem speziellen Protokoll untereinander Daten austauschen können.

Anhang C

Illustrationen

Abbildung C.1: Übersicht der Modellbahnanlage

Die Abbildung C.1 zeigt einen Überblick der Modellbahnanlage. Auf der linken Seite sind die Bahnhöfe des Outer- und Inner-Circle zu sehen, auf der rechten Seite der Paßbahnhof.

Abbildung C.2: Platine mit zwei Interbus-Slaves

Die Abbildung C.2 zeigt eine der eingesetzten Platinen, die jeweils zwei Interbus-Slaves tragen. Diese sind die quadratischen ICs auf der linken Hälfte der Platine. Auf der rechten Hälfte befindet sich die Elektronik zur Ansteuerung der Ein- und Ausgänge, sowie die zugehörigen Anschlußleisten.

Abbildung C.3: Reedkontaktpaar

Die Abbildung C.3 zeigt eines der auf der Anlage eingesetzten Reedkontaktpaare.

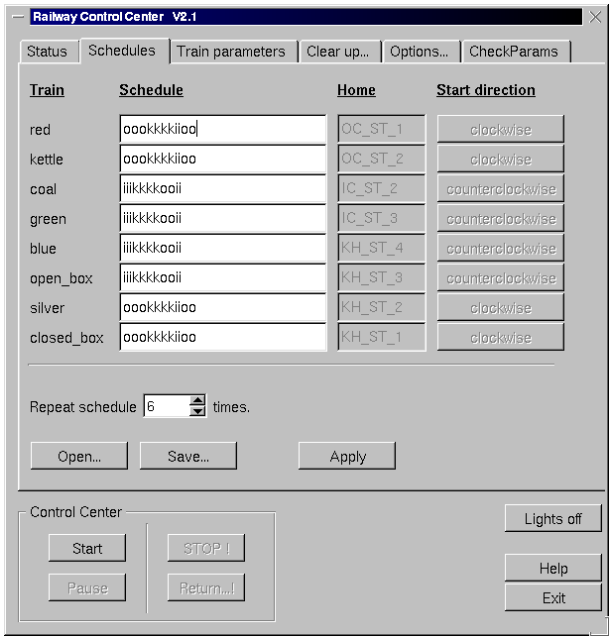


Abbildung C.4: Fahrplaneingabe der grafischen Oberfläche

Die Abbildung C.4 stellt die grafische Oberfläche zur Eingabe oder Auswahl eines vorhandenen Fahrplans dar. Die Fahrpläne ergeben sich durch Eingabe der anzufahrenden Bahnhöfe ('o', 'i' und 'k') und können für eine spätere Nutzung gespeichert werden.

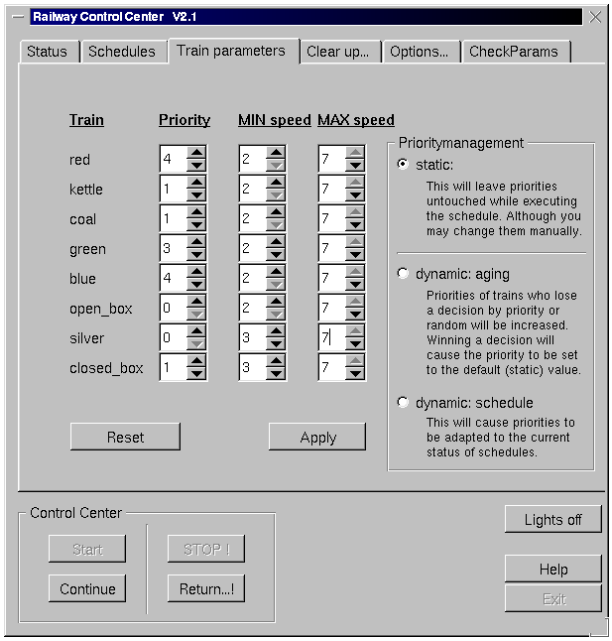


Abbildung C.5: Eingabe der Zugparameter

Die Abbildung C.5 zeigt die Optionen zur Zugbeeinflussung über die Vergabe von Geschwindigkeiten, Prioritäten und das verwendete Prioritätsverfahren.

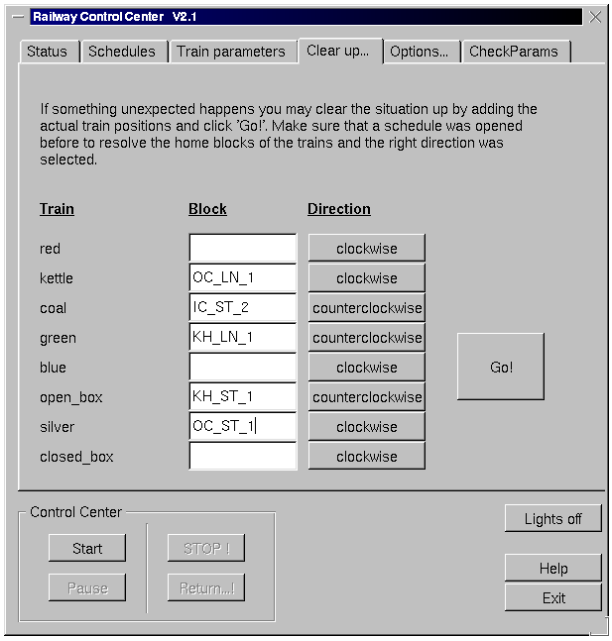


Abbildung C.6: Eingabemaske zur Rückführung von Zügen

Mit Hilfe der in der Abbildung C.6 dargestellten Eingabemaske können Züge von beliebigen Gleisabschnitten auf Ihre Heimatgleise zurückgeführt werden.

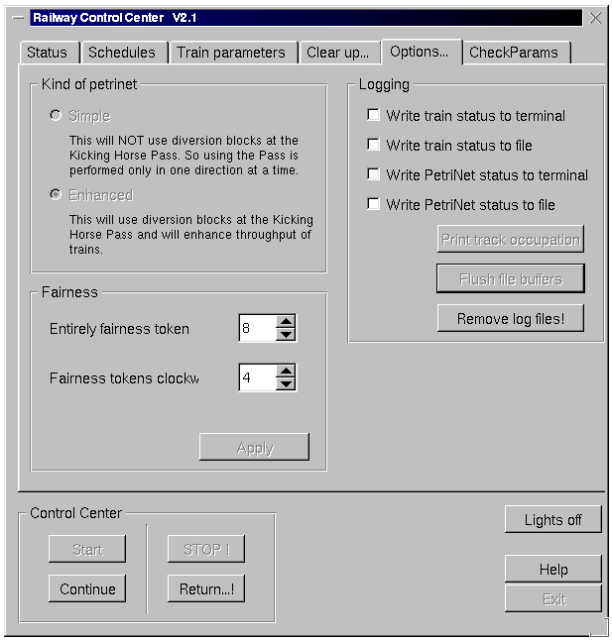


Abbildung C.7: Optionen zur Steuerung und Protokollierung

In der Abbildung C.7 werden die Optionen zur Fehlerprotokollierung und Paßbehandlung dargestellt. Dabei kann zwischen dem eingleisigen Paßmodell oder dem mit Ausweichgleis gewählt, sowie die maximale Anzahl in einer Richtung einfahrenden Züge festgelegt werden.

Literaturverzeichnis

[1] Martin Müller Alfredo Baginski. *InterBus, Grundlagen und Praxis.* Hüthig Verlag Heidelberg, 1998.

[2] Albert S. Woodhull Andrew S. Tanenbaum. *Operating Systems: Design and Implementation.* Prentice-Hall, 1997.

[3] Maurice Bach. *The Design of the Unix Operation System.* Prentice-Hall, 1986.

[4] M. Ben-Ari. *Principles of concurrent and distributed programming.* Prentice-Hall, Incorporated, 1990.

[5] Anton Betten. *Codierungstheorie : Konstruktion und Anwendung linearer Codes.* Springer, 1998.

[6] W. Kriesel; P. Gibas; M. Riedel; W. Blanke. *Feldbus als Mehrebenenkonzept, Messen, Steuern, Regeln (S. 150-153).* 1990.

[7] David R. Butenhof. *Programming with POSIX Threads.* Addison-Wesley, 1999.

[8] http://doc.trolltech.com/2.3/.

[9] http://pasc.opengroup.org/.

[10] http://www.camiresearch.com/Data_Com_Basics/RS232_standard.html.

[11] http://www.cvshome.org/.

[12] http://www.interbusclub.com/itc.

[13] http://www.sgi.com/tech/stl/.

[14] Grady Booch James Rumbaugh, Ivar Jacobson. *The Unified Modeling Language Reference Manual.* Addison-Wesley, 1998.

[15] Werner E. Kluge. *Organisation und Implementierung von Betriebssystemen (UNIX), Vorlesungsmitschrift.* 2001.

[16] Werner E. Kluge. *System-Modellierung mit Petrinetzen, Vorlesungsmitschrift.* 2001.

[17] Karl Franz Fogel Moshe Bar. *Open Source Development with CVS.* The Coriolis Group, 2001.

[18] C. A. Petri. *Kommunikation mit Automaten.* PhD thesis, Schriften des Instituts für instrumentelle Mathematik, Bonn, 1962.

[19] Wolfgang Reisig. *Petri Nets.* Springer-Verlag, 1985.

[20] Brian W. Kernighan; Dennis M. Ritchie. *The C programming language.* Prentice-Hall, Incorporated, 1988.

[21] Graham M. Seed. *An introduction to object-oriented programming in C++ : with applications in computer graphics.* Springer, 2001.

[22] W. Richard Stevens. *Advanced Programming in the Unix Environment.* Addison-Wesley, 1992.

[23] D. Telschow W. Kriesel, T. Heimbold. *Bustechnologien für die Automation.* Hüthig Verlag Heidelberg, 1998.

[24] www.gnu.org/software/autoconf/autoconf.html.

[25] www.gnu.org/software/automake/automake.html.