Bachelorarbeit

Jens Windmüller

Simulation einer industriellen Produktionsanlage unter dem Echtzeitbetriebssystem QNX

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Jens Windmüller

Simulation einer industriellen Produktionsanlage unter dem Echtzeitbetriebssystem QNX

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. rer. nat. Wolfgang Fohl
Zweitgutachter: Prof. Dr. Franz Korf

Abgegeben am 7. Februar 2008

Jens Windmüller

Thema der Bachelorarbeit
Simulation einer industriellen Produktionsanlage unter dem Echtzeitbetriebssystem QNX
Stichworte
Simulation Petri-Netze QNX Echtzeitbetriebssystem CAN industrielle Produktionsanlage Photon Modellbildung Steuerungssoftware
Kurzzusammenfassung
Im Rahmen der vorliegenden Bachelorarbeit wird eine Simulationsumgebung für eine Produktionsanlage aus dem Labor für Technische Informatik der HAW Hamburg entwickelt. Als Mittel der Modellbildung dient die Technik der Petri-Netze. Simuliert wird das Austauschen von Steuernachrichten, die Bewegungen der einzelnen Komponenten, die Produktion, der Transport und die Lagerung von Werkstücken, Ausnahme-, Kollisions- und Fehlerbehandlungen sowie die Interaktion mit dem Benutzer.

Jens Windmüller

Title of the paper
Simulation of a industrial production plant under the Real-time operating system QNX
Keywords
Simulation Petri-Nets QNX Real-time operating system CAN industrial production plant Photon modelling controllersoftware
Abstract
The ambition of this bachelor thesis is to develop a simulation software for a production plant from the laboratory for technical computer science of the HAW Hamburg. For the modelling the method of Petri-Nets is used. The Exchange of control messages, the movings of the several parts, the production, the transport and the storage of workpieces, exception- collision- and error handling, as well as the interaction with the user are simulated.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Vorwort

In der heutigen Industrie werden komplexe Produktions-, Förder- und Lageranlagen durch zunehmend kompliziertere und komplexere Software gesteuert. Um diese Steuerungssoftware weitgehend unabhängig von der jeweiligen Anlage entwickeln und möglichst umfangreich testen zu können, ist es meist sinnvoll und wichtig ein Modell der Anlage zur Verfügung zu haben. (2) (7)
Ein solches Modell ist eine Softwarsimulation. Sie bildet das Verhalten der Anlage, und soweit nötig, das Verhalten der Umwelt nach, und erlaubt es dem Entwickler einer Steuerungssoftware anhand dieses Modells zu entwickeln und zu testen. Es ist wünschenswert während der Entwicklung und des Tests dieser Software möglichst viele Szenarien des Anlagenverhaltens sowie ihrer Umwelt zu berücksichtigen. Meist können aber nicht alle dieser Szenarien ohne Aufwand oder Gefährdung der Anlage in der Realität herbeigeführt werden. Auch die genaue Reproduzierbarkeit von bestimmten Szenarien stellt in der Realität oft ein Problem dar. Die Vorteile einer Simulation gegenüber der realen Anlage liegen nun in der Möglichkeit verschiedenste Zustände der Anlage und seiner Umwelt ohne viel Aufwand gezielt herbeizuführen und anhand dessen die Steuerung zu entwickeln, zu testen und zu verbessern.
Liegen Fehler in einer Steuerungssoftware vor, ist es möglich diese in der Simulation zu erkennen, ohne dabei das Risiko einzugehen die Anlage zu beschädigen oder den Benutzer zu gefährden.
Weitere Gründe für die Simulation von Fertigungs-, Förder-, und Lagerungsanlagen können auch sein, dass Diese zum Zeitpunkt der Entwicklung der Steuerung noch nicht, oder noch nicht vollständig existieren, dass die durchzuführenden Tests zu gefährlich oder zu teuer wären, oder dass vor einem Umbau einer Anlage das daraus resultierende Verhalten vorher untersucht werden soll. (12)
Natürlich ersetzt eine Simulation niemals die Notwendigkeit die zu entwickelnde Software auch an der realen Anlage zu testen, bietet aber eine hervorragende Möglichkeit im Vor-

feld vorhandene Fehler zu finden und zu beseitigen, um das Risiko eines Fehlverhaltens der Steuerungssoftware im späteren Betrieb zu minimieren.
Zu beachten ist, dass es neben diesen Gründen für die Simulation, die aus geringeren Kosten, Reproduzierbarkeit von Experimenten, dem Ausschließen von Risiken, der Vorhersage, der Veränderbarkeit physischer Parameter, der Realisierung noch nicht bestehender Systeme und der Voroptimierung bestehen, Simulationen auch ihre Grenzen und Probleme haben. So können Simulationen niemals besser sein als die realen Systeme auf denen sie beruhen. Desweiteren sind Simulationen immer von der Gefahr umgeben relevante Effekte unberücksichtigt zu lassen, oder als unwichtig erachtete Variablen zu vernachlässigen. (4)

1.2 Ziele

Diese Bachelor-Arbeit hat das Ziel eine Simulationsumgebung für das Modell einer industriellen Produktionsanlage aus dem Labor der Hochschule für Angewandte Wissenschaften Hamburg zu entwickeln.
Der Benutzer soll mit der Simulation in der Lage sein eine Steuerung für die genannte Anlage zu entwickeln, ohne auf die Hardware angewiesen zu sein. Der Benutzer soll seine Software an dieser Simulation entwickeln und testen können, und sie dann ohne Veränderungen auch an der Hardware benutzen können.
Es wird notwendig sein die vorhandene CAN-Schnittstelle zwischen Anlage und Steuerung zu ersetzen, damit eine Kommunikation mit der Simulationsumgebung stattfinden kann. Andere Änderungen sollen in der Steuerungssoftware nach Möglichkeit nicht vorgenommen werden.
Die Simulation soll eine grafische Benutzeroberfläche bereitstellen, die das Anlagenverhalten nachstellen wird. Diese GUI soll dem Benutzer genau vermitteln, in welchem Zustand sich die Anlage gerade befindet, und soll Sensoren, Aktuatoren und eingehende und ausgehende Nachrichten darstellen.

Kapitel 2

Analyse

2.1 Vorstellung der Anlage

Bei der in dieser Bachelorarbeit behandelten Anlage handelt es sich um ein Modell einer industriellen Produktionsanlage, welche über die Auswertung von Sensordaten und Ansteuerung von verschiedenen Aktuatoren gesteuert wird.
Der Aufbau der Anlage lässt sich in vier Einheiten aufteilen, da diese weitgehend unabhängig voneinander operieren: das Hochregal (RFZ), den Kran (KRN), die Presse (PRS) und die Identifizierungseinheit (IE).
Mit der Anlage ist es möglich würfelförmige Werkstücke aus einer Unter- und Oberschale zusammenzusetzen und in einem Regalsystem einzulagern. Die Unterschalen bestehen dabei aus einem metallischen Material und sind an der Oberseite eingekerbt, die Oberschalen bestehen dagegen aus Kunststoff und sind in den Farben schwarz und weiß vorhanden. Unter- und Oberschalen werden in zwei verschiedenen Magazinen gelagert und können von dort auf ein Förderband gelangen. An diesem Förderband sind verschiedene Sensoren zur Identifikation der Beschaffenheit der Werkstücke vorhanden. Vom Förderband können die Werkstücke vom Kran abgeholt werden.
Der Kran ist in der Lage die unterschiedlichen Werkstücke bei den Einheiten RFZ, PRS und IE abzuholen bzw. abzuliefern.
Befinden sich jeweils eine Unterschale und eine Oberschale in der richtigen Reihenfolge in der Presse können diese Werkstücke dort zu einem Würfel gepresst werden.
Das Regallager schließlich ist dafür zuständig die Werkstücke ein- oder zwischenzulagern.

2.1.1 Identifizierungseinheit (IE)

2.1.1.1 Aufbau

Die Identifizierungseinheit besteht im Wesentlichen aus zwei Magazinen, in denen die Unterschalen bzw. Oberschalen gelagert werden, aus einem Förderband, mit dem sich die Werkstücke durch die IE bewegen lassen, einer Reihe von Sensoren, die die Materialeigenschaften der Werkstücke identifizieren können, sowie einer Lichtschranke am Ende des Förderbandes, die erkenntlich macht, wann ein Werkstück die IE komplett durchlaufen hat und zur Abholung durch den Kran bereit ist.
Die Magazine für die Werkstücke werden manuell vom Benutzer der Anlage bestückt.

2.1.1.2 Sensoren

Folgende Sensoren sind an der Identifizierungseinheit vorhanden:

- Je ein Taster in jedem der beiden Magazine, wodurch die Befüllung mit Werkstücken erkannt wird.
- Je ein Sensor in jedem der beiden Magazine, welcher anspricht sobald die Ausschubvorrichtung ausgefahren ist.
- Je ein Sensor in jedem der beiden Magazine, welcher anspricht sobald die Ausschubvorrichtung eingefahren ist.
- Ein Kontursensor, bestehend aus einem ausfahrbaren Stempel, mit dessen Hilfe festgestellt werden kann ob es sich bei dem zu prüfenden Werkstück um eine Ober- oder Unterschale handelt. (Unterschalen haben eine Einkerbung an der Oberseite, wodurch sich der Stempel weiter absenken lässt als bei den Oberschalen.)
- Ein Helligkeitssensor, der bei schwarzen Werkstücken anspricht, bei weißen und metallischen aber nicht.
- Ein Metallsensor, der bei metallischen Werkstücken anspricht.
- Eine Lichtschranke am Ende des Förderbandes, die das Erreichen und Verlassen des Endes des Förderbandes durch Werkstücke signalisiert.

2.1.1.3 Aktuatoren

Über folgende Aktuatoren kann die Identifizierungseinheit gesteuert werden:

- Je eine Ausschubvorrichtung in jedem der beiden Magazine, um Werkstücke vom Magazin auf das Förderband zu befördern.

- Eine Ausfahrvorichung für den Kontur Sensor.
- Ein Aktuator um das Band links herum laufen zu lassen.
- Ein Aktuator um das Band rechts herum laufen zu lassen.
- Ein Aktuator um das Band zu stoppen.

2.1.2 Kran (KRN)

2.1.2.1 Aufbau

Der Kran besteht aus einer drehbaren und vertikal beweglichen Basis, einem horizontal beweglichen Ausleger und einem Sauger der sich am äußeren Ende des Auslegers befindet. Der Kran kann somit Werkstücke, durch Erzeugung eines Unterdrucks am Sauger, aufnehmen und wieder absetzen.

2.1.2.2 Sensoren

Die folgenden Sensoren sind am Kran montiert:

- Positionssensoren für die horizontale Positionsbestimmung jeweils beim RFZ, der Presse und der Identifizierungseinheit. Jeweils beim Verlassen und Erreichen des Sensors wird ein Sensorsignal gesendet.
- Ein Taster, der anspricht sobald der Kran gehoben ist. Sowohl bei der Aktivierung als auch bei der Deaktivierung des Tasters wird ein Sensorsignal gesendet.
- Ein Taster, der anspricht sobald der Kran gesenkt ist. Sowohl bei der Aktivierung als auch bei der Deaktivierung des Tasters wird ein Sensorsignal gesendet.
- Ein Sensor, der anspricht sobald der Ausleger des Krans ausgefahren ist.
- Ein Sensor, der anspricht sobald der Ausleger des Krans eingefahren ist.

2.1.2.3 Aktuatoren

Über folgende Aktuatoren wird der Kran gesteuert:

- Ein Aktuator zum Heben des Krans.
- Ein Aktuator zum Senken des Krans.
- Ein Aktuator zum Einschalten des Saugers.

- Ein Aktuator zum Ausschalten des Saugers.
- Jeweils ein Aktuator um den Ausleger ein- bzw. auszufahren.
- Ein Aktuator um die horizontale Kreisbewegung des Krans im Uhrzeigersinn anzustoßen.
- Ein Aktuator um die Kreisbewegung des Krans gegen den Uhrzeigersinn anzustoßen.
- Ein Aktuator um die Kreisbewegung des Krans zu stoppen.

2.1.3 Presse (PRS)

2.1.3.1 Aufbau

Die Presse besteht aus einem Ausschub, welcher die Werkstücke in das Innere der Presse und wieder heraus transportieren kann, einer Sicherheitstür, welche die Presse abriegeln kann, und einem Stempel im Innern der Presse, der den eigentlichen Pressvorgang durchführt.

2.1.3.2 Sensoren

Die Presse besitzt nur die folgenden Sensoren zur Zustandsbestimmung:

- Jeweils ein Sensor, der anspricht sobald die Sicherheitstür geöffnet bzw. geschlossen ist.
- Jeweils ein Sensor, der anspricht sobald der Ausschub ausgefahren bzw. eingefahren ist.

2.1.3.3 Aktuatoren

Folgende Aktuatoren ermöglichen das Steuern der Presse:

- Jeweils ein Aktuator zum Öffnen bzw. Schließen der Sicherheitstür.
- Jeweils ein Aktuator zum Ausfahren bzw. Einfahren des Ausschubs.
- Jeweils ein Aktuator zum Senken bzw. Heben des Stempels.

2.1.4 Regallager (RFZ)

2.1.4.1 Aufbau

Das RFZ besteht aus einem unbeweglichen Regal mit vier Regalböden, von denen jeder sieben Lagerplätze für Werkstücke bereit hält. Weiterhin enthält das RFZ eine vor dem Regal horizontal bewegliche Stange, an der eine Gabel befestigt ist, welche sich vertikal an dieser Stange hoch und runter bewegen kann. Zudem kann die Gabel ausgefahren werden, um die Regalböden zu erreichen und gegebenenfalls ein Werkstück dort einzulagern.

2.1.4.2 Sensoren

Diese Sensoren stehen dem RFZ zur Verfügung:

- Ein Sensor für die horizontale Positionsbestimmung. Dieser Sensor spricht, mit Hilfe einer Lichtschranke und jeweils einem Loch pro Einlagerungsplatz in der Laufschiene, an jedem Lagerplatz an. Dadurch sendet er jeweils wenn er aktiviert und deaktiviert wird ein Signal.
- Insgesamt vier Sensoren für die vertikale Positionsbestimmung. Auf der Höhe jedes Regalbodens befindet sich ein solcher Sensor, der ein Signal sendet wenn er aktiviert oder deaktiviert wird.
- Jeweils ein Endschalter oben, unten, links und rechts am Regal, welche beim Erreichen und Verlassen der jeweiligen Endposition ein Signal Senden.
- Jeweils ein Sensor, der anspricht sobald die Gabel ausgefahren bzw. eingefahren ist.

2.1.4.3 Aktuatoren

Die Steuerung des RFZ erfolgt über diese Aktuatoren:

- Ein Aktuator für die horizontale Bewegung nach links.
- Ein Aktuator für die horizontale Bewegung nach rechts.
- Ein Aktuator um die horizontale Bewegung zu stoppen.
- Ein Aktuator für die vertikale Bewegung nach oben.
- Ein Aktuator für die vertikale Bewegung nach unten.
- Ein Aktuator um die vertikale Bewegung zu stoppen.
- Jeweils ein Aktuator um die Gabel einzufahren bzw. auszufahren.

2.1.5 CAN-Interface

Die Anlage ist über einen CAN-Bus mit einem Steuerungsrechner, auf dem das Echtzeitbetriebssystem QNX läuft, verbunden. Über dieses Interface können die von der Anlage ausgelösten Sensorereignisse empfangen, und Befehle an die Aktuatoren der Anlage übermittelt werden.

2.2 Vorstellung der Steuerungssoftware

Für die Steuerung der Anlage ist eine Software vorhanden, die unter QNX läuft. Diese Software beinhaltet einen CAN-Treiber, der die Kommunikation mit der Anlage übernimmt, die vom CAN-Bus empfangenen Daten aufarbeitet und diese an eine höhere Softwareschicht weiterleitet. Weiterhin werden vom CAN-Treiber Nachrichten aus dieser höheren Softwareschicht entgegen genommen, an die Spezifikationen des CAN-Protokolls angepasst und über den CAN-Bus an die Anlage verschickt.
Die angesprochene höhere Softwareschicht besteht aus einer grafischen Benutzerschnittstelle (GUI), die es ermöglicht einfache Steuerbefehle an die Anlage zu senden.

2.3 Anforderung an die Simulationsumgebung

Die in dieser Bachelorarbeit zu erstellende Simulationssoftware soll es dem Entwickler und Benutzer der Steuerungssoftware ermöglichen diese zu entwickeln, zu testen und zu benutzen, ohne dafür an die Hardware gebunden zu sein. Dafür muss die Simulation eine Reihe von Anforderungen erfüllen, die hier im Weiteren genauer erläutert werden. Außerdem besitzt die Software eine API, die Funktionalitäten bereitstellt um eine eine automatische Steuerung für die Anlage entwickeln zu können.

2.3.1 Funktionale Anforderungen

Die Simulationssoftware soll in der Lage sein annähernd alle möglichen Benutzungsszenarien, die mit der realen Anlage möglich sind, nachzustellen. Dazu gehören insbesondere die unabhängige Steuerung der einzelnen Anlagenteile RFZ, KRN, PRS und IE und die Funktionalitäten an den Übergabepunkten zwischen RFZ und KRN, zwischen PRS und KRN und zwischen IE und KRN, an denen Werkstücke von einem Teil der Anlage an einen Anderen übergeben werden können.
Der Transport, die Verarbeitung und die Einlagerung der Werkstücke soll so exakt wie möglich nachgestellt werden, da dies die Kernaufgaben der Anlage sind.
Die Simulation soll weiterhin eine Kollisionsbehandlung beinhalten, die erkennt, ob eine Kollision zwischen zwei verschiedenen Teilen der Anlage stattgefunden hat, ob ein Werkstück

vom Förderband, vom Regal oder vom Kran gefallen ist, oder ob eine Kollision innerhalb eines Anlagenteils erfolgt ist. Letztgenanntes ist beim RFZ und in der Presse möglich und wird später genauer behandelt.
Außerdem ist es das Ziel dieser Bachelorarbeit dem Benutzer eine ansprechende, funktionale und anschauliche Benutzerschnittstelle zur Verfügung zu stellen, die die Simulation bedienbar und nachvollziehbar macht. Diese Oberfläche soll zu jedem Zeitpunkt eindeutig darstellen, welchen Zustand die Simulation hat, welche Sensoren und Aktuatoren aktiviert sind und welche Nachrichten zuletzt, über die Schnittstelle zur Steuerungssoftware gesendet, und empfangen wurden. Es sollen also alle Informationen, die an der realen Anlage abrufbar sind, und darüber hinaus weitergehende Informationen, die für die Entwicklung einer Steuerung für die Anlage sinnvoll sind, dargestellt werden.
Die Simulationsumgebung soll als eigenständige Software laufen und nicht in die vorhandene Steuerungssoftware integriert werden. Die Kommunikation dieser beiden Programme wird über ein geeignetes Interface stattfinden.

2.3.2 Zeitliche Anforderungen

Die Simulation muss *nicht* die genauen zeitlichen Eigenschaften und Randbedingungen der realen Anlage nachbilden. Da alle Funktionen der Anlage reaktiv sind (d.h. auf der Reaktion auf Sonsorereignisse basieren), reicht es, wenn die Simulation so modelliert wird, dass sie die ungefähren zeitlichen Randbedingungen einhält.
Die Simulationsumgebung muss aber sicherstellen, dass empfange Nachrichten in der zeitlich richtigen Reihenfolge verarbeitet werden, und zu sendende Ereignisse in der richtigen Reihenfolge gesendet werden.

2.3.3 Integration in die bestehende Steuerungssoftware

Da die vorhandene Software so wenig wie möglich modifiziert werden soll, wird in Diese lediglich eine Kommunikationsschnittstelle zur Simulationssoftware eingebaut. Wie die Schnittstelle aufgebaut ist wird im folgenden Kapitel erläutert.

Kapitel 3

Design

3.1 Allgemeines zur Simulation und Modellbildung

Durch den technischen und wissenschaftlichen Fortschritt werden immer komplexere Techniken, Verfahren und Maschinen entwickelt und eingesetzt. Die erforderlichen Anlaufinvestitionen steigen, bevor ein Nutzen zu erzielen ist. Dem Entwicklungs- bzw. Entwurfsprozeß kommt demnach eine hohe Bedeutung zu. Dabei sind Aussagen über das spätere Verhalten eines geplanten Systems für fundierte Entscheidungen notwendig. Neben Fragen nach der korrekten Funktionsweise sind besonders Aussagen über die Leistungsfähigkeit und Zuverlässigkeit interessant. Ein wichtiges Hilfsmittel zur Lösung der genannten Probleme stellt die Modellierung und Untersuchung der geplanten Systeme dar. Ein Modell kann außerdem für den Informationsaustausch zwischen den Entwurfsebenen eines Fertigungssystems verwendet werden. (13)
Simulation ist die Nachahmung realer Prozesse mittels eines Computers auf der Grundlage eines Modells. Das Erstellen eines solchen Modells wird als „Modellbildung“ bezeichnet. Ein Modell im Sinne der Modellbildung ist ein abstraktes Abbild eines Systems, welches stellvertretend für das System untersucht wird. Man unterscheidet die strukturelle und die pragmatische Modellbildung.
Bei struktureller Modellbildung ist die innere Struktur des Systems bekannt, es wird bewusst abstrahiert, modifiziert und reduziert - hier spricht man vom Whitebox-Modell.
Bei pragmatischer Modellbildung ist die innere Struktur des Systems unbekannt, es lässt sich nur das Verhalten bzw. die Interaktion des Systems beobachten und modellieren. Die Hintergründe lassen sich meist nicht oder nur zum Teil verstehen - hier spricht man vom Blackbox-Modell.
Da die Struktur des in dieser Bachelorarbeit behandelten Systems komplett bekannt ist, handelt es sich hierbei um eine strukturelle Modellbildung.(11) (4) (1)

3.2 Schnittstelle in der Steuerungssoftware

Wie bereits erwähnt kommuniziert die Steuerung über einen CAN-Bus mit der Anlage. Jedes Sensorereignis und jeder Steuerbefehl wird durch einen eigenen Code repräsentiert, der entweder von der Anlage oder von der Steuerung über den CAN-Bus gesendet wird. Für die Kommunikation zwischen der Simulationsumgebung und der Steuerungssoftware soll der CAN-Treiber um eine Softwareschnittstelle erweitert werden. Dabei sollen nur die Programmteile erweitert werden, die direkt für die Kommunikation mit der Hardware verantwortlich sind. Die Nachrichten die über diese Schnittstelle gesendet werden, sowie die Schnittstelle zur höheren Softwareschicht der Steuerungssoftware bleiben davon unberührt. Diese Softwareschnittstelle wird mit jeweils einer Posix Message Queue für die Sende- und die Empfangsrichtung realisiert werden. Dabei wird der vorhandene CAN-Treiber nicht ersetzt, sondern um diese Funktionalität erweitert. Der Benutzer soll dann zur Laufzeit der Steuerungssoftware entscheiden können, ob er das CAN-Interface oder die Simulationsschnittstelle laden möchte.

3.3 Auswahl der Modellierungsmethode

Zur Modellierung einer Software Simulation bieten sich mehrere Möglichkeiten an. In dieser Bachelorarbeit wird das Konzept der Petri-Netze verwendet werden.
Petri-Netze sind gut für die Beschreibung von Systemen geeignet, in denen konkurrierende, synchrone, verteilte und parallele Vorgänge auftreten. Sie vereinen die Beschreibung von Aktivitäten und Zuständen eines Systems. Auch komplexe Systeme mit relativ großem Zustandsraum lassen sich mit ihnen übersichtlich und korrekt wiedergeben. Durch die lokale Beschreibung von Aktivitäten und ihres Einflusses auf den Zustand sind sie für Verfeinerung und modulare Konstruktion eines Modells geeignet. (13)
Ein Petri-Netz ist ein Graph, der aus Plätzen, Transitionen und gerichteten Kanten besteht. Die Plätze bilden die jeweiligen Zustände des zu modellierenden Systems ab, die Transitionen stellen Übergänge zwischen diesen Plätzen dar. An Transitionen können Bedingungen gekoppelt sein, unter welchen diese „schaltet“ und an Plätzen können Aktionen ausgeführt werden. Durch Kanten werden Transitionen mit Plätzen und umgekehrt verbunden. Zur Darstellung dynamischer Aspekte verfügt das Petri-Netz über ein Markierungskonzept: mindestens ein Platz im Netz enthält eine „Marke“. Diese Marke wandert nach definierten Regeln über die gerichteten Kanten durchs Netz. Die in dieser Arbeit verwendete Regeln sind lediglich Bedingungen, die an die Transitionen gebunden sind. Es gibt noch weitere Möglichkeiten solche Bedingungen zu formulieren, diese werden aber hier nicht weiter behandelt.(5)
Weiterhin sind die in dieser Arbeit verwendeten Petri-Netze so einfach wie möglich gehalten, es handelt sich zum größten Teil um Zustandsgraphen. Das bedeutet dass jede Transition

im Netz genau einen Vorplatz und höchstens einen Nachplatz hat, was den Vorteil hat, dass die Netze übersichtlich bleiben, und die spätere Umsetzung in Code einfacher ist.
Folgendes Beispiel soll kurz die Grundlegende Funktion eines sehr einfachen Petri-Netzes darlegen:

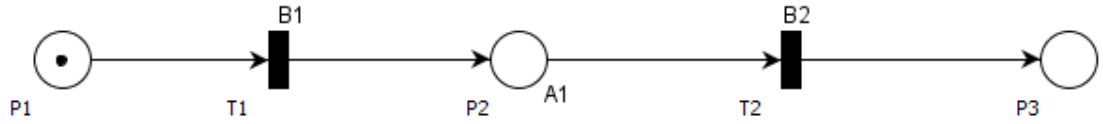


Abbildung 3.1: Beispiel Petri-Netz

In diesem Beispiel ist ein Netz, bestehend aus 3 Plätzen P1, P2, und P3, und zwei Transition T1 und T2 an welche die Bedingungen B1 und B2 gekoppelt sind, zu sehen. An den Platz P2 ist die Aktion A1 gekoppelt. Die Aktuelle Markierung des Netzes ist P1. Ist nun die Bedingung B1 erfüllt, so schaltet die Transition T1, die Marke wird vom Platz P1 entfernt, Platz P2 wird markiert, und die Aktion A1 wird ausgeführt. Wenn nun im nächsten Schritt die Bedingung B2 erfüllt ist, so wandert die Marke auf die gleiche Art weiter zu P3.
Bei diesem Netz handelt es sich um eine spezielle Form eines Petri-Netzes, nämlich um einen Zustandsgraphen. Das bedeutet dass jede einzelne Transition genau einen Vorplatz und höchstens einen Nachplatz hat. Auch alle Netze für die Modellierung der Anlage werden Zustandsgraphen sein.

3.4 Modellierung der Simulationsumgebung

Für die Modellierung der Petri-Netze kommt das Tool PIPE2: Platform Independent Petri Net Editor 2.4 zum Einsatz, welches neben der Editorfunktion auch die Möglichkeit bietet die erstellten Netze als PNG oder PostScrip Bilder zu exportieren.

3.4.1 Grobdesign

Das Design der Simulation besteht aus einem Message Handler für die Kommunikation mit der Steuerungssoftware, aus jeweils einer Manager Routine für jeden Teilbereich der Anlage (RFZ, KRN, PRS, IE), und jeweils einem Worker-Thread für jeden dieser Teilbereiche. Außerdem gibt es einen zentralen Datenbereich für die internen Zustände der Sensoren

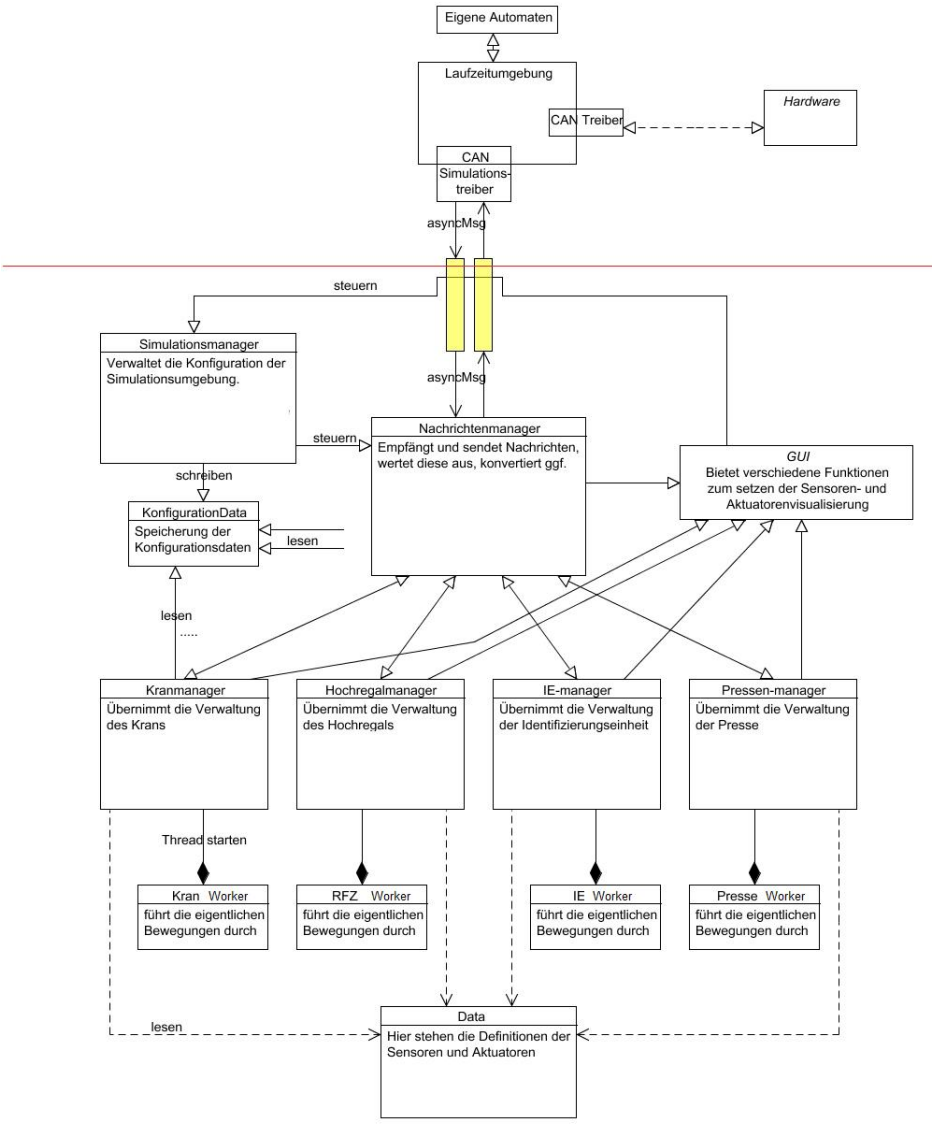


Abbildung 3.2: Struktur des Grobdesigns

und Aktuatoren der Simulation, sowie einen Simulationsmanager für die Verwaltung der Konfigurationsdaten und einer GUI Komponente. (Abb. 3.2)

Das Petri-Netz in Abbildung 3.3 stellt den übergeordneten Ablauf des Systems dar. Es stellt im Wesentlichen den Message Handler dar, der in der Startmarkierung einen blockierenden *mq_receive()* Aufruf macht, also auf eine Nachricht von der Steuerung wartet. Triff eine solche Nachricht ein, so wechselt das Netz in den Zustand *P1*. Von dort aus führen vier Transitionen weg. Damit wird erkannt, für welchen Teil der Anlage die Nachricht bestimmt ist. Abhängig vom Folgezustand wird der entsprechende Manager gestartet, erhält also

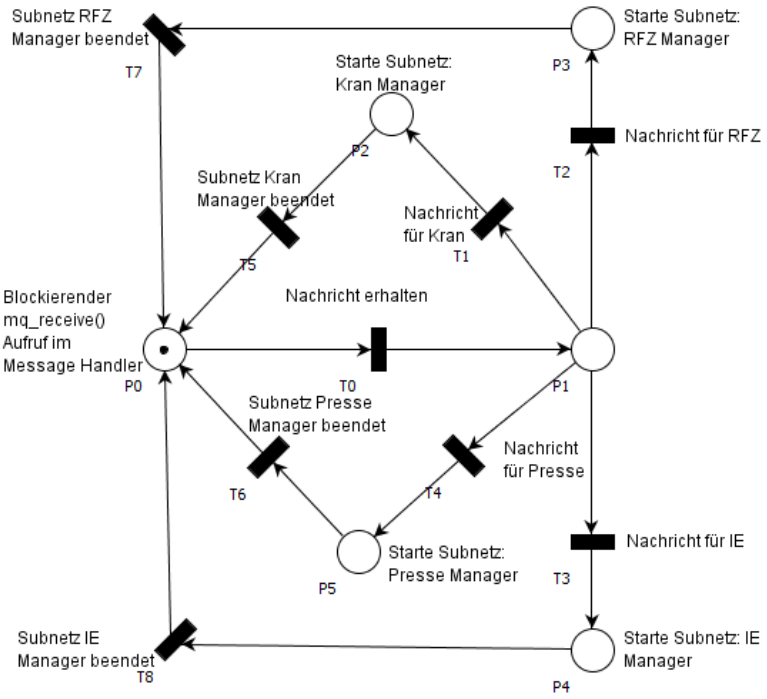


Abbildung 3.3: Petri-Netz: Gesamtsystem

die Marke, der die Nachricht weiterverarbeiten muss. Ist dieser Manager beendet kehrt die Marke in dieses Netz zurück, und das Netz kehrt in seinen Ausgangszustand zurück.

3.4.2 Datenstruktur der Nachrichten

Da von der Steuerungssoftware nur Nachrichten in Form von CAN-Codes gesendet werden, ist es sinnvoll Diese aufzuarbeiten. Zu diesem Zweck wird eine Datenstruktur geschaffen, die den empfangen Code als ID verwendet, den Teil der Anlage speichert, für den die Nachricht bestimmt ist, und eine Beschreibung der Nachricht enthält.
Diese Datenstruktur wird beim Programmstart geladen werden, und während der gesamten Laufzeit unverändert bleiben.

3.4.3 Schnittstelle zur Steuerungssoftware / Message Handler

Die Schnittstelle zur Steuerungssoftware besteht aus zwei Posix Message Queues. Der Message Handler der Simulation wird daher aus einem Empfags-Thread sowie einer Senderoutine bestehen, um die Queues zu bedienen.
Der Empfangs-Thread hat die Aufgabe alle eingehenden Nachrichten zu analysieren und anhand der beim Programmstart geladenen Datenstruktur, welche alle Nachrichten der Simulation enthält, zu identifiziert. Anschließend wird die Nachricht an den zuständigen Manager übergeben. Die Senderoutine ist nur dafür zuständig die ID der Nachricht über die Message Queue zu versenden.

3.4.4 Abbildung der Bewegungen auf das Modell

RFZ Die Bewegung des RFZ wird innerhalb eines 2 dimensionalen Koordinatensystems stattfinden. Die Dimension umfasst in horizontaler Ebene 40 Schritte, und in vertikaler Ebene 80 Schritte. Das Ausfahren der Gabel erfolgt in 4 Schritten, um auch das nicht vollständige Ausfahren simulieren zu können.

Kran Die Drehbewegung des Krans wird als Kreis abgebildet. Der gesamte Kreis umfasst 360 Schritte, von denen aber nicht alle vom Kran erreicht werden können, da dies in der Hardware aufgrund von am Kran verlaufenden Kabeln nicht möglich ist. Der Bewegungsfreiraum liegt zwischen den Schritten 60 und 300. Das Anheben bzw. Absenken des Krans sowie das Ausfahren und Einfahren des Auslegers erfolgt in jeweils 3 Schritten.

Presse Die Bewegungen der Presse, also das Aus- und Einfahren des Ausschubes, das Öffnen und Schließen der Sicherheitstür und das Senken und Heben des Stempels, erfolgen ohne Zwischenschritte, da es nicht möglich ist eine der Bewegungen nur halb durchzuführen.

Identifizierungseinheit Die Bewegung des Förderbandes wird nicht direkt simuliert, da sich dadurch keine Veränderung im Zustand der Simulation bzw. der Anlage ergibt. Liegt ein Werkstück auf dem Band wird direkt dessen Position verändert, es entsteht dadurch der Anschein einer Bewegung des Bandes, der auch völlig ausreichend für die Zwecke der Simulation ist.

3.4.5 Modellierung der Manager

Jeder der vier Manager wird durch ein eigenes Petri-Netz repräsentiert. Bei diesen Netzen handelt es sich um Subnetze, zu dem Netz aus Abb. 3.3, und werden von dort aus gestartet. Die Manager sind dafür zuständig die Vorbedingungen für den Start der Worker-Threads zu

überprüfen, Statusvariablen zu setzen und die Worker-Threads zu starten, welche als Subnetze der Manager modelliert sind. Das Manager Netz generiert beim Start eines Workers eine Marke, welche in den Startzustand des Workers gesetzt wird. Anschließend beendet sich der Manager und gibt seine Marke an das übergeordnete Netz zurück.

3.4.5.1 RFZ Manager

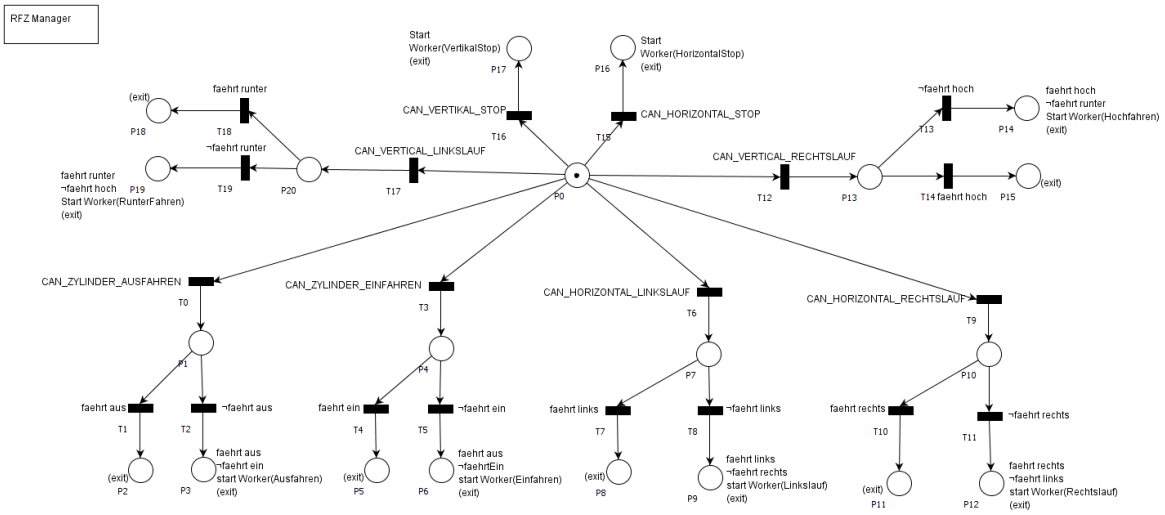


Abbildung 3.4: Petri-Netz: Hochregallager - Manager

Der Platz mit der Startmarkierung befindet sich in der Mitte des Netzes. Wie deutlich zu erkennen ist führen eine Vielzahl von Kanten von diesem Platz aus zu verschiedenen Transitionen. Jede dieser Transitionen repräsentiert eine der Nachrichten, welche die Befehle an die Aktuatoren der Anlage darstellen. Beispielhaft soll hier das Verhalten des Managers beim Empfang einer Nachricht erläutert werden.
Wird dem Manager die Nachricht *CAN_ZYLINDER_AUSFAHREN* übergeben, schaltet die Transition *T0* und die Markierung wandert zum Platz *P1*. Sollte nun die Anlage bereits im Ausfahren begriffen sein (weil vorher schon die selbe Nachricht empfangen wurde), wird die Transition *T1* schalten und die Markierung zum Platz *P2* wandern, wo das Petri-Netz endet, was übertragen auf die später zu implementierende Funktion einem return entspricht. Wenn die Anlage allerdings noch nicht im Ausfahren begriffen ist, dann schaltet die Transition *T2*, welche dafür sorgt, dass die Markierung zum Platz *P3* wandert. Dort werden die entsprechenden Statusvariablen gesetzt, und ein Workerthread wird gestartet.
Für fast alle anderen Nachrichten, die die Aktuatoren des RFZ betreffen, ist der Ablauf der selbe. Nur die beiden Nachrichten *CAN_VERTICAL_STOP* und *CAN_HORIZONTAL_STOP* bilden eine Ausnahme, da in diesen beiden Fällen eine Überprüfung, ob die Aktion bereits

ausgeführt wird, überflüssig ist. In diesen Fällen wird direkt der Workerthread gestartet. Der Manager ist also dafür zuständig zu erkennen, um welche Nachricht es sich handelt, den aktuellen Simulationsstatus bezüglich der angeforderten Aktion zu überprüfen und anhand dessen zu entscheiden ob die Aktion ausgeführt werden soll, oder nicht. (Abb. 3.4)

3.4.5.2 Kran Manager

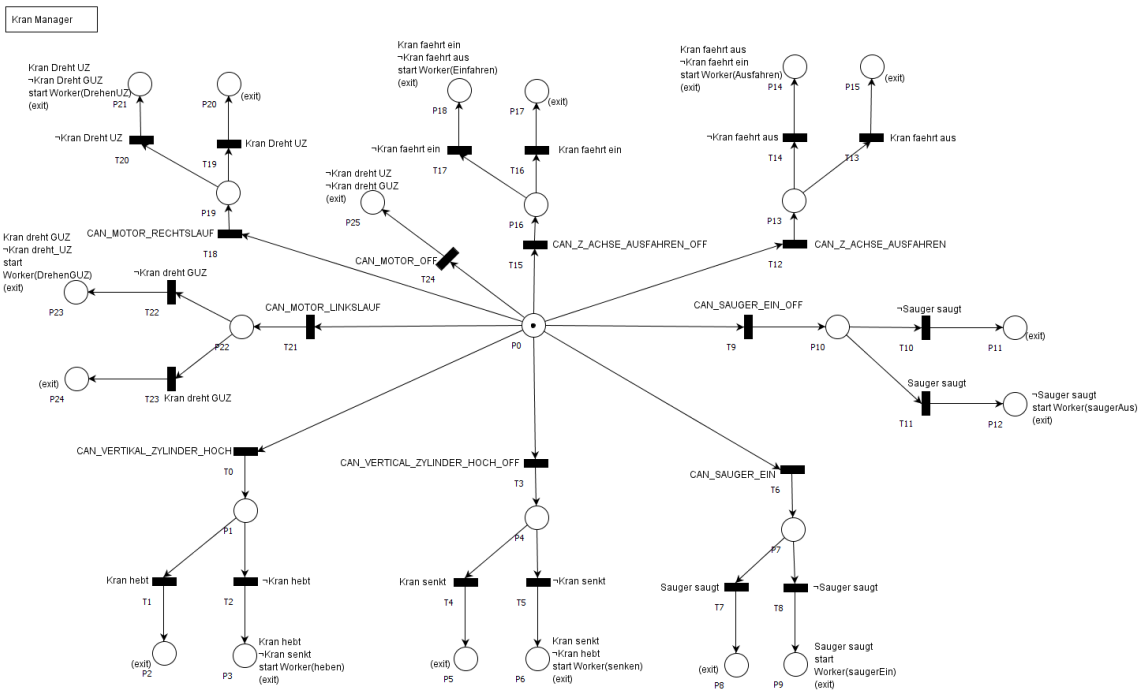


Abbildung 3.5: Petri-Netz: Kran - Manager

Die Manager Funktion des Krans ist so aufgebaut, wie die des RFZ. Wieder repräsentieren die Transitionen, die von der Startmarkierung *P0* wegführen, die Befehle, die an die Aktuatoren des Krans gesendet werden können. Nach dem Schalten der jeweiligen Transition, die für eine Nachricht zuständig ist, kommt eine Prüfung, ob der gewünschte Auftrag bereits ausgeführt wird, auf Basis dessen die entsprechenden Zustände gesetzt und ein Workerthread gestartet wird, oder nicht. (Abb. 3.5)

3.4.5.3 Pressen Manager

Wie bei den beiden vorangegangen Managern wird auch hier von der Startmarkierung ausgehend geprüft, welche Nachricht genau eingegangen ist, und dem entsprechend in den

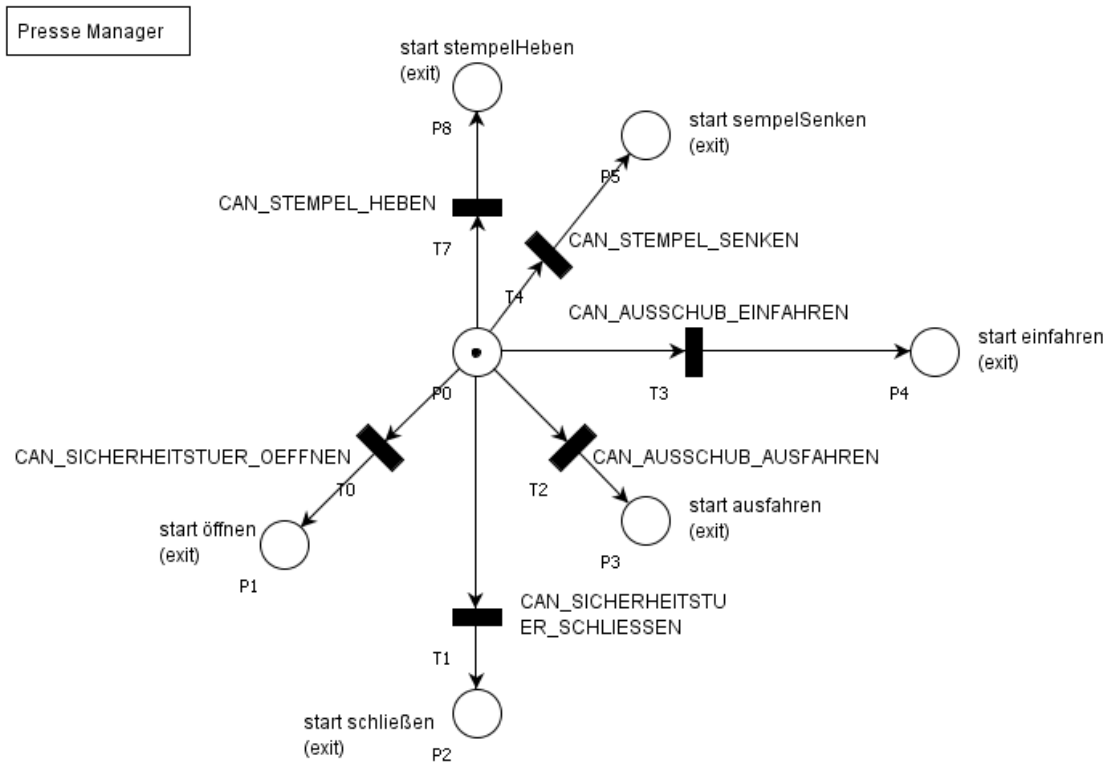


Abbildung 3.6: Petri-Netz: Presse - Manager

jeweiligen Platz gewechselt. Hier finden keine Prüfungen statt, ob der Auftrag bereits eingegangen ist, da die Funktionen der Presse so abstrahiert wurden, dass sie ohne Zeitverlust durchgeführt werden. Geht also der Befehl zum Schließen der Sicherheitstür ein, wird Diese sofort ohne Zwischenschritte geschlossen. (Abb. 3.6)

3.4.5.4 IE Manager

Bei diesem Manager müssen bei einem Teil der Befehle wieder Prüfungen stattfinden, die zeigen ob der Auftrag bereits ausgeführt wird. Transition *T12* repräsentiert die Nachrichten, bei denen dies nicht nötig ist. (Abb. 3.7)

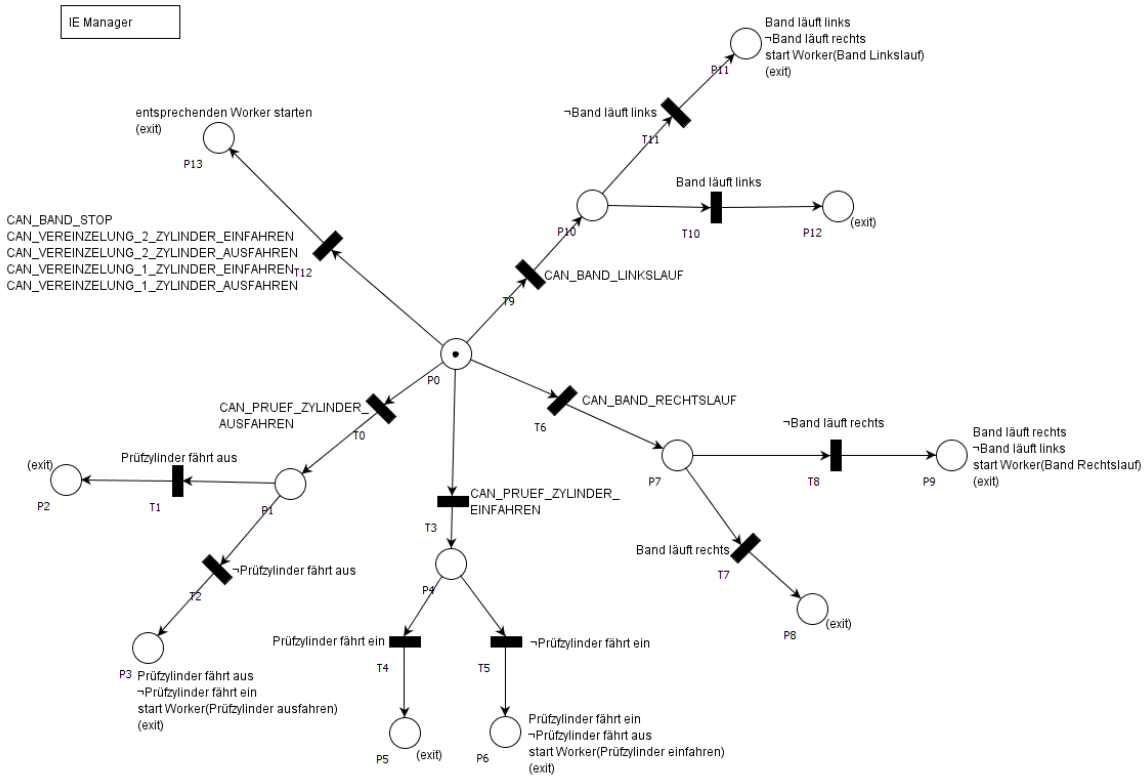


Abbildung 3.7: Petri-Netz: IE - Manager

3.4.6 Modellierung der Worker-Threads

Auch jeder der Worker-Threads wird durch ein eigenes Petri-Netz abgebildet. Diese Netze bilden Subnetze der Manager, erhalten also jedes Mal von dort ihre Marke. Beim Beenden der Worker verschwindet diese Marke wieder. Diese wird nicht mehr benötigt, weil der zuständige Manager bei jedem Start eines Workers eine Marke generiert.
Die Worker sind für die Simulation der Aktionen der Anlage sowie für das Senden von Sensorereignissen über die Senderoutine des Message Handlers an die Steuerungssoftware zuständig.
Aufgrund der Nebenläufigkeit der Worker ist es möglich mehrere Aktionen der Anlage gleichzeitig zu simulieren. Dabei muss von den Managern sichergestellt werden, dass Aktionen die sich gegenseitig ausschließen nicht gleichzeitig gestartet werden (Vorbedingungen).

3.4.6.1 RFZ Worker

Zylinder Ausfahren Diese Funktionseinheit des RFZ Workers ist dafür zuständig die Gabel, die zum Ein- und Auslagern der Werkstücke in das Regal ausgefahren werden kann, auszufahren.

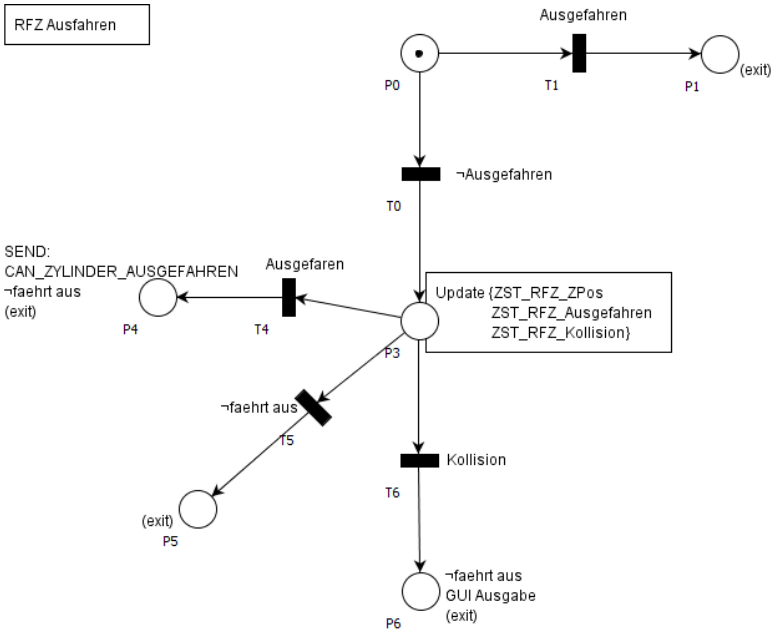


Abbildung 3.8: Petri-Netz: RFZ - Worker: Gabel ausfahren

Die beiden Transitionen *T0* und *T1* prüfen, ob die Gabel bereits ausgefahren ist. Sollte das der Fall sein, ist eine weitere Bearbeitung der Aufgabe nicht nötig bzw. möglich und das Petri Netz endet und der Worker wird beendet. Ist die Gabel noch nicht ausgefahren

kommt das Netz in den Zustand *P3*. Dort findet, mit Hilfe eines Algorithmus, der im Kapitel Implementierung auf Seite 48 erklärt wird, die Berechnung und Aktualisierung von Statusvariablen statt. Aus diesem Zustand führen mehrere Wege heraus. Die erste Möglichkeit ist, dass während der Berechnung im Platz *P3* die Variable gesetzt wird, die repräsentiert, dass die Gabel komplett ausgefahren ist. In diesem Fall schaltet die Transition *T4* und die Markierung wandert in den Platz *P4*. Dort wird dann über den Message Handler die Nachricht *CAN_ZYLINDER_AUSGEFAHREN* an die Steuerungssoftware gesendet, das Ausfahren wird beendet, und das Petri-Netz terminiert.
Die nächste Möglichkeit ist die, dass der Status *fährt aus* nicht mehr gesetzt ist. Dieser Status kann nicht vom Worker, sonder nur vom RFZ Manager gesetzt werden, und bedeutet dass der Manager einen Auftrag bekommen hat, welcher den aktuellen Auftrag des Ausfahrens ausschließt. In diesem Fall kann das nur der Auftrag zum Einfahren der Gabel sein. Tritt dieser Fall nun ein, terminiert das Netz, und damit der Auftrag die Gabel auszufahren.
Der letzte Weg aus dem Zustand *P3* herauszukommen ist der, dass eine Kollision aufgetreten ist. Dies kann eine Kollision mit einem an der Stelle eingelagertem Werkstück sein, an der die Gabel ausgefahren werden soll, oder mit einem Teil des Regals selber, gegen das die Gabel beim Ausfahren gegengestoßen ist. In diesem Fall wird der Benutzer über die GUI zu einer Eingabe aufgefordert, die entscheidet ob die Simulation weitergeführt werden soll. Der Auftrag des Ausfahrens wird aber unabhängig davon beendet. (Abb. 3.8)

Zylinder einfahren Diese Funktionseinheit ist für das Einfahren der Gabel zuständig.

Dieser Teil des RFZ Workers funktioniert identisch wie das Ausfahren, mit der einzigen Abweichung, dass beim Einfahren der Gabel keine Kollision auftreten kann, und diese somit auch nicht im Netz auftaucht. (Abb. 3.9)

Linkslauf Hiermit wird der Linkslauf der horizontal beweglichen Stange des RFZ gesteuert. Als erstes nach der Startmarkierung wird überprüft, ob der linke Endschalter des RFZ aktiv ist (Transitionen *T0* und *T1*), was bedeuten würde, dass die Stange bereits ganz links positioniert ist. In dem Fall würde der Workerthread terminieren. Andernfalls wandert die Markierung in den Zustand *P3*. Dort werden alle relevanten Systemzustände berechnet und aktualisiert.
Die Transition *T4* schaltet dann, wenn die Stange zu Beginn des Auftrags ganz rechts stand, der Endschalter rechts also aktiv war. Sobald die Position ganz rechts verlassen wird wandert die Markierung in den Zustand *P4* und kehrt, nachdem die Nachricht *CAN_ANSCHLAG_RECHTS_OFF* an die Laufzeitumgebung gesendet wurde, in den Zustand *P3* zurück, wo der Algorithmus zur Berechnung der Zustände fortgesetzt wird.
Auch hier besteht wieder die Möglichkeit, dass der RFZ Manager den Auftrag *Linkslauf* storniert, weil er von der Steuerung den Auftrag zum Rechtslauf oder zum Beenden aller horizontaler Bewegungen bekommen hat. Dies repräsentiert die Transition *T6*.

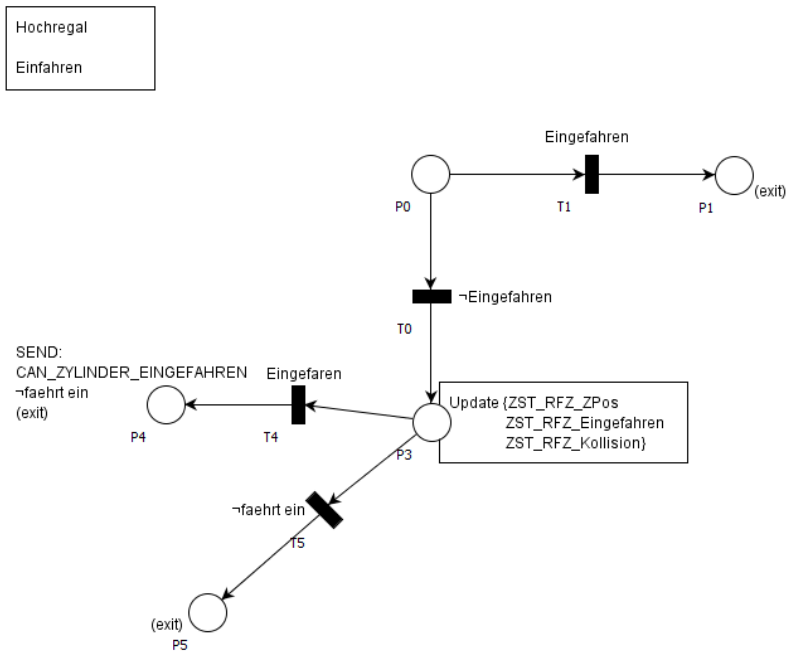


Abbildung 3.9: Petri-Netz: RFZ - Worker: Gabel einfahren

Hat die Stange die Position ganz links erreicht, dann schaltet Transition *T7*, die Nachricht dass der Endschalter links aktiv ist wird gesendet und das Netz terminiert.
Tritt eine Kollision mit dem Kran oder dem Regal selber auf, kommt die Markierung über Transition *T8* in den Zustand *P7*, beendet den Auftrag und terminiert.
Erreicht die Stange auf ihrem Weg nach links eine x Positionsmarke oder eine Übergabe Positionsmarke, wird im Zustand *P8* bzw. *P10* die entsprechende Nachricht gesendet. Anschließend kehrt die Markierung nach *P3* zurück. Beim Verlassen dieser Sensoren wird ebenso verfahren. (Abb. 3.10)

Rechtslauf Der Rechtslauf funktioniert genau so wie der Linkslauf, und bedarf deshalb keiner weiterer Erklärung. (Abb. 3.11)

Heben Das Anheben der Gabel kann dafür verwendet werden, um ein Werkstück aus dem Regal zu holen, indem die Gabel unter das Werkstück geschoben wird, anschließend gehoben wird. Deshalb muss diese Funktionalität, nach der Überprüfung des Endschalters oben am Regal, in das Modell eingebaut werden. Dies geschieht vom Platz *P10* aus über die Transitionen *T13* und *T14*. Die Transition T13 schaltet, wenn die Gabel des RFZ ausgefahren, und sich direkt unter einem eingelagertem Werkstück befindet, und die Markierung wandert zum Platz *P11*, wo das entsprechende Werkstück auf die Gabel gesetzt wird.

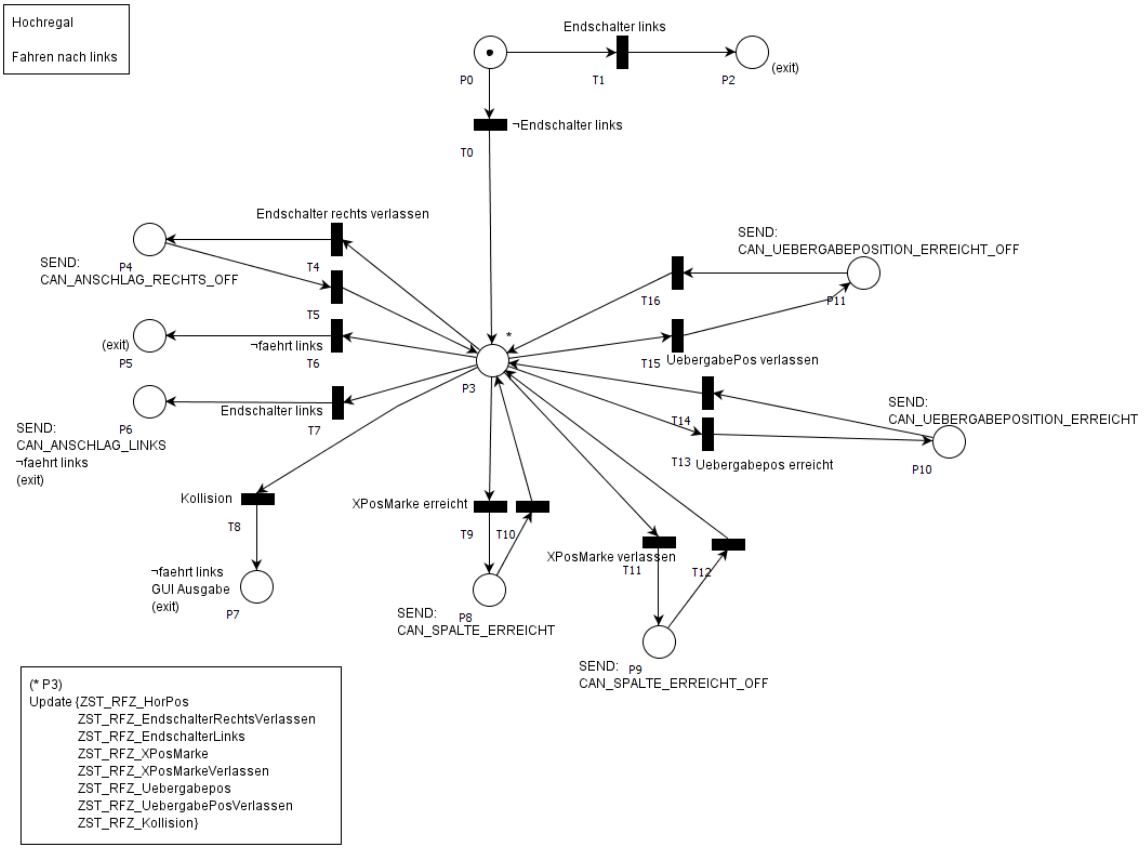


Abbildung 3.10: Petri-Netz: RFZ - Worker: Linkslauf

Anschließend kommt die Markierung in den Platz *P3*. Befindet sich die die Gabel nicht unterhalb eins eingelagerten Werkstücks, oder ist die Gabel nicht ausgefahren, dann wandert die Markierung ohne Umweg in den Platz *P3*, da kein Werkstück ausgelagert werden kann. *P3* beinhaltet wieder einen Update Algorithmus. Von dort ausgehend funktioniert das Netz ähnlich wie die voran gegangenen. Das Netz Terminiert unter den Vorraussetzungen, dass der Endschalter oben erreicht wird, vom Manager der Auftrag storniert wird, oder eine Kollision auftritt. Sensormeldungen werden an die Steuerungssoftware geschickt, wenn der Endschalter unten verlassen wird, ein y Positionssensor erreicht bzw. verlassen wird, und wenn der Endschalter oben erreicht wird. (Abb. 3.12)

Senken Durch das Absenken der Gabel kann ein Werkstück, welches sich auf der Gabel befindet, in das Regal eingelagert werden. Das geschieht, nachdem die Markierung über die Transition *T6* in den Platz *P5* gekommen ist, der Absenkvorgang also beendet wurde. Die Transition *T13* schaltet, wenn sich ein Werkstück auf der Gabel befindet, Diese ausge-

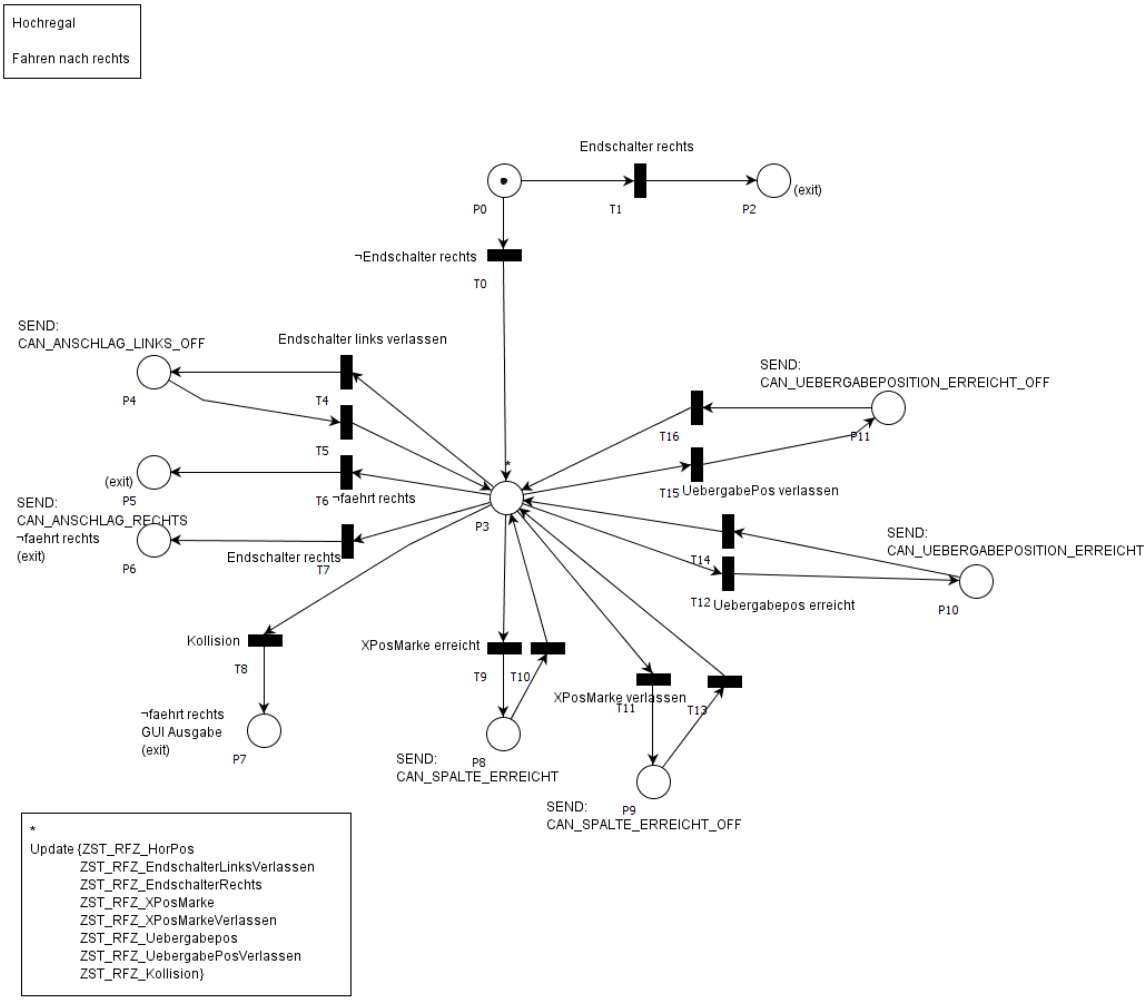


Abbildung 3.11: Petri-Netz: RFZ - Worker: Rechtslauf

fahren ist, und sich horizontal und vertikal an einer Position befindet, an der ein Werkstück eingelagert werden kann. Im darauf folgenden Platz wird dann der Status des Werkstücks auf „eingelagert“ gesetzt. Schaltet *T13* nicht, weil eine der Bedingungen nicht erfüllt ist, dann schaltet Transition *T14* und das Netz wird ohne weitere Aktionen beendet.
Die übrige Funktionalität ist mit dem Heben identisch, außer dass der Zwischenschritt des Auslagerns zwischen *T0* und *P3* wegfällt. (Abb. 3.13)

Horizontaler und vertikaler Stop Die beiden Netze löschen alle Bewegungszustände in horizontaler bzw. vertikaler Richtung. Dadurch kommt die Stange bzw. die Gabel zum Stillstand. (Abb. 3.14, 3.15)

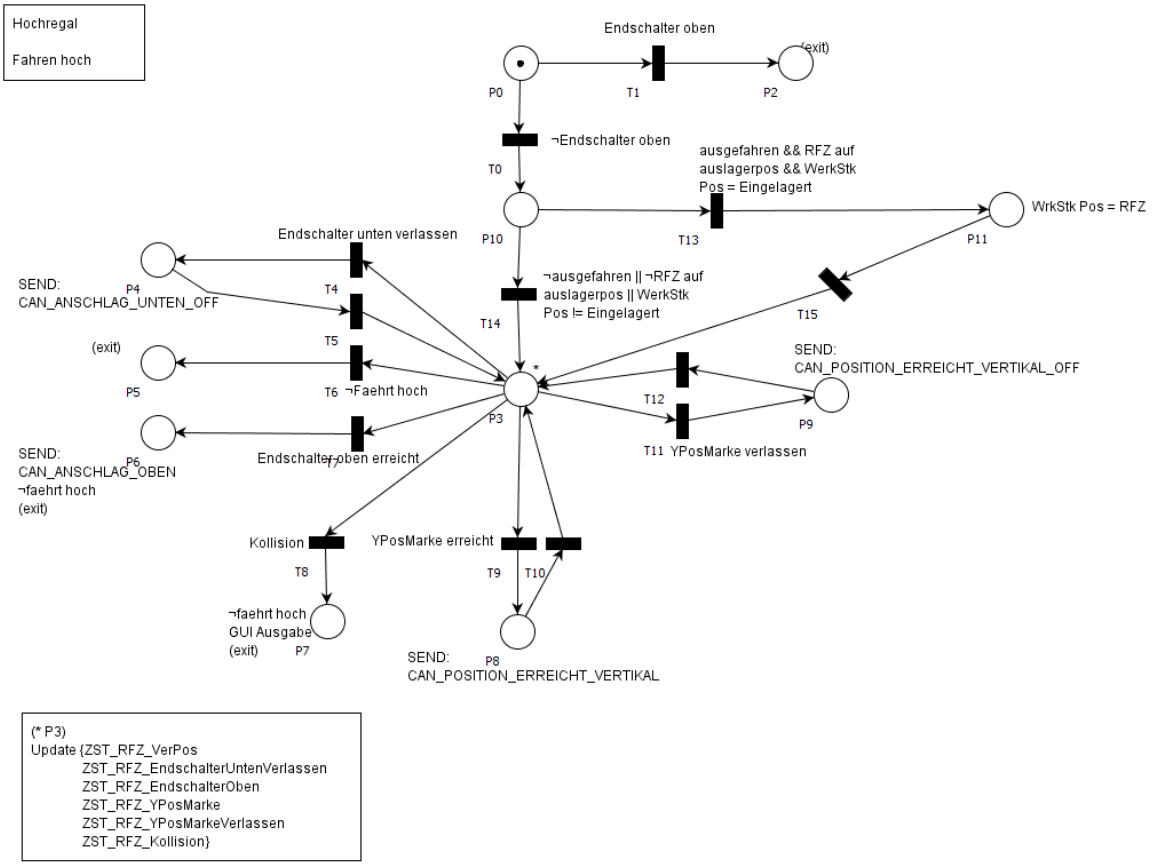


Abbildung 3.12: Petri-Netz: RFZ - Worker: heben

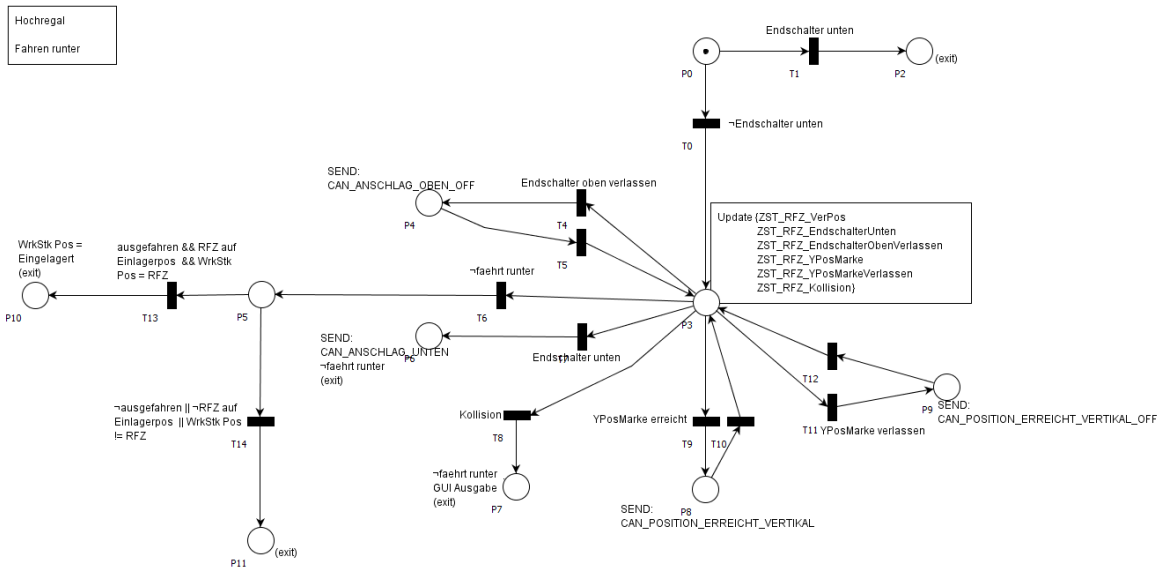


Abbildung 3.13: Petri-Netz: RFZ - Worker: senken

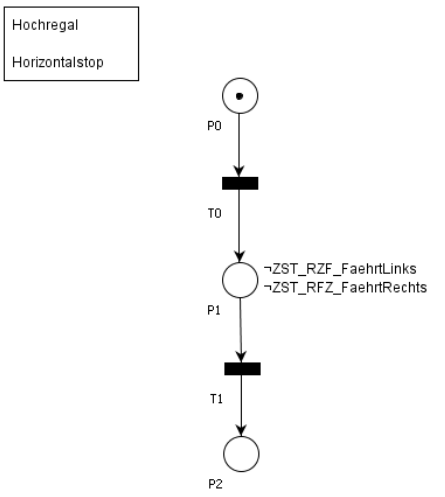


Abbildung 3.14: Petri-Netz: RFZ - Worker: Horizontalstop

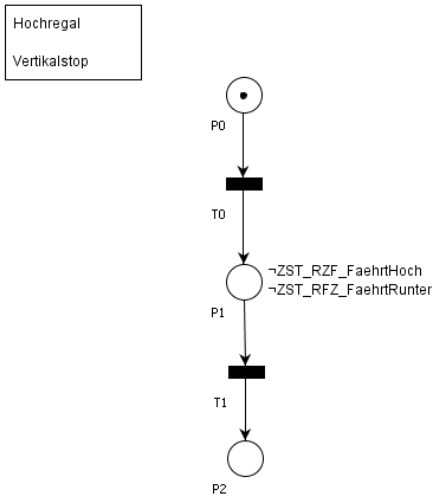


Abbildung 3.15: Petri-Netz: RFZ - Worker: Vertikalstop

3.4.6.2 Kran Worker

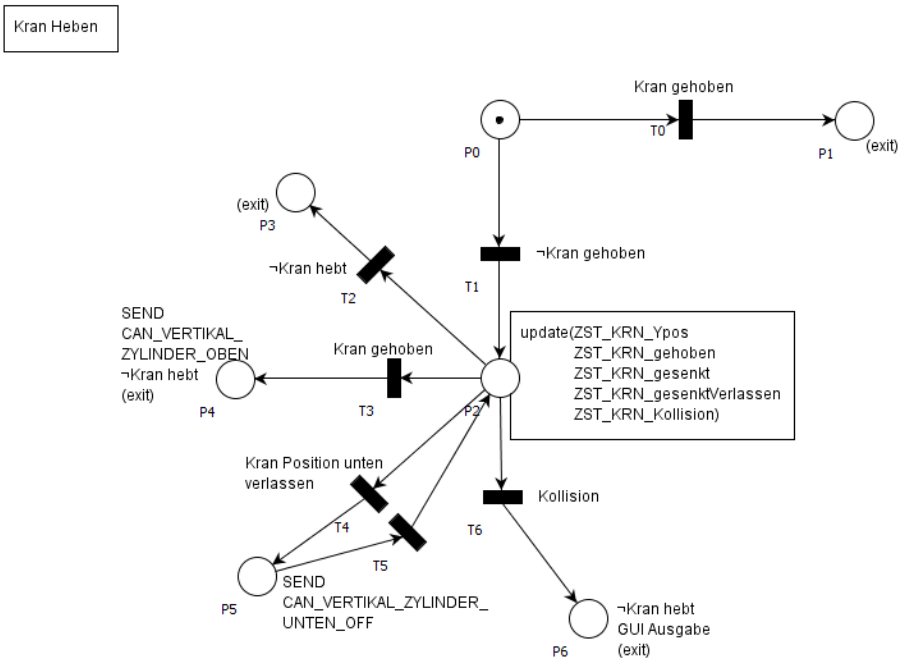


Abbildung 3.16: Petri-Netz: Kran - Worker: heben

Heben Von der Startmarkierung aus wird mit den Transitionen *T0* und *T1* zuerst überprüft, ob der Kran bereits angehoben ist. Ist das nicht der Fall, wandert die Markierung in den Platz *P2*, wo der Update Algorithmus abläuft. Über die Transitionen *T2, T3* und *T6* kann das Netz entweder durch einen Abbruch des Bewegungsauftrages, das Erreichens der oberen Endposition, oder durch eine Kollision terminieren. Ist der Kran vollständig gehoben, wird das Sensorereignis *CAN_VERTICAL_ZYLINDER_OBEN* gesendet, und der Worker wird beendet.
Wird die Position ganz unten verlassen, so wird im Platz *P5* die entsprechende Nachricht an die Steuerungssoftware geschickt. (Abb. 3.16)

Senken Bis auf den Teil hinter Transition *T3* ist das Netz gleich aufgebaut wie das für das Heben des Krans.
Der Teil des Netzes hinter Transition *T3* ist für das Aufnehmen eines Werkstücks vom RFZ, von der Presse und vom Förderband zuständig. Die Aufnahme ist dann möglich, wenn der Ausleger des Krans ausgefahren ist, der Sauger eingeschaltet ist, und sich der Kran an der richtigen Position, entweder am Übergabepunkt für das RFZ, für die Presse oder das Förderband, befindet. Außerdem muss sich ein Werkstück dort befinden, wo es aufgenommen

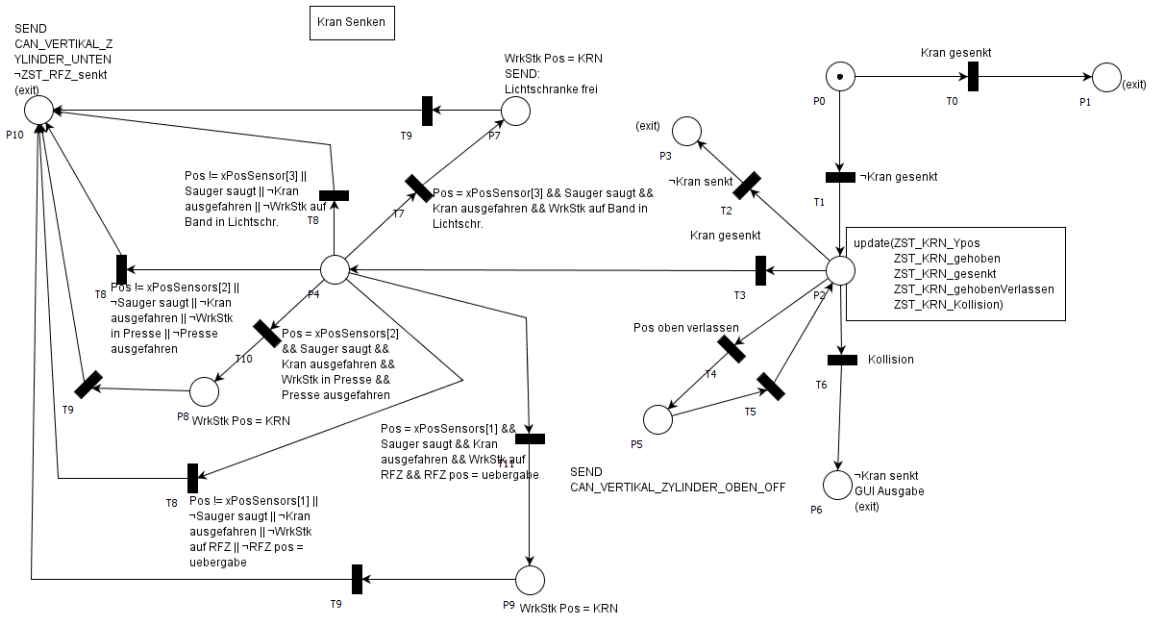


Abbildung 3.17: Petri-Netz: Kran Worker: senken

werden soll, und es darf sich noch kein Werkstück am Kran befinden. Diese Zustände werden an den Transitionen *T7, T10* und *T11* überprüft. Wenn eine dieser Transitionen schaltet, so wird das Aufnehmen des Werkstücks durchgeführt. Wird das Werkstück vom Band aufgenommen muss in Platz *P7* noch das Sensorereignis für das frei werden der Lichtschranke gesendet werden.
Kann kein Werkstück aufgenommen werden, weil eine oder mehrere der oben genannten Bedingungen nicht zutreffen, dann wandert die Markierung über Transition *T8* in den Zustand *P10*, das Sensorereignis *CAN_VERTICAL_ZYLINDER_UNTEN* wird gesendet, und das Netz terminiert. Zu beachten ist, dass die Transition *T8* drei Mal auftaucht. Dies dient nur der Übersichtlichkeit des Netzes, es handelt sich eigentlich immer um die selber Transition, an die alle Bedingungen der drei Transitionen gebunden sind. (Abb. 3.17)

Sauger ein Ist der Ausleger des Krans ausgefahren, der Kran gesenkt, und an einer der drei Übergabepositionen, und befindet sich ein Werkstück an dieser Übergabeposition und noch keines am Kran, so kann auch durch das Einschalten des Saugers ein Werkstück aufgenommen werden. Diese Bedingungen werden an den Transitionen *T0, T1* und *T2* überprüft, und führen bei einer Schaltung dazu, dass die Markierung in den Platz *P1, P2* oder *P3* gelangen, wo das Werkstück aufgenommen wird. Anschließend wandert die Markierung in Platz *P4* und das Netz terminiert.

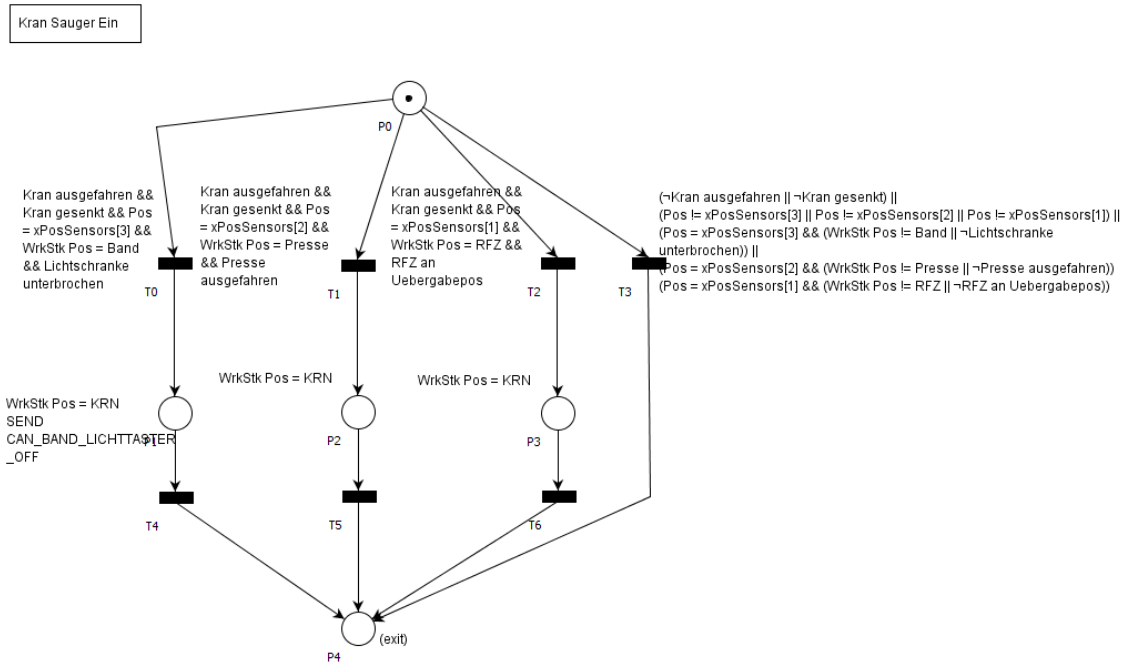


Abbildung 3.18: Petri-Netz: Kran - Worker: Sauger ein

Schaltet keine der drei genannten Transitionen, dann schaltet *T3* und die Markierung wandert ohne Umweg in Platz *P4*. (Abb. 3.18)

Sauger aus Beim Ausschalten des Saugers wird das Werkstück, welches sich am Kran befindet, an das Förderband, die Presse oder das RFZ übergeben, wenn der Kran sich auf einer der drei Übergabepositionen befindet, der Ausleger ausgefahren ist, und der Kran gesenkt ist. Befindet sich kein Werkstück am Kran wird der Sauger ohne weiter Aktionen ausgeschaltet. Befindet sich ein Werkstück am Kran, aber keine der Transitionen, die die Bedingungen für die Übergabe überprüfen, schalten kann, dann fällt das Werkstück herunter. (Abb. 3.19)

Ausleger ein- bzw. ausfahren Beide Modelle arbeiten wieder nach demselben Muster, wie viele der bisherigen Modelle auch.
Der einzige Unterschied zwischen diesen beiden Netzen ist, dass beim Ausfahren eine Kollision mit einem anderen Teil der Anlage auftreten kann, beim Einfahren nicht. Eine Interaktion mit den Werkstücken findet bei diesen beiden Modellen nicht statt. (Abb. 3.20, 3.21)

Kran im Uhrzeigersinn drehen Die Startmarkierung *P2* ist bereits der Platz mit dem Update Algorithmus, da eine vorangehende Prüfung auf Endschalter oder ähnliches wegfällt.

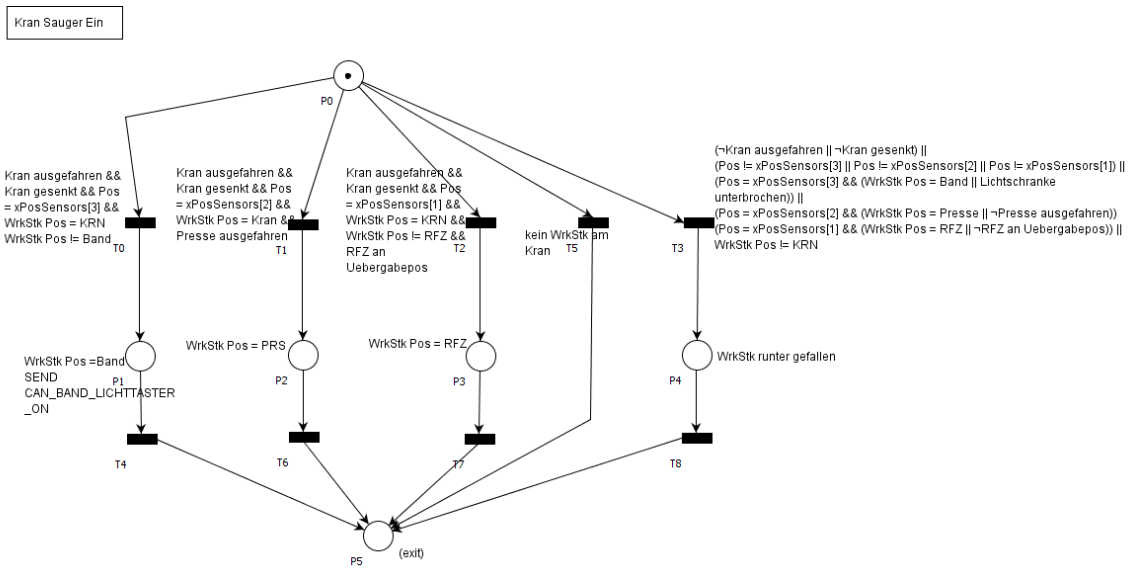


Abbildung 3.19: Petri-Netz: Kran - Worker: Sauger aus

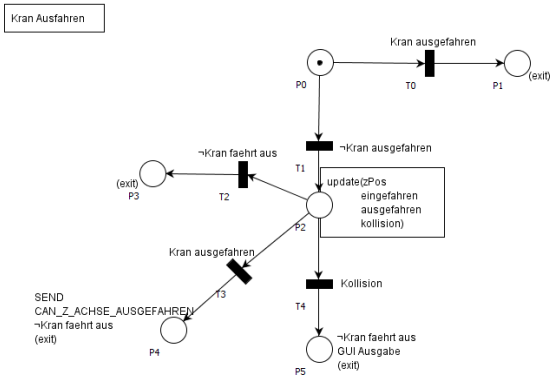


Abbildung 3.20: Petri-Netz: Kran - Worker: Ausleger ausfahren

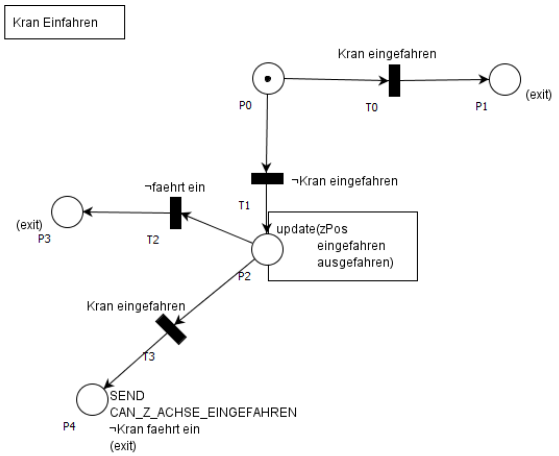


Abbildung 3.21: Petri-Netz: Kran - Worker: Ausleger einfahren

An jeder der drei Übergabepositionen befindet sich eine Schraube, die als Sensor dient. Wird ein solcher Sensor erreicht oder verlassen wird das entsprechende Event an die Steuerung geschickt. Kollisionsbehandlung und Stornierung des Bewegungsauftrages sind auch hier wieder als Optionen das Netz zu beenden vorhanden.
Eine Besonderheit der horizontalen Bewegung des Krans ist, dass es keine Endschalter gibt. Durch die Konstruktion ist aber gegeben, dass er sich nicht um 260 Grad um die eigene Achse drehen kann. Ab einem bestimmten Punkt in beiden Richtungen wickelt sich das Kabel

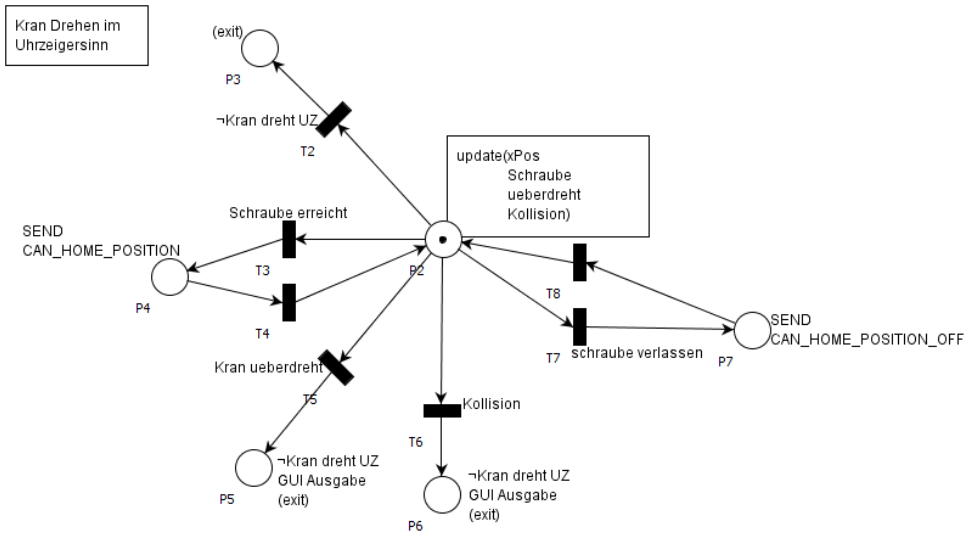


Abbildung 3.22: Petri-Netz: Kran - Worker: Kran drehen UZ

des Krans so um den Kran, dass er sich nicht weiter drehen kann. Dies wird mit Transition *T5* und Platz *P5* simuliert. (Abb. 3.22)

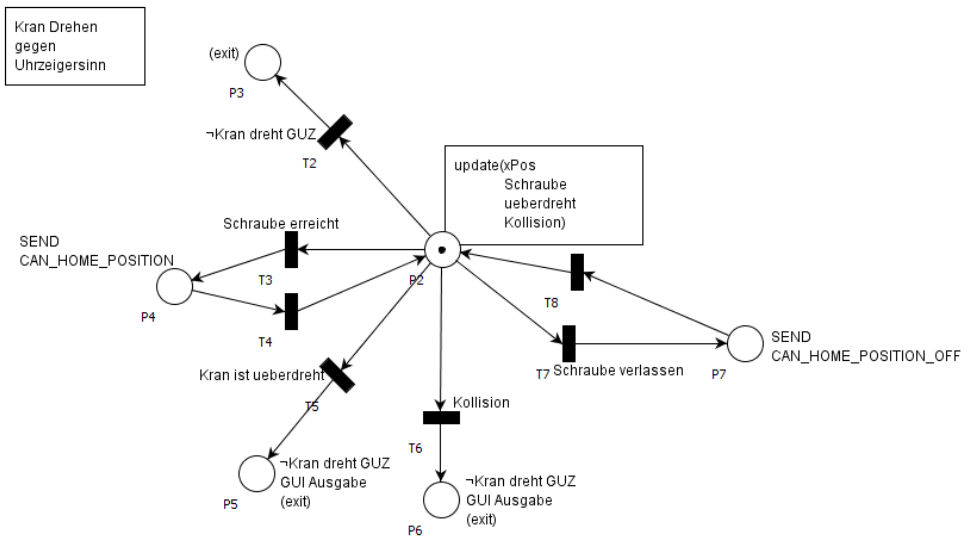


Abbildung 3.23: Petri-Netz: Kran - Worker: Kran drehen GUZ

Kran gegen den Uhrzeigersinn drehen Das Netz arbeitet genau wie das Netz für das Drehen im Uhrzeigersinn, und wird daher nicht weiter beschrieben. (Abb. 3.23)

3.4.6.3 Pressen Worker

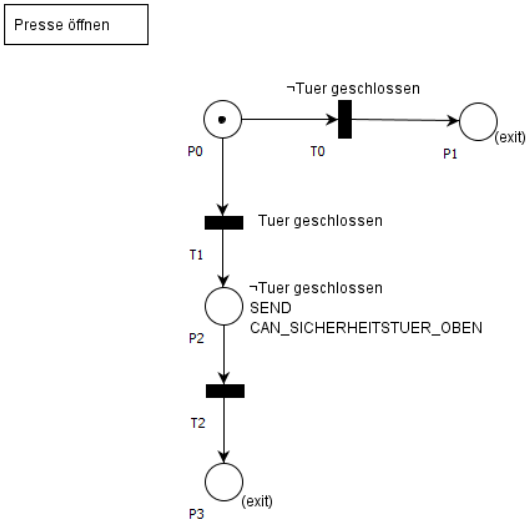


Abbildung 3.24: Petri-Netz: Presse - Worker: Sicherheitstür öffnen

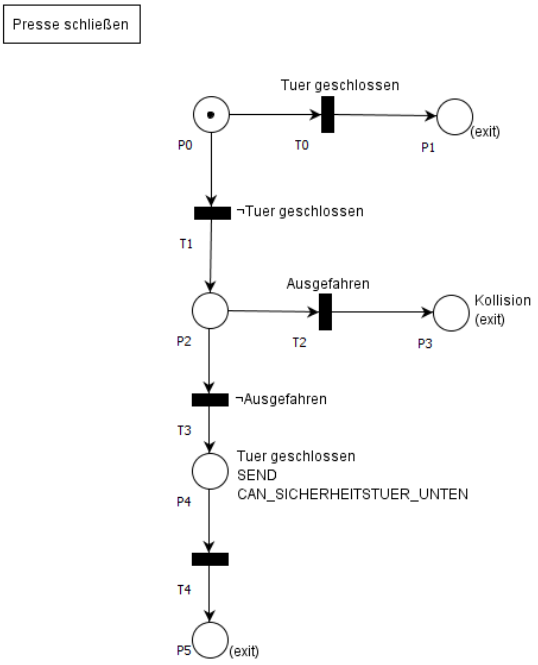


Abbildung 3.25: Petri-Netz: Presse - Worker: Sicherheitstür schliessen

Sicherheitstür öffnen / schließen Wie bereits erwähnt wurden alle Funktionen der Presse soweit vereinfacht modelliert, dass sie ohne Zeitverlust sofort ihren Zustandswechsel durchführen. Deshalb können die Petri-Netze relativ einfach gehalten werden.
Beim Öffnen wird lediglich am Anfang geprüft, ob die Tür bereits geöffnet ist. Anschließen, wenn das nicht der Fall ist, wird der Zustand *Tuer geschlossen* gesetzt, und das entsprechende Event abgeschickt.
Bei Schließen ist es zusätzlich noch nötig, dass der Zustand des Ausschubes der Presse überprüft wird. Ist der Ausschub ausgefahren, kann die Sicherheitstür nicht geschlossen werden, und es kommt zu einer Kollision. (Abb. 3.24, 3.25)

Ausschub ausfahren / einfahren Der Ausschub ist der Teil der Presse, auf dem vom Kran Werkstücke abgegeben bzw. von Selbigem abgeholt werden können. Dafür kann der Ausschub die Werkstücke vom Inneren der Presse nach außen befördern, und umgekehrt.
Das Ausfahren ist nur dann möglich, wenn Sicherheitstür nicht geschlossen ist, was mit den Transitionen *T2* und *T3* sichergestellt wird.
Eingefahren werden kann der Ausschub nur dann, wenn ebenfalls die Sicherheitstür geöffnet

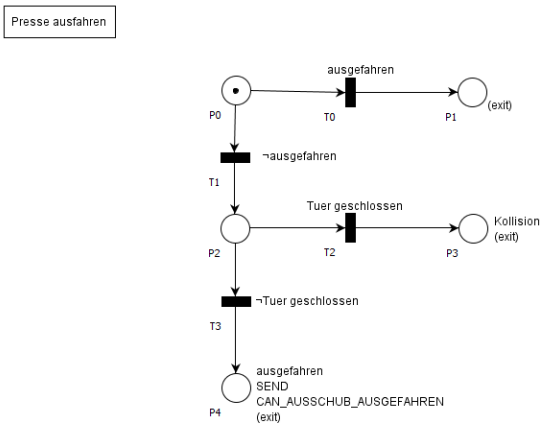


Abbildung 3.26: Petri-Netz: Presse - Worker: Ausschub ausfahren

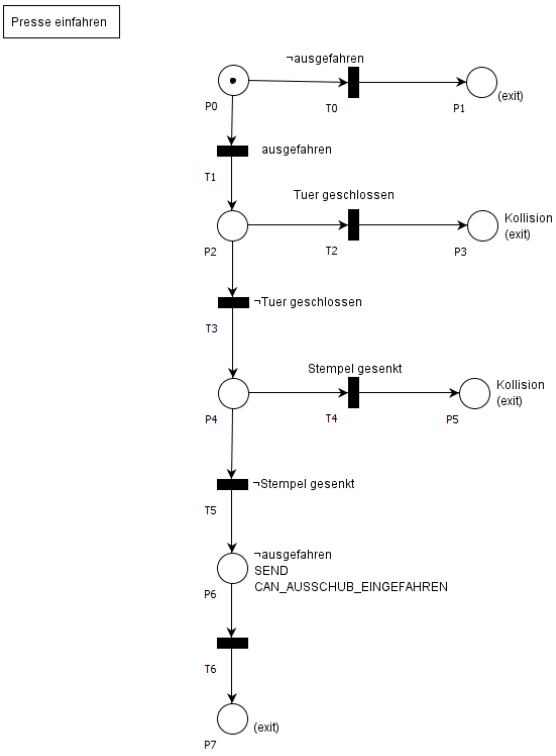


Abbildung 3.27: Petri-Netz: Presse - Worker: Ausschub einfahren

ist. Zusätzlich muss der Stempel der Presse gehoben sein, da es sonst zu einer Kollision kommen kann, wenn zwei ungepresste Werkstücke übereinander auf dem Ausschub liegen. (Abb. 3.26, 3.27)

Stempel senken Bei den Transitionen *T0 - T5* wird überprüft, ob die Bedingungen für das Senken des Stempels erfüllt sind. Ist der Ausschub ausgefahren, kommt es zu einer Kollision, ist die Sicherheitstür geöffnet kommt es lediglich zu einer Warnung, da dies die Funktion des Pressens nicht beeinträchtigt.
Liegen nun eine Ober- und Unterschale in dem eingefahrenen Ausschub bereit, während der Stempel gesenkt wird, so schaltet Transition *T8* und die beiden Teile werden zusammengepresst. Das bedeutet in diesem Modell, die beiden Einzelteile werden aus der Presse entfernt, und durch ein komplettes Werkstück ersetzt. Liegen keine, oder die falschen Werkstücke in der Presse, so findet kein Pressen statt, und das Netz terminiert in Zustand *P7*. (Abb. 3.28)

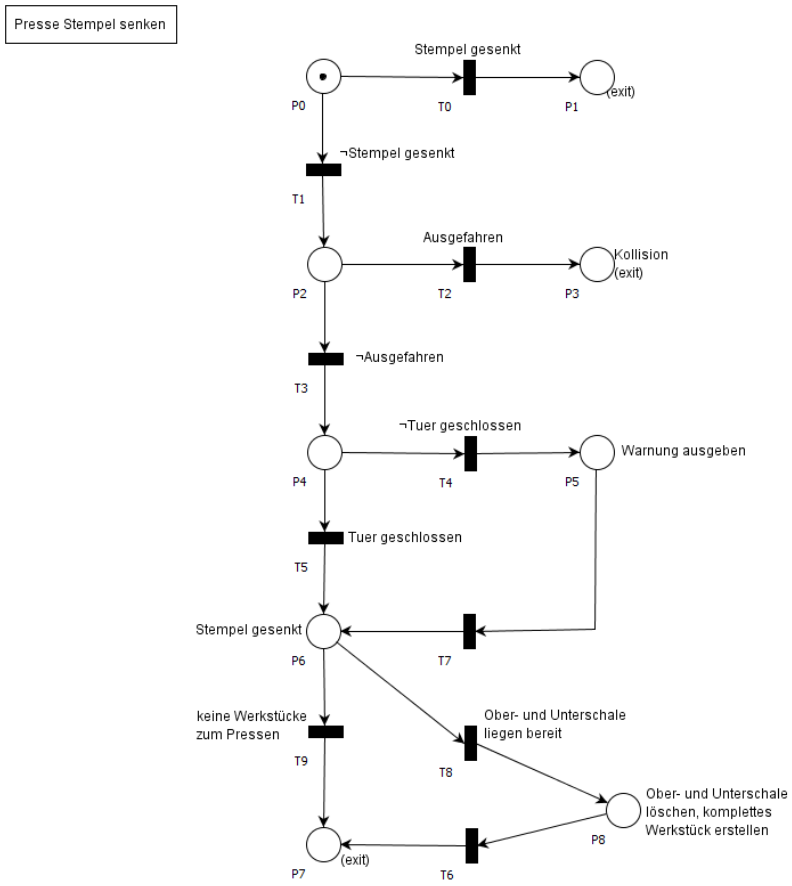


Abbildung 3.28: Petri-Netz: Presse - Worker: Stempel senken

Stempel heben Da der Stempel unter allen denkbaren Szenarien gehoben werden kann, außer natürlich wenn er bereits gehoben ist, findet in diesem Netz keine Prüfung auf die Zustände der anderen Teile der Presse statt. Das Netz terminiert nachdem der Zustand „Stempel gesenkt“ gelöscht wurde. (Abb. 3.29)

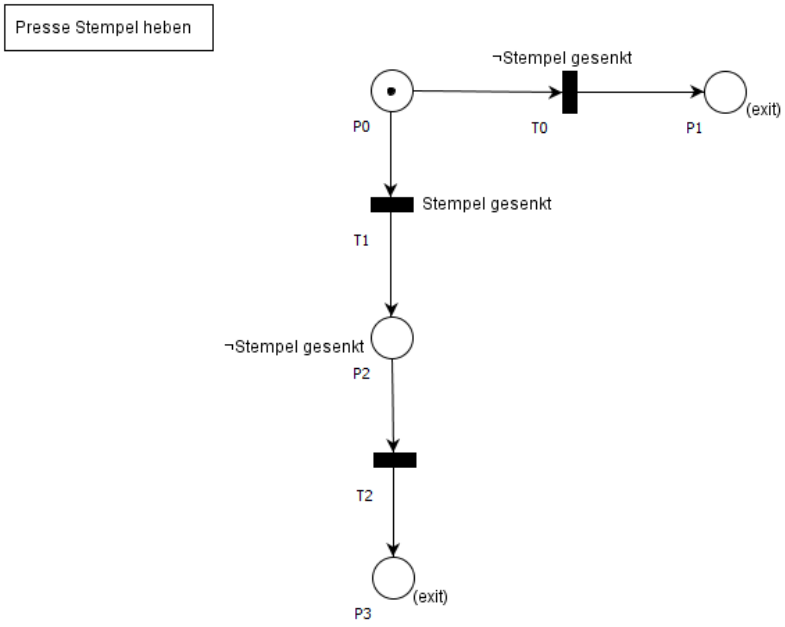


Abbildung 3.29: Petri-Netz: Presse - Worker: Stempel heben

3.4.6.4 IE Worker

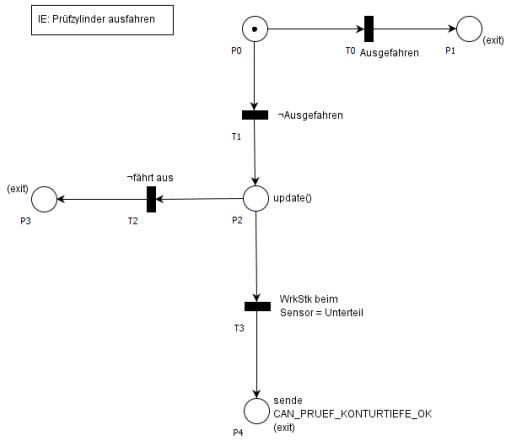


Abbildung 3.30: Petri-Netz: IE - Worker: Prüfzylinder ausfahren

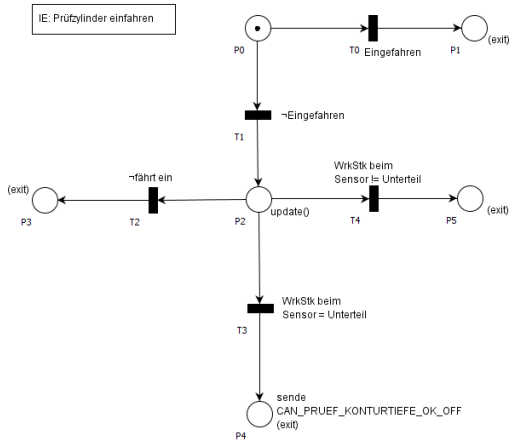


Abbildung 3.31: Petri-Netz: IE - Worker: Prüfzylinder einfahren

Prüfzylinder ausfahren / einfahren Der Prüfzylinder ist der Teil der Identifizierungseinheit, der dafür zuständig ist die Kontur der Werkstücke zu überprüfen. Wie bereits beschrieben, haben Werkstücke vom Typ „Unterschale“ eine Einkerbung in der Mitte, was sie an dieser Stelle niedriger macht als die Oberschalen. Damit ist es diesem Sensor möglich diese beiden Typen voneinander zu unterscheiden.
Ein- und Ausfahren arbeiten wieder weitgehend identisch. Erst wird abgefragt, ob der angestrebte Zustand bereits erreicht ist, ist dies nicht der Fall, wechselt die Markierung in den Platz mit dem Update Algorithmus. Sollte der Manager vorzeitig einen Befehl zum Abbruch des Ausfahrens bekommen, so setzt er den Zustand *fährt aus* auf *False*, und die Transition *T2* des Workers schaltet.
Liegt unter dem Sensor ein Werkstück vom Typ Unterschale auf dem Förderband, so schaltet Transition *T3* und die Steuerung bekommt die Nachricht, dass der Sensor komplett ausgefahren werden konnte. Liegt ein Werkstück vom Typ Oberschale auf dem Band, so kann der Sensor nicht komplett ausfahren, und das Event wird nicht gesendet.
Beim Einfahren wir das Event *CAN_PRUEF_KONTURTIEFE_OK_OFF* ebenfalls nur dann gesendet, wenn der richtige Werkstücktyp unter dem Sensor liegt. Ist das nicht der Fall, so terminiert das Netz ohne das Senden einer Nachricht. (Abb. 3.30, 3.31)

Förderband: Rechtslauf / Linkslauf Das Förderband befördert Werkstücke von den beiden Werkstückmagazinen zum Übergabepunkt mit dem Kran am Ende des Bandes. Auf diesem Weg werden einige Sensoren, welche die Beschaffenheit des Werkstückes identifizieren, passiert. Die Position des Werkstücks wird in Platz *P0*, innerhalb des Update Algorith-

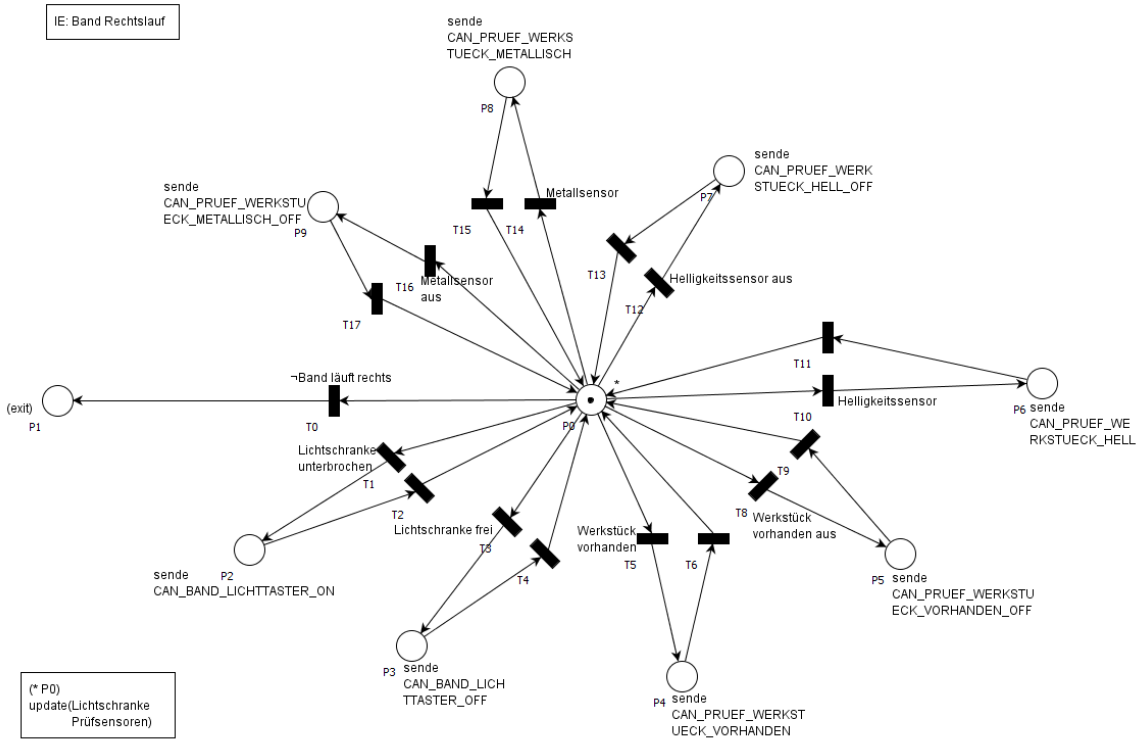


Abbildung 3.32: Petri-Netz: IE - Worker: Band Rechtslauf

mus aktualisiert. Anhand dessen, und anhand des Typs des Werkstücks werden die Zustände der Sensoren gesetzt. Die Funktionalität dieser Sensoren wird mit den Transitionen *T1 - T17* und den Plätzen *P2 - P9* realisiert. Dabei wird immer beim Erreichen und beim Verlassen des Sensors ein Event ausgelöst. Bei den Sensoren handelt es sich um eine Lichtschranke am Ende des Bandes, die das erreichen des Endes des Bandes durch das Werkstück signalisiert, um einen Sensor, der anspricht, sobald überhaupt irgendein Werkstück ihn passiert, um einen Helligkeissensor, der anspricht, sobald ihn ein dunkles Werkstück passiert sowie um einen Metallsensor, der bei den metallischen Unterschalen ein Event generiert.
Nur durch die Transition *T0* kann das Netz beendet werden. Diese schaltet, wenn der Manager den Befehl für *Band stop* oder für die Bewegung in die entgegengesetzte Richtung erhält, und somit den aktuellen Auftrag widerruft. (Abb. 3.32, 3.33)

Magazin 1 und 2 - Ausschub einfahren Beim Einfahren sind keine Besonderheiten zu beachten, und der Zustand des Ausschubes kann sofort gesetzt werden. (Abb. 3.34, 3.35)

Magazin 1 und 2 - Ausschub ausfahren Im Wesentlichen führen zwei Wege durch diese Netze. Ist das Magazin leer, so schaltet Transition *T0*, der Ausschub wird ausgefahren und

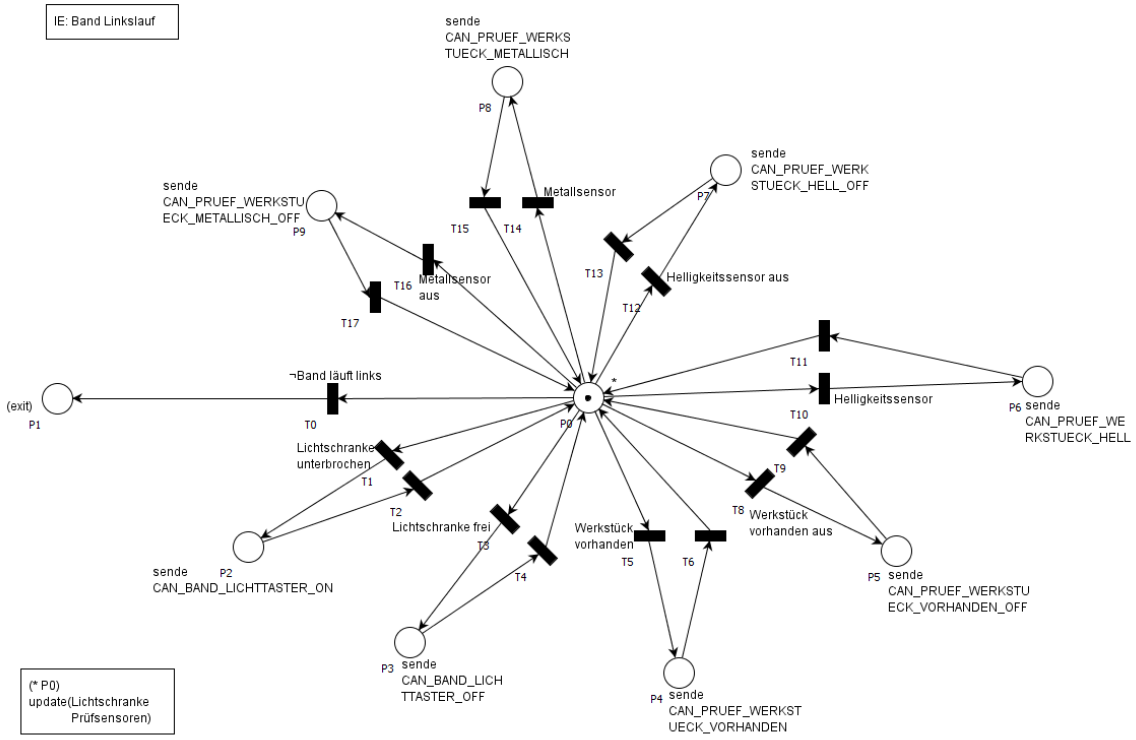


Abbildung 3.33: Petri-Netz: IE - Worker: Band Linkslauf

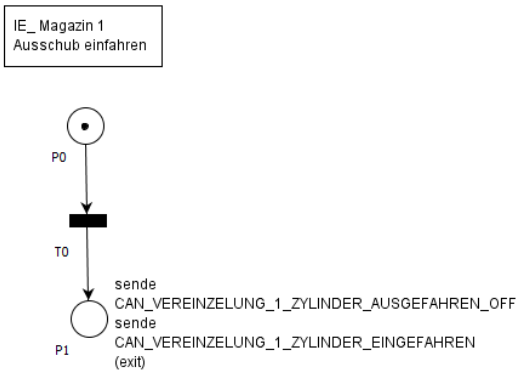


Abbildung 3.34: Petri-Netz: IE - Worker: Magazin 1 Ausschub einfahren

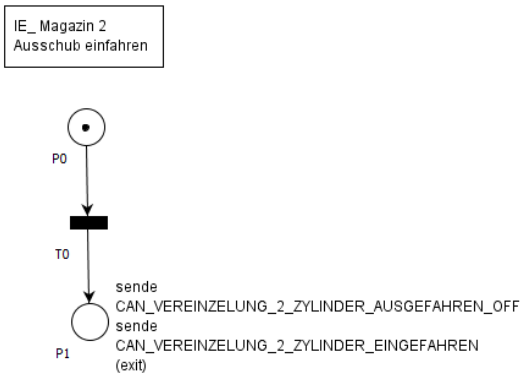


Abbildung 3.35: Petri-Netz: IE - Worker: Magazin 2 Ausschub einfahren

anschließend wird in Platz *P4* die Nachricht *CAN_VEREINZELUNG_1_MAGAZIN_LEER* bzw. *CAN_VEREINZELUNG_2_MAGAZIN_LEER* gesendet. Dann terminiert das Netz.
Ist das Magazin nicht leer, so wird ebenfalls der Ausschub ausgefahren, zusätzlich wird aber

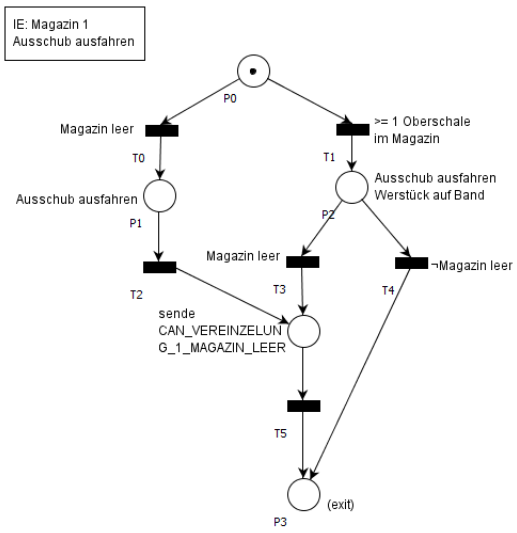


Abbildung 3.36: Petri-Netz: IE - Worker: Magazin 1 Ausschub ausfahren

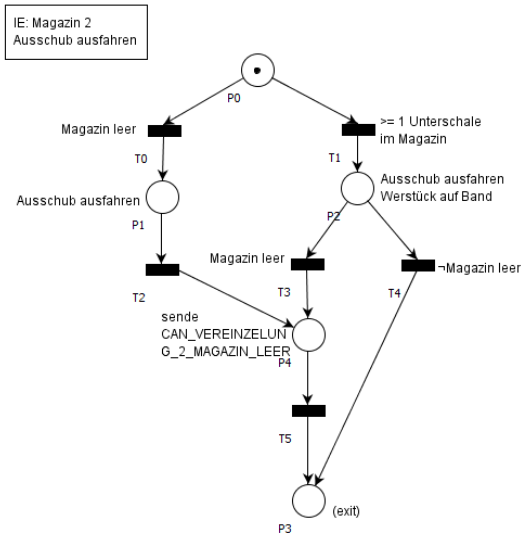


Abbildung 3.37: Petri-Netz: IE - Worker: Magazin 2 Ausschub ausfahren

das ausgeworfene Werkstück auf das Band gesetzt. Anschließend muss ein weiteres Mal überprüft werden, ob das Magazin jetzt leer ist. Ist dies der Fall wird ebenfalls die entsprechende Nachricht gesendet, und das Netz terminiert. Sind noch weitere Werkstücke im Magazin terminiert das Netz ohne das Senden einer Nachricht. (Abb. 3.36, 3.37)

Band Stop (Abb. 3.38)

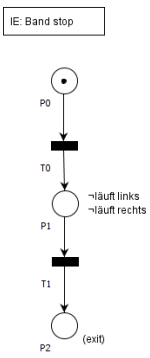


Abbildung 3.38: Petri-Netz: IE - Worker: Band stop

3.5 Kollisionsbehandlung

Eine Kollision bedeutet im Kontext dieser Arbeit, dass in der Realität tatsächlich zwei Teile der Anlage miteinander Zusammenstoßen würden. Dieser Aspekt muss selbstverständlich auch in der Modellierung der Simulationsumgebung auftauchen. Kollisionen können auf unterschiedliche Weise auftreten:

- Zwischen RFZ und Kran beim Linkslauf, beim Rechtslauf, beim Senken und beim heben des RFZ, wenn der Kran sich in einem Bereich zwischen Übergabepunkt der Presse und Übergabepunkt des RFZ aufhält.
- Zwischen RFZ und Regalboden beim Ausfahren der Gabel, wenn sich diese auf der Höhe des Regalbodens befindet.
- Zwischen RFZ und Regalboden beim Senken und Heben, wenn die Gabel ausgefahren ist und dabei ein Regalboden passiert wird.
- Zwischen RFZ und einem im Regal eingelagertem Werkstück beim Senken, Heben, Linkslauf und Rechtslauf, wenn die Gabel ausgefahren ist und ein Werkstück passiert wird.
- Zwischen RFZ und Werkstück beim Ausfahren der Gabel, wenn diese sich an der Einlagerposition des Werkstücks befindet.
- Zwischen Kran und RFZ beim Drehen im und gegen den Uhrzeigersinn im Bereich zwischen den Übergabepunkten von RFZ und Presse, wenn sich das RFZ ebenfalls in diesem Bereich aufhält.
- Zwischen Kran und Presse beim Drehen im und gegen den Uhrzeigersinn im Bereich um den Übergabepunkt der Presse, wenn der Ausleger des Krans ausgefahren ist.
- Zwischen Kran und Förderband beim Drehen im und gegen den Uhrzeigersinn im Bereich um den Übergabepunkt des Bandes, wenn der Kran gesenkt, und der Ausleger ausgefahren ist.
- Innerhalb der Presse beim Ausfahren des Ausschubs, wenn die Sicherheitstür geschlossen ist.
- Innerhalb der Presse beim Schließen der Tür, wenn der Ausschub ausgefahren ist.

Um diese Kollisionen zu erkennen und zu behandeln werden bei den betroffenen Bewegungen innerhalb des zuständigen Workers die Zustände der eventuell betroffenen bzw. gefährdeten Anlagenteile überprüft. Dies geschieht innerhalb des Update Algorithmus, und wird somit nach jedem Bewegungsschritt ausgeführt. Dadurch wird sichergestellt, dass wirklich

jede Kollision zeitnah erkannt wird. Durch diese Technik kann es vorkommen, dass eine Kollision doppelt erkannt wird, wenn sich beispielsweise Kran und RFZ gleichzeitig in einem kollisionsgefährdetem Bereich bewegen. Dies ist aber durchaus erwünscht, da dem Benutzer der Simulation damit deutlich gemacht werden kann, dass zwei Teile der Anlage gleichermaßen für die Kollision verantwortlich waren, und nochmals überprüft werden sollten.
Wenn ein Teil der Anlage nicht in der Simulation berücksichtigt wird, dann treten mit diesem Teil auch keine Kollisionen auf. Wird zum Beispiel die Presse nicht simuliert, dann kann sich der Kran auch in Bereichen bewegen, in denen normalerweise eine Kollision mit der Presse auftreten würde. Dies ist nützlich, wenn man erst einmal die Basisfunktionalität eines Teils der Anlage entwickeln möchte, und dabei nicht von Kollisionswarnungen gestört werden will.

3.6 Handling der Werkstücke

3.6.1 Aufbau

Die Werkstücke sind durch eine Datenstruktur definiert, welche folgende Attribute speichert:

- Die aktuelle Position des Werkstücks. Dies können sein: *bnd, krn, prs, rfz, stored, down, mgzn1, mgzn2*, wobei stored für im Regal eingelagerte, und down für heruntergefallene Werkstücke steht.
- Die Koordinaten auf dem Förderband. Dieses Attribut ist nur gesetzt, wenn sich das Werkstück auf dem Band befindet.
- Die Koordinaten, wo das Werkstück im Regal eingelagert ist, wenn es eingelagert ist.
- Der Typ des Werkstücks: Unterteil, schwarzes Oberteil, weißes Oberteil, kompletter Würfel mit schwarzem Oberteil, kompletter Würfel mit weißem Oberteil.

3.6.2 Erstellung und Entfernen

Die Erstellung eines Werkstücks muss vom Benutzer über eine Funktion in der GUI angestoßen werden. Es ist möglich Unter- und Oberschalen in die Entsprechenden Magazine der Identifizierungseinheit zu füllen, aber auch Werkstücke beliebigen Typs an beliebiger Stelle der Anlage zu platzieren. Dies ist besonders dann Sinnvoll, wenn man nur eine Teilfunktionalität der Anlage entwickelt und testet.
Ebenfalls durch den Benutzer sollen Werkstücke auch von beliebiger Position der Anlage entfernt werden können.

3.6.3 Transport und Übergabe

Der Transport der Werkstücke funktioniert am Kran, auf der Presse und im RFZ so, dass nur das Attribut der aktuellen Position in der Datenstruktur des Würfels richtig gesetzt werden muss. Dadurch ist eindeutig festgelegt, wo sich das Werkstück gerade befindet, da es sich bei diesen Anlagenteilen nur jeweils an einem bestimmten Ort befinden kann - beim Kran hängt es am Sauger, bei der Presse liegt es im Ausschub und beim RFZ liegt es auf der Gabel. Beim Förderband und bei im Regal eingelagerten Werkstücken ist es zusätzlich nötig Koordinaten zu speichern, um den Ort des Werkstücks eindeutig festzulegen.
Bei der Übergabe der Werkstücke zwischen den einzelnen Anlagenteilen ist es die Aufgabe des Workers, der die Übergabe veranlasst hat, die Datenstruktur der Werkstücks entsprechend zu aktualisieren. In den meisten Fällen ist dies der Kran - Worker. Nur beim Einlagern der Werstücke ins Regal ist das RFZ zuständig. Bei der Übergabe zwischen den Magazinen und dem Band ist der IE Worker zuständig.

3.7 Schnittstelle zur GUI

Die GUI soll als eigener gekapselter Teil der Software realisiert werden. Dies geschieht aus zwei Gründen. Zum Einen wird der entstehende Code dadurch übersichtlicher und damit wartbarer, zum Anderen ist es so ohne großen Aufwand möglich Die GUI zu einem späteren Zeitpunkt auszutauschen.
Zu diesem Zweck wird ein Interface zur GUI-Komponente bereit gestellt, das von der Simulationssoftware bedient wird und von der GUI implementiert werden muss.

3.8 Design der GUI

Die GUI soll zur besseren Übersichtlichkeit in verschiedene Sektionen unterteilt werden. Zu diesem Zweck eignet sich die Verwendung von Tabs besonders gut, das heißt dem Benutzer werden mehrere, über Reiter erreichbare, Sektionen der Oberfläche bereitgestellt.
Eine dieser Oberflächen soll eine Übersicht über alle von der Steuerungssoftware empfangenen und an Diese gesendeten Nachrichten darstellen. Über diesen Teil der GUI soll es auch möglich sein per Hand Sensorereignisse an die Steuerungssoftware zu schicken.
Eine weitere Sektion der Oberfläche soll den aktuellen Status der Sensoren und Aktuatoren mit Hilfe von Kontrollleuchten darstellen.
Die Letzte Sektion der GUI wird eine perspektivische Ansicht der Anlage anzeigen. Diese Ansicht wird die reale Anlage in einer Art nachbilden, die es ermöglichen wird alle Vorgänge nachzuvollziehen, soll aber gleichzeitig einfach genug gehalten werden, um die Performance des Systems zu gewährleisten.

Kapitel 4

Implementierung

Die Simulationsumgebung wird in der Programmiersprache C unter Einsatz der Enwicklungsumgebung QNX Momentics 6.3 SP2 entwickelt. Für die Entwicklung der GUI kommt der, in die Momentics Umgebung integrierte, Photon Application Builder zum Einsatz. Über eine, mit Hilfe von Microsoft Virtual PC erstellte, virtuelle Installation des Echtzeitbetriebssystems QNX wird die Software in der für sie bestimmten Umgebung ausgeführt und getestet.

4.1 Vom Petri-Netz zum C-Code

Die Umsetzung von Petri-Netz Modellen in Code ist sehr einfach zu bewerkstelligen und soll hier kurz anhand des Beispiels in Abbildung 3.1 auf Seite 13 erläutert werden.
Zuerst werden alle Transitionen, anschließend die Plätze, an denen Aktionen stattfinden sollen behandelt.

```
// Variablen für die Plätze:
int Platz1 = 1; //Startmarkierung: eine Markierung befindet sich auf dem Platz
int Platz2 = 0;
int Platz3 = 0;

while(fertig == False){
        //Transition T1:
        if(Platz1 > 0 && B1 == True)
                Platz1 = 0; //Platz1 verliert seine Markierung
                Platz2 = 1; //Platz2 erhält eine Markierung
        //Transtion T2:
        else if(Platz2 > 0 && B2 == True)
                Platz2 = 0;
                Platz3 = 1;

        //Platz P2
        if(Platz2 > 0)
```

```
                A1();
        //Platz P3
        if(Platz3 > 0)
                fertig = True;
}
```

Mit dieser Methode lassen sich allgemein alle Petri Netze abbilden, also auch solche, die keine Zustandsgraphen sind und mehr als eine Markierung im Netz besitzen. Um andere Spezialfälle der Petri-Netze, wie zum Beispiel Kantengewichte, abzubilden müsste der Code noch entsprechend erweitert werden.
Aufgrund der Tatsache dass es sich bei den hier verwendeten Netzen ausschließlich um Zustandsgraphen handelt, ist es möglich und sinnvoll die Implementierung zwecks Übersichtlichkeit und Speichereinsparung zu vereinfachen. Da sich immer nur eine Markierung in den Netzen befinden kann, reicht es eine Variable für die Markierung einzuführen, und die Variablen für die einzelnen Plätze wegzulassen.

```
//Startmarkierung
int Platz = 1;

T_logical fertig = False;

while(fertig == False){
        //Transition T1:
        if(Platz == 1 && B1 == True)
                Platz = 2;
        //Transtion T2:
        else if(Platz == 2 && B2 == True)
                Platz = 3;

        //Platz P2
        if(Platz == 2)
                A1();
        //Platz P3
        if(Platz == 3)
                fertig = True;
}
return;
```

4.2 Datenstrukturen

4.2.1 Werkstücke

```
typedef enum cubePos{
        pos_bnd,
```

```
        pos_krn,
        pos_prs,
        pos_rfz,
        pos_stored,
        pos_down,
        pos_mgzn1,
        pos_mgzn2
} cubePos;

typedef enum cubeType{
        lowerPart, white,black, completeWhite, completeBlack
} cubeType;

typedef struct cube{
        cubePos position;
        int bndCoords;
        int storedCoords[2];
        cubeType type;
} cube;
```

Wie in der Design Phase beschrieben müssen für das Handling der Werkstücke verschiedene Attribute gespeichert werden. Dies geschieht über das hier vorgestellte struct cube. Bei der Erstellung der Werkstücke wird der benötigte Speicher allokiert, und ein Zeiger auf diesen Datenbereich in einer Liste abgespeichert. Will man eine Refernz auf ein bestimmtes Werkstück bekommen, so muss durch diese Liste iteriert werden.

4.2.2 Nachrichten

```
typedef enum {
        presse,
        rfz,
        kran,
        ie
}messageType;

typedef enum {
        in,
        out
}direction;

typedef struct message{
        int id;
        char description[MESSAGE_DESCRIPTION_LENGTH];
        messageType mType;
        direction direction;
        int internalAdditive[2];
} message;
```

Die ID der Nachricht ist die selbe, die auch in der Steuerungssoftware verwendet wird. Dies hat den Vorteil, dass nur die ID über die Schnittstelle geschickt werden muss, und anschließend zugeordnet werden kann.
Das Feld messageType dient dem Message Handler für die Zuordung der eingehenden Nachrichten zum richtigen Manager. Direction gibt an, ob es sich um eine Sensornachricht (out) oder um eine Aktuatornachricht (in) handelt.
Das Feld *internalAdditive* wird für interne Funktionen benötigt, um Werkstücke an beliebiger Stelle in der Anlage erzeugen zu können.
Beim Programmstart werden alle verfügbaren Nachrichten in eine Liste geladen.

4.2.3 Zustände der Anlage

Jeder einzelne Teilzustand der Anlage, also jeder einzelne Sensor und jeder Aktuator, wird durch eine Boolsche Variable repräsentiert. Positionszustände, in denen Koordinaten abgespeichert werden müssen, sind als Integer bzw. Integer Arrays implementiert. Durch die Verwendung von Gettern und Settern wird sichergestellt, dass diese Zustandsvariablen nicht direkt beschrieben und gelesen werden können. Zur Sicherung der Threadsicherheit werden diese Zugriffe durch Semaphoren geschützt.

4.3 Implementierung der Simulationssoftware

4.3.1 Message Handler

Der Message Handler besteht aus einem Empfangs Thread und einer Sende Routine, sowie aus jeweils einer Routine um die Message Queues zu öffnen und zu schließen.
Der Empfangs Thread blockiert auf einem `mq_receive()` und wartet damit auf eingehende Nachrichten. Die Empfangenen Message-ids werden im nächsten Schritt mit den gespeicherten Nachrichten abgeglichen, um die gesamte Message Struktur zu bekommen. Anschließend wird mittels des Attributes `messageType` der Nachricht der entsprechende Empfänger ermittelt. Nun kann der zuständige Manager gestartet werden, der die weitere Verarbeitung übernimmt. Sobald der Manager terminiert ist, kehrt der Empfangs Thread zu seinem blockierenden `mq_receive()` Aufruf zurück. Dies geschieht auf diese Weise, um zu verhindern dass mehrere Manager Routinen gleichzeitig aufgerufen werden, was zu Problemen bei der Auswertung der Vorbedingungen für die Worker Threads führen könnte.

4.3.2 Manager Routinen

Die Manager Routinen wurden wie im Abschnitt 4.1 auf Seite 44 beschrieben implementiert.

4.3.3 Worker Threads

Auch die Worker Threads wurden wie im Abschnitt 4.1 auf Seite 44 beschrieben implementiert. Nachfolgend werden nur die Aspekte der Implementierung beschrieben die von diesem Schema abweichen oder als besonders wichtig erachtet werden.

4.3.3.1 RFZ Worker

Die Bewegung des RFZ wird mit Hilfe von Zählvariablen, welche im Update Algorithmus nach einer bestimmten Zeitspanne in- bzw. dekrementiert werden, realisiert. Wie bereits erwähnt betragen die Schritte in horizontaler, vertikaler und in der z-Achse 40, 80 und 4. Die Zeitspanne beträgt für die horizontale Bewegung 0,4 Sekunden, für die vertikale Bewegung 0,25 Sekunden und für das Ausfahren der Gabel 0,25 Sekunden. Damit wird eine adäquate Simulation der Bewegungsgeschwindigkeit erreicht, und ermöglicht das rechtzeitige Reagieren auf eingehende Nachrichten.
Für die Behandlung der Sensoren musste eine Toleranz von ± 1 Schritt eingerichtet werden, um den Nachlauf, den die Anlage nach einem „stop“ Befehl in der Realität und in der Simulation hat, zu berücksichtigen. Diese Toleranz gilt in horizontaler und vertikaler Richtung.
Beim Einlagern eines Werkstücks wird zuerst das RFZ in die Position gebracht, dass ein y Positionssensor und ein x Positionssensor mit der angegebenen Toleranz aktiviert ist, anschließend die Gabel ausgefahren und ein Stück abgesenkt. Dieses Absenken kann unterschiedliche Ausmaße haben, damit das Einlagern funktioniert. Dies wurde so implementiert, dass das Absenken einen bis drei Schritte betragen muss, damit die Simulation ein Werkstück als eingelagert betrachtet.
Die selbe Toleranz existiert auch beim Auslagern eines Werkstücks. Die Gabel muss sich also einen bis drei Schritte unterhalb des entsprechenden y Positionssensors befinden, ausgefahren, und dann angehoben werden.

4.3.3.2 Kran Worker

Die Kreisbewegung des Krans ist durch eine Zählvariable, die in 360 Schritte hoch- bzw. runtergezählt werden kann realisiert. Der Zeitabstand zwischen den einzelnen Zählschritten beträgt aufgrund der großen Anzahl an Schritten und der schnellen Drehbewegung des Krans in der Realität nur 10 Millisekunden. Für die vertikale Richtung und das Ein- und Ausfahren werden 25 bzw. 250 Millisekunden verwendet.
Auch hier gibt es eine notwendige Toleranz für die Behandlung der Positionssensoren. Diese beträgt ± 2.
Die Übergabe der Werkstücke zwischen IE und Kran, Presse und Kran, RFZ und Kran in beide Richtungen funktioniert, indem bei den, im Design identifizierten, für die Übergabe zuständigen Zuständen, durch die Liste der existierenden Werkstücke iteriert wird, um zu

prüfen ob ein Werkstück in der entsprechenden Position bereit liegt. Anschließend wird die Position des Werkstücks aktualisiert.

4.3.3.3 Presse Worker

Für das Pressen der Werkstücke wird in der Funktionseinheit „Stempel senken“ durch die Liste der Werkstücke iteriert, um herauszufinden, ob ein Werkstück vom Typ Unterschale und ein Werkstück vom Typ Oberschale in der Presse bereit liegt. Ist das der Fall werden die beiden Werkstücke aus der Liste entfernt, die Liste neu organisiert, um die Lücken zu füllen und anschließend ein neues Werkstück vom Typ „Komplett Weiß“ oder „Komplett Schwarz“, je nachdem welche Farbe die Oberschale hatte, und in die Liste eingefügt.

4.3.3.4 IE Worker

Das Erzeugen von Werkstücken in den Magazinen wird vom User durch eine Aktion in der GUI angestoßen. Dabei wird eine Datenstruktur mit den folgenden Werten initialisiert:

- position = `pos_mgzn1`oder `pos_mgzn2`, je nachdem ob eine Oberschale oder eine Unterschale erstellt wurde.
- bndCoords = -1 weil sich das Werkstück nicht auf dem Förderband befindet.
- storedCoords[] = [-1,-1] weil das Werkstück nicht eingelagert ist.
- type = `lowerPart`, `white`oder `black`, je nachdem was erstellt wurde.

Das Ausschieben der Werkstücke aus den Magazinen funktioniert gleich wie die Übergaben des Krans. Wieder wird durch die Liste iteriert, und beim betroffenen Werkstück werden die entsprechenden Attribute geändert. Befindet sich ein Werkstück auf dem Band, so wird bei einer Bewegung des Bandes in bestimmten Zeitabständen das Attribut „bndCoords“ aktualisiert - wie bereits gesagt bewegt sich das Band selber nicht im eigentlichen Sinne.
Das Aus- und Einfahren des Prüfzylinders wird anhand des Wissens der Simulation über das Werkstück, welches unter dem Sensor liegt bewerkstelligt. Ist das Werkstück vom Typ Unterschale, wird der Zylinder komplett ausgefahren, ist das Werkstück anderen Typs, dann nicht.

4.3.4 Erzeugen und Entfernen von Werkstücken

Werkstücke können an bestimmten Positionen der Anlage erzeugt werden. Diese Funktion wird von dem Worker übernommen, der den Teil der Anlage repräsentiert, wo das Werkstück erstellt werden soll. Wenn nicht anders angegeben können alle Arten von Werkstücken platziert werden.

- In allen Lagerplätzen des Hochregals.
- Auf der Gabel des RFZ.
- Am Sauger des Krans, wenn dieser eingeschaltet ist.
- Auf dem Ausschub der Presse. Hier kann entweder ein kompletter Würfel, oder jeweils ein Werkstück vom Typ Unterschale, und eines vom Typ Oberschale platziert werden.
- Am Ende des Förderbandes, wo sich die Lichtschranke befindet. Hier können nur Oberschalen und Unterschalen platziert werden.
- In den beiden Magazinen. Hier können nur die dafür vorgesehenen Werkstücke platziert werden.

4.3.5 Konfigurationsoptionen

4.3.5.1 Startpositionen

Über Funktionen in der GUI ist es notwendig am Anfang der Simulation, und möglich während der Simulation die Positionen der einzelnen Komponenten zu setzen.

4.3.5.2 Ausschluss einzelner Anlagenteile aus der Simulation

Am Anfang der Simulation kann man auswählen, welche der vier Teile der Anlage simuliert werden sollen. Soll ein Teil der Anlage nicht simuliert werden, so werden die Eingehenden Nachrichten, die diesen Teil betreffen vom Message Handler ignoriert. Somit wird auch der betreffende Manager und damit auch der Worker niemals gestartet.

4.3.6 Interne Nachrichten

Einige Funktionen der GUI wurden so Implementiert, dass sie Nachrichten an die Aktuatoren Queue senden, um bestimmte Aktionen herbeizurufen. Diese Aktionen sind das Hinzufügen und entfernen von Werkstücken. Die internen Nachrichten sind für den Benutzer nicht von Belang, und werden auch in der GUI nicht sichtbar gemacht. Für die Verarbeitung dieser Nachrichten wurden in den entsprechenden Managern und Workern Funktionen hinzugefügt, die im Design mit den Petri-Netzen nicht auftauchen.

4.4 Threadsicherheit

Die Threadsicherheit der Statusvariablen ist damit sichergestellt, dass die Statusvariablen nicht direkt geschrieben und gelesen werden können. Für den Zugriff werden Getter und

Setter Funktionen bereitgestellt, in denen der Zugriff durch Semaphoren geschützt wird.
Auch für die Liste und die Datenstruktur der Werkstücke werden Semaphoren eingesetzt, da auch diese potentiell gleichzeitig von mehreren Threads verändert werden können.
Zusätzlich zur Sicherung der Statusvariablen wurden Semaphoren für jede einzelne Bewegungsebene (horizontal, vertikal, z-Achse) der einzelnen Anlagenteile implementiert. Es existiert eine geringe Wahrscheinlichkeit, dass die Kontrolle der Manager über die Bewegungsebenen nicht ausreichend ist. Erläutert an einem Beispiel ist dies dann der Fall, wenn gerade ein Kran Worker einen Befehl zum Drehen im Uhrzeigersinn ausführt, und dann in sehr schneller Abfolge ein stop Befehl, und ein weiterer Befehl zum Drehen im Uhrzeigersinn kommt. Dann kann es passieren, dass ein neuer Worker zum Abarbeiten des Befehls gestartet wird, bevor der alte Worker den Abbruch seines Befehls registriert hat. Es würden dann also zwei Worker den selben Befehl ausführen, was zu kritischen Fehlern in der Sensor-, und Positionsverarbeiten führen kann. Um dieses Szenario auszuschließen wird mittels der Semaphoren sichergestellt, dass immer nur ein Worker zur Zeit auf einer Bewegunsebene arbeitet.

4.5 GUI

Die GUI Komponente der Simulationssoftware wurde mit Hilfe des Photon Application Builders, und ebenfalls mit der Programmiersprache C unter Verwendung von QNX Momentics 6.3 SP2 erstellt.

4.5.1 Nachrichtenteil

Auf dem Screenshot (Abb. 4.1) sind zwei Textfelder sowie vier drop-down Menüs erkennbar. Im oberen Textfeld werden alle Nachrichten angezeigt, die von der Steuerungssoftware empfangen wurden, im Unteren die, die an die Steuerung geschickt wurden. Im oberen Textfeld erscheinen also alle Befehle an die Aktuatoren, im Unteren alle Sensorereignisse, die aufgetreten sind.
Die vier drop-down Menüs ermöglichen es dem Benutzer manuell Sensorereignisse an die Steuerung zu verschicken. Dies dient der besseren Fehlersuche während der Entwicklungsphase der Steuerung, da dadurch Sensorereignisse, die Aufgrund eines Fehlers in der Steuerung nicht aufgetreten, oder nicht richtig bearbeitet worden sind, zu erzeugen. Am aktuellen Zustand der Simulation ändert sich durch das Schicken dieser Ereignisse nichts. Wenn zum Beispiel ein Sensorereignis erzeugt wird, das signalisiert, dass der Kran vollständig ausgefahren ist, bedeutet dies nicht, dass der Zustand des Krans entsprechend gesetzt wird.

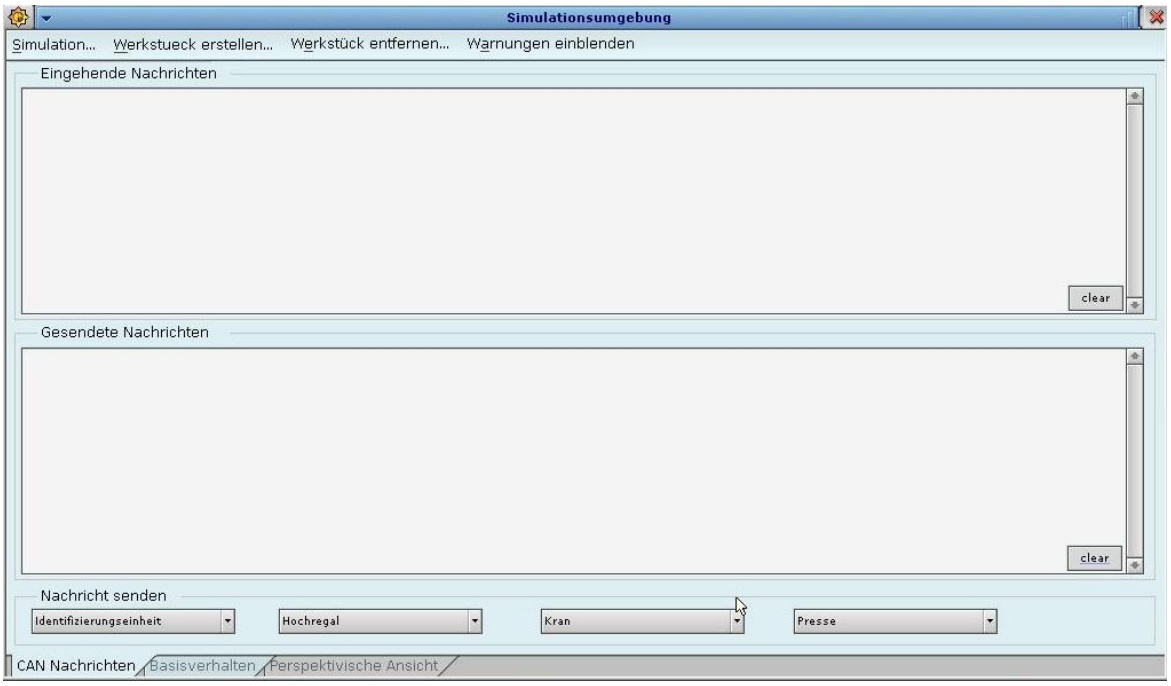


Abbildung 4.1: Screenshot: Nachrichtentab

4.5.2 Basisteil

Dieser Tab (Abb. 4.2) stellt den aktuellen Zustand der Sensoren und Aktuatoren mit Hilfe von Lampen und Switches dar. Die Sektion ist in vier Teile unterteilt, wovon jeder einen der Anlagenteile repräsentiert. Jede dieser vier Sektionen ist wiederum in jeweils einen Teil für die Sensoren und einen Teil für die Aktuatoren aufgeteilt. Ist eine Lampe bzw ein Switch rot, bedeutet das, der Aktuator bzw. der Sensor ist nicht aktiv, grün bedeutet aktiv, und grau bedeutet, dass dieser Sensor bzw. Aktuator gar nicht simuliert wird, also komplett abgeschaltet ist.
Die Sensoren sind mit Hilfe von Switches implementiert worden, was es dem Benutzer ermöglicht den Zustand der Anlage manuell zu verändern. Klickt er auf einen dieser Switches wir dieser Sensor aktiviert bzw. deaktiviert, wodurch, anders als beim Nachrichtentab, nicht nur das Sensorereignis gesendet wird, sondern sich der Zustand des Sensors, und damit der Zustand der Anlage ändert.

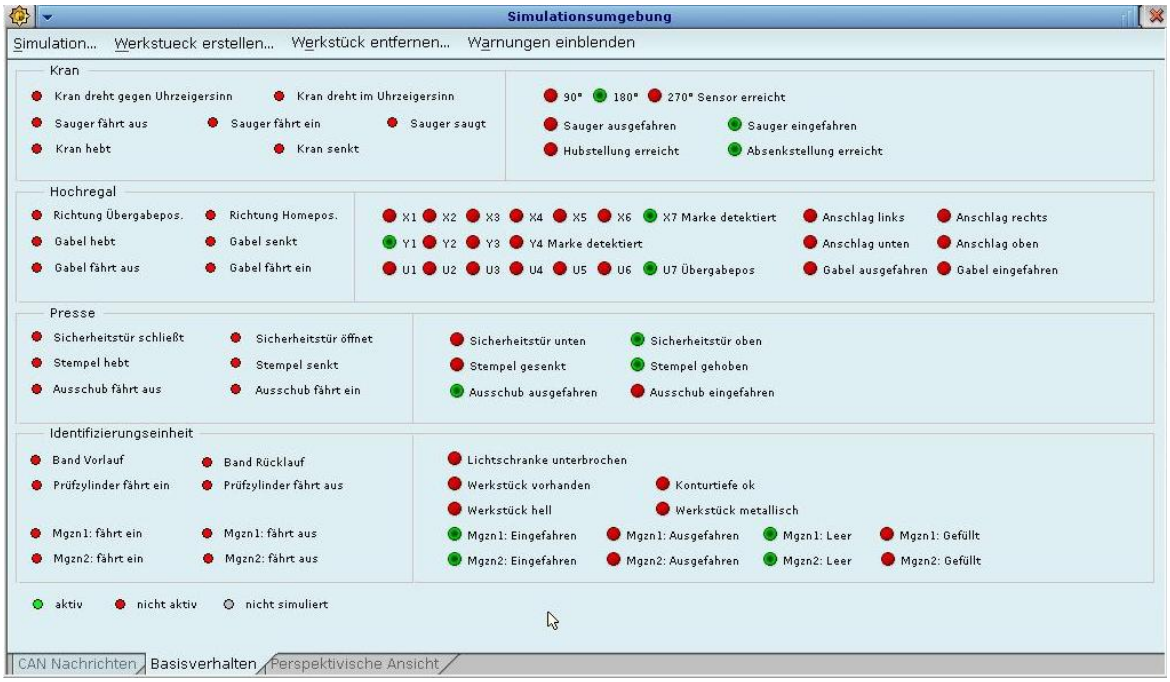


Abbildung 4.2: Screenshot: Basistab

4.5.3 Perspektivische Ansicht

4.5.3.1 Aufbau

Die gesamte GUI wurde mit Hilfe von den von den von Photon bereitgestellten Widgets aufgebaut (8). Einige der Elemente sind statisch, andere können durch Zugriff auf die Parameter der Widgets in Aussehen und Position verändert werden. Für eine solche Veränderung der Widgets zur Laufzeit wird von Photon die Funktion `PtSetResources()` bereitgestellt, die als Argumente eine Referenz auf das zu verändernde Widget, die Anzahl der zu verändernden Parameter und eine `PtArg_t`Struktur erwartet. Diese Struktur besteht aus einem Identifier für den Parameter, und dem neuen Wert für Diesen.
Das RFZ besteht aus einer statischen Visualisierung des Regalsystems, und zwei dynamischen Teilen, die die Gabel und die Stange darstellen.
Der Kran besteht nur aus den beweglichen Teilen, die die Stange, den Ausleger und den Sauger repräsentieren.
Die Presse hat ein statisches Gehäuse, und weiterhin die dynamischen Teile der Sicherheitstür, des Ausschubes, und des Stempels.
Die Identifizierungseinheit besteht aus einem statischen Förderband, aus zwei Pfeilen, die die Bewegung des Bandes wiedergeben, den zwei Magazinen mit jeweils einem dynamischen Ausschub und einer Linie am Ende des Bandes, welche die Lichtschranke darstellt,

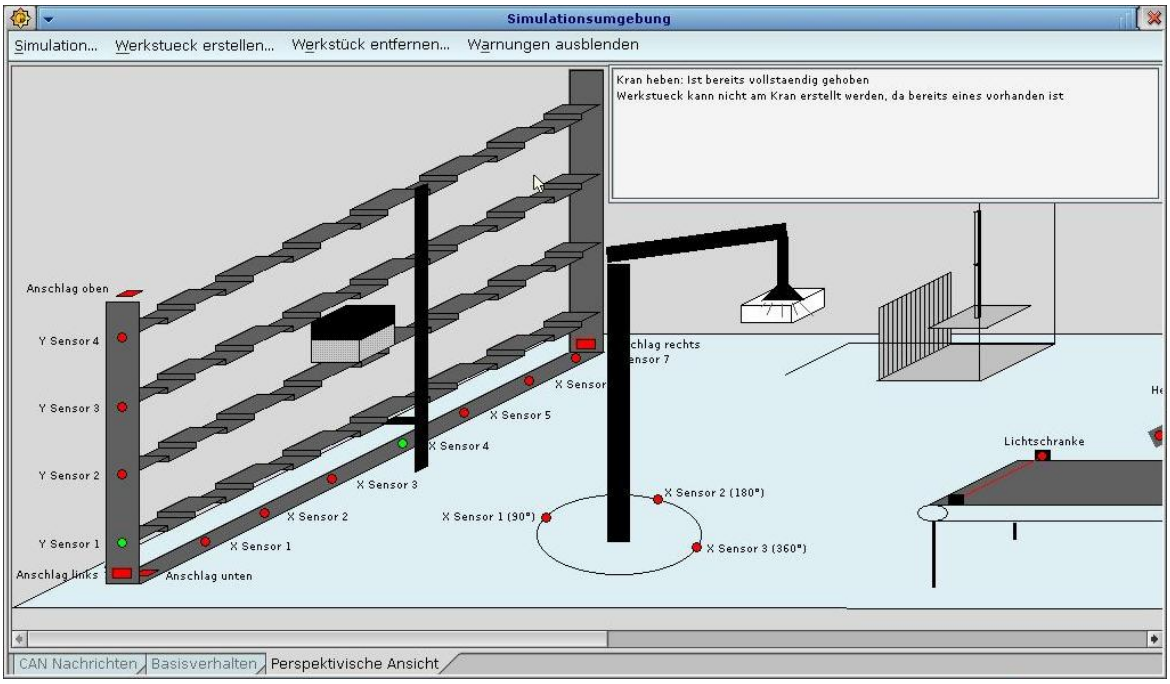


Abbildung 4.3: Screenshot: Perspektiventab

und deren Farbe verändert werden kann.
Die Sensoren der Anlage werden durch Lämpchen dargestellt, deren Farbe zwischen Rot und Grün gewechselt wird.

4.5.3.2 Realisierung der Bewegungen

Die Bewegungen der Anlage können recht einfach realisiert werden, indem die tatsächlichen Positionen der einzelnen Teile, unter Berücksichtigung der Dimensionen der perspektivischen Ansicht, in für die GUI zutreffende Positionen umgerechnet werden.
Die Worker Threads rufen bei jedem Bewegungsschritt Funktionen zum aktualisieren der Positionen innerhalb der GUI auf. Die GUI führt also keine kontinuierlichen Bewegungen durch, weil es dafür notwendig wäre die Statusvariablen der Simulation zu pollen, was sich extrem schlecht auf die Performance auswirken würde.

4.5.3.3 Handling der Werkstücke

Ein Werkstück ist ein Label mit einem Bild, welches das entsprechende Werkstück darstellt. Die GUI muss ihre eigenen Referenzen der vorhandenen Werkstücke verwalten, und kann nicht auf die Liste der Simulation zugreifen. Für jedes der Elemente der Anlage existiert eine Referenz auf das Werkstück, das sich gerade dort befindet, oder bei mehreren Werkstücken,

die im Lager und in den beiden Magazinen existieren können, die Referenz auf eine Liste. Soll nun Beispielsweise der Kran bewegt werden, so wird innerhalb der Bewegungsroutine überprüft ob die Referenz auf das Werkstück `NULL` ist, oder nicht. Ist eine Referenz vorhanden, so wird zusätzlich zu den eigentlichen Elementen des Krans auch die Position des Werkstücks berechnet und aktualisiert.
Findet eine Übergabe zwischen den Teilen der Anlage statt, so wird einfach die Referenz vom einen Anlagenteil auf den anderen kopiert, und zukünftig wird das Werkstück zusammen mit dem neuen „Besitzer“ bewegt.

4.5.4 Übergreifende Elemente

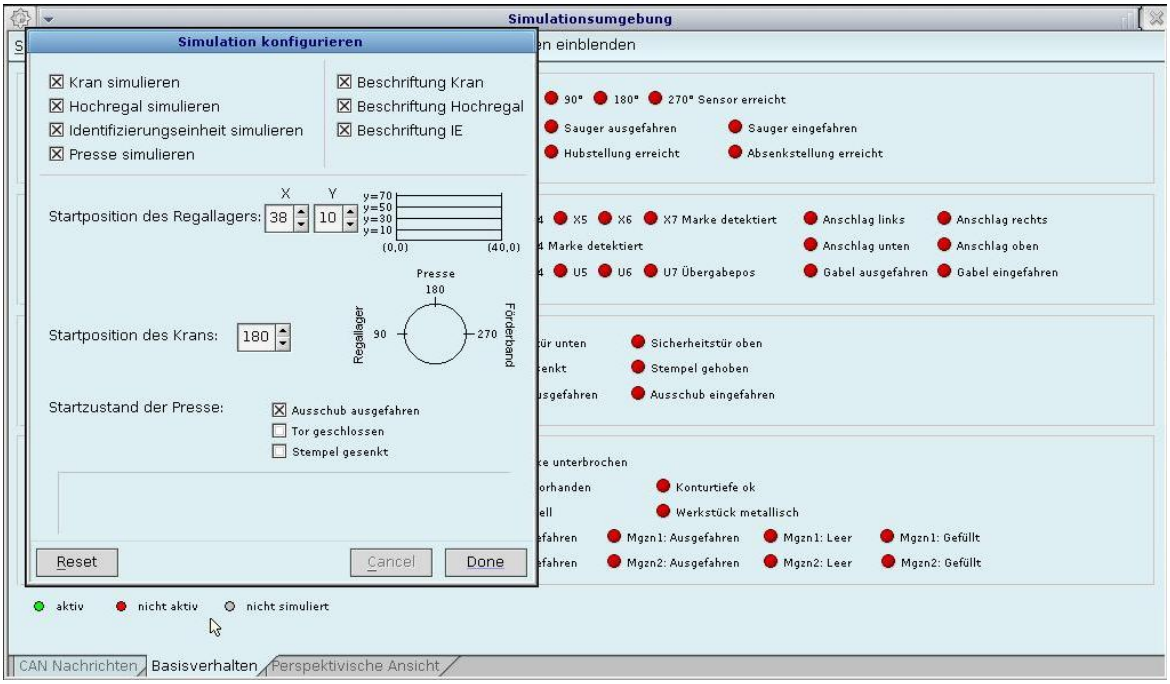


Abbildung 4.4: Screenshot: Konfiguration

Beim Start der Simulationsumgebung wird der in Abb. 4.4 erkennbare Konfigurationsbildschirm aufgerufen. Dort kann festgelegt werden welche Teile der Anlage simuliert werden sollen und ob die Perspektivische Ansicht beschriftet werden soll. Weiterhin können die Startzustände von RFZ, Kran und Presse ausgewählt werden.
Dieser Konfigurationsbildschirm kann auch während der Programmlaufzeit über das Simulations-Menü aufgerufen werden. Eine Änderung dieser Konfigurationsdaten hat aber auch immer einen Neustart der Simulation zur Folge, das heißt alle Bewegungen werden gestoppt, und alle hier nicht konfigurierbaren Zustände werden auf die Standardeinstellung

zurückgesetzt.
Ein weiterer übergreifender Teil der GUI ist das Fenster in dem Warnungen der Simulationsumgebung ausgegeben werden. Dieses in Abb. 4.3 erkennbare Textfeld ist über alle drei Tabs hinweg sichtbar, und kann mit Hilfe des Menübuttons *Warnungen ausblenden* ausgeblendet werden. Tritt eine neue Warnung auf, so wird das Textfeld automatisch eingeblendet. Übergreifend ist auch die Menüleiste, die neben den schon erklärten Funktionen auch noch die Möglichkeiten bietet die Simulation zu beenden sowie Werkstücke beliebigen Typs frei in der Anlage zu platzieren und wieder zu entfernen.

4.5.5 Interface

Dieser Teil der GUI wurde austauschbar designed, und es wurde im Vorfeld ein Interface definiert, welches von der GUI implementiert werden muss. Im Folgenden wird dieses Interface erläutert.

```
void gui_addIncomingMessage(message m);
void gui_addOutgoingMessage(message m);
void gui_addOutgoingMessageCB(message m);

void gui_init(T_logical kran, T_logical rfz, T_logical presse, T_logical ie);
```

Diese Funktionen stellen das Interface zu dem Teil der GUI dar, der für das Anzeigen der empfangenen und gesendeten Nachrichten zuständig ist.
die Funktion `gui_init` wird benötigt um einzustellen, welche Teile der Anlage simuliert werden sollen.

```
/* RFZ */

void gui_setRFZ_xPos(int xPos);
void gui_setRFZ_yPos(int yPos);
void gui_setRFZ_zPos(int zPos);
void gui_setRFZ_xPosSensor(int nummer, T_logical active);
void gui_setRFZ_uebergabeSensor(int nummer, T_logical active);
void gui_setRFZ_yPosSensor(int nummer, T_logical active);
void gui_setRFZ_endschalterLinks(T_logical active);
void gui_setRFZ_endschalterRechts(T_logical active);
void gui_setRFZ_endschalterUnten(T_logical active);
void gui_setRFZ_endschalterOben(T_logical active);
void gui_setRFZ_kollisionRegal(int Strebe);
void gui_setRFZ_kollisionKran();
void gui_setRFZ_kollisionWuerfel(int x,int y);

void gui_setRFZ_addWerkstueck(cubeType type);
void gui_setRFZ_removeWerkstueck();
void gui_setRFZ_addWerkstueckRegal(int x, int y,cubeType type);
void gui_setRFZ_removeWerkstueckRegal();
```

```
void gui_setRFZ_einlagern(int x, int y);
void gui_setRFZ_auslagern(int x, int y);

void gui_initRFZ(); //Eventuelle initialisierungen
```

Die ersten drei Funktionen sind zuständig für das setzen der Position in den drei Dimensionen. `gui_setRFZ_xPosSensor(int nummer, T_logical active)` setzt den Zustand der x Positionssensoren, wobei `nummer`, von links ab null gezählt, den Sensor angibt, der gemeint ist, und `active` angibt, ob der Sensor ein- oder ausgeschaltet werden soll. `gui_setRFZ_uebergabeSensor(int nummer, T_logical active)` und `gui_setRFZ_yPosSensor(int nummer, T_logical active)` funktionieren auf die gleiche Weise. Die nächsten vier Funktionen setzen den Zustand der vier Endschalter.
Der Parameter `Strebe` der Funktion, die eine Kollision zwischen RFZ und Regal in der GUI sichtbar machen soll, gibt den Regalboden, gezählt ab null von unten an, mit dem die Kollision aufgetreten ist. `gui_setRFZ_kollisionKran()` benötigt keine Parameter, da eine Kollision zwischen RFZ und Kran immer eindeutig ist. Die x und y Parameter der Funktion `gui_setRFZ_kollisionWuerfel(int x,int y)` geben die aktuelle Position des RFZ an, woraus die GUI das betroffene Werkstück ermitteln kann.
Die beiden Parameter von `gui_setRFZ_einlagern(int x, int y)` und `gui_setRFZ_auslagern(int x, int y)` geben den Lagerplatz an, gezählt von unten links. Diese Funktionen lagern Werkstücke von der Gabel ins Regal ein, und umgekehrt.
Die Funktionen `gui_setRFZ_addWerkstueck()`, `gui_setRFZ_removeWerkstueck()`, `gui_setRFZ_addWerkstueckRegal(int x, int y)` und `gui_setRFZ_removeWerkstue` werden für das manuelle setzen von Werkstücken durch den User benötigt. Die Funktion zum entfernen entfernt dabei immer das zuerst gesetzte Werkstück im Regal.
Die Funktion `gui_initRFZ()` schließlich ist dafür zuständig eventuell noch erforderliche Initialisierungen der GUI vorzunehmen.
Alle Funktionen, die die Position und die Sensoren betreffen, werden bei Programmstart einmal mit den Startwerten der Simulation aufgerufen. Alle Funktionen werden im Laufe der Simulation immer dann, wenn sich der betroffene Wert ändert, aufgerufen.

```
/* KRN */

void gui_setKRN_xPos(int xPos); // 0 bis 359
void gui_setKRN_yPos(int yPos);
void gui_setKRN_zPos(int zPos);
void gui_setKRN_sensor(int nummer, T_logical active);
void gui_setKRN_sauger_saugt(T_logical active);
void gui_setKRN_kollisionPresse();
void gui_setKRN_kollisionRfz();
void gui_setKRN_kollsionUeberdreht();
```

```
void gui_setKRN_werkstueckFromBnd();
void gui_setKRN_werkstueckToBnd();
void gui_setKRN_werkstueckFromPrs();
void gui_setKRN_werkstueckToPrs();
void gui_setKRN_werkstueckFromRfz();
void gui_setKRN_werkstueckToRfz();
void gui_setKRN_werkstueckDown();

void gui_setKRN_addWerkstueck(cubeType type);
void gui_setKRN_removeWerkstueck();

void gui_initKRN();
```

Die drei ersten Funktionen setzen wieder die aktuellen Koordianten des Krans. `gui_setKRN_sensor(int nummer, T_logical active)` funktioniert wie die drei Sensorfunktionen des RFZ und setzt den Status der Sensoren der horizontalen Kreisbewegung. Die Funktion `gui_setKRN_sauger_saugt(T_logical active)` gibt der GUI an, ob der Suager aktiv ist oder nicht.
Alle drei Funktionen, die die Kollisionen mit der Presse, mit dem RFZ und das überdrehen des Krans visualisieren, brauchen keine Parameter, da sie eindeutig sind.
Die folgenden sieben Funktionen visualisieren das Handling der Werkstücke durch den Kran. Da der Kran die zentrale Funktionseinheit der Anlage ist, die alle anderen Teile miteinander verbindet, gibt für alle drei Teile der Anlage jeweils eine Funktion für die Übergabe vom Kran bzw. zum Kran. Zusätzlich existiert eine Funktion, die herunter gefallene Werkstücke visualisieren soll.
Die beiden nächsten Funktionen sind wieder für das manuelle setzen und entfernen von Werkstücken vorgesehen.
Wieder existiert eine Funktion zur Initialisierung, und wieder werden diese Funktionen, wie beim RFZ erklärt, einmal am Anfang der Simulation und dann zur Laufzeit, wenn sich Werte ändern, aufgerufen.

```
/* PRS */

void gui_setPRS_tuerGeschlossen(T_logical geschlossen);
void gui_setPRS_ausschubAusgefahren(T_logical ausgefahren);
void gui_setPRS_stempelGesenkt(T_logical gesenkt);
void gui_setPRS_kollision();

void gui_setPRS_werkstueckPressen(cubeType type);

void gui_setPRS_addWerkstueck(cubeType type);
void gui_setPRS_removeWerkstuck();
```

```
void gui_initPRS();
```

Die ersten vier Funktionen visualisieren die Zustände der Presse, und den Fall, wenn innerhalb der Presse eine Kollision auftritt.
Die Funktion `gui_setPRS_werkstueckPressen(cubeType type)` informiert die GUI, dass jetzt die beiden Werkstücke welche sich in der Presse befinden durch ein anderes vom Typ `type` ersetzt werden müssen. Dass zwei Werkstücke in der Presse vorhanden sind, wird von der Simulation sichergestellt, und muss in der GUI nicht noch mal überprüft werden.

```
/* BND */

void gui_setBND_linkslauf(T_logical active);
void gui_setBND_rechtslauf(T_logical active);
void gui_setBND_helligkeitsSensor(T_logical active);
void gui_setBND_metallSensor(T_logical active);
void gui_setBND_konturSensor(T_logical endstellung);
void gui_setBND_konturSensorAus();
void gui_setBND_lichtschranke(T_logical active);
void gui_setBND_magazin1Auschub(T_logical ausgefahren);
void gui_setBND_magazin2Auschub(T_logical ausgefahren);

void gui_setBND_werkstueckToMagazin1(cubeType typ);
void gui_setBND_werkstueckToMagazin2();
void gui_setBND_werkstueckFromMagazin1ToBnd();
void gui_setBND_werkstueckFromMagazin2ToBnd();
void gui_setBND_addWerkstueck(int coords, cubeType typ);
void gui_setBND_removeWerkstueck();
void gui_setBND_werkstueckPos(int pos);

void gui_initBND();
```

Die ersten neun Funktionen geben den aktuellen Zustand der Identifizierungseinheit wieder. Da der Kontursensor die Besonderheit hat, dass er drei Zustände annehmen kann, nämlich *eingefahren, ausgefahren* und *halb ausgefahren*, wird für das Visualisieren die Funktion `gui_setBND_konturSensor(T_logical endstellung)` mit dem Parameter, der angibt ob der Sensor nur halb oder ganz ausgefahren ist, und die Funktion `gui_setBND_konturSensorAus()` benötigt.
`werkstueckToMagazin1`, `werkstueckToMagazin2`, `addWerkstueck` und `removeWerkstueck` stellen das durch den User angestoßene Hinzufügen und Entfernen von Werkstücken dar. Die Funktion `werkstueckPos` mit ihrem Argument `pos` aktualisiert die Position des aktuellen Werkstücks, das sich auf dem Band befindet, da sich das Band an sich nicht bewegt. Wird eine `-1` übergeben, bedeutet das, dass das Werkstück hinten vom Band heruntergefallen ist.

```
void gui_BeschriftungRFZ(T_logical beschriftung_RFZ);
```

```
void gui_BeschriftungKRN(T_logical beschriftung_KRN);
void gui_BeschriftungPRS(T_logical beschriftung_PRS);
void gui_BeschriftungBND(T_logical beschriftung_BND);
```

Durch diese Schalter kann die Beschriftung der Sensoren innerhalb der perspektivischen Anschicht an- und abgeschaltet werden.

```
void gui_LampeAn(gui_Funktionsart f);
void gui_LampeAus(gui_Funktionsart f);
void gui_LampeDisable(gui_Funktionsart f);
void gui_LampeEnable(gui_Funktionsart f);

void gui_SwitchAn(gui_Funktionsart f);
void gui_SwitchAus(gui_Funktionsart f);
void gui_SwitchDisable(gui_Funktionsart f);
void gui_SwitchEnable(gui_Funktionsart f);
```

Dieses Interface beschreibt die Funktionen zur Steuerung der Lampen und Switches im Basistab der GUI, welche den aktuellen Gesamtzustand der Anlage darstellen.
(9) (10)

Kapitel 5

Tests

5.1 Test der Schnittstelle

Die Schnittstelle zwischen Simulationsumgebung und Steuerungssoftware wurde mit Hilfe der in der Steuerung vorhandenen Möglichkeit einzelne Nachrichten per Knopfdruck zu senden getestet. Dafür wurde in der Simulationsumgebung eine Ausgabe aller Eingehenden Nachrichten erstellt, wodurch sichtbar wurde, ob alle Nachrichten im richtigen Kontext ankommen. Zusätzlich wurde in der Simulation die Möglichkeit geschaffen, um ausgehende Nachrichten per Knopfdruck zu versenden, in der Implementierung der Steuerung war die Funktionalität zur Anzeige von eingehenden Nachrichten bereits vorhanden.

5.2 Test der Worker

Die Worker konnten weitgehend unabhängig voneinander getestet werden, weil sie weitgehend voneinander unabhängige Anlagenteile repräsentieren. Es wurde eingehend getestet, ob sich nach Empfang verschiedener Kombinationen von Nachrichten das gewünschte Anlagenverhalten ergab.

5.3 Test der gesamten Simulation

Für den Gesamtsystemtest wurde eine bereits beinahe vollständig vorhandene automatische Steuerung erweitert und verwendet. Diese Steuerung beinhaltete die Funktionalität auf Knopfdruck ein gewünschtes Werkstück zu produzieren, also entweder einen Würfel mit weißer Oberschale, oder einen mit schwarzer Oberschale.
Für diesen Zweck wurde zuerst eine Unterschale von der Identifizierungseinheit angefordert, die IE hat sichergestellt, dass es sich wirklich um eine Unterschale handelte, und Diese zum Übergabepunkt mit dem Kran transportiert. Der Kran hat sich die Unterschale dort abgeholt,

und sie bei der Presse abgeliefert.
Anschließend wurde zuerst das RFZ gefragt, ob die benötigte Oberschale bereits dort zwischengelagert war. War dies der Fall, so wurde sie aus dem Regal abgeholt, und zur Presse transportiert. Andernfalls wurde eine Oberschale der gewünschten Farbe von der IE angefordert. Diese hat das Werkstück identifiziert, und zum Übergabepunkt zwischen Kran und Band befördert. Handelte es sich um die richtige Farbe, so wurde die Oberschale ebenfalls zur Presse transportiert. War es die falsche Farbe, so wurde sie im Regallager zwischengelagert, und es wurde eine neue Oberschale von der IE angefordert.
In der Presse wurden die beiden Werkstücke zusammengepresst, und der fertige Würfel wurde im Regal eingelagert. Danach war die Anlage wieder bereit für neue Aufträge.

Kapitel 6

Zusammenfassung

Ein Ziel dieser Bachelorarbeit war es ein bestehendes Steuerungssystem, welches über eine CAN-Schnittstelle mit einer Produktionsanlage kommuniziert, um die Funktionalität der Kommunikation mit einer Simulationsumgebung mittels Softwareschnittstelle zu erweitern, ohne die bestehende Software grundlegend zu verändern. Dieses Ziel wurde erreicht, indem die Software zusätzlich zum CAN-Treiber um zwei Posix Message Queues erweitert wurde. Der Rest der Struktur des bestehenden Programms, inklusive des Nachrichtenformats und der Schnittstelle zur höheren Softwareschicht über dem CAN-Treiber blieben davon unberührt.
Das Ziel zur Simulation der gesamten Anlage wurde im Rahmen der Anforderungen durch eine visualisierte und interaktive Simulationsumgebung erreicht. Diese Umgebung ermöglicht das Entwickeln einer Automatischen Steuerung für die Anlage unter Berücksichtigung von Teilsimulationen, Kollisionsbehandlung, Werkstücktransport und Werkstückproduktion.
Weiterhin kann die Implementierung der GUI unter Verwendung des zur Verfügung gestellten Interfaces durch eine andere ersetzt werden.

Anhang A

Bedienungsanleitung

A.1 Start der Simulationsumgebung

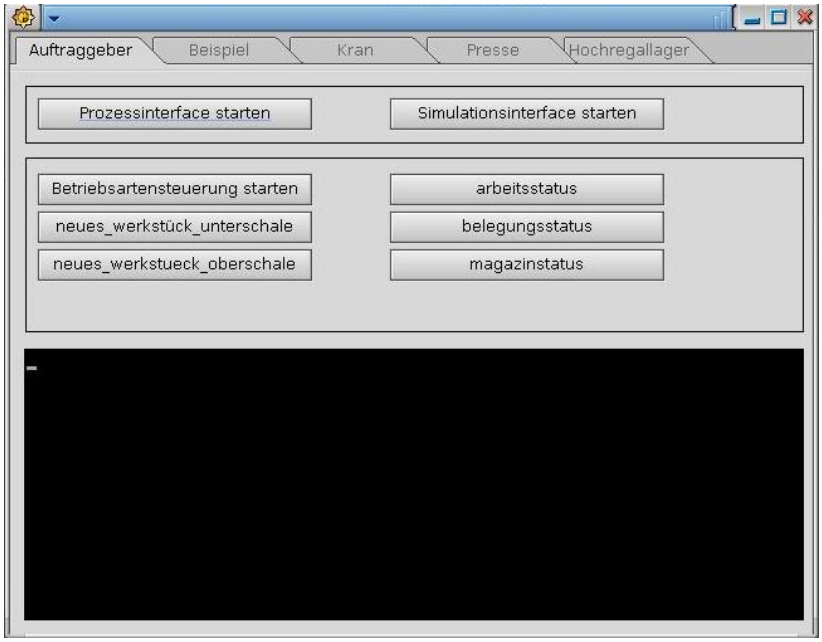
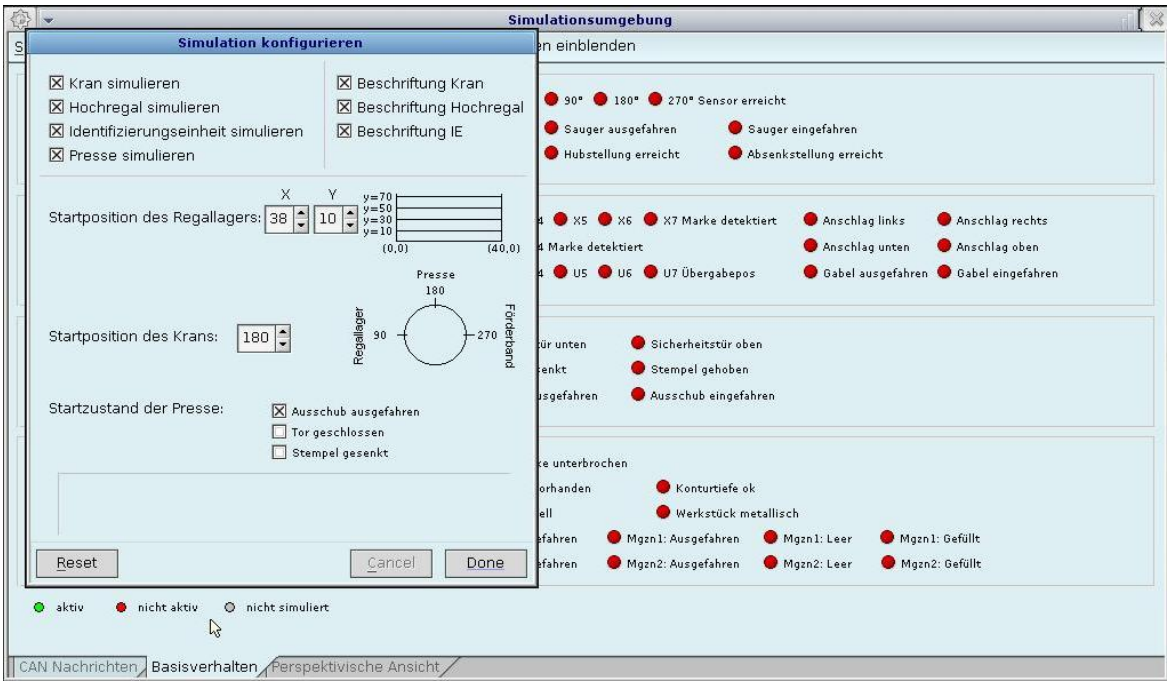


Abbildung A.1: Screenshot: Steuerungssoftware

In der Steuerungssoftware kann zwischen dem Start des CAN Interfaces zur Anlage und dem Start der Simulationssoftware gewählt werden. (Abb. A.1)
Durch einen Klick auf der Start Button öffnet sich die Benutzerschnittstelle der Simulation, und ein Konfigurationsbildschirm wird angezeigt (Abb. A.1). Hier muss eingestellt werden, welche Teile der Anlage simuliert werden sollen. Eingehende Nachrichten für nicht simulierte Teile werden von der Software ignoriert, und somit können diese Teile nicht bewegt werden,



lösen aber auch keine Kollisionen mit anderen Teilen der Anlage aus.
Rechts neben diesen Einstellungen kann gewählt werden, ob und wenn ja welche Teile der Anlage beschriftet werden sollen. Beschriftet werden nur die Sensoren, die anderen Komponenten sind ohne Beschriftung erkennbar.
Als Nächstes wird die Startposition des RFZ eingestellt. Der x Wert bestimmt die horizontale Position und kann zwischen 0 und 40 gesetzt werden, der y Wert bestimmt die vertikale Position und kann zwischen 0 und 80 variiert werden. Die Positionssensoren in der Horizontalen befinden sich an den Stellen x = 4, x = 9, x = 15, x = 21, x = 27, x = 33 und x = 38. Die beiden Endschalter links und rechts befinden sich bei x = 0 bzw. x = 40. Auf der y Ebene befinden sich die Sensoren bei y = 10, y = 30, y = 50 und y = 70. Die beiden Endschalter befinden sich bei y = 0 bzw. y = 80.
Die Startposition des Krans wird als Winkel eines Kreises angegeben. Der Sensor auf der Höhe des RFZ befindet sich bei 90°, der Sensor bei der Presse bei 180° und Der Sensor beim Förderband liegt bei 270°.
Für den Startzustand der Presse kann ausgewählt werden, ob der Ausschub ausgefahren ist, ob das Tor geöffnet oder geschlossen ist und ob der Stempel zum Pressen der Werkstücke gehoben oder gesenkt ist.
Durch einen Klick auf den Button *Reset* werden die hier Abgebildeten Standardwerte wiederhergestellt. Nach einem Klick auf *Done* ist die Simulation betriebsbereit.
Zu beachten ist, dass die Simulation keine Nachrichten der Steuerungssoftware entgegen

nimmt und bearbeitet, solange die Startkonfiguration nicht abgeschlossen ist.

A.2 Tab: CAN Nachrichten

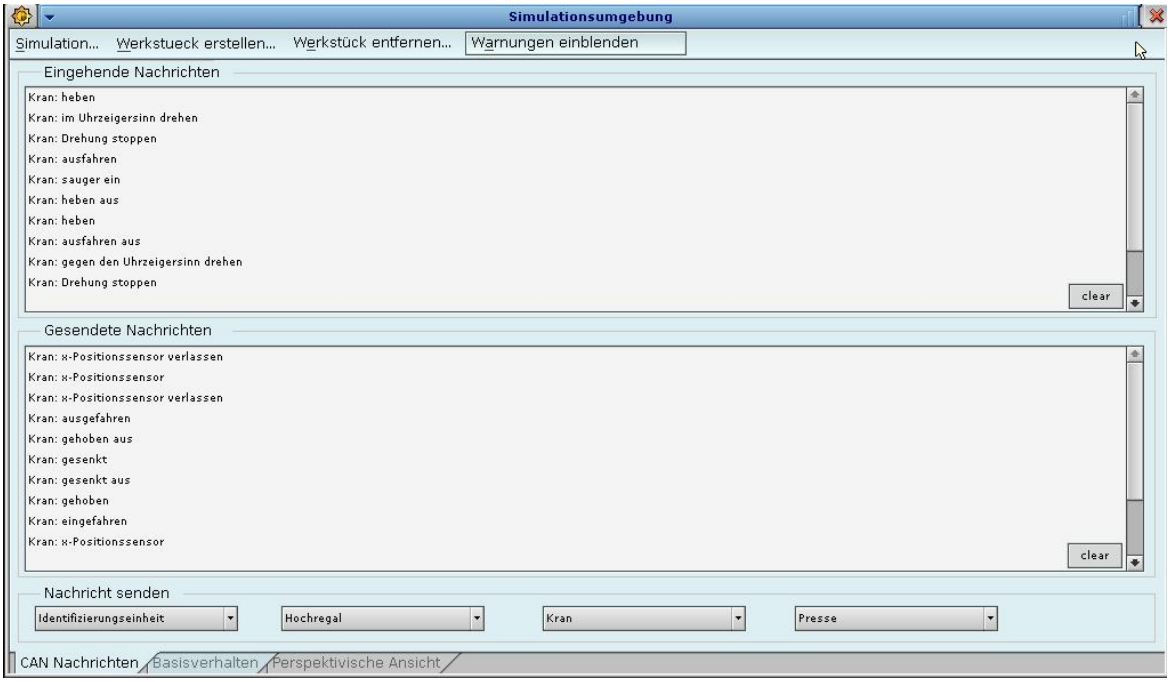


Abbildung A.2: Screenshot: Nachrichten Tab

Hier werden alle eingehenden und ausgehenden Nachrichten in zwei getrennten Textfeldern angezeigt. Im unteren Bereich können für jeden der vier Anlagenteile (RFZ, Kran, Presse, Identifizierungseinheit) alle Sensorereignisse aus dem jeweiligen drop-down Menü ausgewählt und an die Steuerungssoftware gesendet werden. Diese Nachrichten erscheinen dann auch im Textfeld *Gesendete Nachrichten*. (Abb. A.2)

A.3 Tab: Basisverhalten

In diesem Tab kann der Aktuelle Zustand der Sensoren und Aktuatoren der ganzen Anlage nachvollzogen werden. Die Oberfläche ist in 4 Bereiche aufgeteilt. Diese vier Bereiche sind ihrerseits nochmals in Sensoren und Aktuatoren gruppiert. Auf der linken Seite kann jeweils der Zustand der Aktuatoren, auf der Rechten der Zustand der Sensoren eingesehen werden. Die verschiedenen Farben haben folgende Bedeutung:

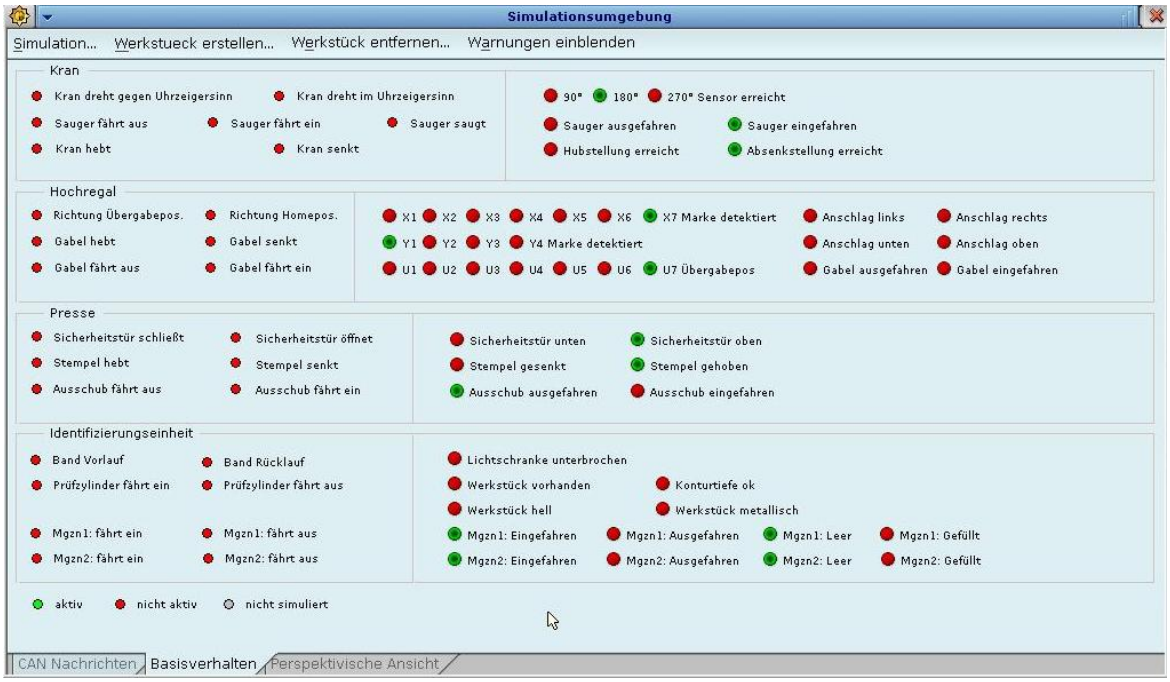


Abbildung A.3: Screenshot: Basisverhalten

Grün Der Sensor bzw. der Aktuator ist gerade aktiv.

Rot Der Sensor bzw. Aktuator ist inaktiv.

Grau Der Sensor bzw. Aktuator wird zur Zeit nicht simuliert.

Die Sensoren können per Hand modifiziert werden. Dazu muss der Entsprechende Switch, der diesen Sensor repräsentiert, angeklickt werden. Der Sensor und evenuell davon betroffene andere Sensoren verändern darauf hin ihren Zustand und die entsprechenden Sensorereignisse werden gesendet, und im Tab *CAN Nachrichten* aufgeführt. Wird zum Beispiel der Sensor für den Endschalter links am RFZ aktiviert, so werden, soweit diese vorher aktiv waren, die x Positions- und Übergabepositionssensoren des RFZ, sowie der Endschalter rechts deaktiviert. (Abb. A.3)

A.4 Perspektivische Ansicht

In dieser Sektion wird die Anlage selbst visuell dargestellt. Es werden Bewegungen, Sensoren, Kollisionen und Werkstücke angezeigt.
Die Bewegungen werden durch die Bewegung der entsprechenden Anlagenteile dargestellt.

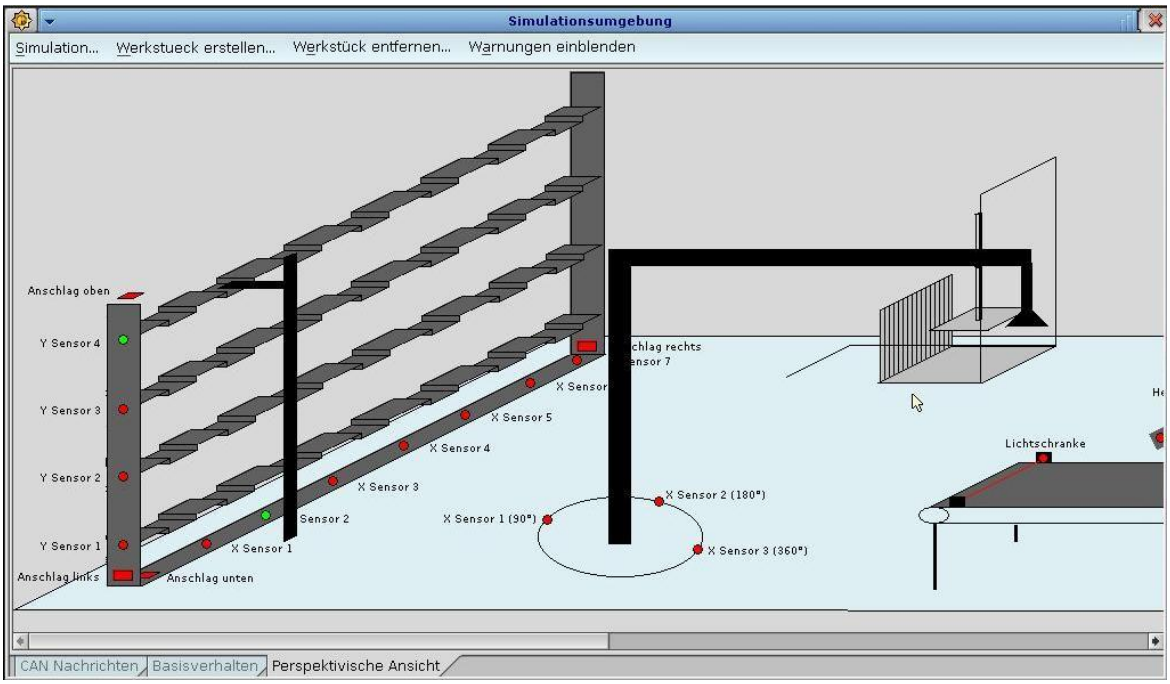


Abbildung A.4: Screenshot: perspektivische Ansicht RFZ, Kran und Presse

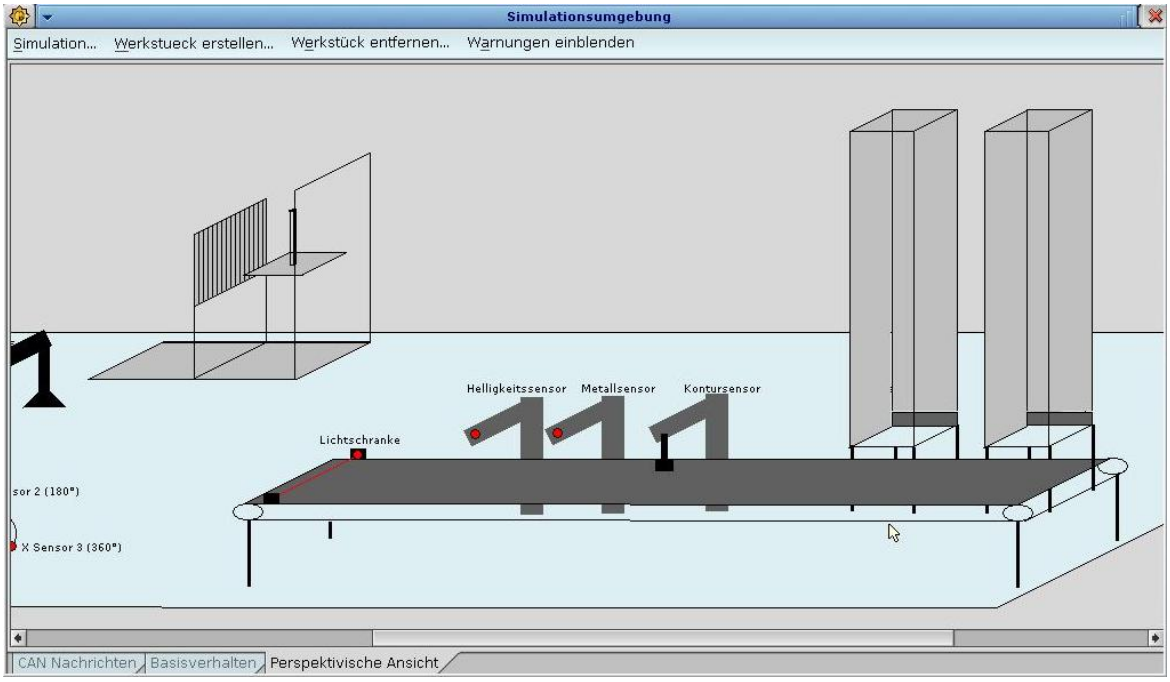


Abbildung A.5: Screenshot: perspektivische Ansicht IE und Presse

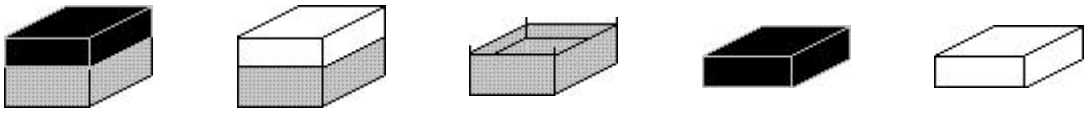


Abbildung A.6: Darstellung der Werkstücke

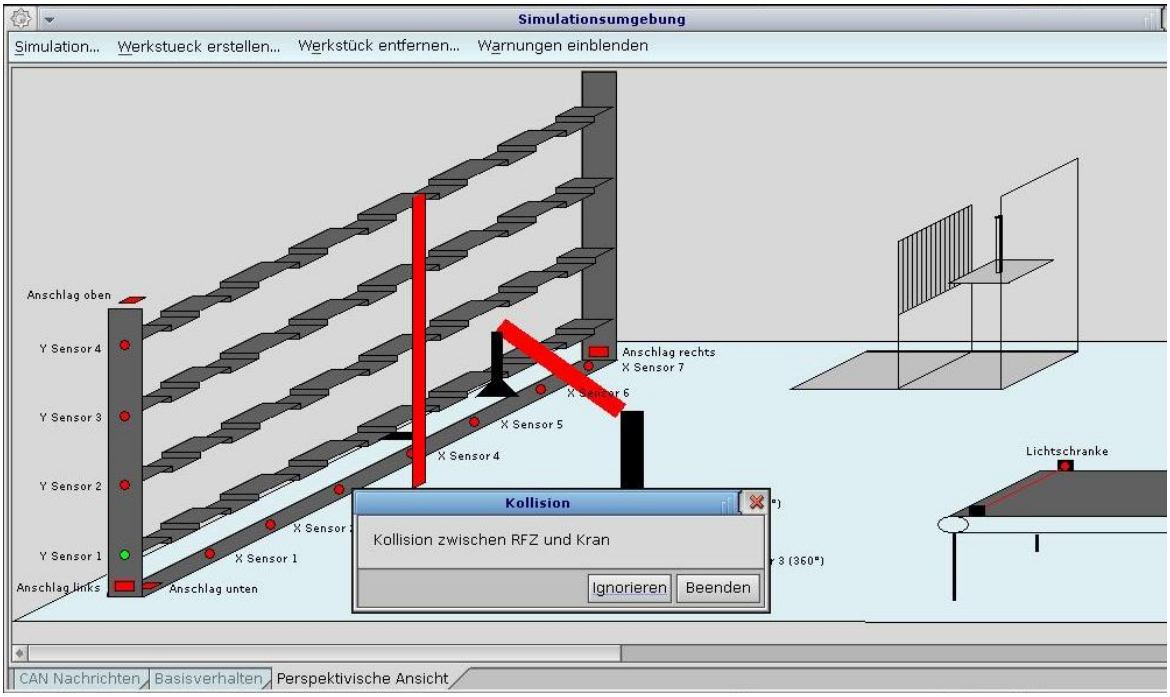


Abbildung A.7: Screenshot: Kollision

Die Sensoren werden wieder durch Lampen repräsentiert.(Abb. A.4, Abb. A.5)
Die fünf verschiedenen Arten von Werkstücken werden wie folgt dargestellt:

Kollisionen werden durch das rot Einfärben der Betroffenen Teile, sowie durch eine Dialog dargestellt. In diesem Dialog wird zur Auswahl angeboten, ob die Simulation beendet werden soll, oder ob der Fehler ignoriert werden soll. Wird ignorieren ausgewählt, so wird die Anlage ein einen Zustand versetzt, der es ermöglicht mit der Simulation weiter zu arbeiten. Da die Position der betroffenen Teile nicht verändert wird, ist es sehr wahrscheinlich dass die Kollision erneut auftritt, in diesem Fall muss eventuell über den Tab *Basisverhalten* manuell der Zustand geändert werden, oder die Simulation ganz neu gestartet werden. (Abb. A.7)

A.5 Menüleiste

A.5.1 Simulation...

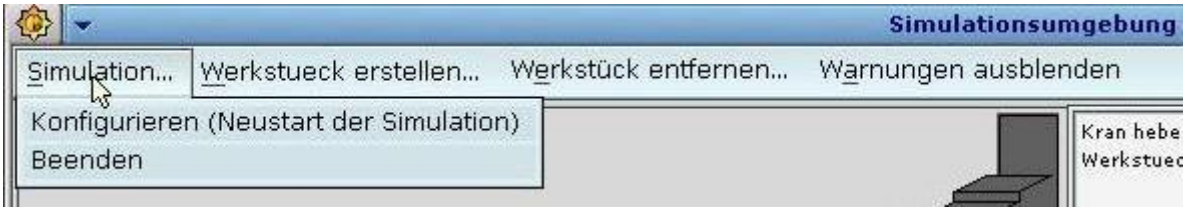


Abbildung A.8: Screenshot: Menue Simulation...

Über dieses Menü kann der Konfigurationsbildschirm aufgerufen werden, in dem die Startwerte der Simulation festgelegt werden. Eine Benutzung der Konfiguration hat immer einen Reset und einen Neustart der Simulation zur Folge.
Weiterhin kann über dieses Menü die Anwendung beendet werden.

A.5.1.1 Werkstueck erstellen...

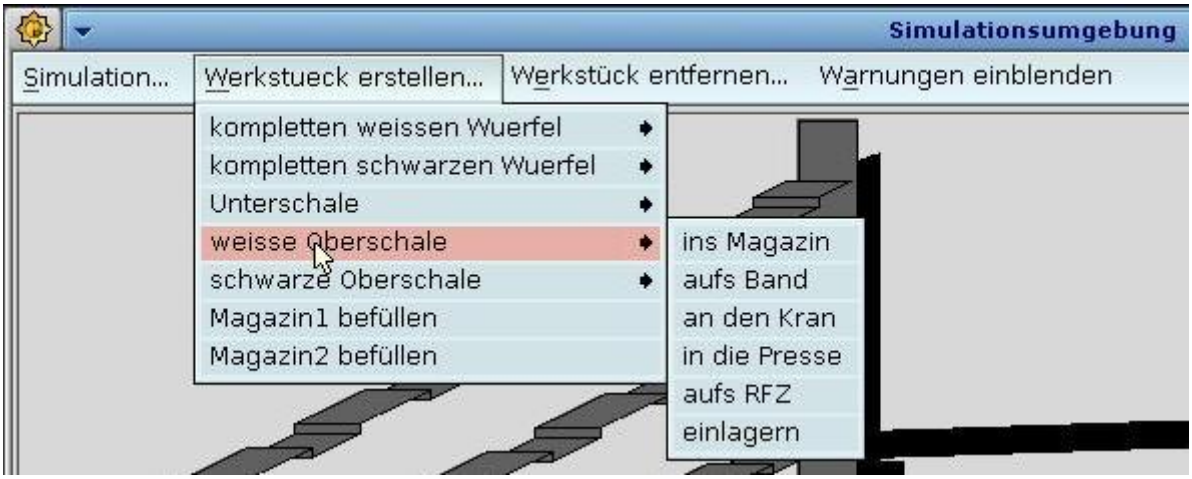


Abbildung A.9: Screenshot: Menue Werstueck erstellen...

Über dieses Menü können Werkstücke an beliebigen Positionen der Anlage erstellt werden. Wird ein Werkstück auf dem Förderband erstellt, so erscheint es immer am Ende des Bandes, wo sich die Lichtschranke befindet. In den beiden Magazinen können nur die jeweils dafür vorgesehenen Werkstücke erstellt werden. Auf dem Förderband können nur Unterschalen und Oberschalen erzeugt werden.

Wird ein Werkstück eingelagert im Regal erstellt, so erscheint ein Dialogfeld, in welchem ausgewählt werden muss, an welcher Stelle genau das Werkstück positioniert werden soll. Dabei werden die Regalplätze von unten links in x und y Richtung gezählt.

A.5.1.2 Werkstück entfernen...

Abbildung A.10: Screenshot: Menue Werkstueck entfernen...

Hierdurch können Werkstücke aus der Simulation entfernt werden. Anzumerken ist, dass wenn ein eingelagertes Werkstück entfernt werden soll, immer das zuerst eingelagerte genommen wird. (Abb A.10)

A.5.1.3 Warnungen einblenden

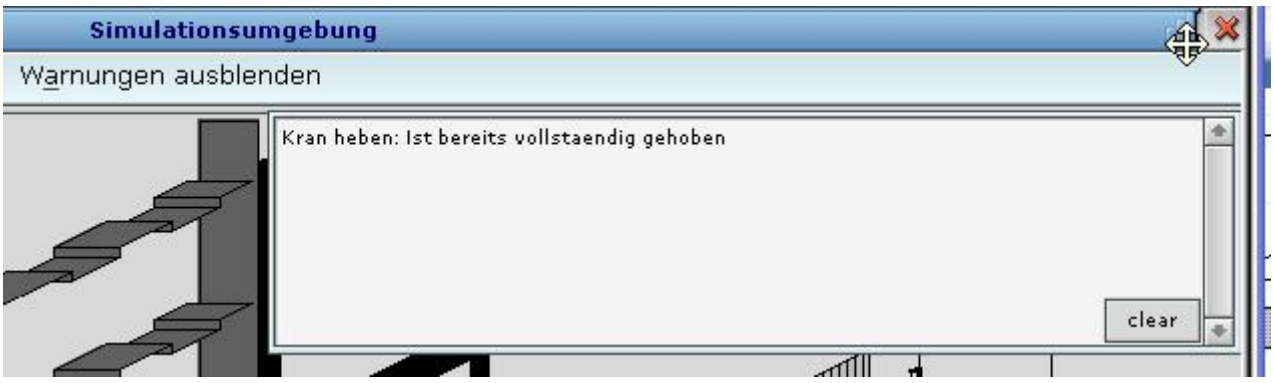


Abbildung A.11: Screenshot: Warnungen

Hiermit kann das Fenster, welches aufgetretene Warnungen anzeigt, ein- und ausgeblendet werden. Dieses Fenster wird automatisch eingeblendet, wenn eine neue Warnung auftritt. (Abb. A.11)

Literaturverzeichnis

[1] BOSSEL, Hartmut: *Modellbildung und Simulation*. 2. Auflage. Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1994. – ISBN 3–528–15242–7

[2] HAHN, Otto: *MaschinenMarkt: Simulation macht Komplexität von Produktionsanlagen durchschaubar*. Fachartikel, 2007. – http://www.maschinenmarkt.vogel.de/themenkanaele/digitalefabrik/cadcam/articles/99002/; 07.12.2007.

[3] LUTZ PRIESE, Harro W.: *Theoretische Informatik Petri-Netze*. 1. Auflage. Springer, 2003

[4] NECULAU, Ulrich Kramer & M.: *Simulationstechnik*. Carl Hanser Verlag Münschen Wien, 1998. – ISBN 3–446–19235–2

[5] RAINER KÖNIG & DR.-ING. LOTHAR QUÄCK, Dr. rer. n.: *Petri-Netze in der Steuerungs- und Digitaltechnik*. R. Oldenburg Verlag, 1988. – ISBN 3–486–20735–0

[6] ROTTJAKOB, Hermann: *Simulation und Animation*. 1. Auflage. Weka, 2004. – ISBN 978–3811125896

[7] SCHÖNERT, Ulf: *Die Zeit: Fabrik im Computer*. Artikel, 2001. – http://www.zeit.de/2001/08/200108_z-ing._fabrik.xml?page=all; 07.12.2007.

[8] SYSTEMS, QNX S.: *Pt–Widget Toolkit*. Website, . – http://www.qnx.com/developers/docs/6.3.2/photon/lib_ref/pt-base.html; besucht im November 2007.

[9] SYSTEMS, QNX S.: *QNX Neutrino – Library Reference*. PDF, . – http://www.qnx.com/download/download/11960/nto_lib_ref.pdf; besucht im November 2007.

[10] SYSTEMS, QNX S.: *QNX Neutrino – Programmer's Guide*. PDF, . – http://www.qnx.com/download/download/11961/nto_prog.pdf; besucht im November 2007.

[11] WIKIPEDIA: *Modellbildung - Wikipedia*. Website, . – http://de.wikipedia.org/wiki/Modellbildung; besucht am 10.01.2008.

[12] WIKIPEDIA: *Simulation - Wikipedia*. Website, . – http://de.wikipedia.org/wiki/Simulation; besucht am 08.01.2008.

[13] ZIMMERMANN, Diplom Informatiker A.: *Modellierung und Bewertung von Fertigungssystemen mit Petri-Netzen*. Artikel, 1997. – http://pdv.cs.tu-berlin.de/~zimmermann/texte/phdthesis.pdf; besucht am 31.01.2008.

Abbildungsverzeichnis

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 7. Februar 2008 Jens Windmüller