# Objektorientierte Modellierung zur Simulation des Steuerverhaltens von modularen Transfersystemen

**Diplomarbeit von Kai Schäfer**

Universität Bremen
Fachbereich: Produktionstechnik
Fachgebiet: Fertigungseinrichtungen
Betreuer: Prof. Dr.-Ing. Andreas Visser
Dipl.-Ing. Jürgen Sahlberg

Deckblatt gem. § 15 (7) DPO

# Diplomarbeit

## Thema:

Objektorientierte Modellierung zur Simulation des Steuerverhaltens von modularen Transfersystemen

## Beschreibung:

Bei der Planung und beim Betrieb moderner flexibler Fertigungsanlagen wird die Simulation heute zunehmend als entscheidendes Hilfsmittel verwendet, um die Planungszeiten drastisch zu verkürzen, die Planungssicherheit zu erhöhen und die Strukturen und Abläufe zu optimieren. Dabei gibt es stark spezialisierte wie auch universell einsetzbare Simulationssysteme. In dem laufenden Forschungsvorhaben Objektorientierte Modellierung zur Simulation und Programmierung flexibler Fertigungsanlagen sollen objektorientierte Modellstrukturen entwikkelt werden, die eine spätere Erstellung von ablauffähigem Steuerungscode ermöglichen.

Im Rahmen dieser Arbeit soll eine generische Objektstruktur für die Simulation von modularen Montagesystemen theoretisch entwickelt werden, die speziell den Aspekt der Steuerungsimplementation mit berücksichtigt.

**Verfasser:** Cand.-Ing. Kai Schäfer
geb.: 7.9.1965
Matr.-Nr.: 66 30 81

**Betreuer, 1. Gutachter:** Prof. Dr.-Ing. Andreas Visser
Fachgebiet Fertigungseinrichtungen
Universität Bremen

**2. Gutachter:** Dipl.-Ing. Jürgen Sahlberg

**Ausgabedatum:** 5.7.1995

**Abgabe:** 27.9.1995

## Erklärung des Kandidaten:

Diese Arbeit wurde vom Verfasser selbständig verfaßt, und es wurden keine anderen als die angegebenen Quellen und Hilfsmittel benutzt.

# Vorwort

Die vorliegende Arbeit bildet den ersehnten Abschluß meines Studiums der Produktionstechnik in Bremen. Meiner Freundin Andrea Zöls und meinem Freund Lothar Filipczak, die durch ihre Anregungen und Korrekturen dazu beigetragen haben, die Arbeit in der vorliegenden Form erscheinen zu lassen, bin ich zu großem Dank verpflichtet.

Nicht unerwähnt bleiben darf die Hilfe von Jürgen Sahlberg, der mir mit seiner Kompetenz und Hilfsbereitschaft in fachlichen und technischen Fragen eine große Hilfe war und Prof. Dr. Visser, der die Arbeit in seinem Fachgebiet ermöglicht hat.

Die Arbeit ist all jenen gewidmet, die sie aus fachlichem oder persönlichem Interesse mit Freude lesen.

*Kai Schäfer*

# Inhaltsverzeichnis

# 1 Einleitung

Die vorliegende Arbeit verknüpft die Bereiche *Materialfluß-Simulation* und *SPS-Programmierung*. Die Ergebnisse werden in einem Softwarekonzept, das den Arbeitstitel Objektorientierte Simulation von Prozeß und Steuerung (OSIMPROST) erhält, zusammengeführt.

In der modernen Anlagenplanung, bei der die Materialfluß-Simulation zunehmend eine Rolle spielt, bietet die Kombination der beiden Bereiche Materialfluß-Simulation und SPS-Programmierung Vorteile. Die Steuerungslogik, die in der Simulation den Materialfluß steuert, kann als Grundlage für die Erzeugung von SPS-Programmen für die spätere Anlage dienen. Um die SPS-Programmierung effizient zu entlasten - die Programme also möglichst automatisch zu erzeugen und auf die Steuerung zu übertragen - sind neue Konzepte nötig. Während für die getrennte Simulation und Steuerung bereits eine Reihe erprobter Softwareprodukte und Ansätze zur Verfügung stehen, fehlen diese für den kombinierten Einsatz. Es werden Konzepte vorgestellt und erarbeitet, mit deren Hilfe die simulationsgestützte Projektierung von Transfersystemen schneller und wirtschaftlicher als mit getrennter Software durchzuführen ist. Diese Arbeit stellt eine Arbeitsgrundlage für die Entwicklung eines Computerprogramms für die Anlagenplanung dar. Es wird der Versuch gemacht, die dabei auftretenden Probleme zu lösen, und so die Voraussetzungen für die Realisierung zu schaffen.

## 1.1 Motivation

Die SPS-Technik hat sich historisch aus der Anlagensteuerung mit Relais (Schütze), den sogenannten *Verbindungsprogrammierten Steuerungen*, entwickelt. Nach einem Verdrahtungsplan wurden die Schütze in den Schaltschränken miteinander, und mit den Schaltern (Sensoren) und Aktoren der Maschinen, verbunden. Diese aufwendige und inflexible feste Verdrahtung wurde durch prozessorgesteuerte Einrichtungen, bei denen sie Steuerungslogik in Form von digitalen Programmen im Speicher vorliegt, ersetzt. Diese digitalen Steuerungen heißen Speicher-Programmierbare Steuerungen (SPS). Bei der Entwicklung von Programmiersprachen lehnte man sich an die gewohnten Verdrahtungspläne an, und schuf z.B. den bis heute gängigen *Kontaktplan*. Um mit einem Programmiergerät mit Tastatur, Bildschirm, Datenträgern usw. mehrere Steuerungen programmieren zu können, wurde die Offline-Programmierung eingeführt. Die auf einem separaten Programmiersystem erstellten Programme werden über Datenträger oder Kabel auf die Steuerungen übertragen.

Die Computersimulation hat sich, weitgehend getrennt von der SPS-Programmierung, auf der Basis von höheren Programmiersprachen wie z.B. FORTRAN entwickelt. Um die Tätigkeiten der Simulationsprogrammierung zu rationalisieren, wurden Bausteinbibliotheken zu den Programmiersprachen hinzugefügt wie z.B. GPSS-FORTRAN. Mit der Einführung der Animation konnte der Ablauf der Simulation auf dem Monitor verfolgt werden. Eine weitgehend grafisch-

interaktive Modellerstellung mit objektorientierten Konzepten kennzeichnet den heutigen Stand der Simulation.

Durch die getrennte Entwicklung von SPS-Programmierung und Simulation, sind die Strukturen beider Bereiche sehr weit voneinander entfernt, und lassen ihre Zusammenführung als schwierig erscheinen. Diese Arbeit möchte eine Brücke zwischen den Bereichen schlagen.

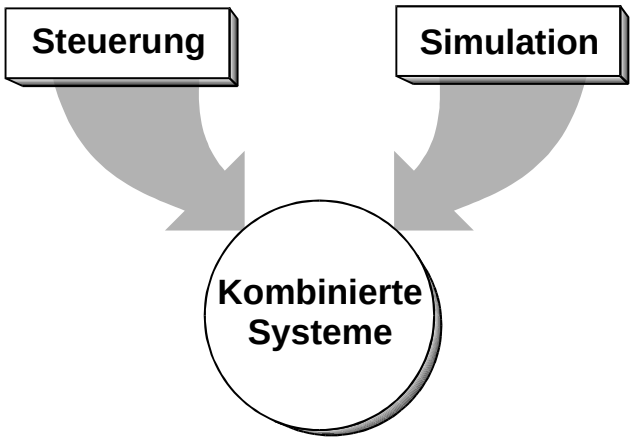


Abbildung 1: Ziel der Arbeit

Der normale Gang beim Projektieren von Anlagen mit SPS ist nach Friedrich Janzen folgender /JANZEN 90/:

1. Konstruktion,
2. Erstellen der Zeichnungen und Unterlagen
3. Ermittlung der Steuerungslogik,
4. Steuerungsauswahl,
5. Bau,
6. Aufstellung,
7. Programmierung,
8. Inbetriebnahme.

Bei dieser Methode ist die Integration einer Simulationsphase schwierig, weil sich die Planungsphase durch das Einschieben von Simulationszeiten verlängert. Aus Zeitgründen kann nur in einer kurzen Zeitspanne mit dem Modell experimentiert werden.

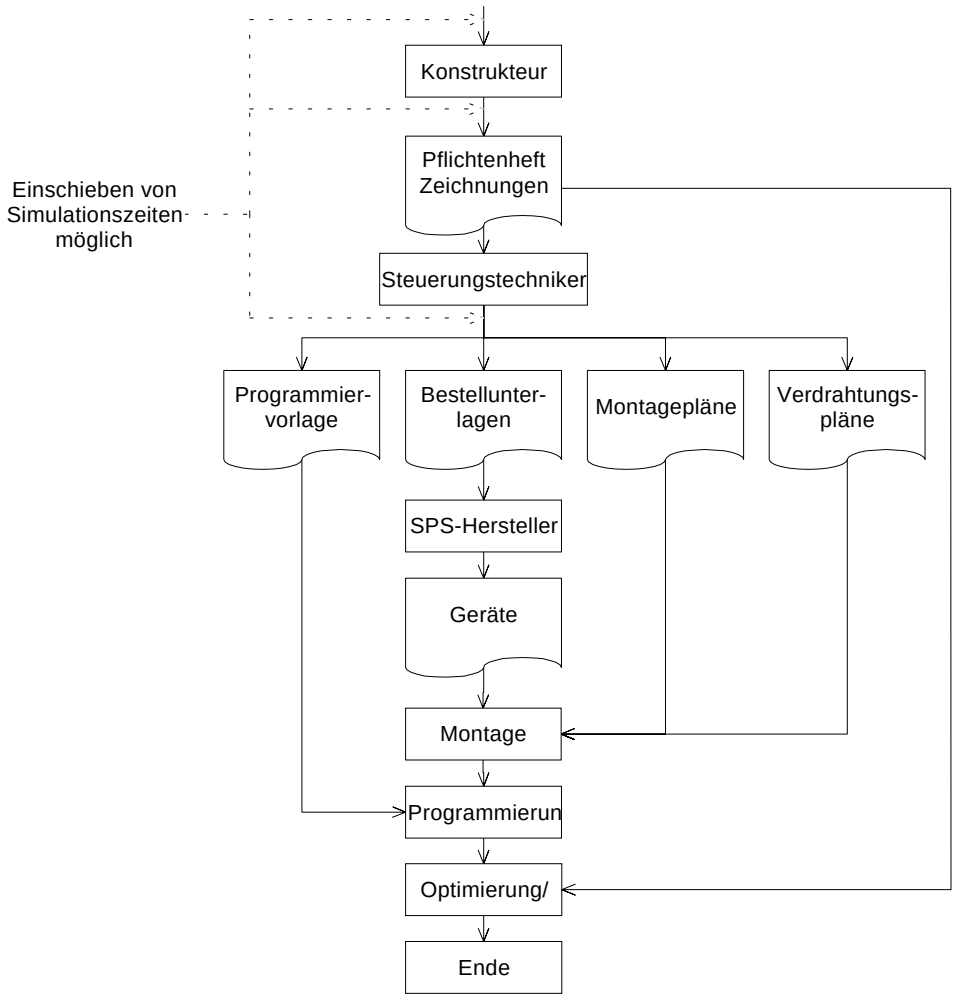


Abbildung 2: Projektierungsabschnitte

Für den Fall, daß ein kombiniertes Planungs-, Simulations- und Programmiersystem zur Verfügung steht, ist eine neue Vorgehensweise für die Anlagenprojektierung möglich. Der neue Ablauf gliedert sich in die Phasen:

1. Anlagenkonzeption und Simulation,
2. Bau,
3. Inbetriebnahme.

Zusätzlich zu den Vorteilen der Simulation[1] kann die Anzahl der Planungsschritte, und damit die Zeit und die Kosten, die diese in Anspruch nehmen, entscheidend reduziert werden. Zusätzlich können *einsatzfertige* und *erprobte* SPS-Programme erzeugt und eingesetzt werden. Abbildung 3 zeigt den Planungs- und Aufstellungsablauf unter Einsatz der kombinierten Software.

[1] Als wesentlich sind die erhöhte Planungssicherheit und eine Anlagenoptimierung in der Planungsphase zu nennen. Auch im Bereich der Auftragssteuerung kann die Simulation hilfreich sein. Nach einer Erhebung des VDI beträgt die Kostenersparnis durch Simulation 20%. Dieser Wert weist aber je nach Einsatzfall *extreme* Schwankungen auf.

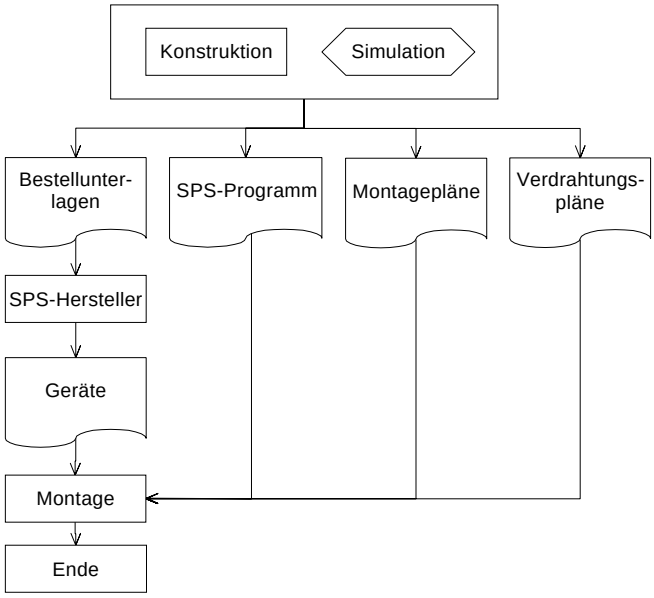


Abbildung 3: Projektierung mit Simulation

Ein mit OSIMPROST vergleichbares kombiniertes Programm ist bisher nicht auf dem Markt verbreitet /ISIS 94//SCHNEIDER 95/. Eine Reihe von Grundlagen, die zu dessen Entwicklung notwendig sind, liegen jedoch schon vor.

Da das Interesse in der Wirtschaft durch die möglichen großen Einsparungen groß ist, und die Zusammenführung von Simulation und SPS ein interessanter Forschungsgegenstand ist, wurde die vorliegende Diplomarbeit geschrieben.

## 1.2 Aufbau der Arbeit

Da bisher kaum Untersuchungen zum gestellten Thema existieren, müssen für die in folgenden Bereichen auftretenden Fragestellungen Antworten und Lösungen gesucht werden.

- Bei der Materialfluß-Simulation ist für eine Darstellung, die jedem Sensor und Aktor der Anlage abbildet, ein sehr hoher Detaillierungsgrad erforderlich. Hieraus ergeben sich Probleme im Speicherbedarf und in der Simulationsgeschwindigkeit, welche gelöst werden müssen.
- In der Materialfluß-Simulation hat sich die *Ereigniszeitsteuerung*[2] am besten bewährt. SPS arbeiten ihre Programme zyklisch ab, um in *Echtzeit*[3] auf Systemveränderungen reagieren zu können. In der Simulation führt die Kombination beider Zeitdarstellungen zu einer Verringerung der Geschwindigkeit, weil der Prozessor des Simulationsrechners durch die vielen SPS-Programmzyklen stark belastet wird.

[2] Die Uhr des Simulators *springt* zum jeweils nächsten Ereignis. Eine genaue Beschreibung der Ereignissteuerung findet sich in Kapitel 3.2.3

[3] Der Begriff Echtzeit besagt, daß eine Steuerung sofort - oder zumindest innerhalb einer vorgegebenen Zeit - auf Eingangssignale reagiert.

- Zum projektbegleitenden Einsatz muß die Simulation der zunehmenden Präzisierung und Verfeinerung des Erkenntnisstandes folgen. Aus der Simulation sind hierfür die Techniken Prototyping und Hierarchisierung/Modelrefinement bekannt[4].
- Transfersysteme[5] werden aus Modulen wie beispielsweise Strecken und Verzweigungen zusammengesetzt. Da diese immer wiederkehrende Grundelemente darstellen, werden sie von den meisten Simulatoren als *Bausteine* angeboten, die in ein Modell eingesetzt werden können. Zu diesen Bausteinen von Transfersystemen gehören Steuerungsabläufe, die in der realen Anlage von einer zugeordneten SPS übernommen werden. Diese Zusammengehörigkeit von Bausteinen und Steuerprogrammabschnitten soll in der kombinierten Simulation ausgenutzt werden, um den kompletten Steuerungscode möglichst automatisch zu erzeugen. Für ihre kombinierte Darstellung ist ein Konzept zu entwickeln.
- Für die übergeordneten Aufgaben der Materialflußsteuerung sind SPS-Programme nötig, die die Abläufe der einzelnen Bausteine der Anlage koordinieren. Diese müssen vom Programmierer aufgabenbezogen erstellt werden. Hieraus leitet sich die Forderung an das Simulationssystem ab, daß es die erforderliche Steuerungslogik abbilden können muß. Auch hier besteht die Möglichkeit, durch SPS-Programm-Makros oder Unterprogrammtechnik die Arbeit zu erleichtern.
- Das System steht Simulationsexperten ebenso wie Anlagenplanern und SPS-Programmierern als Werkzeug zur Verfügung. Für die gemeinsame Nutzung ist eine allgemeinverständliche Benutzeroberfläche der Software wünschenswert.

Der Lösung der oben aufgezählten Probleme widmet sich diese Diplomarbeit.

Eine Reihe von Methoden und Konzepten, die für die gestellte Aufgabe in Frage kommen, findet sich bereits in der Literatur. Daher werden als Arbeitsgrundlage die Bereiche **2 Objektorientierte Programmierung** und **3 Simulation** genauer dargestellt.

Es soll sichergestellt werden, daß mit den gewählten Konzepten die Probleme der Praxis sinnvoll abgebildet werden können. Dazu werden die Transfersystem-Bausteine eines Herstellers und die zugehörigen Steuerungsfunktionen beispielhaft beschrieben, und im Kapitel **4 Beschreibung von Bausteinen für Transfersysteme** auf ihre grundsätzliche Struktur hin untersucht und objektbezogen beschrieben.

---

[4] Prototyping bezeichnet die Analyse von Modellen, die auf vorläufigen Ergebnissen beruhen. Bei der Hierarchisierung werden Blöcke, bei denen nur das äußere Verhalten beschrieben wurde, durch eine detailliertere Beschreibung aus *mehreren* Blöcken ersetzt.

[5] Transfersysteme transportieren Material automatisch zu den einzelnen Stationen innerhalb eines Fertigungssystems. Der Transport der Teile kann z.B. an Hakenketten hängend, auf Förderbändern oder auf selbstfahrenden Transportfahrzeugen erfolgen.

Zur Darstellung der Bausteine im Simulationsmodell wird ein objektorientierter Ansatz gewählt. Um diesen zu realisieren, ist die Definition einer Objektklassenhierarchie erforderlich. Diese wird im Abschnitt **4.2 Darstellung der Bausteine als Objekte** vorgenommen. Durch sie ist die Übertragung der Bausteine auf Rechnerebene möglich.

Um die Ziele der Arbeit zu erreichen, ist eine Reihe weiterer Fragen zu klären. Das geschieht im Kapitel **5 Objektorientierte Simulation von Prozeß und SPS**.

Hier werden für das Projekt OSIMPROST u.a. die Bereiche der

- Zeitsteuerung,
- der Handhabung und
- der SPS- Programmübertragung

genauer betrachtet und Ansätze zu ihrer Lösung vorgestellt.

Abgeschlossen wird die Arbeit durch die Beschreibung der Implementation eines Beispiels, das den vorhandenen Simulator SIMPLE++[6] einsetzt. Dieses Programm wurde in der Arbeit begleitend eingesetzt, um durch Testen verschiedener Lösungsansätze, sich ergebende Probleme zu erkennen. Einschränkungen der Möglichkeiten ergeben sich in SIMPLE++ durch die Vorgaben der Software, die teilweise nicht mit den Konzepten von OSIMPROST vereinbar sind. Die Implementierung der wesentlichen Aspekte von OSIMPROST auf SIMPLE++ ist aber schließlich gelungen und im Abschnitt **5.3 Ein einfaches Simulationsbeispiel in SIMPLE++** dargestellt.

[6] Ein Simulator der Firma AESOP Stuttgart, der neben der grafisch-objektorientierten Modellierung, eine Programmiersprache zur Modellmanipulation bereitstellt.

# 2 Einführung in die objektorientierte Programmierung

Die objektorientierte Programmierung ist ein Merkmal der aktuellen Softwareerstellung. Während bei der herkömmlichen Programmierung Daten und Funktionen zur deren Manipulation getrennt vorliegen, sind Funktionen und Daten bei der objektorientierten Programmierung in ihrer funktionalen Zusammengehörigkeit eine Einheit. Diese Einheit wird als *Objekt* aufgefaßt. Ein Objekt ist also ein Programmteil mit zugeordneten Daten, der mit einer *Hülle* umgeben ist. Dieser Hülle ist ein *Name* zugeordnet, über den der Inhalt angesprochen werden kann. Innerhalb dieser Hülle werden die Daten, die den Zustand des Objektes beschreiben, als *Attribute*, die Funktionen und Prozeduren als *Methoden* bezeichnet. Bei einer sauberen objektorientierten Programmierung sollten keine Daten oder Programmteile außerhalb von Objekten eine Rolle spielen.

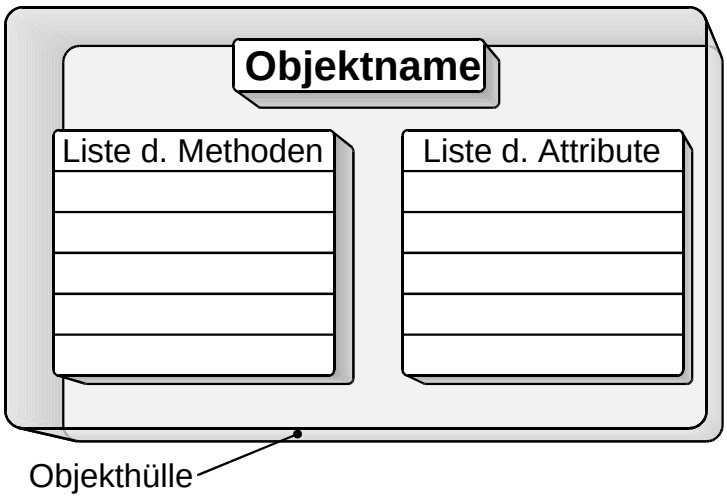


Abbildung 4: Elemente eines Objekts

Die Hülle der Objekte sollte über möglichst wenig Löcher verfügen und somit möglichst wenig von ihrem Inhalt nach Außen preisgeben. Diese Gestaltungsweise wird *Informationhiding* (Kapselung oder Versteckprinzip) genannt, und soll die Beziehungen zwischen den Objekten übersichtlich halten und sie möglichst leicht wiederverwendbar machen. Attribute und Methoden können als *Private* definiert und somit für die Außenwelt versteckt, oder als *Public*, und so für andere Objekte anwendbar und sichtbar gemacht werden[7].

Objekte müssen, bevor sie benutzbar sind, definiert werden. Das geschieht durch die Definition von *Klassen*. Eine Klasse stellt einen allgemeinen Bauplan oder Schablone für die Erzeugung

[7] Die Spezifikation der Zugriffsrechte auf Attribute und Methoden ist von der Programmiersprache abhängig und deshalb uneinheitlich. Es ist die Aufgabe des Programmierers, die Anzahl der Verknüpfungen zur Außenwelt im Sinne einer übersichtlichen objektorientierten Programmierung gering zu halten.

von Objekten dar. In ihnen werden die Attribute und Methoden definiert. Attribute können mit *Defaultwerten* vorbelegt werden.

Um nun ein *Objekt* (*Instanz* oder *Exemplar*) zu erzeugen, das innerhalb des Programms angesprochen werden kann, wird dieses aus der Klasse *abgeleitet*. Dazu erhält es einen Namen, und die Attribute, die nicht durch Defaultwerte belegt sind, werden spezifiziert. Durch Spezifizierung entsteht so aus dem abstrakten ein konkretes Objekt. Ableiten ist ein Vorgang, der äußerlich wie das Kopieren der Klasse aussieht, mit dem Unterschied, daß unter dem Namen der neuen Instanz lediglich ein Verweis auf die Klasse, von der sie abstammt, angelegt wird. Die Instanz auf Programmebene besteht also nur aus einem Namen und einem Verweis auf die Klasse, für den Anwender stellt sie sich aber wie ein real existierendes Objekt dar.

Ein Beispiel soll den Begriff der Klassenhierarchie veranschaulichen /PAGE 91/. Die Basisklasse Fahrzeuge (Abbildung 5) könnte z.B. die Attribute Abmessungen, Gewicht und Position [*X*,*Y*] enthalten. Nur die Luftfahrzeuge haben das zusätzliche Attribut Flughöhe *Z*. Durch immer weitere Spezifizierung von Attributen gelangen wir von einer abstrakten Klasse schließlich zu Klassen, die über einen vollständigen Satz von Eigenschaften verfügen und aus denen funktionsfähige Instanzen abgeleitet werden können.

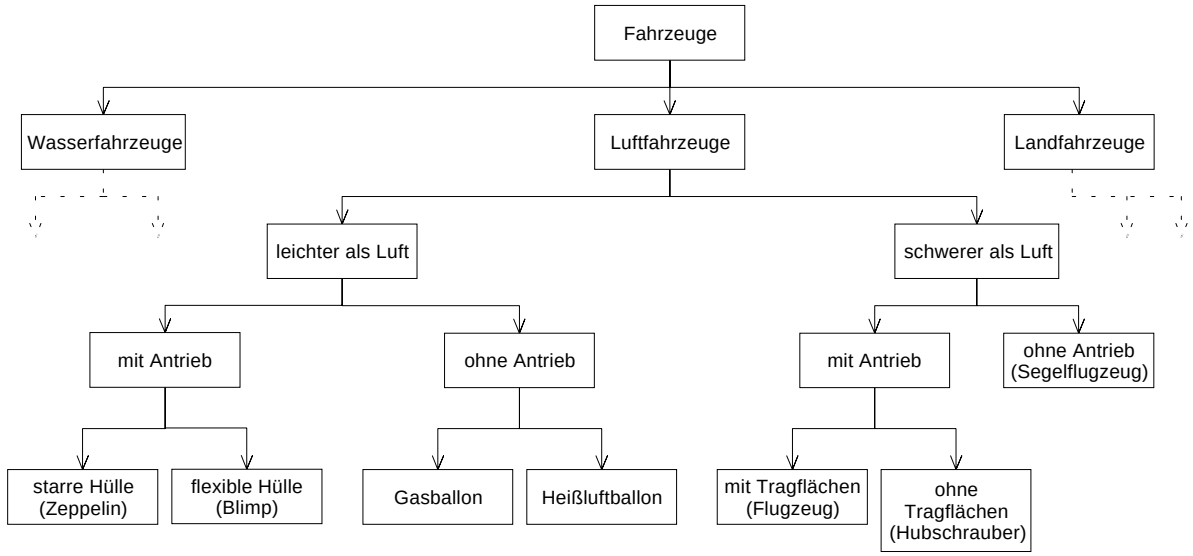


Abbildung 5: Klassenhierarchie von Luftfahrzeugen /BECKER 91/

Wenn bei einer höheren Klasse Änderungen vorgenommen werden, verändern sich auch alle Instanzen, die von ihr *abstammen*, sie *erben* alle Eigenschaften. Das ist nicht immer erwünscht, außerdem sollen Instanzen möglich sein, die von der Klasse und untereinander abweichende Attributzustände haben. In jeder Instanz können deshalb alle Attribute und Methoden überschrieben, verändert oder gelöscht werden. Bei Attributen bezieht sich das auf Name, Datenformat und Inhalt, bei Methoden auf Name und Deklaration. Der Mechanismus der Vererbung wird dadurch bei diesen Elementen ausgeschaltet, die Instanzen enthalten jetzt neben ihrem Namen und dem Verweis auf die Klasse außerdem die individuellen Elemente.

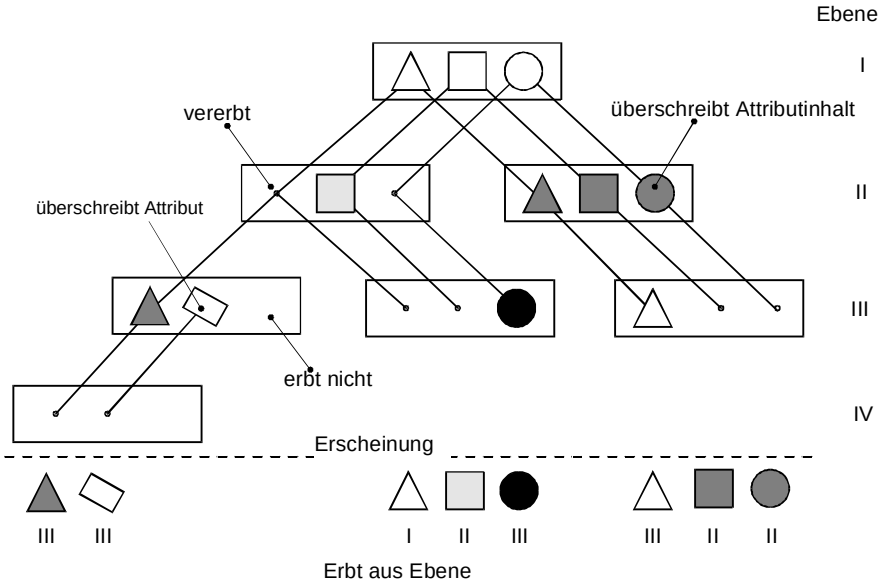


Abbildung 6: Vererbung von Attributen

Bei dem Überschreiben von Methoden gibt es noch eine Besonderheit, die von einigen Programmiersprachen wie z.B. C++ unterstützt wird. Eine Methode kann mit gleichem Namen mehrfach definiert werden und sich nur durch die Deklaration ihrer Eingabeparameter (Zahl und Typ der Argumente) unterscheiden. Wird eine solche Methode angesprochen, kommt automatisch die Methode mit einem passenden Eingabeparametersatz zur Anwendung. Diese Möglichkeit heißt *Überladung* oder *Polymorphismus*. Durch sie können Objekte flexibler gestaltet werden, weil sie unterschiedliche Parametertypen akzeptieren können.

Die so erzeugten Instanzen können nicht nur im Programm verwendet werden, sondern auch als Bauplan für weitere Objekte dienen und somit selbst wieder Klassen sein. In dieser Weise erhalten Programme eine baumartige *Klassenhierarchie*, jede Instanz läßt sich auf eine *Basisklasse* zurückführen. Klassen können neben eigenen Attributen und Methoden auch aus anderen Objekten bestehen. Mit dieser Möglichkeit können Objekte auch aus hierarchisch zusammengeführten Klassen bestehen (*Modellhierarchie*).

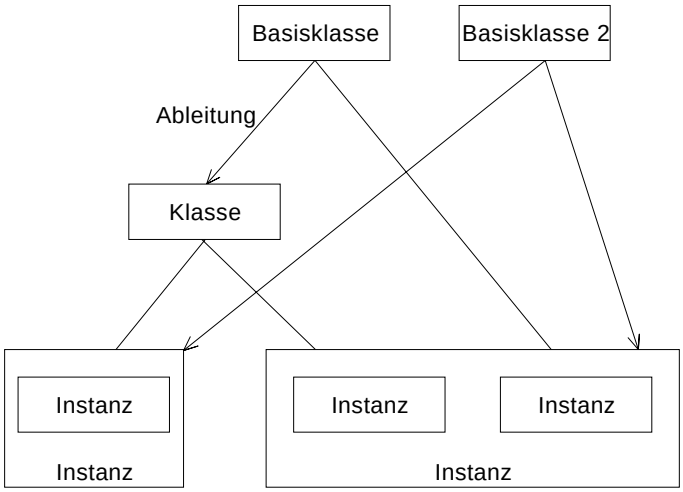


Abbildung 7: Klassenhierarchie und Modellhierarchie

Die erzeugten Objekte liegen nun inaktiv nebeneinander. Um das Programm zu starten, müssen die Methoden der Objekte auch angesprochen werden. Nach einem initialisierenden Aufruf beim Programmstart können die Objekte einander durch Nachrichten (*Messages*) ansprechen. Dieses *Messagepassing* geschieht über das Aufrufen der als Public definierten Methoden des jeweils anderen Objekts. Die aufgerufenen Methoden können

- den Zustand aller Attribute innerhalb des Objekts lesen und manipulieren,
- Public-Attribute anderer Objekte lesen und manipulieren,
- Public-Methoden anderer Objekte aufrufen und
- interne Methoden aufrufen.

Die Regelung der Zugriffsrechte ist nicht in allen objektorientierten Sprachen und Systemen einheitlich. In C++ beispielsweise lassen sich Attribute und Methoden auch als *protected* definieren. Diese können dann nur von Objekten, die in der Klassenhierarchie aus dieser Klasse abgeleitet sind, angesprochen werden. Die Möglichkeit, Elemente durch die Klassifizierung als Private zu verstecken, besteht bei allen objektorientierten Programmiersprachen, da sie für das *Objecthiding* erforderlich ist.

Durch eine objektorientierte Programmierung ergeben sich eine Reihe von Vorteilen. Durch die Vererbung sind Aktualisierungen, die sich auf mehrere Objekte beziehen, leicht durchzuführen, sie müssen nur einmal auf einer höheren Ebene[8] vorgenommen werden, die abgeleiteten Objekte und Klassen übernehmen die Änderungen automatisch. Dadurch, daß die Informationen nicht in jedem Programmelement redundant gespeichert sind, kann sich außerdem eine erhebliche Datenreduktion ergeben. Die Vermeidung von Redundanz ist ein wichtiger Grundsatz, um große Systeme wartungsgerecht zu halten. Durch die Verknüpfung von Daten und Methoden

[8] Innerhalb einer Klassenhierarchie werden die Basisklassen, die abstrakter sind und von denen abgeleitet wird, als *höher* bezeichnet.

innerhalb der Objekte, bleiben die Auswirkungen von Methodenänderungen immer überschaubar, da sich die Maßnahmen durch das Informationhiding nicht auf andere Programmbereiche auswirken. Durch die Beschränkung der Interaktionen zwischen Objekten auf die Public-Methoden bleibt auch der Gesamtzusammenhang des Programms übersichtlich. Der Programmierer wird durch das Objektkonzept bei der Softwareerstellung, der Softwareanwender beim Programmumgang in seinem Denken unterstützt. Das Konzept, das auf Gegenständen, Funktionen, die diese ausüben können und Beziehungen, in denen diese zueinander stehen aufbaut, entspricht der menschlichen Auffassung besser, als gewöhnliche Programme. Nach G. Färber ist der Objektansatz die Lösung der seit den 70er Jahren bestehenden Softwarekrise /HEUMANN 93/.

Objektorientierte Programmierung hat zwei wesentliche Aspekte. Der Eine ist die Erstellung objektorientierter Software. Das bezieht sich auf die Oberfläche, die dem Anwender die Beschreibung seiner Probleme in Objekten ermöglicht. Dieser Ansatz ist unabhängig von der eingesetzten Programmiermethode, die nicht notwendigerweise objektorientiert sein muß. Der zweite Aspekt bezieht sich auf die verwendete Programmiersprache. Die klassische Sprache für objektorientierte Programmierung ist SMALLTALK. Die meisten modernen Programmierhochsprachen bieten aber heute auch die nötigen Erweiterungen für objektorientierte Programmierung an, beispielsweise BORLAND-PASCAL ab Version 6.0, C in der Variante C++. Obwohl der Einsatz objektorientierter Sprachen nicht zwingend für die Erstellung objektorientierter Software nötig ist, wird die Programmierung durch die angebotenen speziellen Sprachelemente doch erheblich besser unterstützt und damit vereinfacht /FISHWICK 95/.

Der objektorientierte Beschreibungsansatz kann nicht nur bei der Computerprogrammierung eingesetzt werden, sondern auch für die *Beschreibung realer Objekte*. Von diesem Umstand wird im nächsten Kapitel Gebrauch gemacht, indem die Bausteine von Transfersystemen und ihre Funktionalität objektorientiert beschrieben werden. In dieser Arbeit ist die Simulation ein zentraler Gegenstand der objektorientierten Programmierung.

# 3 Simulation

Simulation ist überall dort einzusetzen, wo sich die Problematik aufgrund ihrer Beschaffenheit einer mathematischen Analyse entzieht, was bei den meisten realen Prozessen der Fall ist. Allgemein läßt sich als ihr Ziel formulieren, Aussagen über Systeme zu treffen, ohne daß diese direkt zur Lösung der Aufgabe herangezogen werden. Die Simulation schafft ein abstraktes dynamisches Abbild der Realität. Dabei werden Zustände und Zusammenhänge soweit abstrahiert, daß sie in dem gewählten Medium, z.B. Planspiel oder Computerprogramm, darstellbar sind. Bei dieser Umsetzung müssen die wesentlichen Verhaltensmerkmale zutreffend erfaßt werden, weil davon die Richtigkeit der erzielten Planungs- und Optimierungsergebnisse abhängt. Der VDI definiert Simulation folgendermaßen /VDI 3633/:

> *„Simulation ist die Nachbildung eines dynamischen Prozesses in einem Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind.“*

Kormanicki setzt den Schwerpunkt bei seiner Definition auf einen anderen Aspekt /HEUMANN 93/:

> *„Simulation ist in erster Linie eine Technik, die eine Modellgenerierung enthält, d.h. das Abbilden des wahren Zustandes erlaubt; danach werden Experimente an diesem Modell durchgeführt.“*

Der Modellierungsaspekt steht bei dieser Definition im Vordergrund und verweist auf die Möglichkeit, durch die *eindeutige* Beschreibung des Modells ein besseres Verständnis der Problematik eines Projektes zu gewinnen.

Deshalb werden Simulationsprogramme schon in vielen Bereichen eingesetzt. Regelkreisauslegung, Elektronikschaltungsentwurf, CNC-Programmierung, Roboter-Programmierung und -Bewegungsanalyse, Materialflußsteuerung und Werkstatt-/Logistikplanung sind nur einige Beispiele dafür. Simulatoren stehen für Experimente aller Art uneingeschränkt zur Verfügung, was sie als Untersuchungswerkzeug dem realen Prozeß überlegen macht.

Das Werkzeug für die Anwendung stellt der *Simulator* dar, worunter eine Menge von zusammenarbeitenden Rechnerprogrammen, sowie die zugehörige Hardware zu verstehen ist. Das Kernstück stellt das eigentliche Simulationsprogramm dar. Es wird durch Hilfsprogramme zur Eingabedatengenerierung, Ergebnisdarstellung, Kopplung mit anderer Software usw. ergänzt. Zusammen werden sie als Softwarepakete von den Herstellern angeboten. Teilweise kann durch Updates und Zukauf das Anwendungsspektrum der Systeme noch erweitert werden. Eine große Einschränkung ist dabei die Herstellergebundenheit der einzelnen Programmteile.

Bestehende Simulatoren sind meistens parametrisch auf sequentiellen Programmiersprachen aufgebaut. Durch eine klare Modularität der aktuellen Programme ist eine hohe Leistungsfähigkeit und Flexibilität erreicht worden. Eine Begrenzung erfahren sie jedoch durch die mangelnde

Fähigkeit, den komplexen, verschieden aufgebauten Zusammenhängen der Realität gerecht zu werden. Hier greifen Konzepte, die zum Beispiel auf der Modellierung in Netzwerken beruhen. Ein anderer Ansatz ist die wissensbasierte Simulation mit Sprachen der künstlichen Intelligenz wie z.B. ROSS, LISP, PROLOG, POP. Der Ursprung dieser Methode liegt in militärischen Planspielen, wurde aber auch für andere Bereiche umgesetzt, zum Beispiel für ein CIM-System /BEN-ARIEH 88/.

Simulatoren lassen sich nach Aufbau und Dateneingabe in verschiedene Gruppen einteilen (Tabelle 1). Neue Ansätze für Simulationsprogramme sollten aufgrund der großen Vorteile *objektorientiert* aufgebaut sein. Um ein Programm über einen längeren Zeitraum zu nutzen, muß es mit betrieblichen und technologischen Entwicklungen schritthalten. Diese Forderung kann nur erfüllt werden, wenn das Programmkonzept Erweiterungsmöglichkeiten und eine hohe Wartungsfreundlichkeit aufweist. Diese wird in der Regel durch einen streng modularen und objektorientierten Aufbau verbessert.

Um eine Simulation durchführen zu können, wird die Realität durch Daten abgebildet. Diese haben eine unterschiedliche Beschaffenheit. Es liegen die Regeln des Systemverhaltens in Form von Strukturen und Methoden vor, welche sich während der Simulationsläufe nicht ändern. Der Zustand des Systems stellt sich in dynamisch veränderlichen Fakten dar. Der dritte Datentyp, der bei der Simulation auftritt, sind statistische Daten, die bei der Simulation aufgenommen werden, um das Systemverhalten auswerten zu können. Tabelle 2 nennt Beispiele zu den einzelnen Gruppierungsmerkmalen.

Aufgrund der unterschiedlichen Komplexität, Beschaffenheit, Änderungshäufigkeit und Verwendung sollte zwischen ihnen sauber unterschieden werden, um das Programm übersichtlich und wartungsfreundlich zu halten /KUK 88/.

| Konzept | Programmiersprache | Bausteinorientiert | Objektorientiert |
|---|---|---|---|
| **Produktkennzeichen** | • Programmiersprache<br>• nicht spezifische Modellierung<br>• keine Animation als Zusatz | • Modellbausteinbibliothek<br>• Standardsteuerungen mit/ohne Anpassung<br>• grafische Modellierung und Animation | • Bausteinkonzept für Materialfluß und Informationsfluß: Basisobjekte, Anwenderobjekte<br>• Steuerungsdefinition ohne Programmierung<br>• grafische, objektorientierte Modellierung und Animation |
| **Anwendungskennzeichen** | • allgemeine Simulationssprachkonstrukte<br>• aufwendige Modellierung<br>• fehlende Transparenz und Akzeptanz<br>• teure, zeitintensive Simulationsstudien | • spezieller Bausteinsatz vom Hersteller<br>• effiziente Modellierung mit Einschränkungen<br>• Standardsteuerungsstrategien abbildbar<br>• begrenzte Informationsverarbeitung<br>• schnelle, eingeschränkte Simulationsstudien | • beliebige Bausteinsätze vom Anwender definierbar<br>• effiziente Modellierung ohne Einschränkungen<br>• freie Steuerungsdefinition ohne Einschränkung<br>• flexible Informationsverarbeitung<br>• schnelle, uneingeschränkte Simulationsstudien |
| **Qualifikation** | • Simulationsexperten | • "Planungsingenieur" | • "Planungsingenieur" |
| **Produktbeispiele** | • GPSS<br>• SLAM II<br>• SIMAN | • DOSIMIS<br>• SIMULAP<br>• WITNESS | • SIMPLE ++<br>• SIMPRO<br>• CREATE |

Tabelle 1: Einteilung von Simulatoren

| Datentyp | Gruppierungsmerkmal | Beispiele |
|---|---|---|
| **statisch** | • Typ und Nummer der Elemente<br>• Position und technische Leistung der permanenten Elemente<br>• fertigungsbezogene Daten der temporären Elemente | • Identifizierungsnummer<br>• durchschnittliche Be- und Entladezeit an einer Maschinenstation<br>• durchschnittliche Be- und Entladezeit an einer Pufferstation<br>• Bearbeitungszeit<br>• durchschnittliche Transportgeschwindigkeit<br>• Transportweglänge |
| **dynamisch** | • Zustand der Elemente<br>• Position der temporären und ortsveränderlichen permanenten Elemente<br>• Zeitpunkt der Ereignisse | • Zustand der Verfügbarkeit<br>• Bearbeitungszustand der Werkstücke<br>• aktuelle Position auf dem Verfahrweg<br>• Zeitpunkt des Bedienendes an einer Station<br>• Zeitpunkt des Ausfallbeginns<br>• Zeitpunkt des Reparaturendes<br>• Simulationszeit |
| **statistisch** | • Arbeitsverhältnis und Ausnutzung der permanenten Elemente<br>• Durchlaufverhältnis der temporären Elemente | • totale Anzahl der beschickten Werkstücke<br>• totale Bedienzeit<br>• totale Ausfalldauer<br>• Auslastung der Anlagenteile |

Tabelle 2: Datentypen nach /KUK 88/

Die in einem System stattfindenden Ereignisse haben ebenfalls eine unterschiedliche Beschaffenheit. Es ist zwischen kausal verknüpften und zufälligen Ereignissen zu differenzieren. Die kausalen Ereignisse stellen bei Computerprogrammen kein Problem dar, weil sie der Art der elektronischen Datenverarbeitung entsprechen. Problematisch sind die in der Realität auftretenden zufälligen Ereignisse. Sie auf Programme zu übertragen erfordert eine genaue Kenntnis ihrer Wahrscheinlichkeit und Verteilungsfunktion, welche jedoch in der Praxis fast nie vorhanden ist. Das liegt einerseits am Einsatz der Simulation in der Planungsphase, in der noch keine Anlagen vorhanden sind, an denen statistische Untersuchungen vorgenommen werden könnten, andererseits am hohen Aufwand der statistischen Datenermittlung. Simulation mit deterministischem Verhalten durchzuführen, ist deshalb nicht als praxisgerecht zu bezeichnen. Um dennoch simulieren zu können, werden stochastisch auftretende oder variierende Ereignisse durch Zufallszahlen, die der realen Verteilung entsprechen sollen, dargestellt. Zufällige Ereignisse in unterschiedlichen Kombinationen können das System auf viele Weisen beeinflussen, ohne daß die Auswirkungen dieser Ereignisse einer Analyse zugänglich wären. Durch mehrere Simulationsläufe mit zufälligen Ereignissen versucht man daher, auf statistischem Wege zu abgesicherten Daten zu gelangen, was natürlich seine Grenzen in der statistischen Theorie und der Anzahl durchführbarer Simulationsläufe findet. Um Verteilungen wirklichkeitsnah beschreiben zu können, sollten Simulatoren verschiedene Generierungsmöglichkeiten für Zufallszahlen

anbieten, wie z.B. Histogrammverteilung, Normalverteilung, Lognormalverteilung, Bernoulli- und Poissonverteilung.

Soll eine Anlage mit Hilfe einer Simulation konzipiert werden, liegen nicht von Anfang an alle Daten detailliert vor. Auch bei Simulationsprojekten ist davon auszugehen, daß anfangs nur grobe, abstrakte, unvollständige und unsichere Daten vorliegen. Trotzdem besteht schon frühzeitig das Interesse, zu Simulationsergebnissen über das Systemverhalten zu gelangen. Die Simulation wird daher auf einem höheren Abstraktionsniveau durchgeführt. Diese Methode wird als *Prototyping* bezeichnet. Die verschiedenen Ebenen der Abstraktion und Detaillierung sind hierarchisch aufgebaut. Mit der konventionellen parametrischen Simulation sind die Möglichkeiten hierfür begrenzt, weil das Systemverhalten aus allgemeinen Funktionsblöcken mit individuellen Parametern dargestellt wird. Diese müssen eine bestimmte Vollständigkeit haben, damit das Programm lauffähig ist. Bei der objektorientierten Simulation kann unter Verwendung verschieden detaillierter Funktionsblöcke, ein Prototyping ermöglicht werden.

Die moderne Automatisierung erfordert Planungssicherheit und Flexibilität, um die Nutzung der Investitionsgüter auf lange Sicht zu garantieren. Kombinierte Systeme, die neben dem realen Prozeß auch die Prozeßsteuerung einschließlich der Softwarekomponenten simulieren, erfahren eine erhöhte Nachfrage, weil sie neben der Prozeß- auch die Softwareoptimierung erlauben. Diese Entwicklung ist erst in wenigen Anwendungen realisiert wie z.B. CNC- oder Roboter-Programmierung, so daß sie heute eines der wichtigen Entwicklungspotentiale in der Simulation darstellt. Die Gegenstände der Simulation und Steuerung weisen in ihrer Beschreibbarkeit Unterschiede auf. Regelkreise und Schaltungen z.B. müssen kontinuierlich, beziehungsweise quasikontinuierlich beschrieben werden. CNC- und Robotersysteme weisen, je nach Detaillierungsgrad, teils kontinuierliche, teils diskrete Komponenten auf. Bei Materialfluß und SPS überwiegen die diskreten Daten bei der Zustandsbeschreibung. Die Simulation und Steuerung von *diskreten* Prozessen stellt das Schwerpunktthema dieser Arbeit dar.

Generell sollte die Simulation nur dort eingesetzt werden, wo analytische Verfahren versagen. Auch bei der Vorauswahl von Versuchsbedingungen für die Simulation sollten die mathematisch-/analytischen Möglichkeiten voll ausgeschöpft werden, da hierdurch die Experimentierphase entscheidend verkürzt werden kann. Die Simulation ist jedoch sehr oft das *einzige* Mittel, um zu gestützten Erkenntnissen zu gelangen. Deshalb werden in den folgenden Abschnitten einige grundlegende Zusammenhänge vorgestellt.

## 3.1 Grafische und textuelle Programmdarstellung

Die geschichtlichen Anfänge der Simulation liegen in der direkten Benutzung von Programmiersprachen. Häufige simulationsspezifische Funktionen werden durch *Simulationsbibliotheken* unterstützt, um die Simulatoren komfortabler zu gestalten. Ein Klassiker dieser Kategorie ist z.B. die FORTRAN-Simulationsbibliothek GPSS. Der Bedienungskomfort dieser Simulatorkategorie kann durch eine konsequente Modularisierung oder Objektorientierung gesteigert

werden. Der Programmierer kann alle seine Wünsche verwirklichen, weil in den eingesetzten Hochsprachen je nach Bedarf eigene Funktionen implementiert werden können. Dieser Flexibilität stehen aber Nachteile durch einen hohen Qualifikations- und Zeitbedarf gegenüber, außerdem sind die großen Mengen von Programmtext schwer zu warten und anzupassen. Eine Erleichterung bieten *parametrische* Simulationssysteme, in denen die Bausteine vordefiniert sind, und durch reine Parametrierung nach Belieben in ein Modell eingesetzt werden können. Der großen Einfachheit dieses Konzepts stehen die beschränkten Möglichkeiten gegenüber. Die Gemeinsamkeit der beiden vorgenannten Verfahren ist die textgebundene Arbeitsweise.

Um Simulationergebnisse anschaulich darzustellen und das Simulationsmodell leicht verständlich zu machen, sind Grafiken das adäquate Mittel. Eine entsprechende Visualisierung des Modells und dessen Zustands ist mit textorientierten Simulatoren möglich, erfordert aber nach der Modellierung noch eine explizite Programmierung der Grafikausgabe.

Diesen zusätzlichen Arbeitsaufwand umgehen Simulationssysteme, bei denen die Modellierung *grafisch- interaktiv* erfolgt. Das Modell wird dabei aus vorgefertigten oder selbstdefinierten *Bausteinen* am Bildschirm mit Hilfe einer Maus oder eines anderen Eingabegerätes grafisch erzeugt. Jeder eingesetzte Baustein verfügt über eine definierte Funktionalität, die eigentliche Programmerstellung auf Codeebene kann dem Benutzer verborgen bleiben. Für den Anwender sind also keine Programmierkenntnisse erforderlich und Modellierung und Visualisierung des Modells geschehen in einem Arbeitsschritt.

Das Verbergen des eigentlichen Programmcodes der Bausteine vor dem Benutzer ist auch ein Aspekt der *objektorientierten* Programmierung. Die überschaubare Struktur der Interaktionen zwischen Objekten durch Informationhiding (Versteckprinzip) prädestiniert den objektorientierten Ansatz für die grafische Simulation. Durch die Mechanismen der Vererbung und der Erweiterbarkeit von Bausteinkästen durch die Spezifizierung eigener Klassen, erhält der Anwender weitere Unterstützung, die den grafisch- objektorientierten Ansatz als *den* der Zukunft erscheinen lassen.

Ein Maximum an Flexibilität bieten grafisch- objektorientierte Simulatoren. Sie haben zusätzlich die Fähigkeit, Probleme, die die Möglichkeiten der vordefinierten Bausteine übersteigen, durch die Integration einer objektorientierten Programmiersprache, zu lösen.

Nach dem Vergleich grafischer und textueller Simulation erfolgt nun die Betrachtung der Zeitdarstellung, die ein wichtiges Klassifikationsmerkmal von Simulatoren ist.

## 3.2 Zeitdarstellung in der Simulation

Ein wichtiges Merkmal von Simulatoren ist ihr Konzept zur Zeitsteuerung. Um Zeiten für Computerprogramme handhabbar zu machen, muß Zeit abstrahiert und diskretisiert werden. Kontinuierliche Darstellungen sind nur bei Analogrechnern möglich. Eine weitere erwünschte Abweichung von der Realität ist die Beeinflussung der Ablaufgeschwindigkeit durch den Benutzer. Einige schnell ablaufende Vorgänge erfordern eine „Zeitlupe“ zur Beobachtung, andere eine „Zeitraffung“, um möglichst schnell Simulationsergebnisse zu erhalten. Im Folgenden werden die verschiedenen Möglichkeiten, Zeit auf Programmebene darzustellen, beschrieben.

### 3.2.1 Echtzeitsteuerung

Die in allen gängigen Computern eingebaute Uhr kann abgefragt werden und als Eingangsvariable für zeitabhängige Funktionen genutzt werden. Hierfür müssen aber alle Größen als Funktion der Zeit bekannt sein. Bei Differentialgleichungen *f(dt)* oder den Zustandsvektoren komplexer oder stochastischer diskreter Systeme ist das aber nicht möglich. Das Verfahren läßt sich deshalb in den meisten Fällen, wo Simulation interessant ist, nicht einsetzen. Bei Systemen, die simuliert werden sollen, können aber Aussagen über Zeitabschnitte getroffen werden. Bei der Simulation von Differentialgleichungen ist dieses jeweils das Zeitinkrement $\Delta t$, bei diskreter Simulation können wir Ereignisse vorausbestimmen, z.B. daß ein Werkstück *n* Sekunden nachdem es in eine Maschine eingetreten ist, diese wieder verlassen wird. Die so ermittelten Zeitpunkte können an der Rechneruhr synchronisiert werden. Das geschieht dadurch, daß in jedem Abfragezyklus getestet wird, ob die Systemzeit schon größer als die Endzeit eines Ereignisses ist, gegebenenfalls wird dieses Ereignis durchgeführt. Eine Voraussetzung für die Simulation in Echtzeit ist also die Gewährleistung, daß alle Berechnungsergebnisse vor der nächsten Zeitabfrage vorliegen.

Bei schnell ablaufenden Vorgängen oder komplexen Berechnungen kann die Berechnung des Endzustandes zu lange dauern, wodurch die Echtzeitanforderung verletzt wird. In Fällen wo dieses eintritt, ist das Echtzeitverfahren ungeeignet. Nützlich ist das Verfahren dort, wo Programme mit Prozessen interagieren, bei Steuerungen von Maschinen und Anlagen, bei der Prozeßüberwachung und der Steuerung durch Simulation wie beispielsweise bei Fernmanipulatoren der unbemannten Raumfahrt. Für die reine Simulationsanwendung ist der Programmablauf in Echtzeit meistens nicht von Interesse.

### 3.2.2 Quasikontinuierliche Zeitsteuerung

Eine gängige Methode, die Zeit von Simulationen zu steuern, ist die Definition einer programminternen Variablen *t*. Die Berechnungen und Abfragen des Programms werden wiederholt ausgeführt und nach jeder Ausführung wird die Variable *t* um ein Inkrement $\Delta t$ erhöht. Bei zeitschrittabhängigen Funktionen *f(dt)* bzw. $f(\Delta t)$ wird der aktuelle Zustand am Ende des Zyk-

lusses gespeichert. Bei Ereignissen wird untersucht, ob *t* schon größer oder gleich als ihr vorausberechneter Eintrittszeitpunkt ist. Wird der Grenzübergang $\Delta t \rightarrow 0$ durchgeführt, liegt eine kontinuierliche Darstellungsweise vor. Da dieser wegen der gegen unendlich strebenden Zykluszahl nicht realisierbar ist, werden die Schritte $>0$ gewählt, weshalb diese Darstellung nur als *quasikontinuierlich* bezeichnet werden kann.

Die Größe der Konstanten $\Delta t$ beeinflußt das Programm durch den Diskretisierungsfehler. Bei Gleichungen ist jedoch $f(dt) \neq f(\Delta t)$ , bei Ereignissen ist $t_{-1} < T_{Ereignis}$ und $t > T_{Ereignis}$ , wodurch sich Ungenauigkeiten ergeben. Je kleiner $\Delta t$ gewählt wird, um so besser ist in der Regel die Genauigkeit der Simulation. Um einen bestimmten Zeitraum zu simulieren, sind aber auch um so mehr Programmschritte nötig. Die Wahl von $\Delta t$ ist also immer ein Kompromiß zwischen Programmgeschwindigkeit und der zu erwartenden Genauigkeit der Ergebnisse.

Bei diskreten Simulationsmodellen kann die Methode sehr unwirtschaftlich sein, wenn über längere Zeiträume keine Ereignisse anstehen, aber alle laufenden Prozesse in jedem Zyklus abgefragt werden müssen, wodurch die Diskrepanz zwischen Genauigkeit und Geschwindigkeit sehr groß werden kann. Bei kontinuierlichen Modellen ist die Zeitschrittmethode die weitaus verbreitetste Zeitdarstellung.

### 3.2.3 Ereigniszeitsteuerung

Bei der diskreten Simulation kann auch der Systemzustand durch diskrete Größen beschrieben werden. Die Tatsache, daß keine kontinuierlichen Zustandsübergänge möglich sind, erlaubt es, die Betrachtung der Zeit auf die Punkte zu beschränken, wo sich der Zustand ändert. Diese Änderungen heißen Ereignisse[9] und ihnen ist ein eindeutiger Zeitpunkt zugeordnet. Eine andere Tatsache, die Voraussetzung für eine Ereigniszeitsteuerung von Simulationsmodellen ist, ist die, daß der Zeitpunkt jedes elementaren Ereignisses vorher bekannt ist. Das ist möglich, weil das Modell durch Elemente beschrieben wird, die die zeitliche Abfolge von Ereignissen und möglichen Ereignissen beschreiben. Beispielsweise kann ein Prozeß dadurch beschrieben werden, daß ein eingetretenes Element genau *n* Sekunden nach seinem Eintritt fertig bearbeitet ist, und damit den Prozeß verlassen kann. Zum Zeitpunkt des Eintritts kann das Ereignis, daß das Element austrittsbereit wird, vorhergesagt werden. Das ist auch machbar, wenn der Prozeß zufällig gestört werden kann, oder daß die Prozeßdauer zufällig variiert. Das wird durch die Prämisse ermöglicht, daß die Wahrscheinlichkeiten und ihre Verteilungen während der Prozeß läuft als konstant und systemunabhängig angesehen werden. Hierdurch können Ausfälle und statistisch verteilte Bearbeitungszeiten schon beim Eintrittsereignis bestimmt werden. Durch

[9] Ein Ereignis ist bestimmt, unteilbar und eine elementare Änderung im System (Atomitäts- bzw. Diskretheitsaxiom) /KÄMPER 91/.

Addition der vorherbestimmten Prozeßzeit zur aktuellen Zeit wird das Austrittsereignis bestimmt und in eine Tabelle eingetragen.

Um die Simulation durch den Ereignisverwalter steuern zu können, müssen die Funktionen, die beim Ereigniseintritt ausgeführt werden sollen, bekannt sein. Dafür sind sie zusammen mit ihren Parametern in der Ereignistabelle gespeichert. Die Einträge haben also die Form:

*Ereignis(t, Funktion, [Parameter] ).*

Beim Start einer Simulation wird der Startzeitpunkt $t_0$ festgelegt und die Prozesse bestimmt, an denen zu diesem Zeitpunkt ein Ereignis stattfindet. Daraus ergeben sich die Einträge in der Tabelle der Ereignisse. Die Zeit „läuft“ dadurch, daß nachdem alle Ereignisse die zum Zeitpunkt $t_0$ stattfinden bearbeitet sind,

- *t* auf die Zeit des Ereignisses mit der kleinsten Zeit $t_{min}$ gesetzt wird.
- Alle Ereignisse, die zum Zeitpunkt $t_{min}$ stattfinden, werden der Reihe nach bearbeitet[10].
- Alle bearbeiteten Ereignisse werden gelöscht[11].
- Es werden die Folgeereignisse bestimmt und in die Ereignistabelle eingefügt.

Nach dem Abschluß dieser Vorgänge wird ein neues $t_{min}$ bestimmt und der Vorgang beginnt von neuem. So springt die Variable *t* von Ereignis zu Ereignis.

### 3.2.4 Prozeßzeitsteuerung

Die prozeßorientierte Simulation wird ebenfalls von einem Ereignisverwalter gesteuert. In der Queue der Ereignisse finden sich neben den Zeiten in diesem Falle aber keine explizit definierten Ereignisse, sondern *Verweise auf Prozesse,* die Steuerungen von *aktiven Objekten* sind. Diese Prozesse führen, abhängig vom Systemzustand und von ihrem eigenen *Status,* die erforderlichen Aktionen durch, indem sie:

- Objektattribute modifizieren,
- neue Prozesse generieren,
- andere Prozesse aufrufen,
- andere Prozesse zu vorgegebenen Zeitpunkten in die Tabelle des Ereignisverwalters einfügen,
- sich selbst deaktivieren und
- Prozesse beenden /PAGE 91/.

[10] Wenn mehrere Ereignisse, die sich gegenseitig ausschließen, zum gleichen Zeitpunkt eingetragen sind, liegt ein Konflikt vor. Die Reihenfolge der Bearbeitung kann durch die Einfügereihenfolge, eine den Ereignissen gegebene Priorität oder Zufällig bestimmt werden. Die Auswahl der Methode beeinflußt das Simulationsergebnis.

[11] Ereignisse, die durch zuvor bearbeitete Ereignisse undurchführbar geworden sind, können zurückgestellt oder ebenfalls gelöscht werden

Im Gegensatz zur ereignisgesteuerten Simulation wird bei der Prozeßsteuerung erst zum Zeitpunkt des Aufrufs modellzustandsabhängig bestimmt, welche Aktionen durchzuführen sind. Dieses Konzept erlaubt es einem Prozeß, der Aktionen durchzuführen versucht, die zum Eintrittszeitpunkt durch Blockierungen oder andere Umstände verhindert werden, diesen anzuhalten, und ihn abhängig von anderen Prozessen wieder fortzusetzen.

Ein Prozeß kann den Status

- *aktiv* haben, wenn er sich in der Queue des Ereignisverwalters befindet,
- *inaktiv* bzw. *beendet* (terminated), wenn er nicht gestartet, erfolgreich abgeschlossen oder storniert wurde und
- *passiv*, wenn er aufgrund der Unmöglichkeit von Aktionen *angehalten* wurde,

annehmen (Abbildung 8).

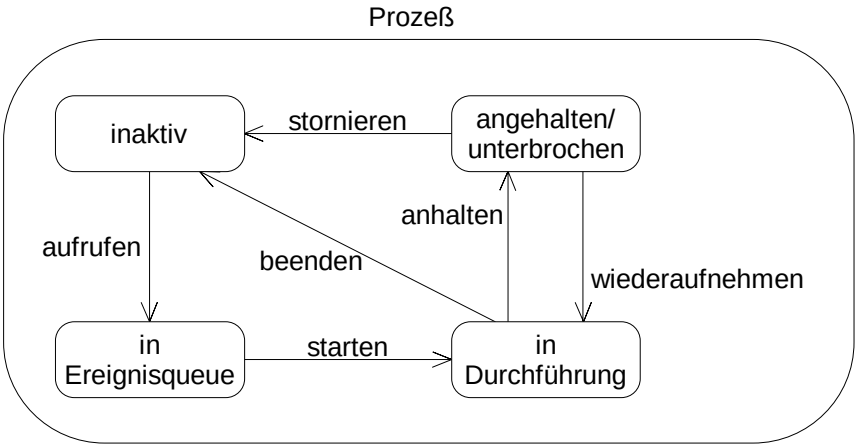


Abbildung 8: Prozeßzustände

Als Beispiel wird eine Rutsche, die als Puffer dient, betrachtet. Wird aus einer vollen Rutsche ein Werkstück entnommen, wird an ihrem Ende ein Platz frei. Ein Prozeß der vergeblich versuchte, diesen zu belegen, und deshalb angehalten wurde, kann wieder aufgenommen werden. Die Steuerung der Aktivitäten geht von den aktiven Objekten aus. Diese Art der Simulation wird auch als maschinenorientiert bezeichnet.

### 3.2.5 Multimodelling

Eine weitere Variante der Simulationssteuerung, die in diesem Zusammenhang interessant ist, wird von Paul A. Fishwick propagiert /FISHWICK 95/. Die als Multimodelling bezeichnete Modellbeschreibung baut auf der objektorientierten Simulation auf. Es werden mehrere Simulationen aufgebaut, die zunächst völlig unabhängig voneinander sind, und auch kontinuierliche, ereignis-, prozeßorientierte oder andere Zeitsteuerungen in Kombination enthalten können. In diesen Modellen werden parallel Simulationsläufe durchgeführt. Es gibt gemeinsame Variablen, auf die verschiedene Prozesse zugreifen. Wird von einem Prozeß eine Eingangsvariable benötigt, deren Wert noch nicht für die aktuelle Zeit aktualisiert ist, wird die Kontrolle an die-

sen Prozeß übergeben, um die gefragten Parameter zu aktualisieren. Das ist notwendig, weil die verschiedenen Modelle zu bestimmten Zeiten (Ereignissen), bei definierten Systemzuständen (Erreichen von Grenzwerten oder Konstellationen), oder kontinuierlich Daten in Form von Parametern und Messages austauschen. Fishwick stellt ein Modellbeispiel vor, das den Vorgang des Teekochens modelliert. Die Wassererwärmung wird dabei als Differentialgleichung kontinuierlich beschrieben. Das Erreichen der Siedetemperatur wird als Ereignis genommen, das den Prozeß „Teebeutel in die Tasse hängen“ in einem prozeßorientierten übergeordneten Modell aufruft (Abbildung 9). Das Modell beinhaltet 3 Hierarchieebenen. Während von Modell 1 nach Modell 2 lediglich eine Verfeinerung stattfindet, wechselt beim Übergang auf Ebene 2 die Betrachtungsweise von einer Beschreibung in diskreten Zuständen zu einer kontinuierlichen Sicht in Differentialgleichungen. Das Modell hat in seiner Gesamtheit einen hybriden Charakter.

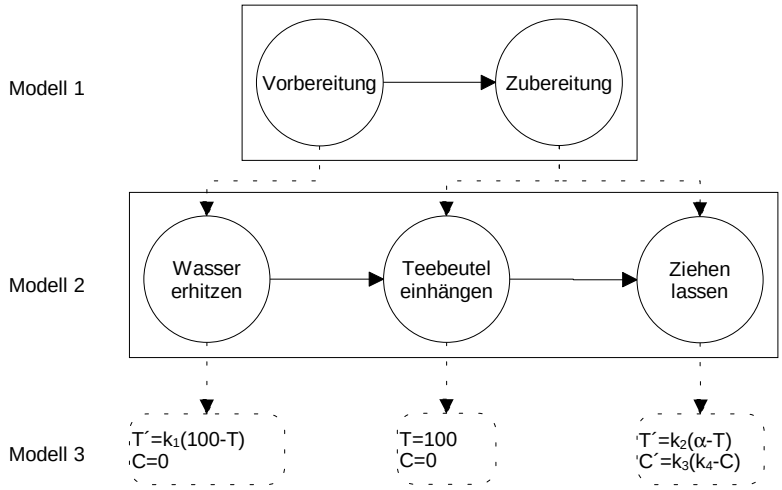


Abbildung 9: Modell in 3 Hierarchieebenen zur Teebereitung (T=Temp., $k_1$=Heizkonstante, C=Konzentration, $k_2$=Abkühlkonstante, $k_3$=Lösungskonstante, $k_4$=Sättigung) /FISHWICK 95/

Interessant ist dieses Konzept für die parallele Simulation von Prozeß und SPS, weil die prozeßorientierte Sicht der Anlage und die quasikontinuierliche Sicht der Steuerung beibehalten werden können.

Das Problem, daß bei Ereignissen, die über lange Zeiträume in der Ereignisqueue warten, viele Programmzyklen im SPS-Modell bearbeitet werden müssen und die Simulation dadurch langsam wird, ist allein durch diesen Ansatz allerdings nicht lösbar.

## 3.3 Differenzen zwischen Simulation und Realität

Bei der Simulation wird von vornherein billigend in Kauf genommen, daß diese von der Realität abweicht. Ohne diese Einschränkung wäre sie unmöglich, ohne das gesamte Universum in jeder Einzelheit darzustellen.

Ein Simulationsmodell muß eine *endliche Systemgrenze* aufweisen, um handhabbar zu sein. Von außerhalb dieser Grenze wirkende Einflüsse können nur durch eine stochastische Daten-

nachbildung, oder eine Nachbildung aus aufgezeichneten Daten berücksichtigt werden. Dazu müssen die relevanten Einflußgrößen bekannt sein. Nicht vorhersagbare Einflüsse wie z.B. Revolutionen, Kriege oder das Abbrennen eines wichtigen Zulieferbetriebes bleiben unberücksichtigt.

Eine weitere Einschränkung ist die *Feinheit* der Modellierung. Sehr komplexe Strukturen wie z.B. Maschinen werden als „Black Boxes“ aufgefaßt, ohne beurteilen zu können, wie die Zuverlässigkeiten der einzelnen Komponenten sich auf das Gesamtsystem oder das Produkt auswirkten. Meistens werden grobe Schätzwerte aus der Praxis für die Gesamtzuverlässigkeit von Komponenten eingesetzt. Die Vorhersagbarkeit eines Systems, das viele schwankende Größen enthält, ist schlecht. Die Fertigung ist als ein solches chaotisches System aufzufassen, das von vielen „Knife´s Edge-Entscheidungen“[12] bestimmt wird.

Die Erforschung chaotischer Systeme steckt noch in den Anfängen und wird zur Zeit mit fraktalen oder statistischen Methoden behandelt. Bisher steht in der Simulation nur die statistische Methode zur Verfügung. Durch zahlreiche Simulationsläufe wird versucht, Mittelwerte für das Verhalten von Systemen zu erhalten und abzusichern. Ob die gebaute Anlage sich annähernd wie die Simulation verhält, oder ob nach „Murphy´s Law“ die Realität völlig vom prognostizierten Verhalten abweicht, bleibt somit fraglich. Der fraktale Ansatz unterstellt, daß auch in scheinbar zufälligen Zahlenströmen immer wiederkehrende Strukturen enthalten sind[13]. Durch die Nachbildung dieser Strukturen läßt sich die Vorhersagewahrscheinlichkeit von stochastischen Systemen signifikant erhöhen. In der Simulation von Fertigungsanlagen werden die Ansätze der fraktalen Mathematik bisher nicht berücksichtigt.

Ein weiterer Unsicherheitsfaktor ist die Unmöglichkeit, alle bekannten Faktoren mit einzubeziehen. Bei der Modellierung werden immer eine Reihe von Faktoren als *unerheblich* nicht im Modell berücksichtigt, ebenfalls mit nicht genau abschätzbaren Auswirkungen.

Trotzdem wird viel Engagement in die Simulation gesteckt. Die Gründe dafür liegen aber wohl weniger in der genauen Vorhersagbarkeit von Ereignissen, als darin, daß man sich mit einer durch Simulation sorgfältig und „richtig“ abgestimmten Anlage immer auf der sicheren Seite befindet, und daß die Simulation das Verständnis für Zusammenhänge fördern kann.

## 3.4 Objektorientierte Simulation

Mit Hilfe von Konzepten der objektorientierten Programmierung lassen sich eine Reihe der Probleme der Simulation lösen. Das hat zwei Aspekte. Zum einen wird dem Benutzer eine Pro-

[12] In chaotischen Systemen kommt es immer wieder zu „zufälligen“ und nicht vorhersagbaren Entscheidungen zwischen verschiedenen Möglichkeiten, die den weiteren Verlauf der Ereignisse oft entscheidend beeinflussen.

[13] Als Beispiel findet man Konturen von Küstenlinien exakt verkleinert in Ausbreitungsgrenzen von Bakterienkulturen wieder. Derartige Parallelen in natürlichen und auch computergenerierten Strukturen sind häufig.

grammoberfläche zur Verfügung gestellt, in der die Bausteine des Modells und möglichst auch die Komponenten des Simulators selbst, als Klassen und Objekte präsentiert und gehandhabt werden können. Zum anderen ist auch die interne Programmierung objektorientiert ausgeführt.

Prozesse werden hier als Objekte aufgefaßt[14]. Ein Prozeß kann verschiedene Zustände annehmen, die innerhalb des Objekts durch Attribute beschrieben sind. Auch alle charakteristischen Eigenschaften und Fähigkeiten bilden eine Einheit mit dem Objekt. Diese gemeinsame Darstellung bietet eine Anzahl von Vorteilen in der Entwicklung und Handhabung von Simulationsprogrammen:

- Einfache (automatische) dynamische Verwaltung der Speicherressourcen des Simulators. Die Reservierung und Freigabe von Arbeitsspeicher geschieht automatisch mit dem Erzeugen und Löschen von Objekten. Bei der Programmierung des Simulators brauchen keine Annahmen über Modellgröße, Bausteinzahl und Beschaffenheit von Komponenten gemacht werden, lediglich der Gesamtspeicher des Programms begrenzt den Modellumfang. Umfangreiche und komplexe Modelle zur Darstellung von Prozessen mit Steuerungen werden optimal unterstützt.
- Eine geringe Anzahl von Verknüpfungen zwischen Modellobjekten untereinander und mit dem Simulationsprogramm erleichtern ihre Wiederverwendbarkeit.
- Die Beschränkung der Verknüpfungen zwischen Prozessen auf die zugelassenen Messages[15] erhöht die Übersichtlichkeit und Zuverlässigkeit des Programms.
- Der Benutzer *sieht* die Objekte wie sie sind. Das ist möglich, weil die Darstellung der Objekte im Computer der Auffassung des Anwenders vom Prozeß entspricht. Der Aufwand zur Gestaltung des Modelleditors sinkt entscheidend, wodurch sich Programmumfang und Geschwindigkeit gegenüber herkömmlichen Programmen erhöhen.

Gemäß des letzten Punktes ist der Bausteinkasten ein zentrales Element der Benutzeroberfläche in grafisch- objektorientierten Simulatoren.

Der Bausteinkasten repräsentiert die Klassen der Klassenhierarchie, aus denen Modellobjekte abgeleitet werden können. Es werden nicht alle Klassen sichtbar, einige abstrakte Klassen, die der Definition dienen, aber nicht selbst einsetzbar sind, werden dem Benutzer verborgen. Den Aufbau einer Klassenhierarchie für die Simulation diskreter Bausteine für Fertigungssysteme hat Bernd-Dietmar Becker vorgestellt (Abbildung 10) /BECKER 91/.

[14] Als Prozeß werden alle physischen Einrichtungen aufgefaßt, die Produkte verändern, bzw. manipulieren. Gemäß der Darstellungsfeinheit kann das z.B. eine Stadt, eine Fabrik, ein Arbeitsplatz oder ein einfacher Vorgang sein.

[15] Messages sind die vom Prozeß verstandenen Befehle und Anfragen. Siehe hierzu Kapitel 2.

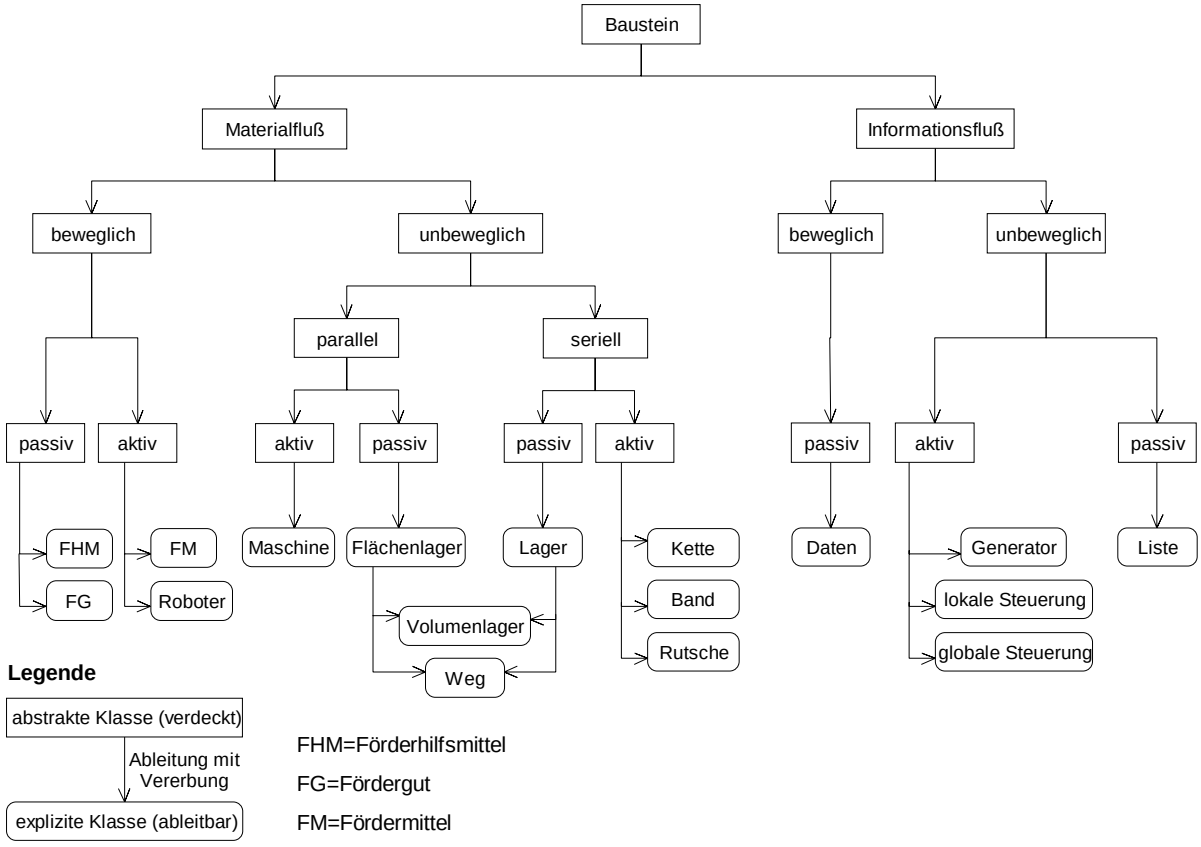


Abbildung 10: Klassenhierarchie für Stückgutprozesse /BECKER 91/

Aus den dargestellten expliziten Bausteinklassen kann der Anwender nach Belieben eigene Klassen ableiten und ihre Instanzen in Modelle einsetzen. Das Modell ist eine gesonderte Klasse. Aus ihr können als einziger Instanzen abgeleitet werden, die für sich allein stehen und nicht selbst in einem Modell enthalten sind. Aus allen anderen Klassen können nur innerhalb von Modellen Instanzen erzeugt werden.

*Aktive Elemente* können, im Gegensatz zu *passiven Elementen*, Ereignisse auslösen. Die Methoden von passiven Elementen können nur von außen aufgerufen werden.

Die aktiven unbeweglichen Informationsflußbausteine, *lokale* und *globale Steuerung*, stellen dem Anwender eine Programmiersprache zur Verfügung, aus denen Klassen erzeugt werden können, die den Zustand der Modelle beliebig manipulieren und den Simulator so zum allgemeinen Werkzeug für diskrete Simulation machen.

Es können Bausteine erzeugt werden, die sich aus der Klasse Modell ableiten und beliebige Instanzen der Klassen Modell und Baustein enthalten. Sie lassen sich dann wie andere Bausteine auch in Modelle einsetzen. Hierdurch wird eine beliebige Modellhierarchie ermöglicht.

Die expliziten Klassen werden dem Anwender in einem Bausteinkasten mit Vorgabeparametern (Defaultwerten) angeboten und können von hieraus abgeleitet werden. Die abgeleiteten Instanzen erben die Defaultwerte. Durch zwei Methoden kann die Vererbung unterbunden werden. Erstens, durch explizites Abschalten, wodurch Veränderungen auf höherer Ebene nicht übernommen, und die alten Werte fixiert werden. Zweitens, durch Überschreiben geerbter

Werte. In diesem Falle werden die geänderten Werte beibehalten. Durch die Vererbung können Änderungen an Klassen für alle Instanzen im Modell übernommen werden, wodurch sich der Änderungsaufwand für Modelle stark reduzieren kann.

Die von Becker vorgestellte Hierarchie ist eine übersichtliche Klassifizierung von Bausteinen für Stückgutprozesse und dient als Grundlage für das objektorientierte Simulationsprogramm SIMPLE++. Für andere Anwendungen können aber völlig andere Elemente und Ableitungshierarchien vorteilhaft sein. Das Beispiel veranschaulicht durch seine Übersichtlichkeit aber sehr gut den Aufbau objektorientierter Simulationsprogramme und stellt eine geeignete Basis für die Entwicklung anderer Klassenhierarchien dar.

Zu einem Simulationsprogramm gehören neben den Bausteinen auch Elemente, die die *Durchführung von Simulationsversuchen* und die *Zeitverwaltung* spezifizieren. *Versuchsauswertung* und *grafische Repräsentation* des Modells können je nach Ausstattung ebenfalls mit dem Simulator vorgenommen werden. Hierfür stehen besondere Klassen zur Verfügung.

Ein wichtiger Baustein dient der Zeitsteuerung und Aktivierung des Modells. Die Funktionalität dieses Bausteins ist vom Konzept der Zeitsteuerung abhängig (siehe Abschnitt 3.2 Zeitdarstellung in der Simulation, S. 23). In ereignis- oder prozeßorientierten Simulatoren sind auch die Ereignisse Instanzen der Klasse „Ereignis“. Der Ereignisverwalter verwaltet die anstehenden Aktionen und aktiviert die entsprechenden Methoden.

## 3.5 Trennung zwischen Material- und Datenfluß

Das Konzept dieser Arbeit beruht darauf, Anlagenteile als Objekte gemäß einer objektorientierten Simulation aufzufassen. Während die Attribute den Systemzustand beschreiben, bestimmen die Methoden die Veränderung des Systemzustands. Attribute können den Zustand von Materialflußelementen beschreiben, z.B. „Werkstückträger in der Spannstation vorhanden“ oder den Zustand von Speichern oder Steuerleitungen, wie z.B. „Merker ´Werkstückeinlauf´ gesetzt“ oder „Steuerleitung ´Vereinzeler öffnen´ führt Spannung“. Methoden können als Ein- und Ausgänge Daten oder Material in allen Kombinationen haben. Tabelle 3 zeigt einige Beispiele für Methoden mit einfachen Ein- und Ausgängen.

| | **Eingang** | |
|---|---|---|
| **Ausgang** | **Signal** | **Material** |
| **Signal** | SPS-Element | Sensor |
| **Material** | Aktor | Mechanisches Element |

Tabelle 3: Beispiele für die Verknüpfung von Materialfluß- und Informationsflußebene

Der Ausgang einer Methode kann einem Eingang dieser Methode selbst entsprechen. In diesem Falle wird der Zustand, der zum Beginn der Methode bestand, überschrieben. Ein Beispiel ist

das Umlagern von Material. Beim Bewegen eines Gegenstands verschwindet dieser von seiner Entnahmestelle und taucht an seinem Ziel auf. Ohne das Löschen an der Entnahmestelle würde er sich abweichend von der Realität verdoppeln.

Die Möglichkeit gleicher Ein- und Ausgänge verdeutlicht die Methode „Umlagern“, die an Ein- und Ausgängen den Attributtyp Material hat:

*Eingänge: Gegenstand an Position 1, Position 2 ist frei*

*Ausgänge: Position 1:=frei, Inhalt v. Position 2:=Gegenstand.*

Im Unterschied dazu stehen Daten auch gleichzeitig an verschiedenen Orten zur Verfügung.

Die Festlegung, ob es sich um Daten oder Gegenstände handelt, ist in der Realität bei einem vorgegebenen Detaillierungsgrad eindeutig. Bei der Veränderung der Detaillierungsfeinheit können aber Veränderungen auftreten. In einer Pneumatikleitung kann der Zustand „Druck vorhanden“ als Informationsattribut betrachtet werden und gleichzeitig an verschiedenen Zylindern Veränderungen verursachen. Bei genauerer Detaillierung wird man aber eine erhöhte Anzahl von Luftmolekülen im Schlauchabschitt finden, die nur jeweils einen Aufenthaltsort haben können, also Objekte des Typs Gegenstand.

Bei Simulationsmodellen soll diese Unterscheidung zwischen Material- und Simulationsfluß häufig nicht so scharf getroffen werden, um das Verhalten von Systemen auf einer abstrakten Ebene zu untersuchen. Ein typisches Beispiel, in dem Material- und Informationsfluß mit gleichen Mitteln dargestellt werden, ist die Modellierung in Petri-Netzen (siehe nächstes Kapitel). Markierungen können das Vorhandensein oder Nichtvorhandensein von Material oder Information repräsentieren. In einem Simulationsprogramm kann also auf die Unterscheidung zugunsten einer größeren Flexibilität verzichtet werden. Für den Anwender ergeben sich aber dadurch auch Fehlermöglichkeiten, durch die grundlegend anderen Eigenschaften. Wenn dem Anwender Bausteine auf einem niedrigen Abstraktionsniveau (Anlagen und Maschinen) zur Verfügung gestellt werden. Zur Vermeidung dieser Fehler ist die Spezifizierung, um welche Kategorie es sich bei Attributen handelt, sinnvoll.

Bei den vorgestellten Objekten zur Darstellung von Prozeß und Steuerung wird aus diesem Grund zwischen *Materialfluß* und *Informationsfluß* unterschieden. Ein Übergang vom einen zum anderen ist nur durch entsprechend definierte Methoden möglich.

Eine sehr abstrakte Repräsentationsmöglichkeit für Simulationsmodelle, in der eine Unterscheidung zwischen Daten und Material nicht zwingend ist, wird im folgenden Abschnitt dargestellt.

## 3.6 Petri-Netze in der Simulation

Petri-Netze sind ein Werkzeug, um Prozesse zu modellieren und zu analysieren. Ein Kennzeichen von Petri-Netzen ist, daß sie Nebenläufigkeiten darstellen können. Das sind Vorgänge, die gleichzeitig ablaufen, sich aber zu bestimmten Zeiten gegenseitig beeinflussen.

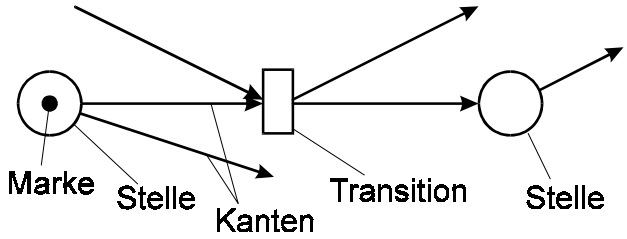


Abbildung 11: Petri-Netz

Petri-Netze verfügen über zwei Arten von Knoten (Stellen für Zustände und Transitionen für Ereignisse) und über gerichtete Kanten. Anschaulich läßt sich das Verhalten von Petri-Netzen auch als Brettspiel beschreiben. Dem Markenspiel liegen folgende Regeln zugrunde:

- Alle Stellen vor einer Transition müssen besetzt sein und
- alle Stellen nach einer Transition frei, um die Transition zu aktivieren.
- Eine aktive Transition entfernt jeden Spielstein, der auf einer Stelle vor der Transition liegt und
- setzt auf jede Stelle, zu der eine Kante führt, eine Marke.
- In jeder Spielrunde werden erst alle Transitionen bestimmt, die aktiv sind und dann die Marken gesetzt.

Zu den einfachen Grundregeln dieser *Bedingungs-/Ereignis-Netze* (B/E) sind bestimmte Erweiterungen gebräuchlich, um komplizierte Sachverhalte einfacher darzustellen:

- *Stellen/Transitions-Netze* (S/T), bei denen Kanten eine bestimmte Anzahl Marken transportieren und Stellen eine bestimmte Anzahl Marken aufnehmen können.
- *Prädikat/Transitions-Netze* (P/T), bei denen Marken über verschiedene Eigenschaften verfügen, die das Aktivieren einer Transition beeinflussen.

P/T-Netze können in S/T-Netze überführt werden und diese wiederum in B/E-Netze. Mit dieser Umwandlung in einfachere Netztypen geht aber eine starke Größenzunahme einher.

Um mit Petri-Netzen Prozesse simulieren zu können, muß die Zeit berücksichtigt werden. Es gibt verschiedene Verfahren, Zeitverzögerungen darzustellen. Hier wird nur die gebräuchlichste und flexibelste Methode erwähnt /KÄMPER 91/. Eine Transition ist mit einer Zeit als Attribut behaftet. Sie muß über diesen Zeitraum ununterbrochen aktiv sein, um zu schalten.

Durch die Darstellung als Petri-Netz können unerwünschte Eigenschafen von Systemen aufgezeigt werden. Es kann passieren, daß keine Transitionen mehr aktiviert werden, weil die Markenanzahl zu hoch ist, keine Marken vorhanden sind oder gegenseitige Blockierungen vorlie-

gen. Es können einzelne Transitionen im Netz vorliegen, die niemals aktiv sind. Diese sind entweder als überflüssig zu entfernen, oder durch eine Veränderung der Netzarchitektur zu beleben. Konflikte liegen vor, wenn mehrere Transitionen auf eine Marke zugreifen, oder dieselbe Stelle zu belegen versuchen. Ob diese Konfliktmöglichkeiten durch Veränderungen im Netz beseitigt, oder Verfahren zur Lösung des Konfliktes eingesetzt werden, hängt vom Anwendungsfall ab. Gebräuchlich ist für die Lösung die *zufällige* Konfliktentscheidung und die Entscheidung durch *Prioritäten*, die den Transitionen oder Kanten als Attribut gegeben werden.

Ob diese unerwünschten Eigenschaften vorliegen, läßt sich bei B/E- und S/T-Netzen durch mathematisch/analytische Verfahren feststellen. Bei P/T- und zeitbehafteten Netzen sind die Analysemöglichkeiten beschränkt, so daß ihr Verhalten simuliert werden muß.

Für die Darstellung großer und komplexer Systeme können hierarchische Petri-Netze entworfen werden. Bei den meisten Hierarchisierungsmethoden verbirgt sich hinter einer einzelnen Transition ein weiteres Netz, das einen Teilprozeß darstellt. Wenn mit dem Projektfortschritt die Feinheit des Modells zunimmt, können einzelne Transitionen nachträglich als Netz *verfeinert* werden (*Petri-Net-Refinement*). Die Einbindung und Handhabung dieser Subnetze kann auf unterschiedliche Weise erfolgen /SCHNEIDER 95/. Es soll an dieser Stelle nicht näher darauf eingegangen werden.

Petri-Netze sind eine sehr allgemeine, abstrakte aber auch universelle Darstellungsform für Fertigungsprozesse, Steuerungen und Programme. Diese Eigenschaften haben zu ihrer weiten Verbreitung im Steuerungs- und Simulationsbereich beigetragen /MÖHRLE 90/.

# 4 Beschreibung von Bausteinen für Transfersysteme

In diesem Kapitel wird eine generische Objektstruktur für die Simulation von modularen Transfersystemen entwickelt, die speziell den Aspekt der Steuerungsimplementation mit berücksichtigt. Die Struktur wird anhand von Beispielen, durch die Analyse lieferbarer Funktionsbausteinreihen erarbeitet.
Es stehen Kataloge für die Transfersysteme TS0, TS2, TS3, TS4 und CTS des Herstellers BOSCH, sowie die Beschreibung einer objektorientierten Modellierung in SIMPLE++ für das modulare Produktionssystem (MPS) FESTO DIDACTIC zur Verfügung /KÜHN 94/. Eine Betrachtung aller Systeme zeigt, daß alle Funktionen durch eine begrenzte Zahl von Grundbausteinen realisiert werden:

1. Werkstückträger (WT) und selbstgetriebene Transportwagen als Förderhilfsmittel zum Transport der Güter im System
2. Informationsträger mit festem oder veränderlichem Informationsinhalt am Förderhilfsmittel
3. Gerade Strecke (uni- und bidirektional)
4. Kurve (uni- und bidirektional)
5. Lift
6. Verzweigung
7. Zusammenführung
8. Kreuzung
9. Streckenabschluß
10. Ein- und ausschaltbarer Stopper (Vereinzeler)
11. Sensor zum Melden eines WTs in einem Streckenbereich
12. Codeleser um den Informationsinhalt der Codiereinrichtung am WT zu lesen
13. Positioniereinheit um WTs in eine genau definierte Lage zu bringen
14. Spanneinheit wie 13, mit der zusätzlichen Fähigkeit, Bearbeitungskräfte aufzunehmen

Aus diesen Bausteinen werden die Transfersysteme der untersuchten Hersteller aufgebaut. In der vorliegenden Arbeit werden Bausteine ausgewählt, die in allen Transfersystemen vorkommen, und so beschrieben, daß aus der Beschreibung Objektklassen für die Simulation gewonnen werden können.

Schon die Beschreibung von Objekten begünstigt immer einen bestimmten Simulationstyp. Hier wird in Hinblick auf die Aufgabenstellung der Arbeit eine *objektbezogene* Darstellung gewählt. Die Beschreibungskonzepte werden so allgemein gestaltet, daß sie *nicht* auf die untersuchten Hersteller und Systeme beschränkt sind, sondern vielmehr als *allgemeines* Konzept zur objektorientierten Modellierung von Transfersystemen unter Einbeziehung ihres Steuerverhaltens verstanden werden können.

## 4.1 Beschreibung des Bausteinkastens TS0 von Bosch

Als Beispiel, an dem die Strukturen entwickelt werden, wird der Bausteinkasten für das Transfersystem „TS0“ der Firma BOSCH gewählt. Für diesen liegen die Bausteinbeschreibungen und Daten in Form eines Katalogs vor /BOSCH TS0/.

### 4.1.1 Beschreibung der Materialflußfunktionen

Das Transfersystem TS0 von Bosch ermöglicht den Transport von Werkstückträgern (WT) einheitlichen Formats. Die Anlagen können für das WT-Format 80x80 mm oder 120x120 mm ausgelegt werden (Abbildung 12). Im gesamten System haben die Werkstückträger eine einheitliche Orientierung. Das bedeutet, daß keine Kurven und Drehstationen vorgesehen sind. Änderungen der Förderrichtung werden über Quertransporte und Querstrecken vorgenommen.

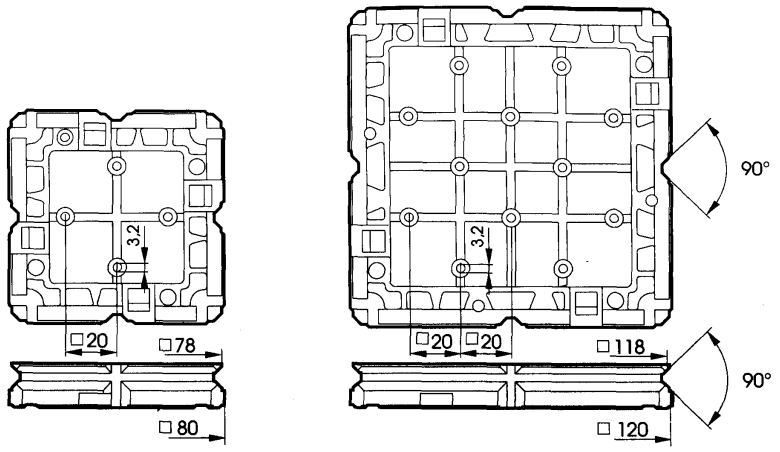


Abbildung 12: Werkstückträger für das Transfersystem TS0 /BOSCH TS0/

Der Transport findet auf Streckenabschnitten variabler Länge durch zwei seitlichen Transportbänder mittels Reibung statt. Diese übernehmen die Beschleunigung, den Transport und können Warteschlangen (*Queues*) bilden. Hierfür braucht nur ein WT gestoppt werden, die auflaufenden WTs gleiten dann auf den weiterlaufenden Transportbändern. Zum Verzweigen werden WTs von kontinuierlich laufenden Quertransporten angehoben, und an die Querstrecken, die auf einem höheren Niveau liegen, weitergegeben (Abbildung 13).

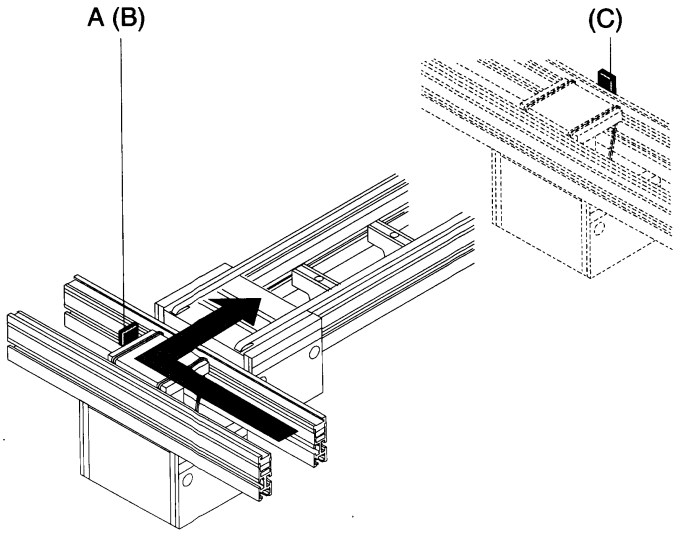


Abbildung 13: Verzweigung im TS0 /BOSCH TS0/

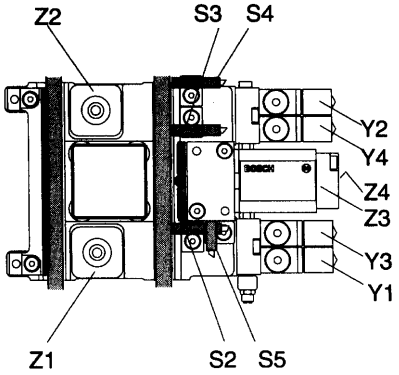


Abbildung 14: Hub- Positioniereinheit HP 0/K /BOSCH TS0/

Abbildung 14 zeigt als erstes Beispiel die Hub-Positioniereinheit, die WTs in einer genau definierten Position spannt und Bearbeitungskräfte aufnimmt. Abbildungen der restlichen Bausteine befinden sich im Anhang 7.1. Die folgende Tabelle beschreibt alle verfügbaren Bausteine des TS0 und beschreibt ihre Funktionen. Anhand der SPS-Ein- und Ausgänge läßt sich der Steuerungsaufwand grob abschätzen.

| TS0 | | | | |
|---|---|---|---|---|
| Bausteine | Aufgabe | Sensoren | Aktoren | SPS Ein- u. Ausgänge |
| Werkstückträger WT 0 (Abbildung 12) | Förderhilfsmittel, Aufnehmen von Werkstücken | - | - | - |
| Strecke ST 0 | Fördern,<br>Stauen,<br>Einbauten aufnehmen | - | - | - |
| Antriebseinheit AS 0 | Band auf Strecke antreiben | - | Motor | E: Motor ein |
| Umlenkeinheit UM 0 | Strecke abschließen | - | - | - |

| TS0 (Forts.) | | | | |
|---|---|---|---|---|
| Bausteine | Aufgabe | Sensoren | Aktoren | SPS Ein- u. Ausgänge |
| Elektrischer Quertransport EQ 0<br>(Abbildung 13, Abbildung 38) | Verzweigen auf Querstrecke | S2: WT nach VE1<br>S3: WT auf EQ<br>S4: Hauptstrecke wieder frei<br>S5: Querstrecke wieder frei | Y1: VE1 Hauptstrecke<br>Y2: WT anheben<br>Y6: VE EQ | E: S2, S3, S4, S5, Anlage bereit, WT übersetzen<br>A: Y1, Y2, Y6 |
| | Verzweigen auf Querstrecke (Streckenende) | S2: WT nach VE 1<br>S3: WT auf EQ<br>S5: Querstrecke wieder frei | Y1: VE1 Hauptstrecke<br>Y2: WT anheben | E: S2, S3, S5, Anlage bereit, WT übersetzen<br>A: Y1, Y2 |
| | Vereinigen von Querstrecke | S6: WT vor VE4<br>S7: WT nach VE4<br>S8: WT auf EQ<br>S9: Hauptstrecke wieder frei<br>S10: WT vor VE5<br>S11: WT nach VE5 | Y3: WT anheben<br>Y4: VE4 Querstrecke<br>Y5: VE5 Hauptstrecke | E: S6, S7, S8, S9 S10, S11, Anlage bereit, WT einschleusen, Priorität Querstrecke<br>A: Y3, Y4, Y5 |
| | Vereinigen von Querstrecke (Streckenende) | S7: WT nach VE4<br>S8: WT auf EQ<br>S9: Hauptstrecke wieder frei | Y3: WT anheben<br>Y4: VE4 Hauptstrecke | E: S7, S8, S9, Anlage bereit, WT eingeben<br>A: Y3, Y4 |
| Elektrischer Quertransport EQ 0/T (Tandemausführung)<br>(Abbildung 41) | Umsetzen auf parallel laufende Strecke (St1 Ende -> St2 Ende) | S2: WT nach VE1<br>S3: WT auf EQ1<br>S8: WT auf EQ2<br>S9: WT nach EQ2 | Y1: VE1 Hauptstrecke1<br>Y2: WT auf EQ1 anheben<br>Y3: WT auf EQ2 anheben | E: S2, S3, S8, S9, Anlage bereit, WT übersetzen<br>A: Y1, Y2, Y3 |
| | Umsetzen auf parallel laufende Strecke (St2 Ende) | S2: WT nach VE1<br>S3: WT auf EQ1<br>S4: WT nach EQ1<br>S8: WT auf EQ2<br>S9: WT nach EQ2 | Y1: VE1 Hauptstrecke1<br>Y2: WT auf EQ1 anheben<br>Y3: WT auf EQ2 anheben<br>Y6: VE4 auf EQ1 stoppen | E: S2, S3, S4, S8, S9, Anlage bereit, WT übersetzen<br>A: Y1, Y2, Y3, Y6 |
| | Umsetzen auf parallel laufende Strecke (St1 Ende) | S1: WT vor VE1<br>S2: WT nach VE1<br>S3: WT auf EQ1<br>S8: WT auf EQ2<br>S9: WT nach EQ2<br>S10: WT vor VE5<br>S11: WT nach VE5 | Y1: VE1 Hauptstrecke1<br>Y2: WT auf EQ1 anheben<br>Y3: WT auf EQ2 anheben<br>Y5: VE5 Hauptstrecke2 | E: S1, S2, S3, S8, S9, S10, S11, Anlage bereit, WT übersetzen, Priorität Querstrecke<br>A: Y1, Y2, Y3, Y5 |
| Positionierung PE 0<br>(Abbildung 36) | Positionierung von Werkstückträgern zur Bearbeitung oder für Ladeopertionen | S2: WT Ankunft<br>S3: WT in Spannposition<br>S4: Station wieder frei<br>S6: WT fixiert | Y1: VE1 vor Station<br>Y2: VE2 nach Station<br>Y3: WT Fixierung | E: S2, S3, S4, S6, Anlage bereit, Spannen, Bearbeitung fertig<br>A: Y1, Y2, Y3, Freigabe Bearbeitung |

| TS0 (Forts.) | | | | |
|---|---|---|---|---|
| Bausteine | Aufgabe | Sensoren | Aktoren | SPS Ein- u. Ausgänge |
| Hub-Positioniereinheit HP 0/K (Abbildung 14) | Positionierung von Werkstückträgern zur Bearbeitung mit Kraftunterstützung | S2: WT Ankunft<br>S3: WT in Spannposition<br>S4: Station wieder frei<br>S5: WT fixiert<br>S7: WT unterstützt | Y1: VE1 vor Station<br>Y2: VE2 nach Station<br>Y3: WT Fixierung<br>Y4: Kraftunterstützung | E: S2, S3, S4, S5, S7, Anlage bereit, Spannen, Bearbeitung fertig<br>A: Y1, Y2, Y3, Y4, Freigabe Bearbeitung |
| Vereinzeler VE 0 | Stoppen von WT | - | Y1: VE öffnen | E: öffnen |
| Positionsschalter PS 0 | Melden von Werkstückträger auf bestimmter Position | S: WT in Position | - | A: WT in Position |

Tabelle 4: Elemente des Bausteinkastens Bosch TS0

### 4.1.2 Zu den Bausteinen gehörige Steuerungs-Funktionen

Zu einigen Bausteinen gehören komplexe Funktionsabläufe, die von Sensoren und Aktoren, sowie Ein- und Ausgängen gesteuert werden. Außerhalb der Bausteine sind nur die Eingänge „Anlage Bereit“, „Ablauf Starten“, bei einigen „Bearbeitung beendet“ und „Priorität Querstrecke vor Längsstrecke“ sowie die Ausgänge „Bereit“ und „Bearbeitung beginnen“ von Bedeutung. Über diese Ein- und Ausgänge findet die gesamte Kommunikation mit anderen Steuerungen statt. Da die Bausteinsteuerungen in allen anderen Aspekten anlagenunabhängig sind, können ihre Funktionen zusammen mit den Bausteinen angegeben werden.

Zur Darstellung der Steuerungslogik stehen eine Reihe genormter Sprachen zur Verfügung, die teilweise ineinander umwandelbar sind. Ihre Auswahl ist daher beliebig. Abhängig von dem späteren Programmeditor, muß das Programm gegebenenfalls in eine andere Sprache übertragen werden, weil nicht alle Editoren eine Schnittmenge von Sprachen haben und sich an die aktuellen Normen halten. Die Definition kann über Funktionspläne (FUP), Anweisungslisten (AWL), Kontaktpläne (KOP), strukturierten Text (ST) sowie Petri-Netze erfolgen. Im Folgenden wird die Steuerung für die Hub-Positioniereinheit HP 0/K des Transfersystems in allen genannten Darstellungsformen gezeigt.

Die Diskussion über die weitere Normung der Darstellungsformen ist nicht abgeschlossen. In dieser Arbeit wird die Darstellung in Petri-Netzen gewählt[16] /KÄMPER 91/. Diese bieten eine gute Übersichtlichkeit, visualisieren jederzeit den Zustand der Anlage, einschließlich der Sensoren und Aktoren, sind leicht erlernbar, haben eine Affinität zur Simulation, und sind jederzeit

[16] Hier kommen einfache Bedingungs-/Ereignis-Netze (B/E) zum Einsatz. Kantengewichte sind immer 1. Es gibt nur schwarze Marken, nur eine Marke pro Stelle ist zulässig. Zum Feuern einer Transition müssen alle vorhergehenden Stellen belegt, alle nachfolgenden frei sein. Für Anforderungen die mit dieser Netzform nicht zu realisieren sind, müssen Erweiterungen wie Zeit, farbige Marken, Kantengewichte usw., wie sie in höheren Netzen (Prädikat-/Transitions-Netze (P/T)) vorkommen, eingesetzt werden. Mit diesen Netzen sind Aufgaben bis hin zu Regelungen mit kontinuierlichen Größen möglich (siehe auch Kap. 3.6).

auf Anweisungslisten übertragbar. Sie sind dadurch auf fast alle Steuerungen transferierbar /ASPERN 93/.

Abbildung 14 zeigt den physikalischen Aufbau der Hub-Positioniereinheit. Den Zylindern *Zi* sind jeweils die Ausgänge *Yi* zugeordnet. Die folgenden Abbildungen zeigen die zugehörige Steuerung in verschiedenen Darstellungsformen. Die Arbeit enthält die Abbildungen aller Bausteine des Bausteinkastens TS0 im Anhang.

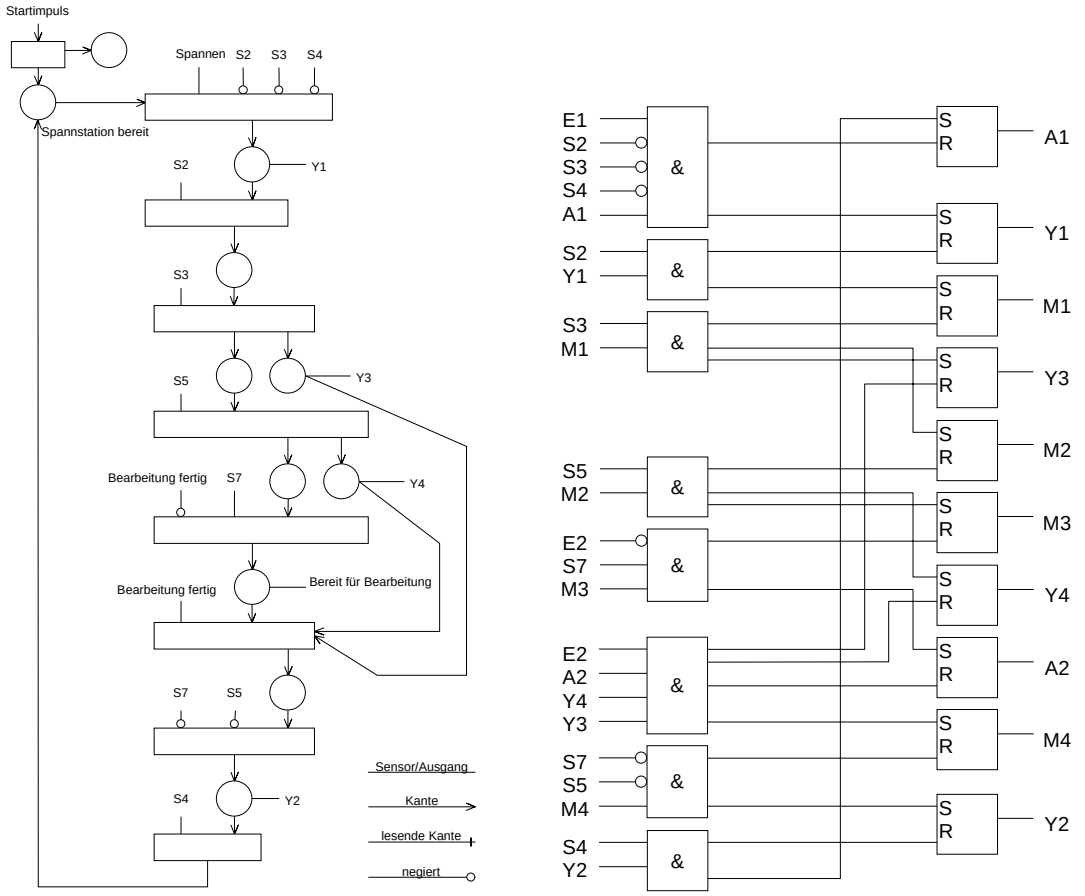


Abbildung 15: Steuerung für die Hub- Positioniereinheit HP 0/K als Petri-Netz und Funktionsplan (Bezeichnungen in Tabelle 4, S. 39)

```
LD    %A1      R     %Y1      R     %M2      AND   %A2      AND   %M4
ANDN  %S2      S     %M1      S     %Y4      AND   %Y4      R     %M4
ANDN  %S3      LD    %S3      S     %M3      AND   %Y3      S     %Y2
ANDN  %S4      AND   %M1      LDN   %E2      R     %Y3      LD    %S4
AND   %A1      R     %M1      AND   %S7      R     %Y4      AND   %Y2
R     %A1      S     %M2      AND   %M3      R     %A2      R     %Y2
S     %Y1      S     %Y3      R     %M3      S     %M4      S     %A1
LD    %S2      LD    %S5      S     %A2      LDN   %S7
AND   %Y1      AND   %M2      LD    %E2      ANDN  %S5
```

Abbildung 16: Anweisungsliste nach /DIN 61131-3/

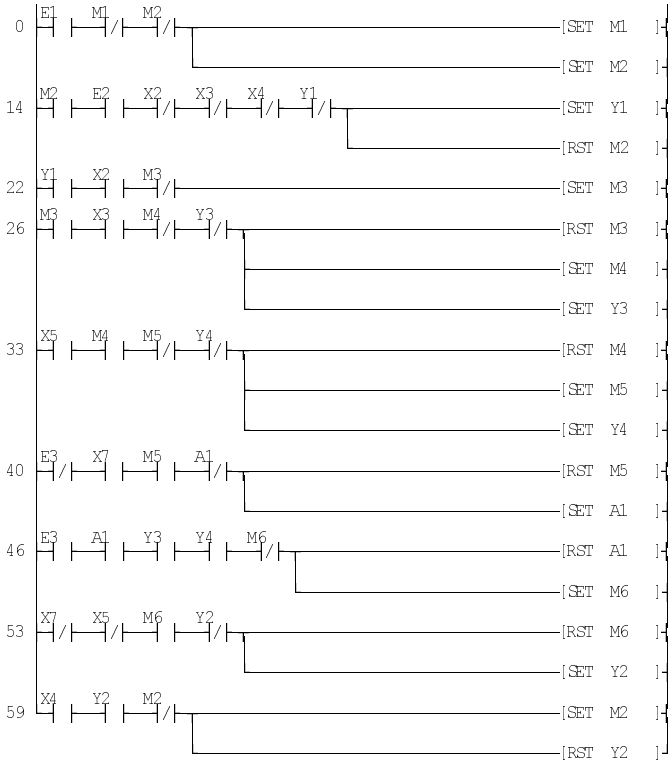

Abbildung 17: Kontaktplan für HP 0/K

Nachdem die Funktionalität der Bausteine beschrieben ist, kann ihre Darstellung in eine objektorientierte Form übertragen werden. Im folgenden Abschnitt werden die Strukturen und Eigenschaften der Bausteine in eine Objektklassenhierarchie eingeordnet.

## 4.2 Darstellung der Bausteine als Objekte

Nachdem die Bausteine im vorigen Kapitel hinreichend genau beschrieben wurden, können aus ihnen Objekte entwickelt werden. Damit ist es möglich, sie im Rahmen einer objektorientierten Simulation so detailliert zu modellieren, daß auch die Steuerungsfunktionen abgebildet werden können.

Zu einem Objekt gehört eine Hülle, die mit einem Namen versehen ist. Zweckmäßigerweise werden aus den Bausteinen, wie sie vom Hersteller angeboten werden, die Objektklassen erzeugt. Diese können im Falle von eigenständigen Teilen, wie z.B. der „Strecke“ in einem Objekt „Halle“ oder „Modell“ verwendet werden, oder sie sind an das Objekt „Strecke“ gebunden, in welches sie eingebaut werden.

Die Bausteine weisen ein charakteristisches Materialflußgrundverhalten auf. Dieses wird durch die Aktoren beeinflußt und über Sensoren beobachtet. Die aufgabengerechte Manipulation des Materialflußverhaltens wird durch die Steuerung mit Hilfe von Aktoren und anhand der Sensoren gesteuert. Die zugehörige Logik ist in einem Steuerungsobjekt untergebracht. Um das Arbeiten der Bausteine in einer komplexeren Fertigungsumgebung zu steuern, haben die Bausteine Ein- und Ausgänge, die von außen abgefragt oder gesetzt werden können. Innerhalb einer vernetzten Fertigungsumgebung können auch die Bausteinsteuerungen Variablen anderer Steuerungen, Prozeß- oder Fertigungsleitrechner abfragen. Für diese Anwendung sind Kom-

munikationsfunktionen zur Verfügung zu stellen. Für die interne Strukturierung der Bausteine ergibt sich aus den Aufgaben eine Unterteilung gemäß Abbildung 18.

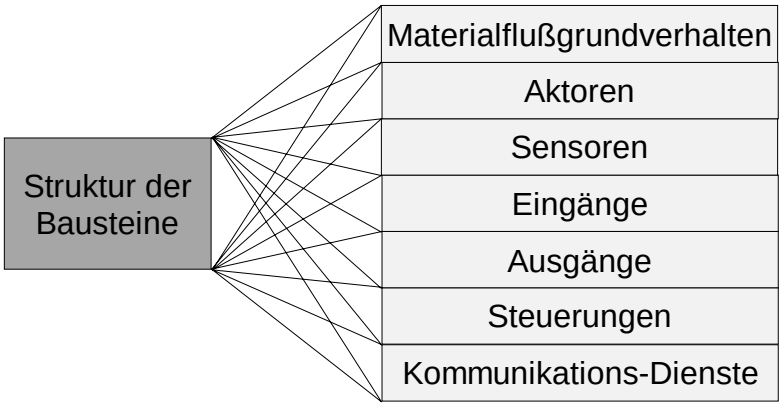


Abbildung 18: Elemente der Bausteine

Für diese Elemente werden Klassen entwickelt und zur Klasse „Baustein“ zusammengefügt. Durch die objektübergreifenden Dienste „Eingänge“, „Ausgänge“, und „Kommunikations-Dienste“ [17] ist der hierarchische Aufbau der Kommunikationsstruktur eines Fertigungssystems möglich. Die Objektklassen für die Bausteine des TS0 sind im folgenden Kapitel definiert.

## 4.2.1 Objektbeschreibungen

Grundlage für eine objektorientierte Darstellung ist die Definition von Klassen, aus denen später die Objekte abgeleitet werden. Sie müssen dazu so allgemein gehalten werden, daß sie die gesamte Gruppe von Elementen, die von der Klasse beschrieben werden soll, als Instanzen von ihr abgeleitet werden und dafür nur die Attribute, mit den entsprechenden Werten belegt werden müssen. Um eine geeignete Strukturierung der Klassen vorzunehmen, werden im folgenden die Eigenschaften der verschiedenen Komponenten des Transfersystems beschrieben. Aus diesen Eigenschaften können später Attribute und Methoden entwickelt werden. Vorgänge, die später als Methoden oder Attribute repräsentiert werden sollen, sind *kursiv* dargestellt.

### 4.2.1.1 Materialflußgrundverhalten

#### Werkstückträger

Der Werkstückträger (WT) ist ein passives[18] Element, welches sich auf Strecken und elektrischen Quertransporten, sowie außerhalb des Transfersystems befinden kann. Ein WT kann ein oder mehrere Werkstücke *aufnehmen* und *abgeben*. Über eine Kodierung kann er für die Steue-

[17] Für die Kommunikation kommen verschiedene Verfahren in Frage. Als herstellerunabhängig sind FMS (Fieldbus Message Specification nach DIN 19245) und MMS (Manufacturing Message Specification nach ISO 9506) zu bevorzugen.

[18] Pssiv bedeutet, daß die Methoden durch Proßesse von außerhalb des WTs aufgerufen werden müssen, und nicht vom Element selbst ausgehen können.

rung lesbare *Informationen transportieren*. Er kann *Sensoren aktivieren* und kann von Aktoren *gestoppt* und *umgelagert* werden.

#### Strecke

Das einzige Objekt, das unabhängig von einer Strecke eingesetzt werden kann, ist die „Strecke“ selbst. Auf Strecken können WTs *aufgelegt* werden. Auf dieses Ereignis folgt die *Beschleunigung* der WT auf *Bandgeschwindigkeit*. Mit dieser Geschwindigkeit werden sie *transportiert*, bis sie *entnommen* werden, oder *auf ein Hindernis laufen* und *gestaut* werden. Vor Hindernissen können sich Queues bilden. Wird der erste WT einer Schlange freigegeben, beschleunigen alle aufgelaufenen WTs wieder auf Bandgeschwindigkeit. Alle Sensoren und Aktoren werden auf der Strecke *montiert*.

### 4.2.1.2 Aktoren

Aktoren verknüpfen den Signalfluß mit dem Materialfluß. Ihr Einwirken auf den Materialfluß ist abhängig von ihrem Signalzustand.

#### Vereinzeler

Der Vereinzeler ist an eine Strecke gebunden, und kann einen ankommenden WT *stoppen*.

#### Elektrischer Quertransport

Wird er *aktiviert* (angehoben), während durch einen Vereinzeler ein WT über ihm gestoppt ist, wird dieser der Strecke *entnommen*, vom Quertransport *aufgenommen* und in Querrichtung *beschleunigt*.

Abbildung 19 zeigt, welche Vorgänge die Elemente „Strecke“ und „Elektrischer Quertransport“ mit WTs durchführen können

#### Fixierung und Kraftunterstützung

Fixierung und Kraftunterstützung, die in Positioniereinheiten enthalten sind, können WTs *festhalten* und verhindern jede Bewegung von WTs. Sie dürfen nur auf gestoppte WTs wirken.

### 4.2.1.3 Sensoren

Sensoren verknüpfen den Materialfluß mit dem Signalfluß. Sie stellen bezüglich der Signalflußrichtung das Gegenstück zu den Aktoren dar.

#### Positionsschalter

Der Positionsschalter wechselt den *Signalzustand* von 0 auf 1, wenn ein WT sich direkt an der Einbauposition befindet. Der Bereich der WT-Erfassung liegt in der Größenordnung von 1 cm.

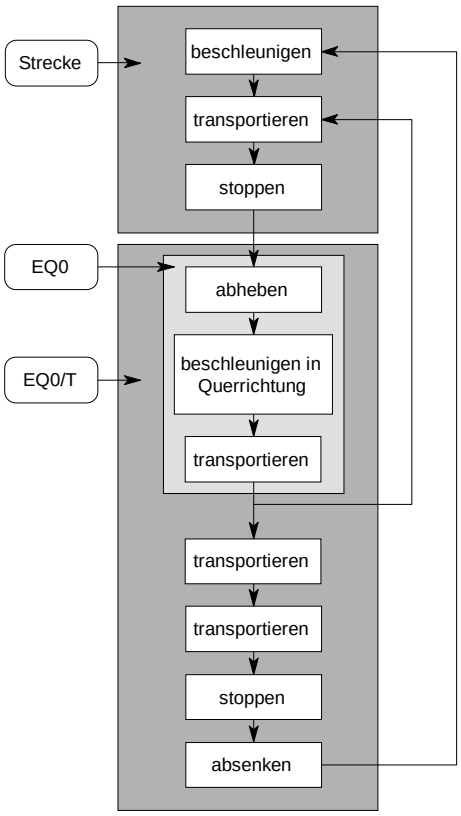


Abbildung 19: Vorgänge des Materialflußverhaltens für WTs

#### 4.2.1.4 Eingänge

Eingänge sind *Variablen* außerhalb einer Steuerung, deren Zustand *abgefragt* werden kann. Eingänge innerhalb einer Steuerung können als *Merker* oder *Ausgänge* direkt abgefragt werden. Abfragen von anderen Steuerungen und Leitrechnern müssen über *Netzwerk-* und *Kommunikationsdienste* vorgenommen werden. Wie externe Abfragen getätigt werden, ist herstellerabhängig. Innerhalb dieser Arbeit werden externe Signalzustände über ihren *Objektpfad*[19] abgefragt, wie es in den modernen objektorientierten Kommunikationsstandards unterstützt wird.

#### 4.2.1.5 Ausgänge

Ausgänge sind analog zu Eingängen Variable, die außerhalb der lokalen Steuerung Bedeutung haben. Zum einen können sie realisiert werden, indem Merker und Ausgänge der Steuerung von anderen Steuerungen und Rechnern über das Netzwerk *gelesen* werden, oder indem Speicheradressen in anderen Steuerungen über das Netzwerk manipuliert werden (*Alarme*). Ebenso wie bei den Eingängen werden externe Adressen in dieser Arbeit über den Objektpfad angesprochen.

[19] Beginnend bei der Basisklasse, die alle anderen Klassen enthält, wird über die Objekthierarchie bis zum gesuchten Objekt verzweigt.
Beispiel: .Fabrik.Halle5.Zerspanung.Bearbeitungszentrum4.Bereit

#### 4.2.1.6 Steuerungen

Steuerungen sind reine Signalflußelemente. Abhängig von *Sensor*-, *Eingangs*- und *Ausgangszuständen*, sowie dem Zustand der Steuerung (*interne Merker* und *Timer*) werden *Ausgänge* und *Aktoren* gemäß der *Programmsyntax* manipuliert. Verschiedene Steuerungen können über Ein- und Ausgänge zu komplexen Systemen verknüpft werden. Mit den Signalen aus dieser Klasse wird der gesamte Materialfluß gesteuert.

#### 4.2.1.7 Kommunikations-Dienste

Kommunikation ist immer dann erforderlich, wenn Daten aus anderen Steuerungen gelesen werden müssen, oder andere Steuerungen Daten abfragen. Wie die hardwaremäßige Realisierung und das zugehörige Protokoll aussehen, ist uneinheitlich. Die aktuellen Protokolle MAP und PROFIBUS (siehe auch Fußnote 17 auf S. 42) bauen auf einer objektorientierten Datenhandhabung auf. Solange die Entscheidung über das Kommunikationsverfahren nicht getroffen ist, wird in dieser Arbeit die systemunabhängige Adressierung externer Komponenten über Objektpfade vorgenommen. Diese ist später leicht auf die genannten Kommunikationsprotokolle zu übertragen.

Aus den beschriebenen Objekten und der Spezifikation ihrer Einsetzbarkeit kann die Klassenhierarchie für den Bausteinkasten abgeleitet werden, und die Methoden für die Objekte können beschreiben werden. Das geschieht im nächsten Abschnitt.

### 4.2.2 Klassendefinition

Für die Bausteine des Bosch-Systems TS0 wird hier eine Klassenhierarchie entwickelt. Tabelle 5 stellt die Basisklassen, die für das System definiert werden, dar. Tabelle 6 zeigt, wie die komplexeren Einbauten des Systems aus Basiselementen zusammengesetzt werden. Die aus den Elementen erzeugbare Hierarchie zeigt Abbildung 20.

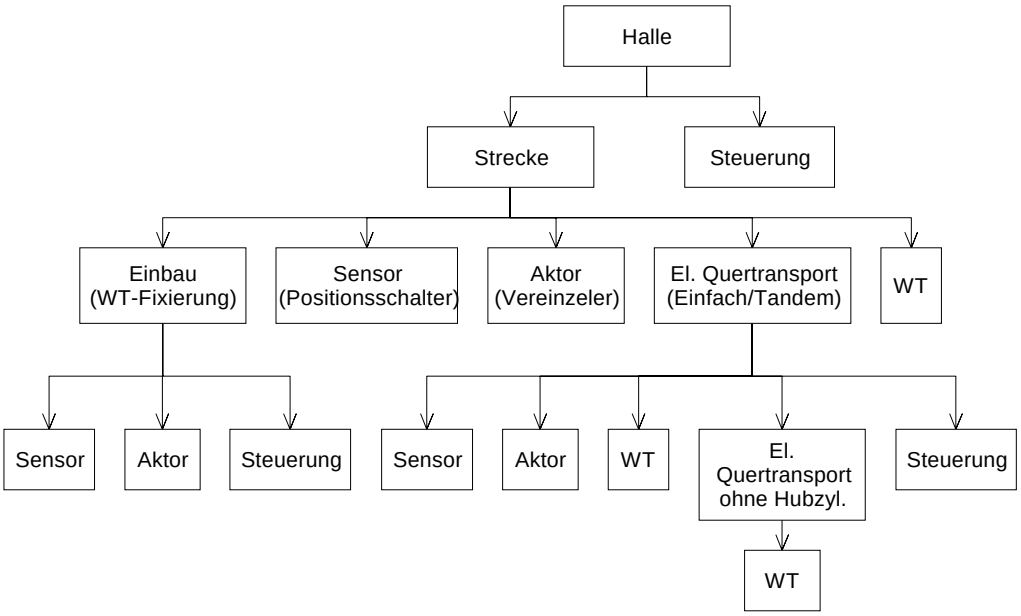


Abbildung 20: Klassenhierarchie des Bausteinkastens TS0

Die Hierarchie soll die anlagentechnische Umsetzung so nah wie möglich wiederspiegeln. Aus diesem Grund sind die Einheiten so definiert, wie sie vom Hersteller bezogen werden. Der Vorgang der Montage von Elementen an den Halteschienen der Strecke hat im Einsetzen der entsprechenden Objekte in eine Instanz der Klasse „Strecke“ seine Entsprechung. Die Klasse „Einbau“ hat im Bausteinkasten TS0 drei Vertreter, die Positioniereinheit, die Hub-Positioniereinheit und die Quertransporte. Diese Einbauten bestehen aus den Klassen „Sensor“, „Aktor“ und „Steuerung“. Die Klasse „Steuerung“ enthält den SPS-Code, der für die jeweilige Einheit immer erforderlich ist. Über Kommunikationsdienste kann die Koordination mit einer zentralen Steuerung als Element der „Halle“ und anderen Bausteinsteuerungen geregelt werden.

| Basisklassen für TS0 | | |
|---|---|---|
| **Klasse** | **Methoden** | **Attribute** |
| Halle | Halle aufstellen | Liste der Strecken<br>Liste der Steuerungen |
| Werkstückträger (WT) | Erzeugen<br>Löschen<br>Werkstück aufnehmen<br>Werkstück abgeben<br>Information Aufnehmen (Kodieren)<br>Information abgeben (Auslesen)<br>Sensor Aktivieren<br>Sensor deaktivieren<br>Stoppen<br>Umlagern | Kodierung<br>Geschwindigkeit<br>Position<br>Gestoppt<br>In Queue aufgelaufen<br>WS-Liste |
| Strecke | Strecke erzeugen<br>Strecke löschen<br>Einschalten<br>Ausschalten<br>WT aufnehmen<br>WT abgeben<br>WT beschleunigen<br>WT transportieren<br>WT läuft auf Hindernis<br>WT stoppen<br>Warteschlangenbildung<br>Einbau montieren<br>Einbau demontieren | Bandgeschwindigkeit<br>Betriebszustand (Ein/Aus)<br>Liste der Einbauten<br>WT-Liste<br>Liste der Queues |
| Vereinzeler | Einbauen<br>Löschen<br>WT Stoppen<br>WT Freigeben | Stoppen/Durchgang |
| Sensor/Positionsschalter | Einbauen<br>Löschen<br>Einschalten<br>Ausschalten<br>Abfragen | Signal (Ein/Aus)<br>Verweis auf WT-Position oder eine Variable, die den abgefragten Anlagenzustand repräsentiert |
| Elektrischer Quertransport ohne Hubzylinder | WT aufnehmen<br>WT beschleunigen<br>WT transportieren<br>WT läuft auf Hindernis<br>WT stoppen | Bandgeschwindigkeit<br>Betriebszustand (Ein/Aus)<br>WT-Liste |
| Elektrischer Quertransport mit Hubzylinder | Aktivieren (anheben)<br>Deaktivieren (absenken)<br>WT von Strecke entnehmen<br>WT aufnehmen<br>WT beschleunigen<br>WT transportieren<br>WT läuft auf Hindernis<br>WT stoppen | Bandgeschwindigkeit<br>Betriebszustand (Ein/Aus)<br>WT-Liste<br>Angehoben (Oben/Unten) |
| Fixierung, Kraftunterstützung | WT festhalten<br>Aktivieren<br>Deaktivieren<br>Sensor „Fixiert“ / „Kraftunterstützt“ aktivieren<br>Sensor „Fixiert“ / „Kraftunterstützt“ deaktivieren | Aktiviert |

| Basisklassen für TS0 (Forts.) | | |
|---|---|---|
| **Klasse** | **Methoden** | **Attribute** |
| Eingang | Lesen | Externe Adresse |
| Ausgang | Schreiben<br>Lesen | Externe Adresse |
| Steuerung | Steuerung aufstellen<br>Steuerung entfernen<br>Eingänge lesen<br>Ausgänge schreiben<br>Sensoren lesen<br>Aktoren schalten<br>Programmzyklus abarbeiten<br>Programm laden<br>Konfigurieren | Merkerliste<br>Ausgangsliste<br>Aktorenliste<br>Eingangsliste<br>Programmcode<br>Timerliste |

Tabelle 5: Basisklassen für das Transfersystem TS0

| Zusammengesetzte Klassen für TS0 | |
|---|---|
| **Klasse** | **Enthaltene Klassen**<br>(Namen gem. Tabelle 4, Beschreibung in Tabelle 5) |
| Positioniereinheit | Vereinzeler (Y1)<br>Vereinzeler (Y2)<br>Fixierung (Y3)<br>Sensor (S2)<br>Sensor (S3)<br>Sensor (S4)<br>Sensor (S6)<br>Steuerung (Abbildung 37) |
| Hub- Positioniereinheit | Vereinzeler (Y1)<br>Vereinzeler (Y2)<br>Fixierung (Y3)<br>Kraftunterstützung (Y4)<br>Sensor (S2)<br>Sensor (S3)<br>Sensor (S4)<br>Sensor (S5)<br>Sensor (S7)<br>Steuerung (Abbildung 15) |
| Elektrischer Quertransport EQ 0<br>Verzweigen auf Querstrecke | Sensor (S2)<br>Sensor (S3)<br>Sensor (S4)<br>Sensor (S5)<br>Vereinzeler (Y1)<br>el. Quertransport (Y2)<br>Vereinzeler (Y6)<br>Steuerung (Abbildung 39) |
| Elektrischer Quertransport EQ 0<br>Verzweigen auf Querstrecke (Streckenende) | Vereinzeler (Y1)<br>el. Quertransport (Y2)<br>Sensor (S2)<br>Sensor (S3)<br>Sensor (S5)<br>Steuerung (Abbildung 39) |

| Zusammengesetzte Klassen für TS0 (Forts.) | |
|---|---|
| **Klasse** | **Enthaltene Klassen**<br>(Namen gem. Tabelle 4, Beschreibung in Tabelle 5) |
| Elektrischer Quertransport EQ 0<br>Vereinigen von Querstrecke | Sensor (S6)<br>Sensor (S7)<br>Sensor (S8)<br>Sensor (S9)<br>Sensor (S10)<br>Sensor (S11)<br>Vereinzeler (Y4)<br>Vereinzeler (Y5)<br>el. Quertransport (Y3)<br>Steuerung (Abbildung 40) |
| Elektrischer Quertransport EQ 0<br>Vereinigen von Querstrecke (Streckenende) | Sensor (S7)<br>Sensor (S8)<br>Sensor (S9)<br>Vereinzeler (Y4)<br>el. Quertransport (Y3)<br>Steuerung (Abbildung 40) |
| Elektrischer Quertransport EQ 0/T<br>(Tandemausführung)<br>Umsetzen auf parallel laufende Strecke (St1 Ende -> St2 Ende) | Sensor (S2)<br>Sensor (S3)<br>Sensor (S8)<br>Sensor (S9)<br>Vereinzeler (Y1)<br>el. Quertransport (Y2)<br>el. Quertransport<br>el. Quertransport (Y3)<br>Steuerung (Abbildung 42) |
| Elektrischer Quertransport EQ 0/T<br>(Tandemausführung)<br>Umsetzen auf parallel laufende Strecke (St2 Ende) | Sensor (S2)<br>Sensor (S3)<br>Sensor (S4)<br>Sensor (S8)<br>Sensor (S9)<br>Vereinzeler (Y1)<br>el. Quertransport (Y2)<br>el. Quertransport<br>el. Quertransport (Y3)<br>Steuerung (Abbildung 42) |
| Elektrischer Quertransport EQ 0/T<br>(Tandemausführung)<br>Umsetzen auf parallel laufende Strecke (St1 Ende) | Sensor (S1)<br>Sensor (S2)<br>Sensor (S3)<br>Sensor (S8)<br>Sensor (S9)<br>Sensor (S10)<br>Sensor (S11)<br>Vereinzeler (Y1)<br>Vereinzeler (Y5)<br>el. Quertransport (Y2)<br>el. Quertransport<br>el. Quertransport (Y3)<br>Steuerung (Abbildung 42) |

Tabelle 6: Zusammengesetzte Klassen für das Transfersystem TS0

# 5 Objektorientierte Simulation von Prozeß *und* SPS

In diesem Kapitel werden Anforderungen und Realisierungsmöglichkeiten die kombinierte Simulation und Programmierung von Transfersystemen dargestellt. Dafür wird zunächst die Zielsetzung betrachtet, auf die sich die Aufgabe richtet. Mit den Mitteln der objektorientierten Simulation wird dann im Projekt OSIMPROST, ein Konzept zur Erreichung der Zielsetzung vorgestellt.

Ein Beispiel, in dem ein Transfersystem in einem objektorientierten Simulator modelliert und programmiert wird, schließt dieses Kapitel ab.

## 5.1 Nutzungs- und Anforderungsstruktur

Um ein Programm zielgerichtet entwickeln zu können, müssen die Anforderungen zuvor festgelegt werden. Diese hängen stark mit der Arbeitsweise, die mit dem System angestrebt wird, zusammen /KUHN 93/.

Aufgrund der zu erwartenden Komplexität des Systems, ist die Nutzung auf die Planungs- und Leitstandebene abzustimmen. Benutzt wird das Programm von Planungsingenieuren, Entwicklern, Steuerungsfachleuten und Anlagenbedienern. Bei diesen ist ein grundlegendes Verständnis der Struktur und Arbeitsweise der Anlage vorauszusetzen und eine Einarbeitung in das System zumutbar. Trotzdem muß die Benutzung schnell erlernbar sein. Die Funktionen des Systems sollen flexibel und erweiterbar sein was z.B. durch Makros möglich ist. Standardaufgaben müssen mit wenigen Schritten durchzuführen sein. Die Erfüllbarkeit der einzelnen Anforderungspunkte muß unter Berücksichtigung schon vorhandener Konzepte geklärt werden.

Der Ansatz dieser Arbeit untersucht schwerpunktmäßig, ob die SPS-Simulation mit einem grafisch- objektorientierten Ansatz sinnvoll zu verwirklichen ist. Einzelheiten der Implementierung und der Bedienungsstruktur sind nicht Inhalt dieser Arbeit.

### 5.1.1 Anforderungen an ein kombiniertes System

Das System soll alle Anforderungen eines modernen, objektorientierten diskreten Simulationssystems erfüllen und für Layoutplanung, Anlagendimensionierung, Materialflußplanung und Logistikplanung einsetzbar sein.

Durch die Erfüllung dieser Voraussetzung kann die Verwendung unterschiedlicher Simulationssysteme vermieden werden, deren Zusammenarbeit und Datenaustausch aufgrund fehlender Konzepte und normierter Schnittstellen bisher nicht allgemein verfügbar ist /LESCHKA 95/. In diesem Bereich ist mit Fortschritten, die zu einsatzfähigen Konzepten führen, zu rechnen. Das zeigen erste Arbeiten, beispielsweise von Ralph Splanemann in seiner Dissertation am BIBA in Bremen 1995, der eine Modellbeschreibung auf der Basis von STEP vornimmt, die auch erfolgreich implementiert wurde.

#### 5.1.1.1 Modellierung

Die Modellierungskomponente muß dem Benutzer die komfortable Beschreibungen der Fertigungssrukturen ermöglichen. Für den Einsatz moderner Planungskonzepte sollte das „*Concurrent Engineering*“ unterstützt werden. Die verschiedenen Projektbereiche werden gleichzeitig gestartet und durch kontinuierlichen Informationsaustausch vorangebracht. Durch die Gleichzeitigkeit wird der Planungszeitraum verkürzt. Das fordert auch vom Simulationssystem eine schrittweise Verfeinerung, vom groben Konzept in wenigen Blöcken, bis zum detaillierten hierarchisch gegliederten Modell zu ermöglichen (*Rapid Prototyping*). Dadurch wird die projektbegleitende Simulationsunterstützung ermöglicht. Eine Detaillgenauigkeit, die Steuerungen mit allen Einzelheiten abbildet, ist erst in der letzen Projektphase sinnvoll, wenn die Implementierung der Programme auf die Steuerungen ansteht. Eine zu frühe, zu genaue Modellierung behindert die Arbeit und vergrößert den Simulationsaufwand.

##### 5.1.1.1.1 Weiterverwendung der Simulationsergebnisse, Modelldaten und Programme

Die Erstellung neuer Simulationsmodelle ist zeitaufwendig. Fertige und erprobte Modelle sollen daher systematisch archiviert werden. Möglichkeiten, die sich durch *Variantenbildung* von Modellen, *Wiederverwendung von Modellteilen* und die Verwendung von *vorgefertigten Bausteinkästen* ergeben, müssen ausgenutzt werden. Dafür sind schon bei der Modellbeschreibung Vorkehrungen zu treffen, wie z.B. sorgfältige Dokumentation und Modularisierung.

#### 5.1.1.2 Modellkomponenten

##### 5.1.1.2.1 Darstellung von Fertigungsplätzen

In der Produktion werden Einrichtungen und Arbeitsplätze mit sehr unterschiedlichen Eigenschaften eingesetzt. Das können z.B. Handmontageplätze, durch einen oder mehrere Personen bediente Maschinen, getaktete und taktfreie Automaten, Roboter, Lager mit automatischer oder manueller Beschickung etc. sein. Diese unterscheiden sich hinsichtlich

- Taktgebundenheit,
- gleichzeitige oder exklusive Betriebsbereitschaft durch geteilte Ressourcen,
- Störungshäufigkeit,
- Umrüstzeiten,
- Pausen,
- Ortsgebundenheit,
- Beschaffenheit der Materialzu- und Abfuhr und andere.

Auch umfangreiche Bausteinkästen können nicht alle diese Eigenschaften in allen Kombinationen abdecken. Der Anwender muß deshalb die Möglichkeit haben, Arbeitsstationen nach eigenen Bedürfnissen zu modellieren. Diese selbstdefinierten Bausteine müssen, im Sinne einer Integrierbarkeit in das Simulationssystem, gemeinsame Attribute und Funktionen aufweisen.

#### 5.1.1.2.2 Darstellung des Materialflusses

Der betriebliche Materialfluß wird durch verschiedene Fördermittel, wie z.B. Tragen von Werkstücken, Gabelstapler und Hubwagen, Transportbänder, führerlose Transportsysteme (FTS) und automatische Transfersysteme erledigt.

Die Organisation der Fertigung kann werkstattorientiert, in Gruppen- oder Fließfertigung (unter Berücksichtigung von Varianten), auf Baustellen oder in flexibel verknüpften Stationen erfolgen. Die jeweiligen Auftragsdaten für jedes Los müssen gelesen und berücksichtigt werden. An einer eventuell abschließenden Qualitätskontrolle muß entschieden werden, ob das Produkt in Ordnung, oder mangelhaft ist, und ob es nachgebessert oder ausgesondert werden muß. Dietmar Heumann stellt einen objektorientierten Ansatz zur Lösung dieser Komponenten dar /HEUMANN 93/. Auch ein kombiniertes Simulationssystem muß die genannten Komponenten für den Materialfluß darstellen.

##### Transfersysteme

Die einzelnen Fertigungsplätze werden durch die oben genannten Fördermittel versorgt. Besonderes Augenmerk gilt hier den *flexiblen programmgesteuerten Transfersystemen*, für die, mit Hilfe des objektorientierten Ansatzes, SPS-Programme generiert werden sollen. Diese bestehen aus Komponenten, die verschiedene Aufgaben zu erfüllen haben. Das reicht von steuerungstechnisch anspruchslosen Aufgaben, wie z.B. Ein- und Ausschalten von Bandstrecken, bis hin zu Verzweigungen, an denen, abhängig von den Auftragsdaten, der Auftragspriorität, der Auslastung und Verfügbarkeit einzelner Anlagen, sowie dem Vorhandensein von Hilfsstoffen, Werkzeugen und Montagekomponenten, komplexe Entscheidungen getroffen werden müssen. Die Programmierung derartiger Entscheidungen ist arbeitsaufwendig, beeinflußt aber die Effizienz der Fertigung in erheblichem Maße.

### 5.1.1.3 Spezifikation der Simulationsläufe

Alle Simulationsexperimente gehen von einer Ausgangssituation aus, die der tatsächlichen Situation im Betrieb am Anfang des Betrachtungszeitraums entsprechen soll. Diese simulationsinterne Datenstruktur wird dann mit Hilfe der Simulation manipuliert und bewertet. Die Ausgangssituation muß immer wieder reproduziert werden können, um das Simulationsexperiment zum Vergleich in unterschiedlichen Varianten durchführen zu können, und um bei nicht deterministischen Simulationen eine statistische Absicherung der Ergebnisse vornehmen zu können. Der Begriff Ausgangssituation bezieht sich nicht nur auf physische Zustände, sondern schließt auch Datenbestände und Speicherinhalte mit ein. Gemäß der Automatentheorie wird das Ausgangssignal durch Eingangszustände, interne Zustandsgrößen und Bewegungsregeln eindeutig festgelegt /AUER 94/. Ein Simulator läßt sich somit als Automat beschreiben.

Die Datenerhebung ist sehr aufwendig. Deshalb sollten Betriebsdaten aus der EDV der entsprechenden Bereiche verwendet werden, falls diese verfügbar und kompatibel sind /CIM 91/. Si-

mulationsseitig kann hierfür mit Datenbankschnittstellen die Voraussetzung geschaffen werden.

Die auf *realen* Daten basierende Simulation bietet ein Höchstmaß an Wirklichkeitstreue. Die in dieser praxisnahen Situation getesteten Steuerprogrammkomponenten bieten ein höheres Maß an Zuverlässigkeit, als bei jeder anderen Art der Programmerstellung.

#### 5.1.1.4 Auswertung

Um zwischen verschiedenen, in Simulationsmodellen getesteten Steuerungs- und Anlagenvarianten auswählen zu können, müssen diese anhand vorgegebener qualitativer und quantitativer Kriterien bewertet werden. Beispiele für Kriterien sind:

- Produktivität,
- Durchlaufzeit,
- Auslastung von Komponenten,
- Zuverlässigkeit/Verfügbarkeit,
- notwendige Ressourcen (Hardware/Software, Kapitalbindung),
- Überschaubarkeit und Steuerbarkeit,
- Kosten.

Wenn die Auswahl einer Modellvariante nicht sofort eindeutig zu treffen ist, müssen die Kriterien gewichtet und ihre Erfüllung durch die getesteten Varianten bewertet werden. Dadurch ist in der Regel eine eindeutige und begründbare Entscheidung zu fällen. Die durchgeführten Versuche werden protokolliert und später der Dokumentation des ausgewählten Programms beigefügt werden.

#### 5.1.1.5 Programmierung von Einrichtungen

Ein SPS-Simulator soll die SPS-Programmierung unterstützen. Dazu sind folgende Punkte notwendig:

1. Darstellung der Logik von SPS-Programmen
2. Es muß eine allgemein akzeptierte Programmiersprache zur Verfügung stehen
3. Die Programmiersprache muß sich an bestehenden Normen orientieren
4. Sie muß so mächtig sein, daß auch die komplexen Zusammenhänge der Materialfluß- und Auftragssteuerung programmiert werden können
5. Die erstellten und getesteten Programme müssen auf die Steuerungen übertragen werden können
6. Es müssen SPS-übergreifende Konzepte der Steuerung auf Fertigungsleitrechnerebene verwirklicht werden

7. Beim Test komplexer Steuerungsstrukturen muß das Verhalten der evtl. in der Produktion eingesetzten PPS-Systeme[20], Lagerprogramme, Leitstände... und ihrer Entscheidungen nachgebildet werden. (Hierfür könnten auch die vorhandenen Steuerungsprogramme direkt herangezogen werden, falls diese über einen Simulationsmodus[21] verfügen und eine Schnittstelle zum Simulationsprogramm haben.)

#### 5.1.1.5.1 SPS-Programmierung

Zu den in Kapitel 4 beschriebenen Bausteinen gehören Steuerungsprogramme. Diese können durch Konfiguration der Ein- und Ausgänge unverändert eingesetzt werden. Für übergeordnete Steuerungsaufgaben müssen aber teilweise komplexe Programme erstellt werden. Hierfür ist ein komfortabler Editor nötig. Die Problematik, komplexe Zusammenhänge in primitiven SPS-Programmen beschreiben zu müssen, kann umgangen werden, wenn es gelingt, in höheren Sprachen geschriebene Programme auf SPS-Code zu übertragen. Wenn ein entsprechender Compiler zur Verfügung steht, kann jede beliebige Programmiersprache im Simulationsprogramm zur Steuerungsmodellierung verwendet werden.

##### Programmübertragung zwischen Programmiersystem und SPS

Nach der Modellierung liegt die Steuerungslogik im Simulationsmodell vor. Um die erstellten SPS-Programme weiter zu nutzen, werden sie auf die eingesetzte Steuerung übertragen. Dazu kommen im wesentlichen 3 Methoden in Frage:

1. Manuelle Übertragung auf den Editor der Steuerung. Dabei können Anpassungen an Besonderheiten der Steuerung individuell vom Programmierer vorgenommen werden.
2. Automatische Übertragung der Programme auf die Steuerung durch einen speziellen Postprozessor, der die Besonderheiten der Steuerung berücksichtigt. Für jeden Steuerungstyp ist dieser neu zu konfigurieren.
3. Automatische Übertragung des Programms über eine einheitliche genormte Schnittstelle. Für diesen Fall ist nur ein einziger Postprozessor nötig.

Die erste Variante stellt an Simulations- und Programmiersysteme die geringsten Anforderungen. Sie erfordert lediglich Personal, welches die Programmierung der verschiedenen Anlagen durchführen kann. Wo SPS im Einsatz sind, besteht aber in aller Regel auch die Möglichkeit, diese zu programmieren, gegebenenfalls müssen Programmierer auf Dienstleistungsebene hinzugezogen werden. Nachteile dieser Methode sind jedoch der große Zeitaufwand für die Übertragung, das Risiko von Übertragungsfehlern und die Schwierigkeit, bei Programmüberarbei-

[20] Production-Planing-System / Produktions-Planungs-System

[21] Unter Simulationsmodus ist zu verstehen, daß das entsprechende Programm gemäß seines betrieblichen Verhaltens mit dem Simulator kommuniziert, ohne dabei den betrieblichen Materialfluß real zu beeinflussen.

tungen alle Steuerungen *gleichzeitig* und *einheitlich* auf neuesten Stand zu bringen. Es sind erhöhte Softwarewartungskosten zu erwarten.

Die zweite Variante kann einen erheblichen Aufwand erfordern, wenn die eingesetzten Steuerungen von verschiedenen Herstellern stammen. Es sind außerdem Probleme durch uneinheitliche Funktionsumfänge, Programmstruktutrierungen und Dokumentationslücken möglich. Dieser Ansatz scheint deshalb am wenigsten zukunftsweisend. In der Realität werden diese heterogenen uneinheitlich zu programmierenden Strukturen aber noch häufig vorgefunden /JANZEN 90/.

Die dritte Variante bietet für die Praxis die größten Vorteile. Es besteht keinerlei Herstellerabhängigkeit, wodurch das Programmier- und Simulationssystem auch in heterogenen Automatisierungsumgebungen[22] problemlos eingesetzt werden kann.

## 5.2 Realisierung

In den vorhergehenden Kapiteln wurden die Komponenten eines kombinierten Simulationssystems für Prozeß und Steuerung dargestellt. In diesem Kapitel werden sie nun zu einem Entwurf für das System OSIMPROST zusammengefügt. Noch fehlende Komponenten werden dabei ergänzt.

### 5.2.1 Modellierungskomponenten

Die Modellierungskomponente stellt den Modelleditor dar. Gefordert ist eine grafische Programm- und Modellrepräsentation, die auf einer fensterorientierten Betriebsystemoberfläche dargestellt wird. Zu diesen Oberflächen gehört auch ein Zeigegerät, wie z.B. Maus, Touchscreen, Lichtgriffel, Cyberhandschuh usw., mit dem auf Bildschirmobjekte gezeigt werden kann. Durch verschiedene Tasten auf Zeigegerät und Tastatur können auf den Objekten verschiedene Aktionen durchgeführt werden.

Bei objektorientierten Programmen wird das Modell aus Objekten gebildet, die aus Klassen abgeleitet werden. Bei der vorgegebenen grafisch- objektorientierten Modellrepräsentation werden die Klassen in einem *Bausteinkasten* durch Icons[23] repräsentiert. Beispielsweise durch Mausklick oder Verschieben werden sie durch Ableiten in das Modell eingesetzt. Beim Ableiten von Klassen müssen fehlende Attribute ergänzt werden. Diese müssen bei der Instanzenbildung ermittelt oder abgefragt werden, was z.B. durch das Öffnen einer Eingabemaske nach dem Anklicken geschehen kann. Später muß eine Änderungsmöglichkeit bestehen. Eingesetzte Objekte müssen dazu wieder geöffnet und bearbeitet werden können.

---

[22] Heterogen bedeutet, daß Steuerungen mit verschiedenen Konzepten und Programmiersprachen von verschiedenen Herstellern im Einsatz sind.

[23] Icon bedeutet Abbild. In seiner Darstellung auf dem Monitor weist es eine Schaltfläche aus, deren Funktion mit dem Symbol der Schaltfläche in Zusammenhang steht.

## 5.2.2 Modellkomponenten

Analog zur Realität wird bei OSIMPROST zwischen Prozeß und Steuerung getrennt. Die Prozeßbausteine sind ohne eine zugehörige Steuerung, die die Aktoren bedient, nicht einsatzbereit.

### 5.2.2.1 Prozeßbausteine

Unter dem Prozeß werden die Komponenten verstanden, die den Materialfluß physisch beeinflussen. Im beschriebenen Transfersystem TS0 wird der Materialfluß durch WTs repräsentiert. Diese werden auf Strecken bewegt. Diese Strecken, auf denen die WTs sich bewegen, bestimmen den Materialfluß und dienen zur Aufnahme der Aktoren und Sensoren. Die Strecke selbst verfügt lediglich über den Antriebsmotor als Aktor. Beeinflußt wird der Materialfluß über Einbauten wie Vereinzeler oder Spannstationen. Diese Einbauten sind Bestandteil der Strecke, WTs die sich in diesen Einbauten befinden, sind weiterhin auf der Strecke als Objekte vorhanden.

Neben der Strecke können sich die WTs auch auf elektrischen Quertransporten befinden. Diese heben die WTs von der Strecke, nehmen diese Objekte auf und transportieren sie in Querrichtung (Abbildung 19 und Abbildung 20).

Abweichend vom Grundverhalten kann der Materialfluß durch Aktoren verändert werden. Die Aktoren verknüpfen die Prozeßbausteine mit der Steuerung, indem sie den Zustand der Steuerung in dem Moment abfragen, in dem ein WT ankommt und ein Ereignis auslöst. Abhängig vom Steuerungssignal bestimmen sie das weitere Verhalten. Die vorgestellten Aktoren sind:

- Vereinzeler,
- WT-Fixierung,
- Kraftunterstützung und
- Hubzylinder des Quertransports.

Die andere Richtung des Informationsflusses wird durch Sensoren vorgenommen. Tritt ein WT in einen Sensorbereich ein oder aus, wird ein Ereignis, das die Steuerung aktiviert, ausgelöst. Das Transfersystem TS0 bietet lediglich Positionsschalter an, die WTs auf einer bestimmten Position an die Steuerung melden.

### 5.2.2.2 Steuerungsbausteine

Passende Steuerungsblöcke sind Bestandteil der Bausteinklassen. Werden vom Anwender Bausteine mit Steuerung in ein Modell eingesetzt, muß eine Konfiguration der Steuerungsblöcke vorgenommen werden, um die Einheit lauffähig zu machen.

#### 5.2.2.2.1 Darstellung der SPS-Aufgaben

Ziel der Arbeit ist es, in der Simulation die Steuerungssyntax mit abzubilden und in den Simulationslauf mit einzubeziehen. Für die Darstellung im Programm muß daher eine geeignete Darstellungsform gewählt werden. In Kapitel 4 wurde hierfür die Repräsentation als *Petri-Netz*

gewählt. In /DIN 61131-3/ sind die Darstellungen als Instructionlist/Anweisungsliste (IL/AWL), Structured Text (ST), Ladder Diagram/Kontaktplan (LD/KOP) und Function Block Diagram/Funktionsblöcke (FBD/FUP) genormt. Die Aufnahme von Petri-Netzen in eine IEC-Norm als Programmiersprache ist in Planung. DIN 61131-3 enthält bereits eine Petri-Netz-ähnliche Ablaufsprache (AS) für die Ablaufsteuerung von SPS-Programmen. Werden die gesteuerten SPS-Programme auf einzelne Sensoren und Aktoren beschränkt, können Petri-Netze mit der AS normkonform dargestellt werden. Die Beschreibung vollständiger SPS-Programme ist aber in dieser Sprache nicht problemlos möglich. In einigen SPS-Programmiersystemen kommen bereits Petri-Netze zum Einsatz wie z.B. in SIMPE/CONPE, GRAPH5 /JANZEN 90/, PEMOM /NOCHE 91/ und SystemSpecs /SCHNEIDER 95/. Aufgrund der in beschriebenen Vorteile (Abschnitt 4.1.2), werden sie auch hier als Repräsentationsform vorgeschlagen.

Für die Modellierung der Netze wird von OSIMPROST ein geeigneter Bausteinkasten bereitgestellt, der Transitionen, Stellen, Kanten und Marken enthält. Wegen der Übertragbarkeit auf die anderen SPS-Programmiersprachen ist diese Festlegung aber nicht weiter bindend. Es können ebenso Bausteinkästen für KOP, FUP, AWL und ST erstellt werden. Durch die Repräsentation der Steuerungen als Objekte, bei denen die innere Struktur nach außen hin verborgen bleibt, können in verschiedenen Steuerungsblöcken unterschiedliche Darstellungsformen nebeneinander eingesetzt werden. Die Wahl der Beschreibungssprache bleibt dann dem Anwender überlassen, der je nach persönlichen Vorstellungen und Aufgabe einen Bausteinkasten mit Steuerungssymbolen auswählen kann.

#### 5.2.2.3 Interaktion zwischen Steuerungs- und Prozeßbausteinen

In Steuerungen werden in dieser Arbeit für externe Verknüpfungen „Eingang“ und „Ausgang“, für direkt zugeordnete Adressen „Sensor“, „Aktor“ und „Merker“ verwendet. Um die Programmierung zu erleichtern, das Programmverständnis zu erhöhen und die Programmwartung zu erleichtern, werden die Adressenbezeichnungen im Klartext vorgenommen, und der Anwender wird vom Umgang mit schlecht einprägsamen Speicheradressen entlastet. Allen Ein- und Ausgängen können vom Programmierer Namen gegeben werden, die nur innerhalb einer Steuerung eindeutig sein müssen. Externe Adressen werden, zusätzlich zu ihrem Namen, über ihren Pfad spezifiziert. In jeder Steuerung existieren 5 Tabellen, die folgenden Inhalt haben.

| **Eingänge** | | |
|---|---|---|
| Pfad der Variablen | Wertebereich | Zustand |
| Leitrechner.Steuerung_X.Priorität_Einschleusen | Boolean | True |
| | | |

Tabelle 7: Eingänge von Steuerungsblöcken

| Ausgänge | | |
|---|---|---|
| Interner Name | Wertebereich | Zustand |
| Bereit_für_Bearbeitung | Boolean | False |

Tabelle 8: Ausgänge von Steuerungsblöcken

| Sensoren | | | |
|---|---|---|---|
| Interner Name | Speicheradresse | Wertebereich | Zustand |
| WT_in_Bearbeitungsposition | X11 | Boolean | False |

Tabelle 9: Sensoren

| Aktoren | | | |
|---|---|---|---|
| Interner Name | Speicheradresse | Wertebereich | Zustand |
| Vereinzeler1 | Y8 | Boolean | False |

Tabelle 10: Aktoren

| Zuordnungen | | |
|---|---|---|
| Name | Station | Steuerung |
| SPS_Spannstation1 | Spannstation1 | Bereichssteuerung 1 |

Tabelle 11: Zuordnungen für den Steuerungsblock

Diese Tabellen gehören als Formblätter zu jedem Steuerungsblock und müssen vom Programmierer ausgefüllt werden. Hier sind jeweils 1 oder 2 Beispiele eingetragen. Einige Inhalte ergeben sich automatisch als Ergebnisse ihrer Verknüpfung und sind daher für die Bearbeitung gesperrt (unterlegt dargestellt).

Die meisten modernen Steuerungen sind Abbildsteuerungen[24]. Ein- und Ausgangsspeicher werden hier durch die entsprechenden Tabellen repräsentiert. Anhand der Tabellen kann der Anlagenzustand überwacht werden.

Die Tabellen enthalten auch die Informationen, die für die Übertragung auf die reale Anlage erforderlich sind. Das Feld „Steuerung“ in „Zuordnungen“ weist das Programm einer Steuerungs-CPU zu, in deren Speicher das Programm übertragen wird. Es weist den Postprozessor an, den Steuerungscode in dem entsprechenden Programmblock abzulegen.

Das Feld „Station“ der „Zuordnungen“ hat nur für die Simulation Bedeutung. Es verweist auf die zugehörige Station, damit nicht vor jedem Ein- und Ausgang der komplette Pfad zum zugeordneten Prozeß angegeben werden muß. In der Realen Anlage ist diese Verknüpfung durch die Verdrahtung der Hardware gegeben

[24] Abbildsteuerungen haben für Ein- und Ausgänge einen Zwischenspeicher. Bei der Zyklusbearbeitung wird im ersten Schritt der Zustand aller Eingänge gelesen und auf den Eingangsspeicher übertragen, dann wird der Zyklus mit den Programmschritten abgearbeitet und die Ergebnisse in den Ausgangsspeicher geschrieben. Erst nach dem Zyklusende werden die Ausgänge auf den Zustand der Ausgangsspeicher gebracht.

### 5.2.3 Darstellungskomponenten

Einige Modellkomponenten haben nur innerhalb der Simulation Bedeutung. Jedes Objekt, das aus einer Klasse abgeleitet wird, muß einen eindeutigen *Namen* haben. Dieser kann für reale Objekte als Vorschlag aus dem Modell übernommen werden.

Jedes Objekt wird grafisch dargestellt. Dafür muß ihm ein *Icon* und eine geometrische Position auf dem Bildschirm zugewiesen werden, über die es repräsentiert wird. Um die Attribute eines Objekts ändern zu können, müssen sie dem Anwender zugänglich gemacht werden. Er erhält die Möglichkeit, ein durch ein Icon repräsentiertes Objekt zu *öffnen* und über eine *Eingabemaske* zu manipulieren. Für Attribute, die grafisch repräsentiert sind, muß ein Grafik-Editor vorgesehen werden, z.B. um Icons zu entwerfen, Kanten zu editieren oder um Objekte in Modellen oder auf Strecken zu verschieben.

### 5.2.4 Zeit und Ablaufsteuerung

Zu einem Simulationsprogramm gehört eine Zeit- und Ablaufsteuerung. Sie startet und beendet die Simulation und verwaltet die Ereignisliste. In OSIMPROST wird eine *prozeßorientierte* Zeitverwaltung gewählt (Abschnitt 3.2.4). Prozeßorientierte Zeitsteuerungen bieten eine maximale Simulationsgeschwindigkeit, weil die Uhr Zeiträume ohne relevante Veränderungen überspringen kann. Auch die Handhabung interagierender Prozesse ist problemlos zu handhaben. Auf die kontinuierliche Beschreibung, wie sie das Multimodelling (Abschnitt 3.2.5) ermöglicht, muß vorerst verzichtet werden, weil bisherige Rechner die Handhabung dieses Verfahrens nicht in vertretbarer Geschwindigkeit beherrschen. Der folgende Abschnitt zeigt eine Möglichkeit auf, wie auch mit einer prozeßgesteuerten Simulation das kontinuierliche Verhalten speicherprogrammierbarer Steuerungen nachgebildet werden kann.

#### 5.2.4.1 Ein Ansatz zur Darstellung von SPS-Zyklen bei ereignisorientierter Simulation

SPS arbeiten ihre Programme zyklisch ab, um eine *kontinuierliche* Überwachung des Systemzustandes nachzubilden, wie sie bei Schützsteuerungen zu finden ist. Bei mittleren Anlagen ergeben sich, je nach der Anzahl der Befehle im Programm, Zykluszeiten in der Größenordnung zwischen 10 und 100 *ms*. Längere Zykluszeiten würden eine zu lange Reaktionszeit auf Veränderungen im System bewirken und damit das Risiko bergen, daß auf Zustände nicht rechtzeitig reagiert wird, mit katastrophalen Folgen möglicherweise.

Bei einer prozeßgesteuerten Simulation steht diese kleine Größenordnung in einem krassen Gegensatz zu den Ereignisabständen im Prozeß, die in der Größenordnung von Sekunden bis Stunden liegen. Das Größenverhältnis kann somit 1/360.000 betragen. In der Realität ist das unproblematisch, weil die SPS ja die Echtzeitbedingungen erfüllt, und der Prozeß selbst nicht durch SPS-Programmzyklen beeinflußt wird. In der Simulation müssen z.B. bis zu einem Ereignis, das erst in einer Stunde eintritt, die 360.000 SPS-Zyklen bearbeitet werden, mit einem

hohen Rechenaufwand und damit Zeitbedarf, der die schnelle Gewinnung von Simulationsergebnissen unmöglich macht. Dieses Problem ist auch durch eine zeitschrittorientierte Simulationssteuerung nicht zu lösen, da auch diese, wegen der vielen Abfragen in jedem Zyklus, langsam ist. Bei der sonst effizienten Ereignissteuerung wird jeder SPS-Zyklus als Ereignis aufgefaßt.

Wenn jedoch keine Veränderungen auf Prozeßebene passieren, führt die Bearbeitung der SPS-Zyklen nicht zu Veränderungen. Hier liegt das Potential, die kombinierte Simulation schnell zu machen. Der Zyklus muß also nur bearbeitet werden, wenn sich Veränderungen im System ergeben haben, die SPS wird immer nur von Ereignissen aktiviert. Diese Steuerung bildet die Realität gut nach, und verhindert, daß sehr kurzzeitig bestehende Systemzustände durch das Netz der Abfragezyklen schlüpfen können.

Es ergeben sich durch diese Methode aber Abweichungen zum realen System, deren Bedeutung untersucht werden muß. Zum einen wird durch die Bearbeitung der Zyklen *genau* zu den Zeitpunkten der Ereignisse eine SPS mit einer Zykluszeit von 0 *s* simuliert - in der Praxis nicht realisierbar. Da die Abfragefrequenz von SPS unabhängig vom Prozeß läuft, und die Ergebnisse erst nach der Zykluszeit auf die Ausgabeadressen übertragen werden, beträgt die Verzögerung mindestens eine, höchstens zwei Zykluszeiten ($T_{Zyklus} \leq t_{Delay} < 2*T_{Zyklus}$). Eine Worst-Case-Untersuchung läßt sich durchführen, wenn die SPS-Programmbearbeitung erst zwei Zykluszeiten nach dem Ereignis durchgeführt wird.

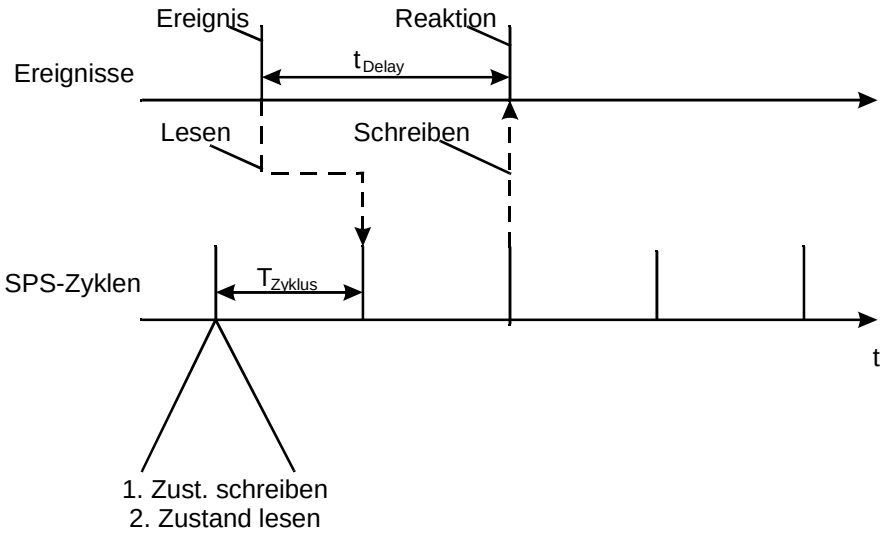


Abbildung 21: Reaktionszeiten bei SPS

Eine andere Ungenauigkeit ergibt sich aus dem Umgang mit Merkern und Ausgangsabfragen in der SPS. Wird ein Ausgang oder Merker verändert, werden die Auswirkungen entweder in Programmschritten, die im Zyklus auf die Änderung folgen wirksam, oder erst im nächsten *Zyklus*. Im letzteren Fall bleibt die Änderung in der Simulation bis zum nächsten *Ereignis* unbearbeitet, wenn der Zyklus nur einmal bearbeitet wird.

Es ist also eine mehrmalige Abfrage nötig, wobei die Anzahl der Wiederholungen von der Verschachtelung des Programms abhängt. Es müssen zu ihrer Bestimmung entweder die Steue-

rungscodes analysiert werden, oder es muß ein ausreichend hoher Defaultwert von Wiederholungen gewählt werden, was unter Inkaufnahme von Geschwindigkeitseinbußen im Programm geschieht.

Ein eleganterer Weg, der mit dem Konzept der Ereignissteuerung besser harmoniert, ist die Veränderung von Ausgängen und Merkern als Ereignis, das zum Zyklusende stattfindet, aufzufassen. Die Steuerung wird so oft bearbeitet, bis sich keine Änderungen an Ausgängen mehr ergeben. Erst dann wird zum nächsten Ereignis weitergegangen. Dieses Vorgehen ist in der Lage, das Verhalten des SPS-Programms richtig nachzubilden.

### 5.2.5 Durchführung des Simulationsexperiments

In Abschnitt 5.1.1.3 wurde beschrieben, wie das Simulationsmodell mit dem Zustand der Fertigung in Übereinstimmung gebracht werden kann. Ausgangssituation eines Simulationsexperiments ist die Festlegung des Versuchsaufbaus (Prozeßbeschreibung) und die Beschreibung seiner veränderlichen Komponenten (Material und Datenbestand). Es gibt zwei Möglichkeiten, einem Startzustand reproduzierbar zu machen. Zum einen kann der dynamische Zustand in ein Modell geladen, und so der Startzustand bestimmt werden. Das ist jedoch problematisch, sobald Veränderung am Modell vorgenommen werden, weil bei diesem Verfahren die veränderlichen Daten hundertprozentig mit dem Modell übereinstimmen und kompatibel sein müssen. Die andere Möglichkeit speichert das gesamte Modell zusammen mit seinen beweglichen Elementen und Variablen ab. Das benötigt zwar mehr Speicherplatz, dafür ist aber stets eine eindeutige, reproduzierbare und lauffähige Startsituation vom Datenspeicher ladbar. Zu Dokumentationszwecken kann das komplette Modell nach der Versuchsdurchführung abgelegt werden. In der Experimentierphase werden zahlreiche Experimente mit verschiedenen Strategien durchgeführt. Diese können alle von einem Modell mit gespeicherter Startsituation durch Manipulation erzeugt werden. Wenn alle Versuchsmodelle auf Datenträgern gesichert werden, kann nach der Entscheidung für die optimale Strategie dieses wieder geladen und für Dokumentationszwecke, Demonstrationszwecke und die Steuerungscodegenerierung verwendet werden. Die objektorientierte Simulation erlaubt die kompakte Speicherung von Modellen und ihre komfortable Veränderung durch Vererbungsmechanismen. Deshalb ergeben sich aus der Speicherung der gesamten Modelle keine Probleme.

### 5.2.6 Übertragung der SPS-Programme auf die Steuerungen

In einem als optimal ausgewählten Modell liegt die gesamte Logik für die Steuerung der Bausteine vor. In Abschnitt 5.1.1.5.1 auf S. 54 wurden drei Möglichkeiten zur Übertragung des Programms auf die Steuerung beschrieben. Für die automatische Übertragung der SPS-Programme auf die Steuerungen muß das Simulationssystem über einen *Postprozessor* verfügen, der ihre Übertragung auf die Steuerung übernimmt. Dieser muß die relevanten Informationen aus dem Modell lesen und sie in ein steuerungskompatibles Format übertragen. Anschließend muß der fertige SPS-Code über ein Netzwerk oder andere Datenträger zur Steuerung

überspielt werden. Diese Vorgänge sind vom gewählten Protokoll abhängig. Der Postprozessor ist für dieses zu konfigurieren.

Es gibt Einsatzbereiche, in denen auf den Steuerungen die Programme während des Betriebs ausgetauscht werden, wenn ein automatischer Produktwechsel stattfindet /MMS 93/. Hierfür werden die Programme vom Simulator an einen Fertigungsleitrechner übertragen, der ihre weitere Verwaltung und Handhabung übernimmt.

## 5.3 Ein einfaches Simulationsbeispiel in Simple++

Begleitend zu dieser Arbeit wurde das Simulationsprogramm SIMPLE++ der Firma AESOP eingesetzt. Ein Modell, das die Möglichkeiten zur Abbildung von Steuerungsfunktionen auf dem Simulator untersuchen sollte, stellte den Anfang dieser Arbeit dar. Mit dem Fortschreiten der Arbeit wurden die erarbeiteten Konzepte auf das Simulationsmodell übertragen, so daß dieses am Ende den erreichten Stand des Projekts OSIMPROST präsentiert. Obwohl nicht alle Komponenten von OSIMPROST in SIMPLE++ umgesetzt werden konnten, ging von den Simulationsexperimenten eine Vielzahl von Anregungen aus, die das Fortschreiten der Arbeit befruchteten.

### 5.3.1 Beschreibung von SIMPLE++

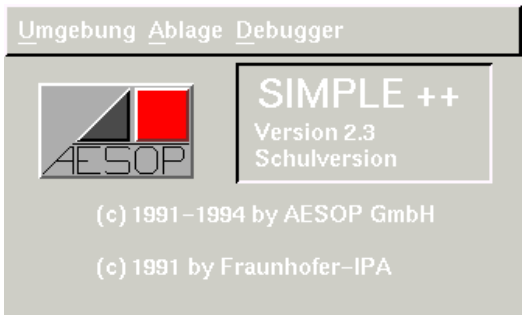


Abbildung 22: Hauptfenster von SIMPLE++

SIMPLE++ (Simulation in Produktion, Logistik und Engineering; in C++ implementiert) ist ein Simulationsprogramm zur Materialflußanalyse. Mit ihm kann der Materialfluß auf einer abstrakten Ebene dargestellt werden. Die abstrakte Darstellung, kombiniert mit einem hierarchischen Modellaufbau, ermöglicht schnelle Simulationsstudien, mit der Möglichkeit, Modelle parallel zum Projektfortschritt zu verfeinern, oder sie für schnelle Analysen zu vergröbern. Der Einsatzschwerpunkt des Simulationsprogramms ist die Analyse und Optimierung des Materialflusses durch seine diskrete Beschreibung, ohne Beschränkung auf bestimmte Fachgebiete.

Das Konzept von SIMPLE++ ist durchgehend objektorientiert in Implementierung und Oberfläche. Die Klassen, die dem Anwender zur Beschreibung seines Modells zur Verfügung stehen, sind die Basis der Modellierungskomponente des Programms (Abbildung 10). Mit der Konzeption dieser Klassenhierarchie schuf Bernd-Dietmar Becker 1991 die Grundlage zur Beschreibung von Fertigungskomponenten. Kombiniert mit einer prozeßorientierten Zeitsteue-

rung (siehe Kap. 5.2.4.1) und einer grafischen Benutzeroberfläche, die die Objektfenster der grafischen Oberfläche unter einem UNIX-Betriebssystem nutzt, ermöglicht das Programm einfache und flexible Simulationsstudien. Im Fachgebiet Fertigungseinrichtungen wird SIMPLE++ in einer PC-Version für SCO-Unix betrieben.

Zur Beschreibung komplexer Komponenten, die nicht im Standardbausteinkasten angeboten werden, kann der Anwender Modelle erstellen, und diese als Klassen dem *Bausteinkasten* hinzufügen. Auf diese Weise entsteht ein anwenderspezifischer Bausteinkasten, aus dem hierarchische Modelle erstellt werden können. Die selbstdefinierten Bausteine sind aufgrund ihres objektbasierten Konzepts sehr gut wiederzuverwenden. Zu diesem Zweck können die Bausteine abgespeichert, und in anderen Simulationsprojekten wiederverwendet werden. Der Umstand, daß bei den Anwendern umfangreiche Bibliotheken entstehen wird genutzt. Die Firma AESOP bietet einige von ihnen als Spezialbausteinkästen für spezifische Anwendungen an. Es sind bereits Bausteinkästen für Logiksteuerung, Werkstattsteuerung, führerlose Transportsysteme, Elektrohängebahnen und andere verfügbar. Detaillgetreue Simulationsstudien sind mit diesen schnell durchzuführen. Abbildung 23 zeigt den Grundbausteinkasten von SIMPLE++.

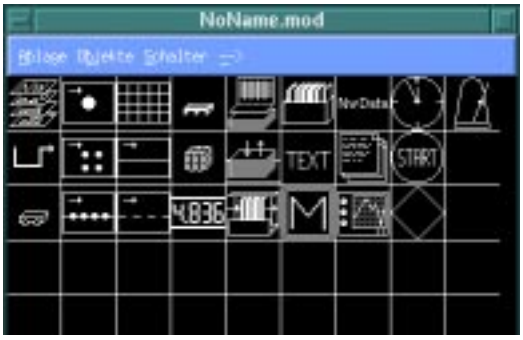


Abbildung 23: Grundbausteinkasten von SIMPLE++

Einige Zusammenhänge sind durch eine grafische Programmierung mit den vorhandenen Bausteinen nicht darzustellen. Für solche Aufgaben können *Methodenbausteine* in SIMPLE-Modelle eingesetzt werden. Diese enthalten Programmtext in der systemeigenen objektorientierten Programmiersprache SimTalk, mit deren Befehlen alle Materialfluß- und Informationsflußkomponenten individuell manipuliert werden können. Auch Programmschleifen, boolsche Verknüpfungen, mathematische Operationen und die Steuerung des Simulationsexperiments sind mit dieser Programmiersprache möglich. Die Methodenbausteine werden durch spezifizierte Ereignisse gestartet, können sich aber auch gegenseitig aktivieren, wodurch die Sprache modular und hierarchisch einzusetzen ist.

Wie in Kapitel 5.1 beschrieben, spielt der Datentransfer zwischen der Simulation und anderen Anwendungen eine große Rolle in der Simulation. SIMPLE stellt hierzu eine Reihe von Verfahren zur Verfügung.

- Tabelleninhalte, die als ASCII-Datei vorliegen, können von SIMPLE ins Modell geladen oder aus dem Modell als ASCII-Datei gespeichert werden.

- Es können *während* der Simulation in SimTalk geschriebene Programme aus Dateien geladen werden.
- Der Befehl „System“ ermöglicht die Ausgabe von Befehlen an das Betriebssystem, womit beliebige externe Programme auszuführen sind. Über Dateien können diese Daten an SIMPLE zurückliefern.
- Ab Version 3.0 steht eine Beschreibungssprache zur Verfügung, mit der komplette Modelle von anderen Softwareprogrammen innerhalb eines CIM-Verbundes an SIMPLE übergeben werden können. Diese steuert ebenfalls die Durchführung und Datenausgabe der Simulation.
- Optional sind Produkte erhältlich, die eine C-Schnittstelle, genetische Algorithmen, ein Mailboxsystem und eine SQL Datenbankschnittstelle zur Verfügung stellen.

Im Bereich Kommunikation liegt eine große Entwicklungschance für die Simulation, so daß die Möglichkeiten durch ständig neue Applikationen ergänzt werden.

Insgesamt ist SIMPLE++ durch die Vielfältigkeit seiner Möglichkeiten auch für Forschungszwecke sehr gut geeignet.

#### 5.3.1.1 Warum SIMPLE++?

Es gibt auf dem Markt eine Vielzahl von Simulationsprogrammen. Unter diesen das beweisbar optimale für ein Projekt herauszufinden, ist sicher nicht möglich, weil alle Anforderungen, die sich aus einem Projekt ergeben, in der Regel nicht vorher bekannt sind. Zum anderen unterliegt der Markt einem schnellen Wandel. Bei der Kaufentscheidung für ein Programm ist man meistens auf Herstellerangaben und Demos angewiesen, während konzeptionelle Schwächen in der Regel erst nach einiger Zeit im Gebrauch deutlich werden. Für die Entscheidung, in diesem Projekt SIMPLE++ einzusetzen, gibt es mehrere Gründe:

- Es können Bausteinkästen erzeugt, erweitert und verwaltet werden.
- Es können hierarchische Modelle definiert werden.
- Das Programm verfügt über eine vollständig grafikorientierte integrierte Fensteroberfläche.
- SIMPLE++ ist durchgehend objektorientiert implementiert, wodurch der vollständig objektorientierte Ansatz dieser Arbeit untersucht werden kann.
- Über eine Dateischnittstelle können SPS-Programme auf die Betriebssystemebene übertragen und für Steuerungen nutzbar gemacht werden.
- Es steht eine umfangreiche und komfortable Programmiersprache zur Verfügung.
- In Fachgebiet Fertigungseinrichtungen besteht eine langjährige Zusammenarbeit mit der Firma AESOP.
- In der Bundesrepublik ist SIMPLE++ Marktführer für Materialfluß-Simulationssysteme, wodurch die Ergebnisse der Arbeit für einen größeren Personenkreis nutzbar sind.

Die hier als relevant genannten Punkte werden durch kein anderes Programm auf dem Markt erfüllt. Das im Fachgebiet vorhandene Programm kann also als solide Grundlage für Untersuchungen in dieser Arbeit dienen.

#### 5.3.1.2 Zeitsteuerung

SIMPLE verfügt bei den vordefinierten Materialflußbausteinen über eine prozeßorientierte Zeitsteuerung. Bei selbstdefinierten Methoden kommt eine Ereignissteuerung zum Einsatz, bei der der Anwender selbst programmieren muß, was passieren soll, wenn nicht ausführbare Ereignisse nicht einfach storniert werden dürfen. Der „Ereignisverwalter“ ist Bestandteil jedes Hauptmodells[25]. Er zeigt die aktuelle Zeit und die Simulationsdauer an. Er ermöglicht dem Anwender das Ende der Simulation auf einen bestimmten Zeitpunkt festzulegen, oder das durch den Button[26] „Stop“ selbst zu tun. Der Button „Start“ beginnt einen Simulationslauf mit dem aktuellen Modellzustand. Durch „Reset“ wird das Modell in die Ausgangsstellung versetzt. Es werden alle laufenden Prozesse abgebrochen, die Simulationsdauer und Auswertungsstatistiken werden zurückgesetzt und alle benutzerdefinierten Methoden im Modell, die den Namen “Reset“ tragen, werden ausgeführt. „Init“ startet alle gleichnamigen Methoden im Modell, die den Anfangszustand der Simulation bestimmen. Die SIMPLE-Zeitsteuerung harmoniert mit dem in Kapitel 5.2.4.1 dargestellten Ansatz zur effizienten Zeitsteuerung für die gleichzeitige Simulation prozeßorientiert dargestellter Prozesse und quasikontinuierlicher Steuerungen.


Abbildung 24: Ereignisverwalter + Icon

[25] Das Hauptmodell ist das in einer Modellhierarchie am höchsten stehende.

[26] Eine interaktive Schaltfläche auf dem Monitor.

### 5.3.2 Modellierung des Beispiels

Die Modellierung basiert auf der Definition von Klassen, mit deren Hilfe sich die Bausteine des Transfersystems, inklusive der Steueralgorithmen, darstellen lassen. Die Objekte sind in den Kapiteln 4, 4.2 und 5 ausführlich beschrieben, so daß an dieser Stelle auf diese verwiesen werden kann. Hier soll die Übertragung der Elemente auf *SIMPLE++* im Vordergrund stehen.

### 5.3.3 Die Elemente des Prozesses

#### 5.3.3.1 Die Strecke

In Abschnitt 4.2.1.1 wird die Strecke als Baustein beschrieben, der WTs transportiert und die Einbauten aufnimmt. Durch diesen Baustein wird ein Ereignis ausgelöst, wenn ein WT in den Bereich eines Sensors eintritt, ihn verläßt, oder ein WT auf ein Hindernis läuft. Die Strecke enthält eine Liste der Einbauten. Die Positionen der Einbauten ist in diesen selbst gespeichert. Diese Darstellungsweise ist mit SIMPLE nicht möglich, weil der definierte Baustein „Einzelstrecke“, und auch kein anderer, in der Lage ist, ereignisauslösende Elemente an einer bestimmten Position aufzunehmen. Dadurch kann ein Aspekt des OSIMPROST-Projekts nicht verwirklicht werden. Als Ersatz wird die Strecke in Einzelstücke zerlegt, welche die Abstände zwischen dem Einbauten ausfüllen. Nachteile entstehen, wenn Einbauten, die WTs an verschiedenen Stellen stoppen oder registrieren, verwendet werden, wie beispielsweise die Spannstation. Die Sensoren und Aktoren, die außerhalb der Einbauposition liegen, wie z.B. bei der Spannstation das Element des Vorvereinzelers (Y1), müssen manuell an der richtigen Position vor dem Einbau eingesetzt werden. Der Einbau erhält statt des entsprechenden Aktors einen Verweis, der den richtigen Aktor auf der Strecke anspricht. Der Transport von WTs und ihr Stauverhalten wird von der Klasse „Einzelstrecke“ in SIMPLE automatisch richtig abgebildet. Zur Simulation von Störungen und Abschaltungen einer Strecke müssen alle Teilabschnitte der jeweiligen Strecke gestört werden. Trotz der veränderten Darstellungsweise der Strecke in SIMPLE, wird das Materialflußverhalten korrekt abgebildet.


Abbildung 25: Einzelstrecke + Icon

### 5.3.3.2 Der Positionsschalter

Der Positionsschalter liefert ein positives Signal, wenn sich ein WT direkt auf der Einbauposition ± 5 mm des Schalters befindet. Dieses Wegstück wird durch eine Instanz der Klasse „Einzelstrecke“ mit einer Länge von 10 mm und einer Transportgeschwindigkeit dargestellt, die der des aktuellen Streckenabschnittes entspricht. Um in der ereignisgesteuerten Darstellungsform die Auswirkungen der Ereignisse „Eintritt“ und „Austritt“ eines WTs in den Bereich zu untersuchen, wird jeweils die Methode „Ereignis“ aufgerufen, die dem Methodenbaustein „SPS“ im Hauptmodell (siehe unten) aktiviert. Abgefragt wird der Zustand des Sensors über eine Methode, die einen Rückgabewert vom Typ Boolean liefert. Diese Methode ist, so wie im gesamten Modell einheitlich für Ein- und Ausgangsmethoden, mit „F“ für Funktion benannt.

`Boolean:Sensor.F`

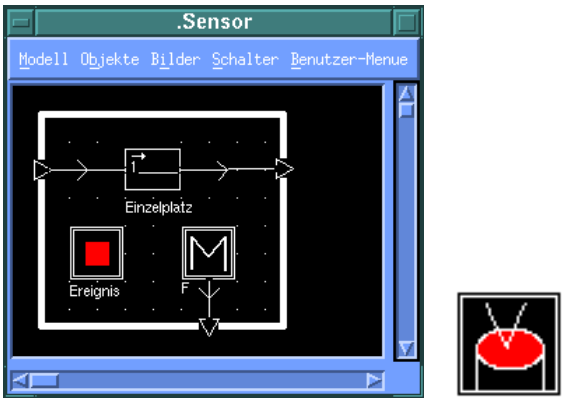


Abbildung 26: Sensor + Icon

### 5.3.3.3 Der Vereinzeler

Der Vereinzeler hat die Aufgabe, einen WT an einer definierten Position anzuhalten, wenn sein Eingangssignal True ist. Er wird durch eine Einzelstrecke der Länge 0 dargestellt. SIMPLE berücksichtigt bei Instanzen der Klasse Einzelstrecke automatisch, daß kein WT der Länge 120 mm in diesen Streckenabschnitt paßt, und der WT vollständig nach hinten in den vorhergehenden Streckenabschnitt übersteht. Ist der Vereinzeler ausgefahren, ist die 0 mm lange Wegstrekke „gestört“, und kann keine Eintritts- und Austrittsereignisse erzeugen. Der WT wird dadurch gestoppt. Die Methode „F“ mit einem Eingangsparameter des Typs Boolean „stört“, bzw. „entstört“ den Streckenabschnitt. Betätigt wird der Vereinzeler durch folgende Message:

`Vereinzeler.F(True)` zum Öffnen und

`Vereinzeler.F(False)` zum Schließen.

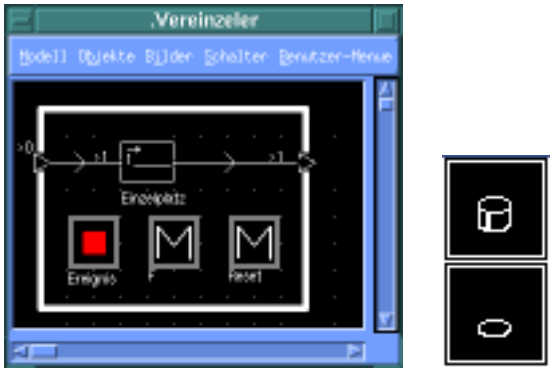


Abbildung 27: Vereinzeler + Icon für geschlossen und offen

Der Vereinzeler speichert Zustände. Um diese zurückzusetzen, erhält er eine Methode „Reset“. Diese wird immer aufgerufen, wenn im Ereignisverwalter „Reset“ betätigt wird. Sie setzt den Zustand der Strecke auf „gestört“, weil Blockieren das Grundverhalten des Bosch-Vereinzelers ist, und aktualisiert die Bildschirmdarstellung entsprechend auf „geschlossen“.

### 5.3.4 Die Elemente zur Modellierung des Steuerverhaltens

Für die Darstellung der Steuerungslogik wurden Petri-Netze gewählt. Den Grundbausteinen des Bausteinkastens TS0 sind Petri-Netze zugeordnet, welche in Abschnitt 3.6, bzw. im Anhang 7.1 dargestellt sind.

#### 5.3.4.1 Petri-Netze

Petri-Netze bestehen aus *Stellen* und *Transitionen*, die mit *Kanten* verbunden sind, und *Token*, die Stellen markieren. Die logische Verbindung „Kante“ wird in SIMPLE als Klasse im Standardbausteinkasten zur Verfügung gestellt. Für die anderen Elemente werden Klassen definiert, die eine Modellierung aus dem Bausteinkasten ermöglichen.

##### 5.3.4.1.1 Stellen

Die Stelle ist bei den verwendeten Bedingungs-/Ereignis-Netzen ein passives Materialflußelement, das genau einen Token aufnehmen kann. Dieses Verhalten findet in SIMPLE seine Entsprechung in der Klasse „Lager“ mit einem Fach. Für das Entfernen von Token steht die Methode „Rst“ im Baustein „Stelle“ zur Verfügung, für das Setzen „Set“. Ob die Stelle frei, und damit in der Lage ist, einen Token aufzunehmen, wird durch die Methode „TestEin“ ermittelt, die einen Rückgabewert des Typs Boolean liefert. „TestAus“ liefert den Booleanwert, der anzeigt, ob ein Token entnommen werden kann. „Reset“ leert die Stelle und in „Init“ kann festgelegt werden, ob bei der Initialisierung des Modells ein Token auf die Stelle gesetzt werden soll.

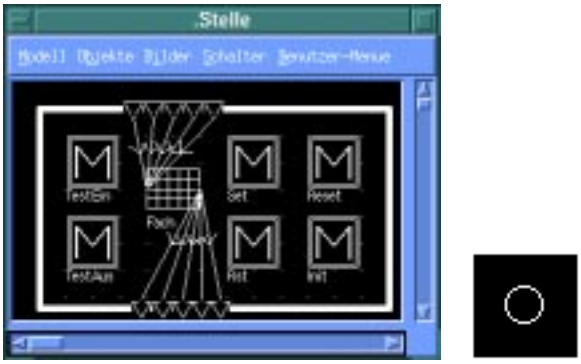


Abbildung 28: Stelle + Icon

#### 5.3.4.1.2 Transitionen

Die Transition überprüft eine Anzahl von Bedingungen auf ihre Erfüllung und löst ein Ereignis aus, falls alle Bedingungen wahr sind. Sie entspricht damit einem Schritt einer Anweisungsliste, bzw. einem Zweig im Kontaktplan.

Der Zyklusschritt einer Transition wird mit der Methode „Trans“ ausgelöst. Diese Methode ist hierarchisch aufgebaut und ruft als erste die Methode „Test“ auf. Diese führt „Vorg“ aus, welche überprüft, ob die „TestAus“ Methoden aller Vorgängerstellen True liefern. In diesem Falle liefert sie selbst True zurück. Dann ruft „Test“ „Nachf“ auf, die die „TestEin“ Methoden aller Nachfolgestellen abfragt. Wenn diese ihrerseits ausnahmslos True zurückliefern, antwortet „Nachf“ ebenfalls mit True. Falls „Vorg“ und „Nachf“ die Schaltbedingungen bestätigen (alle Vorgängerstellen belegt, alle Nachfolger frei), setzt „Test“ das Transitionsattribut „Aktiv“ auf True und erlaubt damit der Transition das Schalten.

Die Methode „Schalten“ löscht alle Token auf Vorgängerstellen mit „Stelle$_{[\text{Liste aller Vorgänger}]}$.Rst“ und erzeugt einen Token auf allen Nachfolgestellen durch „Stelle$_{[\text{Liste aller Nachfolger}]}$.Set“.

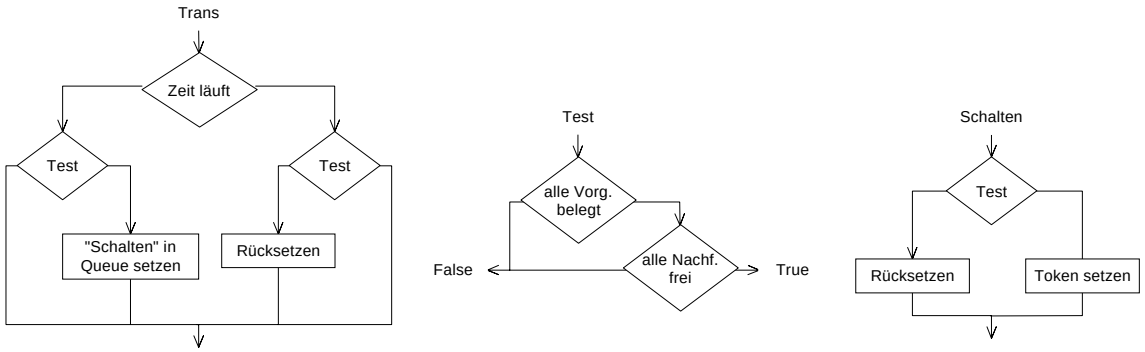


Abbildung 29: Ablauf der Methoden „Trans“, „Test“ und „Schalten“ des Bausteins Transition

Um Timer modellieren zu können, werden Transitionen mit einer Schaltverzögerung versehen. Das Attribut „Zeit“ wird auf einen Wert größer 0 gesetzt wird. In diesem Fall wird die Methode „Schalten“ in die Queue des Ereignisverwalters gesetzt, und erst nach der angegebenen Zeit aktiv. Die Transition wird weiterhin in jedem SPS-Zyklus überprüft. Verliert sie ihre Konzession zum Schalten, weil ein Vorgängertoken entfernt, oder eine Nachfolgestelle besetzt wird, findet ein Rücksetzen des Timers statt, wodurch das Ereignis „Schalten“ storniert wird. Da in

SIMPLE keine Ereignisse aus der Queue des Ereignisverwalters entfernt werden können, wird ein anderer Weg zum Stornieren gegangen. Das Ereignis „Schalten“ erhält eine jeweils eindeutige Nummer, wenn es in die Queue eingetragen wird. Die Nummer wird im Attribut „Zeit_NR“ hinterlegt. Entfällt die Schalterlaubnis, während die Zeit läuft, wird die „SchaltNR“ um einen Wert höher gesetzt. Die Methode „Schalten“ vergleicht den ihr mitgegebenen Wertes mit dem in „SchaltNR“ hinterlegten, und wird nur bei Übereinstimmung aktiv. Mit diesem „Trick“ wird in OSIMPROST der Reset eines Timers durchgeführt.

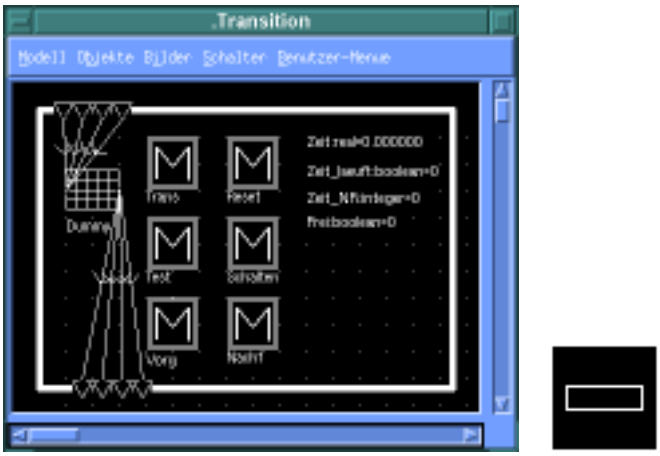


Abbildung 30: Transition + Icon

#### 5.3.4.1.3 Token

Der Token ist ein passives bewegliches Materialflußelement, das als Markierung im Fach einer Stelle erzeugt, bzw. gelöscht wird.

Abbildung 31: Icon des Tokens

#### 5.3.4.1.4 Sensoren

Um das Steuerverhalten einer SPS mit einem Prozeß zu verknüpfen, werden Sensoren an die Steuerung angeschlossen. Diese werden vom Steuerprogramm abgefragt. In SPS-Programmen aus Petri-Netzen werden Sensoren wie Stellen als Vorbedingung einer Transition abgefragt. Der Unterschied ist, daß nicht das Vorhandensein eines Tokens, sondern der Zustand eines Sensors das Ergebnis der Methode „TestAus“ bestimmt. Das Attribut „Verknuepfung“ enthält einen netzinternen Namen, der vom Objekt „Sensor“ abgefragt wird. Die eindeutige Verknüpfung zum Prozeß ergibt sich aus dem Pfad, der in der Variablen „Verknuepfung“ gespeichert ist, dem internen Namen und der Methodenbezeichnung „F“, die den Sensorzustand zurückliefert.

Im Gegensatz zu normalen Stellen, wird der Zustand eines Sensors von der Transition nicht durch Entfernen eines Tokens verändert, sondern ausschließlich vom Prozeß bestimmt. In Ab-

bildsteuerungen greift der Baustein „Sensor“ auf den Abbildspeicher des aktuellen Netzes, die Tabelle „Sensoren“, zu (s.u.).

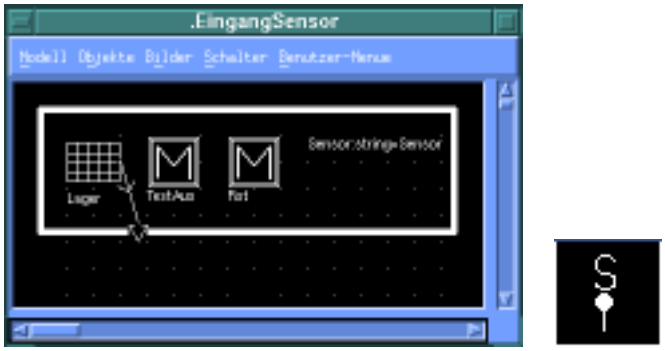


Abbildung 32: Sensor + Icon

#### 5.3.4.1.5 Aktoren

Steuerungen haben die Aufgabe, über Aktoren das Prozeßverhalten zu beeinflussen. Aktoren werden bei Petri-Netzen mit einer Stelle verknüpft und abhängig von ihrer Besetzung auf True oder False gesetzt. In der vorliegenden Modellierung sind die Methoden „Set“ und „Rst“ von Stellen um eine Befehlszeile erweitert, die den Aktor im Prozeß, der über das Attribut „Aktor“ und die Netzvariable „Verknuepfung“ zugeordnet ist, mit dem Belegungszustand der Stelle gleichsetzt. Die Ergebnisse werden in den Zwischenspeicher „Aktoren“ geschrieben (s.u.).

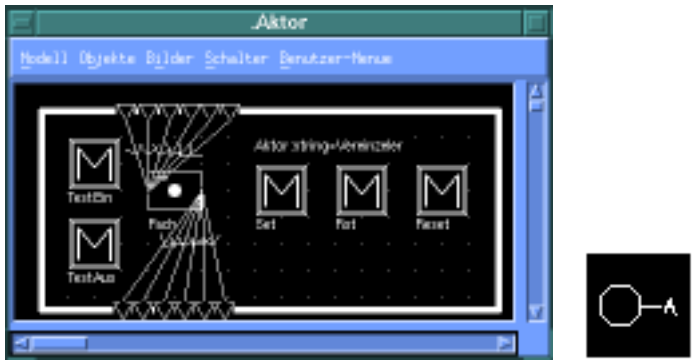


Abbildung 33: Aktor + Icon

#### 5.3.4.1.6 Der Baustein Steuerung

Der Baustein „Steuerung“ enthält jeweils ein Petri-Netz, das aus den zuvor beschriebenen Elementen besteht. Der SPS-Code des gesamten Modells ist aus diesen Modulen zusammengesetzt. Da hier Abbildsteuerungen, wie sie im nächsten Kapitel beschrieben sind, modelliert werden, enthalten die Steuerungsbausteine die Tabellen „Aktoren“ und „Sensoren“ als Abbildspeicher, sowie die Methoden „Lesen“ und „Schreiben“ zu deren Verwaltung.

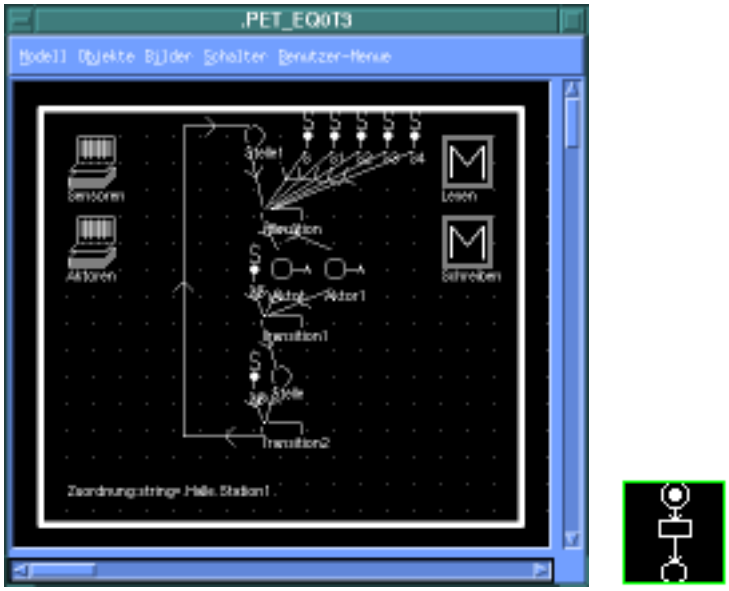


Abbildung 34: Der Steuerungsbaustein + Icon

### 5.3.4.2 Abbildsteuerungen und Verknüpfungen

Fast alle modernen Steuerungen sind Abbildsteuerungen. In OSIMPROST wird deshalb nur dieser Steuerungstyp nachgebildet. Das bedeutet, während ein Programmzyklus bearbeitet wird, werden keine Veränderungen von Eingängen berücksichtigt, und keine Veränderungen an Ausgängen der Steuerungen vorgenommen. Bevor die Methode „SPS“ im Hauptmodell die Methoden „Trans“ der Transitionen aller Netze im Modell anspricht, werden durch die Methode „Lesen“ alle Eingänge der Netze gelesen und in der Tabelle „Sensoren“ bzw. „Eingänge“ abgelegt, welche einen Aufbau gemäß Tabelle 7 und Tabelle 8 haben. Wenn alle Transitionen getestet sind und gegebenenfalls geschaltet haben, ist die Zyklusbearbeitung abgeschlossen, und die Zustände der Tabelle „Aktoren“ werden von der Methode „Schreiben“ auf die Aktoren des Prozesses übertragen. Da in der Realität zwischen Lesen und Schreiben eine SPS-Zykluszeit vergeht, werden die Befehle, die die Methode „Schreiben“ an die Aktoren ausgibt, erst mit einer entsprechenden Verzögerungszeit $T_{delay}$ aufgerufen. Über die Veränderung von $T_{delay}$ kann untersucht werden, wie sich verschiedene Zykluszeiten von Steuerungen auf den Prozeß auswirken, und ob das Überschreiten einer kritischen Zeit zu Betriebsstörungen führt. Der „Interne Name“, wie er in den Tabellen sowie den Sensoren und Aktoren des Netzes verwendet wird, ist alleine innerhalb des Modells nicht eindeutig und vollständig. Jede Steuerung ist einem Baustein zugeordnet, innerhalb dessen der Name genau eine Entsprechung hat. Um die Steuerung und ihre Ein- und Ausgänge eindeutig mit diesem Prozeßbaustein zu verknüpfen, wird dessen Pfad im Steuerungsattribut „Verknuepfung“ angegeben. Die Methoden „Lesen“ und „Schreiben“ berücksichtigen diese Variable, um die entsprechenden Zuordnungen vorzunehmen. Die Tatsache, daß nur das Zuordnungsattribut beim Einsetzen eines Steuerungsbausteins bearbeitet werden muß, erleichtert die Modellierung durch Ableitung von Klassenbausteinen, weil alle anderen Elemente eines Steuerungsbausteins geerbt werden können.

Die Aktivierung aller Methoden mit den Namen „Lesen“, „Trans“ und „Schreiben“, wie sie oben beschrieben sind, wird vom Hauptmodell aus durch die Methode „SPS“ vorgenommen. Diese wird von *jedem* Ereignis aufgerufen und führt drei Aktionen aus.

1. Die Methoden „Lesen“ aktualisieren die Eingangsspeicher der Steuerungen.
2. Die Methoden „Trans“ simulieren eine Zyklusbearbeitung in allen Steuerungen.
3. Die Methoden „Schreiben“ setzen die von den Steuerungen ausgelösten Ereignisse in die Queue des Ereignisverwalters.

Diese 3 Aktionen werden so lange im Abstand der Zykluszeit wiederholt, bis keine Änderungen von den Steuerungen mehr ausgehen. Erst dann wird zum nächsten Ereigniszeitpunkt in der Queue weitergegangen. Die Abfrage, ob zum aktuellen Zeitpunkt noch Veränderungen in den Steuerungen anstehen, wird durch das Attribut „Veränderungen“ realisiert. Alle Steuerungsrelevanten Ereignisse in Prozessen, sowie jedes Schalten von Transitionen, setzt dieses Attribut auf True. Nur wenn es auf True steht, wird „SPS“ aktiv, setzt es auf False zurück und führt seine 3 obengenannten Aktionen aus. Wenn diese es nicht durch ein Ereignis wieder auf True setzen, wird „SPS“ bis zum nächsten Ereignis beendet.

### 5.3.5 Der OSIMPROST-Bausteinkasten

Das Modellierungskonzept von SIMPLE sieht vor, eine Bausteinklasse, die als Icon im Bausteinkasten repräsentiert wird, zu markieren. Der Cursor wechselt auf eine Doppelkreuzdarstellung (✚) und signalisiert dem Benutzer dadurch, daß der markierte Baustein durch Mausklick an der entsprechenden Stelle in ein geöffnetes Modell eingesetzt werden kann. Die oben beschriebenen Bausteine sind durch ihre Icons im OSIMPROST-Bausteinkasten repräsentiert. Ein Doppelklick im Bausteinkasten öffnet die Klasse zur Bearbeitung in einem Fenster.


Abbildung 35: Der Bausteinkasten von OSIMPROST

Mit dem erarbeiteten Bausteinkasten hat der Anwender die Möglichkeit, Transfersysteme aus Bausteinen zusammenzusetzen, vorgefertigte Bausteinsteuerungen und selbstdefinierte Steuerungen aus Petri-Netzen zu modellieren, einzusetzen, und die Anlage mit Steuerung in der Simulation zu testen. Hiermit steht ein Tool zur Verfügung, das die Aufgabenstellung der vorliegenden Arbeit für einfache Fälle erfüllt. Durch die Definition wiederverwendbarer Klassen, für die grundlegenden Aufgaben von Transfersystemen, kann der Bausteinkasten für alle Bausteine

des Transfersystems TS0 erweitert werden. Aus den Petri-Netzen des Modells können manuell Steuerprogramme auf Programmiersystemen vorhandener SPS geschrieben werden. Das Konzept erlaubt auch den Ausblick auf die automatische Erzeugung von SPS-Code in Form von Anweisungslisten durch einen Postprozessor. Im Konzept ist das bereits vorgesehen, indem den Aktoren und Sensoren in den Tabellen eine Speicheradresse im SPS-Format zugeordnet werden kann. Zur automatischen Generierung erhält jede Transition eine Methode, die ihre Vor- und Nachbedingungen analog zur Methode „Test“ in Zeilen einer Anweisungsliste überträgt. SIMPLE ermöglicht diese als ASCII-Datei zu speichern. Über Diskette oder Netzwerk können diese auf Programmiergeräte übertragen werden.

### 5.3.6 Resümee der Modellierung

Das in SIMPLE realisierte Beispiel zeigt, daß durch die generischen Bausteine, wie sie in Abschnitt 4.2 beschrieben sind, ohne große Probleme ereignisgesteuerte objektorientierte Modellierungen möglich ist. Damit steht ein Konzept zur Verfügung, mit dem die kombinierte Simulation von SPS und Prozeß möglich ist. Dieses kann verwendet werden, um auf der Basis eines vorhandenen Simulationsprogramms, auf der Grundlage verfügbarer Toolkits in allgemeinen Programmiersprachen, wie z.B. C++SIM /C++SIM 94/ oder SIMPACK /FISHWICK 95//FISHWICK/, oder durch vollständige Neuprogrammierung eine Anwendersoftware zu schreiben. Durch die Verwendung offener Systeme, deren Programmcode zugänglich ist, ist auch das Problem, Strecken hierarchisch darzustellen lösbar, was in SIMPLE durch das Fehlen einer geeigneten Klasse verhindert wird.

Die Simulationsgeschwindigkeit im OSIMPROST-Modell zeigte sich deutlich geringer, als bei Modellen aus den Materialfluß-Grundbausteinen von SIMPLE. Das ergibt sich zwangsläufig aus der höheren Modellkomplexität. Zur statistischen Untersuchung des Materialflußverhaltens muß eine sehr leistungsfähige Hardware und eine geschwindigkeitsoptimierte Software zur Verfügung stehen. Ein anderer Weg der Beschleunigung ist das Abschalten der Steuerungen und eine Vergröberung des Modells, wie sie in SIMPLE möglich ist. Auch auf andere Weise ist die Simulation der Steuerungen bei statistischen Untersuchungen am Modell zu vermeiden. Die Detaillierung bis herunter zu den einzelnen Steuerbefehlen, wird erst in der Endphase des Projekts zu einem bereits optimierten Modell hinzugefügt. Zu klären ist für diesen Fall, wie die Verfeinerung für Modifikationen am Prozeß und für statistische Untersuchungen zurückzunehmen ist.

Das Simulationsexperiment kann insgesamt als erfolgreich bezeichnet werden. Die Modellierung des Bausteinkastens war mit einem Aufwand, der den Rahmen einer Diplomarbeit nicht sprengt, möglich. Der erzeugte Bausteinkasten ermöglicht die einfache Erstellung lauffähiger kombinierter Modelle aus Prozeß und SPS.

# 6 Zusammenfassung

In der vorliegenden Arbeit wird der Transfersystem-Bausteinkasten TS0 der Firma BOSCH untersucht und eine Objektstruktur für ihn entwickelt, mit der Simulationsmodelle aufgebaut werden können. Der Bausteinkasten wurde zur Entwicklung ausgewählt, weil seine Komponenten in fast allen Transfersystemen wiedergefunden werden. Insgesamt ist die Anzahl der verschiedenen Funktionskomponenten in Transfersystemen sehr überschaubar. In 6 Systembaukästen sind lediglich 13 verschiedene Grundkomponenten zu finden. Durch Änderung der Parametrierung eines als Objektklasse beschriebenen Bausteins, kann dieser für alle Bausteinkästen verwendet werden. Durch die Objektdarstellung ist also eine ökonomische Modellierung verschiedener Baukastensysteme möglich. Häufig vorkommende Kombinationen von Bausteinen können in der objektorientierten Modellierung zu Klassen zusammengefügt werden, die die Modellierung erleichtern.

Auf der gewählten Beschreibungsebene wird zwischen Material- und Informationsfluß unterschieden. Das Verhalten von Bausteinen bezüglich Informations- und Materialfluß wird durch Methoden beschrieben. Jede Methode kann durch die Klassifikation ihrer Ein- und Ausgänge und deren Verknüpfung definiert werden, weil Methoden keinen permanenten Informationsinhalt besitzen. Durch ihre Ausführung wird die Dynamik der Simulation beschrieben. Permanente Zustände werden in Attributen von Bausteinen gespeichert. Durch seine Methoden und Attribute wird ein Modellierungsbaustein eindeutig beschrieben. Durch den Einsatz von Instanzen der definierten Bausteine in ein Modell und die Bestimmung der Messages zwischen ihnen, entsteht ein ablauffähiges Simulationsmodell.

Mit den Bausteinen, für die Methoden und Attribute beschrieben wurden, sind schnelle Simulationsstudien möglich. Die für die Bausteine angewandte Beschreibungsform ist so gewählt, daß ohne weiteres die in dieser Arbeit nicht berücksichtigten Bausteine für Transfersysteme erstellt werden können. Damit liegt eine Grundlage vor, Bausteinkästen für zahlreiche Transfersysteme zu erstellen und diese in der Praxis zu nutzen.

Ein Schwerpunkt bei der Bausteinbeschreibung liegt in der Darstellung ihres Steuerverhaltens. Alle Elemente der Bausteine, die das Materialflußverhalten kontrollieren, werden dazu modelliert. Diese Elemente werden durch Steuerungsobjekte bedient. Die Steuerungsobjekte enthalten den Steuerungscode in einer SPS-kompatiblen Darstellung. Durch diese Darstellungsweise können 3 Ziele erreicht werden:

- Teilautomatische SPS-Programmgenerierung durch das Anbieten fertiger Steuerungsbausteine, die mit den zugehörigen Prozeßbausteinen verknüpft sind
- Test von SPS-Programmen durch ihre Kopplung mit einer Simulation
- Off-Line Programmierung von Steuerungen

In dieser Arbeit wurden für die SPS-Programmdarstellung Petri-Netze eingesetzt. Diese erlauben mit einfachen Algorithmen die Erzeugung von Anweisungslisten, einer Darstellungsform

von SPS-Programmen nach /DIN 61131-3/, die von den meisten Programmiergeräten unterstützt wird.

Für die Simulation von Prozessen, hat sich eine prozeßorientierte Zeitsteuerung bewährt. Steuerungskomponenten, die ein quasikontinuierliches Verhalten aufweisen, werden durch zeitschrittorientierte Simulation am genauesten abgebildet. Die Kombination beider Zeitsteuerungsverfahren in einer Simulation führt zu sehr langsamen Modellen, weil in längeren Zeitabschnitten die Zahl der zu bearbeitenden Steuerungszyklen sehr hoch ist. In dieser Arbeit wird ein Ansatz vorgestellt, der das Steuerungsverhalten mit einer reinen Prozeßzeitsteuerung darstellt. Verzögerungen bei der Reaktion auf Prozeßzustandsänderungen, die bei realen Steuerungen durch die Zykluszeit auftreten, werden im hier vorgestellten Ansatz durch eine Reaktionsverzögerung um einen konstanten Zeitwert dargestellt. Durch Variation dieses Zeitwerts kann untersucht werden, wie sich verschieden schnelle Steuerungen auf den realen Prozeß auswirken würden. Das vorgestellte Verfahren konnte problemlos auf einen prozeßgesteuerten Simulator übertragen werden, und führte zu einer signifikanten Beschleunigung der Simulation gegenüber einer quasikontinuierlichen Steuerungsabbildung.

Durch die Abbildung von Steuerungen in Simulationsmodellen ist das Ziel dieser Arbeit, die Bereiche Steuerung und Simulation zusammenzuführen, erreicht worden.

Der Test von SPS-Programmen auf vollwertigen Simulatoren ist in der Literatur bisher nur in Ansätzen zu finden. Ausgearbeitete Konzepte zur Realisierung fehlen bisher. Auch auf dem Softwaremarkt ist bisher kein entsprechendes Produkt zu finden. Diese Arbeit betritt deshalb bei vielen Fragestellungen Neuland. Für alle Fragestellungen erschöpfende Antworten zu finden und diese bis zu einer anwendbaren Software auszuarbeiten, sprengt den Rahmen einer Diplomarbeit. Die gewählten Ansätze zu den Problembereichen sollen als Diskussionsgrundlage für die weitere Ausarbeitung des kombinierten Simulationskonzepts verstanden werden, das nach meiner Meinung Bewegung in die Steuerungstechnik bringen wird, und damit auch einen weiteren Anreiz zum Einsatz der Simulationstechnik darstellt.

# 7 Anhang

## 7.1 Abbildungen und Steuerungen der Bausteine

### 7.1.1 Positioniereinheit PE 0

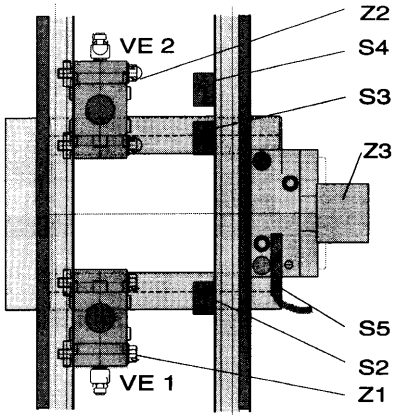


Abbildung 36: Positioniereinheit PE 0 /BOSCH TS0/

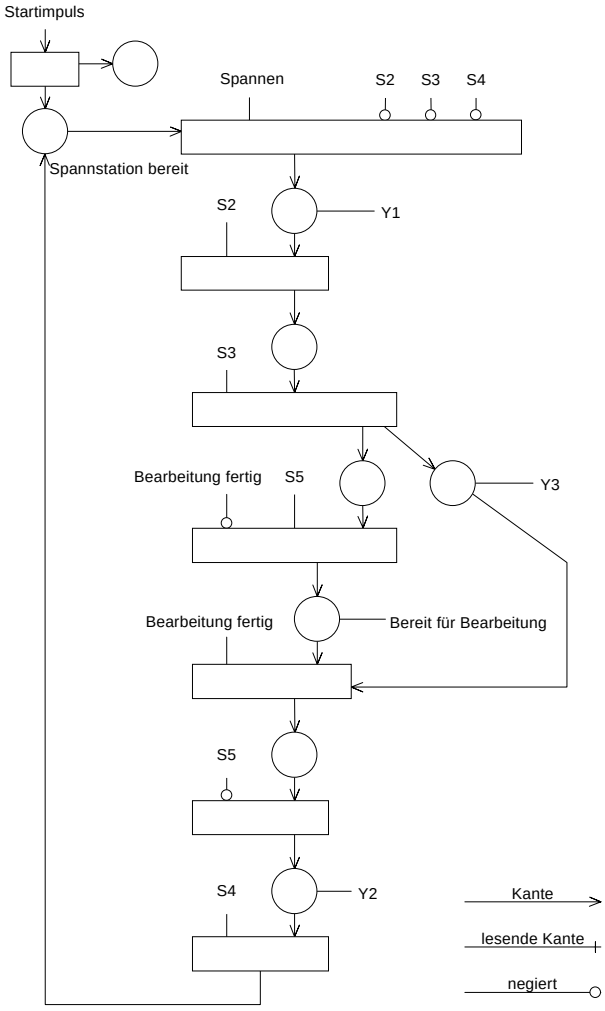


Abbildung 37: Steuerung zur Positioniereinheit

## 7.1.2 Elektrische Quertransporte EQ 0

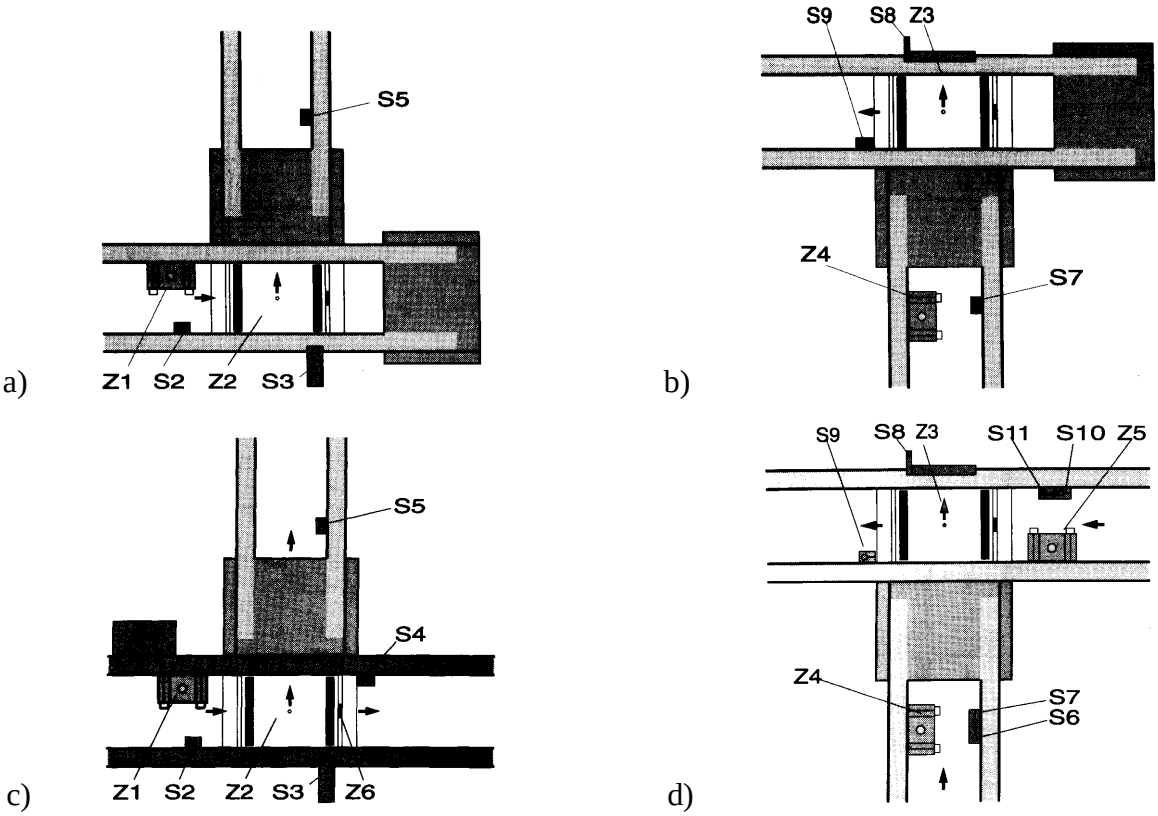


Abbildung 38: Elektrische Quertransporte EQ 0 /BOSCH TS0/

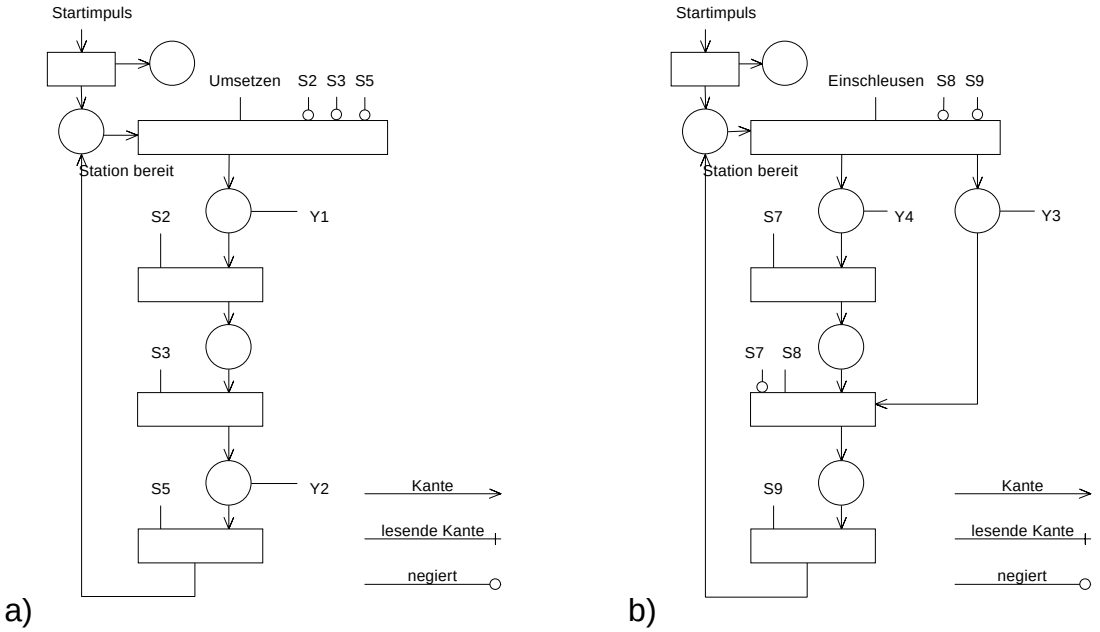


Abbildung 39: Steuerung zum elektrischen Quertransport (Abbildung 38 a,b))

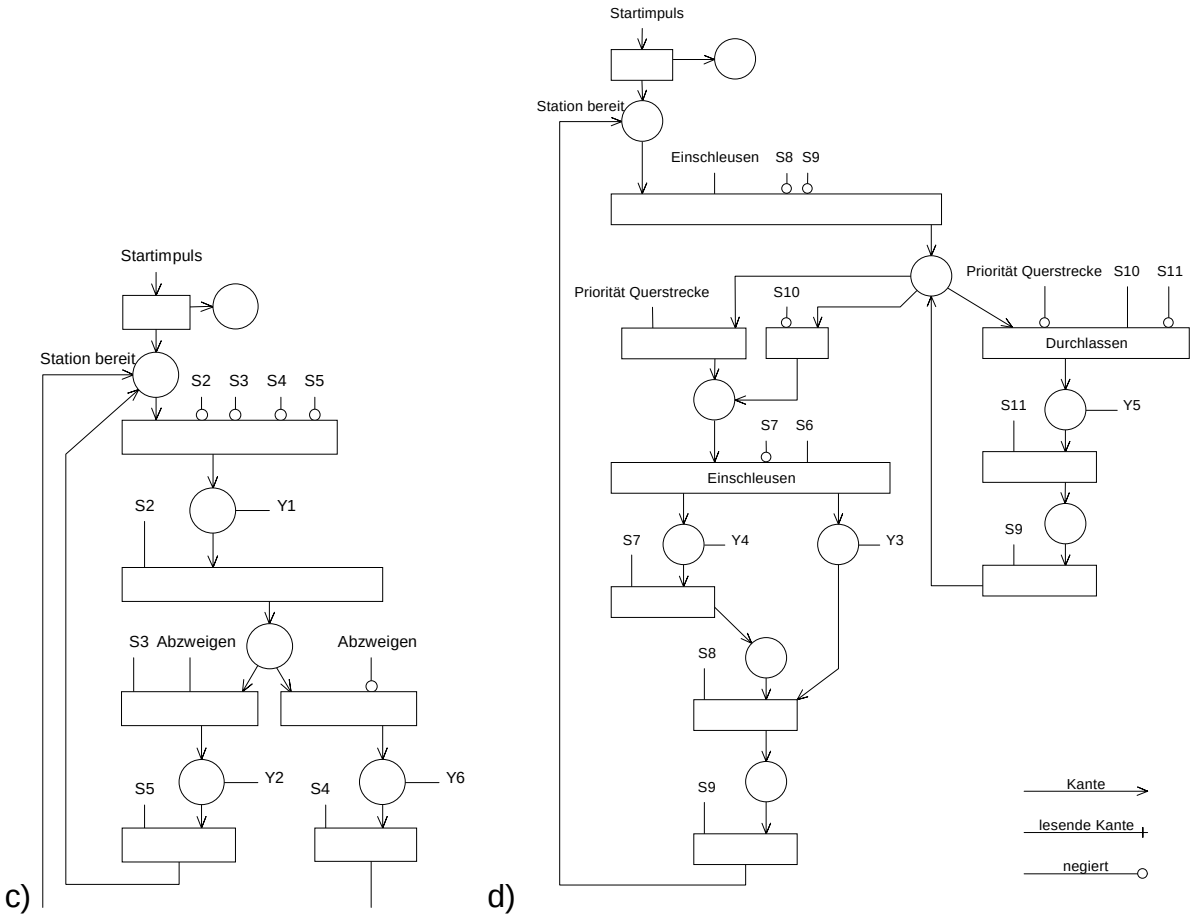


Abbildung 40: Steuerung zum elektrischen Quertransport (Abbildung 38 c,d)

## 7.1.3 Quertransporte in Tandemanordnung EQ 0/T

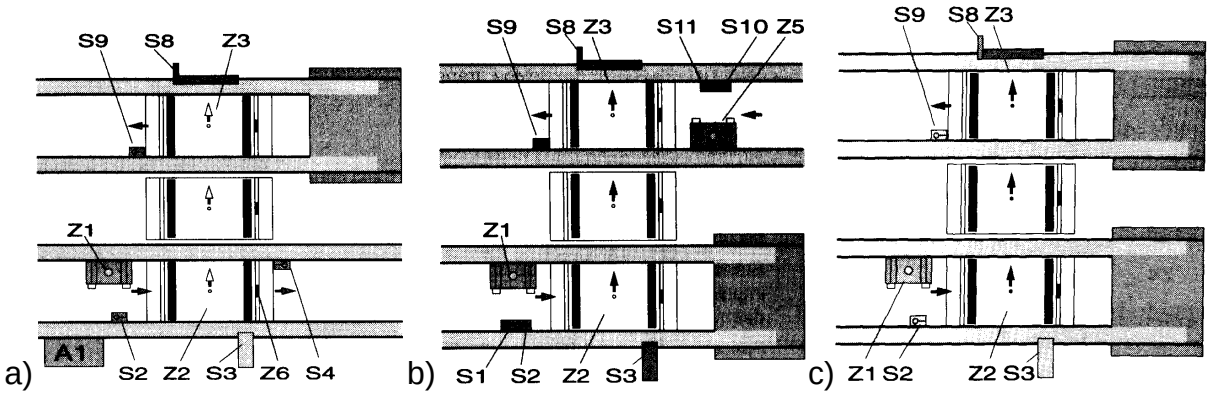


Abbildung 41: Elektrische Quertransporte in Tandemanodrdnung EQ 0/T /BOSCH TS0/

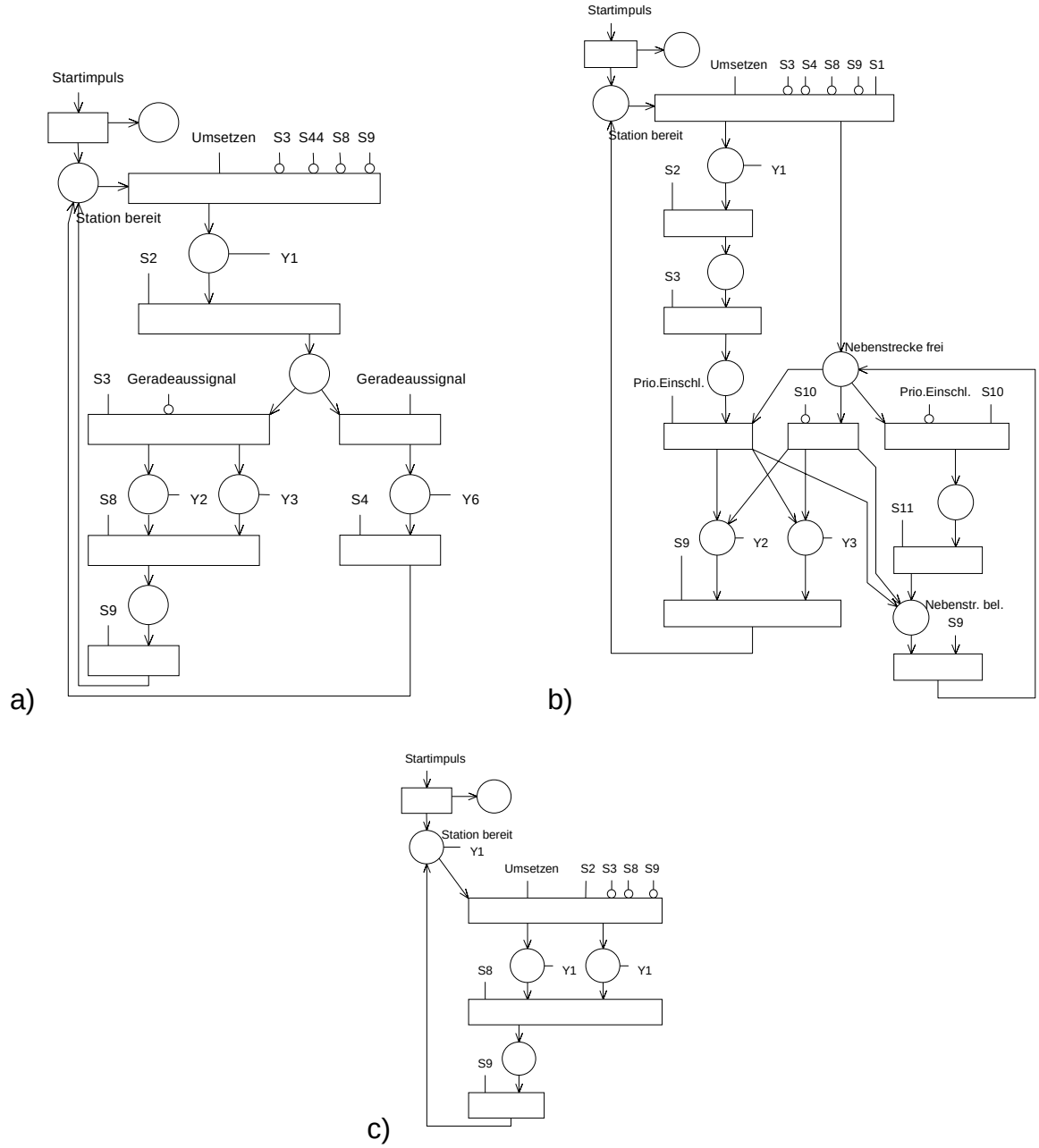


Abbildung 42 a-c: Steuerungen zum elektrischen Tandem-Quertransport (Abbildung 41 a-c)

## 7.2 Abbildungsverzeichnis

## 7.3 Tabellenverzeichnis

# 8 Literatur

/ASPERN 93/
Aspern, Jenes v.:
SPS-Softwareentwicklung mit Petri-Netzen.
Hüthing Verlag, Heidelberg, 1993.
<ta 7380>[27]

/AUER 94/
Auer, Adolf:
Steuerungstechnik und Synthese von SPS-Programmen.
Hüthing Verlag, Heidelberg, 1994.
<a elt 969 f 829>

/BECKER 91/
Becker, Bernd-Dietmar:
Simulationssystem für Fertigungsprozesse mit Stückgutcharakter: ein gegenstandsorientiertes System mit parametrisierter Netzwerkmodellierung.
Springer, Berlin, 1991.
<Sahlberg>

/BEN-ARIEH 88/
Ben-Arieh, D.:
A Knowledge- Based Simulation and Control System.
in: Kusiak, Andrew (Hrsg.):
Artificial Intelligence- Implications for CIM
Springer-Verlag, Berlin, 1988.
<a ing 385 mb/450>

/BOSCH TS0/
Bosch, Flexible Automation:
Transfersystem TS0.
Robert Bosch GmbH, Waiblingen, 1994.

[27] Falls nicht anders angegeben, beziehen sich die Signaturen auf die Universitätsbibliothek Bremen.

/C++SIM 94/
C++SIM User´s Guide:
Object-Orientatet Discrete-Event Simulation in C++.
Public Release 1.5.
Dept. of Computing Science, Univ. Newcastle UK, 1994.

/CIM 91/
CIM-Fachmann:
Simulation in CIM.
Weck, Manfred (Hrsg.).
Verlag TÜV Rheinland, Köln, 1991.
<01 b 1982>

/DIN 61131-3/
DIN/EN 61131-3:
Speicherprogrammierbare Steuerungen.
Teil 3: Programmiersprachen. (IEC 1131-3: 1993)
DIN im Beuth Verlag, Berlin, 1994.
<Patent- und Normenstelle, HFT Bremen>

/FISHWICK 95/
Fishwick, Paul A.:
Simulation, Modeldesign and Execution
Building Digital Worlds.
Prentice Hall, Englewood Cliffs, New Jersey, 1995
<Sahlberg>

/FISHWICK/
Fishwick, Paul A.:
SIMPACK: Getting Started with Simulation Programming in C and C++.
Dept. of Computer & Information Science, Univ. Florida, US.
<Internet: ftp.cis.ufl.edu /pub/simdigest
und WWW gopher://gopher.cis.ufl.edu:70/11/cis/simulation/>

/HEUMANN 93/
Heumann, Diethelm:
Objektorientierte Simulation teilautonomer Fertigungsstrukturen
Ein Beitrag zur Modellierung von Produktionssystemen -Dissertation-
Lehrstuhl für Produktionssysteme und Prozeßleittechnik der Ruhruniversität Bochum, 1993.
<Sahlberg>

/IEC 88/
IEC:
Programmable Controllers, Programming Languages.
IEC, Technical Comittee Nr. 64, 1988.
<Sahlberg>

/ISIS 94/
ISIS Engineering Report:
Software & Konzepte für C-Techniken, Firmenprofile, EDV-Lösungen für die Fertigungsindustrie und Bauwirtschaft.
Nomina, München, 1994.

/JANZEN 90/
Janzen, Friederich:
Projektierung von Speicherprogrammierbaren Steuerungen in einer integrierten, rechenerunterstützten Automatisierungsumgebung. -Dissertation-
Lehrstuhl für Produktionssysteme und Prozeßleittechnik der Ruhruniversität Bochum, 1990.
<Sahlberg>

/KÄMPER 91/
Kämper, Sabine:
PEGROS. Ein Konzept zur Entwicklung eines graphischen objektorientierten Modellbildungs- und Simulationswerkzeugs auf der Basis von Petri-Netzen.
Verlag Dr. Kovac, Hamburg, 1991.
<a inf 814 pen 908>

/KÜHN 94/
Simulation in Manufacturing Systems:
Proceedings of the 1st International Workshop WORKSIMS `94.
Kuehn, Wolfgang; Nagarur, N.N. (Hrsg.).
Asian INSTITUTE of Technology, Bangkok, Thailand, 1994.
<Sahlberg>

/KUHN 93/
Fortschritte in der Simulationstechnik (7):
Handbuch Simulationsanwendungen in Produktion und Logistik.
Kuhn, Axel; Reinhardt, Adolf; Wienahl, Hans-Peter (Hrsg.).
Vieweg, Braunschweig, 1993.
<a bwl 070.4 M 393>

/KUK 88/
Kuk, Kum-Hoan:
Modulares Simulationsmodell für die Abläufe in verketteten Fertigungszellen mit Industrierobotern. -Dissertation-
Fraunhofer-Institut für Produktionstechnik und Automatisierung, Stuttgart, 1988.
Springer-Verlag, Berlin, 1988.

/LESCHKA 95/
Leschka, Stephan; Vitiello, Marco:
Kombiniert.
ROSI - Ein Modul zur Kopplung von Simulationssystemen.
iX, 8/1995.

/MMS 93/
Fertigungsautomatisierung mit MMS:
Geräteunabhängiger Informationsaustausch. Eine Einführung mit Beispielen.
Winfried Blumann, Alhard Horstmann (Hrsg.).
VDI-Verlag, Düsseldorf, 1993.
<a bwl 379.2 128>

/MÖHRLE 90/
Möhrle, Michael:
Petri-Netze in der Produktionstechnik -
Integration und Planung, Simulation und Steuerung von Produktionsanlagen.
-Dissertation- Lehrstuhl für Produktionssysteme und Prozeßleittechnik der
Ruhruniversität Bochum, 1990.
<Sahlberg>

/NOCHE 91/
Noche, Bernd:
Marktspiegel Simulationstechnik in Produktion und Logistik.
Verlag TÜV Rheinland, Köln, 1991.
<UB der Fernuniversität Hagen, QBC/NOCW>

/PAGE 91/
Page, Bernd:
Diskrete Simulation. Eine Einführung in Modula-2.
Springer-Verlag, Berlin, 1991.
<Sahlberg>

/SCHNEIDER 95/
Entwurf komplexer Automatisierungssysteme:
Methoden, Anwendungen und Tools auf der Basis von Petri-Netzen und anderer formaler
Beschreibungsmittel.
Schneider, E (Hrsg.).
4. Fachtagung, Braunschweig vom 7.-9. Juni 1995.
Inst. f. Regelungs- und Automatisierungstechnik, TU Braunschweig, 1995.

/SIMPLE 95/
Werbeschrift der Firma AESOP zu SIMPLE++ 3.0.
AESOP GmbH, Stuttgart, 1995.

/VDI 3633/
VDI-Richtlinie 3633:
Anwendung der Simulationstechnik zur Materialflußplanung.
VDI-Verlag, Berlin, 1983.