# Ein mehrbenutzerfähiges Werkzeug zur Modellierung und richtungsoffenen Simulation von wahlweise objekt- und funktionsorientiert gegliederten Fertigungssystemen

**Meinen Leuchtfeuern**

# Ein mehrbenutzerfähiges Werkzeug zur Modellierung und richtungsoffenen Simulation von wahlweise objekt- und funktionsorientiert gegliederten Fertigungssystemen

Dissertation
zur Erlangung der Würde eines
DOKTORS DER WIRTSCHAFTSWISSENSCHAFTEN
(Dr. rer. pol.)
der Universität Paderborn

vorgelegt von
Dipl.-Wirt. Inf. Christoph Laroque
33104 Paderborn

Paderborn, den 24. Januar 2007

Dekan: Prof. Dr. Peter F. E. Sloane
Referent: Prof. Dr.-Ing. habil. Wilhelm Dangelmaier
Korreferent: Prof. Dr. Leena Suhl

Erstellt am

Heinz Nixdorf Institut
Fachgruppe Wirtschaftsinformatik, insb. CIM
Prof. Dr.-Ing. habil. Wilhelm Dangelmaier
Fürstenallee 11
33102 Paderborn

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Definitionsapparat

# 1 Motivation

*„Wir müssen das Rad*
*mit Mühe vorwärts drehen,*
*zurück rollt es von selbst.*"

(Walter Ludin)

Globalisierung, Produktindividualisierung und Ausweitung der Produktpalette bis hin zu Nischenprodukten prägen die Entwicklung der Angebotsstruktur in vielen Bereichen der industriellen Fertigung. Verkürzte Produktlebenszyklen, kundenorientierte Produktion und eine erhöhte Variantenvielfalt sind kennzeichnend für die Erzeugnisse heutiger Industrieunternehmen. Um dennoch kosten- und zeiteffizient fertigen zu können, wird die Digitalisierung von Produkt- und Prozessplanung mit Nachdruck verfolgt und stetig vorangetrieben [Brac02]. Neue Produkte werden im Idealfall vollständig digital am Rechner konstruiert, modelliert und optimiert. Neben Zusammenbauuntersuchungen lassen sich mit den digitalen Modellen beispielsweise virtuelle Crashtests durchführen oder verschiedene Designvarianten gegeneinander abwägen [Brac02]. Die Vorteile dieser Methodik liegen unter anderem in reduzierten Entwicklungskosten und –zeiten und helfen dadurch dem jeweiligen Unternehmen, seine Wettbewerbsfähigkeit am Markt zu verbessern [VDI4499].

Die fortschreitende Digitalisierung beschränkt sich jedoch keineswegs auf Modelle der herzustellenden Produkte, sondern bezieht sich darüber hinaus auch auf die zur Herstellung benötigten Prozesse. Eine Studie von Roland Berger stellt beispielhaft heraus, dass seitens der Unternehmen hohe Erwartungen an die Digitalisierung der Prozessplanung gestellt werden: *„Eine Zeitersparnis von bis zu 30% bei der Produktionsplanung und dem Produktionsanlauf erhoffen sich die Automobilbauer von der virtuellen Vorplanung bei 15% Kostenersparnis*" [Baum03].

Für die Planung, Absicherung und Verbesserung von Fertigungsprozessen ist die Ablaufsimulation ein etabliertes Werkzeug, welches es dem Anwender ermöglicht Struktur- bzw. Funktionsmodelle zu erzeugen und in einer Simulationsumgebung auszuführen [LaKe00]. Dadurch können während der Planung sowohl gegenwärtige, als auch zukünftige Situationen in ihren dynamischen Zusammenhängen berücksichtigt werden. Die Optimierung solch komplexer und dynamischer Szenarien kann nur noch durch ein experimentelles Betreiben verifizierter Modelle erfolgen, also durch den Einsatz der Ablaufsimulation (vgl. [DMD03]).

Die Weiterentwicklung vorhandener Softwarelösungen hat in den vergangenen Jahren mit den steigenden Anforderungen nur schwer Schritt halten können. Für den Problembereich der Ablaufsimulation von Fertigungssystemen soll deshalb mit dieser Arbeit ein neues Werkzeug entwickelt werden. Um Materialflussmodelle auf Basis einer integrierten Datenhaltung effizienter, möglichst anwenderfreundlich und in einer kooperativen Umgebung erstellen und ausführen zu können, sollen insbesondere folgende Aufgaben und Arbeitsweisen unterstützt werden:

***Synchronisierte, ortsunabhängige Mehrbenutzerunterstützung bei der Modellierung und Simulation von Materialflussmodellen in einer interaktiven, immersiven und virtuellen Umgebung***

Die Planung und Evaluierung eines Fertigungsprozesses kann heutzutage nicht mehr als ein Arbeitsschritt verstanden werden, bei dem der Prozess der Modellierung, Ausführung und Analyse der Simulationsmodelle am Computer einer einzelnen Person stattfindet. Die Komplexität der Planungsprojekte führt vielmehr dazu, dass sie zumeist von Simulationsteams bearbeitet werden. Neben Projektmitarbeitern aus verschiedenen Anwendungsbereichen arbeiten auch mehrere Simulationsexperten an einem einzigen Simulationsmodell. Sie werden durch die aktuell verfügbaren Software-Produkte nur unzureichend in ihrer Teamarbeit unterstützt. Darüber hinaus erfordert insbesondere die Kommunikation mit Nicht-Simulationsexperten innerhalb des Planungsteams eine möglichst immersive Darstellung, die das Verhalten des Simulationsmodells bestmöglich aufzeigt und erklärt. Zur Visualisierung der Modelle und deren dynamischen Verhaltensweisen soll eine virtuelle Umgebung dienen, in der das Simulationsmodell dreidimensional dargestellt wird. Der Anwender selbst soll hier nicht mehr länger ausschließlich passiver Betrachter sein, ohne interaktiv auf die aktuelle Simulation Einfluss zu nehmen, sondern vielmehr in der virtuellen Umgebung die Simulation beeinflussen und modifizieren können. Die Integration in das Simulationsmodell und damit auch in das abgebildete System, erhöht sein Prozessverständnis und schafft eine realistischere Planungsumgebung. Eine dreidimensionale Modellierung und Simulation als Kombination von Layout- und Prozessplanung kann den Anwender beim Aufbau des Fertigungsprozesses zusätzliche Planungsrestriktionen erkennen und von Beginn an berücksichtigen lassen. Die Qualität der Gesamtplanung kann verbessert und weitere Planungszeit innerhalb des Gesamtprojektes eingespart werden.

***Planung, Evaluierung und fortlaufende Verbesserung der Fertigungsprozesse über alle Planungs- und Ausführungsphasen bis zur Rückkopplung in die Fertigungslenkung***

Planung und Evaluierung der Fertigungsprozesse beschreibt den Einsatz der Ablaufsimulation über alle Planungs- und operativen Phasen eines Fertigungsprozesses hinweg. Neben dem Einsatz im Rahmen von Machbarkeitsstudien, Variantenplanungen oder quantitativen Fragestellungen, in denen ein System meist hinsichtlich seiner maximalen Leistungsfähigkeit oder eines optimalen Durchsatzes untersucht wird, soll die Simulation auch Fertigungsprogramme planen oder zumindest absichern können. Das Simulationsmodell eines Fertigungsprozesses kann somit über alle Phasen der Struktur-, Mengen-, Kapazitäts- und Programmplanung bis hin zur Prognose und der laufenden Verbesserung vorhandener Fertigungsprozesse eingesetzt werden. Daten- und Visualisierungsschnittstellen sollen über anpassbare Austauschformate konzipiert werden, um bewertete Simulationsergebnisse von Prognoseläufen zumindest als Entscheidungshilfe in den laufenden Fertigungsprozess zurückspielen zu können oder Planverbesserungen innerhalb der Fertigungslenkung direkt in den operativen Betrieb zu übernehmen. Um auch kundenorientierte Fertigungsprozesse möglichst homogen innerhalb eines Werkzeugs zur Ablaufsimulation abbilden zu können, soll neben einer Vorwärtssimulation auch eine Rückwärtssimulation (rückwärts berechnete Ausführung der Simulationsmodelle) sowie eine zeitorientierte Ausführung in Vor- oder Rückwärtsrichtung unterstützt werden.

***Kooperative Planung innerhalb von Unternehmen, Unternehmensverbünden oder Supply-Chain-Netzwerken***

Großunternehmen, virtuelle Unternehmen oder Supply-Chain-Netzwerke fertigen bereits heute an unterschiedlichen Standorten mit aufeinander abgestimmten Prozessen. Um die Lieferfähigkeit innerhalb eines Supply-Chain-Netzwerks sowie die Stabilität aller einzelnen Fertigungsprogramme besser gewährleisten zu können, müssen die Planungen enger aufeinander abgestimmt und überwacht werden. Die Mehrbenutzerfähigkeit der angestrebten Ablaufsimulation ermöglicht eine kooperative Planung mehrerer Simulationsexperten an einem gemeinsamen, dynamisch detaillierenden Simulationsmodell des Fertigungsnetzwerks unabhängig vom Standort der jeweiligen Experten. Innerhalb einer Supply-Chain oder eines Unternehmensverbundes, aber auch standortübergreifend innerhalb einer Unternehmung, können so auf Basis eines Rechtemanagements und umfangreicher Kommunikationsmechanismen Simulationsexperten aller Standorte gemeinsam Simulationsmodelle erarbeiten, verifizieren und validieren, Simulationsexperimente kooperativ ausführen und in einer einzigen, gemeinsamen Umgebung auswerten. Da die verschiedenen Partner innerhalb solcher unternehmensinternen wie –externen Fertigungsnetzwerke nach unterschiedlichen Fertigungsablaufarten produzieren können, muss neben dem Werkzeug an sich auch die zugrunde liegende Modellbeschreibung Fertigungssysteme beherrschen, die sowohl ein besonders hohes Maß an Komplexität bieten, als auch über rein objektorientiert gegliederte Fertigungssysteme hinausgehen und damit die Modellierung und Simulation von funktional gegliederten Fertigungssystemen bzw. deren Mischformen erlauben. Auch in diesem Themenbereich bietet sich der Einsatz des Werkzeugs über die reinen Planungsphasen hinaus bis hin zur Umplanung und Einsteuerung in der Fertigungslenkung an.

Aus obigen Themenbereichen ergeben sich Fokus und Motivation der Arbeit. ihre Struktur orientiert sich an folgender Vorgehensweise:
Kapitel 2 grenzt den Anwendungsbereich „Fertigung" und den Problembereich „Ablaufsimulation" so ein, wie er in der Arbeit betrachtet werden soll. Es beschreibt die hier aufgezeigten Lösungsideen präziser und erläutert alle zur Lösung erforderlichen Grundlagen. Abschließend werden die Anforderungen an den zugrunde liegenden Arbeitsprozess, die verwendete Modellbeschreibung sowie das Werkzeug selbst anhand der eingeführten Definitionen herausgearbeitet und als Anforderungsbeschreibung zusammengefasst. Die Analyse bestehender Konzepte zur Simulationsmethodik, deren Einzelprozesse „Modellierung", „Simulation" und „Visualisierung" und zu Verfahren des Software-Designs und -Entwicklung findet sich in Kapitel 3. Die identifizierten Ansätze werden jeweils in Bezug auf die gestellten Anforderungen bewertet. Die erkannten Defizite hinsichtlich der Anforderungen werden in Kapitel 4 zusammengefasst und als eigentliche Zielsetzung der Konzeption und Realisierung abgeleitet. Die Festlegung des Arbeitsprozesses, die Konzeption von Modellbeschreibung und Werkzeug erfolgt in Kapitel 5. Kapitel 6 beschreibt die Phase der Realisierung des Werkzeugs. Neben der eigentlichen Implementierung widmet es sich auch einer Beispielanwendung, die als typisch für den Problembereich angesehen werden kann. Die Ergebnisse zeigen, dass die entwickelte Modellbeschreibung mit dem Werkzeug angewandt werden kann und das Erreichen der Anforderungen ermöglicht. Die Zusammenfassung bietet in Kapitel 7 einen Ausblick auf weiterführende Folgearbeiten, die sich in dem gewählten Themenbereich anbieten.

# 2 Ablaufsimulation von Fertigungssystemen

*„In allen Grenzen ist auch etwas Positives"*

*(Immanuel Kant)*

Ziel dieses Kapitels ist das Erstellen einer vollständigen Anforderungsbeschreibung, anhand derer der zur Verfügung stehende Stand der Technik erarbeitet werden kann. Die in Kapitel 1 aufgezeigten Probleme müssen dazu präzisiert und erläutert werden. Dazu wird die Aufgliederung in drei Teilprobleme auch in Abschnitt 2.3 übernommen und fortgesetzt. Zu Beginn muss dazu in Abschnitt 2.1 zunächst der Anwendungsbereich „Fertigungssysteme" sowie unter Abschnitt 2.2 das Problemfeld der Modellierung, Simulation und Analyse mittels der Ablaufsimulation genauer eingegrenzt werden.

## 2.1 Fertigungssysteme

Die bereits in Kapitel 1 angerissenen Probleme stellen sich in einem globalisierten Wettbewerb insbesondere für solche Unternehmen, die Güter produzieren und diese an einem oder verschiedenen Märkten platzieren wollen. Den technisch zugrunde liegenden Prozess der Erstellung solcher Güter bezeichnet man als Fertigung.

**Definition 1: Fertigung**

> *„Die Fertigung umfasst alle technischen Maßnahmen zur Herstellung von Material oder Erzeugnissen. Sie ist grundsätzlich ein diskontinuierlicher Prozess"* [Dang99]

In Abgrenzung zum Begriff der Produktion beschränkt sich die Fertigung auf das Herstellen physischer Erzeugnisse mittels fortschreitender, diskontinuierlicher und zielgerichteter Transformationen in einem abgegrenzten System. Ein Fertigungssystem kann als solch ein operatives System zur Fertigung und damit als Objekt der jeweiligen Planungsaufgaben verstanden werden.

**Definition 2: Fertigungssystem**

> *„Ein Fertigungssystem ist eine technisch, organisatorisch (und kostenrechnerisch) selbstständige Allokation von Potentialfaktoren zu Fertigungszwecken (vgl. [Kern79]). Es besteht aus elementaren Arbeitssystemen, die die kleinste Einheit einer Kombination der Potentialfaktoren Betriebsmittel und Arbeitskräfte darstellen und eine oder mehrere Klassen von Transformationen durchführen können"* (in Anlehnung an [Rose92]

Als Fertigungsprozess soll ein Fortschreiten der Fertigung oder eines Elementes des Fertigungssystems verstanden werden.

**Definition 3: Fertigungsprozess**

> *„Ein Fertigungsprozess kennzeichnet das Fortschreiten im Durchführen eines Vorgangs der Fertigung"* [in Anlehnung an Dang99]

Ein Vorgang soll in dieser Arbeit als

**Definition 4: Vorgang**

> *„Ein Vorgang ist die zielgerichtete Transformation von Elementen in einem Einganszustand in Elemente in einem Austrittszustand mittels eines Verfahrens*" [DaWa97]

verstanden werden.

Die Fertigungsprozesse besser aufeinander abzustimmen und über die verschiedenen Planungsphasen einer Neu- oder Umplanung zu verbessern ist eine der Hauptaufgaben für den Einsatz der Ablaufsimulation. Oftmals sind die dynamischen Zusammenhänge zwischen den Einzelprozessen innerhalb einer Fertigung so komplex, dass sich die Auswirkungen von Änderungen den Planern nicht sofort und in Gänze erschließen. Haupteinsatzgebiet der Simulation ist deshalb heute oftmals die Absicherung von Fertigungsprozessen und damit auch der Anordnung von Fertigungssystemen innerhalb einer Fertigung vor deren physischer Umsetzung: die Fertigungsplanung.

**Definition 5: Fertigungsplanung**

> *„Die Planung der Fertigung umfasst alle einmalig zu treffenden Maßnahmen bezüglich der Gestaltung eines Fertigungssystems und der darin stattfindenden Fertigungsprozesse*"[Dang99]

Innerhalb der Fertigungsplanung kann weiter zwischen der eigentlichen Entwicklung neuer Prozesse und Verfahren und der Gestaltung der Fertigungsprozesse und ihrer Verbindungen in einer Fertigungsprozessplanung unterschieden werden (vgl. Abbildung 1). Um bestehende oder geplante Fertigungssysteme in ihrer realen oder geplanten Anordnung nachzubilden und die Fertigungsplanung somit zu verfeinern, wird in einem begleitenden Planungsschritt die Layoutplanung eines Fertigungssystems durchgeführt.

**Definition 6: Layoutplanung**

> *„Aufgabe der Layoutplanung ist, für eine vorgegebene Menge von Organisationseinheiten, deren wechselseitige Fördervorgänge bekannt sind, in einer gegebenen Planungsfläche einen Anordnungsplan mit minimalen Förderkosten zu suchen*" [Dang99, in Anlehnung an Anordnungsplanung]

Um die entstandenen Simulationsmodelle auch in späteren Planungsphasen und dem laufenden Betrieb nutzbar zu halten, müssen sie permanent gepflegt und gewartet werden. Wie in der Motivation angerissen, kann die Ablaufsimulation zukünftig auch neue Anwendungsgebiete im operativen Betrieb der Fertigungssysteme erschließen, beispielsweise zur Absicherung von Planzahlen oder als Prognoseinstrument in einem Leitstand. Ihre Anwendung weitet sich in den Bereich der Fertigungslenkung aus.

**Definition 7: Fertigungslenkung**

> *„Fertigungslenkung ist die Aufgabe, für ein gegebenes Fertigungssystem – ausgehend von gegebenen Daten – Solldaten, die in sich und mit den Ausgangsdaten konsistent sind, für einen definierten, zielgerichteten Ablauf eines*

*Fertigungsprozesses festzulegen, dem Fertigungsprozess vorzugeben und diesen auf Inkonsistenzen abzuprüfen*" [DaWa97]

In Anlehnung an die Darstellung von Hackstein [Hack89] können innerhalb der Fertigungslenkung (bei Hackstein: Produktionssteuerung) folgende Unterfunktionen unterschieden werden:

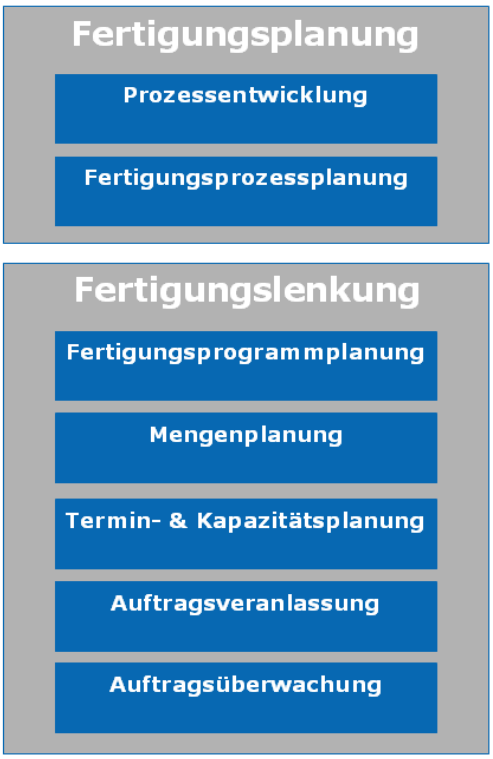


**Abbildung 1: Funktionen der Fertigungslenkung**

Die Unterfunktionen der Fertigungslenkung sollen im Folgenden kurz beschrieben werden, um Einsatzgebiete der Ablaufsimulation in der Anforderungsbeschreibung (vgl. Abschnitt 2.4) präzise formulieren zu können.

**Definition 8: Fertigungsprogrammplanung**

*„Die Planung des Fertigungsprogramms (Produktionsprogrammplanung) besteht darin festzulegen, welche Erzeugnisse in welchen Mengen in einem bestimmten Zeitabschnitt hergestellt und verkauft werden sollen.*" [Dang99]

Die Fertigungsprogrammplanung wird üblicherweise in die Schritte strategische, taktische und operative Planung unterteilt. Die strategische Planungsphase legt die wesentlichen Geschäftsfelder des Unternehmens fest, wohingegen die taktische Planung den Fokus auf die Entwicklung und Auswahl neuer Erzeugnisse legt. Die Ablaufsimulation zur Absicherung von Fertigungsprogrammen kommt vermehrt in der dritten Phase, der operativen Planung zum Einsatz, in der auf Basis vorheriger Planungsphasen die Stückzahlen der zu fertigenden Erzeugnisse für einen festgelegten Zeitraum festgelegt und beplant werden (vgl. [Dang99]). Heutzutage ist die Ablaufsimulation auf den Einsatz als „Frühwarnsystem" beschränkt, liefert also nur Prognosen über zukünftige Systemzustände, um den Fertigungslenker größere Handlungszeiträume zu ermöglichen

[Klos06]. Mit dem hier zu konzipierenden Werkzeug soll die Ablaufsimulation auch die eigentliche Planung von Fertigungsprogrammen unterstützen.

**Definition 9: Mengenplanung**

> *„Die Mengenplanung hat die Aufgabe, Aufträge über Erzeugnisse und davon abgeleitete Materialien an die Fertigung und den Einkauf mit den erforderlichen Zeitpunkten und über die erforderliche Menge zu übermitteln, so dass das Fertigungsprogramm erfüllt werden kann*" [DaWa97]

Eine Reihenfolge der herzustellenden Erzeugnisse wird erst in den nachfolgenden Stufen ermittelt. Teilaufgaben der Mengenplanung sind die Bestands- und Bestellrechnung, die Bedarfsrechnung sowie die Stücklistenorganisation. Die Mengenplanung kann verbrauchsorientiert, wie auch bedarfsorientiert erfolgen[1].

**Definition 10: Terminplanung**

> *„Die Terminplanung dient dazu, Aussagen über Termingerüste für die Bewältigung einer Gesamtaufgabe zu machen. Die Terminplanung betrachtet Ablaufstrukturen, die nur einmal instanziiert werden.*" [DaWa97]

Die Terminplanungsverfahren berücksichtigen nicht die Kapazitätsrestriktionen der Fertigungselemente, sondern beachten nur Zeitabläufe innerhalb der vorgegebenen Ablaufstrukturen. Für jeden einzelnen Fertigungsprozess werden nur Start- und Endtermine berücksichtigt.

**Definition 11: Kapazitätsplanung**

> *„Bei der Kapazitätsplanung sind Beginn-Termin und Fertigstellungs-Termin eines in der Mengenplanung festgelegten Fertigungsauftrags sowie die Zwischentermine der einzelnen Fertigungsprozesse unter Berücksichtigung eines begrenzten Kapazitätsangebots festzulegen.*" [in Anlehnung an DaWa97]

In der Kapazitätsterminierung wird grob geplant, ob die erforderlichen Kapazitäten für das Fertigungsprogramm vorhanden sind. Bei Kapazitätsengpässen müssen einzelne Arbeitsschritte in andere Zeiträume verschoben werden. Sobald dies geschehen ist, können grob terminierte Aufträge an die folgenden Schritte der Fertigungslenkung (Auftragsveranlassung und –überwachung) übergeben werden. Innerhalb der operativen Planungsaufgaben kommen vermehrt Methoden der Rückwärtsterminierung zum Einsatz (vgl. Abschnitt 2.3). Die Ergebnisse der Feinplanungsphasen dienen als Ausgangsbasis für die nachfolgenden Stufen der Auftragsveranlassung und –überwachung.

Fertigungssysteme können nach unterschiedlichen Merkmalen klassifiziert werden. Neben Merkmalen, die sich auf das resultierende Erzeugnis beziehen, können Merkmale gefunden werden, die sich auf den organisatorischen und logischen Aufbau der

[1] Eine genaue Beschreibung der verbrauchs-, bzw. bedarfsorientierten Mengenplanung findet sich bei [DaWa97]

Fertigungsprozesse beziehen. Neben Fertigungsart und –Struktur ist insbesondere die Organisationsform kennzeichnend für den Materialfluss.

**Definition 12: Organisationsform**

> *„Die Organisationsform kennzeichnet die am Fertigungsprozess orientierte Art der Zusammenführung von Betriebsmitteln zu organisatorischen Einheiten. Sie bestimmt ganz wesentlich die Qualität der inter-OE-Beziehungen und die räumliche Anordnung der Organisationseinheiten*" [Dang99]

Neben der räumlichen Anordnung wirkt sich die gewählte Organisationsform entscheidend auf Beziehungen der Organisationseinheiten untereinander und damit auf den Materialfluss der Fertigung aus. Unter einem Materialfluss in der Fertigung wird gemäß VDI 3300 verstanden [VDI3300]:

**Definition 13: Materialfluss**

> *„Materialfluss ist die Verkettung aller Vorgänge beim Gewinnen, Be- und Verarbeiten, sowie bei der Verteilung von stofflichen Gütern innerhalb festgelegter Bereiche. Zum Materialfluss gehören alle Vorgänge während des Durchlaufes von Gütern (z.B. Material, Stoffmengen, Abfall, Datenträger usw.) durch ein System, wie Bearbeiten, Handhaben, Transportieren, Prüfen, Aufenthalte und Lagerungen. ...*" [VDI3300]

Die Organisationsform eines Materialfluss einer Fertigung kann gemäß [DaWa97] nach zwei grundlegenden Prinzipien unterschieden werden: *funktionsorientiert gegliederte Fertigungssysteme* und *objektorientiert gegliederte Fertigungssysteme*[2]. Sie sollen im Folgenden kurz charakterisiert werden.

Ein *funktionsorientiert gegliedertes Fertigungssystem* liegt vor, wenn sich die räumliche Anordnung der Betriebmittel (Maschinen, etc.) an deren Fertigungsaufgabe orientiert. Ein typisches Prinzip für eine funktionsorientiert gegliederte Fertigung ist die Werkstattfertigung, für die die Zusammenfassung von Fertigungsmitteln mit gleichartigen Fertigungsvorgängen zu Abteilungen und das Fehlen fester Transportbeziehungen zwischen den Fertigungsmitteln typisch ist. Neben einer hohen Elastizität kann eine verbesserte Flexibilität gegenüber wechselnden Anforderungen innerhalb der Fertigung garantiert werden. Diesen Vorteilen stehen eine verringerte Übersichtlichkeit des Gesamtprozesses und ein erhöhtes Transportaufkommen gegenüber.

Unter den *objektorientiert gegliederten Fertigungssystemen* werden diejenigen Organisationsformen summiert, deren Anordnung sich an dem herzustellenden Erzeugnis orientiert. Häufige Organisationsformen sind Gruppen-, Linien und Fließfertigung. Bei der Gruppenfertigung werden die zur Bearbeitung ähnlicher Fertigungselemente benötigten Fertigungsmittel räumlich zusammengestellt. Bei einer Linienfertigung sind die Fertigungsmittel nach der Ablauffolge angeordnet und durch einfache Transporteinrichtungen verknüpft. Sowohl Gruppen-, als auch Linienfertigung sind

[2] [Dang99] unterscheidet zwischen ortsgebundener und ortsveränderlicher Fertigung. Die wesentliche Unterscheidung basiert aber auch hier auf der Frage, „*wer sich bei der Zuordnung von Fertigungsobjekt und Betriebsmittel zur Durchführung einer Aufgabe bewegt: Fertigungsobjekt oder Betriebsmittel*"

gegenüber Änderungen der Fertigungsablauffolge flexibel. Gegenüber der Linienfertigung zeichnet sich die Fließfertigung durch eine feste Verkettung der einzelnen Arbeitsstationen aus. Dadurch werden eine besonders niedrige Durchlaufzeit und eine hohe Transparenz bezüglich des Fertigungsprozesses erreicht. Eine Fließfertigung ist dadurch allerdings auch weniger flexibel und reagiert empfindlicher auf Störungen. Je nach Ausprägung des Merkmals Fertigungsart (Losgröße, Wiederholhäufigkeit) kann die Fließfertigung in Serien- oder Massenfertigung differenziert werden, wobei die Massenfertigung als Extremfall einer Serienfertigung angesehen werden kann.

Der Anwendungsbereich dieser Arbeit soll damit hinreichend genug klassifiziert sein, um die Anforderungen an den zu entwickelnden Prozess, die Modellbeschreibung und das eigentliche Werkzeug formulieren zu können. Nachfolgend muss als nächster Schritt auch das bisher eher vage dargestellte Problemfeld der Ablaufsimulation entsprechend präzisiert werden.

## 2.2 Ablaufsimulation: Modellierung, Simulation & Analyse

Die Ablaufsimulation von Fertigungsabläufen ist ein Instrument mit dem strategische, taktische und operative Entscheidungen abgesichert werden können. Ein einmal erstelltes Prozessmodell erlaubt eine schnelle Analyse verschiedener Varianten eines Prozesses. Fragen wie: „Wie viel mehr kann ich durch den Einsatz eines weiteren Flurfördergerätes produzieren?" lassen sich so schnell beantworten. Die Robustheit von Fertigungsprozessen auf exogene und endogene Einflüsse lässt sich durch Sensitivitätsanalysen mittels der Ablaufsimulation untersuchen.

Das zu entwickelnde Werkzeug arbeitet im Problemfeld der diskreten, ereignisgesteuerten Ablaufsimulation, in der Fertigungssysteme hinsichtlich ihres Materialflusses inklusive aller Einflussfaktoren abgebildet werden. Um insbesondere diesen Typus der Simulation im Folgenden abzugrenzen, müssen einige grundlegende Begriffsdefinitionen eingeführt werden.

**Definition 14: Simulation**

> *„Simulation ist das Nachbilden eines Systems mit seinen dynamischen Prozessen in einem experimentierfähigen Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind. Im weiteren Sinne wird unter Simulation das Vorbereiten, Durchführen und Auswerten gezielter Experimente mit einem Simulationsmodell verstanden*" [VDI3633]

Bezieht sich die Aufgabenstellung der Simulation insbesondere auf die dynamischen Prozesse eines Modells, wird von der *Ablaufsimulation* gesprochen. Sie betrachtet zeitliche und kapazitative Auslastungen. Simulationen stützen sich stets auf vom Anwender erzeugte Modelle. Ein Modell kann nach [KlBu71] wie folgt aufgefasst werden:

**Definition 15: Modell**

> *„Ein Objekt, das auf der Grundlage eine Struktur-, Funktions- oder Verhaltens-analogie zu einem entsprechenden Original von einem Subjekt eingesetzt und genutzt wird, um eine bestimmte Aufgabe lösen zu können, deren Durchführung mittels direkter Operation zunächst oder überhaupt nicht möglich bzw. unter gegebenen Bedingungen zu aufwendig ist. …*" [KlBu71]

Modelle als Abbildung von Fertigungssystemen werden in der Praxis häufig so komplex, dass eine Abbildung durch dreidimensionale Modelle oder realitätsnahe Funktionsmodelle heute schnell an zeitliche und wirtschaftliche Grenzen stößt. Im Kontext der Fertigungsplanung und –lenkung werden deshalb zumeist Modelle verwendet, die mittels einer Simulation auf einem Computer berechnet und ausgeführt werden können.

**Definition 16: Rechnerunterstützte Simulation**

> *„Rechnerunterstützte Simulation ist das experimentelle Betreiben eines Modells auf einer Rechneranlage*"[Dang99]

Unter Einsatz der Ablaufsimulation und mit Hilfe eines Simulators kann der Anwender im Rechner als Modell vorliegende Systemalternativen analysieren und hinsichtlich der Zielerfüllung überprüfen.

**Definition 17: Simulator**

> *„Softwareprogramm, mit dem ein Modell zur Nachbildung des dynamischen Verhaltens eines Systems und seiner Prozesse erstellt und ausführbar gemacht werden kann. Ein Simulator beinhaltet einen Simulatorkern, eine Datenverwaltung, eine Bedienoberfläche und gegebenenfalls weitere Schnittstellen*" [VDI3633]

Die Ablaufsimulation ist also eine Methode zur Analyse von dynamischen Zustandsänderungen in einem System, im vorliegenden Fall von Fertigungssystemen. Sie wird in vielen Bereichen in verschiedenen Ausprägungen eingesetzt. Um das Problemfeld weiter einzugrenzen, müssen weitere Klassifikationen herangezogen werden. Als ein wesentliches Klassifikationsmerkmal dient die Differenzierung zwischen „diskreter" und „kontinuierlicher" Simulation.

**Definition 18: diskret**

> *„Diskret – eigtl.: geschieden, unstetig, diskontinuierlich, auch: abgesondert, getrennt. …*" [KlBu71]

**Definition 19: Kontinuierlich**

> *„Kontinuierlich, zusammenhängend, lückenlos. …*"[KlBu71]

Der Zustand zu einem Simulationszeitpunkt wird durch Zustandsvariablen (z.B. der Belegungszustand einer Maschine) beschrieben. In kontinuierlichen Simulationen können diese Zustandsvariablen kontinuierlich über die Simulationszeit ihren Wert ändern. So kann z.B. die Geschwindigkeit eines beschleunigenden Fahrzeuges über die Beschleunigung und die Zeit beschrieben werden. In diskreten Simulationen verändern Zustandsvariablen ihren Wert nur an endlich vielen Zeitpunkten. So kann zum Beispiel der Füllstand eines Lagers über die Anzahl der eingelagerten Elemente beschrieben werden. Die Zeitpunke der Zustandsübergänge werden als Ereignis bezeichnet.

**Definition 20: Ereignis**

*„Atomare Begebenheit, die eine Zustandsänderung bewirkt und keine Zeit verbraucht*" [VDI3633]

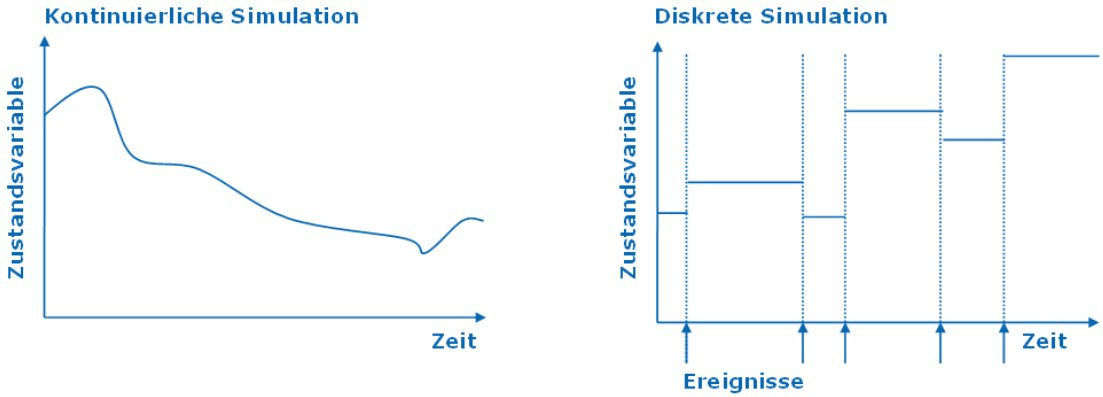


**Abbildung 2: diskrete und kontinuierliche Simulation**

In dieser Arbeit sollen nur diskrete Simulationen betrachtet werden. Hier kann weiter unterschieden werden, wie die einzelnen Zustandsübergänge zeitlich voneinander abhängen. Man unterscheidet zwischen *fixen* und *variablen Zeitinkrementen*. In diesem Werkzeug sollen durch eine sinnvolle Konstruktion des Simulatorkerns und der Modellbeschreibung grundsätzlich beide Möglichkeiten der Zeitfortscheitung unterstützt werden. In Abschnitt 3.3 werden deshalb beide Methoden der Zeitfortschreitung bei der Ausführung von Simulationsmodellen näher betrachtet. Insbesondere wird untersucht, wie ein Simulator strukturiert werden muss, um beide Ausführungsformen zu unterstützen.

Typischerweise wird die Methode Ablaufsimulation in jeglicher Ausführung in Form einer Simulationsstudie angewandt, um Fertigungssysteme hinsichtlich eines bestimmten Untersuchungszwecks zu analysieren.

**Definition 21: Simulationsstudie**

*„Projekt zur simulationsgestützten Untersuchung eines Systems. (...) Eine Simulationsstudie kann mehrere Simulationsexperimente umfassen, die ihrerseits aus mehreren Simulationsläufen bestehen können*" [VDI3633]

Neben typischen Projektaufgaben kann eine Simulationsstudie in drei Hauptaufgaben untergliedert werden. Zunächst den Prozess der Modellerstellung: die Modellierung.

**Definition 22: Modellierung**

*„Modellierung ist der Prozess der Überführung eines Realitätsausschnittes in ein Modell.*"[ FiHe00]

Die zweite Hauptaufgabe beschreibt das Ausführen des dynamischen Modells in einem Simulator, der eigentliche Simulationslauf bzw. das Simulationsexperiment.

**Definition 23: Simulationslauf**

*„Nachbildung des Verhaltens eines Systems mit einem spezifizierten, ablauffähigen Modell über einen bestimmten (Modell-)Zeitraum…*"[VDI3633]

**Definition 24: Simulationsexperiment**

> *„Systematischer Plan zur Ausführung einer Menge von Simulationsläufen mit unterschiedlichen Anfangszuständen und Parametereinstellungen zur effizienten Untersuchung des Modellverhaltens*" [VDI3633]

Innerhalb eines Simulationsexperimentes müssen die in den Simulationsläufen gesammelten Daten ausgewertet werden. Diese Kombination von Simulationsexperimenten und Auswertung hinsichtlich bestimmter Untersuchungskriterien wird als Analyse bezeichnet und beschreibt die dritte Hauptaufgabe innerhalb einer Simulationsstudie.

**Definition 25: Analyse**

> *„Verfahren zur Untersuchung und Erkenntnis materieller oder ideeller Gegebenheiten, dessen Wesen in der praktischen oder gedanklichen Zerlegung eines Ganzen in seine Teile, eines Zusammengesetzten in seine Elemente besteht. Das Ziel der Analyse besteht darin, wesentliche Eigenschaften und Relationen von unwesentlichen, notwendige von zufälligen, allgemeine von individuellen zu unterscheiden und auf diesem Wege von der undifferenzierten Betrachtung der Gesamterscheinung zur Erkenntnis ihres Wesens und der sie bestimmenden Gesetzmäßigkeiten vorzudringen.*"[KlBu71]

Eine genaue Simulationsmethodik, wie sie in der Praxis ihre Anwendung findet, wird in Abschnitt 3.1 beschrieben. Mittels der Ablaufsimulation werden bereits heute zahlreiche Fragestellungen bezüglich der Abstimmung von Fertigungsprozessen erarbeitet. Dennoch lassen sich nachfolgend einige Teilprobleme identifizieren, deren Lösung die Verbreitung und Akzeptanz der Methode Ablaufsimulation weiter fördern sollen. Sie wurden in der Motivation aufgezeigt und sollen nun entsprechend präzisiert werden.

## 2.3 Erweiterungen aus der Motivation

Bereits Kapitel 1 hat verschiedene Szenarien motiviert, in denen die Ablaufsimulation zukünftig angewendet werden kann. Nachfolgend sollen diese Szenarien näher erläutert werden und, soweit nötig, die dafür benötigten Begriffe und Definitionen eingeführt werden.

***Synchronisierte, ortsunabhängige Mehrbenutzerunterstützung bei der Modellierung und Simulation von Materialflussmodellen in einer interaktiven, immersiven und virtuellen Umgebung***

Durch die steigende Bedeutung der Ablaufsimulation werden auch die betrachteten Projektszenarien für Simulationsstudien immer komplexer. Infolgedessen werden die Projekte auch nicht mehr von einzelnen Simulationsexperten bearbeitet, sondern mehrere Anwender, typischerweise mit verschiedenen Aufgaben, Interessen und Erfahrungen mit der Ablaufsimulation, arbeiten innerhalb eines gemeinsamen Projektes

an der Lösung des Untersuchungszieles. Die Planung und Evaluierung eines Fertigungsprozesses kann also nicht als ein Arbeitsschritt verstanden werden, bei dem der Prozess der Modellierung, Ausführung und Analyse der Simulationsmodelle am Computer einer einzelnen Person stattfindet. Verschiedene Projektmitarbeiter erarbeiten ein gemeinsames Simulationsmodell. Dieser Vorgang erfordert eine umfangreiche Interaktion zwischen den Modellierern und die gemeinsame Nutzung von Daten und Informationen über das Simulationsmodell oder dessen Bausteine. Der Modellierungsprozess erstreckt sich typischerweise über einen längeren Zeitraum und erfordert den Zugriff auf komplexe Informationsbestände. Das Modelliererteam kann ebenso wie das erstellte Simulationsmodell Substrukturen beinhalten, wie z. B. Teilprojekte und Arbeitsgruppen.

Während der Durchführung einer Simulationsstudie sollen deshalb im angestrebten Werkzeug mehrere Anwender sowohl gleichzeitig, als auch zeit- und ortunabhängig gemeinsam an einem Simulationsmodell arbeiten können. Die einzelnen Phasen der Modellierung und Simulation sind dazu in einem System zu handhaben, dass für einen solchen Betrieb ausgelegt ist. Ein Modellierungswerkzeug, das den skizzierten Prozess der gemeinschaftlichen Modellierung unterstützt, kann als *kooperatives Modellierungs-werkzeug* bezeichnet werden.

**Definition 26: Mehrbenutzersystem**

> *„Ein Mehrbenutzersystem oder Multiuser-System ist eine Software, die die Fähigkeit hat, Arbeitsumgebungen für verschiedene Benutzer bereitstellen und voneinander abgrenzen zu können.*"[TaSt03]

**Definition 27: Multitasking**

> *„Multitasking bezeichnet die Fähigkeit einer Software, mehrere Aufgaben (tasks) scheinbar gleichzeitig auszuführen. Dabei werden die verschiedenen Prozesse in so kurzen Abständen immer abwechselnd aktiviert, dass der Eindruck der Gleichzeitigkeit entsteht.*" [TaSt03]

Zur Modellierung werden verschiedene Interaktionstechniken benötigt. Dazu werden den benutzten Objekten Verhaltensweisen zugewiesen. Diese Parametrisierung definiert, in welcher Weise Objekte gesetzt bzw. in die Modellierungsumgebung integriert werden können. Elementare Verhaltensweisen stellen das Erzeugen, Selektieren, Erweitern, Parametrieren, Löschen, Bewegen und Verbinden von Objekten dar. Gemäß dem Anspruch des Werkzeugs müssen diese Interaktionsmetaphern kooperativ ausgeführt werden können. Unter *Multitasking-Modellierung* soll also der synchronisierte Prozess der Modellierung verstanden werden, der von mehreren Anwendern gleichzeitig an demselben Modell durchgeführt werden kann.

Der Modellierungsphase schließt sich die Durchführung von Simulationsexperimenten an. Sie beinhaltet die Ausführung selbst sowie die Analyse der Ergebnisse. Initialwerte als Eingabeparameter eines Simulationslaufes bestimmen den mengenmäßigen und zeitlichen Fluss der Marken im Simulationsmodell. Analyseeinstellungen bestimmen für jeden Simulationslauf, wo und in welcher Komplexität die Informationen über die

Simulation gesammelt werden. Während der Durchführung steht die Benutzerfreundlichkeit für den Anwender im Mittelpunkt. Idealerweise kann er Parameter während der Durchführung ändern und die Auswirkung seiner Änderung sofort in einer entsprechenden Visualisierung beobachten: Er interagiert mit dem Simulationsmodell.

**Definition 28: Interaktion**

> *„Interaktion bezeichnet das wechselseitige aufeinander Einwirken von Akteuren oder Systemen…In Bezug auf die Mensch-Maschine-Interaktion meint der Begriff die Gestaltung einer Benutzerschnittstelle zu Programmen…*" (nach [Buur05])

Genau wie bei der Modellierungsphase haben diese Interaktionen mehrerer Anwender Einfluss auf die berechneten Ergebnisse eines Simulationslaufs. Analog zur Multitasking-Modellierung kann unter *Multitasking-Simulation* also der synchronisierte Prozess der Ausführung und Berechnung eines Simulationsmodells verstanden werden, der von mehreren Anwendern gleichzeitig an demselben Modell durchgeführt werden kann. Zusätzlich muss hierbei unterschieden werden, ob die Anwender an einer Instanz eines Simulationsmodells arbeiten, oder dasselbe Basismodell mehrfach instanziieren.

Darüber hinaus erfordert insbesondere die Kommunikation mit Nicht-Simulationsexperten innerhalb des Planungsteams eine möglichst realistische Darstellung, die das Verhalten des Simulationsmodells bestmöglich darstellt und erklärt. Der Anwender soll sich möglichst immersiv in das Simulationsmodell hineinversetzen.

**Definition 29: Immersion**

> *„Immersion meint das Eintauchen in eine künstliche Welt... Durch die Immersion erlebt der Anwender direkt die Dynamik der physikalischen Vorgänge*" [Borm94]

Zur Visualisierung der Modelle und deren dynamischen Verhaltensweisen soll eine virtuelle Umgebung dienen, in der das Simulationsmodell dreidimensional dargestellt wird.

**Definition 30: Virtuelle Realität**

> *„Als Virtuelle Realität (VR) wird die Darstellung und gleichzeitige Wahrnehmung der Wirklichkeit und ihrer physikalischen Eigenschaften in einer in Echtzeit computergenerierten virtuellen Umgebung bezeichnet.*" (nach [FaHa94])

Die Vorteile einer möglichst hohen Immersion in die virtuelle Umgebung werden in Abschnitt 3.4.3 allgemein und in Abschnitt 3.4.4 speziell für die Anwendung im Bereich der Ablaufsimulation näher untersucht. Insgesamt ergibt sich daraus die Anforderung, dem Anwender eine möglichst gute Form der Darstellung zu realisieren. Der Anwender selbst wird nicht mehr länger ausschließlich passiver Betrachter, sondern kann vielmehr in der virtuellen Umgebung die Simulation beeinflussen und modifizieren. Die Integration in das Simulationsmodell und damit auch in das abgebildete System erhöht sein Prozessverständnis und schafft eine realistischere Planungsumgebung.

***Planung, Evaluierung und fortwährende Verbesserung aller Fertigungsprozesse über alle Planungs- und Ausführungsphasen bis zur Rückkopplung in die Fertigungslenkung***

Die Planung und Evaluierung der Fertigungsprozesse unter Einsatz der Ablaufsimulation soll vermehrt ganzheitlich über alle Planungs- und operativen Phasen eines Fertigungsprozesses hinweg erfolgen, weil nur so eine durchgängige Analyse und Verbesserung des Prozesses erreicht werden kann. Eine dreidimensionale Modellierung und Simulation als Kombination von Layout- und Prozessplanung kann den Anwender beim Aufbau des Fertigungsprozesses zusätzliche Planungsrestriktionen erkennen und von Beginn an berücksichtigen lassen. Die Qualität der Gesamtplanung kann weiter verbessert und zusätzliche Planungszeit innerhalb des Gesamtprojektes eingespart werden.

Neben dem Einsatz im Rahmen von Machbarkeitsstudien, Variantenplanungen oder quantitativen Fragestellungen, in denen ein System meist hinsichtlich seiner maximalen Leistungsfähigkeit oder eines optimalen Durchsatzes untersucht wird, muss die Simulation nach diesem ganzheitlichen Verständnis auch Fertigungsprogramme planen oder zumindest absichern können. Ein Simulationsmodell eines Fertigungsprozesses kann dann über alle Phasen der Struktur-, Mengen-, Kapazitäts- und Programmplanung bis hin zur Prognose und der fortlaufenden Verbesserung vorhandener Fertigungsprozesse eingesetzt werden.

Heutige Simulatoren bilden den Materialfluss ausschließlich vorwärts gerichtet ab. Ausgehend von zu parametrierenden Eingangsparametern (in den Werkzeugen meist als Quellen bezeichnet) berechnet der Simulator dann in jedem Simulationslauf den *maximal* möglichen Output zu dem gegebenen Simulationsmodell.

**Definition 31: Vorwärtsterminierung**

> *„Vorwärtsterminierung ist die Methode, von einem Startzeitpunkt ausgehend die frühestens möglichen Zwischen- und Endtermine zu ermitteln.*" [DaWa97]

Dieses Verfahren der Vorwärtsterminierung in Bezug auf die Ausführung von Simulationsmodellen durch einen Simulator wird im weiteren Verlauf als *Vorwärtssimulation* bezeichnet.

Im Bereich der Fertigungslenkung haben sich für die integrierte Mengen-, Termin- und Kapazitätsplanung alternative Verfahren etabliert, um die Planungsphasen und –ergebnisse eines gegebenen Fertigungsprogramms zu optimieren. Viele dieser meist heuristischen Eröffnungs- und/oder Verbesserungsverfahren basieren auf der Methode der Rückwärtsterminierung. Ausgehend von bestehenden Endterminen (beispielsweise der Auslieferung eines Kundenauftrags) sollen die Fertigungsprogramme hinsichtlich Durchlaufzeit, minimalen Kosten und Systemstabilität optimiert werden.

**Definition 32: Rückwärtsterminierung**

> *„Rückwärtsterminierung ist die Methode, von einem vorgegebenen Liefertermin ausgehend die spätesten möglichen Zwischen- und Starttermine zu berechnen*" [DaWa97]

Viele der mittels der Ablaufsimulation abgesicherten Fertigungssysteme werden im Bereich der Fertigungslenkung mit Verfahren zur Rückwärtsterminierung beplant. Heute werden, wenn überhaupt eingesetzt, erst in einem Folgeschritt die Ergebnisse dieser Planungen durch eine Vorwärtssimulation nochmals abgesichert. Eine Verbesserung dieser Planungsschritte ließe sich durch den Einsatz der Ablaufsimulation erreichen, wenn die Richtung des abgebildeten Materialflusses umgekehrt wird. Ausgehend von einem Auftragsbestand mit gegebenen Endterminen berechnet der Materialflusssimulator die spätesten Beginn-Zeitpunkte der Aufträge, die das System durchlaufen. Dieses Verfahren wird im Folgenden *Rückwärtssimulation* genannt. Mittels der Rückwärtssimulation wäre ein Simulator in der Lage, Programmplanungen moderner, kundenorientierter Fertigungsprozesse direkt zu planen oder zumindest abzusichern und in Simulationsstudien zu optimieren. Der Mehraufwand für den Simulationsexperten ist möglichst gering zu halten. Neben der Möglichkeit einer vorwärts und rückwärts gerichteten Ausführung von Simulationsmodellen im Simulatorkern muss also insbesondere ein Verfahren identifiziert werden, mit dem die Richtung der Materialflussmodelle unter möglichst wenigen Arbeitsschritten umgekehrt werden kann. Neben der ereignisgesteuerten Ausführung der diskreten Simulationsmodelle soll eine zeitorientierte Ausführung in fixen Zeitinkrementen in Vor- oder Rückwärtsrichtung grundsätzlich unterstützt werden, um eine mögliche Integration in bestehende Leitstandsysteme zu erleichtern.

***Kooperative Planung innerhalb von Unternehmen, Unternehmensverbünden oder Supply-Chain-Netzwerken***

Großunternehmen, virtuelle Unternehmen oder Supply-Chain-Netzwerke fertigen bereits heute an unterschiedlichen Standorten mit aufeinander abgestimmten Prozessen. Um die Lieferfähigkeit innerhalb eines Supply-Chain-Netzwerks sowie die Stabilität aller einzelnen Fertigungsprogramme besser gewährleisten zu können, müssen die Planungen enger aufeinander abgestimmt und überwacht werden. Die Mehrbenutzerfähigkeit der angestrebten Ablaufsimulation ermöglicht eine kooperative Planung mehrerer Simulationsexperten an einem gemeinsamen Simulationsmodell des Fertigungsnetzwerks unabhängig vom Standort der jeweiligen Experten. Innerhalb einer Supply-Chain oder eines Unternehmensverbundes, aber auch standortübergreifend innerhalb einer Unternehmung können so auf Basis eines Rechtemanagements und umfangreicher Kommunikationsmechanismen Simulationsexperten aller Standorte gemeinsam Simulationsmodelle erarbeiten, verifizieren und validieren, Simulationsexperimente kooperativ ausführen und in einer einzigen, gemeinsamen Umgebung auswerten.

Da die verschiedenen Partner innerhalb solcher unternehmensinternen wie –externen Fertigungsnetzwerke nach unterschiedlichen Organisationsformen fertigen können, muss neben dem Werkzeug an sich auch die zugrunde liegende Modellbeschreibung Fertigungssysteme beherrschen, die sowohl ein besonders hohes Maß an Komplexität bieten als auch über rein objektorientiert gegliederte Fertigungssysteme hinausgehen und damit die Modellierung und Simulation von funktional gegliederten

Fertigungssystemen bzw. deren Mischformen erlauben. Auch in diesem Themenbereich bietet sich der Einsatz des Werkzeugs über die reinen Planungsphasen hinaus bis hin zur Umplanung und Einsteuerung in der Fertigungslenkung an.

Wird diese Strategie weiterverfolgt, so wird auch die Komplexität der Simulationsmodelle weiter steigen. Insbesondere vor dem Hintergrund einer interaktiven Ausführung eines Simulationsmodells durch mehrere Simulationsexperten in einer virtuellen Umgebung kann dies bei beschränkten Rechenressourcen zu Problemen führen. Die Modellbeschreibung und Ausführung von Simulationsexperimenten muss also um eine Funktion erweitert werden, mit der auch komplexe Simulationsexperimente in einer virtuellen Umgebung dargestellt werden können.

## 2.4 Anforderungsbeschreibung

Die Umsetzung der aufgezeigten Themenbereiche führt auf den unterschiedlichen Betrachtungsebenen der Ablaufsimulation zu Konsequenzen. Einzelne Funktionen des angestrebten Werkzeugs wirken sich auf die Art aus, wie Simulationsmodelle hierfür beschrieben werden müssen. Sie können aber auch dazu führen, den Prozess der Modellierung, Simulation und Analyse anders abzubilden, als das mit herkömmlichen Software-Werkzeugen in diesem Bereich möglich ist. Um alle in Abschnitt 2.3 aufgezeigten neuen Funktionen aufnehmen zu können, beschreibt dieser Abschnitt die Anforderungen entlang der drei Ebenen „Basisprozess", „Modellbeschreibung" und „Werkzeug", wie sie in Abbildung 3 dargestellt werden. Hier ist ebenfalls ersichtlich, dass die Entwicklung einer Modellbeschreibung auf dem Prozess der Modellierung, Simulation und Analyse fußt und das ein solides Werkzeug zur Lösung der aufgezeigten Anforderungen nur entwickelt werden kann, wenn die zugrunde liegende Modellbeschreibung die einzelnen Funktionen überhaupt unterstützt.

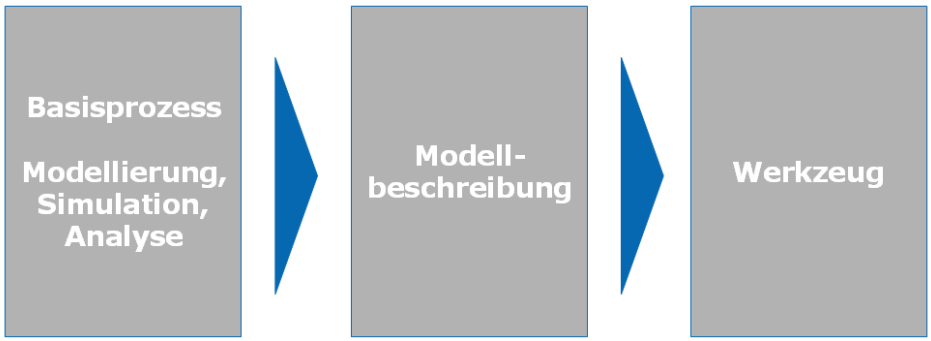


**Abbildung 3: Abhängigkeiten der Entwurfssichten**

### 2.4.1 Basisprozess

Der existierende Basisprozess von Modellierung, Simulation und Analyse muss um die Möglichkeit zur kooperativen Arbeit erweitert werden. Dazu müssen die einzelnen Arbeitsschritte in den Gesamtumfang einer Simulationsstudie eingebettet und hinsichtlich der Bearbeitung durch mehrere Anwender untersucht werden. Grundsätzlich soll die Vorgehensweise nicht revolutioniert, sondern nur den neuen Anforderungen angepasst werden. In einem ersten Schritt sollen die einzelnen Arbeitsschritte durch eine modulare Struktur des Werkzeugs voneinander entkoppelt werden. Auf Basis einer konsistenten und integrierten Datenhaltung wird damit die Arbeit mehrerer Anwender an einem

Simulationsmodell grundsätzlich ermöglicht. Die einzelnen Module des Werkzeugs, wie sie in Abbildung 4 skizziert werden, haben dann jeweils noch die Aufgabe, die Konsistenz innerhalb des Moduls während der Bearbeitung durch mehrere Anwender sicherzustellen.

Die Anforderung nach einer interaktiven Arbeitsumgebung führt zu der Konsequenz einer bidirektionalen Kopplung der Visualisierungkomponente(n) mit dem eigentlichen Simulator, der den Simulationslauf berechnet. Die Änderungen des Anwenders können so in den Simulator eingespielt und die Änderungen in der Visualisierung sofort nachvollzogen werden. Um Simulationsmodelle so einer interaktiven Analyse zuführen zu können, muss auch die zugrunde liegende Modellbeschreibung verschiedene Kriterien erfüllen, die im folgenden Abschnitt zusammen getragen werden sollen. Darüber hinaus müssen zur Kopplung der verschiedenen Funktionsmodule definierte Schnittstellen erstellt werden, die eine solche Vorgehensweise unterstützen.

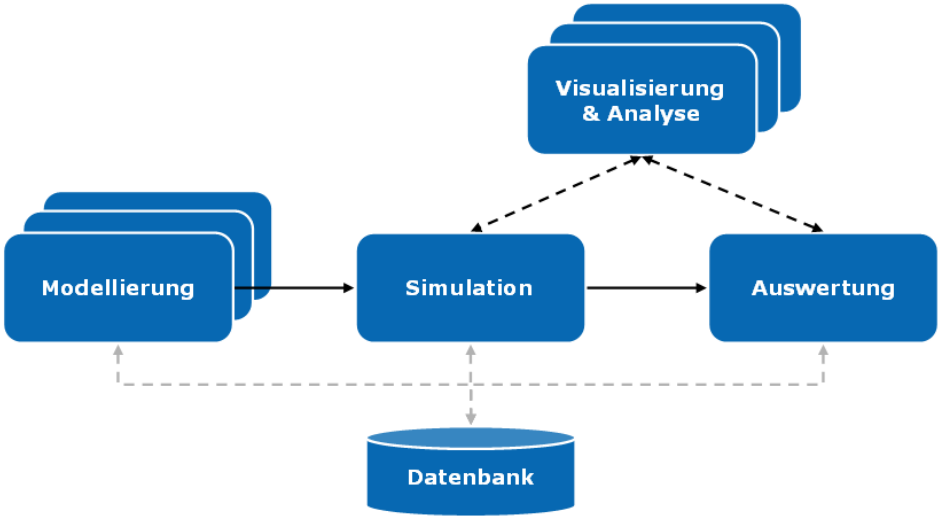


**Abbildung 4: Grundstruktur der Module im Basisprozess**

### 2.4.2 Modellbeschreibung

Die Art und Weise, wie Simulationsmodelle in dem zu entwickelnden Werkzeug beschrieben werden, hat Auswirkungen auf die überhaupt zu ermöglichenden Funktionalitäten der Software. Als grundlegende Datenstruktur, auf der ein Simulationslauf berechnet wird, liefert sie den Rahmen für die möglichen Aktivitäten des Anwenders sowohl auf der Modellierungs- als auch auf der Simulationsseite. Deshalb muss sie alle Funktionen und Möglichkeiten ermöglichen, die dem Anwender geboten werden sollen.

Für das hier zu entwickelnde Werkzeug muss die Modellbeschreibung zunächst einmal ein Fertigungssystem und seine Elemente abbilden können. Die menschliche Denkweise orientiert sich an einer Strukturierung in verschiedene Einzelteile. Die Modellbeschreibung soll deshalb in erster Linie Simulationsmodelle **modular** und im weitesten Sinne **objektorientiert** formalisieren. Die Genauigkeit, mit der ein Fertigungssystem in seine Teile zerlegt wird, soll vom Anwender frei wählbar sein. Prinzipiell soll also ein **beliebiger Detailgrad** erlaubt sein. Das ist insbesondere auch vor dem phasenübergreifenden Einsatz der Ablaufsimulation interessant, da sich bei der Planung von Fertigungssystemen der Detaillierungsgrad der Abbildung stetig verfeinert.

Dadurch wächst natürlich auch der Komplexitätsgrad des Modells. Zur weiteren Strukturierung sollen die Simulationsmodelle also **hierarchisierbar** sein. So kann eine Hierarchieebene auch einer jeweiligen Sicht auf das abzubildende Objekt entsprechen. Wird zu Beginn einer Planung ein Fertigungssystem beispielsweise noch als eine Zusammensetzung von 3 Kernabläufen betrachtet, so lassen sich diese im weiteren Projektverlauf leicht durch Hierarchieebenen ergänzen, die die einzelnen Teilbereiche genauer modellieren.

Die Simulationsmodelle sollen im Rahmen der Modellierung und Simulation von mehreren Anwendern bearbeitet werden können. Obwohl eine Synchronisierung der einzelnen Anwenderinteraktionen letztendlich durch die jeweiligen Module des Werkzeugs erfolgen muss, so muss die Modellbeschreibung eine solche **synchronisierte Interaktion** doch ermöglichen. Auch hier ist eine objektorientierte Darstellung im Vorteil, weil die betroffenen Objekte einzeln vor anderen Zugriffen gesperrt werden können.

Letztendlich soll mit diesem Werkzeug die Anwendung der Ablaufsimulation erweitert werden und so ist nur realistisch anzunehmen, dass auch zusätzliche Anforderungen an das Werkzeug und seine Modellbeschreibung gestellt werden. Die Modellbeschreibung muss deshalb in besonderem Maße auch **flexibel erweiterbar** sein. Eine der neueren Anwendungen ist schon jetzt bekannt: Die Rückwärtssimulation. Weil diese auf rückwärts gerichteten Simulationsmodellen arbeitet, die Zeit zur doppelten Modellierung eines Fertigungssystem aber sicherlich nicht vorhanden ist, muss die Modellbeschreibung darüber hinaus **richtungsoffen** sein, d.h. das Konvertieren von vorwärts gerichteten Simulationsmodellen in rückwärts-gerichtete Simulationsmodelle ermöglichen.

Neben diesen Anforderungen an die Modellbeschreibung gibt es darüber hinaus auch Einschränkungen, die für die zu konzipierende Modellbeschreibung nicht gelten sollen. Wegen der zu unterstützenden ereignis- oder zeitorientierten Ausführung in Vor- und Rückwärtsrichtung soll die Modellbeschreibung **keine festen Zeitangaben und/oder Zeiteinheiten** enthalten, sondern angegebene Parameter nur in relativ angegebenen Zeitpunkten beschreiben. Die jeweilige Berechnung der Simulationszeit hängt dann nicht zuletzt auch von dem verwendeten Verfahren im Simulatorkern ab und spielt für das Beschreiben der zugrunde liegenden Simulationsmodelle keine Rolle.

In Summe kann die Modellbeschreibung daraufhin die Datenstruktur der Anwendung liefern, die mit dem Werkzeug entwickelt werden soll und in dem die eigentlichen Funktionen implementiert werden.

### 2.4.3 Werkzeugentwicklung

Das Werkzeug selbst soll mit modernen Methoden des Software-Designs entwickelt werden. Neben einer objektorientierten Programmierung ist dafür insbesondere auch die Definition von erweiterbaren Schnittstellen und einer modularen Programmstruktur von großer Bedeutung. Somit können auch einzelne Funktionen des Werkzeugs später mit relativ geringem Aufwand durch Weiterentwicklungen ersetzt werden.

Die einzelnen Module der Gesamtanwendung strukturieren sich in Anlehnung an den zu unterstützenden Arbeitsprozess des Anwenders, sind also im besten Sinne anwendungs- und prozessorientiert. Neben einem Werkzeug zum Aufbau der Simulationsmodelle muss

das Werkzeug einen Simulatorkern oder Kernel enthalten sowie minimal eine Visualisierungskomponente zur Analyse der modellierten Abläufe und deren dynamischen Verhaltensweisen in dem Simulationsmodell. Zur konsistenten Datenhaltung empfiehlt sich darüber hinaus die Verwendung einer Datenbankstruktur. Sie erlaubt einen schnellen Zugriff auf die anfallenden Datenmengen (Simulationsmodelle, Experimentdaten, 3D-Modelle etc.)

Die Anforderungen an die einzelnen Module sind aus Abschnitt 2.3 bekannt und sollen deshalb an dieser Stelle nur kurz aufgelistet werden. Sie sollen im besonderen Maß als Anforderungen für die Entwicklung des Werkzeugs dienen.

In Summe bilden die dargestellten Anforderungen an Basisprozess, Modellbeschreibung und Werkzeug die Gesamtanforderungen an diese Arbeit. In einem nächsten Schritt muss nun der Stand der Technik hinsichtlich existierender Lösungen oder Lösungsideen untersucht werden. Das soll im folgenden Kapitel 1 geschehen.

| Modul | Funktion |
|---|---|
| Modellierung | ▪ Synchronisierte Multitasking-Modellierung<br>▪ Beliebiger Detailgrad<br>▪ Hierarchische, modulare und objektorientierte Modelle<br>▪ Umkehrung von Simulationsmodellen<br>▪ Einfache Modellierung von Transportwegen für funktional gegliederte Fertigungssysteme |
| Simulation | ▪ Dynamische Detaillierung<br>▪ Interaktiv veränderbar<br>▪ Ereignis- & zeitorientierte Simulation in vorwärts und rückwärts Richtung |
| Visualisierung | ▪ Interaktive 3D-Darstellung des Materialflusses aus der Simulation<br>▪ Synchronisierte Multitasking-Simulation für mehrbenutzerfähige Analyse<br>▪ Immersive Visualisierung in virtueller Umgebung |

**Tabelle 1: Anforderungen aus den neuen Einsatzfeldern der Ablaufsimulation**

# 3 Stand der Technik

„Zähmen sollen sich die Menschen,
die sich gedankenlos der Wunder der
Wissenschaft und Technik bedienen und
nicht mehr davon geistig erfasst haben
als die Kuh von der Botanik der Pflanzen,
die sie mit Wohlbehagen frisst."

*(Albert Einstein)*

Nachdem im vorigen Kapitel die Anforderungen an diese Arbeit zusammengetragen wurden, soll nun Stand der Technik hinsichtlich dieser Fragestellungen untersucht werden. Welche Lösungsansätze sind in den verschiedenen Bereichen bereits bekannt und inwiefern können sie helfen, die gestellten Anforderungen umzusetzen?

Analog zur Strukturierung der Anforderungsbeschreibung soll auch der Stand der Technik Schritt für Schritt untersucht werden. In einem ersten Schritt muss zunächst untersucht werden, welche Prozesse sich bei der Durchführung einer Simulationsstudie etabliert haben und inwiefern sich dieses Vorgehen auch in einer mehrbenutzerfähigen und interaktiven Arbeitsumgebung realisieren lässt. Abschnitt 3.1 soll dazu die bekannte Methodik zur Durchführung einer Simulationsstudie vorstellen und bewerten.

Weil im Folgeschritt dieser Arbeit darauf aufbauend eine Modellbeschreibung konzipiert werden soll, widmen sich die Abschnitte 3.2 und 3.3 der Bewertung von bekannten Methoden zur Modellierung und Ausführung von diskreten Simulationsmodellen. Danach wird in Abschnitt 3.4 zusätzlich untersucht, inwiefern sich speziell virtuelle Umgebungen zur Visualisierung von Ablaufsimulationen eignen und welche Vorraussetzungen dafür erfüllt sein müssen.

Erst im Folgeschritt widmet sich der Abschnitt 3.5 den Methoden des Software-Designs zur Entwicklung des Werkzeugs. Die Erkenntnisse aus Basisprozessgestaltung und der Modellbeschreibung sollen hier in die Bewertung mit einfließen. Der Abschnitt zeigt allgemeine und objektorientierte Prinzipien des Software-Designs auf, fokussiert im Speziellen auf Organisationsformen und Architekturmuster von Mehrbenutzersystemen sowie Prinzipien zur Schnittstellengestaltung, jeweils vor dem Hintergrund der gestellten Anforderungen.

Abschließend fasst Abschnitt 3.6 den Stand der Technik hinsichtlich der Fragestellung zusammen, so dass im folgenden Kapitel die Zielstellung der Konzeption und Implementierungsphase formuliert werden kann.

## 3.1 Simulationsmethodik

Die Simulation von Fertigungssystemen hinsichtlich einer bestimmten Aufgabenstellung wird laut Definition 21 als Simulationsstudie bezeichnet. [VDI3633] beschreibt eine Vorgehensweise bei der Durchführung von Simulationsstudien, die sich in diesem Themenfeld etabliert hat. Sie soll nachfolgend näher beschrieben werden. Im

Wesentlichen gliedert sich die Vorgehensweise in die Schritte Vorbereitung, Modellierung, Durchführung und Auswertung. Insbesondere die Modellierung und Durchführung von Simulationsexperimenten müssen iterativ durchgeführt werden, um die erstellten Simulationsmodelle zu verifizieren, zu validieren und ggf. alternative Szenarien zu erarbeiten.

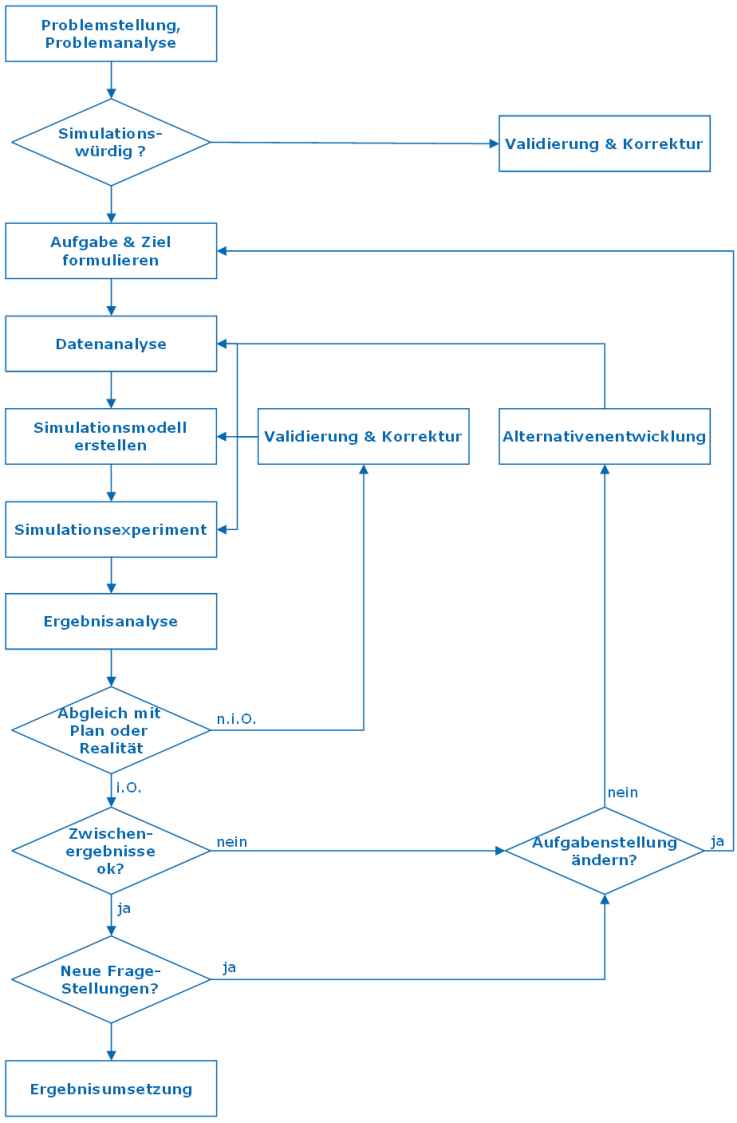


**Abbildung 5: Ablauf einer Simulationsstudie nach [VDI3633]**

Ausgangspunkt jeder Simulationsstudie ist demnach die exakte Festlegung eines Untersuchungsziels, welches das grundlegende Problem inklusive einer grundsätzlichen Beschreibung des Vorgehens und Lösungsmöglichkeiten eingrenzt. Auf Basis des Untersuchungsziels kann die nachfolgende Phase der Datenaufbereitung erfolgen. Beschaffung, Aufbereitung und ggf. Anpassung der Daten ist oftmals mit hohem Aufwand verbunden, da sie nur selten bereits in der benötigten Form vorliegen. In manchen Fällen liegen über das abzubildende System noch gar keine Daten vor (wenn beispielsweise das Fertigungssystem in der Realität noch nicht existiert). In diesem Fall müssen die benötigten Daten bestmöglich abgeschätzt werden. Insbesondere räumliche, zeitliche und mengenmäßige Daten sind als Eingabeparameter für das Simulationsmodell interessant. Wie oben bereits erwähnt, sind die vorliegenden Daten auf Plausibilität und

Richtigkeit zu prüfen, da nur dann auch vernünftige Ergebnisse aus der Simulationsstudie zu erwarten sind.

Im Folgeschritt wird das Fertigungssystem zunächst als logisches, später dann als experimentierfähiges Modell in einem Simulator abgebildet. Das Simulationsmodell muss alle für die Fragestellung relevanten Sachverhalte repräsentieren, wobei alle Elemente und Strukturen, Regeln, Einflüsse und Verhaltensweisen aus dem realen System in dem benötigten Detaillierungsgrad abzubilden sind. Durch die ersten Simulationsläufe mit dem experimentierfähigen Simulationsmodell wird das Simulationsmodell bezüglich der inneren Logik hin verifiziert und im nächsten Schritt hinsichtlich des Systemverhaltens der Realität bestmöglich angepasst (Modellvalidierung). Das Vorgehen erfolgt hierbei iterativ bis der gewünschte Genauigkeitsgrad erreicht werden konnte.

Die nächste Phase beinhaltet nun verschiedene Schritte der Durchführung von Simulationsläufen im experimentellen Rahmen. Zunächst werden einfache Analysen gefahren, um erste Aussagen zur Leistungsfähigkeit und Problembereiche des Fertigungssystems zu erhalten. Anschließend werden einzelne Experimentreihen durchgeführt, die entweder die Ausplanung des Fertigungssystems betreffen oder Sensitivitätsanalysen enthalten, die die Abhängigkeit des Gesamtsystems von einzelnen Eingabefaktoren herausarbeiten. Dieser iterative Prozess wird so lange durchgeführt, ggf. mit Anpassungen des Simulationsmodells, bis schließlich für den Anwender hinsichtlich der Aufgabenstellung zufrieden stellende Ergebnisse vorliegen. Den Abschluss der Simulationsstudie bildet die Dokumentation, Auswertung und Aufbereitung der gesammelten Ergebnisse. Die Simulationsergebnisse werden hier zu aussagefähigen Informationen verdichtet, interpretiert und angemessen dargestellt. Hierbei muss jeweils durch den Anwender entschieden werden, inwiefern sich ein Rückschluss von den Simulationsergebnissen auf das reale Verhalten des abgebildeten Fertigungssystems erlaubt.

Bei dem angestrebten Basisprozess zur Modellierung und Simulation soll sich an der für Simulationsstudien geltenden Vorgehensweise orientiert werden. Die Darstellung der Vorgehensweise bei einer Simulationsstudie schränkt auch den in Abschnitt 2.4.1 anvisierten Basisprozess der Modellierung, Simulation und Analyse in einer mehrbenutzerfähigen Umgebung nicht ein, weil keine klare Aussage darüber getroffen wird, wie die Modellbildung möglichst effektiv und effizient zu gestalten ist. So bildet die vorgeschlagene Vorgehensweise nur den inhaltlichen Rahmen, innerhalb dem der Basisprozess umgesetzt werden soll. Der Prozess der Modellvalidierung und Modellverifikation soll aber durch den hohen interaktiven Grad des Werkzeugs verkürzt werden. Weil der Anwender das Simulationsmodell während der Ausführung manipulieren kann, sollte schneller ein verbessertes Modell erstellt werden können. Die eigentliche Experimentphase einer Simulationsstudie muss dennoch ohne Benutzerinteraktion durchgeführt werden, um eine statistisch auswertbare Datenbasis aus dem Simulationsmodell generieren zu können. Dafür sollte jeder einzelne Simulationslauf möglichst schnell ausgeführt werden können. Bei der angestrebten Ausführung im Simulator, bei der nur nach Anforderung eine Visualisierungskomponente als eigenständiges Modul angekoppelt wird, sollte das in der bisher geplanten, modularen Bauweise gut unterstützt werden.

## 3.2 Modellierungsmethoden

Simulation beschreibt immer ein Arbeiten mit und an experimentierfähigen Modellen, meist eingebettet in die Durchführung einer Simulationsstudie. Damit ist die Struktur der abzubildenden Fertigungssysteme zumindest hinsichtlich des Untersuchungszwecks bekannt. In einem ersten Schritt muss also das abzubildende System als ein Modell beschrieben werden. Dieses muss in dem Werkzeug gemäß einer noch festzulegenden Modellbeschreibung erzeugt und manipuliert werden können. Dieser Abschnitt zeigt verschiedene Modellierungsmethoden und deren Anwendung auf Materialflusssimulation und bewertet sie vor dem Hintergrund der an eine Modellbeschreibung gestellten Anforderungen.

Die „Kunst" innerhalb dieses Modellierungsprozesses ist es, eine befriedigende Mischung zwischen dem Abstraktionsgrad, der Richtigkeit und der Genauigkeit eines Modells und der Wirklichkeit zu finden. Der Zielkonflikt, der zwischen diesen drei Aspekten besteht, soll minimiert werden. Der Abstraktionsgrad beschreibt das äußere „Erscheinungsbild" eines Modells. Je weiter der Abstraktionsgrad fortschreitet, desto weniger realitätsnah wird das Modell. Ein fortschreitender Abstraktionsgrad bedeutet aber auch einen Gewinn an Genauigkeit, da der Fokus immer weiter auf den zu analysierenden Teil gelenkt wird. Die Nutzbarkeit des Modells wiederum wird an der Richtigkeit gemessen.

Es existieren zahlreiche Methoden zur Modellierung von Materialflüssen [ScWe00]. In dem nachfolgenden Abschnitt sollen deshalb zunächst Stellen/Transitionsnetze als eine Ausprägung formaler Beschreibungen von Simulationsmodellen untersucht werden. In einem weiteren Schritt sollen unter Abschnitt 3.2.2 Programmier- und Simulationssprachen hinsichtlich ihrer Eigenschaften untersucht und bewertet werden. Abschließend werden existierende Software-Lösungen in Form von grafischen Simulationswerkzeugen und Bausteinkästen untersucht, um das Spektrum möglicher Lösungsalternativen an Modellierungsmethoden möglichst gut abzudecken.

### 3.2.1 Stellen/Transitionsnetze

Stellen/Transitionsnetze sind als formale Beschreibungen zur Modellierung und anschließender Analyse von asynchronen Prozessen weit verbreitet (vgl. [LeEg88], [Star90]). ihre formale Basis ist auf der allgemeinen Netztheorie begründet und es existieren Erweiterungen zur Abbildung komplexerer Systeme, beispielsweise zur Integration stochastischer Prozesse [LeEg88], zur Abbildung hierarchischer Strukturen oder der Individualisierung (Färbung) der Marken [LeEg88], die durch den bipartiten Graphen laufen. Stellen/Transitionsnetze können unterschiedlich visualisiert werden. Üblicherweise werden Stellen als Kreise, Transitionen als ein Rechteck oder als eine Linie visualisiert, die orthogonal zu den eingehenden und ausgehenden Verbindungen steht. Flussrelationen werden mit Pfeilen zwischen den Stellen und Transitionen dargestellt (vgl. Abbildung 6). Mit diesen Netzen können Zustände von Montage-, Kommissionier- oder Transportvorgängen modelliert werden. Insb. geteilte Ressourcen von nebenläufigen Prozessen können durch diese Netze gut modelliert werden [Schm92]. Durch die nicht deterministische Reihenfolge der Schaltungen bieten sich Petri-Netze zur Analyse und Vermeidung von Verklemmungen an [Star90]. Die Planung der Schaltungen und die damit implizite Planung von der Benutzung der geteilten Ressourcen kann für das reale Problem übernommen werden.

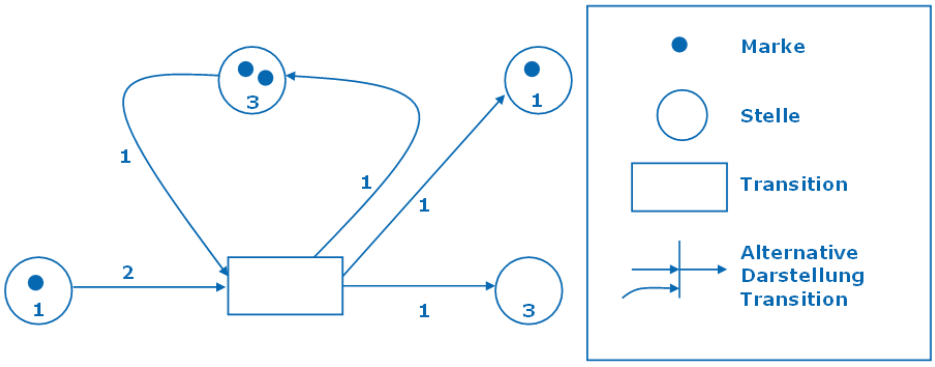


**Abbildung 6: Beispiel eines Petri-Netzes**

Stellen/Transitionsnetze sind von sich aus nicht zeitbewertet. Es existieren jedoch erweiterte, bzw. angepasste Beschreibungen, beispielsweise MFert, in denen eine solche Zeitbewertung verwendet wird (vgl. [Schn96]). Das MFert-Modell unterscheidet Faktoren und Faktortransformationen. Ein Graph des Fertigungsablaufs repräsentiert die Menge möglicher Fertigungsprozesse. In diesem Graphen sind die Faktorknoten und die Faktortransformationsknoten die Knoten. Die Kanten symbolisieren die möglichen Faktorströme.

**Anwendung von Stellen/Transitionsnetzen zur Modellierung von Materialflüssen**
Mit Stellen/Transitionsnetzen lassen sich Materialflussprozesse modellieren (vgl. [GiVa03] und [Dang99]). Insb. Ressourcen, die geteilt werden müssen, lassen sich so in ein Modell fassen. Muss z.B. ein Arbeiter zwei Maschinen bedienen, kann die Ressource Arbeiter als eine Marke, die beiden Fertigungsprozesse als jeweils eine Transition, der Materialzugang bzw. -abgang als jeweils eine Stelle und die Teile als Marken modelliert werden. Gibt eine Maschine den Arbeiter frei, muss die Marke auf eine Stelle gelagert werden, auf die beide Maschinen zugreifen können (vgl. Abbildung 7). Handelt es sich um eine zeitbewertete Erweiterung eines Stellen/Transitionsnetzes können die Zeiten den Bearbeitungsdauern der Maschinen entsprechen. Es entsteht ein Modell, welches den beschriebenen Prozess und den darin enthaltenen Ressourcenkonflikt abbildet.

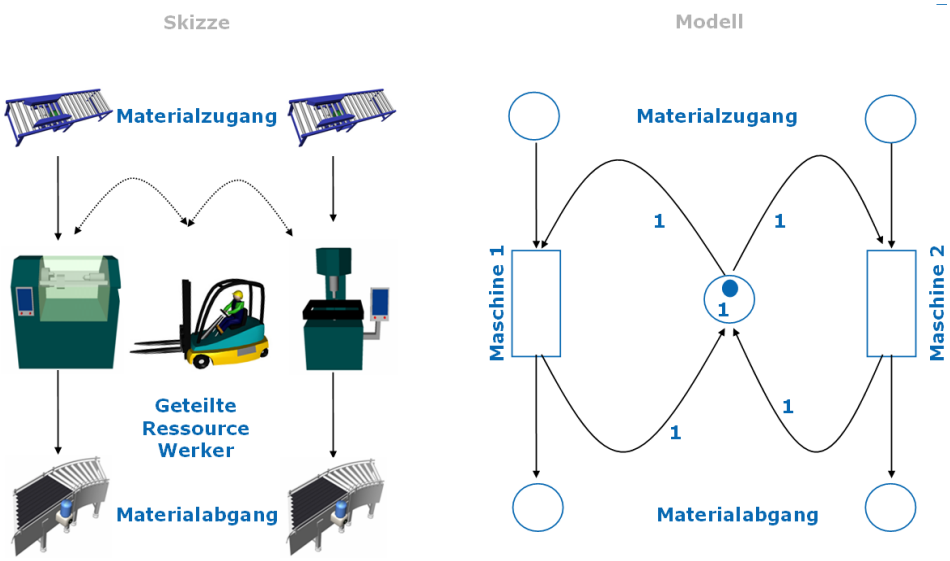


**Abbildung 7: Petri Netz Modell eines Arbeiters, der zwei Maschinen bedient**

Auf Basis von Stellen/Transitionsnetzen sind zahlreiche weitere formale Modellierungsmethoden bekannt, die sich zumeist jedoch wieder auf Stellen/Transitionsnetze zurückführen lassen.

**Bewertung**

Stellen/Transitionsnetze beschreiben Fertigungssysteme auf Basis eines bipartiten Graphen. Es wird bei der formalen Beschreibung also explizit zwischen der Speicherung und der Veränderung von Marken unterschieden. Diese können zwar hierarchisch modelliert und aus verschiedenen Substrukturen zusammengesetzt werden, eine Abbildung der in der Realität vorhandenen Fertigungssysteme wird durch eine fehlende systematische Gliederung und Objektorientierung jedoch unnötig erschwert. Je nach gewähltem Detaillierungsgrad unterscheidet der Anwender beispielsweise bei einem Lager nicht mehr zwischen den einzelnen Lagerplätzen und deren übergeordneten Lagersteuerung. Eine Abbildung durch ein Stellen/Transitionsnetz ist damit nur noch schwer möglich. Auch für die Abbildung von komplexen Steuerungen und insbesondere deren Umkehrung vor dem Hintergrund einer richtungsoffenen Ablaufsimulation erscheinen Stellen/Transitionsnetze als wenig geeignet, weil die jeweilige Umkehrung von Verteilregeln im Materialfluss durch eine implizite Modellierung die Komplexität der Stellen/Transitionsnetze stark ansteigen würde. Eine übersichtliche layoutgetreue Modellierung wird dadurch unnötig erschwert.

### 3.2.2 Programmier- und Simulationssprachen

Für eine rechnerunterstützte Simulation von Materialflüssen existieren neben den bisher vorgestellten formalen Beschreibungen eine Reihe spezifischer Verfahren. Zur Modellierung und Ausführung werden sowohl allgemeine Programmiersprachen als auch spezialisierte Simulationssprachen verwendet.

Während allgemeine Programmiersprachen (z.B. C++ [Brey05], Java [Ulle05]) sehr flexibel im Einsatz zur Simulationen sind, erfordern sie einen hohen Einarbeitungs- und Modellierungsaufwand. Sie bieten keine spezielle Unterstützung für Simulationen. Sollen abstrakte Methoden zum Einsatz kommen, müssen diese explizit implementiert werden.

Der spezielle Einsatz von Java als Simulationssprache für ein Simulations-Framework (vgl. 3.5.5.3) ist durch DESMO-J oder DEMOS bekannt ([PaKr05], [Birt82]). Auf Basis einer festgelegten Modellbeschreibung können Simulationsmodelle textbasiert programmiert und ausgeführt werden. Die Umsetzung des Frameworks erlaubt aber keine Erweiterung der vorhandenen Simulatorfunktionen, wie sie für das zu entwickelnde Werkzeug benötigt werden, denn beide Frameworks bieten keine Mehrbenutzer- oder Multitasking-Fähigkeit. Auch sind keine Modellierungs- oder Simulationsframeworks bekannt, mit denen Simulationsmodelle gemäß der spezifischen Beschreibung erstellt, ausgeführt und visualisiert werden können.

Den aus allgemeinen Programmiersprachen resultierenden hohen Aufwand bei der Implementierung reduzieren simulationsspezifische Sprachen (z.B. MODSIM, SIMULA, SIMSCRIPT, GPSS [Boss92], [Holm-ol], [Sims-ol], [Chis92]). Sie unterstützen den Anwender mit simulationsspezifischen Konstrukten und Bibliotheken. Obwohl ihr Einsatz einfacher als der der allgemeinen Programmiersprachen ist, erfordert ihre Anwendung im Allgemeinen dennoch Programmierkenntnisse.

**Bewertung**
Ebenso wie bei den bekannten Implementierungen in Frameworks von allgemeinen Programmiersprachen ist der Anwender beim Einsatz stets gezwungen, sich den jeweiligen Restriktionen der Bibliotheken zu beugen. Dadurch wird die Implementierung eines Mehrbenutzerbetriebs, einer dynamischen Detaillierung und einer dreidimensionalen Visualisierung verhindert. Die meisten Frameworks ermöglichen trotz modularem Aufbau und Objektorientierung auch keine interaktive Analyse, weil Datenstruktur und Berechnung voneinander getrennt sind. Sie verfolgen darüber hinaus jeweils nur ein Konzept der Zeitfortschaltung bei der Ausführung von Simulationsmodellen (vgl. Abschnitt 3.3).

Zur Lösung der hier gestellten Anforderung erweisen sich die bekannten Implementierungen also als nicht verwendbar. Die grundlegende Idee solcher Frameworks sollte jedoch in der Konzeptionsphase aufgegriffen werden, da bereits implizit durch die Gestaltung der Modellbeschreibung in einem Framework eine erste Gültigkeitsprüfung der Simulationsmodelle erfolgen kann. Dem entgegen steht der Nachteil einer hohen Einarbeitung und geringen Benutzerunterstützung. Für die Modellierung und Simulation müssen grafische Benutzeroberflächen gestaltet werden, um den oder die Anwender bei ihrer Arbeit besser zu unterstützen. Im folgenden Abschnitt sollen zur Komplettierung der möglichen Modellierungsmethoden auch existierende grafische Werkzeuge untersucht werden.

### 3.2.3 Grafische Simulationswerkzeuge

Anwendungsspezifische Simulatoren geben dem Modellierer auf eine spezielle Anwendung oder Domäne zugeschnittene Werkzeuge und Bausteine an die Hand und bieten grafische Modellierungswerkzeuge (z.B. Simul8, ED-Falcon, Quest oder Plant Simulation [Simu03], [EDFa-ol], [Deml-ol], [UGS-ol]). Modelle können durch Kombination, Parametrisierung und Verbindung von Bausteinen aus einer Bibliothek erstellt werden. Diese Modelle können als neue Bausteine für die Bibliothek dienen.

Durch diese Hierarchien können auch große, komplexe Modelle erstellt und gewartet werden. Für eine hohe Flexibilität enthalten diese Simulatoren oftmals eine Programmierschnittstelle oder/und eine eingebaute Programmiersprache. Der Einsatz grafischer Werkzeuge vereinfacht und beschleunigt das Erstellen von Simulationsmodellen. Die visuellen Modelle sind einfacher zu kommunizieren, ihre Erstellung ist intuitiver und die generierten Ergebnisse können Nicht-Simulationsexperten leichter kommuniziert werden. Im Gegenzug hierfür sind die grafischen Methoden nicht so flexibel und vielseitig wie allgemeine Programmiersprachen. Simulationsberechnungen bieten das Potential schneller zu sein, wenn der Ablauf direkt in einer allgemeinen Programmiersprache speziell für das Problem implementiert und optimiert worden ist (vgl. [DMD03]).

**Bewertung**
Mit den vorgestellten Werkzeugen ist es den Anwendern möglich, Modelle für den Fertigungsprozess in der jeweiligen Modellbeschreibung zu erzeugen und in einer zugehörigen Simulationsumgebung anzuwenden. Viele der kommerziellen Werkzeuge basieren auf den bekannten formalen Beschreibungen (vgl. Abschnitt 3.2.1) und beinhalten eine eigene, simulationsspezifische Programmiersprache, wie sie exemplarisch in Abschnitt 3.2.2 vorgestellt wurde, um das Verhalten der vorgegebenen Modellbausteine präzisieren zu können und somit den gewählten Detaillierungsgrad der Bibliotheken zumindest teilweise zu umgehen. Dennoch werden die Möglichkeiten des Anwenders hier unnötig eingeschränkt, was eine freie Wahl des gewünschten Detaillierungsgrades verhindert.

Der Prozess der Modellierung, Ausführung, Analyse und Modifikation der einzelnen Simulationsmodelle findet am Computer einer einzelnen Person statt. Möchten mehrere Anwender ein Modell bearbeiten, kann dies nur sequentiell durchgeführt werden. D.h., dass zu einem Zeitpunkt nur eine Person an dem Modell arbeiten kann. Die vorhandenen Software-Lösungen ermöglichen nur in Teilen eine interaktive Beeinflussung des Simulationslaufs.

Hinsichtlich einer Visualisierung in einer virtuellen Umgebung existieren Arbeiten, die die vorhandenen Software-Werkzeuge anbinden können. Die Werkzeuge selbst stellen hier (noch) nur unzureichende Lösungen bereit. Keines der bekannten Werkzeuge ist in der Lage, Simulationsmodelle zeitlich rückwärts zu berechnen, bzw. vorwärts-gerichtete Modelle umzukehren. Spezielle Methoden zur Unterstützung von funktional gegliederten Fertigungssystemen sind ebenfalls nicht bekannt. In jedem Fall ist der Anwender an die spezifische Implementierung des Simulatorkerns gebunden, womit sich beispielsweise eine dynamische Detaillierung verbietet.

## 3.3 Ausführung von Simulationsmodellen

Die gemäß einer gültigen Modellbeschreibung erstellten Simulationsmodelle müssen zur Durchführung einer Simulationsstudie schließlich auch im Simulatorkern ausgeführt werden. Für das hier zu entwickelnde Werkzeug wird eine Ausführung in Vorwärts- und Rückwärtsrichtung angestrebt. Die beiden folgenden Abschnitte betrachten den Stand der Technik bei den jeweiligen Ausführungsformen detaillierter.

### 3.3.1 Vorwärtssimulation

Wie in Abschnitt 2.2 bereits angerissen existieren zur Zeitfortschaltung während der Simulation dazu zwei grundlegende Verfahren, die richtungsoffen ausgeführt werden können: Fortschaltung nach fixen und variablen Zeitinkrementen.

**Fixe Zeitinkremente**

Bei fixen Zeitinkrementen wird die zu simulierende Zeitspanne in gleichförmige Intervalle zerteilt. Für alle Intervalle wird einheitlich ein Ereigniszeitpunkt zugeordnet (Anfang, Mitte oder Ende des Intervalls). Für jedes Intervall werden die darin auftretenden Ereignisse gespeichert, als würden sie an dem Intervall zugeordneten Zeitpunkt ausgeführt werden müssen. Die Berechnung versucht, beginnend beim zeitlich ersten Intervall, alle darin enthaltenen Ereignisse abzuarbeiten. Eine implizite Diskretisierung findet statt. Werden hierbei neue Ereignisse erzeugt, werden diese innerhalb der Ereignisliste in das jeweilige Intervall einsortiert.
Wird die Intervalllänge kurz gewählt, müssen mehr Intervalle verwaltet werden. Ist die Anzahl der Intervalle deutlich größer als die Anzahl der auftretenden Ereignisse, muss die Simulation viele leere Intervalle verwalten und auch beim Voranschreiten betrachten. Die eigentliche Berechnung der Simulation wird somit verlangsamt. Wird die Intervalllänge groß gewählt, findet die Diskretisierung sehr grob statt. Die Differenz zwischen dem Zeitpunkt, an dem ein Ereignis stattfinden soll, und dem Zeitpunkt, der dem zugeordneten Intervall zugehört, kann groß werden. Daraus ergibt sich unter Umständen ein großer Fehler in der Berechnung. Durch die Größe der Intervalle enthalten diese im Durchschnitt mehr Ereignisse, die innerhalb des Intervalls sequentiell oder zumindest synchronisiert berechnet werden müssen. Verfahren zum Auflösen dieser Gleichzeitigkeit kommen demnach vermehrt zum Einsatz und führen ebenfalls zu einer ungenaueren Berechnung. Abbildung 8 zeigt im linken Teil die Zeitfortschaltung mittels fixer Zeitinkremente. Im Gegensatz dazu zeugt der rechte Teil der Abbildung die Zeitfortschaltung mittels variabler Zeitinkremente, die nachfolgend näher erläutert wird.

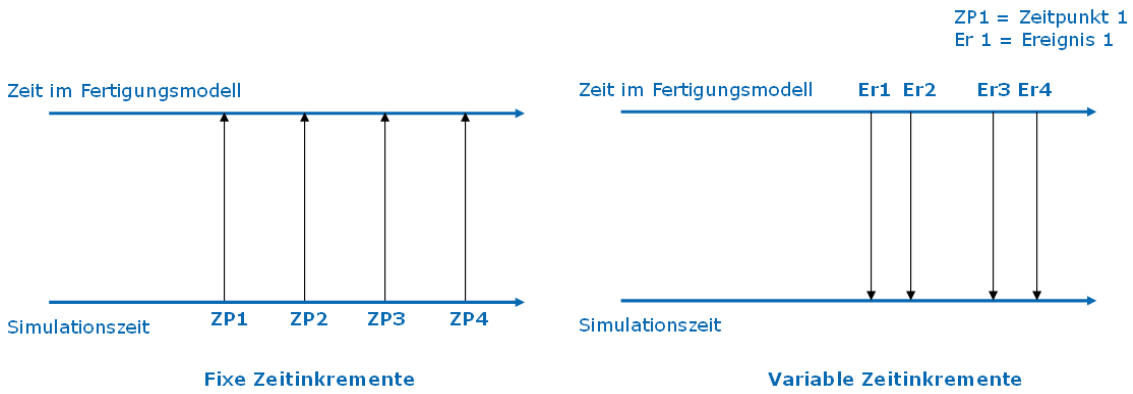


**Abbildung 8: Zeitfortschaltung mit fixen und variablen Zeitinkrementen**

**Variable Zeitinkremente**

Methoden die mit variablen Zeitinkrementen arbeiten haben keine vorbestimmten Zeitinkremente, in denen sie voranschreiten. Die Systeme springen direkt zum jeweils zeitlich nächsten Ereignis im Fertigungsmodell. Die Menge der in der Zukunft zu berechnenden Ereignisse wird sortiert in der Ereignisliste gespeichert. Die Berechnung muss das nächste Ereignis aus der Datenstruktur entnehmen, die eigene Uhr auf die Zeit

des Berechnungszeitpunktes des Ereignisses setzen und berechnen. Neue Ereignisse werden wieder in die Datenstruktur eingefügt (vgl. Abbildung 9). Im Fall einer Ausführung des Simulationslaufs in Echtzeit oder einem Faktor der Echtzeit (beispielsweise 4-mal so schnell) muss die Ausführung der Ereignisse noch mit der externen Simulationszeit des verwendeten Kalenders synchronisiert werden.

Oftmals sollen mit Simulationen Fertigungssysteme mit beschränkten Ressourcen modelliert werden. Ereignisse können nur in einzelnen Zuständen ausgeführt werden. So kann sich z.B. eine Warteschlange in einem Puffer vor einer belegten Maschine bilden. Ein Ereignis zum Umlagern des vordersten Objekts in der Warteschlange auf die Maschine muss warten, bis diese den Zustand frei erreicht. Für die Berechnung der Simulation bedeutet dies, dass nach dem Freiwerden der Maschine untersucht werden muss, ob bisher blockierte Ereignisse nun ausgeführt werden können[3]. Abbildung 9 zeigt schematisch die Ausführung eines diskret ereignisorientierten Simulationsmodells in einem Simulator [Dang03].

In diesem Werkzeug sollen durch eine sinnvolle Konstruktion des Simulatorkerns und der Modellbeschreibung grundsätzlich beide Möglichkeiten der Zeitfortscheitung unterstützt werden.

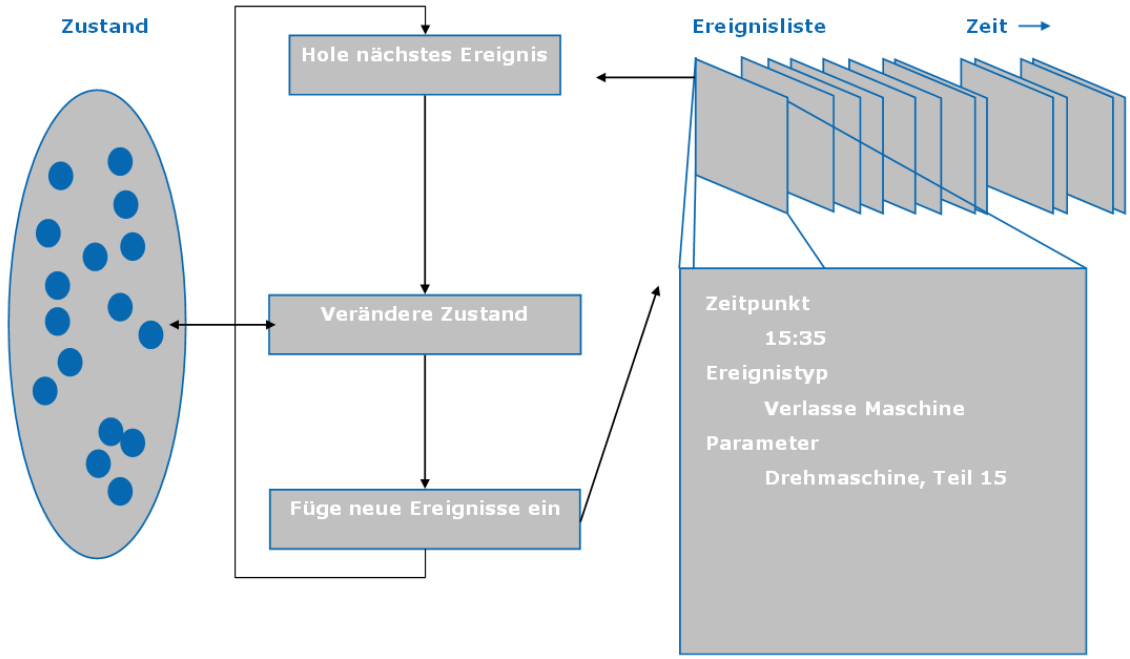


**Abbildung 9: Ablauf einer ereignisgesteuerten Simulation**

**Bewertung**

Beide Methoden können und sollen in die Implementierung des Simulatorkerns einfließen. Die Anforderung nach einer interaktiven Beeinflussung des Fortlaufes der Simulation unterstreicht die Notwendigkeit einer bidirektionalen Kopplung zwischen Simulatorkern und der entsprechenden Visualisierungskomponente über eine erweiterbare Schnittstelle. Die Berechnungen des Simulators müssen nicht nur angezeigt,

[3] Evans [Evan88] bietet hierfür verschiedene Lösungsmethoden an.

sondern aus der Visualisierung heraus auch manipuliert werden können. Die direkte Veränderung der Modellparameter führt dann zu einem veränderten Aufbau der jeweiligen Ereignisliste und veränderten Simulationsergebnissen.

### 3.3.2 Rückwärtssimulation

Im Rahmen eines erweiterten Einsatzes der Methode Ablaufsimulation soll auch eine rückwärtsgerichtete Ausführung von Simulationsmodellen durch das Werkzeug unterstützt werden. Wie Abschnitt 3.2 gezeigt hat, existieren keine Software-Lösungen oder formale Modellierungsbeschreibungen, die eine rückwärts berechnende Ausführung von Simulationsmodellen ermöglichen. Dennoch existieren mit [YiCl94], [JaCh97], [WaMe97], [OhHa04] und [GrBo04] einige Arbeiten, in denen rückwärts-gerichtete Simulationsmodelle und Experimente mit unterschiedlichem Erfolg eingesetzt wurden. Für die Auswertung der entsprechenden Simulationsdaten mussten in allen Fällen umfangreiche Transformationen der generierten Datenmengen erfolgen, weil die eingesetzten Werkzeuge dieses Vorgehen nicht implizit unterstützt haben. Oftmals wurde hier mit einer Transformation und Neuberechnung aller einzelnen Ereigniszeitpunkte gearbeitet, so dass auch während der visuellen Analyse eines Simulationslaufs keine Erkenntnisse über das dynamische Verhalten des modellierten Systems gezogen werden konnten. Des Weiteren wurden die betrachteten Simulationsmodelle speziell für diese Untersuchungszwecke modelliert und die entsprechende Verhaltenssteuerung auf eine rückwärts gerichtete Ausführung hin optimiert. Ein phasenübergreifender Einsatz eines Simulationsmodells zur richtungsoffenen Simulation eines einzigen Simulationsmodells erfolgt in keinem der dargestellten Fälle.

**Bewertung**

Für die Konzeption und Implementierung des Werkzeugs sind insbesondere zwei Fragestellungen zu lösen: Zum einen eine zielgerichtete Transformation bestehender Simulationsmodelle in ihr entgegengesetzt gerichtetes Pendant und eine bessere Integration einer rückwärts gerichteten Ausführung eines Simulationslaufes in den Simulatorkern und die Visualisierungskomponenten.

## 3.4 Visualisierung von Ablaufsimulationen

Ein Ziel des anvisierten Werkzeugs ist es, durch die Visualisierung in einer virtuellen Umgebung eine höchstmögliche Immersion des Anwenders in das zu beplanende Fertigungssystem zu erreichen. Seine Wahrnehmung muss bis zu einem gewissen Grad getäuscht werden. Neben der Täuschung der menschlichen Wahrnehmung durch die virtuelle Umgebung und Repräsentationsformen von Objekten in der virtuellen Realität wird in diesem Abschnitt insbesondere die Eignung von VR-Systemen als Benutzerschnittstelle untersucht.

Für die Planung und Gestaltung eines Fertigungssystems ist eine umfangreiche, ganzheitliche Betrachtung seiner Komponenten nötig, inklusive deren gegenseitige Wechselwirkungen. Simulationssysteme, deren Visualisierungsmöglichkeiten sich auf eine zweidimensionale Darstellung beschränken, sind jedoch oftmals nicht in der Lage, alle Elemente und Strukturen so vollständig abzubilden und liefern somit nur ein unvollständiges oder verfälschtes Abbild der Realität [Berg02].

Betrachtet man die Entwicklung der vergangenen Jahre auch in anderen Bereichen der IT-Industrie, so lässt sich ein Trend hin zu einer dreidimensionalen Repräsentation einzelner Objekte erkennen. Diese Form der Darstellung eröffnet dem Anwender eine verbesserte Möglichkeit der Erfassung von Elementen und ihrer Anordnung in Fertigungsstrukturen. Im Gegensatz zur zweidimensionalen Darstellung erhält er nicht nur quantitative Ergebnisse und statistische Kenngrößen eines Simulationsmodells, sondern in einem begrenzten Rahmen auch Informationen über die räumliche und zeitliche Anordnung und Verhaltensweisen des abgebildeten Fertigungssystems. Simulationsmodelle, die eine dreidimensionale Darstellung offerieren, können daher aussagefähigere und zuverlässigere Ergebnisse anbieten und den Prozess der Modellvalidierung und –verifikation wesentlich beschleunigen [Berg02].

Die realistischere Darstellung des abgebildeten Fertigungssystems bietet darüber hinaus wesentliche Vorteile, wenn es um die Kommunikation einzelner Planungsschritte mit Nicht-Simulationsexperten geht. Die grundlegende Verhaltensweise eines Systems erschließt sich umso leichter, je realistischer die Abbildung eines Systems in der virtuellen Umgebung erfolgt. Ein möglichst immersiver Umgang mit dem Simulationsmodell in einer virtuellen Umgebung kann diesen Effekt auch für den Anwender selbst weiter verstärken. Das erscheint insbesondere vor dem Hintergrund einer zunehmenden Komplexität der Simulationsprojekte und dem Anwachsen der Planungsteams umso mehr sinnvoll [Berg02].

### 3.4.1 Menschliche Wahrnehmung in virtuellen Umgebungen

Menschen können mit ihrer Umwelt in unterschiedlicher Weise interagieren. Informationen werden aus der Umwelt aufgenommen und können an die Umwelt abgegeben werden. Die Eingangskanäle eines Menschen sind Sehen, Hören, Riechen, Schmecken und Fühlen. Riechen und Schmecken spielen in diesem Arbeitskontext eine untergeordnete Rolle. Die meisten Eindrücke sammeln Menschen mit dem Auge. Insbesondere bei dem Ausfall des Sehsinns (z.B. wegen Dunkelheit) werden Hören und Fühlen wichtiger.

#### 3.4.1.1 Täuschung des Sehsinns

Für eine Täuschung des Sehsinns müssen Modelle von virtuellen 3D-Welten in Bilder für die Augen übersetzt werden. Der Prozess unterteilt sich in die Berechnung der Bilder und das physische Erzeugen der Bilder.

**Berechnung der Bilder**

Um eine virtuelle Szene auf dem Bildschirm zu zeichnen (Rendering), müssen die Geometriedaten der Szene eine Rendering-Pipeline durchlaufen. Aus dreidimensionalen Vektordaten wird ein Bild generiert. Innerhalb der Rendering-Pipeline kann zwischen drei Arbeitsschritten unterschieden werden:

1. Die *Aufbereitung* ist dafür verantwortlich, die Geometriedaten der Szene zu reduzieren und in ein für die weitere Verarbeitung optimiertes Format zu bringen. Gegebenenfalls findet hier auch die Transformation bewegter Objekte an ihre Position innerhalb der Szene statt.

2. Die Schritte *Projection* und *Clipping* projizieren die transformierten Szenendaten auf den Bildschirm und schneiden „überhängende" Polygone auf die Bildschirmgröße zurecht.
3. Der abschließende Schritt der Rasterung wird auf Pixel[4]-Basis ausgeführt. Hier werden die zu zeichnenden Pixel in Abhängigkeit von transformierten Vektoren, Farben und Texturen eingefärbt. Die Sichtbarkeit eines Pixels wird durch Z-Buffering[5] ermittelt. Die Rasterung wird seit der Einführung von Grafikkarten mit 3D-Beschleunigung komplett in Hardware durchgeführt.

Große Teile der Rendering-Pipeline sind mittlerweile in Software-Schnittstellen verlagert, die aufgrund von Unterschieden in der Grafik-Hardware notwendig sind, um die Lauffähigkeit der Anwendung auf unterschiedlichen Hardware-Plattformen zu garantieren. Diese Grafik-APIs sollen unabhängig von der Grafik-Hardware einen festen Funktionsumfang bieten, so dass auch gegebenenfalls Funktionen, die in der Hardware nicht vorhanden sind, durch die Software-Schnittstelle emuliert werden.

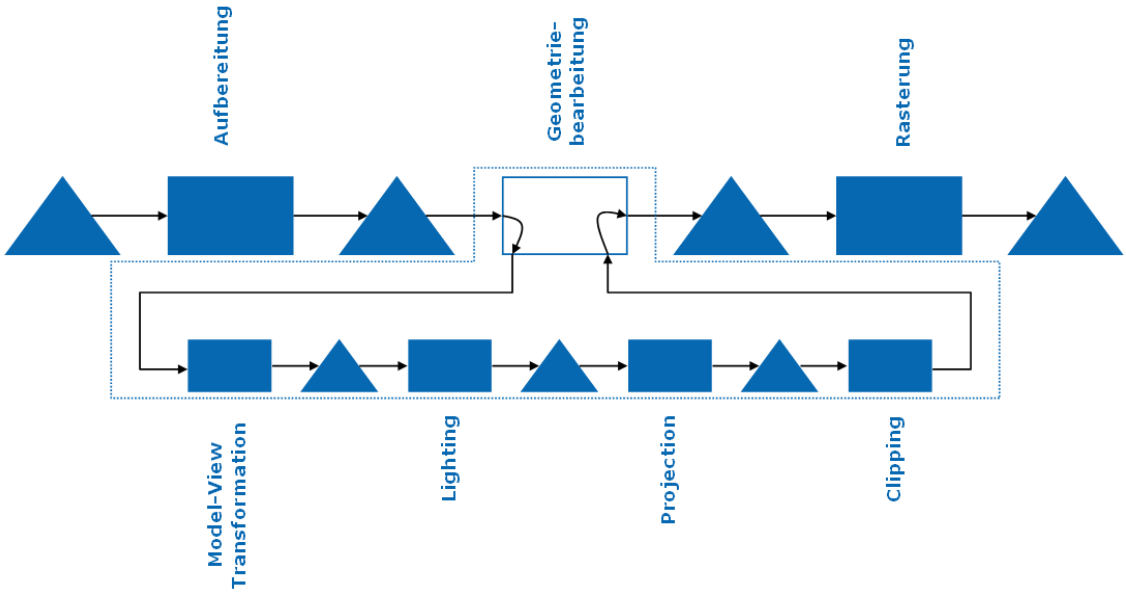


**Abbildung 10: Rendering Pipeline**

**Erzeugen der Bilder**

Entsprechend der Erkenntnis, dass die meisten Wahrnehmungen über den Sehsinn erfolgen, orientiert sich die Entwicklung von Ausgabegeräten auch auf die Täuschung der Augen. Durch verschiedene Abstände und Winkel, die die Augen zu einem realen Objekt haben, entstehen zwei perspektivisch verschobene Bilder. Um den Sehsinn zu täuschen, müssen diese beiden Bilder für das rechte und linke Auge erzeugt werden. Dies kann durch ein am Kopf befestigtes Display (Head mounted Display (HmD)) mit getrennten Bildern für das rechte und linke Auge geschehen. Neben den HmDs, die das Bild direkt am Kopf des Benutzers erzeugen, existieren Lösungen, bei denen Bilder entstehen, die

[4] Ein Pixel bezeichnet die kleinste Einheit der Darstellung auf einem Bildschirm [Balz05].

[5] Das Z-Buffering wird in der Computergraphik angewendet, um die verdeckten Flächen in einer dreidimensionalen Grafik zu ermitteln. Durch die Informationen im Z-Buffer (deutsch Z-Puffer) stellt das Verfahren fest, welche Elemente einer Szene gezeichnet werden müssen und welche verdeckt sind [BeBr03].

Informationen für beide Augen enthalten. Zur Trennung des rechten und linken Bildes haben sich zwei Lösungen in Form von Shutterbrillen oder der Trennung durch Polarisation durchgesetzt [DFAB98]. Während die Shutter-Technik nur einen Projektor benötigt, braucht die Technik mit Hilfe einer Polarisationstechnik zwei Projektoren. Trotzdem müssen für beide Verfahren zwei verschiedene Bilder erzeugt werden. Um einen möglichst weiten Bereich des Sichtfeldes des Anwenders auszufüllen, können mehrere Projektionssysteme zu einer größeren Fläche kombiniert werden. Diese Fläche kann planar sein, aber auch den Anwender partiell oder ganz umgeben. Eine mögliche Anordnung der Flächen ist ein Würfel, in dem sich der Anwender befindet. Die Bilder werden mit Projektoren von außen aufgebracht (Rückprojektion). Verschiedene Installationen von solchen Systemen unterscheiden sich in der Anzahl der verwendeten Flächen des Würfels. In der einfachsten Variante werden nur die Flächen vor und rechts wie links neben dem Anwender verwendet [Brac02]. Verbessert werden kann dies, indem man die Rückseite, die Decke und/oder den Boden hinzunimmt. Ist der Anwender komplett von dem Würfel umschlossen, kann er in alle Richtungen blicken, ohne eine Unterbrechung der virtuellen Umgebung zu erfahren. Neben dem Problem der Kanaltrennung muss auch die Kopfposition des Betrachters erfasst werden, um perspektivisch richtige Bilder zu erzeugen. Hier sind unterschiedliche Tracking-Verfahren (Mechanische Tracker, Ultraschalltracker und/oder elektromagnetische Tracker) bekannt [RDB01].

#### 3.4.1.2 Täuschung des Tast- und Hörsinns

Um den Hörsinn durch eine 3-dimensionale Tonwiedergabe zu überlisten, existieren einige Arbeiten. Töne aus Kopfhörern kommen immer aus der Richtung, aus der sie aufgenommen wurden. In der Realität würde ein Ton, der sich vor einer Person befindet, nach einer Drehung hinter ihr erscheinen. Töne aus Kopfhörern drehen sich jedoch mit. Um diesen Effekt auszuschalten, muss ein Computer Lage- und Richtungsortung beherrschen und diese dann mathematisch umsetzen. Die Position des Anwenders und dessen Kopflage werden geortet und die Töne an dessen Bewegung angepasst. Je nach Anwendung werden Kopfhörer oder im Raum angebrachte Boxen benutzt [Bega94].

Weit weniger Arbeiten existieren zum Täuschen des Tastsinns. Um eine realistische Interaktion mit der virtuellen Umgebung zu erzeugen, reicht es nicht aus, Objekte anzustoßen und zu bewegen, es ist zusätzlich nötig eine Rückkopplung zu erlangen, die auf den Anwender wirkt. Unterschieden wird zwischen dem *taktilen* Feedback, also der Rückwirkung des Tastens in Form von Druck, Wärme oder Schmerz und dem *Force* Feedback, also der Rückkopplung der entstehenden Kräfte durch Lenken der Finger und Hand.

### 3.4.2 VR als User Interface

Im stetig härter werdenden Wettbewerb wird die Umsetzung von Softwaresystemen, die die *Virtual Reality* (vgl. Definition 30) einsetzen, als eine Technologie angesehen, die es den betroffenen Unternehmen (heutzutage aufgrund der hohen Entwicklungskosten meist Großunternehmen) nicht nur ermöglicht, am Markt zu bestehen, sondern Wettbewerbsvorteile gegenüber Konkurrenten zu erarbeiten [Berg02]. Die von einem Computer generierte und kontrollierte Umgebung für die Mensch-Maschine-Interaktion (hier analog zum Begriff der virtuellen Umgebung) umfasst somit alle Technologien zur

Definition und echtzeitfähigen Aufbereitung eines rechnerinternen, dreidimensionalen Modells für die menschlichen Sinne, die es dem Anwender ermöglichen durch Einbeziehung seiner Person in das Modell (vgl. Immersion) und infolge durch das Modell initiierter Rückkopplungen direkt zu manipulieren (vgl. Interaktion) [Berg02].

Die beschriebenen Systeme sind zum Teil sehr aufwendig und kostenintensiv zu implementierten. In vielen Anwendungsbereichen wird ein so hoher Grad der Immersion nicht benötigt. In CAD-Systemen[6] beispielsweise sind die traditionellen, preiswerten Systeme durch Formen der Animation aufgewertet worden, so dass Objekte durch vorab definierte Bewegungsgleichungen kontrolliert werden können. In der Konstruktion oder in der Architektur z.B. reicht es oft aus, den Objekten durch Schattierungen einen Tiefeneindruck zu verleihen oder sie durch Drehen von einer anderen Seite zu betrachten, um dem Kunden eine Vorstellung zu vermitteln.

Die Aufgaben eines VR-Systems liegen demnach in der Modellierung der Objekte einer virtuellen Umgebung, der Präsentation der virtuellen Umgebung sowie der Realisierung der Interaktion mit der virtuellen Umgebung. Für jede dieser drei Teilaufgaben sind spezifische Hard- und Softwarekomponenten erforderlich [Berg02].

**Modellierung**
Inhalt der Modellierung ist im Wesentlichen die Generierung dreidimensionaler Modelle, die in einer virtuellen Umgebung genutzt werden können. Sie umfasst somit die Erzeugung von 3D-Modellen, also die gestaltorientierte Modellierung sowie die Definition der Funktion der Objekte, d.h. die funktionsorientierte Modellierung. Für die gestaltorientierte Modellierung stehen umfangreiche CAD-Werkzeuge zur Verfügung, auf die auch in der hier vorliegenden Arbeit zurückgegriffen werden soll. Die Funktion der 3D-Modelle beschränkt sich hierbei auf die Repräsentation spezieller Funktionsblöcke (Modellbausteine) eines Simulationsmodells, so dass eine implizierte Modellierung des Verhaltens durch die Modellierungskomponente des zu entwickelnden Werkzeugs erfüllt wird.

**Präsentation**
Im Rahmen der Präsentation der virtuellen Umgebung erfolgt die auf den zuvor definierten 3D-Modellen basierende Erzeugung der von den menschlichen Sinnen wahrnehmbaren Präsentationsformen. Hierfür wird neben speziellen Grafikcomputern eine Software-Applikation notwendig, die die dreidimensionale Repräsentation erzeugt. Darüber hinaus müssen entsprechende Ausgabegeräte bereitgestellt werden. Technisch realisierbar sind heutzutage die visuelle, die akustische sowie mit einigen Einschränkungen die haptische Präsentation, wobei sich die meisten VR-Systeme auf die visuelle Präsentation beschränken. Im Rahmen des hier zu entwickelnden Werkzeugs soll sich ebenfalls auf die visuelle Präsentation des Simulationsmodells in einer virtuellen Umgebung beschränkt werden. Die entsprechende VR-Software soll im Rahmen der Werkzeugentwicklung als Visualisierungskomponente erstellt und auf die speziellen Anforderungen einer immersiven, interaktiven und dreidimensionalen Visualisierung

[6] Computer-Aided-Design (CAD) ist die rechnergestützte Entwicklung und Konstruktion von Bauteilen, Baugruppen, Erzeugnissen und Anlagen unter Einschluss technischer Berechnungen sowie der Bewegungssimulation von Objekten und der Erarbeitung von Dokumentationen [FiHe00].

abgestimmt werden. Da die vorliegenden 3D-Daten meist zu reich an Informationen sind, können im Rahmen der Umsetzung der 3D-Visualisierung spezielle Methoden und Algorithmen zur Komplexitätsreduktion eingesetzt werden (vgl. [KlKr02], [WaFi01]).

**Interaktion**
Unter Interaktion können alle Mechanismen summiert werden, welche die direkte Einflussnahme des Anwenders auf die Objekte in der virtuellen Umgebung ermöglichen (vgl. Definition 28). Die beiden wesentlichen Komponenten zur Interaktion sind 3D-Positionsmeßsysteme (Trackingsysteme) und 3D-Eingabegeräte (Devices). Tracking-systeme ermöglichen die Verfolgung von Kopf- bzw. Handbewegungen des Anwenders, indem die Position und Orientierung des Senders im Raum in den sechs möglichen Freiheitsgraden relativ zu einem Bezugspunkt erfasst wird. Für diese Form der Interaktion in einer virtuellen Umgebung werden spezielle Eingabegeräte benötigt, die eine Bewegung im Raum bestmöglich unterstützen. Viele VR-Probleme lassen sich aber auch mit handelsüblichen Software-Tools und Desktop-PCs und deren Ein- bzw. Ausgabegeräten angemessen lösen. Als Ausgabegeräte kommt hierbei meist der Bildschirm (eventuell Stereo-Lautsprecher) zum Einsatz. Für die Eingabe werden die Tastatur und die Maus verwendet. Für die Implementierung des hier zu entwickelnden Werkzeugs sollen die zuletzt genannten Methoden genügen, wobei während der Umsetzung eine mögliche Erweiterung um spezielle Darstellungs- und Interaktionsmöglichkeiten zu berücksichtigen ist.

### 3.4.3 Interaktion in virtuellen Umgebungen

Handelt und agiert ein Anwender in einer virtuellen dreidimensionalen Umgebung, so spricht man von 3D-Interaktion. Elementar für eine intuitive Interaktion mit einem System ist die so genannte Reiz-Reaktions-Korrespondenz. Diese liegt immer dann vor, wenn z.B. auf eine Bewegung des Eingabegerätes nach rechts, auch eine entsprechende Bewegung des selektierten Objektes nach rechts erfolgt. Liegt diese nicht vor, muss der Anwender ein mentales Abbild der gewünschten Aktion in Verbindung mit der erforderlichen Bewegung erstellen. Nach Stork [Stor00] lassen sich folgende Basis(inter-)aktionen unterscheiden:

- Das Navigieren, verstanden als zielgerichtete Bewegung im Raum.
- Das Positionieren, verstanden als das Platzieren eines Objektes an einer definierten Raumposition.
- Das Orientieren, verstanden als das Rotieren und Ausrichten eines Objektes im Raum.
- Das Selektieren bzw. Deselektieren, verstanden als das Auswählen eines Punktes oder Objektes im Raum.

Navigation, Objekt-Selektion und -Manipulation sind die Basisoperationen, die von jeglichen dreidimensionalen Anwendungen umgesetzt werden müssen. Hierbei ist es üblich, dass Positionieren und Orientieren als eine Operation zusammengefasst werden, da ein 3D-Eingabegerät mit sechs Freiheitsgraden, diese Operationen gleichzeitig durchführen kann. In der 3D-Interaktion wird unter Navigation die Bewegung des Anwenders in der Umgebung der dreidimensionalen virtuellen Szenerie verstanden. Häufig ist eben diese Navigation die erforderliche Grundlage zur Durchführung der tatsächlichen Interaktionsaufgaben und nicht die eigentliche Aufgabe. Navigation ist also

oft „nur Mittel zum Zweck"; sie darf den Anwender auf keinen Fall von seiner ursprünglichen Aufgabe ablenken. Je nach Art der Umgebung, lassen sich nach Bowman et. al. [BKLP01] unterschiedliche Formen der Navigation unterscheiden:

- Exploration, d.h. Navigation ohne spezielles Ziel bzw. als Umgebungserkundung.
- Search, d.h. Aufsuchen des Ziels und sich dorthin bewegen.
- Maneuvering, d.h. Durchführung kleiner, präziser Bewegungen, um eine bessere Position für eine anstehende Arbeit einzunehmen.

Es existieren noch zwei weitere Arten der Navigation, bei denen gezielt die Freiheitsgrade eines navigierenden Akteurs eingeschränkt sind:

- Walking, wobei der Anwender sich auf einem zweidimensionalen Untergrund bewegt. Es sind lediglich die Bewegungen vor, zurück, links und rechts möglich. Der Akteur bewegt sich immer in die Richtung, die seiner Blickrichtungsachse entspricht. Die Walking-Metapher erfordert eine kontinuierliche Eingabe von Seiten des Anwenders. Erfolgt diese nicht, so bleibt er stehen. Dient diese Art der Navigation nicht dem eigentlichen Aufgabenzweck, so kann die erforderliche Dauereingabe als störend empfunden werden. In weiträumigen Szenerien, in denen sich der Anwender via Exploration einen Überblick verschaffen will, ist diese Metapher aufgrund der eingeschränkten Bewegungsfreiheit ungeeignet. Vorteilhaft ist sie in Situationen, bei denen der Anwender beengte Räume erkundet bzw. bestimmte Zielpunkte exakt ansteuern möchte.
- Flying, ermöglicht eine kontinuierliche Bewegung entlang einer geraden Flugbahn. Richtungsänderungen z.B. durch einen Mausklick sowie Änderungen der Geschwindigkeit sind möglich. Nachteilig ist diese Navigationsart in beengten Räumen, da man durch Begrenzungselemente wie z.B. Wände einfach hindurch fliegt. Auch hier lässt sich die Orientierung der Kamera nicht unabhängig von der Bewegung des Akteurs verändern.

Um die Positionsbestimmung zu vereinfachen, kann ein Akteur sich Navigationshilfen bedienen. Diese können fester Bestandteil der 3D-Umgebung (z.B. Hinweisschilder) bzw. externe Objekte (z.B. Übersichtskarte mit eingeblendeter Position und Blickrichtung, Kompass) sein. Genau wie in der realen Welt, bieten Viewpoints einen besonderen Überblick über eine komplette Szenerie bzw. ausgewählte Teilbereiche einer Szenerie. Solche Viewpoints können zusätzliche Informationen, über das, was zu sehen ist, für den Anwender bereithalten. Für die Erreichung eines Viewpoints kann sich der Anwender beispielsweise der Beam-Metapher bedienen, bei der er zu dem gewünschten Ort teleportiert wird. Problematisch ist hierbei, dass der Akteur schnell die Übersicht über seine derzeitige Position verliert.

### 3.4.4 Virtuelle Umgebung von Fertigungssystemen

Das Problemfeld, dem sich diese Arbeit widmet, liegt im Umfeld der Modellierung und Simulation komplexer Fertigungssysteme, die hinsichtlich verschiedenster Fertigungsprinzipien organisiert werden und in einer virtuellen Umgebung dargestellt werden, um sie möglichst immersiv und interaktiv parallel zu ihrer Ausführung analysieren und optimieren zu können. Dafür ist es notwendig, alle zur Erfüllung dieser

Aufgaben notwendigen Teilsysteme des abzubildenden Systems mit ihren technischen Objekten (Anlagen, Maschinen, Werker, etc.) sowie die unterschiedlichen, sich gegenseitig bedingenden Vorgänge abzubilden, die das dynamische Verhalten des Systems beeinflussen. Die modellierten Teilmodelle (Modellbausteine) sind verschiedenartig gestaltet und üben darüber hinaus verschiedenartige Funktionen aus. Neben der eigentlichen Gestalt der Modellbausteine in der virtuellen Umgebung stellt daher auch die Funktion des Modellbausteins ein wesentliches Merkmal dar.

Das Gestaltmodell als Partialmodell zur Repräsentation der Gestalt der in einer virtuellen Umgebung präsentierten Objekte wird meist durch 3D-Modelle realisiert, die mittels Volumen-[7] oder auch Flächenmodell[8] erstellt werden. Die Darstellung eines Fertigungssystems in einer virtuellen Umgebung mit ausschließlich statischen Objekten hat für technische Anwendung wie die Simulation, die insbesondere Aufgabenstellungen hinsichtlich der Funktionalität des abgebildeten Systems erfüllen sollen, wenig Aussagekraft und damit einen geringen Stellenwert. Die Evaluation eines Fertigungssystems mittels der Virtual Reality erfordert daher die Simulation oder zumindest Animation der Bewegung einzelner Modellbausteine sowie die Möglichkeit zum direkten Eingriff des Anwenders [Berg02]. Die verschiedenen interaktiven Steuerungsmöglichkeiten werden dem Anwender über ausgewählte Bedienelemente zur Verfügung gestellt und variieren in ihrer Komplexität stark hinsichtlich des gewünschten Untersuchungsziels.

### 3.4.5 Verständnisoptimierung durch virtuelle Umgebungen

Die Komplexität eines Fertigungssystems erklärt sich nicht nur aus der Anzahl aller Elemente, sondern insbesondere auch aus deren gegenseitiger Vernetzung, durch die sich die einzelnen Elemente gegenseitig mehr oder weniger stark beeinflussen. So erzeugen also die Verbindungen zwischen den einzelnen Elementen die eigentliche Komplexität und bedingen die gleichzeitige Beachtung sehr vieler Merkmale. Die hohe Komplexität eines Fertigungssystems stellt somit hohe Anforderungen an die Fähigkeiten des Anwenders, Informationen zu sammeln, zu integrieren und Handlungen zu planen. Die Komplexität an sich wird damit zur subjektiven Größe. Für den vereinfachten Umgang mit komplexen Fertigungssystemen ist es also ausschlaggebend, dass es in einer Form präsentiert wird, welche dem realen Erscheinungsbild bestmöglich entspricht, um den Assoziationseffekt des Menschen bestmöglich auszunutzen. Die meisten Menschen beschreiben die wirkliche, physische Welt mit Begriffen aus ihrem Wahrnehmungsbereich, mit Objekten, die sie umgeben und die sich möglicherweise bewegen. Oftmals wird diese Wahrnehmung auch räumlich beschrieben, zum Beispiel dadurch, wo man sich im Raum befindet und wie die umgebenden Dinge in Relation zueinander stehen. Für die Visualisierung eines Fertigungssystems ist es daher erforderlich, die beiden Definitionsbereiche, also das Aussehen und die räumliche Anordnung abzubilden, da sie für ein reales Empfinden des Anwenders ausschlaggebend sind (vgl. [Berg02]).

---

[7] Volumenmodell: Bei einem Volumenmodell hat jedes räumliche Objekt ein definiertes Volumen um Raum und geometrische Eigenschaften (Oberfläche, Volumen, Mittelpunkt) sowie physikalische Eigenschaften [FiHe00].

[8] Flächenmodelle beschreiben die zu modellierenden Objekte als einzelne Flächen. Sie lassen sich je nach Komplexität in die unterschiedlichen Typen Ebene, Quadrike und Freiformflächen unterscheiden [FiHe00].

Die Technologie der Virtual Reality erlaubt die räumliche und realitätsnahe Darstellung von Vorgängen und Gegenständen der realen Umwelt oder erdachter Welten und die Immersion des Anwenders in diese synthetische Welt. Der Anwender bekommt den Eindruck, tatsächlich in der virtuellen Welt anwesend zu sein [Lani91] und kann sie nach eigenen Wünschen in natürlicher Form erfahren. Durch die wirklichkeitsgetreue Abbildung, die zumindest potentiell durch diese Technologie erreicht werden kann, erhält der Anwender eine verbesserte Rückmeldung der Modellbausteine seines Simulationsmodells und kann so sein Erfahrungswissen optimaler auf das abgebildete Problem anwenden [Berg02].

Neben der realitätsnahen Darstellung ist die Integration des Anwenders in das virtuelle Simulationsmodell von entscheidender Bedeutung. Der Anwender wird so Teil des Systems und kann unmittelbarer dessen Funktionsweise erleben, überprüfen und verbessern. Neben einer monokausalen Denkweise nach dem Ursache-Wirkung-Prinzip kann der Anwender ein multikausales Geflecht aufbauen und das System so „von innen heraus" verstehen. Die räumliche Darstellung und die Option der Bewegung ermöglicht dem Anwender die dargestellte Szene aus verschiedenen Perspektiven zu erleben. Räumlich-zeitliche Muster können ergangen und durch den unmittelbaren Erlebnisbezug kann der Aufbau und das Verhalten des abgebildeten Fertigungssystems überprüft werden [Lani91]. Die Effektivität und Effizienz des Problemlösungsprozesses kann ebenso wie die Aussagekraft der Entscheidungen erhöht werden.

Durch die Darstellung in einem dreidimensionalen Raum kann der Anwender zusätzlich erheblich mehr Daten gleichzeitig im Blickfeld halten, aufnehmen und schneller zu entscheidenden Erkenntnissen gelangen, weil er Strukturen besser und schneller erkennt und so neu arrangieren kann, um das zugrunde liegende Problem zu lösen. Ein unzusammenhängendes Ensemble von Modellbausteinen erhält eine Bedeutung. Insgesamt kann der Anwender durch den Einsatz der Technologie der Virtual Reality zu einem besseren Verständnis der Zusammenhänge gelangen und dadurch ohne größeren Einarbeitungsaufwand ein Fertigungssystem analysieren und optimieren.

Zusammenfassend kann man also festhalten, das bereits umfangreiche Arbeiten zur Gestaltung von virtuellen Umgebungen existieren, deren Ergebnisse in die Konzeption des hier zu entwickelnden Werkzeugs einfließen können. Verschiedene Kopplungen von Simulatoren an virtuelle Umgebungen sind bereits gelungen und erfordern neben einer Festlegung der benötigten Schnittstellen keine besonderen Anforderungen, die über die Gestaltung von echtzeitfähigen Benutzeroberflächen hinausgehen. Das Werkzeug an sich muss aber aus mehr als nur den Visualisierungskomponenten bestehen. Im folgenden Abschnitt sollen deswegen Methoden und Paradigmen des Software-Designs vorgestellt werden, die in der Phase der Konzeption des Werkzeugs in seiner Gesamtheit wie der einzelnen Module benötigt werden.

## 3.5 Software-Design von Mehrbenutzersystemen

Nach der fachlich orientierten Betrachtung des Stands der Technik hinsichtlich der Anforderungen an Basisprozess, Modellierungsmethode und Simulationswerkzeug in den vorherigen Abschnitten werden hier die informationstechnischen Grundlagen betrachtet,

soweit sie die Konzeption und Realisierung des Werkzeugs betreffen. Die Grundlagen des Software Engineering[9] sowie Merkmale von Software-Architekturen[10] werden beschrieben. Aus der Entscheidung für eine bestimmte Architektur innerhalb eines vorgegebenen Design-Paradigmas resultieren Konsequenzen für die spätere Implementierung.

### 3.5.1 Allgemeine Paradigmen des Software-Designs

Zur Ausführung eines Software-Entwurfs werden in der Literatur allgemeine Verhaltensregeln oder Prinzipien angegeben und diskutiert [Henk97]. In diesem Abschnitt werden die Paradigmen Abstraktion, Hierarchisierung, Strukturierung und Modularisierung behandelt, zwei weitere Prinzipien (Kapselung und Typisierung) werden im Rahmen des folgenden Abschnitts über objektorientiertes Design betrachtet, da sie dort besondere Bedeutung erfahren. Zwar ist eine objektorientierte Vorgehensweise nicht unbedingte Vorraussetzung für die Anwendung dieser Prinzipien, dennoch haben viele objektorientierte Methoden mindestens implizit die Unterstützung dieser Prinzipien zum Ziel. Hieraus begründet sich auch, wenn auch nicht ausschließlich, die Entwicklung des objektorientierten Paradigmas.

**Abstraktion**

> *„Abstraktion (von lat. abstrahere: abziehen, wegziehen) bezeichnet allgemein den Prozess rationaler Verarbeitung von konkretem Sinnesmaterial, wobei von bestimmten äußeren, individuellen oder zufälligen Merkmalen, Eigenschaften und Beziehungen des betreffenden Objekts abgesehen wird; andere, allgemeingültige, strukturelle Eigenschaften werden dagegen als wesentlich herausgehoben und zugleich variabel betrachtet*" [Balz82].

Abstraktion ist eines der wichtigsten und leistungsfähigsten softwaretechnischen Prinzipien [Henk97]. Die Komplexität der zu entwickelnden Programme verhindert eine ganzheitliche Betrachtung mit allen Details, daher werden sie durch die Methode Abstraktion auf ihre wesentlichen Charakteristika reduziert. Die Anwendung des Paradigmas Abstraktion im Software-Design bringt zusammenfassend folgende Vorteile:

- Erkennen, Ordnen, Klassifizieren, Gewichten von wesentlichen Merkmalen
- Erkennen allgemeiner Charakteristika,
- Trennung zwischen wesentlichen und unwesentlichen Eigenschaften

Eine funktionale Abstraktion stellt eine Leistung in Form einer abstrakten Funktion, Operation oder Prozedur zur Verfügung. Sie wird daher oft auch operationale oder prozedurale Abstraktion genannt [Balz05].

**Hierarchisierung**

---

[9] Das Software Engineering als Teilgebiet der Informatik beschäftigt sich mit der standardisierten, ingenieursmäßigen Herstellung von Software und den damit verbundenen Prozessen [Somm92].

[10] Software-Architekturen beschreiben die Zerlegung einer Anwendung in Einheiten, den globalen Kontrollfluss, die Handhabung von Randbedingungen und Kommunikationsprotokolle zwischen Subsystemen [BrDu04].

> „E*ine Hierarchie bezeichnet ein System von Elementen, die einander über- bzw. untergeordnet sind, so dass jedem Element nur höchstens ein anderes unmittelbar übergeordnet ist*." [nach Henk97].

Bereits die Bedeutung des Abstraktionsprinzips zeigt die menschliche Notwendigkeit, ein gedankliches Modell eines komplexen realen Systems anzufertigen, um dieses System verstehen zu können. Reale Systeme sind häufig gut geeignet, hierarchisch gegliedert zu werden. Ein komplexes System enthält miteinander in Beziehung stehende Subsysteme, diese setzen sich wiederum aus weiteren Subsystemen zusammen, bis schließlich die Elemente eines Subsystems nicht weiter sinnvoll dekomponiert werden können. Auf diese Weise entsteht eine aus Hierarchieebenen gebildete Struktur, die das Verstehen des Gesamtsystems wesentlich erleichtert oder sogar Voraussetzung dafür ist. Weil die Struktur hierarchischer Systeme das menschliche Verständnis erleichtert und Softwaresysteme Modelle realer (hierarchisch gegliederter) Systeme voraussetzen, bietet es sich an, auch künstliche Systeme (z. B. Softwaresysteme) nach dem Prinzip der Hierarchisierung zu gliedern. Hierarchisierung ist demnach ein weiteres wichtiges Paradigma des Software Designs. Es stellt sich die Frage, nach welchen Kriterien die Hierarchisierung vorgenommen werden soll. Zwei Arten von Hierarchien haben sich als besonders gut zur Strukturierung von Softwaresystemen geeignet erwiesen:

Die erste Art der Hierarchie ergibt sich aus der Verallgemeinerungsbeziehung zwischen zwei Elementen eines künstlichen oder realen Systems. Diese Beziehung ist sehr häufig zwischen Systemelementen anzutreffen: Ein „Möbel" stellt beispielsweise die Verallgemeinerung von „Schrank", „Bett", „Tisch" etc. dar. Diese Art von Hierarchie wird deshalb häufig IS-A-Hierarchie genannt. Die zweite Art ist die Part-Of-Hierarchie, die auf dem Kriterium des Enthaltenseins basiert. Ein Fahrradventil ist ggf. ein Teil des Vorderrades und dieses ist Teil des Fahrrades als Ganzes. Anders herum: Ein Fahrrad hat ein Vorderrad und ein Vorderrad hat ein Ventil.

Folgende Vorteile der Anwendung des Hierarchisierungsprinzips können also festgehalten werden:

- Strukturierung eines Softwareproduktes,
- Erhöhung der Verständlichkeit,
- Verbesserung der Wartbarkeit,
- Reduktion der Komplexität; Erhöhung der Einfachheit,
- Strukturierung des Entwicklungsprozesses.

Die besondere Bedeutung der Is-A- und der Part-Of-Hierarchie liegt darin begründet, dass bei Anwendung des objektorientierten Paradigmas der Softwareentwicklung diese Hierarchien eine direkte Entsprechung in der Klassen- und Objekthierarchie des zu gestaltenden Softwaresystems finden können (vgl. hierzu auch 3.5.2). Wichtig neben diesen beiden Arten sind darüber hinaus die Hierarchie der Module und die Prozesshierarchie, also die Beziehungen zwischen dynamischen Systemkomponenten.

**Strukturierung**

> *„Unter Strukturierung versteht man die Tätigkeit, bei der einer homogenen oder homogen erscheinenden Menge von Dingen eine Struktur, insbesondere im abstrakten Sinn eine Klassifikation, aufgeprägt wird.*" [nach Henk97]

Weil große und komplexe Systeme sich dem menschlichen Verständnis entziehen, müssen solche Systeme in Teile zerlegt werden, die leichter beherrschbar sind. Wenn diese Teile unabhängig voneinander bearbeitet werden können, genügt für das Verständnis des Gesamtsystems das Verständnis der kleineren Teile und ihres Zusammenspiels. Eine geeignete Zerlegung des Systems ist also dazu geeignet, die Komplexität zu reduzieren. Die Betrachtung des zerlegten Systems beinhaltet nicht mehr alle Informationen über die Beziehungen zwischen Systemelementen, die im kompletten System vorhanden sind. Deshalb kann eine ungeeignete Zerlegung zu einer Betrachtungsweise führen, die den Charakter des Gesamtsystems nicht offen darlegt. Die Zerlegung eines Systems muss seiner Struktur entsprechen.

Abbildung 11 zeigt ein willkürlich gewähltes Beispielsystem A, eine ungeeignete Zerlegung B und eine Zerlegung, die geeignet ist, die Systemstruktur C zu veranschaulichen. Während die Darstellungen A und B dem Betrachter wegen ihrer höheren Komplexität kaum ermöglichen, den Aufbau des Systems mit einem Blick zu erfassen, stellt das in C kein Problem mehr dar: Die geeignete Strukturierung des Gesamtsystems hat dessen Komplexität wesentlich reduziert.

Für das Software Design hat das Prinzip der Strukturierung doppelte Bedeutung: Sowohl für das eigentliche Softwareprogramm, wie auch für den Designprozess selbst muss eine geeignete Struktur identifiziert werden. Das Paradigma der Strukturierung unterstützt also auch das methodische Vorgehen beim Software Design. Die explizite Strukturierung des Systems zwingt den Softwareingenieur, sich die Vor- und Nachteile unterschiedlicher Designalternativen vor Augen zu führen.

Die Anwendung des Prinzips der Strukturierung bringt folgende Vorteile:

- gute Verständlichkeit
- leichte Einarbeitung
- Änderungsfreundlichkeit
- gute Wartbarkeit

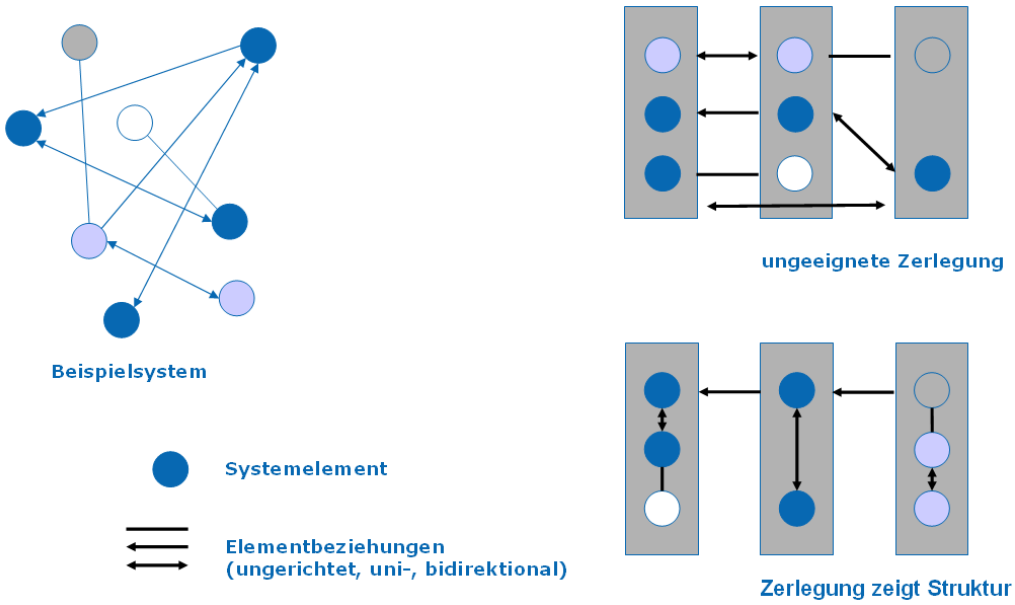


**Abbildung 11: Komplexitätsreduktion durch Strukturierung [Henk97]**

**Modularisierung**

> *„Ein Modul ist ein Bauteil eines größeren Baukastensystems. Module werden hauptsächlich verwendet, um sie leicht gegen andere Module austauschen zu können, oder neue Module zu besagtem Ganzen hinzuzufügen. Deshalb ist für Module eine Schnittstelle vonnöten, um sie mit dem Ganzen zu verbinden."* [Henk97]

In den vorherigen Abschnitten wurde gezeigt, dass die Dekomposition von Systemen in seine Komponenten genutzt werden kann, um die Komplexität des Gesamtsystems zu verringern. In einem mehr technischen Sinne bezeichnet man solche Teilsysteme als Module, wenn sie als Softwarebausteine dienen, indem sie Daten und Algorithmen[11] zu einer funktionalen Einheit zusammenfassen. Die Festlegung der Module eines Softwaresystems wird entsprechend als Modularisierung bezeichnet.

Die interne Struktur jedes Moduls sollte einfach sein, so dass es leicht verstanden werden kann. Dennoch ist es ein Ziel, die Kenntnis dieser Interna ausdrücklich nicht zur Voraussetzung der Benutzung eines Moduls zu machen (vgl. funktionale Abstraktion). Stattdessen definieren Module Schnittstellen für den Zugriff auf die der Umgebung zur Verfügung gestellten Ressourcen (Export-Schnittstellen) und die aus der Umgebung benutzten Ressourcen (Import-Schnittstellen). Für die Benutzung der Module ist dann lediglich die Kenntnis der Spezifikation dieser Schnittstellen nötig. Die tatsächliche Implementierung eines Moduls wird durch diese Schnittstellenspezifikation nicht festgelegt und kann vor dem Modulbenutzer verborgen werden (Geheimnisprinzip).

[11] Ein Algorithmus wird hier verstanden als *„ein schematisches Verfahren zur Lösung bestimmter Klassen von Aufgaben, wobei jeder einzelne Schritt dieses Verfahrens genau definiert ist. (...) Er ist die Gesamtheit von Grundoperationen und Prüfungen bestimmter logischer Bedingungen, die in einer bestimmten Reihenfolge angeordnet sind"* [KlBu71]

Das Ziel der Modularisierung ist, Module zu finden, die unabhängig voneinander entworfen und getestet werden können. Dazu soll jedes Modul von seiner Umgebung unabhängig

- zu entwickeln,
- übersetzbar,
- prüfbar (testbar, verifizierbar),
- wartbar,
- verständlich sein.

Wenn Module mit anderen Modulen in Beziehung stehen, kann eine vollständige Kontextunabhängigkeit nicht erreicht werden. Die Kommunikation zwischen verbundenen Modulen führt zu Abhängigkeiten, die als Kopplung bezeichnet werden. Diese Kopplung auf das notwendige Maß zu beschränken, ist eine Designvorgabe der Modularisierung[12].

Für das Verfahren des Software-Entwurfs sind also verschiedene Prinzipien zur Komplexitätsreduktion bekannt, die in der Praxis angewendet werden, um Software-Produkte schneller, effektiver und vor allem weniger fehleranfällig zu erstellten. Moderne Programmiersprachen unterliegen heutzutage dem Prinzip der Objektorientierung. Der folgende Abschnitt soll also weitere Organisationsprinzipien vorstellen, mit denen der Prozess der Erstellung und Wartung von Software-Produkten weiter vereinfacht werden kann.

### 3.5.2 Objektorientierte Software-Programmierung

Objektorientiertes Vorgehen bei der Analyse, dem Design und der Programmierung von Softwarewerkzeugen wird als besonders geeignet angesehen, um robuste und leicht erweiterbare Softwareprogramme zu erzeugen [GaHe01]. Nach einigen grundlegenden Ideen der objektorientierten Programmierung (OOP) werden in diesem Abschnitt spezielle Paradigmen der objektorientierten Programmierung näher erläutert: Kapselung, Polymorphie und Vererbung.

> *„Objektorientierte Programmierung (Abkürzung OOP) ist ein Verfahren zur Strukturierung von Computerprogrammen, bei dem zusammengehörige Daten und die darauf arbeitende Programmlogik zu Einheiten zusammengefasst werden, den so genannten Objekten.*"[nach GaHe01]

Objekte als fundamentale Bausteine der OOP werden also verwendet, um Aspekte realer Systeme zu modellieren. In der Objektorientierung finden sich daher auch wesentliche Begriffe aus der Systemtheorie wieder. Das eigentlich Neue an der Objektorientierung sind also nicht die zugrunde liegenden Konzepte, sondern deren Adaption und Anwendung auf die Softwaretechnologie. Die Komposition der Objekte entspricht der Bildung einer Part-Of-Hierarchie. (vgl. Abschnitt 3.5.1) Objekte werden dem

[12] In der Literatur wird häufig die innere Bindung von Modulen als *Kohäsion* bezeichnet; *Kopplung* bezeichnet entsprechend die externe Bindung von Modulen. Sowohl Kohäsion, als auch Kopplung können weiter nach Typen unterschieden werden, vgl. z.B. [Henk97]. Für die vorliegende Arbeit sollen diese Details außer Betracht bleiben.

Modularisierungsgedanken gerecht, indem sie Daten und Algorithmen zu einer Einheit mit systemweiter Identität und aktuellem Status zusammenfassen.

Allgemeine und objektorientierte Prinzipien als solche sind nicht neu und ihre Anwendung ist nicht auf die Programmierung beschränkt. Dennoch werden sie grade hier synergetisch zusammengefügt, insbesondere in den objektorientierten Modellen. Die Grundidee basiert auf der Annahme, mit ihrer Anwendung die Komplexität von Softwaresystemen und damit auch den Projekten zur Softwareentwicklung zu reduzieren, indem das Gesamtsystem inkrementell aus Objekten zusammengefügt wird, die jeweils unabhängig voneinander entworfen und getestet werden können. Idealerweise können einzelne Objektspezifikationen in späteren Projekten wieder verwendet werden.

#### 3.5.2.1 Grundlagen

Die OOP unterteilt sich im Wesentlichen in die Phasen objektorientierte Analyse (OOA) und objektorientiertes Design (OOD). Im Rahmen der OOA wird ein statisches Modell entwickelt, das die Bausteine des Systems mit ihren Attributen und Methoden sowie deren Beziehungen untereinander beschreibt. Insbesondere ist bereits in der OOA darauf zu achten, das Abbild des realen Systems so zu erstellen, dass eine Implementierung in einer Programmiersprache ermöglicht wird. Die logischen Aspekte der Modellierung beinhalten sowohl Objektstruktur, wie auch Modul- und Prozessstruktur. Ziel des OOD ist darauf aufbauend die Umsetzung der Ergebnisse der Analysephase unter den geforderten, technischen Randbedingungen. Im Idealfall werden nur die bereits identifizierten Bausteine aus fachlicher Sicht um DV-technisch benötige Bausteine ergänzt [nach GaHe01]. Durch diese Vorgehensweise wird implizit die eigentliche Funktionslogik von der Gestaltung der Oberfläche und der Speicherung der Daten getrennt, was neben der Erhöhung der Flexibilität die Wartbarkeit solcher Systeme erheblich vereinfacht.

Zumindest konzeptionell arbeitet ein objektorientiertes Programm demzufolge nicht mehr sequenziell einzelne Funktionsbereiche eines Algorithmus ab. Die Programmlogik entfaltet sich vielmehr in der Kommunikation und den internen Zustandsveränderungen der Objekte, aus denen das Programm besteht. Vorteile der objektorientierten Programmierung liegen hierbei in der besseren Modularisierung des Codes und dadurch bedingt in einer höheren Wiederverwendbarkeit der Einzelmodule sowie einer gesteigerten Flexibilität der gesamten Software, insbesondere in Bezug auf die Benutzerführung. Programme dieser Art sind weniger stark gezwungen, dem Anwender bestimmte Bedienabläufe aufzuzwingen. Die einzelnen Bausteine, aus denen ein objektorientiertes Programm während seiner Abarbeitung besteht, werden als Objekte bezeichnet. Die Konzeption dieser Objekte erfolgt in der Regel auf Basis der Paradigmen Abstraktion, Kapselung, Polymorphie und Vererbung.

Zur besseren Verwaltung gleichartiger Objekte bedienen sich die meisten Programmiersprachen des Konzeptes der *Klasse*. Klassen sind Vorlagen, aus denen Objekte zur Laufzeit erzeugt werden, so genannte Instanzen einer Klasse. Im Programm werden daher nicht einzelne Objekte, sondern eine Klasse gleichartiger Objekte definiert. Die Klasse entspricht in etwa einem komplexen Datentyp; sie legt aber nicht nur die Datentypen der Attribute fest, aus denen die erzeugten Objekte bestehen, sie definiert

darüber hinaus die Algorithmen, die auf diesen Daten operieren. Während also zur Laufzeit eines Programms einzelne Objekte miteinander interagieren, wird das Grundmuster dieser Interaktion durch die Definition der einzelnen Klassen festgelegt. Klassen können den Status und das Verhalten einer anderen Klasse durch Vererbung übernehmen. Die so entstehende Hierarchie entspricht einer IS-A-Hierarchie. Die erbende Klasse, auch *Subklasse* oder Kindklasse, ist von der Art der vererbenden Klasse, auch als Vater- oder *Superklasse* bezeichnet.

Klassen werden in der Regel in Form von *Klassenbibliotheken* zusammengefasst, die häufig thematisch organisiert sind. So können Anwender einer objektorientierten Programmiersprache Klassenbibliotheken erwerben, die beispielsweise den Zugriff auf Datenbanken ermöglichen.

Die einer Klasse von Objekten zugeordneten Algorithmen bezeichnet man als *Methoden*. Die Gesamtheit der Methoden eines Objektes oder einer Klasse definiert das Verhalten eines Objekts. Sie stellen definierte Schnittstellen zum Zugriff auf den Zustand des Objektes dar. Methoden werden von anderen Objekten aufgerufen. Dieser Vorgang wird auch als Übergabe einer Nachricht bezeichnet. Die Parameter des Methodenaufrufs als Inhalte der übergebenen Nachrichten sind wiederum Objekte. Häufig wird der Begriff Methode synonym zu Funktion oder Prozedur gebraucht, obwohl die Funktion oder Prozedur eher als Implementierung einer Methode zu betrachten ist. Eine besondere Rolle spielen Methoden für das Designparadigma der Kapselung (vgl. Abschnitt 3.5.2.2). Spezielle Methoden zur Erzeugung bzw. "Zerstörung" von Objekten heißen Konstruktoren und Destruktoren.

Im folgenden Abschnitt wird speziell auf die objektorientierten Design-Paradigmen eingegangen. Das Paradigma der Abstraktion wurde oben bereits näher beschrieben. Jedes Objekt im System kann dementsprechend als ein abstraktes Modell eines Akteurs betrachtet werden, der Aufträge erledigen, seinen Zustand berichten und ändern kann und mit den anderen Objekten im System kommuniziert, ohne offen zu legen, wie diese Fähigkeiten implementiert sind (Black-Box).

### 3.5.2.2 Paradigmen

**Kapselung**

> *„Als Kapselung bezeichnet man das Verbergen von Implementierungsdetails. Der direkte Zugriff auf die interne Datenstruktur wird unterbunden und erfolgt stattdessen über definierte Schnittstellen.*"[nach GaHe01]

Von der internen Repräsentation eines Objektes soll der Verwender (hier sowohl die Algorithmen, die mit den Objekten arbeiten, als auch der Programmierer, der diese entwickelt) möglichst wenig wissen müssen (Geheimnisprinzip). Objekte können den internen Zustand anderer Objekte nicht in unerwarteter Weise lesen oder ändern. Ein Objekt hat eine Schnittstelle, die darüber bestimmt, auf welche Weise mit dem Objekt interagiert werden kann. Dies verhindert das Umgehen von Invarianten[13] des Programms. Denn durch die Kapselung werden über die Schnittstelle nur Informationen

[13] *„Eine Invariante ist eine Zusicherung, die … immer gültig ist."* [Balz05]

über den Leistungsumfang eines Objektes nach außen sichtbar, nicht aber deren interne Repräsentation. Dadurch wird eine Schnittstelle nach außen definiert und zugleich dokumentiert.

Aus dem Paradigma der Kapselung ergibt sich eine Reihe von Vorteilen. Durch die Umsetzung des Geheimnisprinzips kann die interne Implementierung geändert werden, ohne die Zusammenarbeit mit anderen Objekten zu beeinträchtigen, da die Schnittstelle konstant bleibt. Für den Verwender ergibt sich eine erhöhte Übersichtlichkeit und Komplexitätsreduktion. Lediglich Informationen über den Gebrauch der Schnittstelle sind für ihn relevant. Darüber hinaus rufen auch interne Änderungen an einem gekapselten Objekt keine Folgefehler in anderen Programmteilen hervorrufen. Beim Zugriff über die Schnittstellen spielt es für den Verwender keine Rolle, ob diese Funktion 1:1 im Inneren des Objekts existiert, das Ergebnis einer Berechnung ist, oder möglicherweise aus anderen Quellen (z.B. einer Datei oder Datenbank) stammt. Durch die separaten Objekte erreicht der Programmierer insgesamt eine deutlich verbesserte Testbarkeit, Stabilität und Änderbarkeit der Software.

**Polymorphie**

> *„Polymorphie („Vielgestaltigkeit") ist ein Konzept der Programmierung, das es erlaubt, einem Wert oder einem Namen (z.B. einer Variablen) mehrere Typen zuzuordnen.*"[nach GaHe01]

Verschiedene Objekte können auf die gleiche Nachricht also unterschiedlich reagieren. Wird die Zuordnung einer Nachricht zur Reaktion auf die Nachricht erst zur Laufzeit aufgelöst, dann wird dies auch späte Bindung (oder dynamische Bindung) genannt. In älteren typisierten Programmiersprachen wird jedem Namen und jedem Wert im Quelltext eines Programms höchstens ein Typ zugeordnet. Polymorphie kann zwischen universeller Polymorphie und Ad-hoc Polymorphie (auch Überladen) unterteilt werden. Universelle Polymorphie unterscheidet sich von Ad-hoc-Polymorphie in mehreren Aspekten. Bei Ad-hoc-Polymorphie kann ein Name oder ein Wert nur endlich viele verschiedene Typen besitzen. Diese sind zudem zur Übersetzungszeit bekannt. Universelle Polymorphie dagegen erlaubt es, unendlich viele Typen zuzuordnen. Ein weiterer Unterschied liegt darin, dass die Implementierung einer universell polymorphen Funktion generell gleichen Code unabhängig von den Typen ihrer Argumente ausführt, während ad-hoc-polymorphe (also überladene) Funktionen abhängig von den Typen ihrer Argumente unterschiedlich implementiert sein können.

**Vererbung**

> *„Die Definition eines neuen Objektes kann gegebenenfalls auf der Definition eines bereits vorhandenen Objektes aufbauen, so dass das neue Objekt die Merkmale des vorhandenen übernimmt und um neue Bestandteile ergänzt. Die Übernahme der Merkmale des vorhandenen Objektes bezeichnet man als Vererbung*."[ nach GaHe01]

Neue Klassen von Objekten können auf der Basis bereits vorhandener Objekt-Klassen festgelegt werden. Neue Bestandteile können in der Kindklasse hinzugenommen werden. Wird keine Vererbung zugelassen, so wird zur Unterscheidung oft von objektbasierter Programmierung gesprochen. Die Überdeckung eines neuen Merkmals über ein bei der

Vererbung übernommenes Merkmal wird als Überschreiben bezeichnet. Die Nutzung der Vererbung bietet sich an, wenn es Objekte gibt, die konzeptionell aufeinander aufbauen. Gegebenenfalls lassen sich Objektdefinitionen von vorneherein so aufteilen, dass identische Merkmale in der Definition eines "vererbenden" Objektes zusammengefasst werden. Vererbung bildet mit diesem Verfahren also eine IS-A-Hierarchie ab. Vererbungsbeziehungen zwischen den Objekten werden in der Regel durch Klassendefinitionen hergestellt. Die "vererbende" Klasse wird als Basisklasse oder auch Superklasse und die "erbende" Klasse als abgeleitete Klasse bzw. Subklasse betitelt.

Objektorientierte Design-Paradigmen können den Anwender beim Software-Entwurf also unterstützen, in dem sie helfen, den Modellierungsprozess und das eigentliche Software-Programm zu strukturieren und seine Komplexität so zu reduzieren, dass ein fehlerfreies Entwerfen deutlich erleichtert wird. Die grundlegende Denkweise ist streng objektorientiert, orientiert sich also an der für uns Menschen leicht nachvollziehbaren Klassifizierung und Typisierung von Objekten zu Gruppen gleichartiger Objektklassen. In den letzten Jahren wurden immer wieder Anstrengungen unternommen, die Modellierung von Software unter Zuhilfenahme grafischer Werkzeuge zu unterstützen. Mit der im folgenden Abschnitt beschriebenen Unified Modelling Language (UML) existiert ein solcher Standard, der während der Entwurfs- und Entwicklungsphase des in dieser Arbeit angestrebten Werkzeugs angewendet werden soll.

### 3.5.3 Grafisches Software-Design mittels der UML

Im folgenden Abschnitt soll einen Überblick über die Unified Modelling Language (UML) als fundamentale Beschreibungssprache für die Modellierung von Software und anderen Systemen und deren Einsatzmöglichkeiten gegeben werden. Um zu verhindern, dass während des meist komplexen Softwareentwicklungsprozesses fundamentale Fehler gemacht werden (Entwicklung des falschen Softwareproduktes bzw. falsche Entwicklung des Softwareproduktes) ist die Intention der UML die Ausarbeitung jedes benötigten Bearbeitungsschrittes durch grafische Elemente. Damit auch Projektbeteiligte, die keine Softwareentwickler sind, den Inhalt des Softwareprojektes verstehen, soll mit ihr eine allgemein verwendete Modellierungssprache eingesetzt werden.

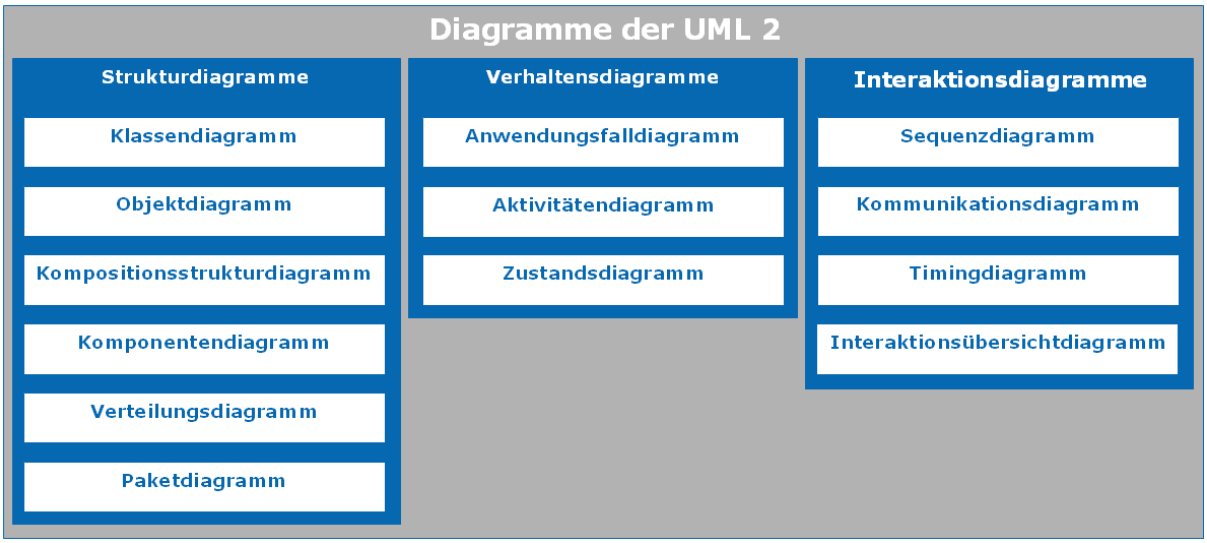


**Abbildung 12: Übersicht über die Diagrammtypen der UML**

Durch ihre ständige Weiterentwicklung ist die UML mittlerweile als Version 2.0 freigegeben und hat sich als „Quasi-Standard" durchgesetzt. Die Diagrammtypen der UML lassen sich auf die Klassen Struktur-, Verhaltens- und Interaktionsdiagramme aufteilen und werden in den nachfolgenden Abschnitten teilweise vorgestellt (vgl. auch [OOSE] und [Kech05]). Bei der Darstellung der einzelnen Diagrammtypen wird sich nachfolgend an der Beschreibung nach [Kech05] orientiert.

#### 3.5.3.1 Grundlagen MDSD und MDA

Modellgetriebene Entwicklung (MDSD) und Model Driven Architecture (MDA) werden für Unternehmen immer attraktiver. Mit Hilfe von MDSD wird Software nicht mehr „traditionell" programmiert, sondern aus Modellen teilweise oder mitunter vollständig generiert. Diese Modelle müssen jedoch zunächst erstellt werden. Dazu werden Modellierungssprachen eingesetzt wie z.B. UML 2.0, welche sich in großen Softwareprojekten durchgesetzt hat, jedoch nicht ausschließlich auf solche beschränkt ist. Mit der durch die UML 2.0 zur Verfügung gestellten Diagramme und Notationselemente lassen sich statische und dynamische Aspekte verschiedenster Anwendungsgebiete modellieren [Stah05]. UML 2.0 wird in Softwareprojekten häufig zur Unterstützung bei der Erstellung eines Pflichtenhefts eingesetzt. Das erspart dem Programmierer später wertvolle Zeit und dem Unternehmen nicht zuletzt Ressourcen, wenn beispielsweise vom Kunden in letzter Minute noch Änderungen an der Software gefordert werden. Als Vorteile von MDSD ergeben sich somit unter anderem

- größere Entwicklungseffizienz
- bessere Integration der Fachexperten
- leichtere Änderbarkeit von Software
- verbesserte (Umsetzung der) Softwarearchitektur
- die Möglichkeit, Fachlogik leichter auf andere Plattformen portieren zu können

Der Ansatz der Model Driven Architecture (MDA) basiert auf der Annahme, dass für die Konstruktion eines Softwaresystems ein einziges Modell für die Abbildung einer größeren, komplexeren Applikation zu unscharf und überladen ist. Bei den meisten "klassischen" (UML)-Modellen sind geschäftsrelevante und technische Informationen vermischt. MDA unterteilt das Gesamtmodell in mehrere Schichten (CIM[14], PIM[15], PSM[16], ein Codemodell und die Zielplattform). Die Trennung der Modelle stellt eine inhaltliche Erweiterung des UML-Standards dar. Insbesondere durch das CIM und vor allem das PIM soll nicht nur Plattformunabhängigkeit gewährleistet werden, sondern auch die Sprach- und System-unabhängigkeit. MDA definiert neben der inhaltlichen Trennung der Modelle auch die Transformation der Modelle und unterscheidet zwei Typen:

- Die Modelltransformation von einem Modell in ein anderes Modell
- Die Codetransformation von einem Modell in den Code

[14] CIM: hier (Computation Independent Model) als umgangssprachliche Beschreibung [Stah05]

[15] PIM (plattformunabhängiges Modell, Platform Independent Model) Abbildung der Geschäftsprozesse [Stah05]

[16] PSM (plattformabhängiges Modell, Platform Specific Model) für Architektur, Services [Stah05]

Die Transformationen erzeugen aus den Elementen des Quellmodells die Elemente des Zielmodells. Die Transformation geschieht üblicherweise von der abstrakteren Ebene in die konkretere Ebene (CIM-PIM-PSM-Code). Dadurch kann aus einfacheren Modellelementen eine komplexere Anwendung erzeugt werden, indem erfahrene Architekten ihre Konstruktionsregeln in solche Transformationsprozesse einprogrammieren [Stah05].

MDA ist ein junger Standard der Object Management Group. Ein Ziel der MDA ist die Steigerung der Entwicklungsgeschwindigkeit. Das Mittel dazu heißt Automation durch Formalisierung. Aus formal eindeutigen Modellen soll durch Generatoren automatisch Code erzeugt werden. Dadurch soll auch die Softwarequalität gesteigert werden. Fehler in den generierten Codeanteilen können an einer Stelle - in den Generatorschablonen - beseitigt werden. Die Qualität des generierten Quellcodes ist gleich bleibend, was zu einem höheren Grad der Wiederverwendung führen soll. Ein weiteres wesentliches Ziel ist die bessere Handhabbarkeit von Komplexität durch Abstraktion. Mit den Modellierungssprachen soll Programmierung auf einer abstrakteren Ebene möglich werden, die klare Trennung von fachlichen und technischen Anteilen zielt auf eine bessere Wartbarkeit durch Trennung von Verantwortlichkeiten ab. Die abstraktere, technologieunabhängige Beschreibung von Schlüsselkonzepten mit eindeutigen Modellierungssprachen verspricht eine verbesserte Handhabbarkeit des Technologiewandels. Und nicht zuletzt soll eine verbesserte Interoperabilität durch Standardisierung erreicht werden [Völt-ol].

#### 3.5.3.2 Diagrammarten der UML

Einzelne Diagrammtypen der UML, die in der Konzeptionsphase dieser Arbeit verwendet werden sollen, werden nachfolgend kurz erläutert.

**Strukturdiagramme**

Strukturdiagramme modellieren statische, zeitunabhängige Elemente eines Systems. Unter diese Diagrammart fallen Klassendiagramme, Objektdiagramme, Kompositionsstrukturdiagramme, Komponentendiagramme, Verteilungsdiagramme und Paketdiagramme.

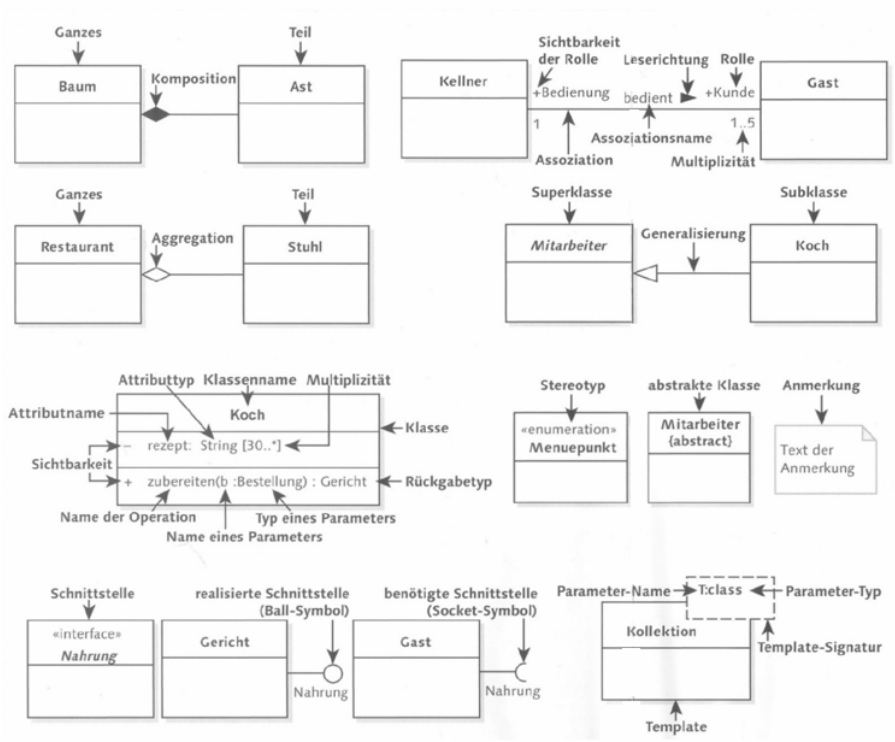


**Abbildung 13: Notationselemente des Klassendiagramms**

Ein Klassendiagramm ist eine Art „Bauplan", um Instanzen für Objektdiagramme zu erzeugen. Instanzen sind konkrete Ausprägungen der jeweiligen Klasse. Klassendiagramme legen Attribute, Operationen und ihre Beziehungen zueinander fest. Programmiersprachlich ausgedrückt bedeutet dies, dass die Attribute einer Klasse ihre Variablen, die Operationen ihre Methoden und Funktionen und Assoziationen die Beziehungen zu anderen Klassen darstellen.

Wie bereits erwähnt sind Objektdiagramme eine Spezifizierung ihrer jeweiligen Klasse. Das heißt, Objekte nehmen konkrete Werte an (Beispiel-Bild). Ein Objekt hat einen Objektnamen (nicht bei unbenannten Objekten) und gibt die Attributwerte zu den in der Klasse definierten Attributen an. Zwischen Objekten können ebenfalls Beziehungen bestehen (sog. „Links"). Ein Link ist die konkrete Ausprägung einer Assoziation mit einem Linknamen und einer Leserichtung. Die Multiplizität ist höchstens 1, da ein Link immer genau zwei Objekte verbindet. Kollaborationen *„[...] beschreiben Strukturen von Objekten, die in ihren speziellen Rollen kollektiv gewünschte Funktionalitäten bereitstellen sowie die Verbindungen (Konnektoren) der Objekte untereinander.*" [Kech05].

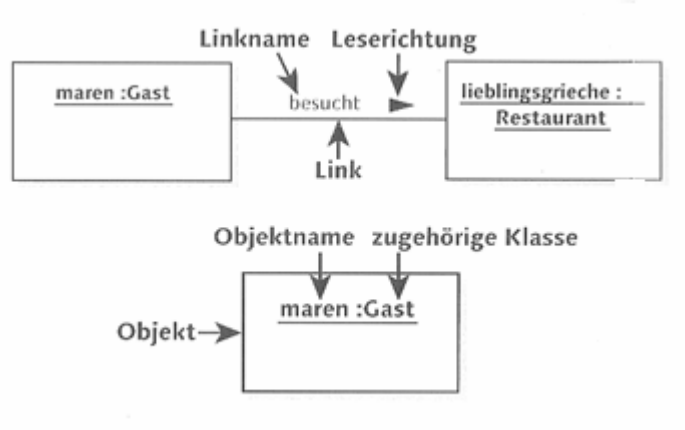


**Abbildung 14: Notationselemente des Objektdiagramms**

Verteilungsdiagramme spezifizieren die physische Hardware- und Softwareumgebung und die Verteilung der Komponenten in dieser Umgebung. Notationselemente in diesem Diagramm sind Knoten und Kommunikationspfade. Ein Knoten besitzt einen Namen und stellt eine Systemressource dar. Knoten können weitere Knoten enthalten.

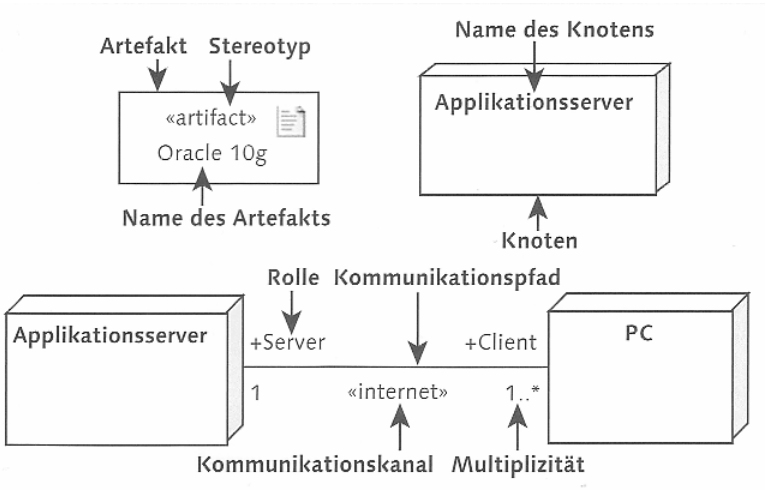


**Abbildung 15: Notationselemente des Verteilungsdiagramms**

**Verhaltensdiagramme**

Bei Verhaltensdiagrammen liegt das Hauptaugenmerk auf den dynamischen Aspekten eines Systems. Sie beschreiben das Verhalten. Hierunter fallen Use-Case-Diagramme, Aktivitätendiagramme und Zustandsdiagramme.

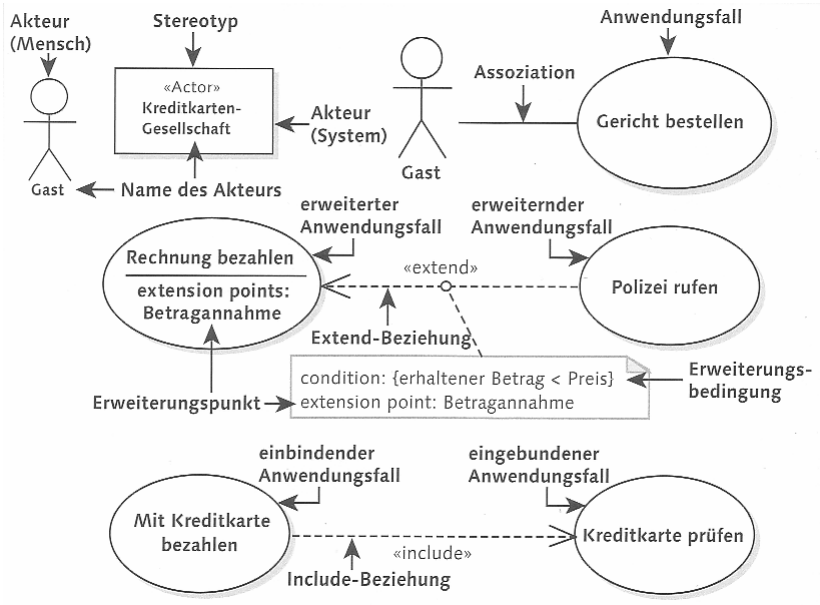


**Abbildung 16: Notationselemente des Use-Case-Diagramms**

Diese Diagrammart stellt die Sicht auf das System durch externe Anwender dar. Das heißt es gibt Akteure, die bestimmte Rollen innehaben und mit dem System interagieren. Ein Akteur weiß nichts von den systeminternen Abläufen. Er kennt nur die für ihn relevanten Abläufe. Der Akteur „sieht" nun einen Use-Case als so genannte Black-Box. Das heißt, dass die Menge der Aktionen innerhalb des Use-Cases für den Akteur verborgen bleiben. Zwischen den Akteuren und Use-Cases können Assoziationen mit entsprechenden Multiplizitäten modelliert werden. In Use-Case-Diagrammen kann auch Vererbung durch Generalisierung modelliert werden. Das heißt, ein Akteur erbt die Rechte seiner übergeordneten Rolle. Eine *include*-Beziehung sagt aus, dass falls Use-Case 1 ausgeführt wird auch Use-Case 2 ausgeführt werden muss. Im Gegensatz dazu muss bei einer *extend*-Beziehung nicht zwingend Use-Case 2 ausgeführt werden. Use-Case 1 kann aber durch Use-Case 2 erweitert werden.

Um das Verhalten eines Systems abzubilden werden Aktivitätendiagramme eingesetzt. Diese Diagrammart ist sehr vielseitig und es können, nicht zuletzt aufgrund der zahlreichen Notationselemente, viele Situationen damit modelliert werden. Dazu gehören unter anderem alternative Abläufe, Reihenfolgen von Aktivitäten und parallele Aktivitäten. Aus diesem Grund wird im Folgenden nur auf die gängigsten Notationselemente eingegangen. Eine Aktion *„[...] stellt die fundamentale Einheit ausführbarer Funktionalität dar, die im Modell nicht weiter zerlegt wird [...]*" [Kech05].

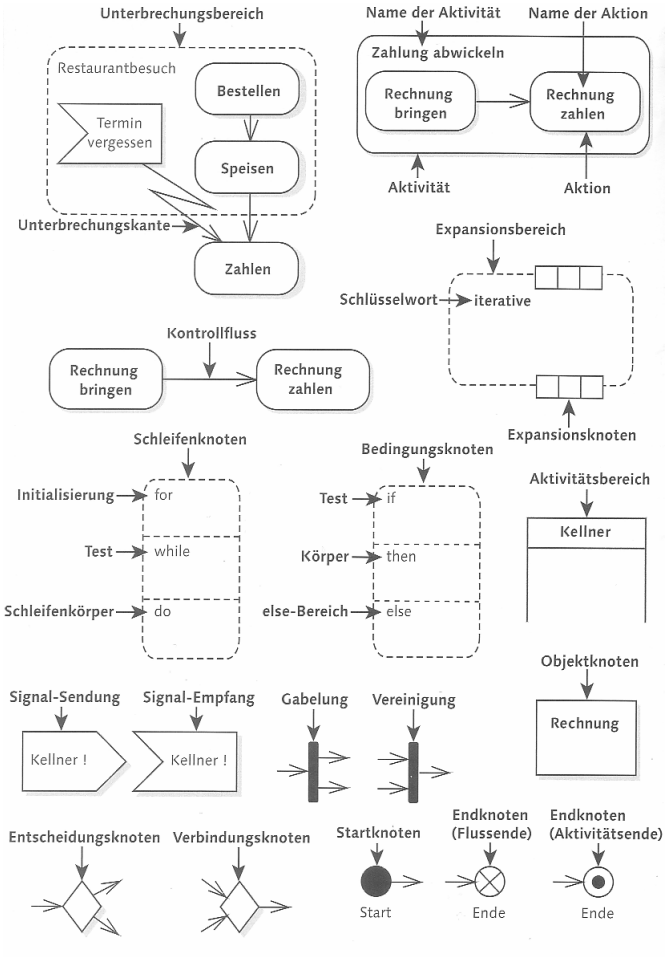


**Abbildung 17: Notationselemente des Aktivitätendiagramms**

Die Ausführungsreihenfolge zwischen modellierten Aktionen wird durch gerichtete Kanten, den Kontrollflüssen dargestellt. Damit das Diagramm durch sehr viele Kontrollflüsse nicht unübersichtlich wird, können Konnektoren mit jeweils eindeutigen Namen und einem Kontrollfluss benutzt werden. Das Diagramm bekommt eine noch besser lesbare Struktur, wenn Aktivitätsbereiche (sowohl horizontal als auch vertikal) benutzt werden, die zugehörige Aktionen zu Organisationseinheiten zusammenschließen. Zur Übergabe von Objekten zwischen Aktionen werden Objektknoten eingesetzt, die als Speicher der zugehörigen Klasse betrachtet werden. Weiterhin lassen sich so Streams und Puffer realisieren. Eine Aktivität beschreibt eine Folge von Aktionen. Start- und Endknoten geben die Einstiegs- und Ende-Aktion einer Aktivität vor. Durch Entscheidungsknoten wird ein Kontrollfluss verzweigt. An diese Verzweigungen können Bedingungen geknüpft sein, so dass sich alternative Wege modellieren lassen. Verbindungsknoten führen die verschiedenen Flüsse wieder zusammen. Parallele Aktionen werden durch Gabelungen und Vereinigungen dargestellt. „Eine Gabelung teilt einen Kontrollfluss in mehrere parallele Kontrollflüsse auf", während eine Vereinigung diese wieder zusammenfasst.

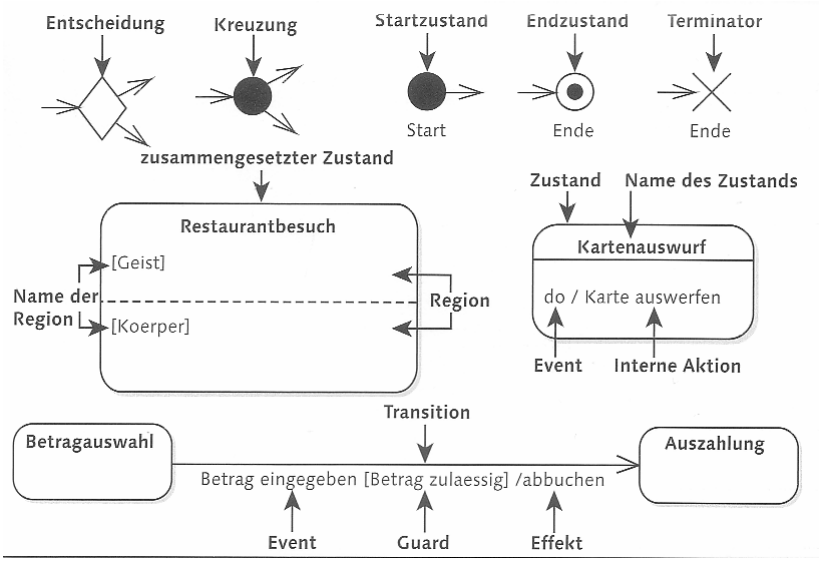


**Abbildung 18: Notationselemente des Zustanddiagramms**

Mit Zustandsdiagrammen werden die Reaktionen eines Systems auf Ereignisse dargestellt. Die wichtigsten Notationselemente sind Zustände, Transitionen, Events, Start- und Endzustände und Terminator, Entscheidungen und Kreuzungen und Regionen. Ein Zustand hat einen Namen und *„[...] modelliert eine Situation, in der gewisse genau definierte Bedingungen gelten*" [Kech05]. Um von einem Zustand in einen anderen zu gelangen ist eine Transition, das heißt eine gerichtete Kante, zwischen Zuständen erforderlich, damit diese schalten kann, sobald ein entsprechendes Event ausgelöst wurde. Es werden fünf verschiedene Eventtypen unterschieden, auf die an dieser Stelle aber nicht weiter eingegangen werden soll. Start- und Endzustände werden ähnlich wie bei Aktivitätendiagrammen modelliert. Sobald ein Terminator erreicht wird endet der komplette Zustandsautomat. Ist der Endzustand erreicht wird nur die Ausführung einer Ebene von Zuständen beendet. Kreuzungen arbeiten wie Verzweigungen, an die auch Bedingungen gebunden sein können und modellieren *„[...] eine Hintereinanderschaltung von Transitionen*"[Kech05]. Entscheidungen können ebenfalls wie Kreuzungen an Bedingungen gebunden sein, modellieren aber dynamische Verzweigungen. Zustände können auch andere Zustände beinhalten, um somit zusammengesetzte Zustände zu modellieren. Durch Regionen lassen sich Zustandsautomaten in disjunkte Bestandteile aufteilen. Somit kann jede Region ihre eigenen Start- und Endzustände haben. Der Zustandsautomat kann aber erst verlassen werden, wenn in allen Regionen der jeweilige Endzustand erreicht ist.

**Interaktionsdiagramme**

Als eine Untergruppe der Verhaltensdiagramme konzentrieren sich die Interaktionsdiagramme auf die Interaktionen und den Nachrichtenaustausch zwischen Objekten. Sequenzdiagramme eignen sich, um den zeitlichen Ablauf des Nachrichtenaustauschs zwischen Objekten zu modellieren.

Die wichtigsten Notationselemente sind Interaktionsrahmen, Lebenslinien und Nachrichten. Jedes Interaktionsdiagramm lässt sich in einem Interaktionsrahmen modellieren und kapselt damit „eine Verhaltensdefinition, deren Fokus auf der Darstellung eines Informationsaustauschs liegt". Ein Teilnehmer einer Interaktion wird

durch eine Lebenslinie dargestellt. Nachrichten dienen der Kommunikation zweier Teilnehmer und geben die Flussrichtung an.

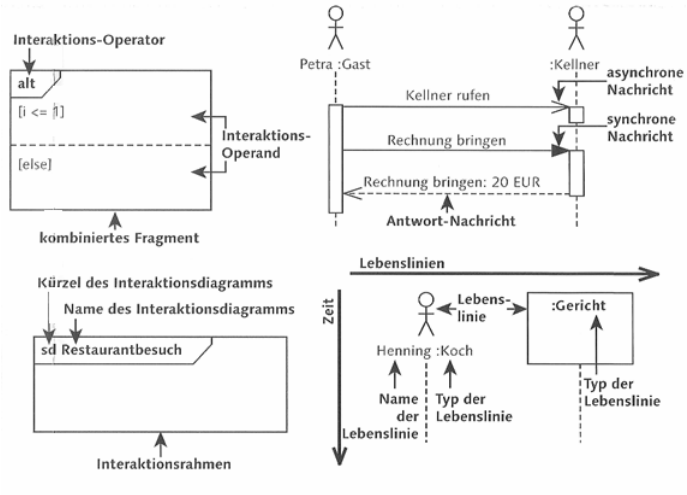


**Abbildung 19: Notationselemente des Sequenzdiagramms**

In ihrer Gesamtheit bilden alle Diagrammtypen die UML 2.0 (vgl. Abbildung 12). Sie kann zur Konzeption und Modellierung von Softwaresystemen verwendet werden und steht somit in Kapitel 1 als Hilfsmittel zur Darstellung der Entwicklung des Werkzeugs zur Verfügung.

### 3.5.4 Mehrbenutzersysteme

Die vorangegangenen Abschnitte 3.5.1 und 3.5.2 haben allgemeine wie objektorientierte Prinzipien des Systementwurfs aus softwaretechnischer Sicht diskutiert und eine Beschreibungssprache zur grafischen Modellierung und Implementierung von Softwareprogrammen vorgestellt. Im folgenden Abschnitt wird näher auf die Anforderungen bezüglich der Funktionalitäten des Systems eingegangen. Zwischen menschlicher und technischer Betrachtungsweise wird implizit unterschieden. Die Abschnitte 3.5.4.1 bis 3.5.4.3 betrachten die Funktionalitäten mehr aus Anwendersicht, bzw. der Sicht des Systementwicklers, wohingegen sich Abschnitt 3.5.4.4 auf Funktionalitäten aus technischer Sicht fokussiert. Menschliche Anforderungen an kooperative Modellierungswerkzeuge gründen z.B. auf der Notwendigkeit der effektiven Zusammenarbeit bei der Benutzung eines solchen Systems. Werden diese Anforderungen bei der Systementwicklung vernachlässigt, kann ein zwar im technischen Sinne funktionsfähiges System herauskommen, das aber den Bedürfnissen der Anwender nicht gerecht und daher von diesen nicht angenommen wird. Wie das System technischen Anforderungen gerecht wird, nehmen Anwender zumeist nur dann wahr, wenn die Realisierung Schwächen aufweist. Für den Systementwickler ist die genaue Kenntnis der technischen Notwendigkeiten aber von Bedeutung, weil von ihnen abhängt, ob überhaupt ein funktionsfähiges System erstellt werden kann. Die Entwicklung eines funktionsfähigen Werkzeugs mit breiter Akzeptanz erfordert daher, sowohl die technischen als auch die anwenderabhängigen Anforderungen an das System in der Entwurfphase genau zu kennen und zu beachten.

### 3.5.4.1 Verteilte Systeme

*„Ein Verteiltes System ist ein Zusammenschluss unabhängiger Computer, welcher sich für den Benutzer als ein einzelnes System präsentiert.*"[nach TaSt03]

Das gemeinschaftliche Zusammenwirken mehrerer Anwender am Prozess der Erstellung eines Fabrikmodells erfordert die Nutzung verteilter (EDV-)Systeme, weil diese besser als monolithische Systeme der gegebenen räumlichen Trennung der Angehörigen unterschiedlicher betrieblicher Bereiche und der von ihnen genutzten und in die Modellierung eingebrachten Daten gerecht werden. Sie erlauben die gleichzeitige Benutzung der benötigten materiellen (Hardware) und immateriellen (Software) Ressourcen und gewährleisten den heterogenen Ansprüchen unterschiedlicher menschlicher Anwender und/oder unterschiedlicher Komponenten des Programmsystems gerecht zu werden und eine variable Anpassung der Systemgröße an unterschiedliche Aufgabestellungen (Skalierbarkeit) umzusetzen. Allgemeine Ziele der Nutzung verteilter Systeme sind eine Leistungssteigerung hinsichtlich des Durchsatzes und den Antwortzeiten des Systems durch die Bereitstellung zusätzlicher Hardware, eine bessere Erweiterbarkeit und Anpassungsfähigkeit sowie eine erhöhte Fehlertoleranz. Die Verbundstruktur verteilter Systeme bietet das Mittel zur Erreichung dieser Ziele [CoDo02]. Sie umfasst die folgenden Substrukturen:

- Funktionsverbund: Die Gesamtfunktionalität wird erbracht, indem dedizierte Funktionen der Anwendung auf unterschiedlichen Rechnern integriert werden.
- Datenverbund: Der Zugriff auf verteilte Datenbestände von verschiedenen Rechnern aus ermöglicht es, Datensätze aus unterschiedlichen Dateisystemen oder Datenbanken miteinander zu verknüpfen und integriert zu verarbeiten.
- Lastverbund: Das verteilte Leistungspotential mehrerer Rechner soll so genutzt werden, dass die durch eine Anwendung gegebene Last möglichst gleichförmig verteilt wird.

In der Literatur wird, je nach Perspektive des Autors, häufig die Bezeichnung „verteiltes System" ohne Zusätze für verteilte Hardwaresysteme, verteilte Betriebssysteme oder verteilte Anwendungssysteme benutzt. Im Folgenden wird die Bezeichnung „verteiltes System" als Sammelbezeichnung für diese drei Spezialisierungen benutzt, wenn nicht auf einen bestimmten Aspekt der Verteilung (Hardware, Betriebssystem, Anwendungen) Bezug genommen wird. Das ist sinnvoll, weil Hardware, Betriebssystem und Anwendungen drei Ebenen der Gesamtarchitektur darstellen, zwischen denen eine enge Abhängigkeit besteht: Die Nutzung verteilter Betriebssysteme oder verteilter Anwendungssysteme ergibt nur auf entsprechender Hardware Sinn.

### 3.5.4.2 CSCW und Groupware

*„Computer Supported Cooperative Work (CSCW) ist die Bezeichnung des Forschungsgebietes, welches auf interdisziplinärer Basis untersucht, wie Individuen in Arbeitsgruppen oder Teams zusammenarbeiten und wie sie dabei durch Informations- und Kommunikationstechnologie unterstützt werden können.*" [FiHe00]

Im Forschungsgebiet des CSCW sollen, unter Verwendung aller zur Verfügung stehenden Mittel der Informations- und Kommunikationstechnologie, Gruppenprozesse zu untersuchen und die Effektivität und Effizienz der Gruppenarbeit zu erhöhen. Im Mittelpunkt steht die Konzeption, Implementierung und Erweiterung von Werkzeugen für die Unterstützung der Teamarbeit. Die Wurzeln dieses Ansatzes sind unter anderem in Entscheidungsunterstützungs- und Kommunikationssystemen zu sehen. Die Hilfsmittel für die Kooperation innerhalb von Gruppen und Teams werden als *Groupware* oder *Workflow-Management-Systeme* bezeichnet; dies schließt sowohl Hardware (beispielsweise Kameras) als auch Software ein [FiHe00].

Eine weit verbreitete Taxonomie für Groupware-Systeme liefert die Raum-Zeit-Matrix nach [Joha91], die in Abbildung 20 abgebildet ist. Die Klassifikation liefert vier verschiedene Arten von Groupware-Systemen. Bei der *face-to-face-interaction* befinden sich die Kooperationspartner zur selben Zeit am selben Ort. Als Beispiel können Group Decision Support Systems dienen, die Entscheidungsprozesse in Gruppen durch persönliche Arbeitsplatzrechner und einen für alle sichtbaren Großbildschirm unterstützen. In die Kategorie der *synchronen verteilten Interaktion* fallen Mehrbenutzer- oder auch Gruppeneditoren, also auch kooperative Modellierungswerkzeuge. *Asynchrone Interaktion* findet am selben Ort, aber zu unterschiedlichen Zeiten statt und kann z. B. als elektronische Version einer Pinnwand realisiert werden. E-Mail-Systeme sind das klassische Beispiel für *asynchrone verteilte Interaktion*.

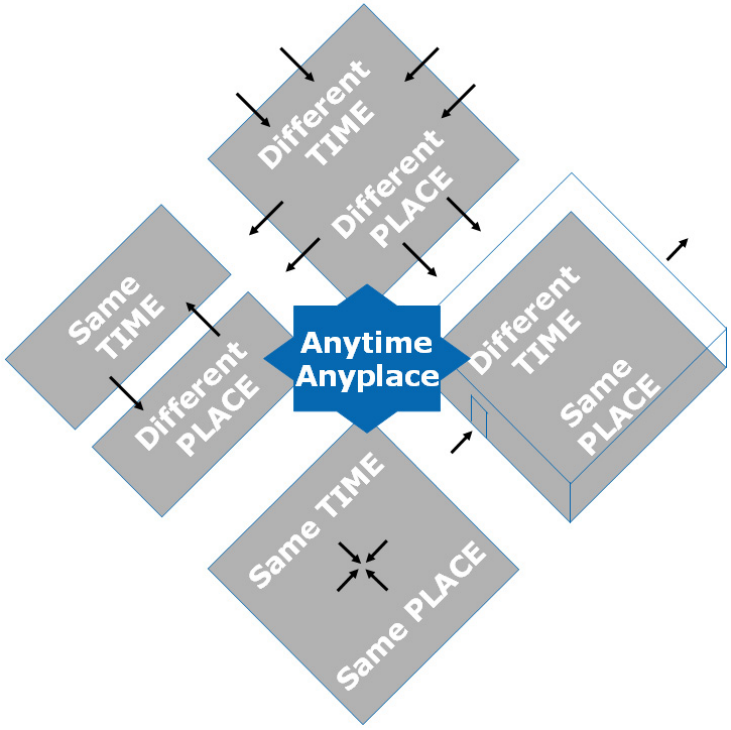


**Abbildung 20: Raum/Zeit Matrix nach [Joha91]**

Problematisch ist bei dieser Einteilung jedoch die fehlende eindeutige Zuordnungsmöglichkeit bestimmter Applikationen zu einzelnen Kategorien. Geeignet ist eine solche Darstellung eher für die Klassifikation der Verwendung von Groupware-Applikationen als für die Beschreibung der möglichen Groupware-Funktionalitäten.

Zentrale Aspekte jeder Groupware sind:

- Awareness: Viele Systeme setzen eine oder mehrere Formen der Awareness um, d. h. die Software ermittelt selbständig (implizit) Eingabedaten, um dem Anwender Zeit und Arbeit abzunehmen.
- What You See Is What I See: Das Prinzip beschreibt, welche Teile einer Anwendung bei verschiedenen Anwendern exakt gleich dargestellt werden.
- Synchronisation und Konsistenzerhaltung. Die Wahrung eines einheitlichen Datenzustandes (Konsistenz) trotz gleichzeitiger Zugriffe auf das Datenmaterial, bzw. die Visualisierung von Konflikten, wo dies nicht möglich ist.
- Floor-Control: Die Verwaltung der Systemressourcen: Welcher Teilnehmer darf gerade welche Ressource nutzen?
- Session-Control: beschreibt die Verwaltung und Administration der Teilnehmer in Arbeitssitzungen hinsichtlich Autorisierung, Authentifizierung und Einteilung in Rollenschemata [FiHe00].

### 3.5.4.3 Kommunikation, Kooperation und Koordination

Unabhängig von den eingesetzten Groupware-Technologien lassen sich die Applikationen nach ihren elementaren Unterstützungsfunktionen gliedern. Hierbei werden vor allem Kommunikations-, Kooperations- und Koordinationsfunktionalitäten unterschieden, die aber eng miteinander verknüpft sind. Hinsichtlich dieser Merkmale eingegrenzt, beschreiben Ellis et al. das Ziel von Groupware-Applikationen wie folgt: *„the goal of groupware is to assist groups in communicating, in collaborating, and in coordinating their activities*"[ FiHe00].

**Kommunikation**

> *„Kommunikation bezeichnet den Austausch von Nachrichten zwischen Menschen. Im erweiterten Rahmen der Informationstheorie versteht man darunter … jeden Austausch von Informationen zwischen dynamischen Systemen… , die in der Lage sind, Informationen aufzunehmen, zu speichern, umzuformen usw.*"[KlBu71]

Der Kommunikation kommt insbesondere bei der Teamarbeit eine entscheidende Rolle zu. Kommunikationsmechanismen bilden die unverzichtbare Basis aller Kooperations- und Koordinationsbemühungen. Bei aktiv initiierter Kommunikation steht das Send-Prinzip im Vordergrund. Kommunikation bezieht sich hier insbesondere auf Systeme, die elektronische Objekte speichern und weiterleiten können. Die in diesen Objekten gespeicherten Informationen werden möglicherweise über eine Fülle von Zwischenstationen transferiert. Dieses oft auch als Push-Modell beschriebene Prinzip erlaubt einen asynchronen Informationsfluss bei dem eine Synchronisation der kommunizierenden Elemente nicht notwendig ist. Generell kann in diesem Bereich zwischen *synchroner* und *asynchroner* Kommunikation differenziert werden [FiHe00]. Unter synchroner Kommunikation wird die zeitlich unmittelbare Übermittlung und Sichtbarkeit beim Empfänger verstanden. Demgegenüber gelangen die elektronischen Objekte bei der asynchronen Kommunikation zunächst in einen virtuellen Eingangskorb,

um zu einem späteren Zeitpunkt empfangen und ggf. verarbeitet zu werden. Darüber hinaus können Nachrichten als eine Form der elektronischen Objekte entweder *explizit* von Anwender zu Anwender übermittelt, oder *implizit*, also ohne bewusstes Wissen des Anwenders, durch das jeweilige System verschickt werden, beispielsweise als Reaktion auf eine Benutzereingabe [FiHe00].
Klassische Systeme zur Kommunikationsunterstützung sind beispielsweise E-Mail-Systeme. Die Form der Kommunikation kann weiter nach Anzahl der Absender und Anzahl der Empfänger untergliedert werden. Hierbei sind prinzipiell alle Kommunikationsarten von der 1:1 Kommunikation über die 1:n Kommunikation bis hin zur n:m Kommunikation erlaubt und finden in den entsprechenden Bereichen auch Verwendung. Aufgrund der fehlenden Strukturierung und dem zu transportierenden Informationsvolumen steigt die Komplexität dieses Modells im Rahmen der n:m Kommunikation stark an. Eine Lösung dieser Problematik kann durch den Einsatz des Pull-Prinzips erreicht werden, welches insbesondere im Bereich der Kooperations-unterstützung besondere Bedeutung erfährt.

**Kooperation**

> *„Die Kooperation baut auf der Kommunikation auf und stellt den Austausch von Informationen mit einem gemeinsamen Ziel dar. Sie bedingt, dass mindestens zwei Personen in einem gemeinsamen, zielgerichteten Kooperationsprozess involviert sind.*"[FiHe00]

Systeme zur Kooperation unterstützen die gemeinsame Arbeit einer Gruppe nach dem Share-Prinzip. Die Gruppenmitglieder haben Zugriff auf einen gemeinsamen Datenbestand, den sie in beliebiger Reihenfolge verändern und erweitern können, ohne ein vorgegebenes Ablaufschema und nicht notwendigerweise sequentiell. Applikationen zur Kooperationsunterstützung forcieren das Pull-Modell, um Informationen miteinander zu teilen, gemeinsam zu bearbeiten, zu strukturieren und somit auch weiterzuentwickeln. Jeder Nutzer kann in diesem Modell die für ihn relevanten Informationen selektieren, individuelle zusammenstellen und unabhängig von Raum und Zeit abrufen. Koordination weist den höchsten Komplexitätsgrad auf, da die jeweilige Konsistenz der Daten zu jedem Zeitpunkt gewährleistet sein muss. Kommunikationsmechanismen unterstützen die Anwender bei der Ausführung kooperativer Tätigkeiten und können Konflikte schnell und unkompliziert lösen [FiHe00].

**Koordination**

> *„Als Koordination wird diejenige Abstimmung bezeichnet, die für eine Arbeit der Kooperationspartner auf ein gemeinsames Ziel hin erforderlich ist.*"[Henk97]

Wird im Rahmen der Kooperation kommuniziert und bezieht sich diese Kommunikation auf die Abstimmung der aufgabenbezogenen Tätigkeiten, so wird diese Dimension der Kommunikation als Koordination bezeichnet. Auf Basis der Kommunikationsmechanismen ermöglicht die Koordination die notwendige Abstimmung der dezentral handelnden und entscheidenden Anwender hinsichtlich einer optimalen Zielerreichung innerhalb der Gesamtaufgabe. Sie baut damit sowohl auf der Kommunikation als auch auf der Kooperation auf und erlaubt erst den aufgabengerechten Einsatz der Ressourcen und eine effiziente Arbeit der verschiedenen Anwender in einem Team. Grade deswegen kann hier im Allgemeinen nicht auf die Mitwirkung des Anwenders verzichtet werden. Denn

zwischen den Aktivitäten der unterschiedlichen Anwender bestehen zum einen Abhängigkeiten, die nur durch die direkte Kommunikation der betroffenen Anwender aufgelöst werden können. Zum Anderen sind die Fragestellungen hinsichtlich optimaler Reihenfolge, Zulässigkeit und Zweckmäßigkeit der einzelnen Bearbeitungsschritte nicht durch das Anwendungssystems selbst aufzulösen, weil es eine Bewertung der aktuellen Situation innerhalb des Systems, beispielsweise eine Gewichtung hinsichtlich Dringlichkeit der Aufgabe, nicht vornehmen kann. Diese muss durch den Anwender des zu entwickelnden Werkzeugs erfolgen, dessen Arbeitsweise durch die Kommunikations-, Kooperations- und Koordinationsmechanismen bestmöglich unterstützt werden sollen [FiHe00].

#### 3.5.4.4 Funktionalitäten

Im folgenden Abschnitt werden die Grundlagen der informationstechnischen Realisierung kooperativer Werkzeuge diskutiert, um aus der allgemeinen Beschreibung der wesentlichen Charakteristika einer solchen Anwendung konkrete Basisfunktionalitäten abzuleiten. Es geht um die Funktionalitäten, die im Zusammenhang von Kommunikation, Nebenläufigkeit und Synchronisation in kooperativen Systemen zur Verfügung stehen. Unterscheiden lassen sie sich in den wesentlichen Bausteinen *Datenhaltung* und *Datenaustausch*. Abschließend soll kurz auf die Gestaltung einer *Benutzerschnittstelle* eines kooperativen Systems eingegangen werden.

**Datenhaltung**

Der Prozess der Modellerstellung und die Arbeit an einem Simulationsmodell erstreckt sich typischerweise über einen lang andauernden Zeitraum. Wesentliche Grundfunktion ist daher die Möglichkeit, Daten für diesen Zeitraum sicher aufbewahren zu können. Insbesondere ist davon auszugehen, dass die Lebensdauer der Daten über die Dauer der Ausführung eines Prozesses hinausgeht. In objektorientierten Systemen wird diese Eigenschaft als Persistenz von Objekten bezeichnet [Henk97].

Um ein sinnvolles Arbeiten mit dem Modellierungswerkzeug zu ermöglichen, sind Anforderungen an die Qualität und die Quantität des Datenzugriffs zu beachten: Der Zugriff darf nicht auf einen einzelnen Anwender bzw. Prozess beschränkt sein; dennoch muss die Sichtbarkeit von Daten und die Zulässigkeit von Operationen wie Lesen, Schreiben, Erzeugen und Löschen von Daten flexibel und dynamisch geregelt werden können. Das Antwortzeitverhalten des Systems muss in akzeptablen Grenzen liegen; gleiches gilt für die Ausfallsicherheit. Diese beiden Anforderungen können nicht immer von einer zentralisierten Datenhaltungskomponente erfüllt werden, so dass in diesen Fällen eine verteilte Datenhaltung zu fordern ist.

Essentiell ist die Forderung, dass der nebenläufige Zugriff mehrerer Anwender auf gemeinsame Daten nicht zu einer Inkonsistenz des Datenbestandes führen darf. Zu den Basisdiensten eines Datenhaltungssystems gehören das Entgegennehmen, Abspeichern, Ändern, Löschen, Auswählen, Identifikation und Bereitstellen von Daten sowie Verwalten von Datenbeständen [Lock93]. Die Menge der gespeicherten Daten wird als Datenbasis bezeichnet, die Erfüllung der genannten Funktionalität gewährleistet das Datenverwaltungssystem. Die Anwender greifen auf das System über die Datenhaltungsschnittstelle zu.

Das Datenhaltungssystem hat die Aufgabe, diese Transaktionen zu synchronisieren, d. h. sicherzustellen, dass jede Transaktion aus Anwendersicht genauso abläuft, als wäre sie die einzige zu dem jeweiligen Zeitpunkt ausgeführte. Das ist insbesondere dann eine nicht-triviale Aufgabe, wenn Transaktionen als Leser oder Schreiber auf dieselben Datenbasisausschnitte zugreifen. Sperren dienen nicht nur zur Synchronisation von Transaktionen, sondern können unabhängig davon auch verwendet werden, um den Zugriff der Anwender auf Datenbereiche zu regeln. Das Setzen einer Sperre (Lock) für einen Datenbasisausschnitt bewirkt, dass die Zugriffe anderer Anwender/Prozesse/-Transaktionen auf diesen Bereich eingeschränkt oder ausgeschlossen werden. Die Einschränkung des Zugriffs besteht häufig darin, dass nicht jede dieser Zugriffsarten erlaubt wird, wenn eine Sperre gesetzt ist. Beispielsweise kann das Lesen eines Datums auch anderen Transaktionen gestattet werden, wenn der Sperrhalter selbst dieses Datum ebenfalls lediglich zu lesen beabsichtigt. Dadurch kann der Nebenläufigkeitsgrad der Transaktionsbehandlung wesentlich gesteigert werden.

**Datenaustausch**
In einem kooperativen System besteht häufig die Notwendigkeit, eine Anzahl von Kommunikationspartnern vom Eintreten eines Systemereignisses, etwa der Änderung des Datenbestandes durch einen Systemteilnehmer, zu informieren. Diese Art der Verteilung von (Änderungs-)Ereignissen (-nachrichten) wird auch als data change propagation bezeichnet.

Der Entwurf eines kooperativen Entwicklungswerkzeugs setzt eine stabile Kommunikations-Infrastruktur voraus. Dazu gehört mindestens die Möglichkeit, zwischen zwei Prozessen Informationen auszutauschen. Die verfügbaren Dienstprimitive sollen eine zuverlässige Informationsübermittlung bieten und den Systementwickler soweit wie möglich von der Fehlerbehandlung entlasten. Auch für den Austausch von Informationen zwischen mehr als zwei Kommunikationspartnern sollen Dienstprimitive zur Verfügung stehen.

Wenn das Anwendungssystem die Verwendung von Arbeitssitzungen unterstützen soll, ist für die Registrierung zugelassener Teilnehmer, den Beginn und das Beenden der Sitzung, die Lokalisierung der laufenden Sitzungen sowie das An- und Abmelden bei einer Arbeitssitzung die Definition entsprechender Entwurfskonstrukte (z. B. Funktionen, Klassen, Module) wünschenswert. Die Übermittlung einer Nachricht erfordert Dienstprimitive für das Senden und Empfangen der Nachricht. Die Primitive *send* und *receive* werden von Netzwerkbetriebssystemen, verteilten Betriebssystemen im engeren Sinne oder Betriebssystemerweiterungen (z. B. Middleware) zur Verfügung gestellt. In kooperativen Systemen tritt häufig der Fall auf, dass viele oder alle Anwender von einer Änderung des Datenbestandes verständigt werden müssen. Daher ist die Möglichkeit eines Broadcasting oder Multicast-Nachrichtenaustausches wünschenswert.

Die Kommunikation zwischen Prozessen wird häufig nach dem Client/Server-Modell gestaltet. Client ist ein Prozess, der bei einem Serverprozess eine bestimmte Dienstleistung nachfragt bzw. eine Anfrage an einen Server richtet. Der Vorteil des Client/Server-Modells besteht in der Verbindung von Einfachheit und Flexibilität. Der Client sendet eine Anfragenachricht, die den gewünschten Dienst beschreibt, an einen

Server, der diesen Dienst erfüllt, indem er die nachgefragten Daten oder eine Fehlermeldung zurückliefert.

Es wurde bereits darauf hingewiesen, dass die Kommunikation in objektorientierten Softwaresystemen wahlweise als Methodenaufruf oder Nachrichtenversand interpretiert werden kann (vgl. Abschnitt 3.5.2). Die Betrachtung eines objektorientierten Softwaresystems als Ansammlung von Objekten, die untereinander Nachrichten austauschen, erlaubt eine nahtlose Abbildung des Client/Server-Konzeptes auf objektorientierte Systeme. Objekte fungieren als Server, wenn sie Nachrichten ihrer Clients entgegennehmen und beantworten; der Client ruft eine (entfernte) Methode des Servers auf. Die Überwindung der Grenzen von Prozessen, Rechnern, Betriebssystemen oder Netzen leistet die sog. objektorientierte Middleware, die diese Verteilungsaspekte für den Systementwickler transparent macht.

Bei der Gestaltung kooperativer Werkzeuge ist darüber hinaus darauf zu achten, Mechanismen für Identifizierung, Adressierung angeschlossener Rechner, Objekte oder Dienste und das Routing von Informationen durch das Netzwerk bereitzustellen. Insbesondere für das Verteilen von Nachrichten zur Unterstützung der Kommunikation ist eine eindeutige Adressierung wichtiges Merkmal für die angeschlossenen Clients der Anwender. Im Rahmen dieser Adressierung und Identifikation spielen auch Sicherheits- und Zugangsaspekte eine große Rolle, um die verwendeten Systeme vor unsachgemäßem Gebrauch zu schützen. Hierfür existieren leistungsfähige Sicherheitsmechanismen wie abgestufte Zugriffskontrollen, Verschlüsselungskonzepte und Rollenprivilegien, wie sie oben dargestellt wurden (vgl. 3.5.4.3).

**Benutzungsschnittstelle**

In der Literatur (vgl. etwa [Rose93]) wird verschiedentlich gefordert, die Programmierung der Benutzungsschnittstelle durch speziell für die Erstellung von Mehrbenutzeranwendungen angepasste Oberflächenobjekte zu unterstützen (shared visual objects). Dieser Auffassung steht jedoch entgegen, dass im allgemeinen eine Trennung zwischen der internen Datenhaltung, der Zugriffsschnittstelle auf diese Daten und ihrer visuellen Repräsentation angestrebt wird, wie dies etwa im bekannten Model-View-Controller-Konzept geschieht (vgl. Abschnitt 3.5.5.5). Der Oberfläche kommt dann lediglich die Aufgabe zu, dem Anwender Informationen darzubieten und Benutzeraktionen entgegenzunehmen, nicht jedoch, etwa Benutzeraktionen zu propagieren. Dies ist Aufgabe einer tiefer liegenden Dienstebene. Auch das WYSIWYG-Konzept (What-you-see-is-what-you-get) verlangt nicht, Informationen mit grundsätzlich anderen Mitteln darzustellen, als dies in Einbenutzeranwendungen geschieht; die vorhandenen Mittel (Oberflächenelemente, Mauszeiger, Textcursor) werden lediglich anders genutzt. Aus diesem Grund beschränkt sich die Notwendigkeit, die Oberflächengestaltung an den Mehrbenutzerbetrieb anzupassen, auf die Visualisierung von Informationen, die in Einbenutzeranwendungen nicht vorhanden sind, etwa die Struktur des Anwenderkreises. Dafür reichen im Allgemeinen die üblichen Oberflächenelemente (Schaltflächen, Listen, Zeichenflächen, grafische Symbole usw.) aus. In dieser Arbeit wird daher nicht davon ausgegangen, dass speziell an den Mehrbenutzerbetrieb angepasste Oberflächenobjekte für die Erstellung eines kooperativen Mehrbenutzerwerkzeugs benötigt werden, wohl aber, dass während der Gestaltung der Oberflächen das Augenmerk auf zusätzliche Objekte innerhalb der

Oberfläche gelegt wird, die die Kommunikation, Kooperation und Koordination der an einem Simulationsmodell arbeitenden Anwender abbildet.

### 3.5.5 Organisationsformen des Software-Designs von Mehrbenutzersystemen

Software an sich ist zunächst eine abstraktes Gebilde und muss für den Umgang des Menschen mit dieser Software handhabbar gemacht werden. Dabei sind Entscheidungen über die Gestaltung des zu schaffenden Systems zu treffen: Was sind die Elemente dieses Systems, in welchen Beziehungen stehen diese zueinander, usw. Die Gesamtheit dieser Regelungen zur Gestaltung der Ordnung des Softwaresystems soll hier als *Organisationsform* bezeichnet werden. In Anlehnung an [Woeh90] bezeichnet Organisation hier sowohl den Prozess der Entwicklung einer Ordnung, als auch das Ergebnis dieses gestalterischen Prozesses, d.h. die Gesamtheit aller getroffenen Regelungen. Beispiele für Software-Organisationsformen sind Funktionsbibliotheken, Klassenbibliotheken, Frameworks, Software-Entwurfsmuster sowie spezielle Lösungen (Toolkits und konfigurierbare Anwendungen). Sie sollen in diesem Abschnitt vorgestellt und hinsichtlich ihrer Eignung zur Realisierung des angestrebten Werkzeugs, bzw. hinsichtlich der Verwirklichung der allgemeinen Ziele und Prinzipien des Software Designs (vgl. 3.5.1 und 3.5.2), bewertet werden.

Die Wahl einer geeigneten Organisationsform für Software ist abhängig von unterschiedlichen Einflussfaktoren. Die Gewichtung dieser Faktoren wiederum ist abhängig von der jeweiligen konkreten Aufgabenstellung. Es ist deshalb nicht möglich, lediglich anhand eines einzelnen softwaretechnischen Zieles allgemeingültige Aussagen über die Eignung einer bestimmten Organisationsform zu treffen. Vielmehr ist jeweils das gesamte Zielsystem der Softwaretechnik unter den gegebenen individuellen Randbedingungen zu berücksichtigen. Es gibt jedoch zwei Gründe, warum dem Ziel Wiederverwendung eine besondere Rolle bei der Beurteilung einer Software-Organisationsform zukommt:

- *Bedeutung des Zieles Wiederverwendung an sich*: Bereits bei der Diskussion des softwaretechnischen Zielsystems wurde diese herausgestellt und begründet. Erfolgreiche Wiederverwendung kann die Zuverlässigkeit und Wartungsfreundlichkeit eines Softwaresystems steigern, das Risiko von Fehlentwicklungen verringern, das Wissen von Experten und ganzen Organisationen konservieren und transferieren und den finanziellen und zeitlichen Aufwand eines Projektes reduzieren [Somm92].

- *Einfluss der Organisationsform auf die Wahrscheinlichkeit der Wiederverwendung*:. Software kann nur dann wieder verwendet werden, wenn eine Organisationsform gewählt wurde, die die Identifikation, den Zugriff und das Extrahieren der wieder zu verwendenden Komponenten gestattet. Die unbefriedigende Situation, dass viele Systementwickler das Gefühl haben, ständig „das Rad neu zu erfinden", kann nur beseitigt werden, wenn vorhandene Problemlösungen bekannt sind und zur Verfügung stehen und wenn außerdem eine ggf. notwendige Adaption an eine geänderte Problemstellung möglich ist.

Wiederverwendung von Software erfährt aus technologischen, ökonomischen, psychologischen und organisatorische Gründen steigende Bedeutung. Der Gedanke liegt wegen des stark wiederholenden Charakters vieler Phasen der Systementwicklung nahe. Wiederverwendung beinhaltet die systematische Nutzung bereits existierender Modelle, Entwürfe bzw. Entwurfsfragmente, Programmtexte und Dokumentationen. Es geht also nicht lediglich um die Wiederverwendung von Programmcode, sondern sämtlicher (Zwischen-) Ergebnisse des Entwicklungsprozesses. In jedem Fall kann aber nur erneut verwendet werden, was in Form eines irgendwie gearteten Dokumentes zur Verfügung steht. In Frage kommen Quellcode, Objektcode, Personal, Dokumentationen. Diese Dokumente können sich auf Softwarekomponenten wie komplette Systeme, auf Subsysteme, Module oder Objekte oder einzelne Funktionen beziehen. Die besondere Eignung objektorientierter Technologien zur Unterstützung der Wiederverwendung wird als ein Standardargument zugunsten der Objektorientierung gebracht (vgl. Abschnitt 3.5.2.2). Einige Voraussetzungen erfolgreicher Wiederverwendung sind nach [Booc94]:

- Es muss möglich sein, die passenden Komponenten wieder zu finden.
- Die Komponenten müssen vom Systementwickler verstanden und als geeignet akzeptiert werden.
- Zu jeder Komponente müssen detaillierte Informationen darüber vorliegen, wie sie wieder verwendet werden kann.

Wiederverwendung wird dagegen erschwert oder verhindert, wenn der Systementwickler die Leistungsfähigkeit der angebotenen Komponenten nicht versteht oder ihre Notwendigkeit nicht akzeptiert. Darüber hinaus müssen die Komponenten selbst auch für die Wiederverwendung geeignet sein, d.h., sie müssen erweiterbar, bzw. anpassbar sein und sie dürfen nicht einem zu sehr eingeschränkten Anwendungsbereich dienen und damit die erforderliche Flexibilität vermissen lassen. Die Entwicklung wieder zu verwendender Komponenten erfordert daher ein besonders hohes Maß an Erfahrung und gestaltet sich schwieriger und aufwendiger als die Entwicklung von Software zur einmaligen Verwendung. Die Berücksichtigung der in Kapitel 3.5.1 diskutierten Prinzipien spielt daher hierbei eine besonders große Rolle.

#### 3.5.5.1 Funktionsbibliotheken

Funktionsbibliotheken sind eine Organisationsform wieder verwendbarer Software, die dem Paradigma der prozeduralen Programmierung entstammt. Programmiersprachen organisieren Programme in Funktionen, Unterprogramme oder Prozeduren, die sich gegenseitig aufrufen und ggf. Modulen angehören können. Es entsteht ein hierarchischer Aufrufbaum, wie in Abbildung 21 gezeigt.

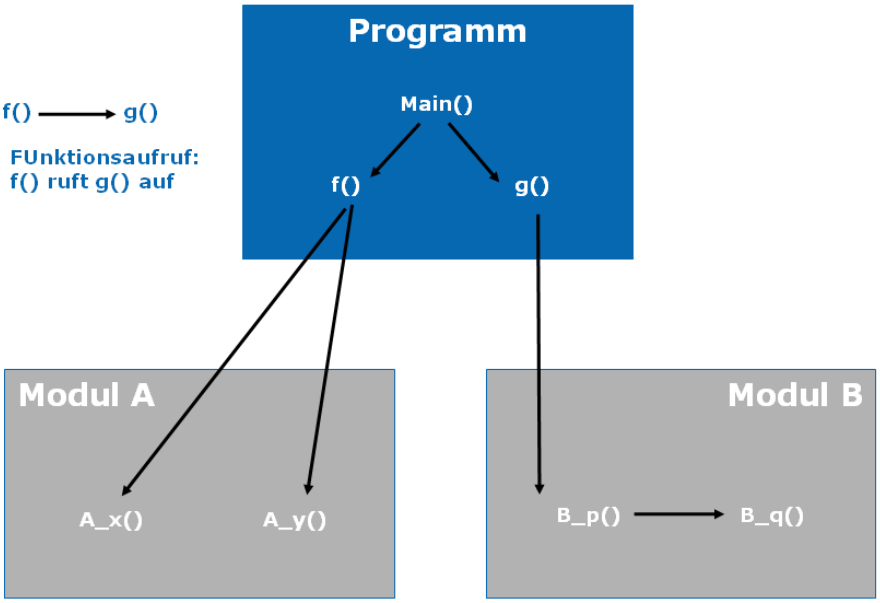


**Abbildung 21: Topologie prozeduraler Programmabläufe [Henk97]**

Funktionsbibliotheken sind Sammlungen von Funktionen, die zumeist einen gemeinsamen Anwendungsbereich (z.B. mathematische Funktionen, Stringverarbeitung, Datenbankzugriff, grafische Ein-/Ausgabe,...) behandeln. Sie können als wieder verwendbare Module aufgefasst werden. In Abbildung 21 könnte anstelle eines der Module also auch die Bezeichnung einer Funktionsbibliothek treten. Es handelt sich daher um eine Form der Wiederverwendung von Code, der entweder als Quelltext oder als statisch oder dynamisch zu bindender Objektcode vorliegt. Dem prozeduralen Paradigma entsprechend, sind Datenstrukturen nicht Gegenstand der Wiederverwendung. Funktionsbibliotheken stellen daher einen einfachen, softwaretechnisch jedoch noch nicht voll befriedigenden Mechanismus zur Wiederverwendung dar, der Modularisierung und Hierarchisierung ermöglicht, dessen Mangel an Abstraktion jedoch die Wahrscheinlichkeit einer Wiederverwendung mindert.

Für den Entwurf und die Realisierung verteilter Systeme kommt der Einsatz von Funktionsbibliotheken vor allem in den betriebssystemnahen Systemebenen in Frage. Ein Beispiel im Bereich der Kommunikation zwischen Prozessen sind Sockets. Sie basieren auf dem TCP/IP-Protokoll[17] und bilden einen Puffer, über den Nachrichten gesendet und empfangen werden können. Die Adresse eines Socket beinhaltet die Adresse des Rechners im Netzwerk sowie eine für diesen Rechner eindeutige SocketID. Funktionsbibliotheken setzen oftmals Detailkenntnisse über die jeweiligen Funktionen voraus und erfordern somit einen erhöhten Einarbeitungsaufwand. Der Ansatz bietet daher keine reibungslose Wiederverwendung von Software

[17] Das TCP/IP-Protokoll ist ein Netzwerkprotokoll, das die Basis für die Kommunikation im Internet bildet. Das Transmission Control Protocol (TCP) ist eine Vereinbarung darüber, auf welche Art und Weise Daten zwischen Computern ausgetauscht werden sollen [Balz05].

Das Internet Protokoll (IP) bildet die erste vom Übertragungsmedium unabhängige Schicht der Internet-Protokoll-Familie. Das bedeutet, dass mittels IP-Adresse und Subnetzmaske (subnet mask) Computer innerhalb eines Netzwerkes in logische Einheiten gruppiert werden können. Auf dieser Basis ist es möglich, Computer in größeren Netzwerken zu adressieren und Verbindungen zu ihnen aufzubauen [Balz05].

### 3.5.5.2 Klassenbibliotheken

> *"Eine Klassenbibliothek besteht aus einer Menge von verwandten und wieder verwendbaren Klassen, die entworfen wurden, um nützliche und allgemeine Funktionalitäten zur Verfügung zu stellen."*[GaHe01]

Die Funktionalitäten der Klassenbibliothek sind in der Regel unabhängig vom Kontext der Anwendung. Die Wiederverwendung von Klassen innerhalb der Bibliothek kann durch zwei Arten erfolgen: Entweder werden Klassen der Klassenbibliothek erzeugt (instanziiert) oder neue Klassen werden durch gegebene Klassen der Bibliothek abgeleitet (Prinzip der Vererbung, Abschnitt 3.5.2.2).

Klassenbibliotheken bilden das objektorientierte Gegenstück zu Funktionsbibliotheken. Typische Beispiele für eine solche Ansammlung wieder verwendbarer Klassen sind Pakete zur Verwaltung von Datenstrukturen, zur Realisierung von Oberflächenelementen oder zum Aufbau von Kommunikationsschnittstellen mit Datenbanksystemen. Das Zusammenspiel der Klassen ist vom Designer der Bibliothek nur in weiten Grenzen vorgegeben. Es wird im Wesentlichen vom Entwickler der Anwendung bei der Implementierung eines konkreten Programms festgelegt. Abbildung 22 zeigt die aus der Verwendung einer Klassenbibliothek resultierende Topologie eines Programms.

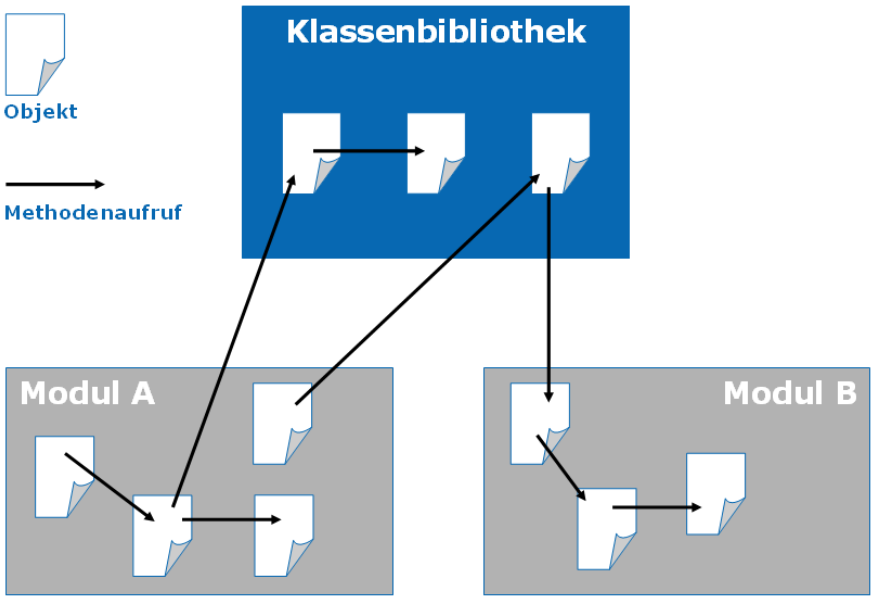


**Abbildung 22: Programmtopologie auf Basis einer Klassenbibliothek [nach Booc94]**

Mögliche Zugriffe auf Klassenbibliotheken sind durch das Application Programming Interface (API) definiert. Dabei handelt es sich um eine Gesamtheit der öffentlich verfügbaren Klassen und Schnittstellen in Abgrenzung zu den privaten Einheiten der Bibliothek, die nicht zugänglich sind. Klassenbibliotheken werden deshalb auch häufig als API bezeichnet. Durch die Einhaltung der Paradigmen des objektorientierten Entwurfs (vgl. 3.5.2) ergeben sich für Klassenbibliotheken einige Vorteile. Durch ihren Einsatz kann ein höherer Abstraktionsgrad erreicht werden und Methodenaufrufe können typsicher formuliert werden. Das reduziert die Fehlerwahrscheinlichkeit. Durch den Zugriff über die API, bzw. die Schnittstellen wird eine saubere Trennung des Zugriffs und

der eigentlichen Implementierung erreicht. Klassenbibliotheken lassen sich damit leichter portieren.

Insgesamt ergibt sich aus der Beachtung allgemeiner und objektorientierter Paradigmen des Software-Entwurfs eine höhere Wahrscheinlichkeit, dass ein Anwendungsentwickler in einer Klassenbibliothek Softwarekomponenten findet, die ihm für eine effiziente Wiederverwendung geeignet erscheinen und zur Verfügung stehen. Eine spezielle Form von Klassenbibliotheken bilden so genannte Frameworks, oder Rahmenwerke, die im folgenden Abschnitt näher betrachtet werden sollen.

#### 3.5.5.3 Frameworks

> *"Ein Framework besteht aus einer Menge von zusammenarbeitenden Klassen, die einen wieder verwendbaren Entwurf für eine bestimmte Klasse von Applikationen darstellen."*[GaHe01]

Im Gegensatz zu Klassenbibliotheken, die eine Ansammlung nur lose verbundener Klassen darstellen und dem Systementwickler wenig Vorgaben über die Verwendung der Klassen machen, enthalten Frameworks Klassen, zwischen denen eine genau definierte Beziehung besteht. Ein Framework gibt damit in der Regel eine Architektur der jeweiligen Anwendung vor, wobei eine Umkehrung der Kontrolle stattfindet: der Programmierer registriert konkrete Implementierungen, die dann durch das Framework gesteuert und benutzt werden, statt — wie bei einer Klassenbibliothek — lediglich Klassen und Funktionen zu benutzen. Ein Framework definiert insbesondere den Kontrollfluss der Anwendung und die Schnittstellen für die konkreten Klassen, die vom Programmierer erstellt und registriert werden müssen. Frameworks werden also im Allgemeinen mit dem Ziel einer Wiederverwendung architektonischer Muster entwickelt und genutzt. Da solche Muster nicht ohne die Berücksichtigung einer konkreten Domäne entworfen werden können, sind Frameworks meist domänenspezifisch oder doch auf einen bestimmten Anwendungstyp beschränkt. Frameworks sind nach dieser Definition immer objektorientiert, obwohl sie grundsätzlich auch mit anderen Technologien erstellt werden können.

Die wesentlichen Vorteile von Frameworks gegenüber den Bibliotheken liegen in einem reduzierten Wartungsaufwand, einer verbesserten Wiederverwendung allgemeiner Funktionalitäten und insbesondere in der Möglichkeit einer zuverlässigen Erweiterung und Spezialisierung innerhalb einer vorgegebenen Infrastruktur. Je nach der Art und Weise, wie Frameworks vom Entwickler verwendet werden müssen, unterscheidet man zwischen White-Box- und Black-Box-Frameworks. White-Box-Frameworks bestehen aus einer Reihe von abstrakten Klassen. Die Aufgabe des Entwicklers besteht dann darin, jene abstrakten Methoden dieser Klassen in den abgeleiteten Unterklassen der konkreten Applikation zu überschreiben. Dafür ist eine gute Kenntnis der Framework-Klassen erforderlich. Das Gegenstück bilden die Black-Box-Frameworks. Diese bieten bereits fertige Klassen an, die der Entwickler instanziieren und so zu einer Applikation zusammensetzen kann. Verfügbare Frameworks sind zumeist Mischformen zwischen White-Box- und Black-Box-Frameworks. Mit steigendem Reifegrad tendieren Frameworks zum Black-Box-Ansatz.

Frameworks stellen also einen Großteil des notwendigen Codes bereit und halten Lösungen für wichtige Designentscheidungen bereit, die sich in einem Problembereich typischerweise stellen, während Funktions- und Klassenbibliotheken den Systementwickler vor allem in der Implementierungsphase von wiederkehrenden Aufgabe entlasten können, die Verantwortung für wesentliche Entwurfsentscheidungen aber beim Systemdesigner belassen. Frameworks bieten daher eine Wiederverwendung von Designwissen, das in den implementierten Klassen enthalten ist. Diese Wiederverwendung auf Entwurfsebene verlangt vom Entwickler des Frameworks besonders viel Wissen und Erfahrung, wird aber dem Abstraktionsprinzip in besonderem Maße gerecht, weil quasi eine Referenz-Architektur zur Verfügung gestellt wird. Als Dokument der Wiederverwendung kommt wiederum die Verwendung von Quellcode oder Objektcode in Frage. Die Basis der Wiederverwendung ist allerdings nicht die Klasse, sondern das Klassenteam.

Die Topologie eines mit Hilfe eines Frameworks definierten Programms unterscheidet sich nicht von der in Abbildung 22 gezeigten Darstellung. Der Unterschied besteht vielmehr darin, dass die im Bild als Kanten des Beziehungsgraphen dargestellten Objektinteraktionen bereits im Framework weitgehend festgelegt sind, während diese Aufgabe bei Verwendung einer konventionellen Klassenbibliothek dem Systementwickler zufällt.

Wie unter Abschnitt 3.2.2 aufgezeigt, existieren im Bereich der Ablaufsimulation verschiedene Frameworks, die wegen der speziellen Anforderungen an das zu entwickelnde Werkzeug nicht in dieser Arbeit verwendet werden können.

#### 3.5.5.4 Toolkits und konfigurierbare Anwendungen

Eine Möglichkeit der gezielten Wiederverwendung von Komponenten (typischerweise als Objektcode), die gegenüber dem allgemeingültigeren Ansatz der Wiederverwendung von Basisdienst-Komponenten einen reduzierten Overhead und außerdem einen geringeren Entwicklungsaufwand verwirklichen kann, ist der Rückgriff auf Gesamtlösungen. Hierbei wird der Versuch unternommen, die angebotene Funktionalität gezielt auf einen bestimmten Problem- bzw. Anwendungsbereich abzustimmen. Zwei wesentliche Kategorien von dedizierten Gesamtlösungen werden im Folgenden unterschieden und kurz dargestellt: Toolkits und konfigurierbare Anwendungen.

**Toolkits**

Toolkits gelten als Sammelbezeichnung für Zusammenstellungen von Hilfsmitteln wie Funktions- oder Klassenbibliotheken, Frameworks, fertig verwendbare Anwendungskomponenten, Dienstleistungsprogrammen (Server), Entwicklungshilfen (CASE[18]-Tools, Programmierumgebungen, Compiler[19], Programmgeneratoren,...), Interface Builder, Datenhaltungskomponenten usw. Diese Aufzählung macht deutlich, dass die Abgrenzung gegen andere Software-Organisationsformen schwierig sein kann. Wesentliche Eigenschaften von Toolkits sind allerdings die Adressierung eines bestimmten Aufgabenbereiches sowie die Spezifikation eines Anwendungsmodells. Durch

[18] CASE: Computer Aided Software Engineering

[19] Ein Compiler analysiert ein Programm auf fehlerfreie Syntax und übersetzt es vollständig in ein Zielsystem, das ohne erneute Analyse beliebig oft hintereinander ausgeführt werden kann [Balz05].

die Adressierung bestimmt sich der Umfang und Inhalt der angebotenen Funktionalität. Der Anwender hat zumeist nicht die Möglichkeit zu beliebigen Erweiterungen. Der Anwender muss darüber hinaus das Modell der Anwendungsentwicklung als gegeben hinnehmen. Dieses Modell beinhaltet Annahmen über den Kenntnisstand der Anwender des Toolkits, das zugrunde liegende softwaretechnische Paradigma, die verwendete Programmiersprache, die Hardware- und Softwareumgebung der Anwendungs-entwicklung und -verwendung usw.

**Konfigurierbare Anwendungen**
Fertige Anwendungen stellen den Extremfall dedizierter Gesamtlösungen dar und setzen die Grundidee am konsequentesten um. Die Systementwicklung wird auf die Wahrnehmung der vorhanden Einstellmöglichkeiten reduziert, d. h. die Konfiguration bzw. Parametrisierung z.B. durch Aufrufoptionen, statische oder dynamische Ressourceneinstellungen, Editieren von Datendateien, -objekten oder -tabellen sowie einfache Eingriffe in den Kontrollfluss durch Skriptdateien.

Für den Anwendungsbereich der synchronen verteilten Interaktion sind konfigurierbare Anwendungen, die speziell für die Realisierung eines kooperativen Werkzeuges zur Modellierung und Simulation von Fertigungssystemen, wie sie in dieser Arbeit angestrebt wird, zurzeit nicht bekannt.

Das größte Problem im Zusammenhang mit dem Einsatz von dedizierten Gesamtlösungen resultiert gerade aus dem wichtigsten Vorteil: Die gezielte Einschränkung der angebotenen Funktonalität erfordert es, eine Lösung zu finden, die genau auf die gegebene Problemstellung passt oder eine ausreichende Flexibilität bietet, um eine Anpassung zu ermöglichen. In diesem Fall ermöglichen Speziallösungen einen hohen Anteil wieder verwendeter Software am Gesamtumfang eines Projekts. Praktische Erfahrungen zeigen aber, dass gerade die Suche nach einer ideal passenden Lösung häufig Schwierigkeiten bereitet und insbesondere spätere Änderungen der Anforderungsdefinition zu Problemen führen. Der Grund dafür liegt in der mangelnden Berücksichtigung (bzw. dem bewussten Verzicht) des softwaretechnisches Prinzips der Abstraktion. Wichtige softwaretechnische Ziele, wie Flexibilität, Interoperabilität, Portabilität, Erweiterbarkeit und Wartbarkeit werden deshalb von dedizierten Gesamtlösungen nicht immer voll erfüllt. Für den Bereich der Ablaufsimulation existieren zahlreiche Anwendungen, die jedoch kaum auf die Anforderungen der Anwender konfiguriert werden können. Für die unter Abschnitt 2.3 aufgezeigten Anforderungen existiert kein Toolkit.

#### 3.5.5.5 Software-Entwurfsmuster

> *"Jedes Muster beschreibt ein in unserer Umwelt beständig wiederkehrendes Problem und erläutert den Kern der Lösung für dieses Problem, so dass sie diese Lösung beliebig oft anwenden können, ohne sie jemals ein zweites Mal gleich auszuführen."* [AlIs77]

Ein Entwurfsmuster (engl. design pattern) beschreibt eine in der Praxis erfolgreiche, generische Lösung für ein mehr oder weniger häufig auftretendes, wiederkehrendes Entwurfsproblem und stellt damit eine wieder verwendbare Vorlage zur Problemlösung dar. Entstanden ist der Ausdruck in der Architektur, von wo er für die

Softwareentwicklung übernommen wurde [GaHe01]. Der primäre Nutzen eines Entwurfsmusters liegt in der Beschreibung einer Lösung für eine bestimmte Klasse von Entwurfsproblemen. Weiterer Nutzen ergibt sich aus der Tatsache, dass jedes Muster einen Namen hat. Dies vereinfacht die Diskussion unter Softwareentwicklern, da man abstrakt über eine Softwarestruktur sprechen kann. So sind Entwurfsmuster zunächst einmal unabhängig von der konkreten Programmiersprache. Wenn der Einsatz von Entwurfsmustern dokumentiert wird, ergibt sich ein weiterer Nutzen dadurch, dass durch die Beschreibung des Musters ein Bezug zur dort vorhandenen Diskussion des Problemkontextes und der Vor- und Nachteile der Lösung hergestellt wird.

Die Beschreibung eines Entwurfsmusters folgt nach [GaHe01] dem folgenden Schema:

- *Name* und *Klassifikation* des Musters.
- *Synonyme*: Andere bekannte Namen des Musters.
- *Beispiel* eines Musters
- *Kontext*: Einsatzbereiche für das Muster.
- *Problembeschreibung*
- *Lösungsprinzip* des Musters
- *Struktur*: Beschreibung der allgemeinen Struktur des Musters.
- *Dynamische Aspekte*: Typische Szenarien zur Beschreibung des Laufzeitverhaltens
- *Implementierung*: Praxisrelevante Tipps, Tricks und Techniken sowie Warnung vor Fehlern, die leicht passieren können.
- *Musterlösung*
- *Varianten*
- *Praxiseinsatz*: Wo wird das Muster bereits eingesetzt?
- *Auswirkungen*: Vor- und Nachteile der Anwendung des Musters
- *Querverweise*: Wie spielt das Muster mit anderen Mustern zusammen?

Generell sollte die Dokumentation eines Entwurfsmusters ausreichende Informationen über das Problem, welches das Muster behandelt, über den Kontext der Anwendung und über die vorgeschlagene Lösung bereitstellen. [GaHe01] klassifiziert Muster nach den beiden Kriterien des Zwecks (*purpose*) und des Bereichs, auf dem sie wirken (*scope*). Nach dem Zweck des jeweiligen Musters unterscheidet sie drei Klassen: Die erste Gruppe der Erzeugungsmuster bezieht sich auf die Erzeugung von Objekten. So können etwa die Anzahl von erzeugten Objekten einer Klasse kontrolliert, oder man der konkrete Typ der erzeugten Objekte - abhängig von den jeweiligen Bedingungen – angepasst werden. Die zweite Gruppe umfasst Strukturmuster, welche eine Vereinfachung der Struktur zwischen Klassen ermöglichen sollen. Komplexe Beziehungsgeflechte können beispielsweise über vermittelnde Klassen oder Schnittstellen logisch vereinfacht werden. Die dritte Gruppe der Verhaltensmuster betrifft das Verhalten der Klassen. Hierbei handelt es sich um die größte Gruppe von Mustern. Sie beziehen sich auf die Zusammenarbeit und den Nachrichtenaustausch von Klassen. Tabelle 2 zeigt eine Übersicht zur Kategorisierung der in [GaHe01] angegebenen Entwurfsmuster hinsichtlich ihres Zwecks und ihrer Wirksamkeit.

Nach ihrem Wirkungsbereich lassen sich Muster in Klassen- und Objektmuster einteilen. Klassenmuster beschreiben Beziehungen zwischen Klassen und bauen vorrangig Vererbungsstrukturen auf. Die Strukturen sind damit zur Übersetzungszeit festgelegt.

Hingegen nutzen Objektmuster vorrangig Assoziationen und Aggregationen zur Beschreibung von Beziehungen zwischen Objekten. Die durch sie beschriebenen Strukturen zwischen Objekten sind zur Laufzeit dynamisch änderbar.

| | | Aufgabe | | |
|---|---|---|---|---|
| | | **Erzeugungsmuster** | **Strukturmuster** | **Verhaltensmuster** |
| **Gültigkeits-bereich** | **Klassen-basiert** | Fabrikmethode | Adapter | Interpreter<br>Schablonenmethode |
| | **Objekt-basiert** | Abstrakte Fabrik<br>Erbauer<br>Prototyp<br>Singleton | Adapter<br>Brücke<br>Dekorierer<br>Fassade<br>Fliegengewicht<br>Kompositum<br>Proxy | Befehl<br>Beobachter<br>Besucher<br>Iterator<br>Menento<br>Strategie<br>Vermittler<br>Zustand<br>Zuständigkeitskette |

**Tabelle 2: Dimensionen der Ausprägung von Entwurfsmustern nach [GaHe01]**

Nicht jedes Muster lässt sich heutzutage ohne weiteres als Entwurfsmuster klassifizieren. Vielmehr gibt es, unter anderem, graduelle Unterschiede in der Körnigkeit von Mustern. So wird etwa das Model-View-Controller-Muster sowohl als Architekturmuster, als auch als Entwurfsmuster betrachtet. Software Entwurfsmuster stellen also einen Versuch dar, ein professionelles Medium für den Transfer des Wissens von Software-Entwicklern über die Erstellung, Pflege und Dokumentation komplexer Softwaresysteme bereitzustellen. Sie helfen, geeignete Lösungen für Entwurfsprobleme zu identifizieren, indem sie vorhandenes Entwurfswissen dokumentieren. Innerhalb der Konzeptionsphase der hier vorliegenden Arbeit sollen Entwurfmuster insbesondere in den ersten Phasen die Designentscheidungen des Werkzeugs unterstützen und helfen, das Gesamtprojekt gliedern zu können.

### 3.5.6 Architekturmuster von Mehrbenutzersystemen

Realisierbare Software-Architekturen sind auf ein paar grundsätzlichen Strukturierungs-prinzipien aufgebaut. Diese Prinzipien werden als Architekturmuster beschrieben.

> *"Architekturmuster beschreiben fundamentale, strukturelle Organisationsschemata für Softwaresysteme. Sie bieten eine Anordnung von Subsystemen und deren wechselseitige Beziehungen und beinhalten Regeln und Richtlinien zur Organisation der Beziehungen." (nach* [BuMe96])

Ein Architekturmuster spiegelt also ein grundsätzliches Strukturierungsprinzip von Software-Systemen wieder. Es beschreibt eine Menge vordefinierter Subsysteme, spezifiziert deren jeweiligen Zuständigkeitsbereich und enthält Regeln zur Organisation der Beziehungen zwischen den Subsystemen. Sie können als Schablonen für konkrete Software-Architekturen verstanden werden. Aufgrund der impliziten Auswahl der strukturellen Eigenschaft eines Anwendungssystems mit der Wahl eines Architekturmusters, wird mit der Wahl eines speziellen Architekturmusters eine Grundsatzentscheidung im Entwurf eines Softwaresystems getroffen. Architekturmuster

lassen sich nach [GaHe01] in vier verschiedene Kategorien einteilen, die in Tabelle 3 kurz dargestellt werden.

Die Architekturmuster sollen im Folgenden nicht detailliert vorgestellt und diskutiert werden. Erläuterungen zu Aufgaben und Wirkungsweisen finden sich unter [Design Patterns]. Die aufgezeigten Muster sollen nachfolgend vielmehr hinsichtlich ihrer Eignung für die Verwendung als Software-Architektur für das zu entwickelnde, mehrbenutzerfähige Werkzeug zur Ablaufsimulation untersucht werden. Die Betrachtung ausgewählter Architekturmuster beschränkt sich auf die ersten drei Kategorien von Architekturmustern. Muster für adaptierbare Systeme lassen sich nur schwerlich hinsichtlich der Aufgabenstellung verwenden und werden nachfolgend nicht genauer betrachtet.

| Kategorie | Beschreibung | Zugeordnete Muster |
|---|---|---|
| Mud-To-Structure | Diese Architekturmuster sollen helfen, die Unmengen von Komponenten und Objekten eines Softwaresystems zu organisieren. Die Funktionalität des Gesamtsystems wird hierbei in kooperierende Subsysteme aufgeteilt. | Layers<br>Pipes-And-Filters<br>Blackboard |
| Verteilte Systeme | Diese Kategorie unterstützt die Verwendung verteilter Ressourcen und Dienste in Netzwerken. | Broker<br>Client-Server<br>Microkernel<br>Pipes-And-Filters |
| Interaktive Systeme | Pattern dieser Kategorie helfen Mensch-Computer-Interaktionen zu strukturieren. | Model-View-Controller<br>Presentation-Abstraction-Control |
| Adaptierbare Systeme | Architekturmuster dieser Kategorie unterstützen besonders die Erweiterungs- und Anpassungsfähigkeit von Softwaresystemen. | Microkernel<br>Reflection |

**Tabelle 3: Kategorien von Architekturmustern**

**Bewertung**

Die Anwendung des Architekturmusters *Layer* bietet einige Vorteile hinsichtlich einer ersten Strukturierung einzelner Programmteile. Für die Umsetzung des angestrebten Werkzeugs bietet es durch seine Ausrichtung aber allenfalls dazu an, innerhalb bestimmter Programmmodule Basisdienste zu strukturieren und zu definieren. Insbesondere einer Ausrichtung hinsichtlich der geforderten Interaktivität und Mehrbenutzerfähigkeit kann mit diesem Muster nicht entsprochen werden.

Das *Pipes-and-Filters*-Muster bietet sich für die flexible Gestaltung schwieriger Datentransformationen und -Verarbeitungen im besonderen Maße an. Inwiefern solche komplexen Transformationen bei der Umsetzung des zu entwickelnden Werkzeuges berücksichtigt werden müssen, kann erst im Konzeptionskapitel entschieden werden. Für die Gesamtanordnung des Systems scheint das *Layers*-Muster als die bessere Entwurfsalternative, da hier insbesondere die Fehlerbehandlung als wichtiger Bereich der Software-Entwicklung besser unterstützt wird. Das grundsätzliche Vorgehen erscheint beim *Pipes-and-Filters*-Muster eher prozess- als objektorientiert. Für komplexe Datentransformationen kann es aber ggf. eingesetzt werden.

Das *Blackboard*-Muster eignet sich insbesondere für neue Anwendungsbereiche, wo eine zuverlässige Gesamtlösung nicht deterministisch generiert werden kann. Für die Entwicklung des angestrebten Modellierungs- und Simulationswerkzeuges gilt dieses nur bedingt, da die prinzipielle Struktur eines Simulators ein bekanntes Arbeitsgebiet ist, in dem „nur" eine auf die Fragestellung angepasste Architektur entwickelt werden muss. Der Entwurf des eigentlichen Werkzeuges trifft diese Definition somit nicht. Eine Anwendung im Bereich der Simulation, bzw. innerhalb der Logik des eigentlichen Simulationsmodells erscheint aber in den Bereichen möglich, wo die Terminierung eines Folgeereignisses als Folge einer Aktivität aus verschiedenen Teilbereichen zusammengesetzt werden muss. Ggf. kann das *Blackboard*-Muster also im Rahmen der Modellierung spezieller Fragestellungen von funktionsorientierten Fertigungssystemen eingesetzt werden.

Der Entwurf des Modellierungs- und Simulationswerkzeuges wird zwangsläufig auf ein modulares, verteiltes System fokussieren. In diesem Rahmen kann während der Entwurfsphase das *Broker*-Muster berücksichtigt werden, um ein abgesichertes Objektmodell zu entwerfen, das eine robuste Kommunikation zwischen den Modulen erlaubt. Die prinzipiell mögliche Ausrichtung des *Broker*-Musters auf eine Kombination von Systemen, die in verschiedenen Programmiersprachen entwickelt werden, wird wohl nicht verwendet werden müssen, da eine einheitliche Verwendung von Java eine Grundannahme des zu entwickelnden Werkzeuges werden soll. Dadurch wird das beschriebene *Broker*-Muster aber nur vereinfacht, da die zur Kapselung eingesetzten Stellvertreter-Objekte aus dem Grundmodell herausfallen können.

Die *Client-Server*-Architektur ist eine der momentan am weitesten verbreiteten Softwarearchitekturen für Business-Systeme. Insbesondere in den Funktionsmodulen zur Modellierung und/oder Visualisierung der Simulationsmodelle bietet sich dieses Software-Muster an, um den Multitasking-Betrieb an einem gemeinsamen Simulationsmodell zu ermöglichen. Auf dem zentralen Anwendungsserver soll in diesem Fall das aktuelle Simulationsmodell sowie die Verwaltung der angeschlossenen Anwender erfolgen. Für das angestrebte Werkzeug würde sich vermutlich eine weitergehende Lösung in Form einer 3-Tier-Architektur anbieten, so dass alle anfallenden Simulationsdaten in einer Simulationsdatenbank gespeichert werden können.

Die Anforderungen an das zu entwickelnde Werkzeug lassen sich auf einem hohen Abstraktionsgrad gut in verschiedene Subsysteme unterteilen, die in sich geschlossen funktionieren. Beispielsweise lässt sich der Gesamtprozess der Anwendung in Modellierung, Simulation und Analyse aufteilen. Für die Gestaltung auf einer detaillierten Ebene erscheint das *Model-View-Controller*-Muster besser geeignet, da hier die Aufteilung funktionsübergreifend erfolgt. Adaptiert man das *Model-View-Controller* Pattern auf die vorgestellte 3-Tier-Architektur, ergibt sich eine erste denkbare Softwarearchitektur für das zu entwickelnde Werkzeug. Vor allem hinsichtlich der Visualisierung der Simulationsläufe bietet sich Trennung nach eigentlicher Funktion und deren Darstellung an. Neben verschiedenen Darstellungsformen wird damit auch eine Simulation erlaubt, die ohne die Anzeigemethoden eines Visualisierungsmoduls zumindest berechnet werden kann.

Zusammenfassend kann man also sagen, dass verschiedene Architekturmuster zur Konzeption des Werkzeugs herangezogen werden können. Je nach Anforderung der Teilmodule sollen sie in Abschnitt 5.3 herangezogen werden.

### 3.5.7 Software-Schnittstellen

> *„Schnittstellen (interfaces) definieren Dienstleistungen für Anwender, d.h. aufrufende Klassen, ohne etwas über die Implementierung der Dienstleistung festzulegen.*" [Balz05]

Eine Schnittstelle wird durch eine Menge von Regeln beschrieben, der Schnittstellenbeschreibung. Neben der Beschreibung, welche Funktionen vorhanden sind und wie sie benutzt werden, gehört zu der Schnittstellenbeschreibung auch ein so genannter Kontrakt, der die Semantik der einzelnen Funktionen beschreibt. Standardisierte Schnittstellen bieten den Vorteil, dass Komponenten oder Module, die die gleiche Schnittstelle unterstützen, gegeneinander ausgetauscht werden können, das heißt sie sind zueinander kompatibel.

Softwareschnittstellen oder Datenschnittstellen sind dementsprechend logische Berührungspunkte in einem Softwaresystem: sie definieren, wie Kommandos und Daten zwischen verschiedenen Prozessen und Komponenten ausgetauscht werden. In der objektorientierten Programmierung (OOP) vereinbaren Schnittstellen gemeinsame Signaturen von Klassen. Das heißt, eine Schnittstelle vereinbart die Signatur einer Klasse, die diese Schnittstelle implementiert. Das Implementieren einer Schnittstelle stellt eine Art Vererbung dar. Man kann zwischen Schnittstellen zur Interprozesskommunikation (Kommunikation zwischen verschiedenen Programmen) und Schnittstellen für Programmkomponenten (dienen der Modularisierung der Software-Architektur) unterscheiden.

Die eXtensible Markup Language (XML) ist ein Standard zur Erstellung maschinen- und menschenlesbarer Dokumente in Form einer Baumstruktur, der vom World Wide Web Consortium (W3C) definiert wird [Rock00]. Die XML definiert Regeln für den Aufbau und die Struktur solcher Dokumente. Für einen konkreten Anwendungsfall ("XML-Anwendung") müssen die Details der jeweiligen Dokumente spezifiziert werden. Dies betrifft insbesondere die Festlegung der Strukturelemente und ihre Anordnung innerhalb des Dokumentenbaums. Eine Festlegung und Eingrenzung der Struktur von Simulationsmodellen wird somit ermöglicht und kann auf einfache Weise überprüft werden. Bei der Verwendung von XML spricht man hier auch von der Gültigkeit und Wohlgeformtheit von XML-Dateien hinsichtlich einer durch XML-Schemata oder Document Type Definition (DTD) festgelegten Grammatik.

Die Namen der einzelnen Strukturelemente (XML-Elemente) für eine XML-Anwendung lassen sich frei wählen. Ein XML-Element kann ganz unterschiedliche Daten enthalten und beschreiben, als prominentestes Beispiel Text, aber auch Grafiken oder abstraktes Wissen. Ein Grundgedanke hinter XML ist es, Daten und ihre Repräsentation zu trennen, also beispielsweise Wetterdaten einmal als Tabelle und einmal als Grafik auszugeben, aber für beide Anwendungen die gleiche Datenbasis im XML-Format zu nutzen. Ein

weiterer Vorteil, der sich durch den Einsatz von XML ergibt, ist die leichte Erweiterbarkeit der entsprechenden Grammatiken. Bei den meisten Erweiterungen der Grammatik bleiben die vorher erstellen XML-Dateien, in dem hier vorliegenden Fall also Simulationsmodelle, auch gültig bezüglich der neuen Grammatik.

In den letzten Jahren hat die XML vermehrt in die Gestaltung von Software-Schnittstellen Einzug gehalten. Die XML wird zum einen bei der Gestaltung von Dateiaustauschformaten eingesetzt, um Dateninhalte zwischen verschiedenen Anwendungen oder Anwendungsmodulen auszutauschen [Rock00]. Zum anderen gewinnt sie wegen ihrer leichten Erweiterbarkeit steigende Bedeutung bei der Gestaltung von Interprozess-kommunikationsschnittstellen, wo Anwendungen Informationen zur Laufzeit mittels eines festgelegten Formats austauschen [Rock00].

**Bewertung**
Für das hier zu entwickelnde Werkzeug kann der Einsatz der XML in beiden Bereichen als sinnvoll erachtet werden. Die XML kann sowohl dazu dienen, die entsprechenden Simulationsmodelle entsprechend der Modellbeschreibung in einem festgelegten Format abzulegen, als auch die Kommunikation zwischen den verschiedenen Werkzeugmodulen, beispielsweise Visualisierungskomponente und Simulatorkern, auf Basis eines Nachrichtenformates zu realisieren.

## 3.6 Fazit

Für die Modellierung und Simulation von Fertigungssystemen auf der Betrachtungsebene einer diskreten Materialflusssimulation sind zahlreiche Methoden bekannt. Als Vorgehensmodell werden spezifische Projekte meist in Form einer Simulationsstudie umgesetzt. Der Basisprozess der schrittweisen Modellierung, Simulation und Analyse der abzubildenden Systeme lässt sich potentiell durch mehrere Anwender in einer interaktiven Umgebung erarbeiten, auch wenn dies noch in keinem bekannten Software-Werkzeug unterstützt wird.

Im Rahmen der Untersuchung des Standes der Technik konnten keine Modellbeschreibungen identifiziert werden, die alle Anforderungen bereits umsetzen. Alle vorgestellten Modellbeschreibungen haben in ihrem Bereich durchaus ihre Berechtigung, schränken die Anwendung aber soweit ein, dass die hier gestellten Anforderungen nicht mehr in Gänze erfüllt werden können. Für die hier vorliegende Arbeit bedeutet das, nach der Festlegung einer Prozessstruktur eine Modellbeschreibung zu entwickeln, die durch ein modular aufgebautes Software-Werkzeug implementiert werden kann, so dass alle Anforderungen erfüllt werden können. Die relevanten Kriterien sind aus Abschnitt 2.3 bekannt und können auch nach Sichtung des Standes der Technik übernommen werden. Eine Kombination aus Framework-basierter Modellierung mittels einer Programmiersprache mit Anbindung an die wesentlich anwenderfreundlichere, grafische Modellierung erscheint besonders vielversprechend. Die Implementierung des Werkzeugs als dritter Schritt stützt sich dann auf diese grundlegende Modellbeschreibung. Für den Simulatorkern ergibt sich die Anforderung, sowohl die zeit- wie auch die ereignisdiskrete Zeitfortschreitung zu implementieren, um den Einsatz des Werkzeugs in den verschiedenen Planungsphasen zu ermöglichen.

Das Problemfeld dieser Arbeit liegt im Bereich der Modellierung und Simulation komplexer Fertigungssysteme, die in einer virtuellen Umgebung dargestellt werden sollen, um sie möglichst immersiv und interaktiv parallel zu ihrer Ausführung analysieren und optimieren zu können. In diesem Feld sind verschiedene Implementierungen zur Darstellung von Fertigungssystemen in einer virtuellen Umgebung bekannt, an die sich bei der Umsetzung angelehnt werden kann. Neben einer hoch auflösenden, dreidimensionalen Visualisierung sollen dem Anwender aber auch weitere Darstellungsformen in dem zu entwickelnden Werkzeug angeboten werden.

Im Rahmen der Umsetzung der zu entwerfenden Modellierungsphilosophie (hier verstanden als die Entwicklung eines Basisprozesses inklusive einer passenden Modellbeschreibung) in ein entsprechendes Software-Werkzeug sind insbesondere für die objektorientierte Programmierung zahlreiche Paradigmen bekannt, die bestmöglich berücksichtigt werden sollen. Für den Aufbau von modular organisierten, interaktiven Anwendungssystemen existieren mehrere Architekturmuster, die bei dem strukturierten Aufbau des Werkzeuges innerhalb einer festzulegenden Organisationsform angewendet werden können. Insbesondere die Unterstützung von Kommunikations-, Koordinations- und Kooperationsfunktionen muss umgesetzt werden, damit das Werkzeug als Mehrbenutzersystem angewendet werden kann. Bei der Entwicklung der eigentlichen Software ist die höchstmögliche Verwendung der UML zu befürworten, um Planungsfehler im Rahmen der Software-Entwicklung im Vorfeld vermeiden zu können. Als Software-Schnittstelle erscheint die XML sowohl als Datenaustauschformat, als auch als Kommunikationsschnittstelle als besonders geeignet.

Das folgende Kapitel nimmt die in Kapitel 2.3 aufgezeigte Strukturierung der Gesamtaufgabe auf und definiert die Umsetzungsschritte unter Berücksichtigung der in Kapitel 1 aufgezeigten Lösungsmöglichkeiten.

# 4 Zielstellung

*„Der Mensch ist ein zielstrebiges Wesen, aber meistens strebt es zu viel und zielt zu wenig."*

*(Günter Radtke)*

In dieser Arbeit soll ein Materialflusssimulator konzipiert und umgesetzt werden, der mit einem diskreten, wahlweise zeit- oder ereignisorientierten Verfahren richtungsoffene Simulationsmodelle so berechnet, dass Mehrbenutzer-Modellierung, -Simulation und -Analyse in einer interaktiven, immersiven und virtuellen Umgebung ermöglicht wird (vgl. Abschnitt 2.3).

Eine Analyse des Standes der Technik hat verschiedene Modellbeschreibungen aufgezeigt, die eine Abbildung von Fertigungssystemen in einem frei wählbaren Abstraktionsgrad prinzipiell ermöglichen. Keine der identifizierten Modellbeschreibungen erlaubt jedoch die Erfüllung aller aufgezeigten Anforderungen in Gänze. Nach der Festlegung grundlegender Annahmen für den Prozess der Modellierung, Simulation und Analyse, wie er durch das Werkzeug unterstützt werden soll (siehe Abschnitt 4.1), muss in einem zweiten Schritt eine Modellbeschreibung von ausführbaren Materialfluss-modellen als Kombination und Erweiterung der vorgestellten Lösungen konzipiert werden, die alle aufgezeigten Anforderungen erfüllen kann. Abschnitt 4.2 beschreibt die dazu nötigen Schritte, beginnend mit der sukzessiven Entwicklung einer modularen, objektorientierten Modellbeschreibung zur mehrbenutzerfähigen Modellierung interaktiver Simulationsmodelle. In einem Folgeschritt wird diese Modellbeschreibung hinsichtlich einer Transformation in ein Simulationsmodell zur Rückwärtssimulation hin untersucht, überprüft und gegebenenfalls erweitert.

Letztendlich soll dem Anwender ein Werkzeug zur Verfügung gestellt werden, dass eine benutzerfreundliche Bearbeitung der dargestellten Simulationsmodelle erlaubt. Abschnitt 4.3 untergliedert den Entwurfs- und Implementierungsprozess des Werkzeugs in die Phasen Systementwurf, Systemarchitektur und Realisierung. Der erste Bereich beschreibt die Strukturierung der Software in Module, um den ganzheitlichen Ansatz der angestrebten Lösung zu unterstreichen. In der Phase der Systemarchitektur werden die unter Anwendung der UML (Use-Case-Diagramme) identifizierten Teilsysteme miteinander verzahnt. Daraufhin werden die einzelnen Funktionsmodule mittels der in Abschnitt 3.5.6 aufgezeigten Architekturmuster entworfen und überprüft. Schließlich werden im Rahmen der Realisierung die einzelnen Module innerhalb des Gesamt-konzeptes implementiert und anhand eines Beispielmodells validiert.

## 4.1 Gestaltung eines Basisprozesses

Die Entwicklung des angestrebten Materialflusssimulators setzt einige Basisannahmen über den Arbeitsprozess mit dem Software-Werkzeug voraus. Dessen Gestaltung muss vor der eigentlichen Entwicklung von Modellbeschreibung und Werkzeug ebenso festgelegt und beschrieben werden, wie alle Überlegungen hinsichtlich aller übergreifenden Entscheidungen (Programmiersprache, etc.). Die entsprechende Wahl der

Grundentscheidungen kann wieder den eigentlichen Arbeitsprozess der Modellierung und Simulation beeinflussen. Als Basis dient das unter Abschnitt 2.4.1 vorgestellte, grundsätzliche Vorgehen, das nach Abschnitt 3.1 innerhalb einer Simulationsstudie angewendet werden kann.

**Mehrbenutzerbetrieb und Interaktion**
Die einzelnen Bereiche innerhalb eines Simulationsmodells müssen sich so voneinander trennen lassen, dass eine Manipulation des Gesamtmodells ermöglicht wird. Alle wesentlichen Modelleigenschaften müssen sich während der Ausführung eines Simulationsmodells manipulieren lassen, die entsprechenden Interaktionen müssen protokolliert werden. Die noch zu spezifizierenden Interaktionsmetaphern (Erzeugen, Selektieren, Parametrieren, Erweitern, Löschen etc.) führen unter Umständen zu umfangreichen Änderungen an dem Simulationsmodell. Sie müssen über die entsprechenden Benutzerschnittstellen nicht nur ermöglicht werden, sondern auch in einem Rahmen verwendet werden, der die Konsistenz des Gesamtmodells nicht gefährdet. Basis der Arbeit mehrerer Anwendern an einem gemeinsamen Modell ist der Entwurf und die Implementierung einer Benutzerverwaltung inklusive eines Rechtemanagements. So können verschiedene Benutzergruppen mit unterschiedlichen Aufgabenbereichen voneinander unterschieden werden.

## 4.2 Anforderungen an eine Modellbeschreibung

Zweiter Baustein der Umsetzung vor Systementwurf und Implementierung des Werkzeugs muss die Entwicklung einer Modellbeschreibung sein, die allen aufgezeigten Anforderungen an die Modellierung und Simulation von Fertigungssystemen genügt. Dabei sind alle Anforderungen zu berücksichtigen, die aus einem Multitasking-Betrieb und den Interaktionsmöglichkeiten resultieren, um diese in den Einsatzfeldern Vorwärts- und Rückwärtssimulation von komplexen Simulationsmodellen zu erlauben. Die jeweiligen Vorteile der unter Abschnitt 3.2 aufgezeigten Lösungsalternativen sollen bestmöglich in einem Ansatz integriert werden. Die Benutzerfreundlichkeit soll gesteigert werden, indem zunächst eine grafische Modellierung anhand von Bausteinbibliotheken ermöglicht wird. Zur Modellierung spezieller Algorithmen und Methoden innerhalb eines Modellbausteins soll ein Framework-basierter Ansatz verfolgt werden, indem die Gültigkeit des erstellten Programmcodes direkt während der Modellierung überprüft werden kann. Die Gestaltung soll schrittweise erfolgen, je nach Anforderung erweitert werden können. Dabei soll von einer allgemeinen Modellbeschreibung ausgegangen werden; zusätzliche Anforderungen werden Schritt für Schritt integriert.

### 4.2.1 Vorwärtssimulation

**Allgemeine Simulationsmodelle**
Die allgemeine Modellbeschreibung der Simulationsmodelle soll sich konsequent an Grundsätzen der objektorientierten Programmierung orientieren, um eine modulare und objektorientierte Struktur des Simulationsmodells zu erhalten, die als Datenmodell des Simulators verwendet werden kann. Die Simulationsmodelle sollen unter Anwendung von Synchronisationsverfahren interaktiv verändert werden können und prinzipiell einen beliebigen Detaillierungsgrad in hierarchischen Strukturen abbilden können.

**Anforderungen zur dynamischen Detaillierung**

Aus den Integrationsbemühungen innerhalb einer ganzheitlichen Planung resultiert zwangsläufig eine wachsende Komplexität der Simulationsmodelle und ihrer 3D-Visualisierungen. Um beispielsweise die Simulation einer kompletten Supply-Chain mit dem zu entwickelnden Werkzeug dennoch zu ermöglichen, muss im Rahmen der Modellbeschreibung ein bekanntes Verfahren zur dynamischen Detaillierung von Simulationsmodellen adaptiert und in die Modellbeschreibung integriert werden [Muec05]. Darüber hinaus muss auch im Rahmen der Entwicklung des Simulatorkerns eine entsprechende Umsetzung dieser Erweiterung der Modellbeschreibung berücksichtigt werden.

**Anforderungen funktionsorientiert gegliederter Fertigungen**
Neben der Abbildung von Fertigungssystemen, deren Gliederung sich an dem zu fertigenden Erzeugnis orientiert, soll auch die Modellierung und Simulation von funktional gegliederten Fertigungssystemen, bzw. deren Vermischung in gemischt objekt- und funktionsorientierten Fertigungssystemen durch das zu entwickelnde Werkzeug und damit die dem Werkzeug zugrunde liegende Modellbeschreibung unterstützt werden. Als spezielle Anforderung der funktionsorientierten Fertigungsprinzipien ist insbesondere die Möglichkeit zur leichten und flexiblen Modellierung von Transportwegen innerhalb eines vorgegebenen Layouts zu berücksichtigen.

**Mehrbenutzerfähige Modellierung**
Prinzipiell müssen alle Anforderung über entsprechende Mechanismen so ausgelegt werden, dass eine Ausführung auch in einem Multitasking-System mit mehreren an einem gemeinsamen Simulationsmodell angeschlossenen Anwendern möglich ist. Die entsprechenden Zugriffe müssen aufgenommen, verwaltet und konsistent ausgeführt werden. Potentielle Konflikte zwischen Interaktionen verschiedener Anwender müssen bestmöglich automatisiert durch das Werkzeug aufgelöst oder im Vorhinein durch eine entsprechende Gestaltung der Benutzerschnittstelle verhindert werden.

### 4.2.2 Rückwärtssimulation und Modelltransformation

Der Einsatz der Ablaufsimulation soll sich mit diesem Werkzeug über die reine Fertigungsprozessplanung hin zur Absicherung der Planungsphasen der Fertigungslenkung und Prognose aktueller Systemzustände aus der Fertigungssteuerung erweitern. Darum ist die Modellbeschreibung so zu gestalten, dass der abgebildete Materialfluss in einer Vorwärts- und Rückwärtssimulation berechnet werden kann. Um den Modellierungsaufwand des Anwenders nicht signifikant zu erhöhen, soll eine Möglichkeit identifiziert werden, eine (semi-)automatische Transformation zwischen den verschieden gerichteten Modellen zu erreichen. Neben Fragestellungen bezüglich der maximalen Leistungsfähigkeit eines Fertigungssystems kann dadurch auch die Qualität bestehender Fertigungsprogramme auf Basis desselben Modells überprüft und ggf. verbessert werden. Bestehende Fertigungsprogramme oder vorliegende Kundenaufträge mit Auslieferungsterminen dienen hierbei als Eingabequellen für das rückwärts gerichtete Simulationsmodell. In einem Folgeschritt können diese Verfahren eingesetzt werden, um Fertigungsprogramme auch direkt aus der Ablaufsimulation zu erzeigen und zu optimieren.

Abbildung 23 zeigt schematisch die Vorgehensweise beim Entwurf der Modellierungsmethode für das zu entwickelnde Werkzeug. Darauf aufbauend wird im nachfolgenden Schritt das eigentliche Werkzeug konzipiert.

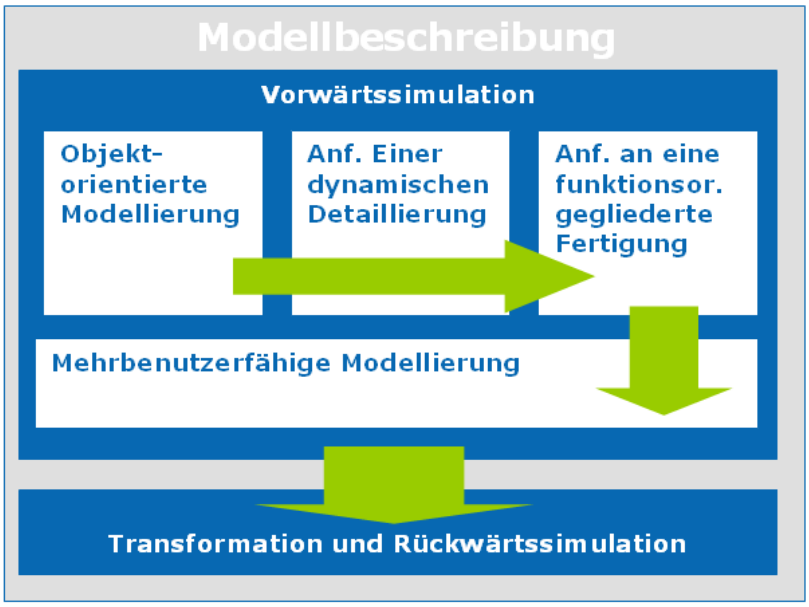


**Abbildung 23: Entwurfsphasen der Modellierungsmethode**

## 4.3 Entwurf eines Werkzeuges

Dem Anwender soll im gesamten Arbeitsprozess innerhalb der jeweiligen Visualisierungskomponenten ein hohes Maß an Immersion zur Verfügung stehen. Neben einer möglichst guten Darstellungsqualität in der virtuellen Realität sind bei der Gestaltung der Software Benutzerschnittstellen zu realisieren, die ein reales, interaktives Verhalten bestmöglich abbilden. Die interaktive Ausführung setzt eine bidirektionale Kopplung zwischen den Modulen Simulatorkern und den angeschlossenen Visualisierungskomponenten voraus, damit die Interaktionen direkte Auswirkungen auf Modellierung oder Ausführung der Simulation haben.

Das Werkzeug selbst muss im Rahmen eines Software-Entwicklungsprozesses konzipiert, modelliert und implementiert werden. Anhand eines Beispielmodells soll anschließend der Nachweis geführt werden, dass das entwickelte Werkzeug den gestellten Anforderungen genügt und sich für den Einsatz als integriertes Werkzeug eignet.

**Systementwurf**

Innerhalb der ersten Phase, dem Systementwurf, werden zunächst grundlegende Grobgliederungen anhand darzustellender Use-Cases erstellt und modular strukturiert. Dabei kommen erste Architekturmuster zum Einsatz, die eine trennscharfe Strukturierung in Module erlauben und die jeweiligen Modulgrenzen aufzeigen können. Entsprechende Kommunikationsschnittstellen zwischen den Modulen werden in einem Grobentwurf festgelegt, um die Tauglichkeit des Systementwurfs darstellen und nachweisen zu können.

**Systemarchitektur**

In der Folgephase werden mit Hilfe weiterer Architekturmuster die einzelnen Module detaillierter konzipiert und hinsichtlich ihrer Aufgabenstellung verbessert. Insbesondere die Übertragung des Datenmodells aus der Modellbeschreibung in die jeweilige Kernfunktionalität der einzelnen Module spielt hier eine wesentliche Rolle. In einem weiteren Schritt werden die jeweiligen Software-Schnittstellen zwischen den einzelnen Modulen genauer ausgearbeitet und die Kommunikation detailliert. Als Ergebnis dieser Entwurfsphase sind die einzelnen Module sowie ihre Beziehungen untereinander definiert und abgestimmt. In einem nächsten Schritt müssen sie schließlich in der Implementierung umgesetzt und die entsprechenden Benutzerschnittstellen gestaltet werden.

**Realisierung**
Unter Berücksichtigung der Zielstellung werden in der Realisierungsphase zunächst die einzelnen Programmmodule implementiert und die definierten Kommunikations-schnittstellen umgesetzt. Die Implementierung erfolgt auf Basis der in der vorherigen Phase festgelegten Schematisierung und innerhalb der entsprechenden Architektur-muster, bzw. Organisationsformen.

In der Realisierung wird die Implementierung anhand eines umfangreichen Beispielmodells erprobt und der eigentliche Modellierungs- und Simulationsprozess innerhalb des Werkzeuges beschrieben. Das Beispiel wird in Form einer Simulationsstudie durchgeführt, dazu also zunächst der Untersuchungsgegenstand erläutert und nachfolgend modelliert, simuliert und visualisiert.

# 5 Konzeption

*„Der Anfang ist die Hälfte vom Ganzen."*

*(Aristoteles)*

In dem zu entwickelnden Werkzeug sollen Simulationsmodelle erstellt und mittels eines Simulatorkerns ausgeführt werden. Grundlegende Basis für ein solches Software-Tool ist zum Einen eine Struktur des Arbeitsprozesses, der durch das Werkzeug unterstützt werden soll, zum Anderen Grundannahmen über die Beschaffenheit des Werkzeuges und eine Modellbeschreibung, anhand derer gültige Simulationsmodelle definiert werden. In Abschnitt 5.1 sollen zunächst einige Basisannahmen über den Arbeitsprozess beschrieben werden. In Abschnitt 5.2 wird daraufhin eine Modellbeschreibung, bzw. ein objektorientiertes Strukturmodell aufgebaut, mit dem Simulationsmodelle in dem zu entwickelnden Tool beschrieben werden.

Im Abschnitt 5.3 soll das Werkzeug selbst im Rahmen eines Software-Entwicklungsprozesses konzipiert und modelliert werden. Auf Basis der in Abschnitt 4.3 gestellten Anforderungen wird zunächst ein Systementwurf durchgeführt, der die Gesamtanforderungen in Funktionsmodule strukturiert. Diese werden in den folgenden Abschnitten ausgeplant und höher detailliert entworfen, um sie in der Implementierungsphase umsetzen zu können. Bei der Modellierung des Werkzeugs werden die Ergebnisse aus der Konzeption der Modellbeschreibung und des Nachrichtenprotokolls berücksichtigt.

## 5.1 Gestaltung eines Basisprozesses

### 5.1.1 Arbeitsprozess Modellierung und Simulation

Der Basisprozess von Modellierung und Simulation, wie er in anwendungsorientierten Simulatoren heute unterstützt wird, bildet die Grundlage für das zu entwickelnde Werkzeug, weil dieser den Anwender von allen vorgestellten Methoden am effizientesten bei seiner Arbeit unterstützt. Er basiert auf Grundlage des in Abschnitt 3.1 gezeigten Ablaufs einer Simulationsstudie. Auf Basis eines logischen Modells, das bereits in früheren Phasen einer Simulationsstudie für den abzubildenden Materialfluss erstellt wurde, wird in einem ersten Schritt das Fertigungssystem mittels einer grafischen Oberfläche aus Bausteinen zu einem Simulationsmodell zusammengesetzt, indem einzelne Bausteine aus Bibliotheken entnommen, in ein übergeordnetes Modell eingefügt und mit anderen Bausteinen über logische Kanten verknüpft werden. Das resultierende Modell kann gespeichert und im Simulator ausgeführt werden. Je nach Anwendung wird der logisch abgebildete Materialfluss animiert, lässt sich ggf. modifizieren und/oder mittels Interaktion beeinflussen. Im Rahmen der Modellierung wird das entstehende Simulationsmodell zunächst verifiziert. Komplexere Steuerungen werden über spezielle Methoden in den Modellbausteinen implementiert. Im Anschluss werden Simulationsexperimente und deren einzelne Simulationsläufe geplant. Abbildung 24 zeigt schematisch den typischen Prozess von der Modellierung über die Ausführung von Simulationsexperimenten bis hin zur Auswertung der gesammelten Daten.


**Abbildung 24: Idealtypischer Simulationsprozess**

Aus dieser funktionalen Unterscheidung der Aufgabe des Simulationsexperten kann direkt eine modulare Aufteilung des Werkzeugs geschlussfolgert werden. Diese soll im Folgenden näher beschrieben werden.

### 5.1.2 Modulare Architektur

Auf Basis des grundlegenden Modellierungsprozesses können verschiedene Funktionsmodule unterschieden werden, die unter Berücksichtigung der besonderen Anforderung nach Interaktivität das zu entwickelnde Gesamtsystem bilden sollen. Abbildung 25 greift die schematische Struktur des Gesamtsystems aus Abschnitt 2.4.1 nochmals auf und erweitert sie um administrative Funktionen.

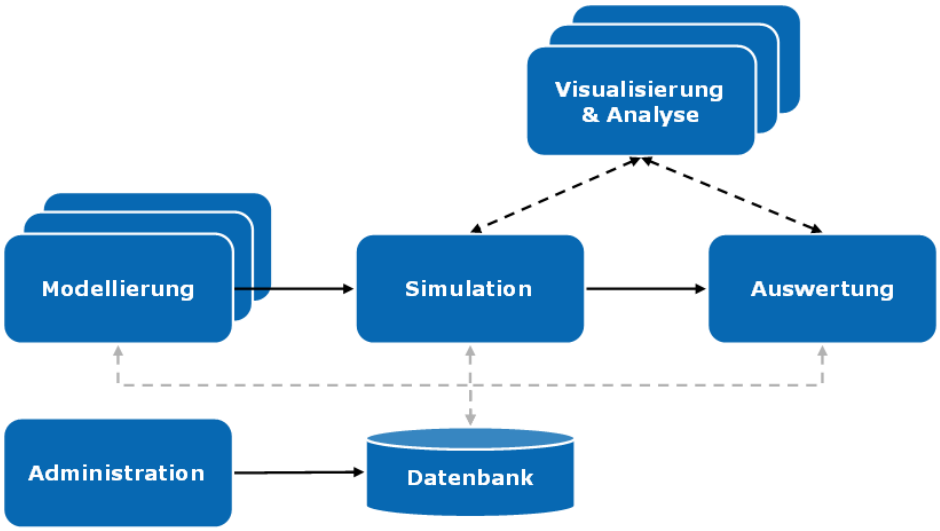


**Abbildung 25: schematische Darstellung der Funktionsmodule des Werkzeugs**

Nach der grafischen Modellierung der Simulationsmodelle durch den Anwender sollen diese in einem Simulatorkern berechnet und mittels einer Visualisierungskomponente angezeigt werden. Modellierung und Visualisierung sollen sowohl mehrbenutzerfähig in zwei-, als auch dreidimensionaler Darstellung ermöglicht werden. Die einzelnen Bausteine sollen gemäß Anforderung keinen Detaillierungsgrad der Modellierung vorgeben, sondern aus Grundelementen zusammengesetzt sein. Zur Modellierung der Verhaltenslogik soll eine integrierte Programmiersprache dienen, wie sie analog in kommerziellen Simulatoren angewendet wird. Basis der Modellbeschreibung und Programmierung ist aber ein einheitliches Verständnis der Modelle und Funktionen des Simulators, wie es aus der Verwendung von Frameworks bekannt ist (vgl. 3.2.2). Das Simulationsmodell kann im nächsten Schritt im Simulatorkern berechnet werden; die dynamischen Prozesse werden in den angeschlossenen Visualisierungskomponenten animiert. Diese beinhalten auch die Möglichkeit zur Interaktion mit dem

Simulationsmodell und zeigen durchgeführte Änderungen direkt an, um eine stetige, immersive Verbesserung des Simulationsmodells zu erlauben.

Basis aller Module ist neben einer einheitlichen Beschreibung der Simulationsmodelle und einem Nachrichtenformat für die Kommunikationsschnittstelle auch eine integrierte Datenhaltung über alle Module für die anfallenden Simulationsdaten während der Durchführung einer Simulationsstudie. Neben dem Simulationsmodell an sich und den benötigten Funktionsbibliotheken sind das insbesondere auch die zwei- bzw. dreidimensionalen Repräsentanten der Funktionsbausteine, die in den entsprechenden Benutzeroberflächen verwendet werden. Auf Basis vorhandener dreidimensionaler Repräsentanten für einen bestimmten Modellbaustein soll die zweidimensionale Darstellung automatisch abgeleitet werden, um den Aufwand der Datengenerierung zu minimieren.

### 5.1.3 Integration von Layout- und Fertigungsprozessplanung

Das Erstellen und Bearbeiten des Simulationsmodells innerhalb der Modellierungskomponente soll layoutgerechtes Anordnen der Modellbausteine erlauben. Auf Basis eines Rasters oder durch die Einbeziehung von vorhandenen Layouts können die Modellbausteine eines Simulationsmodells bereits in der Modellierungskomponente layoutgerecht angezeigt werden. Neben einer realistischeren Darstellung des Simulationsmodells können somit auch direkt Entfernungen von Transportwegen oder die Auslegung von Förderstrecken richtig angelegt werden. Innerhalb einer ganzheitlichen Planung können somit zwei aufeinander folgende Aufgaben innerhalb des zu entwickelnden Werkzeugs integriert werden: Die Layoutplanung und die Fertigungsprozessplanung (vgl. hierzu auch Abschnitt 2.1).

Bei der Konzeption und Umsetzung des eigentlichen Werkzeugs ist auf die Möglichkeit zu achten, Modellbausteine in das Simulationsmodell zu integrieren, die keine eigentliche Funktion im Sinne der Ablaufsimulation verfolgen. Sie dienen lediglich der realistischen Ausgestaltung des Szenarios bzw. Layouts oder als Grundlage für die Anordnung der Modellbausteine (beispielsweise die Abbildung einer Fabrikhalle in Form eines Hallenlayouts). Die integrierte Datenhaltung erlaubt darüber hinaus zumindest prinzipiell auch eine Aufteilung der beiden Arbeitsaufgaben in verschiedene Module, solange auf das einheitliche Datenmodell zurückgegriffen wird.

### 5.1.4 Programmiersprache JAVA als Simulationssprache

Für die Abbildung komplexer Steuerungslogiken verwenden Materialflusssimulatoren innerhalb der Modellbausteine spezifische Modellierungssprachen, mit denen das gewünschte Verhalten programmiert und in dem Baustein hinterlegt werden kann. Vorhandene Modellbausteine müssen durch speziell angepasste Methoden auf das individuelle Verhalten eingestellt werden können. Das Erlernen der dazu benötigten spezifischen Simulationssprachen ist ein zeitaufwendiger Prozess und während der Modellierung eine häufige Fehlerquelle. Um die Fehleranfälligkeit hier zu reduzieren und das Erlernen der Modellierungssprache zu erleichtern, soll im Rahmen der vorliegenden Arbeit eine „handelsübliche" Programmiersprache als Modellierungssprache verwendet werden. Sie muss wegen der Objektorientierung der Simulationsmodelle eine objektorientierte Sprache sein. Innerhalb der hier vorliegenden Arbeit wurde Java als

Programmiersprache ausgewählt. Alternativ ständen andere objektorientierte Programmiersprachen oder simulationsspezifische Programmiersprachen zur Verfügung. Aus den nachfolgend kurz dargestellten Gründen wurde jedoch auf Java zurückgegriffen.

Java wurde ursprünglich von Sun Microsystems entwickelt. Es ist eine objektorientierte Programmiersprache, die sich durch einige zentrale Eigenschaften auszeichnet. Diese machen sie universell einsetzbar und für die Industrie als robuste Programmiersprache interessant. Da Java objektorientiert ist, spiegelt es den Wunsch der Anwender wider, moderne und wieder verwendbare Komponenten zu programmieren. Im Gegensatz zu herkömmlichen Übersetzern einer Programmiersprache, die Maschinencode für eine spezielle Plattform generieren, erzeugt der Java-Compiler Programmcode für eine virtuelle Maschine, den so genannten Bytecode, der prinzipiell auf allen Rechnerplattformen interpretiert werden kann. Java wird in allen IT-Bereichen von Handel, Industrie und Verwaltung eingesetzt und ist für unzählige Betriebssysteme und Plattformen, vom mobilen Telefon bis hin zur Echtzeit-Großrechneranlage kostenlos verfügbar. Neben einer hohen Anzahl von Anwendern stehen zahlreiche Bibliotheken zur Verfügung, die bei der Modellierung verwendet werden können. Beispielsweise stehen umfangreiche Mathematik- und Statistik-Bibliotheken zur Verfügung, die in das System integriert werden können. Die Eigenschaft als objektorientierte Sprache ermöglicht den Einsatz von Kapselung, Vererbung und Hierarchisierung (vgl. Abschnitt 3.5.2), wie sie in den Modellbausteinen benötigt werden. Sie ergänzt damit den objektorientierten Ansatz der Modellbeschreibung, der später genauer beschrieben wird. Java eignet sich prinzipiell auch für den Einsatz von Mehrbenutzersystemen, da eine wesentliche Eigenschaft, das parallele Ausführen mehrerer Prozesse (Threads), unterstützt wird. Java ist darüber hinaus für spätere Anwender des Systems leichter zu lernen, da aufgrund der weiten Verbreitung zahlreiche Tutorien und Kurse zur Verfügung stehen. Das Erlernen einer anwendungsspezifischen Simulationssprache, die nur für dieses spezifische Simulationswerkzeug gültig ist, entfällt somit. Eine Einführung des Werkzeuges in den laufenden Betrieb bei potentiellen Kunden wird somit erleichtert.

Aus den oben genannten Gründen soll Java auch als Implementierungssprache für das zu entwickelnde Werkzeug dienen. Der dadurch resultierende, parallele Einsatz von Java als Modellierungs- und Implementierungssprache bietet das Potential einer engeren Abstimmung zwischen Werkzeug und Modelllogik, was durch die Transformation der Modellbeschreibung in ein objektorientiertes, Java-basiertes Datenmodell weiter gefördert werden kann und soll. Durch diese enge Verzahnung der Daten und der Berechnung im Simulator soll der Geschwindigkeitsnachteil, den Java gegenüber anderen, objektorientierten Programmiersprachen wie beispielsweise C++ noch hat, weitestgehend kompensiert werden. Eine Verschmelzung von Simulator und Simulationsmodell in einem einzigen, lauffähigen Java-Programm ermöglicht darüber hinaus ein einfaches Ankoppeln unterschiedlicher Visualisierungs- und Auswertungs-module. Die Gestaltung eines Mehrbenutzerbetriebs, bzw. die Verarbeitung eines bestimmten Simulationsmodells mit mehreren Visualisierungsmöglichkeiten wird dadurch zumindest potentiell schon ermöglicht. Der Basisprozess muss für eine solche Ausführung marginal angepasst werden, indem die zwischenzeitliche Speicherung des Simulationsmodells um das Übersetzen in ein lauffähiges Java-Programm erweitert wird. Dadurch ergibt sich als weiterer Vorteil eine implizite Validierung der in den Methoden hinterlegten Verhaltensbeschreibung. Durch die Integration von vorhandenen

Debuggern[20] kann auch die Verifikation des Simulationsmodells in der Modellierungskomponente erleichtert werden. Auch hier können existierende Lösungen im Bereich der Programmiersprache in das entsprechende Modul integriert werden, um die Arbeit des Anwenders zu erleichtern. Der angepasste Basisprozess gestaltet sich demnach wie folgt:

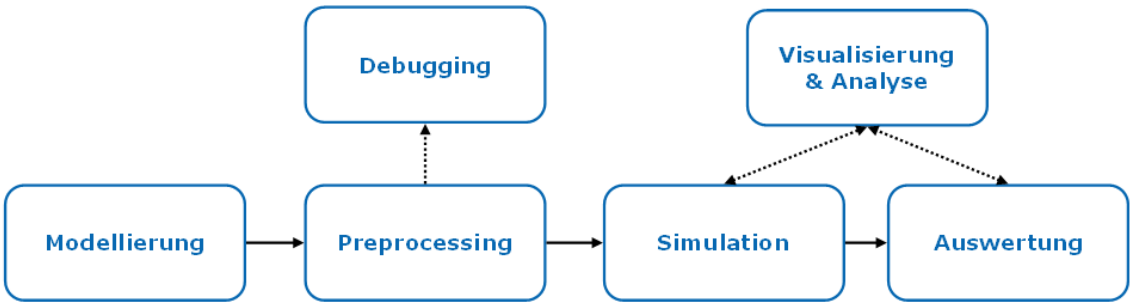


**Abbildung 26: angepasster Basisprozess für die Modellierung und Simulation**

### 5.1.5 Grundlegende Merkmale der Modellierung und Simulation

Vor dem eigentlichen Start eines Simulationslaufs muss eine Initialisierungsphase ausgeführt werden, mittels der das Simulationsmodell mit Eingabedaten aus Simulationsdatenbank oder dem Dateisystem vorbelegt werden kann. Prozessabbilder oder Fertigungsprogramme können als Startparametrierung dienen, um die Einschwingphase des Simulationsmodells zu minimieren oder ganz zu kompensieren. Hier können Initialnachrichten an die angeschlossenen Visualisierungskomponenten gesendet werden, um die entsprechende Darstellung des Simulationsmodells aufbauen zu können (vgl. hierzu auch Abschnitt 3.4.2). Danach beginnt der Simulationslauf. Ein *Experimentmanager*-Modul soll mehrere Simulationsläufe verwalten können, die hintereinander oder parallel ablaufen können, damit der Anwender in der Experimentierphase einer Simulationsstudie die benötigten Simulationsläufe an einer zentralen Stelle parametrieren und ausführen kann.

Zur Generierung der benötigten Simulationsdaten innerhalb eines Simulationslaufs sollen einzelne Variablen eines Modellbausteins „abonniert" werden können, deren Werteveränderungen dann in dem Simulationslauf entweder vom Simulationskernel oder dem Visualisierungsmodul protokolliert werden. Alternativ können den durch das Simulationsmodell laufenden Marken Parameter aufgeprägt werden, um diese in einem zentralen Modellbaustein auszuwerten. Die Datenmenge, die während eines Simulationslaufs anfällt, kann somit durch den Anwender spezifisch festgelegt und je nach Zweck des Simulationslaufs angepasst werden. Wegen der Interaktionsmöglichkeiten des Anwenders während des Simulationslaufes müssen darüber hinaus alle vorgenommenen Interaktionen protokolliert werden, denn während des Simulationslaufes können Variablen von den Anwendern eingesehen und, ggf. in vorgegebenen Grenzen, verändert werden. Die Auswertungen des Simulationsexperimentes können nach Abschluss des Simulationslaufs individuell oder über Standardauswertungen in Modellbausteinen erfolgen. Durch die Darstellung dieser

[20] Ein Werkzeug zur Fehlerbereinigung von Software. Es ermöglicht in der Regel eine Ablaufverfolgung des zu untersuchenden Programms in einzelnen Schritten oder zwischen definierten Haltepunkten (nach [Balz05]).

Auswertungsbausteine und deren aktuellen Werte während der Ausführung eines Simulationsmodells wird eine erste Analyse schon während der Ausführung erreicht. Dadurch wird der Anwender befähigt, eventuelle Schwachstellen im Simulationsmodell früher zu erkennen und sein Simulationsmodell zu verbessern.

Simulationsmodelle sollen in der Modellbeschreibung hierarchisch abgelegt werden können. Ebenso sollen in dem zu entwickelnden Werkzeug auch die durch das Simulationsmodell laufenden Marken hierarchisch organisiert werden können. Diese „Token" beschreiben die durch das abzubildende System laufenden Erzeugnisse, Informationen oder andere Material- oder Informationsflüsse. Unter Anwendung des Composite-Patterns[21] soll eine Baumstruktur aus Token eingeführt werden, so das Token wiederum Token enthalten können, die sie mit durch das Simulationsmodell transportieren. Dadurch wird eine tokenspezifische Auswertung des Simulationsmodells ermöglicht. Zusätzlicher Vorteil dieser Vorgehensweise ist es, Hierarchien aus Token innerhalb eines Simulationsmodells zu bilden und im weiteren Verlauf wieder zu dekomponieren. Beispielsweise können 10 Token eines Typs „Messer" in ein Token vom Typ „Karton" gepackt werden. 50 Kartons können auf einer „Palette" gebündelt werden (Token vom Typ „Palette") und im weiteren Verlauf, beispielsweise bei einer Kommissionierung in einem späteren Bereich des Simulationsmodells wieder einzeln verwendet werden. In einem Auswertungsbaustein kann jedes Token seinen individuellen Weg durch das Fertigungssystem beschreiben und ermöglicht so eine sehr genaue Form der Datenauswertung, ohne dass die Funktion in jedem Modellbaustein des Simulationsmodells einzeln entwickelt werden müsste. Diese Informationsspeicherung in den Token erlaubt bei einer Integration verschiedener Simulationsmodelle eines Unternehmensnetzwerkes auch eine einfache Form der Datenübertragung in andere Simulationsmodelle oder andere Bereiche desselben, integrierten Simulationsmodells.

Das Simulationsmodell muss vor seiner Ausführung im Simulatorkern in ein lauffähiges Java-Programm kompiliert werden (vgl. Abbildung 26). Für die Speicherung des Simulationsmodells und seiner Parametrierung in einer Datenbank oder auf der Dateiebene des Betriebsystems bietet es sich an, ein effizienteres Format der Datensicherung zu wählen und auf Basis dieses Formats das Simulationsmodell in das zugehörige Java-Programm zu transformieren und anschließend zu kompilieren. Durch die automatisierte Verarbeitung wird während der Übersetzung nur unwesentlich Zeit verloren, im Gegenzug kann die Speicherung aber wesentlich effizienter und vor Allem erweiterbar gestaltet werden. Eine Versionierung sowie die Speicherung in einer Datenbank werden damit deutlich erleichtert. Im Rahmen der Entwicklung wird als Speicherformat auf die Extensible Markup Language (XML) zurückgegriffen, deren Vorteile unter Abschnitt 3.5.7 bereits aufgezeigt wurden. Alternativ bliebe nur ein spezifisches Speicherformat. Aufgrund der hohen Verbreitung der XML-Technologien wurde hier, ebenso wie bei der Auswahl der Programmiersprache Java auf einen offenen und erweiterbaren Standard zurückgegriffen. Nachfolgende Integrationen mit anderen Datenformaten können so durch relativ einfache Transformationen erreicht werden.

[21] Das Composite-Pattern fügt „Objekte zu Baumstrukturen zusammen, um Teil-Ganzes Hierarchien zu repräsentieren. Es ermöglicht es, einzelne Objekte ebenso wie Kompositionen von Objekten einheitlich zu behandeln." [nach GaHe94]

Insbesondere im Rahmen der Entwicklung und vor Allem kontinuierlichen Weiterentwicklung der Modellbeschreibung kann eine Wiederverwendung bestehender Simulationsmodelle problemlos erfolgen. Die leichte Erweiterbarkeit prädestiniert auch für den Einsatz im Rahmen der Kommunikationsschnittstellen, wo die ausgetauschten Nachrichtenformate mittels XML definiert und dadurch leicht hinsichtlich ihrer Gültigkeit überprüft werden können. Durch den Einsatz der Transformationssprache XSLT[22] können die im XML-Format vorliegenden Daten bzw. Nachrichten wieder in Java-Objekte umgewandelt und im weiteren Programmverlauf verwendet werden. Die bekannten Vorteile von XML zeigen sich auch hier: Die Kommunikationsschnittstelle, die auf Basis eines XML-Formates Nachrichtentypen untereinander austauscht, ist flexibel gegenüber zukünftigen Erweiterungen.

Während der Ausführung eines Simulationsexperimentes mit mehreren Simulationsläufen sammeln sich eine Menge von Auswertedaten an, die für eine spätere Auswertung zur Verfügung stehen und dauerhaft gesichert werden sollen. Um die Konsistenz der anfallenden Simulationsdaten bewältigen zu können, wird zur Datenhaltung eine zentrale Simulationsdatenbank konzipiert (vgl. Abschnitt 3.5.4.4). Ihre genaue Struktur wird in Abschnitt 5.3.2.5 näher erläutert. Diese Experimentdaten sollen innerhalb der Datenbank ebenfalls in einem XML-Format gesichert werden, inklusive einer Kopie des Nachrichtenstromes zur zeitversetzten Reproduktion der Animation des Simulationslaufs in einer der Visualisierungskomponenten. Die flexible Struktur ermöglicht auch hier eine konsistente Sicherung, auch wenn sich das Speicherformat der Experimentdaten erweitern sollte.

### 5.1.6 Konfliktvermeidung & Rechtemanagement im Mehrbenutzerbetrieb

Wesentlich für die Entwicklung des Materialflusssimulators ist die Absicherung eines fehlerfreien Mehrbenutzerbetriebes. Neben der Berücksichtigung des Multitasking-Betriebs innerhalb der Modellierung und Visualisierung muss die Mehrbenutzerfähigkeit auch als grundlegendes Merkmal stets mit berücksichtigt werden. Basis für jede Umsetzung eines Mehrbenutzersystems ist eine Benutzerverwaltung, um die verschiedenen Zugriffe nach Anwendern unterscheiden zu können. Nicht jeder Anwender soll in dem Werkzeug dieselben Möglichkeiten zur Manipulation von Simulationsmodellen und Simulationsläufen bekommen, um eine anwenderorientierte Bedienung gestalten zu können. Dazu wird nachfolgend ein Rechtemanagement entwickelt, um verschiedene Anwenderrollen berücksichtigen zu können.

Neben der in Abschnitt 3.5.4.3 genannten Unterstützung der Kommunikation, Koordination und Kooperation sowie der Umsetzung der unter Abschnitt 3.4.2 aufgeführten Interaktionsmetaphern Erzeugen, Selektieren, Löschen, etc. müssen insbesondere potentielle Konflikte aufgelöst werden, die durch das gemeinsame Modellieren und/oder Simulieren entstehen können, da sich die verschiedenen Interaktionen der Anwender gegenseitige beeinflussen können. Bei auftretenden Konflikten kann nach 4 Typen unterschieden werden:

[22] XSL-Transformation: Transformation einer in XML vorliegenden Beschreibung in ein Zielformat mittels der eXtensible Stylesheet Language (XSL) [W3C].

1. *Modellierung & Modellierung:* Zwei oder mehr Anwender pflegen gleichzeitig Änderungen in ein Simulationsmodell ein. Das erfordert eine Möglichkeit zur parallelen Modellierung. Dabei stellt z.B. der Zugriff auf dasselbe Objekt einen Konflikt dar, der durch das entsprechende Modul aufgelöst werden muss.
2. *Simulation & Simulation:* Dasselbe Simulationsmodell soll zeitgleich für zwei oder mehr Simulationsläufe benutzt werden. Die Lösung dieses Konflikts führt zur Möglichkeit, zwei Instanzen eines Simulationsmodells mit verschiedenen Parametereinstellungen parallel zu simulieren.
3. *Modellierung & Simulation:* Der dritte Typ ist die Kombination der beiden ersten Klassen. Bei der Durchführung eines Simulationslaufes, würden Änderungen am gleichen Modell zu einer Verfälschung der Ergebnisse führen, weil die verschiedenen Experimentläufe nicht mehr miteinander vergleichbar sind.
4. *Simulationslauf mit mehreren Anwendern:* Ähnlich wie beim Modellieren mit mehreren Anwendern müssen Interaktions-Konflikte verschiedener Anwender durch das entsprechende Modul aufgelöst werden.

Zum Auflösen der möglichen Konfliktklassen sind drei spezifische Methoden für verteilte Systeme vorgesehen (vgl. Abschnitt 3.5.4.1): *Locking*, *Cloning* und *Versioning*. Ein möglicher Weg zur parallelen Arbeit an einem Objekt ist das Cloning, hier verstanden als mehrfaches Instanzieren desselben Objekts. Zu Beginn einer Bearbeitung wird für jeden Anwender ein Klon als exakte Kopie des aktuellen Simulationsmodells erzeugt. Die Schwierigkeit liegt in der Integration der verschiedenen Klone am Ende der Bearbeitung. *Cloning* bietet sich insbesondere für die Konfliktklasse 3 an. Durch eine Kombination mit der Versionierung kann für jeden Klon seine Versionsnummer beim Kopieren übergeben werden. Bei der Ausführung von Simulationsläufen kann dann jeweils immer dieselbe Version eines Simulationsmodells geladen werden, um eine Vergleichbarkeit der Ergebnisse zu gewährleisten.
Für die Modellierung (Konfliktklasse 1) oder das Simulieren eines Simulationsmodells mit mehreren Anwendern (Konfliktklasse 4) bietet sich dieses Vorgehen nicht an, da hier durch die umfangreichen Interaktionen des Anwenders zu viele Konflikte auftreten können, die sich nicht immer im Nachhinein auflösen lassen. Die Nutzung eines Sperrmechanismus (Locking) umgeht dieses Problem (vgl. Abschnitt 3.5.4.1). Damit können mehrere Anwender gleichzeitig an einer gemeinsamen Modellinstanz arbeiten, wobei das Arbeiten in unterschiedlichen Teilbereichen desselben Modells unkritisch ist. Greifen zwei Anwender auf dasselbe Objekt innerhalb einer Modellinstanz zu, soll dieses automatisch durch den ersten Zugriff gesperrt werden, bis der erste Anwender seine Modifikationen durchgeführt hat. In den Visualisierungskomponenten muss die Sperrung entsprechend kenntlich gemacht werden, um andere Anwender über das Systemverhalten zu informieren. Durch die geplante Kommunikationsunterstützung (vgl. Abschnitt 3.5.4.3) kann die Zusammenarbeit in der virtuellen Umgebung koordiniert werden. Unabhängig von dem realen Standort der Anwender kann so eine Form der Kooperation erreicht werden.

Unter Berücksichtigung von Simulationsmodellen, die hierarchisch angeordnet sind oder in verschiedenen Detaillierungsgraden vorliegen, gestaltet sich die Implementierung des Sperrmechanismus komplexer. Die Sperrung eines spezifischen Objekts muss sich direkt auf die anderen Detaillierungsgrade und/oder Hierarchieebenen des Simulationsmodells auswirken. Abbildung 27 zeigt ein Beispiel für die anvisierte Funktionsweise des

Sperrmechanismus. Zwei Anwender können damit genau dann parallel an einer Modellinstanz arbeiten, wenn die selektierten Modellbausteine nicht direkt durch ihre hierarchische Anordnung verbunden sind. Konflikte treten genau dann auf, wenn der Zugriff innerhalb eines Zweiges stattfindet. Das bedeutet, dass es direkten bzw. indirekten Einfluss auf Objekte einer anderen Hierarchieebene oder Detaillierungsstufe hat, wenn ein Untermodell im selben Teilbaum bearbeitet wird. Dieses pessimistische Verfahren kann dazu führen, dass mehr Instanzen von Teilmodellen gesperrt werden, als in Realität benötigt werden, erlaubt aber in der Praxis eine einfach zu implementierende Form der konfliktfreien Bearbeitung innerhalb einer Modellinstanz.

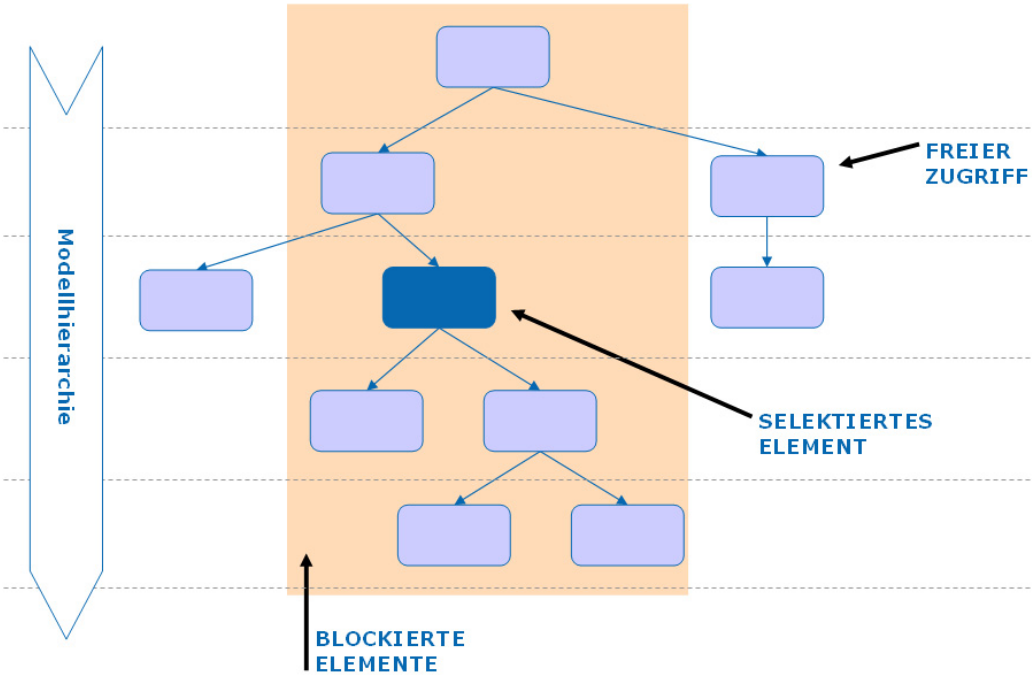


**Abbildung 27: Sperrmechanismus im Modellbaum**

Zur Sicherung von Modelländerungen existieren unterschiedliche Verfahren der Änderungskoordination. Das Erstellen einer neuen Version mittels eines Versionierungsverfahrens ist mit geringem Aufwand möglich. Zu jeder Versionsnummer muss der bearbeitende Anwender zugewiesen werden. Dadurch kann jedem Anwender die von ihm zuletzt bearbeitete Version geladen werden. Durch Überprüfung auf höhere Versionsnummern kann er zusätzlich darüber informiert werden, dass aktuellere Bearbeitungen des Simulationsmodells verfügbar sind. Jedes Simulationsmodell hat also eine eigene Zeithistorie, um die Änderungen bzgl. der unterschiedlichen Versionen nachvollziehen zu können. Auch damit kann letztendlich die Zusammenarbeit mehrerer Anwender in einem gemeinsamen Team verbessert werden, weil Änderungen an Modellbausteinen für alle anderen Teammitglieder erkennbar werden.

Sollen Experimente mit einem speziellen Simulationsmodell vorgenommen werden, bleibt das Klonen der effizienteste Weg. Die Versionierung erlaubt das Registrieren der Modellversion zu einem Experiment. Der Klon selbst existiert ausschließlich für die Dauer des Simulationslaufs, um die Datenmenge in der Simulationsdatenbank auf ein notwendiges Maß beschränken zu können.

Das Arbeiten mehrerer Anwender mit unterschiedlichen Kenntnisständen und Anwendungsbereichen in einer zentralen Anwendung erfordert eine Unterscheidung nach verschiedenen Anwendergruppen. Zur Sicherstellung der Qualität der Simulationsmodelle muss z.B. das ungewollte Verändern durch einen Laien verhindert werden können. Es wird deshalb ein Rechtemanagement definiert, um nach verschiedenen Gruppen und Bearbeitungsrechten differenzieren zu können. Das anwenderspezifische Rechte-management wird in der Modellbeschreibung des Simulationsmodells hinterlegt. Ziel ist es, erstellte Bibliotheken und Modelle vor Schaden durch falsche Nutzung und Unwissen zu bewahren und geistiges Eigentum schützen zu können. Jeder Anwender soll dazu Gruppen zugeteilt werden; die Zuteilung erfolgt anhand von Kenntnissen und Aufgaben. Es können die folgenden Gruppen unterschieden werden:

1. Programmierer (Individuum)
   Der Programmierer eines Simulationsmodells kann alle Modellbausteine administrieren, die er erstellt hat. Er hat zudem die Möglichkeit, die Zugriffsrechte für die Modellbausteine zu beschränken. Ein Programmierer kann verhindern, dass Administratoren Rechte ändern können. Einem Objekt können mehrere Anwender als Programmierer zugewiesen werden. Er besitzt alle Rechte für das erstellte Simulationsmodell
2. Anwender(Gruppe)
   Anwender haben keinen Zugriff auf die Programmierung der Methoden, sondern können ausschließlich vorhandene Bausteine aus Bibliotheken instanziieren, verknüpfen und verwenden.
3. Administratoren (Gruppe)
   Die Gruppe der Administratoren sind Anwender, die Rechte für Simulationsmodelle vergeben und ändern können. Sie können neue Untergruppen erstellen und weisen neue Anwender bestehenden oder neuen Gruppen zu. Sie müssen nicht unbedingt Nutzer der Simulationsanwendung („Simulationsexperten") sein.

Auf Ebene des Simulationsmodells können bei der Vergabe der Rechte vier Funktionsgruppen unterschieden werden. Dabei ist eine implizite Hierarchiebildung unter den verschiedenen Rechten abgebildet, so dass die individuellen Rechte stets höherwertig gegenüber den Gruppenrechten eines Anwenders sind.

- Modifizieren:
  Modifizieren erlaubt das vollständige Verändern des Simulationsmodells inklusive des grundlegenden Aufbaus und der modellierten Steuerungen und Ereignisse. Struktur und Standardwerte der Variablen können verändert werden.
- Verwenden:
  Verwenden erlaubt der jeweiligen Benutzergruppe das Einfügen, Benutzen und Löschen eines Modellbausteins als Instanz in anderen Simulationsmodellen. Dabei kann auf die öffentlichen Variablen des Bausteins zugegriffen werden, um diese für die jeweilige Instanz verändern zu können. Der Aufbau und die Methoden des Bausteins können nicht verändert werden. Das Modell kann in einem Simulationslauf gestartet werden.
- Löschen:
  Diese Funktion erlaubt das Löschen von Simulationsmodellen oder Modellbausteinen aus Bibliotheken oder der Simulationsdatenbank

- Administrieren:
  Diese Funktion erlaubt das Vergeben von Rechten. Üblicherweise haben das Administrationsrecht die Programmierer und die Administratoren des Simulationsmodells

Zusammenfassend können durch die Einführung eines anwenderspezifischen Rechtemanagements bereits im Vorfeld viele mögliche Konflikte aufgelöst werden, die beim gleichzeitigen oder zeitlich unabhängigen Bearbeiten eines Simulationsmodells durch mehrere Anwender entstehen können. Eine der grundlegenden Voraussetzungen für ein Mehrbenutzersystem wurde damit geschaffen. Alle unter Abschnitt 3.4.3 aufgezeigten Interaktionsmetaphern können in die obigen Gruppen einsortiert werden und Konflikte können durch die vorgestellten Verfahren aufgelöst werden.

Modellierung und Ausführung eines Simulationsexperimentes bleiben die Hauptaufgaben des Anwenders bei jeder Simulationsstudie. Es wurde beschrieben, wie der Prozess von Modellierung und Simulation unter Anwendung von Verfahren zur Konfliktvermeidung und eines Rechtemanagements in dem angestrebten Werkzeug umgesetzt werden soll. Alle grundlegenden Merkmale wurden für das zu entwickelnde Werkzeug festgelegt. Der folgende Abschnitt widmet sich der Entwicklung einer Modellbeschreibung, auf deren Basis Simulationsmodelle in dem zu entwickelnden Materialflusssimulator aufgebaut, strukturiert und abgelegt werden sollen und in denen das grundlegende Verhalten des Simulationsmodells so beschrieben wird, das eine Ausführung des Modells in einem Simulationsexperiment erlaubt wird.

## 5.2 Konzeption Modellbeschreibung

Ein wesentlicher Entwicklungsschritt während der Entwicklung des Materialflusssimulators ist die Formalisierung der zugrunde liegenden Modellbeschreibung. Auf Basis dieser objektorientierten Datenstruktur wird das Werkzeug konzipiert und seine benötigten Funktionen umgesetzt. Durch die Modellierungs- und Visualisierungskomponenten wird die zu entwickelnde Struktur von Simulationsmodellen umgesetzt und benutzerfreundlich dargestellt bzw. der Manipulation durch den Anwender zugeführt. Gemäß Abschnitt 4.2 wird in einem ersten Schritt eine allgemeine Modellbeschreibung generiert, die in weiteren Schritten auf die speziellen Anforderungen des Werkzeugs angepasst wird. Zunächst soll aber eine Beschreibung von Simulationsmodellen zur Vorwärtssimulation erstellt werden.

### 5.2.1 Vorwärtsgerichtete Materialflussmodelle

In Anlehnung an die in Abschnitt 3.2 beschrieben formalen Modellstrukturen wie Stellen/-Transitionsnetze unterscheidet der hier entwickelte Materialflusssimulator zwischen zwei Objektklassen: *Modellen* und *Token*. Modelle repräsentieren alle möglichen Formen von Bearbeitungs-, Lager- oder Transporteinrichtungen eines Fertigungsprozesses, *Token* entsprechen den Marken, die das Simulationsmodell zur Ausführungszeit dynamisch durchlaufen. Sie repräsentieren die beweglichen Elemente der realen Welt, beispielsweise Aufträge, die das System durchlaufen, Werker, Bauteile oder Paletten. Dadurch kann zwischen der formalen Struktur und dem dynamischen Verhalten des Simulationsmodells unterschieden werden. Token repräsentieren beliebige Objekte, die erst durch die Modellierung des Anwenders eine logische Bedeutung im Simulationsmodell erhalten. Sie

können neben Variablen auch weitere Token enthalten. Somit kann beispielsweise ein Gabelstapler eine Palette aufnehmen, die wieder mit Kisten beladen ist. Token bewegen sich zur Ausführungszeit des Simulationsmodells im Simulator durch das modellierte System. Die Reduzierung auf eine Objektklasse wurde hier auch deswegen bewusst vermieden, um die intuitive Unterscheidung zu unterstützen. Diese Grobklassifikation wird im weiteren Verlauf verfeinert, indem Modellbausteine Elemente enthalten, die für die Token nicht benötigt werden.

Im Gegensatz zu kommerziellen Materialflusssimulatoren (vgl. Abschnitt 3.2.3) sind die Basiselemente der hier beschriebenen Modellstruktur keine Modellbausteine im funktionalen Sinne, sondern nur Grundelemente, aus denen ein einzelnes Simulationsmodell zusammengesetzt werden kann. Dadurch lässt sich der Detaillierungsgrad des abgebildeten Modellbausteins durch den Anwender beliebig wählen. Neben diesen Basiselementen kann ein Simulationsmodell Submodelle (Instanzen anderer Simulationsmodelle) enthalten, um eine hierarchische Anordnung von Simulationsmodellen zu ermöglichen, die neben strukturellen Vorteilen für die Übersicht komplexer Modelle auch ein Umschalten zwischen den verschiedenen Detaillierungsgraden während der Simulationszeit potentiell erlauben. Damit ist eine wesentliche Vorraussetzung für die Integration der dynamischen Detaillierung nach [Muec05] geschaffen, die in Abschnitt 5.2.1.1 in die Modellbeschreibung integriert wird.

Auf der Hierarchieebene mir der höchsten modellierten Detaillierung wird ein Simulationsmodell durch folgende Basiselemente beschrieben, die innerhalb eines Simulationsmodells oder Modellbausteins eindeutig benannt sein müssen:

| Bezeichnung | Beschreibung |
|---|---|
| Variable | Attribute eines Modells |
| Event | Ereignisse eines Simulationsmodells, die das spezielle Verhalten genauer beschreiben |
| Link | logische Verknüpfungen zwischen Modellbausteinen |
| Channel | Eingangs- bzw. Ausgangskanäle als Schnittstellen eines Modells zu seinen Vorgängern/Nachfolgern im Modell |
| | |
| Submodel | Kein Basiselement. Simulationsmodelle komplexerer Bauart können ihrerseits wieder Modellbausteine als Instanzen anderer Simulationsmodelle oder Modellbausteine enthalten |

**Tabelle 4: Basiselemente eines Simulationsmodells**

*Channel* repräsentieren die Schnittstellen eines Modells oder eines Modellbausteins zu seinen logischen Vorgängern/Nachfolgern. Die Anzahl der Input- bzw. Output-Channel für ein Simulationsmodell ist durch den Modellierer wählbar. Es existieren nur Input- und Output-Channel. Jeder Channel, unabhängig von seiner genauen Spezifikation, kann zu einem Zeitpunkt nur maximal ein Token beinhalten. Input-Channel werden deshalb automatisch geschlossen, sobald ein Token den Channel erreicht und müssen für nachfolgende Token erst wieder geöffnet werden. Dem jeweiligen Output-Channel des Vorgängers ist es so lange nicht möglich ein Token an seinen Nachfolger zu senden, bis dieser wieder geöffnet wurde. Geschlossene Input-Channel können durch die Vorgängermodelle beobachtet werden und lösen dort ein spezielles Event aus, wenn der

Input-Channel wieder geöffnet wird. Channel bilden analog zu den Methoden objektorientiert programmierter Klassen die Schnittstellen eines Simulationsmodells zu seiner direkten Umwelt (vgl. Abschnitt 3.5.2.1).

Durch logische Verknüpfungen, hier *Links* benannt, werden die Channel, und damit die Simulationsmodelle bzw. Modellbausteine untereinander, verbunden. Token, die sich während des Simulationslaufs durch ein Simulationsmodell bewegen, werden gemäß der Beziehungen zwischen den Modellbausteinen versendet. Ein Token verlässt die Instanz eines Modellbausteins immer durch einen Output-Channel und tritt in dem nachfolgenden Simulationsmodell immer über einen Input-Channel in den folgenden Modellbaustein ein. Ein Output-Channel darf jeweils nur mit einem Link versehen werden. Eine Verzweigung, beispielsweise die Weiche einer Fördertechnik, muss über separate Output-Channel modelliert werden. Output-Channel können nur dann direkt miteinander verbunden werden, wenn sie eine einfache Weiterleitung des Token an die nächst höhere Hierarchieebene darstellen (vgl. Abbildung 28 rechts). Eintretende Token an einem Input-Channel lösen immer ein standardisiertes Ereignis (Input-Event) aus, in dem das weitere Verhalten innerhalb des Modellbausteins abgebildet werden kann. Um das Verhalten detailliert spezifizieren zu können, kann der Anwender sowohl auf von ihm definierte Verhaltensbeschreibungen in Ereignissen als auch auf eine beliebige Anzahl von Variablen zugreifen, die er in dem Modellbaustein anlegen und verwenden kann. Innerhalb eines Bausteins kann auf alle Ereignisse und Variablen dieses Modellsbausteins zugegriffen werden, da alle Basiselemente eindeutige Namen bzw. IDs besitzen. Input-Channel können genau dann direkt miteinander verbunden werden, wenn sie eine Weiterleitung von einer höheren Hierarchieebene zu einem Submodell darstellen (vgl. Abbildung 28 links). Die Strukturierung des Simulationsmodells über das beschriebene Konzept der Channel erlaubt eine strikte Modularisierung in einzelne Bausteine. Deren spezifische Funktionsweise wird nach dem Black-Box-Prinzip gekapselt und kann somit leichter adaptiert und verbessert werden, ohne die Funktionsweise der Gesamtmodelle zu gefährden.

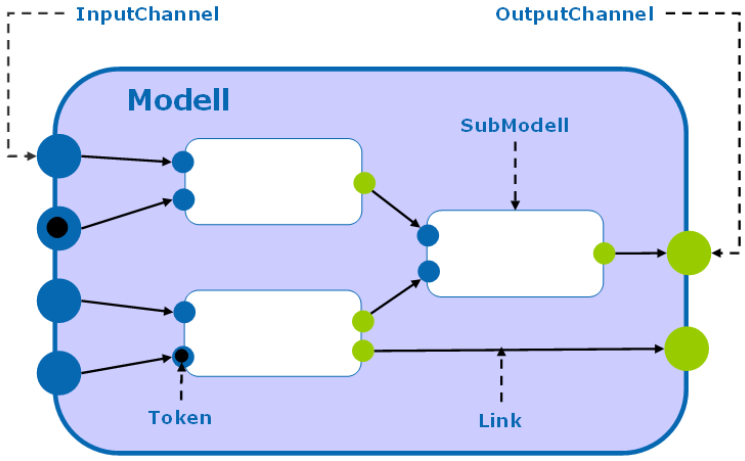


**Abbildung 28: Struktur eines Simulationsmodells**

Die Modellbausteine als einzelne Funktionsmodule können Elemente einer Bausteinbibliothek (im Folgenden: *Library*) sein, die zur Modellierung weiterer Simulationsmodelle verwendet werden können. Je umfangreicher diese hinterlegten Bibliotheken sind, umso weniger Simulationsmodelle, bzw. Modellbausteine muss der Anwender selbst anlegen. Im Idealfall kann ein Anwender ausschließlich unter Nutzung in

Bibliotheken vorhandener Modellbausteine ein Simulationsmodell erstellen und parametrieren. Bibliotheken können strukturiert abgelegt werden, damit benötigte Modellbausteine schneller identifiziert werden können. Die Instanz eines Modellbausteins wird im Simulationsmodell so gekapselt, dass nur diejenigen Basiselemente parametriert werden können, die vom Programmierer dafür vorgesehen wurden. Die Modellierung des spezifischen Bausteinverhaltens wird somit nicht automatisch öffentlich. Dieses Vorgehen entspricht der bereits unter Abschnitt 3.5.2.2 aufgezeigten Kapselungsstrategie der objektorientierten Programmierung und wendet die Vorteile der objektorientierten Paradigmen auf die Anwendung in der Modellierung von Simulationsmodellen an.

Modellbausteine, die innerhalb eines Simulationsmodells verwendet werden, besitzen eine Menge von Attributen, die im Folgenden aufgeführt werden. Auch hier wurde darauf geachtet, nur die Attribute in die Instanzen zu „verschieben", die die jeweilige Bausteininstanz von seiner Modellbaustein-Klasse unterscheiden.

| Bezeichnung | Beschreibung |
|---|---|
| ID | Eindeutiger Bezeichner |
| Name | Name |
| Mesh | Referenz auf ein 3D-Modell, das den Modellbaustein in den Modellierungs- und Visualisierungskomponenten darstellt |
| Meshscale | Skalierungsfaktor des 3D-Modells |
| X,Y,Z Size | Größe im Modell |
| Color | Farbe eines Modellbausteins |
| | |
| **Zusätzliche Attribute von Modellinstanzen innerhalb eines Simulationsmodells** | |
| X,Y,Z Scale | Größe im Modell |
| X, Y, Z Rotation | Rotation des Modellbausteins im Modell |
| Variable_value | Wert einer öffentlichen Variablen, wenn er sich vom Standardwert unterscheidet |
| srcid | Referenz auf eindeutigen Bezeichner des zugehörigen Modellbausteins (Klasse) |
| color | Farbe der Instanz, wenn sie sich von Vorbelegung unterscheidet |
| layer | Darstellungsebene |

**Tabelle 5: Attributliste eines Simulationsmodells**

Aus der bis hierhin entwickelten Modellbeschreibung, die das Einbinden von Submodellen als Instanzen vorhandener Modellbausteine erlaubt, folgt eine Gesamtbeschreibung eines Simulationsmodells als Liste aller im Modell verwendeten Modellbausteine. In dem auszuführenden Simulationsmodell, das im Rahmen dieser Gesamtbeschreibung als solches bei der Speicherung gekennzeichnet werden muss, werden diese Modellbausteine als Submodelle verwendet. Um die Konsistenz des Simulationsmodells zu erhalten, müssen die Modellierungskomponenten eine spezielle Funktion implementieren, die das Bilden von *Zirkelschlüssen* innerhalb des Modellierungsprozesses erkennt und vermeidet, damit Modellbausteine nicht rekursiv durch sich selbst beschrieben werden können. Abbildung 29 zeigt schematisch einen solchen Zirkelschluss, der zu einer ungültigen Modellbeschreibung führen würde.

Abbildung 29 zeigt, dass der zu modellierende Modellbaustein *A* durch Instanzen der Modellbausteine *B* und *C* beschrieben werden soll, so dass jeweils zwei Instanzen des Modellbausteins *B* und eine Instanz des Modellbausteins *C* den Ablauf innerhalb des Modellbausteins *A* beschreiben. Im Fall von Modellbaustein *B* ist das möglich, weil sich dieser aus Instanzen der Modellbausteine *D* und *E* zusammensetzt. Die Instanz *C_1* von Modellbaustein *C* führt jedoch zu einem unerlaubten Zirkelschluss, weil dieser durch Instanzen der Modellbausteine *F*, *G* und dem Modellbaustein *A* beschrieben wird. Durch die Instanz des Modellbausteins *C* würde für den Modellbaustein *A* eine rekursive Beschreibung existieren, die nicht ermöglicht werden darf, weil das Simulationsmodell sonst während der Ausführung in einen nicht auflösbaren Zustand gelangt.

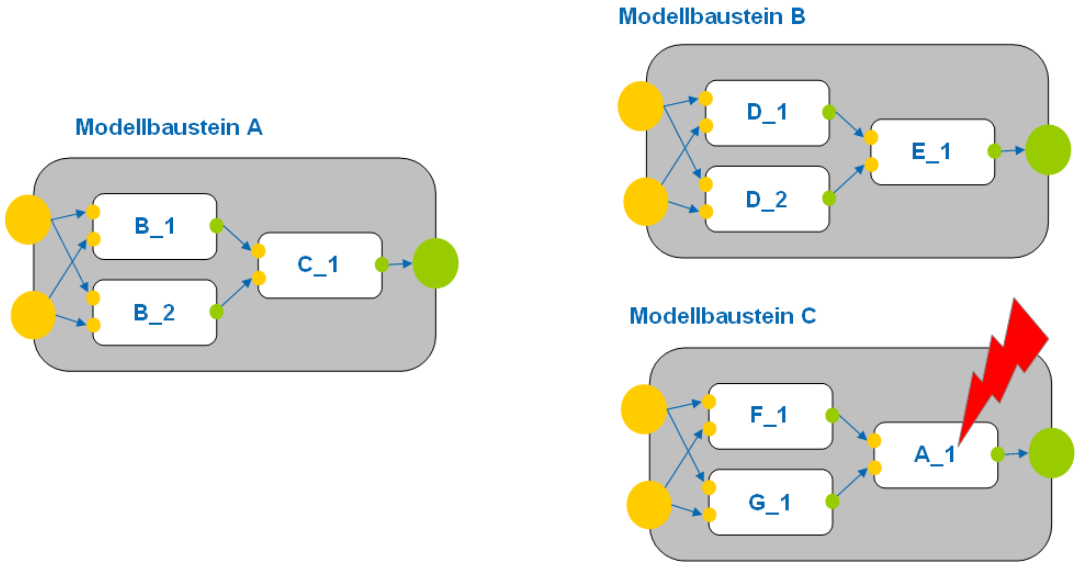


**Abbildung 29: unerlaubter Zirkelschluss in einem Simulationsmodell**

Durch die Implementierung einer Funktion zum Überprüfen von solchen Zirkelschlüssen kann dieses Problem bereits im Vorfeld umgangen werden, wodurch valide Modelle gefördert werden sollen. Das Problem tritt auf, weil die Modellbeschreibung so generisch gewählt wurde, dass der Anwender bei der Gestaltung seiner Modellbausteine im Gegensatz zu vorhandenen Simulationswerkzeugen den Detaillierungsgrad frei wählen soll. Das Erkennen von Zirkelschlüssen soll bereits in der Modellierungskomponente erfolgen, um den Anwender frühzeitig auf Modellierungsfehler hinzuweisen. Eine Erkennung im Simulator würde dadurch deutlich erschwert, dass objektorientierte Programmiersprachen die Definition mittels rekursiver Methoden prinzipiell ermöglichen und deshalb im Rahmen der Übersetzung in Programmcode den Fehler nicht erkennen würden. Das Simulationsmodell könnte zur Ausführungszeit somit in eine Endlosschleife laufen, deren Grund durch den Anwender nur schwer ersichtlich wäre.

Ein weiteres Basiselement zur Modellierung innerhalb eines Modellbausteins sind die *Variablen*. In ihnen können Token sowie alle für die Modellierung benötigten Werte zwischengespeichert werden. Beispielsweise kann die Bearbeitungszeit einer Maschine oder die Puffergröße innerhalb eines Modellbausteins in einer Variablen festgelegt werden. Aus den Events kann auf die Variablen zugegriffen werden, um ihren Wert zu verändern oder auszulesen. Jeder Variable kann der Modellierer verschiedene Attribute zuweisen. Sie bestimmen, ob die Variable protokolliert, von anderen Modellbausteinen

angesehen werden kann oder versteckt wird. Alle Variablen können mit Standardwerten parametriert werden und durch obere oder untere Schranken begrenzt werden. Die folgende Tabelle 6 zeigt alle Typen von Variablen, die in der Modellbeschreibung verwendet werden können. Sie entsprechen im Wesentlichen den primitiven und/oder höheren Datentypen, wie man sie aus Programmiersprachen kennt. Modellbausteine haben zu Beginn ihrer Erstellung neben den festgelegten Attributen keine Standardvariablen, wie es aus kommerziellen Simulatoren bekannt ist. Da der Detailgrad vom Anwender frei zu wählen ist, macht eine solche Vorbelegung innerhalb dieser Modellbeschreibung keinen Sinn.

| Typ | Wertebereich | Vorbelegungen |
|---|---|---|
| Integer | Natürlich Zahlen | Standardwert, obere und untere Schranke, Auswertungstyp |
| Long | Natürliche Zahlen | Standardwert, obere und untere Schranke, Auswertungstyp |
| Double | Reelle Zahlen | Standardwert, obere und untere Schranke, Nachkommastellen, Auswertungstyp |
| Float | Reelle Zahlen | Standardwert, obere und untere Schranke, Nachkommastellen, Auswertungstyp |
| Enum | Alle | Standardwert, Auswahlliste |
| ArrayList | Alle | Auswertungstyp |
| Token | Token | Keine |
| Boolean | True, False | Standardwert, , Auswertungstyp |
| String | Zeichenfolge | Standardwert, max. Länge |
| Time | Millisekunden | Standardwert |
| HaspMap | Objekte | Keine |
| Table | Objekte | Auswertungstyp |
| Random | Zufallszahlen verschiedener Verteilungen | Verteilung mit jeweiligen Parametern, Startwert, Auswertungstyp |

**Tabelle 6: Typen von Variablen der Modellbeschreibung**

Über ein spezielles Attribut kann durch den Anwender jeweils eingestellt werden, ob die Variable aus der Visualisierungskomponente heraus verändert, ausschließlich angezeigt oder grafisch ausgewertet wird. Alternativ kann sie in den Visualisierungskomponenten verborgen bleiben. Durch die auf Java basierende Zugriffsverwaltung kann der Anwender explizit die Interaktions-, Manipulations- und Auswertungsmöglichkeiten jedes Modellbausteins beeinflussen. Zur Verbesserung der grafischen Darstellung kann der Variablen zusätzlich ein bestimmter Auswertungstyp zugewiesen werden. Dadurch wird es möglich während eines Simulationslaufs in den verschiedenen Visualisierungskomponenten spezielle Darstellungsformen zu implementieren, um dynamisch die aktuellen Werte aus den Bausteininstanzen abzufragen und anwenderfreundlich darzustellen, bzw. deren Manipulation durch den Anwender zu erlauben. Die jeweilige Darstellungsform kann zwar typisiert werden, die eigentliche Implementierung der spezifischen Darstellungen erfolgt jedoch erst in den einzelnen Visualisierungs- und Auswertungsmodulen des Werkzeugs (vgl. Abschnitt 5.3.2). Für den Anwender ist dies eine der wesentlichen Funktionalitäten, die in den Visualisierungskomponenten benötigt wird, da neben der Animation des Simulationsverlaufs insbesondere die aktuellen Werte in den unterschiedlichen Bausteininstanzen für

die Modellverifikation und –validierung interessant sind. Für die verschiedenen Typen von Variablen sind die in Tabelle 7 aufgeführten Auswertungsmöglichkeiten vorgesehen. Sie orientieren sich an den vorhanden Analysemöglichkeiten bekannter Werkzeuge zur Datenauswertung und den Report-Darstellungen vorhandener Simulationswerkzeuge. Speziell in dem entwickelten Werkzeug ist es möglich, sich diese Auswertungen dynamisch und während der Ausführung eines Simulationslaufs anzeigen zu lassen. Auch dadurch soll der Validierungs- und Verifikationsprozess beschleunigt werden, weil Auswertungen direkt zur Laufzeit und nicht erst nach einem Simulationslauf zur Verfügung stehen.

| Name | Angewendeter Typ | Beschreibung |
|---|---|---|
| Display | Alle | Anzeige des aktuellen Wertes der Variablen |
| Signal | Boolean | Rot/Grün-Schalter oder ähnliche Darstellung |
| Trafficlight | Integer, Long, Double Float, Random | Aufwertung in Form einer Ampel |
| Gauge | Integer, Long, Double Float, Random | Auswertung in Form eines Füllstands oder eines Drehzahlmessers etc. |
| Meanvariance | Arraylist, Table | Statistische Auswertung durch Mittelwert, Minimum und Maximum |
| Timetable | Table | Auswertung in Form eines Diagramms |
| Histogramm | Arraylist, Table | Auswertung in Form eines Histogramms |

**Tabelle 7: Auswertemöglichkeiten der Variablen**

Die Zeitfortschaltung während der Ausführung eines Simulationsmodells geschieht sowohl bei der Vorwärts- als auch bei der Rückwärtssimulation standardmäßig ereignisorientiert. Der Simulator führt nacheinander die zeitlich geordneten Ereignisse aus, die in den verschiedenen Bausteininstanzen des auszuführenden Simulationsmodells ausgelöst werden. Das in diesen *Events* programmierte Verhalten ist durch den Modellierer angelegt worden. Events können andere Events innerhalb ihres Modellbausteins oder sich selbst aufrufen, indem sie eine neue Instanz eines Events terminieren und in der Ereignisliste des Simulators anmelden. Viele Events innerhalb eines Modellbausteins können einem bestimmten Channel zugewiesen werden. Darüber hinaus existiert pro Modellbaustein genau ein Init-Event, das in einer ersten Initialisierungsphase vor der eigentlichen Simulation ausgeführt wird. Es dient dazu, eine hinterlegte Vorbelegung des Modellbausteins aus Datenbank oder Dateien auszulesen oder Start-Events, beispielsweise bei Quellen eines Simulationsmodells, zu terminieren. Benutzerdefinierte Events (*User_Defined_Event*) können beliebig in Modellbausteinen hinterlegt werden, um komplexe Verhaltensweisen funktionsorientiert zu strukturieren oder überhaupt abbilden zu können. Sie werden durch sich selbst oder andere Events aufgerufen. Analog zum Init-Event existiert ein *Final-Event*, das beim Beenden des Simulationslaufs aufgerufen wird. Es dient dazu, in einem Modellbaustein gesammelte Daten des Simulationslaufs zu sichern, um eine Analyse zu späteren Zeitpunkten zu ermöglichen. Diese Vorgehensweise erlaubt die Beschleunigung der Berechnung des Simulationslaufs, da Datenbank- oder Dateisystemzugriffe während der Berechnung

gespart werden können, die eine schnelle Berechnung eventuell ausbremsen würden. Der Aufruf von Events in anderen Bausteininstanzen erfolgt ausschließlich mittels der Versendung von Token über die entsprechenden Channel-Schnittstellen der Modellbausteine. Die folgende Tabelle gibt eine Übersicht über alle standardisierten Events:

| Name | Beschreibung |
|---|---|
| Init | Wird bei Simulationsstart ausgeführt |
| Final | Wird beim Stop der Simulation ausgeführt |
| Input | Bei Eintritt eines Token in zugeordneten Input-Channel |
| Output | Bei Eintritt eines Token in zugeordneten Output-Channel |
| ReOpen | Beim Öffnen des nachfolgenden Input-Channel |

**Tabelle 8: Standardisierte Events**

Spezielle Methoden können in eigene Modellbausteine ausgelagert werden (beispielsweise für eine aggregierte Auswertung oder übergeordnete Steuerungen). Die Kommunikation erfolgt in diesem Fall über Token, die den Informationsfluss innerhalb des Simulationsmodells darstellen: den *InfoToken*. Sie unterscheiden sich nicht von den bisher bekannten Token, da diese ja explizit alle dynamischen Objekte repräsentieren können. Das objektorientierte Prinzip der Kapselung muss somit auch für übergeordnete Methoden nicht aufgeweicht werden.

#### 5.2.1.1 Dynamische Detaillierung von Simulationsmodellen

Zur Realisierung einer Echtzeitanalyse großer Fertigungssysteme in dem umzusetzenden Werkzeug soll die Methode nach [Muec05] in das Werkzeug integriert werden. Damit soll sich insbesondere die Simulation von komplexen Unternehmensnetzwerken oder Supply-Chain-Netzwerken einer Echtzeitanalyse nicht mehr entziehen. Alternativen zur Abarbeitung solch komplexer Modelle auf einem einzelnen Rechner ohne die Qualität beziehungsweise Granularität der Simulation zu verringern sind nicht bekannt. Vorraussetzung für die Methode ist ein Simulationsmodell in verschiedenen Detaillierungsstufen, eine Möglichkeit zur Umschaltung zwischen den Detaillierungsstufen und eine Stimulation der Umschaltvorgänge zur Laufzeit eines Simulationslaufes. Letzteres ist mit dem angestrebten System leicht zu realisieren, indem eine entsprechende Nachricht in das Kommunikationsprotokoll zwischen Simulatorkern und Visualisierungskomponente eingebunden wird (vgl. hierzu Abschnitt 5.2.2.3), die die Übergabe der aktuellen Benutzerposition aus der Visualisierung erlaubt. Je nach Benutzerposition in der virtuellen Umgebung kann der Detaillierungsgrad in der Simulation dann adaptiert werden.

Zur Abbildung mehrerer Detaillierungsebenen muss die Modellbeschreibung so angepasst werden, dass jede Bausteininstanz sowohl ihr höher detailliertes als auch das gröbere Modell bekannt ist. Dies ist durch den Anwender festzulegen und wird bei der Umschaltung zur Laufzeit vom Simulatorkern berücksichtigt. Die Attributliste eines Simulationsmodells muss dennoch nur um die in Tabelle 9 dargestellten Attribute erweitert werden, weil jede Detaillierungsstufe für sich wiederum ein ausführbares Simulationsmodell sein muss. Darüber hinaus beschränkt die Methode die Modellbeschreibung der Simulationsmodelle nicht, sondern betrachtet diese als gekapselte Einheit, die genannte Anforderungen erfüllen muss.

| Bezeichnung | Beschreibung |
|---|---|
| Moredetailed | Verweis auf ein Modell mit höherem Detaillierungsgrad |
| Lessdetailed | Verweis auf ein Modell mit niedrigerem Detaillierungsgrad |

**Tabelle 9: Zusätzliche Attribute eines Simulationsmodells für das MRS**

Da die gesamte Modellbeschreibung eines Simulationsmodells ohnehin aus einer Liste aller verwendeten Modellbausteine besteht, ist eine Erweiterung der Modellbeschreibung auf mehrere Detaillierungsstufen zunächst problemlos. Das als Hauptmodell beschriebene, auszuführende Simulationsmodell gibt dann implizit die Startdetaillierung des Gesamtmodells vor, die zu Beginn des Simulationslaufes vom Simulator überprüft und ggf. angepasst wird.

Das zeitliche Fortschreiten des Simulationslaufs basiert auf Events, die jeweils für einen bestimmten Zeitpunkt eingeplant werden und beim Erreichen der Simulationszeit dieses Zeitpunkts ausgeführt werden (vgl. Abschnitt 2.2). Zur Integration der dynamischen Detaillierung wird das Auslösen der Umschaltvorgänge mittels eines neu einzuführenden *Switch-Events* umgesetzt. Es wird vom Modellierer angelegt, spezifiziert und muss den Zustand des entsprechenden Modellbausteins auf sein höher bzw. weniger detailliertes Pendant abbilden und diesen in einen Zustand versetzen, der demjenigen Zustand der aktuellen Bausteininstanz bestmöglich entspricht. Dazu gehören Zahl und Position der Token, die aktuellen Werte der Variablen und die eingeplanten Events dieses Modells. Es muss klar definiert sein, wie diese auf das Ersatzmodell übertragen werden. Die zu entwickelnden Umschaltmethoden müssen effizient gestaltet werden, da die Umschaltung während der Berechnung des Simulationslaufs erfolgt. Vorschläge zur Umsetzung finden sich in [Muec05]. Die Art der Gestaltung des Switch-Events wurde nicht eingegrenzt, sondern kann vom Modellierer der entsprechenden Modellbausteine frei gewählt werden.

| Bezeichnung | Beschreibung |
|---|---|
| Switch_event | Ereignis zur Berechnung des Zustandsübergangs im Rahmen des MRS |

**Tabelle 10: Zusätzliches Ereignis zur Abbildung von Modellen im MRS**

Zur Beschreibung einer möglichen Implementierung eines Switch-Events wird auf [Muec05] verwiesen. Es soll aber darauf hingewiesen werden, dass der Modellierer wohl in den meisten Anwendungen zwischen den Fällen „höhere Detaillierung" und „niedrigere Detaillierung" unterscheiden muss. Da aber nicht zwangsläufig davon ausgegangen werden kann, dass sich in jeder Implementierung eine Unterscheidung zwischen den Umschaltungen ergibt, wurde auf die Einführung zweier unterschiedlicher Events bewusst verzichtet.

#### 5.2.1.2 Modellierung funktionsorientierter Fertigungssysteme

Im Rahmen der Modellierung funktionsorientierter Fertigungssysteme lassen sich zunächst alle Anforderungen mit der dargestellten Modellbeschreibung realisieren. Die verbesserte Unterstützung der speziellen Anforderungen von funktional gegliederten Fertigungssystemen (vgl. 4.2.1) führt zu der Fragestellung, wie der Anwender bei deren Modellierung weitergehend unterstützt werden kann. Wegen der speziellen Abläufe

innerhalb solcher Fertigungssysteme erfährt insbesondere die Abbildung von Transportwegen von Menschen oder Maschinen eine steigende Bedeutung. Durch die Integration von Layout- und Fertigungsprozessplanung (vgl. 5.1.3) wird ein layoutgerechtes Abbilden aller Transport- oder Bewegungswege durch das Werkzeug bereits erleichtert und erlaubt darüber hinaus eine realistischere Darstellung des gesamten Simulationsmodells. Dennoch ist die layoutgetreue Modellierung von Wegen in heutigen Werkzeugen einer der zeitaufwendigsten Schritte, eine der Hauptquellen für mögliche Fehler und wird infolgedessen kaum eingesetzt. Als Konsequenz werden die abzubildenden Transportwege meist abstrahiert und verfälschen so in den Animationen den realitätsnahen Eindruck in der dargestellten, virtuellen Umgebung. Die gröbere Modellierung und Simulation führt darüber hinaus oft zu ungenaueren Ergebnissen.

Für das intendierte Werkzeug soll die Wegberechnung deshalb weitestgehend automatisch erfolgen. Auf Basis des jeweiligen Simulationsmodell-Layouts wird durch ein spezielles Modul im Simulatorkern unter schwachen Restriktionen ein Bewegungsgraph berechnet, auf dem sich abgebildete Transporteinheiten oder Menschen bewegen. Die Steuerung und Verfügbarkeit dieser speziellen beweglichen Elemente (im Folgenden: SPMs) wird in Modellbausteinen abgebildet, die in der Modellierungsoberfläche in das Simulationsmodell eingebunden werden können. Mittels eines Algorithmus wird innerhalb des Moduls der kürzeste Weg für den jeweiligen Auftrag auf dem Graphen berechnet und unter Berücksichtigung eventueller Interaktionen zur Simulationszeit ausgeführt. Die Funktionsweise dieses Moduls zur Wegplanung, im Folgenden *Motion Planning* (MP), wird nachfolgend näher beschrieben.

Neben dieser speziellen, automatischen Form der Wegmodellierung soll in dem Werkzeug dennoch die explizite Modellierung von Wegen durch Modellbausteine weiterhin ermöglicht werden. Eine Abbildung von festen Wegstrukturen, wie sie in herkömmlichen Werkzeugen verwendet werden muss, kann damit realisiert werden. Das konservative Vorgehen verursacht im Vergleich jedoch einen deutlich erhöhten Modellierungsaufwand, weil alle Wegeelemente einzeln abgebildet werden müssen. Insbesondere vor dem Hintergrund eines effektiveren Einsatzes der Methode Ablaufsimulation ist die automatische Wegberechnung also zu bevorzugen.

Im Rahmen aller bekannten Softwarelösungen zur Ablaufsimulation ist eine solche Funktionalität der automatischen Wegfindung nicht bekannt., obwohl eine solche Aufgabe prinzipiell gut automatisiert werden kann.

**Ziel des Motion Planning**

Das Motion Planning berechnet die Bewegung aller dynamischen Einheiten innerhalb eines Simulationsmodells, die sich im abgebildeten Layout prinzipiell autark bewegen können und die sich in der Realität einen möglichst kurzen Weg durch das Fertigungssystem suchen (sie werden im Weiteren als *Shortest-Path-Mover* (SPM) bezeichnet). Im Gegensatz zu Token, die auf festgelegten Pfaden innerhalb eines Modellbausteins animiert werden, ist die Bewegung der SPMs bidirektional an den Simulator gekoppelt. Die Ergebnisse der Wegberechnung fließen direkt in die Simulation ein und dienen nicht nur der Animation.

Je nach Anwendung können unter SPMs beispielsweise Gabelstapler, Werker mit und ohne Hubwagen oder Trolleyzüge verstanden werden. Als Teilmodul des Simulators dient das Motion Planning der layoutgetreuen Berechnung aller nicht vom Modellierer festgelegten Bewegungen während der Ausführung eines Simulationslaufs. Die Kommunikation mit dem Simulationskern (Bewegungsanfragen) sowie den Visualisierungskomponenten findet über das in Abschnitt 5.2.2 definierte Nachrichtenformat statt. Das Motion Planning arbeitet zweidimensional, setzt also als erste Restriktion eine Unterscheidung verschiedener Fertigungsebenen voraus. Die verwendeten Algorithmen funktionieren unter geringen Einschränkungen an Simulationsmodell und Layout, die jedoch vor dem Hintergrund der Anwendung als akzeptabel erachtet werden können. Die Anwendung und die Ergebnisse werden durch die Restriktionen nicht so sehr verfälscht, dass der Einsatz des Motion Planning Modul den Einsatz der Ablaufsimulation konterkariert. Folgende Restriktionen sollen deshalb gelten:

- Komplexitätsreduktion durch Anfahrtspunkte: Es ist nicht nötig, jeden Punkt innerhalb einer Ebene anzufahren: nur vorher festgelegte Punkte an den einzelnen Bausteininstanzen müssen erreichbar sein (im Folgenden *dockingpoints*). Dies erlaubt die Berechnung der Bewegungsgraphen in der Initialisierungsphase des Simulationslaufs und dadurch die beschleunigte Beantwortung von Anfragen zur Laufzeit einer Simulation.
- Begrenzung der Layoutfläche des Simulationsmodells: Wird ein Simulationsmodell beispielsweise durch eine umschließende Halle begrenzt, muss diese gekennzeichnet werden. Dadurch ergibt sich eine natürliche Begrenzung des Bewegungsgraphen.
- Wege innerhalb geschlossener Formen sollen nicht auftreten können, beziehungsweise die entsprechenden Instanzen der SPMs im Simulationsmodell können diese Grenzen nicht überwinden.

Ein weiterer Unterschied zwischen Animation und Motion Planning besteht darin, dass Animationen nur der Darstellung einer Simulation in einer entsprechenden Visualisierungskomponente dienen. Das Motion Planning trägt demgegenüber essentiell zur Berechnung zukünftiger Ereignisse im Simulator bei, muss also unabhängig von der Visualisierung berechnet werden.

Zusätzlich sorgt das Motion Planning für eine kollisionsfreie Bewegung aller SPMs durch das Simulationsmodell während dessen Ausführung im Simulatorkern. Es ist integriertes Teilmodul des Simulatorkerns, soll aber im Rahmen der Implementierung als abgegrenztes Modul integriert werden, um durch Weiterentwicklungen und/oder alternative Wegberechnungen ausgetauscht werden zu können. Für den Simulationskernel ist die Ausführung eines Simulationsmodells bisher ein rein logischer Ablauf, in dem Token von einem Modellbaustein zu festgelegten Zeitpunkten an deren Nachfolger im Materialfluss weitergereicht werden. Die räumliche Anordnung der Modellbausteine ist deshalb zur Ausführung zunächst unerheblich. Sie werden vom Modellierer dem Simulationsmodell nur deshalb hinzugefügt, um eine optimale Visualisierung erreichen zu können. Im Rahmen der Umsetzung des Verfahrens zur Wegeberechnung wird das hinterlegte Layout zum Bestandteil der Simulation, da die Fahrtzeit der SPMs auf den Wegen abhängig von den verschiedenen Fabriklayouts ist.

Das Nachrichtenprotokoll zwischen dem Simulatorkern und dem Motion Planning Modul ist also derart zu gestalten, dass ein Token erst dann durch den Simulator an den Nachfolger innerhalb des Simulationsmodells weitergereicht wird, wenn sichergestellt ist, dass der Transport durch ein SPM das Ziel der Bewegungsanfrage auch erreicht hat. Mögliche Verzögerungen können hier durch Interaktionen des Anwenders oder durch direkte, gegenseitige Beeinflussung der SPMs entstehen. Sie werden ebenfalls erkannt und führen zu einer verzögerten Ankunft eines SPMs an seinem Zielort.

Aufgabe des Motion Planning ist es also, alle Bewegungsanfragen für SPMs vom Simulationskern entgegenzunehmen, deren Bewegungen auf dem zuvor berechneten Graphen zu planen und den Simulationskern über die Ankunft eines SPMs am Ziel zu informieren. Zusätzlich kann das Motion Planning auf Basis des errechneten Layouts auch für die Kollisionsabfrage der Avatare[23] verwendet werden. Damit diese nicht durch Maschinen, SPMs oder weitere Avatare hindurch laufen, die sich in der Szene befinden, müssen die Bewegungen eines Avatars regelmäßig vom Motion Planning überprüft und bestätigt werden. Das trägt in der Visualisierungskomponente, die eine solche Anbindung und Überprüfung implementiert, wesentlich zu einer immersiven Darstellung bei.

Die Animation der Token innerhalb eines Modellbausteins fällt explizit nicht in das Aufgabengebiet des Motion Planning. Deren Bewegung, beispielsweise eine Kiste auf einem Förderband, wird durch Animationsnachrichten entlang eines vorgegebenen Pfades visualisiert. Im Rahmen dieser Animation wird aus Komplexitäts- und damit Effizienzgründen keine Kollisionsabfrage durchgeführt. Eine entsprechend realistische Darstellung des Bausteinverhaltens muss also durch den Modellierer erfolgen, zum Beispiel durch die Verwendung mehrerer Animationsabschnitte.

**Funktionsweise des Motion Planning**

Das Motion Planning arbeitet zweidimensional, d.h. die Bewegung der SPMs wird nur in der XY-Ebene abgebildet und nur um die Z-Achse gedreht.[24] Zur Berechnung des Weggraphen und zur Vermeidung möglicher Kollisionen mit Modellierungsbausteinen genügt also eine zweidimensionale Projektion der Maschinen in die XY-Ebene (entspricht einer Draufsicht auf das Simulationsmodell). Diese Projektionen werden für jeden 3D-Repräsentanten eines Modellbausteins einmal generiert und als *Outline* in der Simulationsdatenbank zu dem 3D-Modell gesichert. Die SPMs selbst werden ebenfalls durch ihre spezifische Outline repräsentiert. Abbildung 30 verdeutlicht dies an einem Ausschnitt aus einem Simulationsmodell.

[23] Unter einem Avatar wird hier die Repräsentation eines Anwenders im Simulationsmodell verstanden, der sich während der Ausführung eines Simulationslaufs durch das Layout des Simulationsmodells bewegt und dadurch dynamisch das Fortschreiten der Simulation unter Umständen beeinflusst..

[24] Die Ausdehnung in XY-Richtung beschreibt die Ausdehnung nach Länge und Breite. Die jeweilige Höhe wird durch den Z-Vektor, oder die Z-Ebene bestimmt

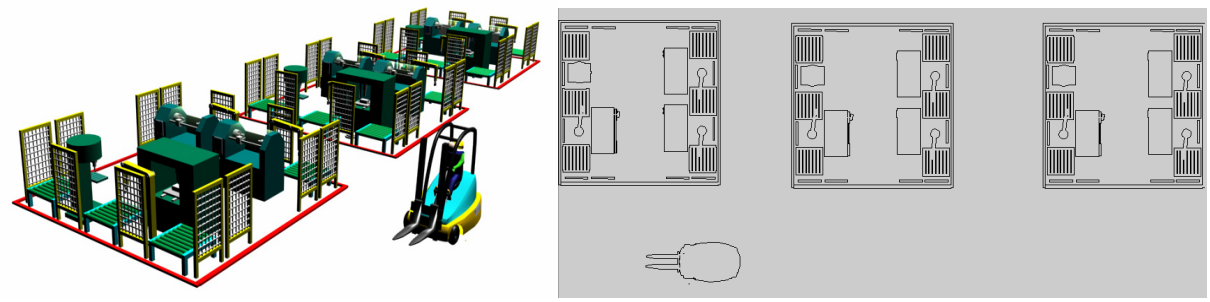

**Abbildung 30: 3D-Szene und entsprechende 2D-Projektion**

Obwohl das Motion Planning keine Bewegung in Z-Richtung, also in die Höhe plant, werden die unterschiedlichen Höhen der fahrenden SPMs dennoch durch eine höhenabhängige Projektion berücksichtigt. Für eine Kollisionsvermeidung zwischen einem SPM und den Repräsentanten der Modellbausteine ist es unerheblich, welche Ausdehnung die Maschinen oberhalb des SPM haben, weil er dort nicht mit ihnen kollidieren kann. Eine Projektion des gesamten Layouts des Simulationsmodells in die XY-Ebene kann eventuell ein falsches Abbild der Realität erzeugen und dazu führen, dass mögliche Wege eines bestimmten SPM-Typs nicht als solche erkannt werden. Durch einen Sicherheitsabstand fährt ein SPM in der entsprechenden dreidimensionalen Visualisierung nicht so nahe an den entsprechenden Repräsentanten des Modellbausteins heran, wie es technisch möglich wäre. Abbildung 31 versucht dies anhand einer Seitenansicht zu verdeutlichen.

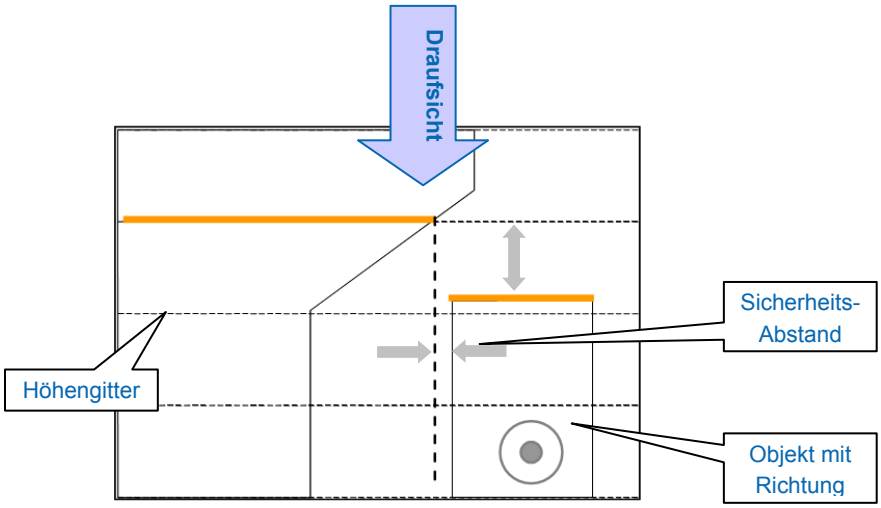


**Abbildung 31: Höhenabhängige Projektion in die XY-Ebene**

Die zu einem 3D-Repräsentanten erzeugten Outlines bestehen deshalb aus mehreren Schichten, wobei eine Schicht einem Schnitt auf einer bestimmten Höhe entspricht. Der 3D-Repräsentant wird dazu entlang der Z-Achse in mehrere Schichten unterteilt, deren einzelne wie überlagerte Projektionen in die XY-Ebene eine unterschiedliche Ausdehnung besitzen. Alle einzelnen Projektionen werden zusammen mit den zugehörigen Höhenintervallen als *Layer* gespeichert. Die Summe aller vorhandenen Layer eines 3D-Repräsentanten bildet die eigentliche Outline des 3D-Modells. Die SPMs besitzen ebenfalls jeweils eine Outline, die für die Berechnung verwendet werden kann. Zur Kollisionsvermeidung mit den statischen 3D-Repräsentanten der Bausteininstanzen werden aus deren Outline nur diejenigen Layer betrachtet, die im Höhenintervall der

Outline des entsprechenden SPMs liegen. Nachdem die Umrisse der Schichten vereinigt wurden, kann wieder auf eine zweidimensionale Ebene reduziert werden (vgl. Abbildung 30).

**Berechnung des Wegegraphen**

In der Initialisierungsphase des Motion Planning wird für jeden unterschiedlichen SPM-Typ jeweils ein Graph berechnet, auf dem die SPMs während des Simulationslaufs bewegt werden. Der einem SPM zugehörige Graph enthält alle Informationen über Positionen und Blickrichtungen des SPMs sowie die entsprechenden Outlines. Die Vorberechnung des Graphen in der Initialisierungsphase soll eine effiziente Realisierung der Kollisionserkennung zur Laufzeit ermöglichen. Die Kollisionen unter den SPMs soll die Berechnung der Simulation möglichst wenig ausbremsen. Für die Initialisierungsphase muss dafür ein höherer Aufwand in Kauf genommen werden. Auch vor dem Hintergrund der Berechung einer Kollisionsabfrage mit mehreren Avataren erscheint eine Verlagerung aufwendigerer Berechungen in die Initialisierungsphase sinnvoll. Neben der Beschleunigung der Berechnung zur Ausführungszeit eines Simulationslaufs unterstützt dieses Vorgehen die Komplexitätsreduktion der Simulationsmodelle. Durch die Möglichkeit zur effizienten Speicherung einmal erstellter Graphen und deren Neuberechnung nur bei veränderten SPM-Typen oder einem veränderten Layout, kann in der Praxis dieser Mehraufwand fast vollständig kompensiert werden.

Jeder Knoten des Graphen repräsentiert eine mögliche Position des SPMs im Simulationsmodell. Dazu werden im Knoten Position und Ausrichtung des SPMs hinterlegt. Um die Kollisionsvermeidung effizient zu ermöglichen, speichert jeder Knoten auch die entsprechende Outline, welche Position und Ausrichtung des SPMs im Simulationsmodell repräsentiert. Basierend auf der Annahme, dass zwei entgegen gesetzt fahrende SPMs sich an den meisten Stellen passieren können, wird ein Rechtsverkehr um alle Hindernisse eingeführt, wobei Hindernisse Maschinen oder auch Teile von Maschinen sind. Jede Kante im Graphen wird dazu als Vorwärtskante (gewichtet mit der euklidischen Distanz der Knoten in der Ebene) und als Rückwärtskante (gewichtet mit der Distanz multipliziert mit einem konstanten Faktor) bewertet. Damit kann explizit die Gewichtung von Vorwärtsbewegung zu Rückwärtsbewegung für die einzelnen SPM-Typen festgelegt werden.

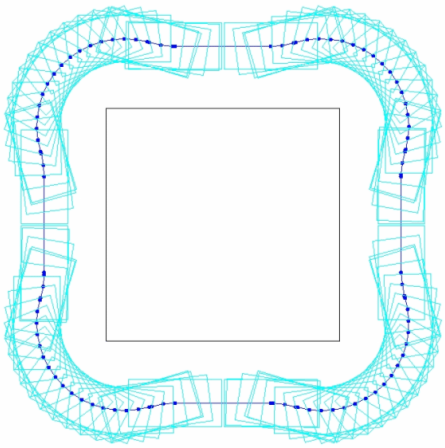

**Abbildung 32: Interpolation von Kurven durch feingranulare Auflösung**

Es wird davon ausgegangen, dass die Interpolation der SPM-Outline entlang einer Geraden (und dadurch ohne Änderung des Blickwinkels) wenig rechenintensiv ist. Rechenintensivere Interpolation von Kurven zur Ausführungszeit eines Simulationslaufs werden vermieden, indem vorhandene Kurven im Graphen so feingranular aufgelöst werden, dass die gespeicherten Outlines des SPMs sowohl zur Kollisionsvermeidung als auch zur flüssigen Animation verwendet werden können. Das führt zu einer Erhöhung der Knoten des abgebildeten Weggraphen, wodurch die Suche nach Wegen verzögert wird. Diese Einschränkung kann aber an dieser Stelle in Kauf genommen werden, wenn das entwickelte Verfahren dennoch schnell arbeitet und die Berechnung der Simulation nicht übermäßig bremst. Abbildung 32 verdeutlicht diese Idee. Die eigentliche Berechung des Graphen erfolgt schrittweise:

1. Berechnung von umschließenden Graphen (im Folgenden: *Boundaries*), um die Repräsentanten der Bausteininstanzen im Simulationsmodell; Identifikation von *Connectionpoints* als Verknüpfungspunkte zwischen existierenden Boundaries
2. Berechnung und Erstellen von Verbindungen zwischen den Connectionpoints und damit zwischen den Boundaries, die die 3D-Repräsentanten im Simulationsmodell umschließen
3. Bereinigung des Graphen durch Löschen überflüssiger Knoten und Kanten mit dem Ziel, die Wegberechnung zu beschleunigen
4. Zuordnung festgelegter Dockingpoints der einzelnen Bausteininstanzen zu existierenden Knoten des Graphen, wobei der Abstand zwischen Dockingpoint und Knoten auf dem Graphen minimiert wird.

In der ersten Berechnungsphase werden Knoten und Kanten um die 3D-Repräsentanten der Modellbausteine gelegt, so dass diese durch einen Graphen umhüllt sind (vgl. Abbildung 30). Dieser Rahmen wird im Folgenden als *Boundary* bezeichnet. Für jeden Graphen werden so genannte Connectionpoints identifiziert und gespeichert, die als Eckpunkte oder Andockpunkte verstanden werden können. Im folgenden Berechnungsschritt werden Verbindungen zwischen den berechneten Connectionpoints berechnet. Eine Auswahl der gefundenen Verbindungen wird schließlich im Graphen gespeichert. Abbildung 33 zeigt einen Beispielgraph nach dem Hinzufügen dieser Verbindungen.

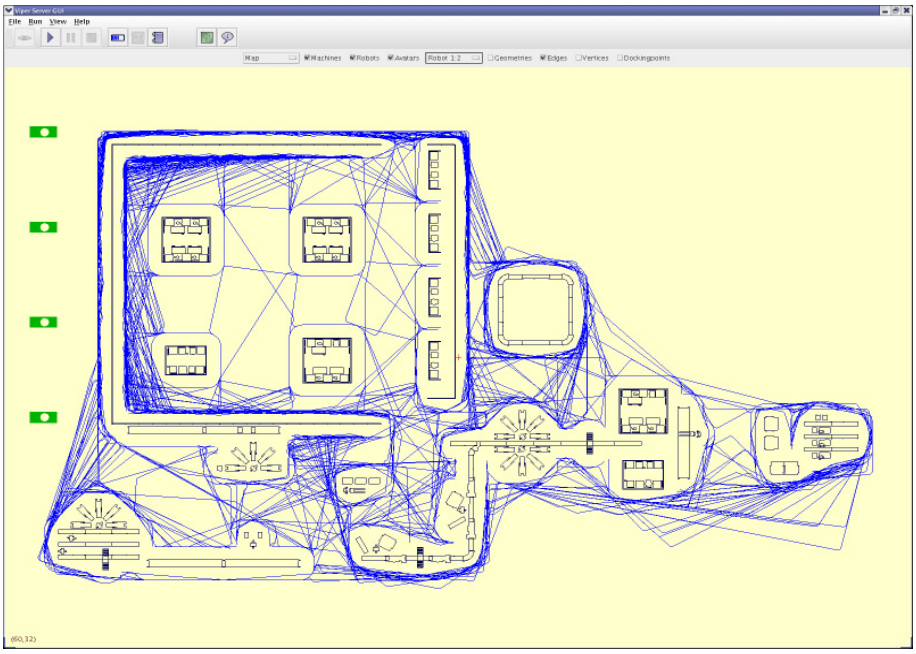

**Abbildung 33: Graph des Motion Planning**

Die dritte Berechnungsphase dient der Reduzierung der Komplexität des Graphen. Sie löscht nicht benötigte Knoten und Kanten, wobei der starke Zusammenhang[25] des resultierenden Graphen sichergestellt bleibt. Damit existiert mindestens eine Lösung für jede gültige Wegeberechnung und es kann immer ein Weg zwischen allen vorhandenen Dockingpoints gefunden werden. Überflüssige Knoten und Kanten können z.B. bei der Berechnung der Boundaries entstehen, wenn Modellbausteine so nah beieinander platziert sind, dass der SPM nicht zwischen ihnen hindurch fahren kann. In der abschließenden Berechnungsphase werden Assoziationen zwischen den möglichen Dockingpoints und den nächstliegenden Knoten des Graphen hergestellt, so dass Anfragen der Form „Kürzester Weg zwischen den Dockingpoints A und B" auf Graphberechnungen der Form „Kürzester Weg zwischen den Knoten v_A und v_B" abgebildet werden können.

**Bewegungsanfragen**

Die eigentliche Logik einer SPM-Steuerung wird durch einen Modellbaustein im Simulationsmodell abgebildet. Diese Steuerungslogik schickt eine Bewegungsanfrage an das Motion Planning Modul und wartet mit der Weiterleitung des Token an den SPM, bis das Motion Planning die Ankunft des SPM am Startpunkt mitteilt. Es existieren zwei verschiedene Anfragetypen, weil SPMs optional Pools zugeordnet werden können. Sie bilden eine Gruppierung gleicher SPMs, damit auch eigentlich gleiche SPM-Typen beispielsweise hinsichtlich eines beschränkten Einsatzortes unterschieden werden können (etwa: Gruppe A nur Halle 1, Gruppe B nur Halle 2). Die Modellierung einer komplexen

[25] Der Grad des Zusammenhangs bzw. die Konnektivität von Graphen bedeutet, dass Wege zwischen mindestens zwei Knoten im Graphen bestehen. Ein gerichteter Graph G=(V = Knoten, E = Kanten) heißt zusammenhängend von einem Knoten v aus, falls es zu jedem Knoten w aus V einen gerichteten Weg in G gibt, mit v als Startknoten und w als Endknoten. G heißt stark zusammenhängend, falls G von jedem Knoten v aus V zusammenhängend ist (vgl. [Corm90]).

Steuerung wird durch die Anwendung des Pool-Konzeptes weiter vereinfacht, weil implizit der nächste freie SPM innerhalb des Pools vom Motion Planning beauftragt wird. Durch die beiden Anfragemöglichkeiten kann ein bestimmter SPM sowohl anhand seiner eindeutigen ID beauftragt werden oder aber ein beliebiger SPM eines bestimmten SPM-Pools anhand dessen Pool-ID. Im zweiten Fall wählt das Motion Planning automatisch den nächsten freien SPM aus dem angeforderten Pool aus. Sobald eine Bewegungsanfrage beim Motion Planning eintrifft, wird für den entsprechenden SPM, bzw. alle SPMs des Pools, der Dijkstra-Algorithmus[26] auf dem zugehörigen Bewegungsgraph gestartet, um den kürzesten Weg zu identifizieren. Anschließend wird die minimal benötigte Zeit bis zur geschätzten Ankunft an die Simulation zurückgeschickt. Diese Ankunft ist im Voraus nicht genau bestimmbar, da durch mögliche Kollisionen mit Avataren oder SPMs Verzögerungen auf dem Transportweg entstehen können. Zu diesem frühesten möglichen Zeitpunkt fragt der Simulator beim Motion Planning nach der Ankunft des SPM. Als Antwort wird entweder eine neue Schätzung oder im Erfolgsfall eine Bestätigung der Ankunft verschickt.

**Kollisionsvermeidung**

Das Motion Planning ist sowohl für die Kollisionserkennung der Avatare untereinander, der Avatare mit den SPMs, als auch zwischen den SPMs verantwortlich. Klammert man eine Kollision mit den eventuell angeschlossenen Avataren zunächst aus, geschieht die Vermeidung von Kollisionen zwischen SPMs und den Repräsentanten der Modellbausteine bereits implizit durch die Berechnung des Bewegungsgraphen. Die während der Ausführung eines Simulationslaufs auf dem Graphen berechneten Pfade als Ergebnisse der Weganfragen können also keine Kollisionen mit den Modellbausteinen erzeugen. Um die Kollision der SPMs untereinander zu vermeiden, wird deren Bewegung in einem *Motion Schedule* festgeschrieben. Dieser enthält in fixen Zeitinkrementen die Positionen aller SPMs inklusive der sie beschreibenden Outlines. Der Abstand der Zeitinkremente ist so klein gewählt (z.B. 200ms), dass der Motion Schedule zur Erkennung von Kollisionen genutzt werden kann. Aus Effizienzgründen wird der Motion Schedule in größeren zeitlichen Intervallen vorausberechnet (z.B. alle 10s). Als Ergebnis entsteht eine Abfolge von Zeit-Ebenen mit den Positionen der jeweiligen SPMs zu jedem berechneten Zeitpunkt. Die Zeit-Ebenen werden aufsteigend geordnet und mit den SPM-Positionen gefüllt. Dazu werden die berechneten Pfade, auf denen sich die SPMs bewegen sollen, Schicht für Schicht miteinander verglichen, so dass keine Kollisionen entstehen. Mögliche Kollisionen werden zunächst durch Warten aufgelöst. Ein SPM wartet, bis der kollidierende SPM seine Route fortgesetzt hat, bevor es den Weg auf seinem Pfad fortsetzt. Zur Animation werden die berechneten Roboterpositionen in regelmäßigen Abständen vom Motion Planning an die angeschlossenen Visualisierungskomponenten verschickt. Um eine flüssige Animation zu erreichen werden aus den gespeicherten Pfaden, auf denen sich die SPMs bewegen, Zwischenschritte extrahiert, da die Auflösung des Motion Schedule für eine flüssige Animation nicht ausreicht. Bevor die SPM-Positionen verschickt werden, wird zusätzlich überprüft, ob Kollisionen mit den Avatarpositionen existieren. Sollte eine solche Kollision auftreten, wird der entsprechende SPM angehalten und als blockiert markiert. Der Motion Schedule muss dann neu berechnet werden, da aus dieser Blockade neue Kollisionen entstehen können. Unter

[26] Eine genauere Beschreibung des Dijkstra-Algorithmus findet sich bei [SuMe01].

Verwendung dieses Verfahrens wird also sowohl die Kollision der SPMs untereinander, wie auch mögliche Kollisionen mit sich bewegenden Avataren vermieden.

Die Kollisionsvermeidung der Avatare soll auf einem leicht abgeänderten Prinzip basieren. Basis der Wege aber vor allem der Graphberechung ist das höhenabhängige zweidimensionale Layout des Simulationsmodells. Um die Kollisionserkennung der Avatare untereinander sowie der Avatare mit den 3D-Repräsentanten der Modellbausteine realisieren zu können, muss die angeschlossene Visualisierungs-komponente dem Motion Planning einen Wegabschnitt schicken, entlang dem sich der jeweilige Avatar bewegen will. Das Motion Planning berechnet daraufhin die letzte Position auf dem übermittelten Weg, die keine Kollision mit 3D-Repräsentanten oder anderen Avataren ergibt. Als Konsequenz muss der Avatar eventuell durch die Visualisierungskomponente zurückgesetzt werden, wodurch in der Anzeige ein Ruckeln entstehen kann. Die Auswirkungen dieses Effektes hängen aber stark von der Frequenz der Übermittlung des Wegabschnitts zusammen, die die Visualisierungskomponente realisiert. Dieses Verfahren basiert auf der Annahme, dass die Bewegung eines Avatars durch die Szene nur schwer im Vorhinein abgeschätzt werden kann. Innerhalb der Aufgabenstellung ist dieser möglichst hohe Freiheitsgrad für den Anwender gewollt, damit dieser sich möglichst realitätsnah und immersiv durch die Szene bewegen kann. Ist darüber hinaus die Frequenz der Übermittlung des Avatar-Wegs hoch genug, wird der Avatar durch das System so rechtzeitig gebremst, dass dieser „Ruckel-Effekt" kaum störend wahrgenommen wird.

Die eingesetzte Kollisionsvermeidung dient der Realisierung unterschiedlicher Direktiven. Während die Kollisionsvermeidung der SPMs untereinander hauptsächlich darauf ausgerichtet ist, die Wegeplanung so realistisch wie möglich abzubilden und erst als Nebeneffekt eine saubere Visualisierung in den entsprechenden Komponenten ermöglicht (Gabelstapler fahren dann beispielsweise nicht durcheinander durch), ist genau diese präzise Visualisierung ein nützliches Instrument zur Erfüllung der Anforderung nach bestmöglicher Immersion. Der Anwender wird gezwungen, auch in der virtuellen Darstellung des Simulationsmodells und damit im abgebildeten System nur diejenigen Wege zu benutzen, die ihm auch im realen System zur Verfügung ständen. Durch den direkten Effekt seiner virtuellen Präsenz im Simulationsmodell (Gabelstapler halten vor ihm an) führt das Motion Planning dadurch zu einer Steigerung der Immersion in der virtuellen Umgebung. Das der Anwender als Konsequenz das Fortschreiten der Simulation während dieses spezifischen Simulationslaufs beeinflusst, ist ein auf Grund der stochastischen Auslegung von Simulationsmodellen zu vernachlässigender Effekt, bietet darüber hinaus sogar eine weitere unmittelbare Interaktionsmöglichkeit in der virtuellen Umgebung und unterstützt so die an das Werkzeug gestellten Anforderungen. Die Echtzeit-Simulation wird hauptsächlich in der Phase der Modellvalidierung und Verifikation eingesetzt. Die Berechnung statistisch signifikanter Simulationsläufe erfolgt anschließend ohne Visualisierungskomponenten und in einer streng ereignisorientierten bzw. analogen schnellstmöglichen Berechnung. Die zur Modellvalidierung, Modellverifikation oder Vorführung eingesetzte dreidimensionale Visualisierung eines einzelnen Simulationslaufes unterliegt weiteren stochastischen Einflüssen und ist somit nicht notwendigerweise repräsentativ für das dynamische Verhalten des abgebildeten Systems. Im Rahmen der Realisierung ist darauf zu achten, diese Interaktionsmöglichkeit zumindest optional ausschalten zu können, um dem Anwender optional eine bekannte

Visualisierungsform anzubieten. Alternativ können Verfahren implementiert werden, wie sie unter Abschnitt 3.4.1.1 bereits aufgezeigt wurden.

**Benutzerführung zu signifikanten Prozesspunkten**
Bewegt sich der Anwender durch die dreidimensionale Szene wird das Maß der Immersion für ihn für alle diejenigen Abschnitte der Szene erhöht, die in seinem Wahrnehmungsbereich liegen. Bei großen Szenen steigt dem gegenüber derjenige Anteil der Szene, der dem Anwender verborgen bleibt (beispielsweise in seinem Rücken, hinter einer Wand, etc.). Um dennoch eine optimale Analyse des visualisierten Simulationsmodells zu ermöglichen, muss der Anwender auf diejenigen signifikanten Prozesse hingewiesen werden, die nicht in seinem Sichtfeld liegen, beispielsweise in Form einer Liste aller Bausteinsinstanzen, die nach den aktuellen *Signifikanzwerten* sortiert ist. Voraussetzung hierfür ist die Berechnung und Belegung jeder Bausteininstanz des Simulationsmodells mit einem objektspezifischen Signifikanzwert und einer im Baustein hinterlegten Methode zu dessen Berechnung während der Ausführung der Simulation. Während der Simulation wird der Anwender auf kritische Prozesse innerhalb der Simulation aufmerksam und kann seine Aufmerksamkeit umlenken. Als Navigationshilfe soll ihm der kürzeste Weg zu einem selektierten Baustein in die Szene eingeblendet werden und der selektierte Modellbaustein entsprechend markiert werden, um ihn innerhalb des Simulationsmodells schneller identifizieren zu können. Hier wirken die Anforderungen nach einer hohen Immersion für die Aufgabenerfüllung des Anwenders eher hinderlich. Vor der Hintergrund einer realitätsnahen Visualisierung muss das an dieser Stelle allerdings in Kauf genommen werden. Während der Implementierungsphase sollten alternative Visualisierungen ergänzend berücksichtigt werden, um diese Fragestellung mit einer besser angemessenen Darstellungsform zu bearbeiten.

Grundlage dieser Navigationshilfe muss ein auf dem zweidimensionalen Layout berechneter Bewegungsgraph sein, wobei das entwickelte Verfahren erneut zum Einsatz kommt. Der Avatar des Anwenders entspricht einem speziellen SPM. Der kürzeste Weg muss für den Anwender in die virtuelle Umgebung eingeblendet werden, beispielsweise in Form von Pfeilen, die auf dem Boden angeordnet werden. Verschiedene Realisierungen sind an dieser Stelle vorstellbar, von der einmaligen Berechnung und Darstellung des Weges ausgehend von der aktuellen Position des Avatars bis zur dynamischen Nachführung und Adaption des Weges nach Avatarbewegungen innerhalb der Szene. Zumindest optional soll eine Funktion vorgehalten werden, die das direkte „Beamen" zum nächst möglichen Punkt in der Umgebung des selektierten Modellbausteins erlaubt (vgl. hierzu auch Abschnitt 3.4.2).

**Erweiterung der Modellbeschreibung zur Integration des Motion Planning**
Die der Integration des Motion Planning Moduls in den Simulationskernel zur dynamischen Berechnung der Wege für alle Typen von SPMs erfordert eine Erweiterung der vorhandenen Modellbeschreibung und eine Erweiterung des Nachrichtenprotokolls (vgl. Abschnitt 5.2.2.3). Tabelle 11 listet die zusätzlich benötigten Attribute für einen Modellbaustein auf, die sich aus dem Motion Planning ergeben.

| Bezeichnung | Beschreibung |
|---|---|
| Significance | Signifikanzwert für jede Modellinstanz |
| simulationBoundary | Markierung für reine Layoutobjekte |

| | |
|---|---|
| Dockingpoint | Relative Anfahrtspunkte der SPMs für jeden Modellbaustein |

**Tabelle 11: Zusätzliche Attribute zur Integration des Motion Planning**

Durch die Erweiterung um ein spezielles Modul zur Wegplanung eignet sich das hier entwickelte Werkzeug besser für die Modellierung und Simulation von funktionsorientierten Fertigungsprozessen, weil der Aufwand zur Modellierung der benötigten Wege und deren Bearbeitungsdauer während der Simulation durch entsprechende Motion Planning Bausteine in Bibliotheken auf ein Minimum reduziert werden kann. Darüber hinaus wird ein präziseres und durch den Anwender potentiell manipulierbares Ergebnis erlaubt. Die Nutzung des hier entwickelten Verfahrens ist aber nicht zwingend vorgegeben; die Modellbeschreibung erlaubt ebenso die Beschreibung über genau festgelegte Wege, wie sie in vorhandenen Materialflusssimulatoren zum Einsatz kommen. Da die Graphenberechung für das Motion Planning ein arbeitsintensiver Prozess ist, müssen während der Implementierungsphase des Werkzeuges entsprechende Methoden zur Reduzierung des Aufwands berücksichtigt werden. Beispielsweise muss ein zu einem Layout gehörender Graph mit dem Simulationsmodell in der Simulationsdatenbank gespeichert werden können, um das Laden eines Simulationsmodelllayouts inklusive seines Bewegungsgraphen im Simulatorkern zu beschleunigen. Voraussetzung hierfür ist, dass sich auch alle anderen für das Motion Planning relevanten Informationen des Simulationsmodells nicht geändert haben. So könnte der Austausch eines 3D-Repräsentanten für einen Pool von SPMs Auswirkungen auf die zu benutzenden Wege und damit den zugrunde liegenden Bewegungsgraphen haben. Zur Identifikation von Änderungen am Simulationsmodell stehen neben der Versionierung weitere Verfahren zur Verfügung. Ein vorhandener Weggraph darf in jedem Fall nur dann aus der Simulationsdatenbank geladen werden, wenn es keine Änderungen am Simulationsmodell gegeben hat, die für das Motion Planning Konsequenzen haben

Neben den eingepflegten Erweiterungen der Modellbeschreibung muss im weiteren Verlauf der Entwicklung auch das vorhandene Nachrichtenformat erweitert werden, um eine effektive Kommunikation zwischen dem Simulatorkern und dem Motion Planning Modul zu ermöglichen. Alle Details der benötigten Erweiterungen sowie die Gestaltung des Nachrichtenablaufs, können aber unter Abschnitt 5.2.2.3 gefunden werden.

Ein weiterer wesentlicher Punkt der Aufgabenstellung ist die Modellierung und Simulation in einem Multitasking-Betrieb, die das Modellieren und Simulieren mehrerer Anwender auf Basis eines einzelnen Modells erlaubt. Der folgende Abschnitt beleuchtet dieses Themenfeld genauer hinsichtlich zusätzlich benötigter Erweiterungen für die Modellbeschreibung.

#### 5.2.1.3 Multitasking Modellierung und Simulation

Um in dem Werkzeug den Mehrbenutzerbetrieb sowohl im Bereich der Modellierung, als auch in der Simulation umzusetzen, wurden in 5.1.6 die Methoden *Locking*, *Cloning* und *Versionierung* identifiziert. Darüber hinaus wurde dort ein Rechtemanagement entwickelt, das ebenfalls berücksichtigt werden soll. Dieser Abschnitt beschreibt die notwendigen Erweiterungen für die Modellbeschreibung, um die identifizierten Methoden umsetzen zu können.

Um das Locking überhaupt zu erlauben, muss es für jede Bausteininstanz möglich sein, einem einzigen Anwender durch ein entsprechendes Attribut den exklusiven Zugriff auf diese Instanz zu garantieren. Durch die Möglichkeit zur hierarchischen Modellierung müssen zumindest alle untergeordneten Bausteininstanzen während der Bearbeitung gesperrt werden (vgl. Abbildung 27). Alle Modellbausteine, die dem aktuell gesperrten Bausteinobjekt übergeordnet sind, sollten ebenfalls entsprechend markiert werden. Verschiedene Zustandsausprägungen des Attributs regeln diese verschiedenartigen Zugriffsmöglichkeiten (vgl. Abschnitt 3.5.4.1). Das Locking-Attribut muss nicht in der Modellbeschreibung des Simulationsmodells gespeichert werden, sondern wird nur während der Bearbeitung oder Ausführung eines Simulationsmodells benötigt. Dennoch soll es an dieser Stelle in die Modellbeschreibung mit aufgenommen werden, um während der Implementierungsphase des Werkzeugs nicht übergangen zu werden.

Die Umsetzung des Cloning-Verfahrens selbst erfordert keine Erweiterung der Modellbeschreibung. Durch die Implementierung einer Versionierung können die einzelnen Versionen des Simulationsmodells separat gespeichert und somit stets reproduziert werden.

Zusätzliches Attribut für die Anwendung der Versionierung ist minimal die Angabe einer Versionsnummer, unter der verschiedene zusätzliche Informationen gespeichert werden können (beispielsweise der Name des Bearbeiters und ein erläuternder Kommentar). Mit diesem Verfahren ist ein Vergleich verschiedener Versionen möglich und eine Änderung in einem Modellbaustein, die zu einer neueren Version führt, birgt nicht die Gefahr, dass vorhandene Simulationsmodelle nicht mehr korrekt funktionieren, die eine frühere Version des Modellbausteins verwenden. Über spezielle Abfragen soll der Anwender in der Visualisierung der Modellierungskomponente über neuere Versionen der verwendeten Modellbausteine informiert werden, die zusätzlichen Attribute einsehen und ggf. die verwendeten Modellbausteine durch die aktuellste Versionen ersetzen können. Über die Versionierung ist es auch möglich, dem Anwender genau den Zustand des Simulationsmodells zu präsentieren, der seinem letzten Bearbeitungsschritt entspricht. Theoretisch wäre es möglich, sich das Attribut Version für jeden Modellbaustein zu sparen und stattdessen jeder neuen Bausteinversion auch einen neuen eindeutigen Bezeichner aufzuprägen. Um das oben skizzierte Verfahren der automatischen Identifizierung von neueren Bausteinversionen anwenden zu können, müssten dann in der Simulationsdatenbank zusätzliche Verweise gespeichert werden, die eine solche Identifikation ermöglichen. Im Rahmen der Implementierung soll die erste Vorgehensweise bevorzugt werden. Die Umsetzung der zusätzlichen Funktionalitäten im Rahmen der Versionierung soll den Anwender bei der Modellierung im Team unterstützen, indem ihm die Änderungen anderer Anwender dargestellt werden. Die Kooperation innerhalb eines Projektteams wird gefördert und doppelte Arbeiten werden vermieden.

Zur Umsetzung des in Abschnitt 5.1.6 beschriebenen Rechtemanagements wird davon ausgegangen, dass die spätere Implementierung eine Aggregation auf ein einziges Attribut ermöglicht, dass für alle Bausteine einer Bibliothek bzw. für jeden Modellbaustein hinterlegt werden kann. Dies kann beispielsweise umgesetzt werden, indem jegliche verwendete Kombination in der Datenbank abgelegt wird und somit durch einen eindeutigen Bezeichner identifiziert werden kann. Tabelle 12 listet die für die

Mehrbenutzer-Modellierung und –Simulation benötigten zusätzlichen Attribute der Modellbeschreibung zusammenfassend auf.

| Bezeichnung | Beschreibung |
|---|---|
| version | Versionsnummer des Modellbausteins |
| locking_status | Aktueller Status (nur zur Laufzeit) |
| rights | Eindeutiger Bezeichner zur Umsetzung des Rechtmanagements |

**Tabelle 12: Zusätzliche Attribute für die Mehrbenutzer-Modellierung und -Simulation**

Die um die oben beschriebenen Erweiterungen ergänzte Modellbeschreibung wird für die Mehrbenutzer-Modellierung und -Simulation eines dynamisch detaillierenden Simulationsmodells, das sowohl einen objekt- als auch einen funktionsorientierten Fertigungsprozess beschreiben kann, als ausreichend angesehen. Durch die Konzeption der Rückwärtssimulation und insbesondere der (semi-)automatischen Transformation von Simulationsmodellen zwischen Vorwärts- und Rückwärtssimulation, können weitere Erweiterungen der Modellbeschreibung nötig werden. Diese sollen in 5.2.4.1 genauer analysiert werden. Im Folgenden wird vorher auf das XML-basierte Nachrichtenprotokoll eingegangen werden, das die Kommunikation des Simulatorkerns mit den angeschlossenen Visualisierungskomponenten und dem Motion Planning erlaubt, um die Konzeption hinsichtlich einer vorwärtsgerichteten Modellbeschreibung abschließen zu können.

### 5.2.2 Nachrichtenbasierte Kommunikationsschnittstelle

Durch den modularen Aufbau des Werkzeugs muss der Simulatorkern das in einem XML-Format gespeicherte Simulationsmodell in ein Java-Programm transformieren, kompilieren und ausführen (vgl. Abschnitt 5.1.4). Simulationsmodell und Simulator ergeben dadurch ein einziges, eng verzahntes Modul, an das sich zur Ausführungszeit eines Simulationsmodells verschiedene Visualisierungskomponenten anschließen können. Um die Kommunikation des Simulatorkerns mit den Visualisierungen zu standardisieren, wird im Folgenden eine Nachrichtenbasierte Kommunikationsschnittstelle konzipiert, die alle Anforderungen strukturiert und deren Bewältigung ermöglicht. Wie in Abschnitt 5.1.5 erläutert, wird auch diese Kommunikationsschnittstelle über ein erweiterbares XML-Format spezifiziert. Die Vorteile der XML vor dem Hintergrund einer erweiterbaren Implementierung wurden dazu unter Abschnitt 3.5.7 bereits erläutert. Für die Gestaltung einer Kommunikationsschnittstelle für ein erweiterbares Werkzeug, dessen Module eigenständig auf verschiedenen Computern laufen sollen, gibt es zu diesem Nachrichtenbasierten Kommunikationsprotokoll keine sinnvollen Alternativen. Der direkte Zugriff auf gegenseitige Methoden verbietet sich durch die mögliche Verteilung der Anwendung und der Anwender.

Grundsätzlich sollen drei verschiedene Nachrichtentypen unterschieden werden: *Normal-Message*, *Request-Message* und *Reply-Message*. Die Typisierung richtet sich nach der Art der transportierten Information und strukturiert den Nachrichtenverkehr auf einer logischen Ebene. *Normal-Messages* dienen dem einfachen Austausch von Information, beispielsweise der Übertragung von Steuerungsbefehlen an den Simulatorkern. *Request-Messages* sind Nachrichten, die eine Anfrage an den entsprechenden Kommunikationspartner senden und als Antwort eine *Reply-Message* erhalten. In den

folgenden Abschnitten sollen die einzelnen Nachrichten dargestellt werden. Die Unterteilung des folgenden Abschnitts erfolgt nach der Reihenfolge der Ausführung im Rahmen eines Simulationslaufs mit einer angeschlossenen Visualisierung. Anschließend werden die benötigten Erweiterungen für die speziell entworfenen Module betrachtet.

#### 5.2.2.1 Initialisierungsnachrichten

Initialisierungsnachrichten werden zu Beginn der Kommunikation zwischen Simulatorkern und Visualisierungskomponente verwendet. Sie dienen der Beschreibung der Szene, Token, Grundlagen der Animation und der Einstellung von Modulparametern. Die nachfolgende Tabelle 13 beschreibt die vorhandenen Nachrichten und ihre jeweilige Typisierung.

| Bezeichnung | Typ | Beschreibung |
|---|---|---|
| Databaseinfo | Normal | Übermittelt die Datenbankinformationen zum Laden der 3D-Repräsentanten zur Darstellung der Modellbausteine |
| Buildingblock | Normal | Übermittelt Position und 3D-Darstellung einer spezifischen Bausteininstanz; Wird ggf. für jede Bausteininstanz des Simulationsmodells gesendet. |
| Tokenpath | Normal | Übermittelt für jeden Modellbaustein die benötigten Animationspfade der Token und ihre relative Position zum Modellbausteine |
| Token | Normal | Übermittelt die Position und 3D-Repräsentanten eines Token |
| EndOfInitialization | Normal | Zeigt, dass alle Informationen der Initialisierungsphase übertragen wurden |

**Tabelle 13: Initialisierungsnachrichten**

Verbindet sich eine Visualisierungskomponente mit dem Simulatorkern, übermittelt dieser alle benötigten Informationen in einer Initialisierungsphase an die neu angeschlossene Visualisierungskomponente. Mit der *EndOfInititialization-Message* wird die Initialisierungsphase beendet; die Simulation wird fortgeführt oder explizit mittels einer Steuerungsnachricht gestartet.

#### 5.2.2.2 Steuerungs- und Manipulationsnachrichten

In der Gruppe Steuerungs- und Manipulationsnachrichten werden alle Nachrichten zusammengefasst und beschrieben, die zur Steuerung des Simulatorkerns aus der Visualisierungskomponente heraus zur Verfügung stehen. Darüber hinaus werden alle Nachrichten aufgeführt, mit deren Hilfe der Anwender einzelne Attribute der Modellbausteine abfragen, aktualisieren und manipulieren kann. Eine genaue Beschreibung der zur Verfügung stehenden Nachrichten findet sich nachfolgend in Tabelle 14.

Die erste Gruppe der Nachrichten dient der Steuerung des Simulatorkerns aus der Visualisierungskomponente. Neben den üblichen Steuerungsbefehlen kann eine Simulation gespeichert werden. Die Modellvalidierung kann mit dieser Zwischenspeicherung von Simulationsläufen erheblich verbessert werden, weil z.B. ein spezieller Zustand eines Simulationslaufs rekonstruiert werden kann. Der Anwender ist nicht gezwungen, eine Simulation stets neu zu beginnen und bis zur problematischen

Stelle zu simulieren. Für alle Nachrichten werden mögliche Fehlernachrichten vorgehalten.

| Bezeichnung | Typ | Beschreibung |
|---|---|---|
| **Steuerungsnachrichten** | | |
| Start | Normal | Startet die Simulation im Simulator |
| Pause | Normal | Hält die Simulation an. |
| Stop | Normal | Stoppt die Simulation |
| SaveSimulation | Normal | Speichert den aktuellen Zustand der Simulation ab |
| Timefactor | Normal | Verändert die Ausführungsgeschwindigkeit der Simulation (Timefactor 100 entspricht Echtzeit) |
| Maxspeed | Normal | Streng Ereignisgesteuerte Berechnung der Simulation ohne Verhältnis zur Echtzeit |
| Normalspeed | Normal | Setzt von der maximal möglichen Geschwindigkeit auf den letzten Timefactor |
| Timestamp | Normal | Liefert die aktuelle Simulationszeit |
| Error | Reply | Fehlernachricht mit Beschreibung |
| No-Error | Reply | Bestätigung einer Nachricht, bei der kein Fehler aufgetreten ist |
| **Manipulationsnachrichten** | | |
| SubscribeProperty | Request | Abonniert die Variablen einer bestimmten Bausteininstanz, beispielsweise zur Anzeige in der Visualisierung |
| UnsubscribeProperty | Normal | Beendet ein Abonnement |
| ObjectProperties | Reply | Enthält eine Liste aller Variablen einer Bausteininstanz als Antwort auf die SubscribeProperty-Nachricht |
| PropertiesChanged | Normal | Ändert die Werte einer Variablen oder teilt der Visualisierung einen geänderten Wert einer Variablen mit |
| Significance | Normal | Ändert die Signifikanz eines Objektes. |
| | | |
| IncludeToken | Normal | Dient dem Hinzufügen eines bestimmten Token zu einem anderen Token, beispielsweise einer Schraube ein einer Kiste |
| ExcludeToken | Normal | Dient dem Extrahieren eines Token aus einem bestimmten Token. |
| AnimateToken | Normal | Nachricht zur Animation eines Token in einem Modellbaustein entlang eines vorgegebenen Tokenpaths |
| RemoveToken | Normal | Entfernt das Token aus der Visualisierung |

**Tabelle 14: Steuerungs- und Manipulationsnachrichten**

Die zweite Gruppe an Nachrichten dient der Abfrage und Manipulation von Attributen der Bausteininstanzen. Neben einer dynamischen Anzeige können die Werte aus den Visualisierungskomponenten heraus manipuliert und zum Simulator übertragen werden. Über eine *PropertyChanged-Nachricht* werden andere angeschlossene Visualisierungskomponenten über die Veränderung informiert. Die letzte Gruppe der Nachrichten dient der Umsetzung des Token-in-Token Konzeptes in der Visualisierung, also der Tatsache, dass Token andere Token enthalten können. Damit wird es möglich, Token in Gruppen zusammenzufassen oder Materialflüsse genauer abbilden zu können, ohne Informationen zu verlieren (vgl. Abschnitt 5.1.4).

#### 5.2.2.3 Nachrichtenerweiterung für das MRS

Die Integration der dynamischen Detaillierung von Simulationsmodellen erfordert eine Erweiterung des Nachrichtenprotokolls. Tabelle 15 gibt einen Überblick über die zusätzlich benötigten Nachrichten:

| Bezeichnung | Typ | Beschreibung |
|---|---|---|
| Avatar | Normal | Übermittelt die aktuelle Position der angeschlossenen Avatare |
| MRMQuantifier | Normal | Ermöglicht das Steuern des MRS aus der Visualisierung heraus |
| lockMRM | Normal | Beendet die dynamische Detaillierung und berechnet das Modell auf Basis der aktuellen Detaillierung der Modellbausteine |
| unlockMRM | Normal | Schaltet die Berechnung der dynamischen Detaillierung wieder ein |

**Tabelle 15: Erweiterung für das MRS**

Damit soll zum einen aus den Visualisierungskomponenten die aktuelle Position des Avatars des Anwenders übertragen werden, um eine hohe Detaillierung an den Punkten zu ermöglichen, wo sich der Avatar bewegt. Zum Anderen wird in dieser Nachricht die Blickrichtung des Avatars übertragen, um die dynamische Detaillierung auf Basis dieser beiden, benutzerstimulierten Kriterien zu ermöglichen. Zusätzliche Nachrichten werden benötigt, um die dynamische Berechnung des Detaillierungsgrades ein- bzw. ausschalten zu können. Durch die Umschaltung des Detaillierungsgrades in der Simulation ergibt sich keine neue Darstellung in der entsprechenden Visualisierungskomponente, weil deren Darstellung immer auf dem am höchsten detaillierten Simulationsmodell basiert. Für die einzelnen 3D-Repräsentanten der Bausteininstanzen ergibt sich aber ggf. eine neue Zugehörigkeit zu einer anderen Bausteininstanz und damit neue oder andere Attribute. Dieser Austausch der Attribute kann mit den bestehenden Nachrichten bereits vollständig abgebildet werden (bspw. Properties-Changed-Message).

#### 5.2.2.4 Nachrichtenerweiterung für das Motion Planning

Auch die Kommunikation zwischen dem Motion Planning Modul und dem Simulatorkern soll über das Nachrichtenprotokoll abgewickelt werden. Alternativ zum Nachrichtenkonzept bietet sich an dieser Stelle auch eine unmittelbare Anbindung des Motion Planning an den Simulator an. Zunächst sollen jedoch die Nachrichten des Gesamtsystems in einem gemeinsamen Protokoll integriert werden, unabhängig von der tatsächlichen Implementierung in einer späteren Entwicklungsphase des Werkzeugs. Da hier ein besonders intensiver Austausch von Informationen benötigt wird, müssen entsprechend viele Nachrichten dem Protokoll hinzugefügt werden. Der Ablauf der Bewegungsanfragen folgt der in Abschnitt 5.2.1.2 beschriebenen Reihenfolge.

| Bezeichnung | Typ | Beschreibung |
|---|---|---|
| MPPrefs | Normal | Nachricht zur Festlegung von Motion Planning Parametern. Wird in der Initialisierungsphase gesendet |
| Robot | Normal | Legt einzelne SPMs an |
| Robotpool | Normal | Legt einen Pool von SPMs an |
| | | |
| moveRobot | Request | Fordert die Bewegung eines bestimmten SPMs an. |

| | | |
|---|---|---|
| movePoolmember | Request | Fordert die Bewegung eines SPMs aus einem bestimmten Pool an. |
| freeRobot | Request | Freisetzen eines SPMs nach Ausführung eines Auftrags |
| removeRobot | Request | Entfernt einen SPMs aus der Visualisierung |
| | | |
| StillComputingPath | Reply | Beinhaltet die voraussichtlich benötigte Zeit zur Berechnung eines Wegs |
| ExpectedArrivalTime | Reply | Voraussichtlich benötigte Zeit eines SPMs für einen Auftrag |
| CheckArrival | Request | Prüfung auf erfolgreiche Ausführung eines Auftrags |
| Arrival | Reply | Ankunft eines SPMs am Ziel seines Auftrags |

**Tabelle 16: Nachrichtenerweiterung für das Motion Planning**

Abbildung 34 zeigt eine schematische Darstellung einer optimalen Fahrt eines SPM durch ein Simulationsmodell. Wäre der entsprechende SPM auf seinem Weg aufgehalten worden, würde die jeweilige Check-Arrival-Message mit einer neuen ExpectedArrivalTime-Message beantwortet werden.

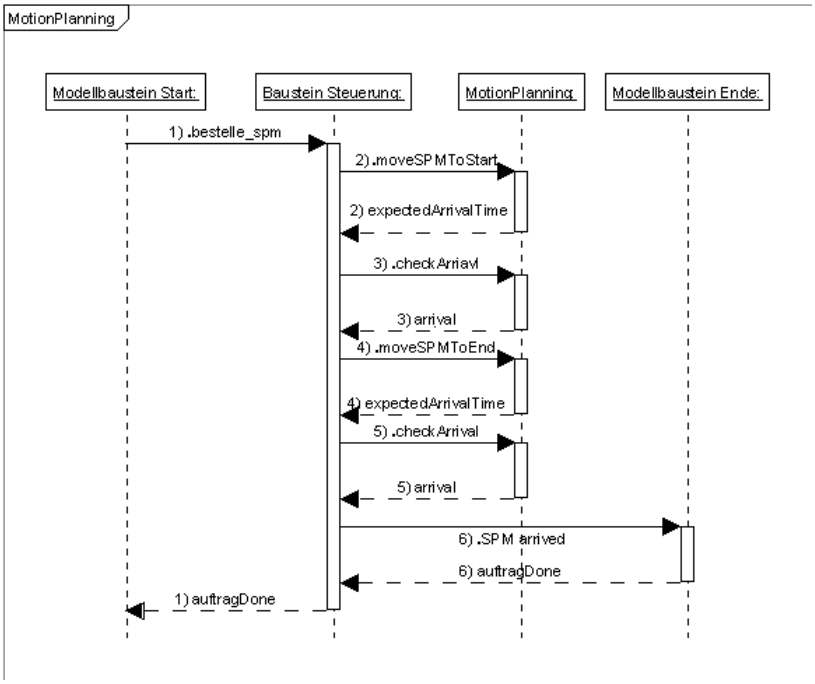


**Abbildung 34: Sequenzdiagramm des Motion Planning Nachrichtenprotokolls**

Mit dem hier dargestellten und für dynamische Detaillierung und Motion Planning erweiterten Nachrichtenprotokoll wird eine Kommunikation zwischen Visualisierungs-komponenten, Motion Planning Modul und Simulatorkern ermöglicht. Durch die Einführung der Rückwärtssimulation im folgenden Abschnitt muss das Nachrichtenprotokoll wie auch die Modellbeschreibung eventuell ausgebaut werden. Für die hier dargestellten Aufgaben ermöglicht das Protokoll eine Manipulation der Simulationsdaten zur Ausführungszeit eines Simulationsmodells und schafft damit neben der Modellbeschreibung die Grundlagen zu einer immersiven und interaktiven Modellierungs- und Simulationsumgebung für den Anwender.

### 5.2.3 Verwaltung der Experimentdaten

Zur Speicherung der Datenmenge eines Simulationsexperimentes soll in diesem Abschnitt eine Struktur vorgestellt werden, die durch eine Beschreibung im XML-Format umgesetzt werden soll. Neben der vollständigen Sicherung aller für ein Simulationsexperiment erforderlichen Daten sollen hier auch Szenarios definiert werden, die bei Bedarf mit wechselnden Eingabedaten simuliert werden können. Dieses Vorgehen unterstützt den Anwender beim erweiterten Einsatz der Ablaufsimulation, weil das wiederholt Parametrieren des Simulationsmodells durch die Verwendung eines Szenarios entfällt. Nur die jeweils aktuellen Prozessabbilder als Eingabedaten werden aktualisiert und führen zu veränderten, auf die aktuelle Situation angepassten Ergebnissen.

Aufgrund der stochastischen Einflüsse zufallsverteilter Variablen innerhalb der Modellbausteine müssen Simulationsmodelle mehrfach ausgeführt werden, um die statistische Schwankungsbreite der Zielkriterien einer Simulation erkennen zu können. Deshalb bestehen Simulationsexperimente immer aus mehreren Simulationsläufen (vgl. Abschnitt 3.1). Ein Experiment kann also in einem ersten Schritt in die einzelnen Simulationsläufe getrennt werden. Darüber hinaus vervollständigen Angaben zu Simulationsmodell, inklusive Versionsnummer, Anwender Start- und Endzeit der Simulation, Ausführungsdatum sowie untersuchtem Szenario etc. die Angaben des Experimentes. Jeder Simulationslauf als Element des Experiments ist durch die Menge der Eingabedaten, die Menge der Interaktionen, einer Protokolldatei des Nachrichtenaustauschs und die eigentlichen Ergebnisdaten gekennzeichnet, so dass für jeden Lauf diese Unterteilung getroffen werden kann. Die Eingabedaten lassen sich auf die einzelnen Modellbausteininstanzen des Simulationsmodells verteilt auflisten. Für jedes Attribut kann hier der Startparameter gesichert werden, so dass eine Wiederholung des Simulationslaufs erfolgen kann. Alle Interaktionen des Anwenders werden mitprotokolliert, so dass eine exakte Kopie des Simulationslaufes geschaffen werden kann. Eventuell kann diese als Vorlage für eine Wiederaufnahme in Form eines weiteren Simulationsexperimentes dienen, indem der Anwender weitere Interaktionen vornehmen kann. Ebenfalls protokolliert wird der Nachrichtenaustausch zwischen Simulation und Visualisierung, damit Wiederholungen einzelner Ausschnitte des Simulationslaufs erneut in den Visualisierungskomponenten dargestellt werden können. Dadurch wird es ermöglicht, im Nachhinein Ursachen von Effekten zu ermitteln, die in der Simulation erst später auftreten. Besonderheiten eines Simulationslaufs können automatisch erneut wiedergegeben und beispielsweise in einer Expertenrunde diskutiert werden. Die eigentlichen Ergebnisdaten enthalten eine komplette Liste aller Attribute, die überhaupt in dem Simulationslauf ausgewertet werden sowie jeweils die einzelnen Datensätze in Einzelwerten oder Listen, wiederum aufgeteilt auf diejenigen Bausteininstanzen, in denen die Kennziffern gesammelt wurden. Für eine Analyse auf Basis von Token, die durch das Simulationsmodell laufen, werden spezielle Auswertungsbausteine im Simulationsmodell hinterlegt, deren Attribute ebenfalls unter den Ergebnisdaten abgelegt werden. Abbildung 35 zeigt schematisch die Baumstruktur der Experimentdatenverwaltung.

Zusammenfassend kann festhalten werden, dass mit dieser Experimentdatenverwaltung die einzelnen Simulationsexperimente so vollständig in der Simulationsdatenbank gesichert werden, dass alle relevanten Daten in einer reproduzierbaren Form gespeichert

werden. Einzelne Simulationsläufe können erneut visualisiert werden, ohne dass eine Wiederholung der Simulation erfolgen muss. Eine erneute Analyse wird somit erleichtert.

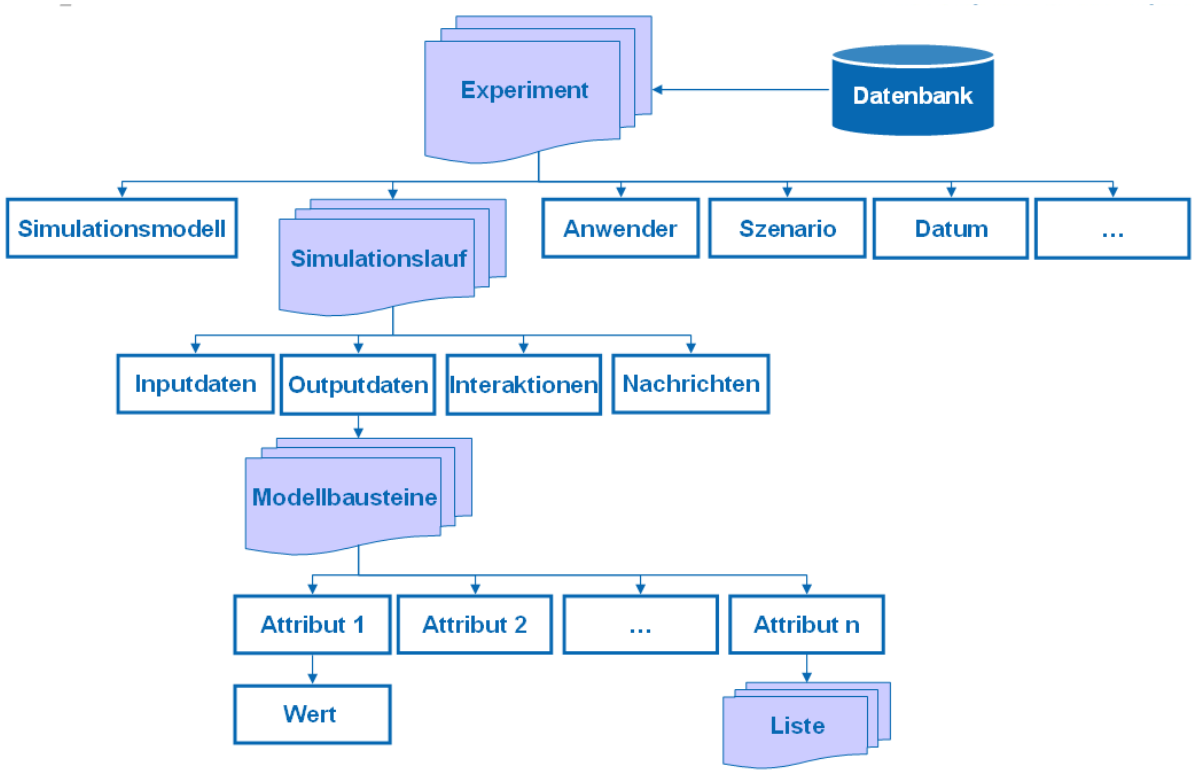


**Abbildung 35: schematische Übersicht über die Experimentdatenverwaltung**

### 5.2.4 Rückwärtssimulation und Modelltransformation

Im folgenden Abschnitt soll die Konzeption einer Rückwärtssimulation beschrieben werden, um eine Simulation hinsichtlich spätester Beginn-Zeitpunkte eines vorgegebenen Fertigungsprogramms mittels der Ablaufsimulation zu ermöglichen. Der Materialfluss ist rückwärts gerichtet und berechnet auf Basis vorliegender Kundenaufträge die spätesten Beginn-Zeitpunkt der benötigten Erzeugnisse auf Basis des Simulationsmodells. Anschließend wird ein Verfahren zur Modelltransformation entworfen, auf dessen Basis Vorwärts gerichtete Simulationsmodelle in rückwärts gerichtete Simulationsmodelle weitestgehend automatisch transformiert werden können. Mit dem Verfahren soll der Modellierungsaufwand für den Anwender für die zwei unterschiedlichen Anwendungsbereiche reduziert werden. Er soll jeweils nur ein Simulationsmodell modelliert werden; die jeweils umgekehrte Richtung des Materialflusses lässt sich daraufhin über das Verfahren weitestgehend automatisch generieren. Dadurch wird die Bearbeitung der mittels Rückwärtssimulation untersuchten Fragestellungen für den Anwender erheblich erleichtert bzw. überhaupt erst in einem zu vertretenden Aufwand realisierbar.

#### 5.2.4.1 Rückwärtssimulation

Im folgenden Abschnitt sollen die einzelnen Schritte zum Aufbau eines rückwärts gerichteten Simulationsmodells beschrieben werden, um sie hinsichtlich besonderer Schwierigkeiten bei der Umsetzung bzw. benötigter Adaptionen der existierenden Modellbeschreibung zu untersuchen und bewerten zu können.

**Modellierung eines rückwärts gerichteten Simulationsmodells**
Die Modellierung eines rückwärts gerichteten Simulationsmodells gestaltet sich durch die Verwendung von Bausteinbibliotheken ähnlich einfach, bzw. ohne erhöhten Aufwand im Vergleich zu einer herkömmlichen Modellierung eines vorwärts gerichteten Materialflusses. Die jeweiligen Modellbausteine müssen für eine Rückwärtssimulation ausgerichtet sein. Im Vorfeld erhöht sich damit der Aufwand für das Erstellen einer Bausteinbibliothek deutlich. Neben der Umkehrung des Materialflussgraphen muss insbesondere die Modellierung der Quellen auf die Denkweise der Rückwärtssimulation adaptiert werden.

Als Startzeitpunkt der Rückwärtssimulation wird eine Zeit aus der Zukunft gewählt. Am Ende des eigentlichen Simulationslaufes erhält man als Simulationsergebnis die Startzeiten der einzelnen Fertigungsprozesse, um die geplanten Erzeugnisse termingerecht fertigen zu können. Im Zuge der Rückwärtssimulation müssen alle Verteilungsregeln umgekehrt werden und ggf. an anderen Stellen im Modell implementiert werden. Abbildung 36 verdeutlicht die Unterschiede in der Steuerung des Materialflusses zwischen Vorwärts und Rückwärts gerichteten Materialflüssen. Dabei müssen die entsprechenden Verteilregeln zwischen Vorwärts- und Rückwärtssimulation entsprechend adaptiert werden.

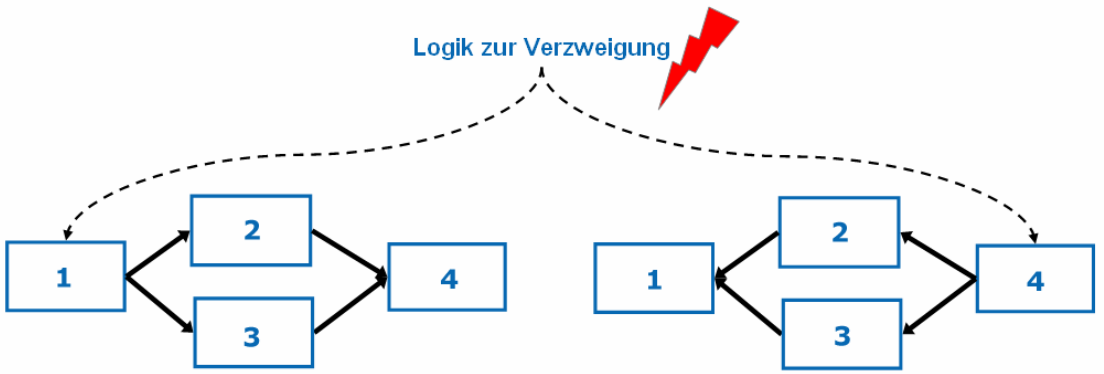


**Abbildung 36: Vergleich eines Materialflusses in Vorwärts und Rückwärts Richtung**

Wie obige Abbildung zeigt, liegt bei dem vorwärts gerichteten Simulationsmodell die Steuerungslogik zur Aufteilung des Materialflusses innerhalb des Modellbausteins *A*. Bei der Umkehrung des Simulationsmodells wird diese Information in Modellbaustein *D* benötigt, um das prinzipiell gleiche Flussverhalten zu erzeugen.

Durch die in Abschnitt 5.2.1 entwickelte Modellbeschreibung können prinzipiell rückwärts gerichtete Materialflüsse modelliert und gespeichert werden, ohne Erweiterungen an der Modellbeschreibung vornehmen zu müssen. Lediglich der Informationsgehalt der durch das Simulationsmodell laufenden Marken ändert sich durch die rückwärts gerichtete Ausführung. Um dem Anwender die Modellierungsarbeit zu vereinfachen, können auch hier Bibliotheken von Modellbausteinen angelegt werden, die durch entsprechende Links verbunden werden. Dabei wandeln sich in der rückwärts gerichteten Modellierung Verzweigungen zu Zusammenführungen und umgekehrt. Die Transformation des eigentlich vorwärts laufenden Materialflusses zu einem rückwärts gerichteten

Simulationsmodell erfolgt hier aber implizit durch den Anwender. Für die Quellen und Senken eines Simulationsmodells müssen dennoch neue Modellbausteine konzipiert und in den Bibliotheken abgelegt werden. Durch die unterschiedlichen Fragestellungen bei Vorwärts- und Rückwärtssimulation können die vorhandenen Bausteine nicht weiter verwendet werden.

**Ausführung eines rückwärts gerichteten Simulationsmodells**

Die Ausführung eines rückwärts gerichteten Simulationsmodells in einem diskreten Materialflusssimulator, wie er durch das zu entwickelnde Werkzeug realisiert werden soll, stellt hinsichtlich des reinen Markenflusses keine besonderen Anforderungen an das Simulationsmodell. Lediglich die Fortschreibung der externen Simulationszeit während der Berechnung des Simulationsmodells im Simulatorkern und damit die Terminierung von Folgeereignissen eines bestimmten Zeitpunktes erfordert eine Unterscheidung zwischen vorwärts und rückwärts gerichtetem Modell und damit fortschreitend bzw. in der Zeit zurück schreitendem Zeitmodell. Hier bieten sich für die Implementierung innerhalb des Simulatorkerns zwei Methoden an, die eine zeitlich rückwärts gerichtete Berechnung erlauben.

Eine strikte Umsetzung von Vorwärts- und Rückwärtssimulation müsste im Simulatorkern explizit zwischen beiden Simulationsarten unterscheiden. Dementsprechend läuft die Simulationszeit vorwärts oder rückwärts, so dass in der Konsequenz der *Scheduler* des Simulatorkerns als Verwalter geplanter Ereignisse die Liste aller geplanten Ereignisse rückwärts durchlaufen muss, um die zeitlich am weitesten fortgeschrittenen und damit die nächsten für die Rückwärtssimulation zu verwendenden Ereignisse abzuarbeiten. Insbesondere für die Ausführungsgeschwindigkeit sind die dafür anfallenden Fallunterscheidungen zwischen Vorwärts- und Rückwärtssimulation als negativ zu betrachten.

Die alternative Methode betrachtet den Simulator als eigenständiges Objekt und bildet die Vorwärts- bzw. Rückwärtssimulation über verschiedene Abbildungen auf den jeweils darzustellenden Kalender ab. Das entspricht zunächst auch dem Ziel der Modularisierung der einzelnen Funktionen, wie sie durch die objektorientierte Programmierung bevorzugt werden (vgl. 3.5.2). Zusätzlich kann im Simulatorkern dadurch eine auf der internen Simulationszeit basierende vollständige und eindeutige Berechnung der Ereignisse ohne jeweilige Fallunterscheidung erfolgen. Alle verwendeten Datenstrukturen können auf diese spezielle Verwendung hin optimiert werden, was insgesamt eine beschleunigte Berechnung erlaubt. Durch die Transformation der internen Simulationszeit auf einen externen Kalender kann eine Parametrierung und Darstellung über prinzipiell beliebige externe Kalender erfolgen. Der Simulator bleibt so ebenfalls erweiterbar für die Integration von weiteren Kalendern wie Werkskalendern von Fertigungsunternehmen oder externen Kalendern mit fixen Zeitinkrementen (Schichtgenaue Betrachtung, etc.). Dieses Vorgehen wird aus den genannten Gründen bevorzugt und soll in der Implementierungsphase umgesetzt werden.

**Auswertung eines rückwärts gerichteten Simulationsmodells**

Ebenso wie die Modellierung unterscheidet sich die Auswertung von rückwärts gerichteten Simulationsmodellen nicht grundlegend von der Auswertung des entsprechenden Pendants. Aufgrund der unterschiedlichen Untersuchungszwecke bei Vorwärts- und Rückwärtssimulation unterscheiden sich die Zielkriterien und damit

mögliche Auswertungen von denen eines entsprechenden vorwärts gerichteten Simulationsmodells. Die in den Auswertungs- und Visualisierungsmodulen zur Verfügung stehenden Auswertungsmöglichkeiten sollen aber auch für die rückwärts gerichteten Simulationsmodelle ausreichend sein, da sie nur eine entsprechende Darstellungsform repräsentieren, ohne diese an eine inhaltliche Aussage zu knüpfen.

### 5.2.4.2 Modelltransformation

Um dem Modellierer die Erstellung eines Simulationsmodells zur Rückwärtssimulation zu erleichtern, soll ein Verfahren entwickelt werden, anhand dessen Simulationsmodelle weitgehend automatisch invertiert werden können. Sie sollen in der unter 4.2.1 vorliegenden Modellbeschreibung vorliegen. Auf Basis vorliegender Materialflüsse werden Grundstrukturen identifiziert, die in einem Folgeschritt hinsichtlich ihrer Invertierung in ein rückwärts gerichtetes Pendant untersucht werden. Der Vorteil dieser Zerlegung des Materialflusses in einzelne Komponenten liegt in der Möglichkeit zur Automatisierung der Erkennung und Invertierung der jeweiligen Grundstrukturen. Dazu soll zuerst eine geeignete Form der Darstellung ausgewählt und erläutert werden. Der nachfolgende Abschnitt befasst sich mit der Identifikation und Auflistung der Grundstrukturen. Sie werden bezüglich ihrer Umkehrung im Sinne einer Rückwärtssimulation untersucht. Die dadurch erreichte Vereinfachung eines Simulationsmodells durch die Zerlegung in Grundstrukturen wird anschließend anhand eines Beispiels gezeigt und das intendierte Verfahren zur automatischen Modelltransformation vorgestellt.

**Darstellung der Materialflüsse**

Die Darstellung der Materialflüsse soll auf die spezifische Problemstellung zugeschnitten werden. Im vorliegenden Fall kann die Darstellungsform auf das Abbilden der Bausteininstanzen und ihrer Beziehungen simplifiziert werden. Bausteininstanzen werden durch einfache Rechtecke dargestellt. Jeder Block verfügt über beliebig viele Input- und Output-Channel. Zur Vereinfachung wurde auf die Darstellung der Input- und Output-Channel verzichtet. Bausteininstanzen werden direkt durch Pfeile verbunden. Darüber hinaus gilt, dass entsprechend der Modellbeschreibung alle Pfeile nur verlinkenden Charakter haben und demnach eine rein logische Abfolge darstellen. Sie enthalten zunächst keine oder zeitliche Information. Förderstrecken oder Transportsysteme werden demnach ebenfalls durch Modellbausteine dargestellt.

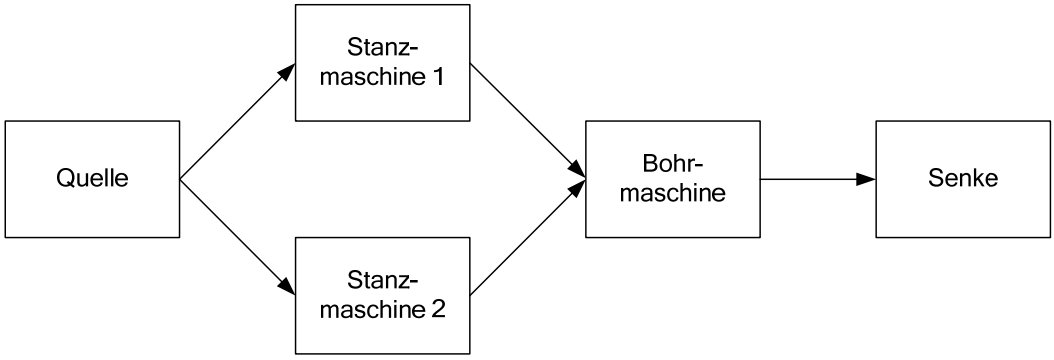


**Abbildung 37: Darstellung einer einfachen Fertigungslinie**

**Identifikation der Grundstrukturen**

Wesentlicher Aspekt bei der Suche nach Grundstrukturen innerhalb von Materialflussstrukturen ist die Frage nach deren Anzahl und Typisierung. Viele Strukturformen sind sich so ähnlich, dass sie problemlos in Typen zusammengefasst werden könnten. Trotzdem ergeben sich auch unter Verwendung von Grundstrukturen einige Formen, die nicht weiter zusammenzufassen sind. Das Hauptproblem stellen hierbei die Schnittstellen zu anderen Modellbausteinen, bzw. zu anderen Grundstrukturen dar: Wird eine bestimmte Struktur als Grundstruktur definiert, dürfen nur an festen Modellbausteinen Ein- oder Ausgänge sein, andernfalls gehen wichtige Informationen verloren und eine Grundstruktur kann nicht mehr eindeutig beschrieben werden. Abbildung 38 verdeutlicht dieses Problem: Wäre die in a) dargestellte Struktur eine Grundstruktur, so müsste trotz der Ähnlichkeit b) eine weitere Grundstruktur sein, da sich an den Modellbausteinen 2 und 3 unzulässige Ein- bzw. Ausgänge befinden.

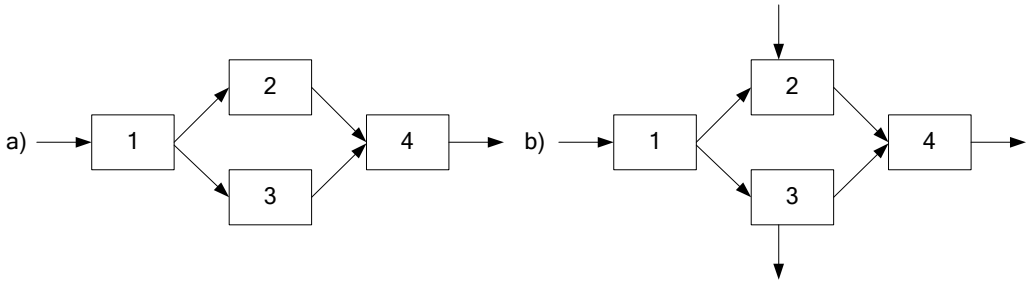


**Abbildung 38: Schnittstellen bei Grundstrukturen**

Wäre aber die in b) dargestellte Struktur ebenfalls eine Grundstruktur, würde das Problem der Grundstrukturen nur verschoben, weil die Anzahl der Strukturen deutlich vergrößert würde. Alternativ dazu soll im weiteren Verlauf eine Liste von Grundstrukturen erstellt werden, deren Umfang vertretbar ist und die alle besonders häufig auftretende Strukturen enthält, so dass die meisten Materialflüsse dadurch zumindest deutlich zu vereinfachen sind. Alle weiteren Strukturen, die in Materialflussmodellen zu finden sind, sollen vom Anwender manuell auf Grundstrukturen zurückgeführt oder alternativ manuell invertiert werden.

Listen von elementaren Grundmustern in Materialflüssen finden sich in der Fachliteratur häufig. Sie dienen in der Regel nicht dem Zweck der Vereinfachung oder Zerlegung komplexer Strukturen, sondern geben vielmehr typische und praxisnahe Anordnungen von Fertigungsmitteln wieder. Der Zweck solcher Listen besteht meist darin, an diesen Beispielen typische Eigenschaften oder Probleme von Materialflüssen zu verdeutlichen. Dennoch bilden solche Listen eine solide Ausgangsposition für einen Katalog von Grundstrukturen zum Vereinfachen von Materialflüssen. Abbildung 39 zeigt eine solche Liste nach [FiHe00]. In der dort gewählten Darstellung sind jedoch die Kanten bzw. Pfeile nicht ohne Bedeutung. Zunächst müssen diese Verkettungsstrukturen also an die gewählte Darstellungsform angepasst und interpretiert werden. Dazu müssen Strukturen, die in Abbildung 39 aus Pfeilen oder ähnlichen Darstellungsformen bestehen, mit Hilfe von Modellbausteinen dargestellt werden. Dies kann je nach Detaillierungsgrad auf unterschiedliche Weise geschehen, weil komplexe Transport- oder Fertigungssysteme durch einen einzigen Modellbaustein, oder auch als Struktur aus vielen Modellbausteinen dargestellt werden können. Die Struktur der Pfeile aus Abbildung 39 kann also unterschiedlich interpretiert werden. Dies spielt jedoch solang keine Rolle, wie alle gewählten Strukturen als Grundstrukturen identifiziert werden können.

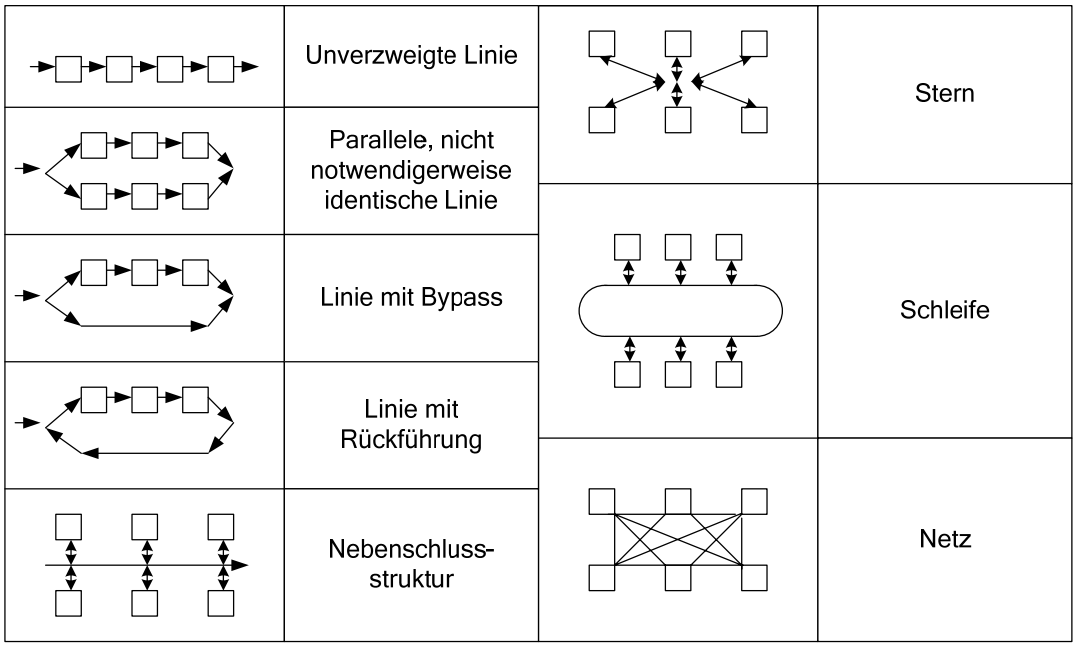


**Abbildung 39: Verkettungsstrukturen in Fertigungssystemen [nach FiHe00]**

Abbildung 40 zeigt die Anpassung an die gewählte Darstellung. Da in der bisher gestalteten Modellbeschreibung lediglich Modellbausteine über Input- und Output-Channel verfügen, müssen alle Verzweigungen und Transportstrecken durch Modellbausteine repräsentiert sein. Exemplarisch sollen im Folgenden einige Umformungen näher erläutert werden.

Bei den *parallelen Linien* müssen beispielsweise bei Verzweigung und Zusammenführung Modellbausteine eingefügt werden. Die Länge der Linien wird nicht weiter dargestellt, da die einzelnen Linien ihrerseits durch unverzweigte Linien ersetzbar sind. Die Anordnung der Modellbausteine ist davon unbetroffen. Die *Nebenschlussstruktur* kann auf zwei Weisen angepasst werden. Entweder wird jede einzelne Verzweigung durch einen Modellbaustein repräsentiert oder die Verteilung des Materialflusses durch ein einziges, komplexeres Transportsystem interpretiert und dieses durch einen einzigen Modellbaustein dargestellt. Das Ergebnis wäre dann eine Sternstruktur. Das *Netz* wird nicht übernommen, da es im Grunde alle Strukturen repräsentiert, die nicht durch die anderen Strukturen darzustellen sind. Aus den Vorüberlegungen ergab sich bereits, dass solche Strukturen zur Vereinfachung von Materialflüssen nicht zweckmäßig sind und vom Anwender manuell zu invertieren sind.

Grundsätzlich sind die Pfeilrichtungen zu beachten. Bidirektionale Pfeile, wie in den Abbildung 39 und Abbildung 40 verwendet werden, sind in der gewählten Darstellungsform nicht vorgesehen, weil sie durch die Modellbeschreibung aktuell nicht abgebildet werden können. Sie werden an dieser Stelle nur zur besseren Übersicht verwendet. Ein bidirektionaler Pfeil symbolisiert zwei gegensätzlich gerichtete Einzelpfeile.

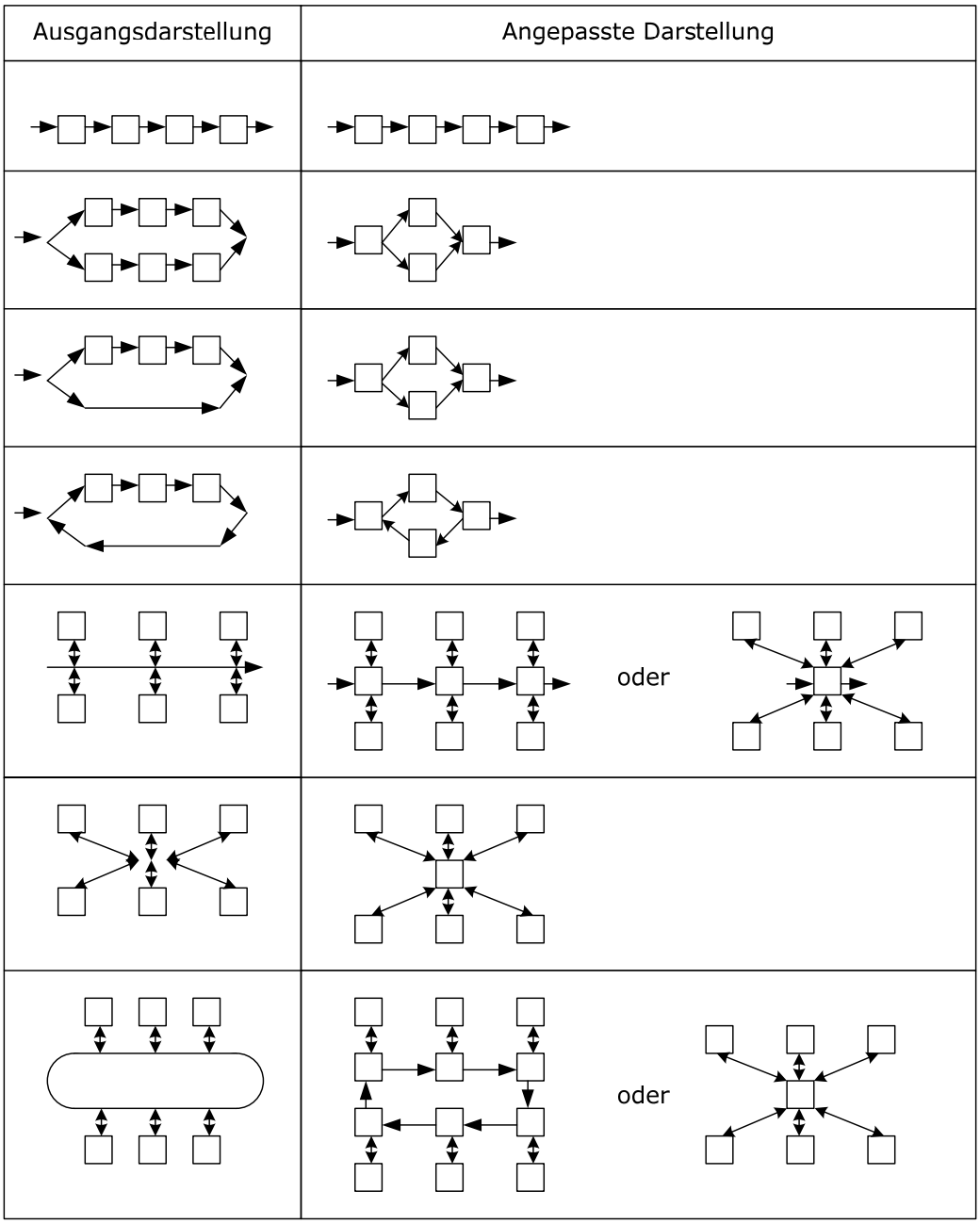


**Abbildung 40: Anpassung der Strukturen an die gewählte Darstellungsform**

In einem weiteren Schritt werden die gefundenen Ausgangslösungen für Grundstrukturen durch die Anwendung von Zerlegungen an realen Materialflüssen verfeinert. In diesem Rahmen wird festgestellt, welche Grundstrukturen such durch Kombinationen von zwei oder mehr Grundstrukturen ausdrücken lassen. Im folgenden Abschnitt werden die resultierenden Grundstrukturen beschrieben. Diese Form der Identifikation von Grundstrukturen mittels empirischer Untersuchung führt nicht zwangsläufig zu einer vollständigen Lösung, bietet aber im Rahmen einer praxistauglichen Anwendung des Werkzeugs gute Ergebnisse. Durch die flexible Erweiterung aller benötigten Strukturen lassen sich auch im Nachhinein weitere Strukturen der Menge der Grundstrukturen hinzufügen.

**Typisierung der Grundstrukturen**

Bei der Zerlegung und Vereinfachung eines Materialflusses muss überlegt werden, wie eine möglichst starke Vereinfachung ohne Informationsverlust erreicht werden kann. Jede der festzulegenden Grundstrukturen muss daher eindeutig beschreibbar sein. Ihre Anwendung muss einheitliche Merkmale aufweisen, die nicht zusätzlich gesichert werden müssen. Um die Zahl der Grundstrukturen jedoch auf einen sinnvollen Rahmen zu begrenzen, sind einige wenige variable Parameter zweckmäßig. Sie sollten aber nur die Größe der Grundstruktur und nicht ihre Struktur bzw. ihre äußere Form betreffen. Ziel ist es also, eine einheitliche Funktionsstruktur bei variabler Größe der Grundstrukturen sicherzustellen. Es muss daher zwischen fixen und variablen Parametern bei Grundstrukturen unterschieden werden.

F*ixe Parameter* umfassen bei allen Grundstrukturen die Form der Struktur und ihre jeweiligen Schnittstellen. Das heißt insbesondere, dass Input- und Output-Channel nur an den dafür vorgesehenen Modellbausteinen erlaubt sind. *Variable Parameter* sind in der Regel die Anzahl von Ein- und Ausgängen sowie die Anzahl bestimmter Modellbausteine. Die Anzahl der Ein- und Ausgänge stellt kein Problem dar, weil sie durch die Anwendung einer Vereinfachung nicht verschwinden. Verdeutlichen soll dies Abbildung 41.

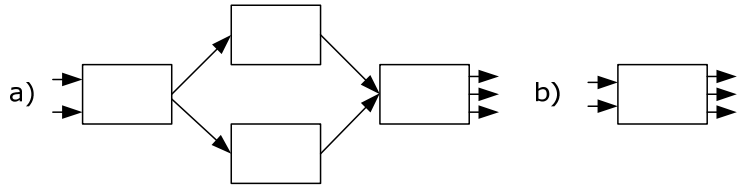


**Abbildung 41: Vereinfachung einer Struktur**

Abbildung 41 links (Teil a) zeigt exemplarisch die Grundstruktur „parallele Linie", wie sie bereits aus Abbildung 39 bekannt ist. Im dargestellten Fall verfügt sie über zwei Ein- und drei Ausgänge. Wird diese Struktur nun vereinfacht, also durch einen einfachen Modellbaustein ersetzt, der die Grundstruktur „parallele Linien" repräsentiert, so geht die Information der Anzahl der Ein- und Ausgänge offensichtlich nicht verloren. Anders sieht es bei der Anzahl bestimmter Modellbausteine innerhalb der Struktur aus. Dies zeigt Abbildung 42.

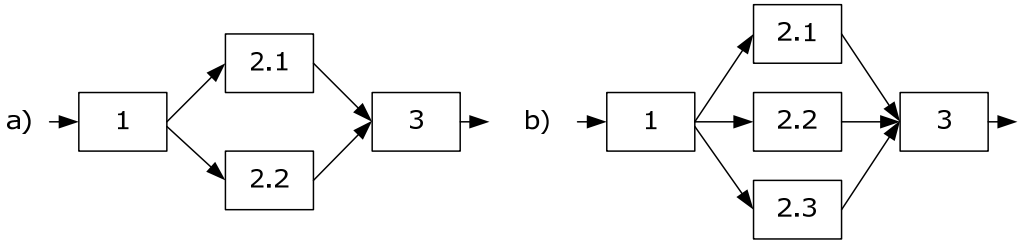


**Abbildung 42: Variable Anzahl Modellbausteine innerhalb Grundstrukturen**

Der linke Teil a) von Abbildung 42 zeigt erneut die bekannte Grundstruktur. Die im rechten Teil b) der Abbildung dargestellte Struktur kann allerdings auch unter diese Grundstruktur fallen, da sie grundsätzlich eine gleiche Form aufweisen. Das Event, in dem die Aufteilung des Materialflusses beschrieben wird, wird für jede Struktur individuell erstellt; es macht folglich keinen Unterschied, ob der Fluss auf zwei oder mehr

Modellbausteine aufgeteilt wird. Solche variablen Parameter treten bei vielen Grundstrukturen auf. Sie müssen vor bzw. während des Vereinfachungsprozesses nachvollzogen und gesichert werden. Im Folgenden werden die identifizierten Grundstrukturen beschrieben und ihre fixen und variablen Parameter erläutert. Abschließend werden jeweils Sonderfälle betrachtet. Dies dient insbesondere zur gegenseitigen Abgrenzung einzelner Strukturen untereinander. Zunächst sollen die Grundtypen *Quelle*, *Senke*, *unverzweigte Linie*, *Verzweigung* und *Zusammenschluss* erläutert werden. Im Anschluss werden Grundstrukturen einer höheren Ordnung vorgestellt, das heißt, sie lassen sich aus den vorhergehenden Strukturen erzeugen, bilden aber selbst wieder feste Strukturen, wie sie in Materialflüssen häufig vorkommen und qualifizieren sich damit als eigener Typ einer Grundstruktur. Zu Ihnen gehören die *Kreuzung*, die *parallele Linie*, die *Linie mit Rückführung* und der *Stern*.

*Quelle und Senke*
Quellen bzw. Senken bilden den Ursprung bzw. das Ende eines jeden Materialflussmodells ab. Je nach abgebildetem Fertigungsprozess kann ein Simulationsmodell durchaus mehrere Quellen und/oder Senken beinhalten. Quellen und Senken sind immer einzelne Modellbausteine, die als solche leicht identifiziert werden können. Sie erzeugen bzw. vernichten die durch den Materialfluss „wandernden" Token nach der vom Anwender implementierten Methodik. Eine Quelle zeichnet sich dadurch aus, dass sie keinen Eingang, aber mindesten einen Ausgang besitzt. Analog zeichnen sich Senken insbesondere dadurch aus, das sie keinen Ausgang haben, aber mindestens einen Eingang.

*Unverzweigte Linie*


**Abbildung 43: Grundstruktur „Unverzweigte Linie"**

Abbildung 43 zeigt eine *unverzweigte Linie*. Bei dieser relativ einfachen Grundstruktur sind die Flussrichtungen der einzelnen Modellbausteine zu beachten. Die Grundstruktur *unverzweigte Linie* herrscht nur dann vor, wenn alle logischen Verknüpfungen in die gleiche Richtung zeigen. Eingänge sind lediglich an Modellbaustein 1 zulässig, Ausgänge nur an Modellbaustein n. Die Anzahl der jeweiligen Ein- und Ausgänge der Modellbausteine innerhalb der Grundstruktur ist beliebig, muss aber innerhalb der Grundstruktur konstant sein. Darüber hinaus muss sichergestellt werden, dass innerhalb der Grundstruktur keine Kreuzungen der Materialflüsse entstehen, beispielsweise wenn die unterschiedlichen Stränge unterschiedliche Token transportieren. Neben der Anzahl von Ein- und Ausgängen bildet die Anzahl der Modellbausteine zwischen 1 und n bzw. die Gesamtanzahl der Modellbausteine in der Struktur die variablen Parameter der Grundstruktur. Als Sonderfall dieser Struktur muss darüber hinaus erkannt werden, dass eine Vereinfachung einer unverzweigten Linie nicht zu einem *Kreis* innerhalb des dann neu entstandenen Modellbausteins führt (vgl. Abbildung 44). In diesem Fall kann die Struktur nicht zur unverzweigten Linie zusammengefasst werden.

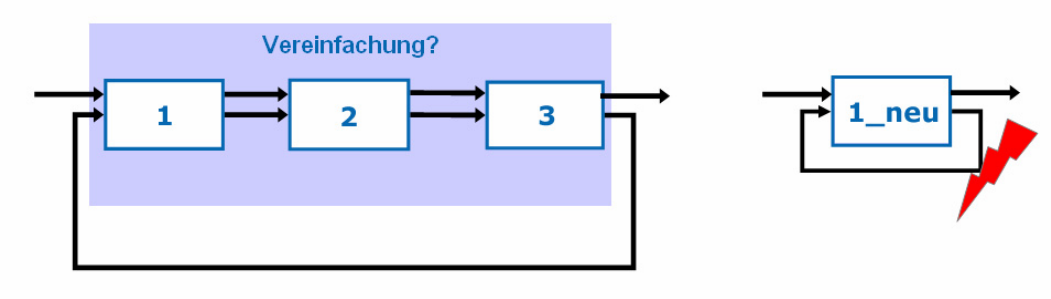


**Abbildung 44: Vereinfachung der unverzweigten Linie führt zum Kreis**

*Verzweigung*

Abbildung 45 zeigt die Grundstruktur „Verzweigung". Die Pfeile, die die Richtung des Materialflusses symbolisieren, gehen hier lediglich in eine Richtung, weg vom zentralen Modellbaustein (in Abbildung Baustein 1) hin zu den äußeren, angehängten Modellbausteinen. Deren Anzahl ist beliebig (variabler Parameter), aber größer oder gleich eins. Somit müssen die Modellbausteine 2.1 bis 2.n letztendlich die Form einer Senke darstellen oder in einer Senke enden. Eine beliebige Menge von Ein- und Ausgängen sind an Modellbausteine 1 selbst zulässig.

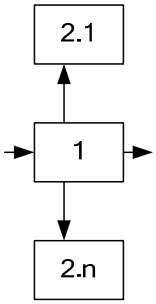


**Abbildung 45: Grundstruktur „Verzweigung"**

Einen Sonderfall stellt die Struktur dar, wenn sie, wie in Abbildung 46, keinen Ausgang besitzt. Diese Struktur unterliegt denselben Regeln wie eine übliche Verzweigung. Endet die Verzweigung ohne Ausgang nicht in jedem angeschlossenen Modellbaustein (2.1 … 2.n) in einer Senke, kann eine Grundstruktur höherer Ordnung vorliegen, für die vereinfachte Umkehrregeln erstellt werden können.

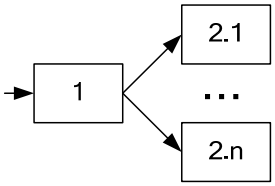


**Abbildung 46: Verzweigung ohne Ausgang**

*Zusammenführung*

Abbildung 47 zeigt den analogen Fall zur vorgestellten Verzweigung: die Zusammenführung. Hier ist lediglich die Flussrichtung umgedreht. Die Anzahl der angehängten Modellbausteine ist mindestens eins, die Anzahl der Ein- und Ausgänge ist beliebig. Ein- und Ausgänge sind nur an Modellbaustein 1 zulässig.

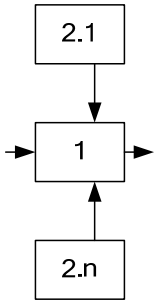


**Abbildung 47: Grundstruktur „Zusammenführung"**

Die Modellbausteine 2.1 bis 2.n müssen aus Quellen resultieren. Der Sonderfall ist hier analog der Fall ohne Eingang (vgl. Abbildung 48). Die eigentliche Funktionsweise unterscheidet sich ebenfalls nicht vom vorgestellten Typ. Dieser Sonderfall ist in der Praxis nicht selten am Anfang von Materialflüssen aufzufinden.

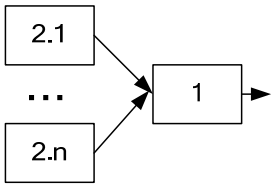


**Abbildung 48: Zusammenführung ohne Eingang**

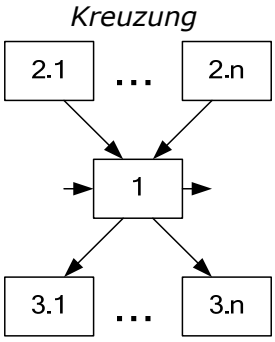


**Abbildung 49: Grundstruktur „Kreuzung"**

Abbildung 49 zeigt die Grundstruktur „Kreuzung". Diese Struktur ist die erste Grundstruktur einer höheren Ordnung, denn sie stellt eine Kombination aus Zusammenführung und Verzweigung dar. Die Anzahl der „eingehenden" (2.1 bis 2.n) und „ausgehenden" (3.1 bis 3.n) Modellbausteine ist jeweils mindestens eins. Ein- und Ausgänge sind nur am zentralen Modellbaustein (in Abbildung: Modellbaustein 1) zulässig. Ihre Anzahl ist prinzipiell beliebig (siehe oben). Von dieser Struktur sind keine Sonderfälle bekannt.

*Parallele Linie*

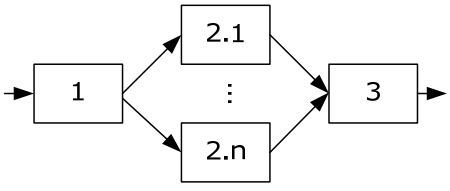


**Abbildung 50: Grundstruktur „Parallele Linien"**

Abbildung 50 zeigt die Grundstruktur „Parallele Linie". Fixe Parameter sind neben der Form der Struktur die Position der Ein- und Ausgänge an den Modellbausteinen 1 bzw. Modellbaustein 3. Die Anzahl der jeweiligen Ein- bzw. Ausgänge zählt zu den variablen Parametern und ist beliebig. Die Zahl der parallelen Linien (in Abbildung: 2.1 … 2.n) ist ebenfalls variabel, aber größer eins. Sonderfälle dieser Struktur sind ebenfalls nicht bekannt, da der Sonderfall nur einer parallelen Linie von der Grundstruktur unverzweigte Linie (siehe oben) bereits abgebildet wird.

*Rückkopplung*

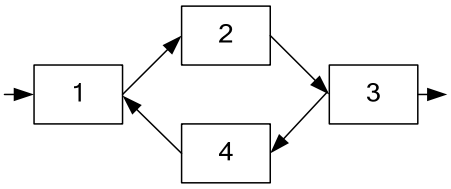


**Abbildung 51: Grundstruktur „Rückkopplung"**

Im Gegensatz zur parallelen Linie ist die Flussrichtung bei der Grundstruktur Rückkopplung in einem Zweig umgedreht (vgl. Abbildung 51). Darüber hinaus besteht die Rückkopplung nach allen möglichen Vereinfachungen innerhalb der einzelnen Modellbausteine aus immer genau vier Modellbausteinen. Eingänge sind nur an Modellbaustein 1, Ausgänge ausschließlich an Modellbaustein 3 zulässig. Die Anzahl von Ein- und Ausgängen ist beliebig und stellt den einzigen variablen Parameter dieser Struktur dar. Sonderfälle dieser Struktur sind nicht bekannt.

*Stern*

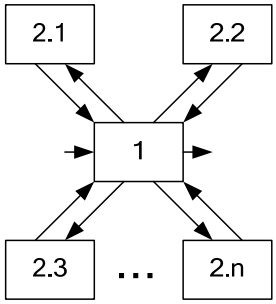


**Abbildung 52: Grundstruktur „Stern"**

Die Grundstruktur „Stern" ist in Abbildung 52 dargestellt. Charakteristisch für eine Sternstruktur ist eine bidirektionale Verbindung einer prinzipiell beliebigen Anzahl von Modellbausteinen mit dem zentralen Modellbaustein. Diese bidirektionalen Verbindungen typisieren im Wesentlichen die Sternstruktur und grenzen diese von anderen Grundstrukturen ab. Es ist daher großer Wert auf die Flussrichtungen der Verknüpfungen zu legen. Die Anzahl der angehängten Modellbausteine stellt neben der Anzahl der Ein- und Ausgänge einen variablen Parameter dar. Fixer Parameter ist dagegen die Position der Ein- und Ausgänge, die beiden lediglich an Modellbaustein 1 zulässig sind. Einen Sonderfall der Sternstruktur bildet die Sternstruktur mit nur einem angehängten Modellbaustein (vgl. Abbildung 53). Dieser Fall hat äußerlich nicht viel mit der Vorstellung eines Sterns gemein, stellt aber dennoch im Grunde den einfachsten Fall einer Sternstruktur dar.

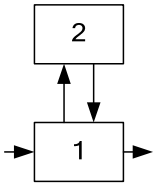


**Abbildung 53: Stern mit nur einem angehängten Modellbaustein**

Abschließend sollen die identifizierten Grundstrukturen auf die in Abbildung 40 dargestellten Strukturen angewendet werden, die den Ausgangspunkt für die Identifikation bildeten. Es fällt zunächst auf, dass die Strukturen „Unverzweigte Linie", „Parallele Linie", „Linie mit Rückführung" sowie „Stern" unverändert übernommen werden, wobei in der gewählten Darstellungsform „Parallele Linie" und „Linie mit Bypass" äquivalent sind. Erweitert wurden diese um die Grundstrukturen „Verzweigung", „Zusammenführung" und „Kreuzung", da diese häufig auftreten und in ihren Eigenschaften nicht durch andere Grundstrukturen darzustellen wären. „Nebenschlussstruktur" und „Schleife" können durch die Anwendung von Grundstrukturen auf eben diese zurückgeführt werden. Abbildung 54 zeigt diesen Vorgang beispielhaft für die Nebenschlussstruktur.

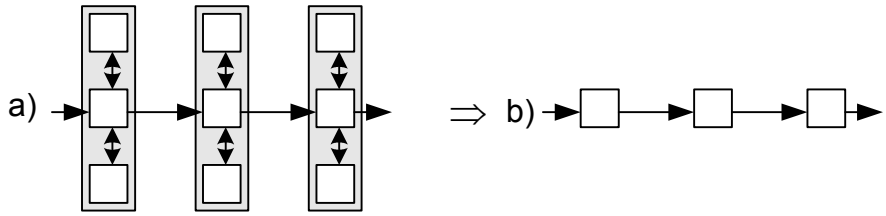


**Abbildung 54: Vereinfachung der Struktur „Nebenschluss"**

Teil a) der Abbildung (links) zeigt die Ausgangsstruktur. Nach dreimaliger Anwendung der Grundstruktur „Stern" entsteht, wie Teil b) (in Abbildung 54 rechts) zeigt, eine „Unverzweigte Linie", die ihrerseits wiederum eine Grundstruktur darstellt. Ähnlich die Verfahrensweise bei der Schleife (vgl. Abbildung 55). Hier wird unter a) zunächst sechsmal die Grundstruktur „Stern" zur Vereinfachung angewendet. Das Ergebnis dieser Vereinfachung zeigt Teil b) der Abbildung. In Teil c) wird nun noch zweimal die Grundstruktur „Unverzweigte Linie" angewendet. Das Ergebnis, dargestellt in Teil d) der Abbildung, bildet die Grundstruktur „Rückkopplung". Es ist demnach nicht notwendig die Strukturen „Nebenschlussstruktur" und „Schleife" in den Katalog der Grundstrukturen aufzunehmen, da sie sich durch Kombination verschiedener anderer Grundstrukturen bereits ausdrücken lassen. Im Gegensatz zu den festgelegten Grundstrukturen höherer Ordnung, die sich ja ebenfalls durch andere Grundstrukturen ausdrücken lassen, treten Nebenschlussstruktur und Schleife auch weniger häufig auf, bzw. sind innerhalb des Materialflusses deutlich schwerer zu identifizieren. Insbesondere deshalb wurde hier auf die Aufnahme der Strukturen in den Katalog der Grundstrukturen verzichtet.

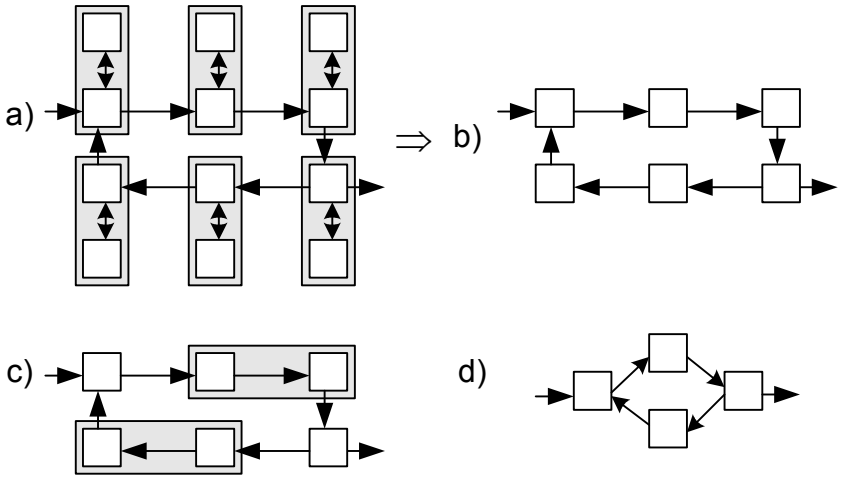


**Abbildung 55: Vereinfachung der Struktur „Schleife"**

Explizit soll an dieser Stelle nochmals darauf hingewiesen werden, dass die identifizierten und in den Katalog aufgenommenen Grundstrukturen keineswegs als „vollständig" anzusehen sind. Wie bereits bei der Identifikation der Grundstrukturen dargelegt wurde, kann keine reduzierte Liste existieren, mit der alle denkbaren Materialflüsse vollständig zerlegbar wären. Erklärtes Ziel soll deshalb sein, einen Katalog an Grundstrukturen zu entwickeln, mit dem möglichst viele in der Praxis vorkommende Materialflüsse möglichst weit zu vereinfachen sind. Es kann daher in speziellen Anwendungsbereichen der Ablaufsimulation sinnvoll sein, diesen Katalog spezifisch zu erweitern.

**Vereinfachung des Materialflussmodells durch Zerlegung**

Nach der Identifizierung der Grundstrukturen soll an dieser Stelle auf den Vorgang der eigentlichen Zerlegung und damit Vereinfachung von Materialflüssen eingegangen werden, die in einem Simulationsmodell beschrieben wurden. Zunächst werden dazu Regeln erläutert, die bei jeder Zerlegung zu berücksichtigen sind, um die Konsistenz des Simulationsmodells zu wahren. Anschließend wird anhand eines Beispiels die Zerlegung eines Materialflusses durchgeführt.

*Regeln für die Zerlegung*

Bei der Zerlegung von Materialflüssen müssen neben den bereits eingeführten variablen Parametern, die vor allem die Struktur der Modelle betreffen, weitere Gesichtspunkte berücksichtigt werden. Zum besseren Verständnis soll zunächst das beabsichtigte Vorgehen bei der Erstellung eines Rückwärtssimulationsmodells auf Basis eines vorliegenden Vorwärtsmodells kurz erläutert werden. Es beinhaltet im Wesentlichen drei Schritte: Die Zerlegung und Vereinfachung des vorwärts gerichteten Ausgangsmodells, die Invertierung aller einzelnen Komponenten und abschließend die Zusammensetzung der invertierten Komponenten zu einem rückwärts gerichteten Simulationsmodells. Die Zerlegung innerhalb des ersten Schrittes soll unter Zuhilfenahme der Grundstrukturen geschehen. Alle identifizierten Grundstrukturen können im zweiten Schritt automatisch invertiert werden. Es ist aber anzunehmen, dass in manchen Fällen „Restmodelle" nach der Vereinfachung übrig bleiben, die nicht durch Grundstrukturen abzubilden sind. Diese müssen vom Anwender individuell interpretiert und manuell invertiert werden. Abschließend werden die invertierten Modellbausteine wieder in einem Simulationsmodell zusammengesetzt. Die Reihenfolge der Zerlegung ist exakt umzukehren. Um sowohl für

die Invertierung als auch für das Zusammensetzen der invertierten Strukturen alle benötigten Informationen bereitzustellen, sind bereits in der Phase der Zerlegung bestimmte Gesichtspunkte zu beachten.

Die Modellbeschreibung des zu entwickelnden Werkzeugs wurde bewusst so angelegt, dass einzelne Modellbausteine mit beliebigen Variablen und Events ausgestattet werden können. Diese individuellen Modellinformationen dürfen bei einer Zerlegung nicht verloren gehen. Die Zerlegung oder Vereinfachung von Simulationsmodellen darf nicht die eigentlichen Inhalte der Grundstrukturen reduzieren. Dies gilt insbesondere dann, wenn eine identifizierte Grundstruktur Quellen oder Senken beinhaltet. Zur Ausführung eines Simulationslaufs würde das Löschen einer Quelle oder Senke den Simulationsablauf in jedem Fall verändern, meist unmöglich machen. Das Ausgangsmodell soll daher durchgehend als Referenz bei der Zerlegung und auch der anschließenden Invertierung und Zusammensetzung verwendet werden. Im Rahmen der Zerlegung werden identifizierte Grundstrukturen durch Modellbausteine ersetzt, um in einem Folgeschritt diese zusammengefassten Modellbausteine eventuell als Teilelemente weiterer Grundstrukturen erneut zu vereinfachen (vgl. Abbildung 55). Das Konzept der hierarchischen Modellbausteine, die weitere Bausteininstanzen beinhalten können, unterstützt diese Vorgehensweise bereits implizit. Jede Zerlegung eines Simulationsmodells läuft daher stufenweise in einzelnen Schritten ab: Zunächst wird das auszuführende Simulationsmodell auf Grundstrukturen untersucht, die sofort identifiziert werden können. Sie werden durch übergeordnete Modellbausteine ersetzt, deren Inhalt zwar nicht verschwindet, aber für den weiteren Verlauf der Zerlegung nicht weiter betrachtet wird. Durch diese Vereinfachung können neue Grundstrukturen entstehen; das resultierende Simulationsmodell wird iterativ in weiteren Schritten durchsucht und alle identifizierten Grundstrukturen werden ersetzt. Die Iteration bricht ab, wenn keine weiteren Grundstrukturen im Restfluss identifiziert werden können oder das gesamte Simulationsmodell in einem Modellbaustein zusammengefasst ist. Die als Grundstrukturen identifizierten Modellbausteine müssen eindeutig bezeichnet und hinsichtlich der Grundstruktur typisiert werden. Alle variablen Parameter müssen zu der identifizierten Grundstruktur protokolliert werden. Außerdem muss die Reihenfolge der Zerlegung festgehalten werden, da sie für das Zusammensetzten der invertierten Modellbausteine in der dritten Phase der Transformation von Bedeutung ist. Um die Vorgehensweise bei der Zerlegung von Materialflüssen zu veranschaulichen, wird im Folgenden beispielhaft ein Materialfluss zerlegt.

*Beispiel der Zerlegung eines Materialflusses*
Abbildung 56 zeigt den Ausgangsfluss des zugrunde liegenden Simulationsmodells. Zur besseren Übersicht sind alle Modellbausteine gestrichelt dargestellt, die Quellen repräsentieren. Analog sind die im Simulationsmodell vorhandenen Senken kariert dargestellt. Jedes Auftreten einer Quelle oder Senke in einer zusammenzufassenden Grundstruktur führt zur Markierung des resultierenden Modellbausteins. So ist selbst dem vereinfachten Simulationsmodell die Grundrichtung des Materialflusses zu entnehmen (von der Quelle zur Senke). Diese Darstellung dient im vorliegenden Fall lediglich der Übersichtlichkeit, kann aber für die zu implementierende, automatische Transformation als Markierung übernommen werden. Das Zusammenfassen einer Grundstruktur, in der Quellen und Senken auftreten, ist nach den beschriebenen Regeln abhängig von der Grundstruktur zulässig.

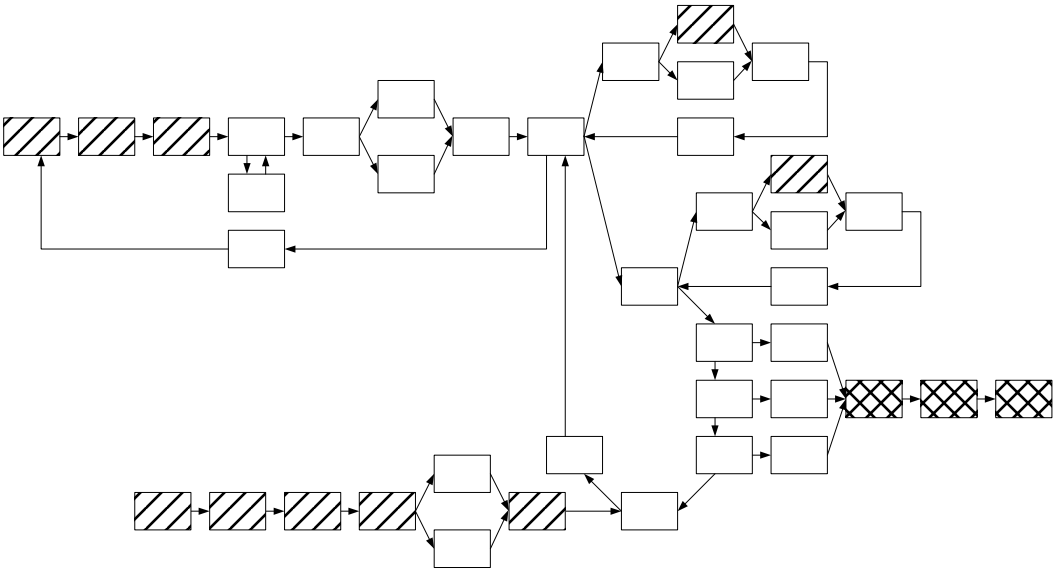

**Abbildung 56: Ausgangsfluss**

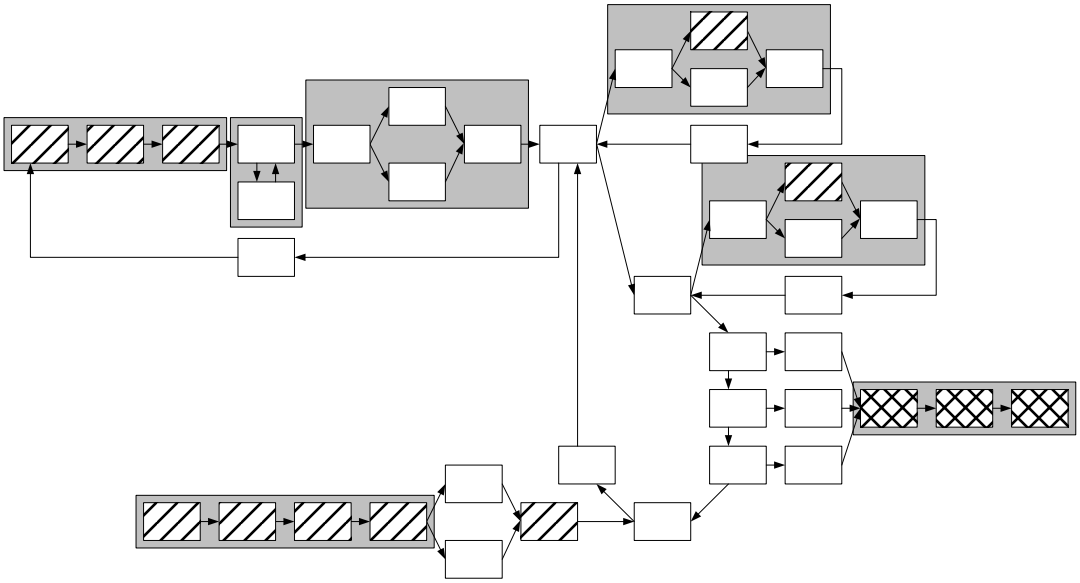

**Abbildung 57: Erster Iterationsschritt**

In Abbildung 57 wird der erste Zerlegungsschritt durchgeführt, indem das Simulationsmodell auf Grundstrukturen untersucht wird. Im Beispiel werden drei *unverzweigte Linien* , drei *parallele Linien* und ein *Stern* identifiziert (in Abbildung 57 grau unterlegt). Diese werden jeweils zu einem Modellbaustein zusammengefasst. Der daraus resultierende Materialfluss ist in Abbildung 58 dargestellt.

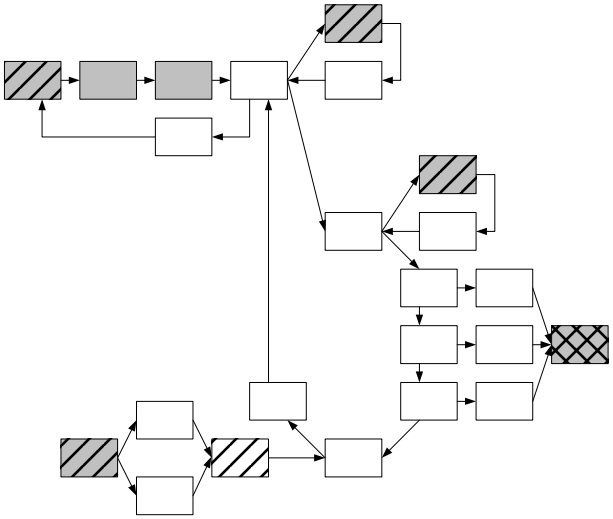

**Abbildung 58: Resultat von Iterationsschritt 1**

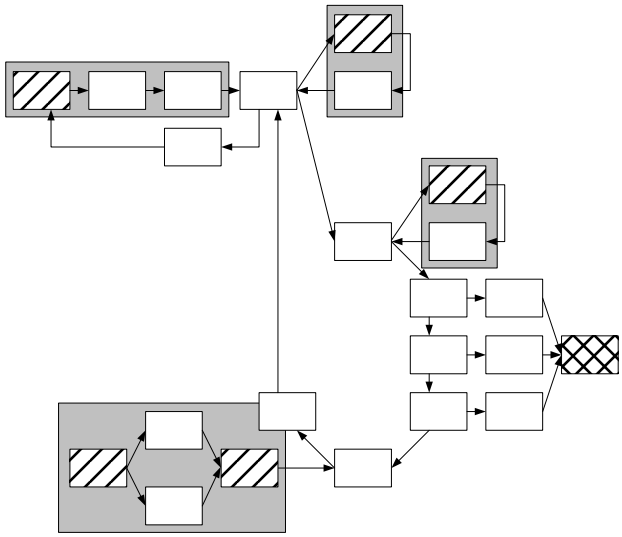

**Abbildung 59: Zweiter Iterationsschritt**

Im zweiten Schritt wird der Materialfluss erneut hinsichtlich neu entstandener Grundstrukturen analysiert (Abbildung 59). Es werden drei weitere Grundstrukturen „Unverzweigte Linie" und eine „Parallele Linie" identifiziert. Diese werden wieder zu Modellbausteinen zusammengefasst; das Resultat des zweiten Analyseschrittes zeigt Abbildung 60.

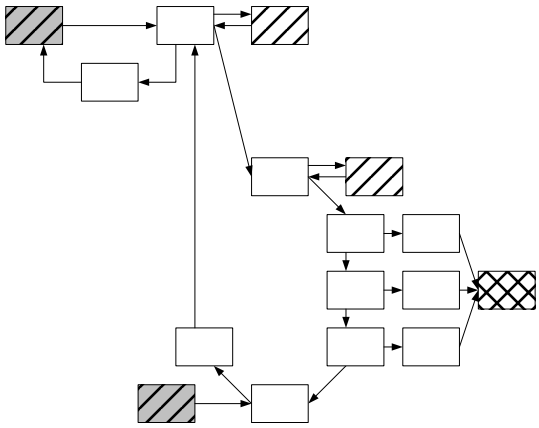

**Abbildung 60: Resultat von Iterationsschritt 2**

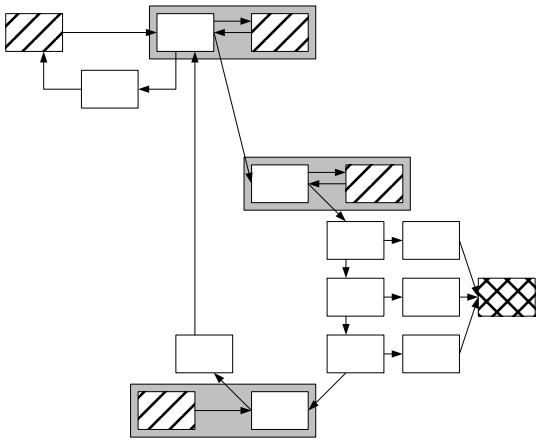

**Abbildung 61: Dritter Iterationsschritt**

Abbildung 61 stellt den dritten Schritt der Zerlegung dar. Es treten hier eine „Zusammenführung" und zwei „Sterne" auf. Bei diesem Schritt ist gut zu erkennen, wie sich aus dem Zusammenfassen von Grundstrukturen im vorherigen Schritt (den „Unverzweigten Linien" und „Parallelen Linien") neue Grundstrukturen („Stern" und „Zusammenführung") ergeben. Das Resultat von Schritt 3 zeigt Abbildung 62. Aufgrund der Vereinfachungen kann zur besseren Übersichtlichkeit der Fluss an dieser Stelle etwas zusammengezogen dargestellt werden.

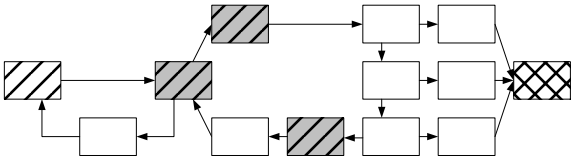

**Abbildung 62: Resultat von Iterationsschritt 3**

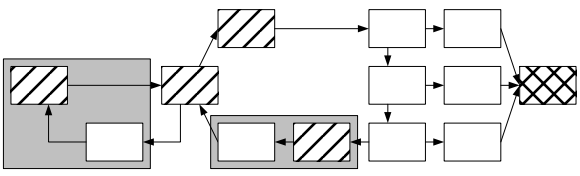

**Abbildung 63: Vierter Iterationsschritt**

Den vierten Schritt zeigt Abbildung 63. An dieser Stelle werden lediglich zwei „Unverzweigte Linien" identifiziert. Abbildung 64 zeigt bereits das Ergebnis des 4. Analyseschrittes.

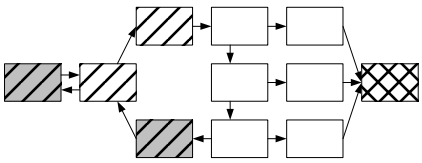

**Abbildung 64: Resultat von Iterationsschritt 4**

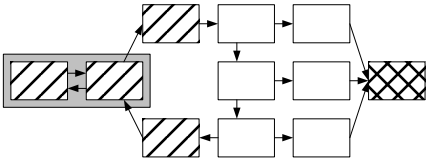

**Abbildung 65: Fünfter Iterationsschritt**

In den letzten zwei Schritten wird jeweils nur noch eine Grundstruktur identifiziert. Trotzdem sind zwei weitere Schritte notwendig, da sich diese Grundstrukturen auseinander ergeben. Im fünften Schritt wird ein „Stern" identifiziert (vgl. Abbildung 65). Dieser wird zusammengefasst (vgl. Abbildung 66).

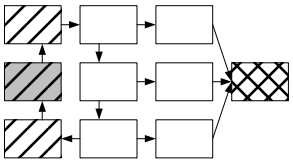

**Abbildung 66: Resultat von Iterationsschritt 5**

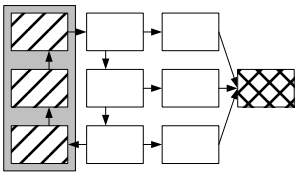

**Abbildung 67: Sechster Iterationsschritt**

Der sechste und letzte Iterationsschritt der Zerlegung und Vereinfachung ist in Abbildung 67 dargestellt. Erneut kann eine „Unverzweigte Linie" festgestellt werden. Sie wird zusammengefasst, das Resultat zeigt Abbildung 68.

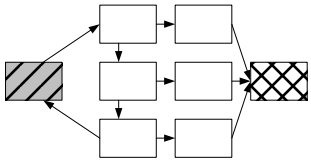

**Abbildung 68: Resultat von Iterationsschritt 6**

Dieses Resultat stellt auch den Restfluss des Simulationsmodells dar, der sich nicht weiter durch Grundstrukturen des Kataloges beschreiben lässt. Ab hier muss das restliche Simulationsmodell vom Anwender manuell invertiert werden. Ein Vergleich des Ausgangsmodells mit dem aus der Iteration resultierenden Restfluss zeigt aber das Potential des angewendeten Verfahrens. Die Arbeit des Anwenders zur Invertierung des

Simulationsmodells kann signifikant reduziert werden, da alle identifizierten Grundstrukturen automatisch invertiert werden können. Eine detaillierte Beschreibung der automatischen Invertierung von Grundstrukturen als zweiter Schritt des Verfahrens ist Gegenstand des folgenden Abschnitts.

**Invertierung der einzelnen Grundstrukturen**

Zur Invertierung der im Katalog festgelegten Grundstrukturen sollen zunächst einige Vorbetrachtungen durchgeführt werden. Zuerst wird mittels Bewertungsmatrizen eines Simulationsmodells ein Konzept eingeführt, das Informationen über die einzelnen Flusstypen des vorwärts gerichteten Simulationsmodells bereitstellt. Voraussetzung für dieses Verfahren ist allerdings mindestens ein vorhandener Simulationslauf des Basismodells. In einem weiteren Schritt werden verschiedene Fälle von Verzweigungen und Zusammenführungen betrachtet, die an Modellbausteinen auftreten können. Für jeden dieser Fälle wird dann eine Methode zur Invertierung in die Rückwärtssimulation entwickelt. Abschließend werden diese Methoden auf die einzelnen Grundstrukturen angewendet. Dadurch wird gezeigt, dass alle Grundstrukturen mit den dargestellten Verfahren hinsichtlich ihrer Simulationsrichtung umgekehrt werden können.

*Analyse von Teilflüssen mittels Bewertungsmatrizen*

Um eine Rückwärtssimulation durchführen zu können, ist es in vielen Fällen nützlich, Informationen über die jeweiligen Teilflüsse des Simulationsmodells zu haben, wie sie sich aus der Vorwärtssimulation ergeben. Aus diesem Grund soll an dieser Stelle als Hilfsmittel für die spätere Umkehrung von Strukturen ein Konzept eingeführt werden, das diese Informationen durch Materialflussmatrizen bereitstellt. Dieses Konzept setzt voraus, dass vor der ersten Rückwärtssimulation das Modell mindestens einmal vorwärts simuliert wurde.

Die Basis dieses Konzepts bilden *Bewertungsmatrizen*[27]. Sie beschreiben Graphen, indem alle Flüsse zwischen den Knoten des betreffenden Graphen als Einträge einer Matrix auftauchen, und zwar sowohl in ihrer Qualität als auch in ihrer Quantität. Für einen aus n Knoten bestehenden Graphen ist die Bewertungsmatrix C eine n*n Matrix. Alle Knoten des Graphen werden durchnummeriert. Das Matrixelement $c_{ij}$ beschreibt dann den Fluss vom Knoten i zum Knoten j. Da analog auch ein Matrixelement $c_{ji}$ existiert, ist auch die Flussrichtung berücksichtigt. Alle Elemente $c_{ij}$ mit i = j, also die Elemente der Hauptdiagonalen, sind null, da keine Schlingen existieren. Existiert keine Kante zwischen i und j, so ist $c_{ij}$ formal unendlich, in diesem Fall jedoch wird diese Stelle einfach nicht gesetzt und damit praktisch $c_{ij} = 0$ gesetzt. Dies soll anhand eines Beispiels erläutert werden (vgl. Abbildung 69).

[27] Eine genaue Beschreibung von Bewertungsmatrizen findet sich bei [Arno95].

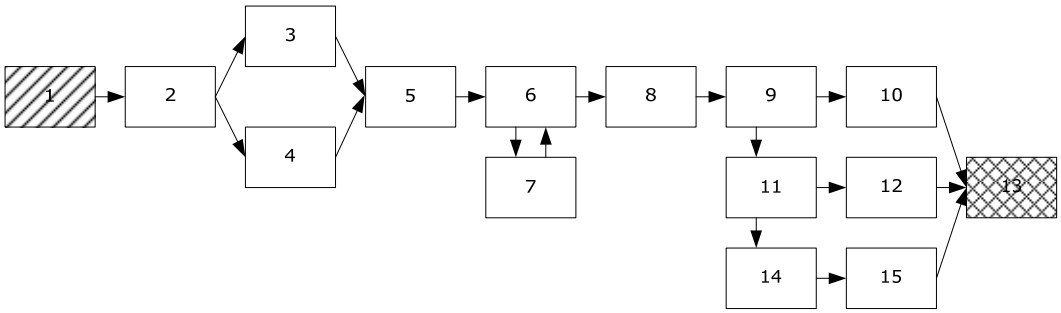


**Abbildung 69: Ausgangsfluss des Beispiels**

Im Gegensatz zu allen bisher betrachteten Darstellungen der Materialflüsse sind hier die Modellbausteine durchnummeriert. Eine dazu passende Bewertungsmatrix $C_1$ zeigt Tabelle 17. Die Darstellung der Matrix in Tabellenform soll die Lesbarkeit verbessern.

In diesem Fall sind die einzelnen Elemente der Matrix lediglich Zahlenwerte, die Mengen pro Zeiteinheit repräsentieren. Denkbar sind auch sehr viel umfangreichere Matrixelemente. Treten in einem Simulationsmodell Token unterschiedlichen Typs auf, so können die Datenstrukturen der einzelnen Token mit einer absoluten oder relativen Mengenangabe Matrixelemente sein. Das formale Konzept der Bewertungsmatrizen wird damit um komplexere Datenstrukturen mit eigenen Eigenschaften oder Parametern erweitert. Relative Mengenangaben können auf den Gesamtinput oder den Gesamtoutput eines Blocks bezogen werden. So können Verzweigungen mit Prozentwerten versehen werden.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 100 | | | | | | | | | | | | | |
| **2** | | 0 | 30 | 70 | | | | | | | | | | | |
| **3** | | | 0 | | 30 | | | | | | | | | | |
| **4** | | | | 0 | 70 | | | | | | | | | | |
| **5** | | | | | 0 | 100 | | | | | | | | | |
| **6** | | | | | | 0 | 50 | 100 | | | | | | | |
| **7** | | | | | | 50 | 0 | | | | | | | | |
| **8** | | | | | | | | 0 | 100 | | | | | | |
| **9** | | | | | | | | | 0 | 50 | 50 | | | | |
| **10** | | | | | | | | | | 0 | | | 50 | | |
| **11** | | | | | | | | | | | 0 | 25 | | 25 | |
| **12** | | | | | | | | | | | | 0 | 25 | | |
| **13** | | | | | | | | | | | | | 0 | | |
| **14** | | | | | | | | | | | | | | 0 | 25 |
| **15** | | | | | | | | | | | | | 25 | | 0 |

**Tabelle 17: Bewertungsmatrix $C_1$**

Die Art der Matrixelemente richtet sich nach den Vorgaben des Simulationsmodells. Für das vorliegende Beispiel sind jedoch absolute Mengenangaben ausreichend. Die Zeilenwerte der Matrix repräsentieren den Output einer Bausteininstanz, die Spaltenwerte den Input. Da bei einer Rückwärtssimulation der Input einer Bausteininstanz zum Output und analog der Output zum Input wird, kann diese Bewertungsmatrix $C_1$ durch Transponieren einfach zu einer Rückwärtsbewertungsmatrix $C_1'$ umgewandelt werden. Da sich während der Zerlegung eines Materialflusses stetig neue Grundstrukturen ergeben, erscheint es sinnvoll, auch für die jeweiligen Zerlegungsschritte eigene Bewertungsmatrizen zu generieren, um dort die für die Invertierung der Grundstrukturen benötigten Informationen zugreifen zu können.

Während der Zerlegung von Simulationsmodellen ist aber keine Vorwärtssimulation aller einzelnen Iterationsschritte der Zerlegung vorgesehen, so dass die Matrix $C_1$ mit zerlegt werden muss.

Den ersten Zerlegungsschritt für obiges Beispiel zeigt Abbildung 70. Die resultierenden Modellbausteine erhalten als neuen, eindeutigen Bezeichner hier die niedrigste Nummer der in ihnen enthaltenden Modellbausteine. Grundsätzlich sind diese Bezeichner frei wählbar. Analog zum Iterationsschritt muss auch die Bewertungsmatrix während dieser Zerlegung angepasst werden. Die Modellbausteine 1 und 2 werden demzufolge zu einem neuen Modellbaustein 1 zusammengefasst. In der resultierenden Struktur sind lediglich der Input von Modellbaustein 1 und der Output von Modellbaustein 2 von Interesse. Da in der Bewertungsmatrix der Input durch die Spalten und der Output durch die Zeilen repräsentiert werden, können in $C_1$ die Zeile 1 und Spalte 2 gelöscht werden. Die übrigen Spalte 1 sowie Zeile 2 repräsentieren In- und Output des neuen Modellbausteins und müssen daher dessen Bezeichner erhalten (hier: 1). Im Falle der identifizierten Sternstruktur der Modellbausteine 6 und 7 sind In- und Output an einem Modellbaustein (hier: 6); es können daher in Tabelle 17 Zeile 7 und Spalte 7 gelöscht werden. Die resultierende Matrix $C_2$ bildet die Bewertungsmatrix für den aus Iterationsschritt 1 resultierenden Fluss (vgl. Tabelle 18).

| | 1 | 3 | 4 | 5 | 6 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 30 | 70 | | | | | | | | | | |
| 3 | | 0 | | 30 | | | | | | | | | |
| 4 | | | 0 | 70 | | | | | | | | | |
| 5 | | | | 0 | 100 | | | | | | | | |
| 6 | | | | | 0 | 100 | | | | | | | |
| 8 | | | | | | 0 | 100 | | | | | | |
| 9 | | | | | | | 0 | 50 | 50 | | | | |
| 10 | | | | | | | | 0 | | | 50 | | |
| 11 | | | | | | | | | 0 | 25 | | 25 | |
| 12 | | | | | | | | | | 0 | 25 | | |
| 13 | | | | | | | | | | | 0 | | |
| 14 | | | | | | | | | | | | 0 | 25 |
| 15 | | | | | | | | | | | 25 | | 0 |

**Tabelle 18: Bewertungsmatrix $C_2$**

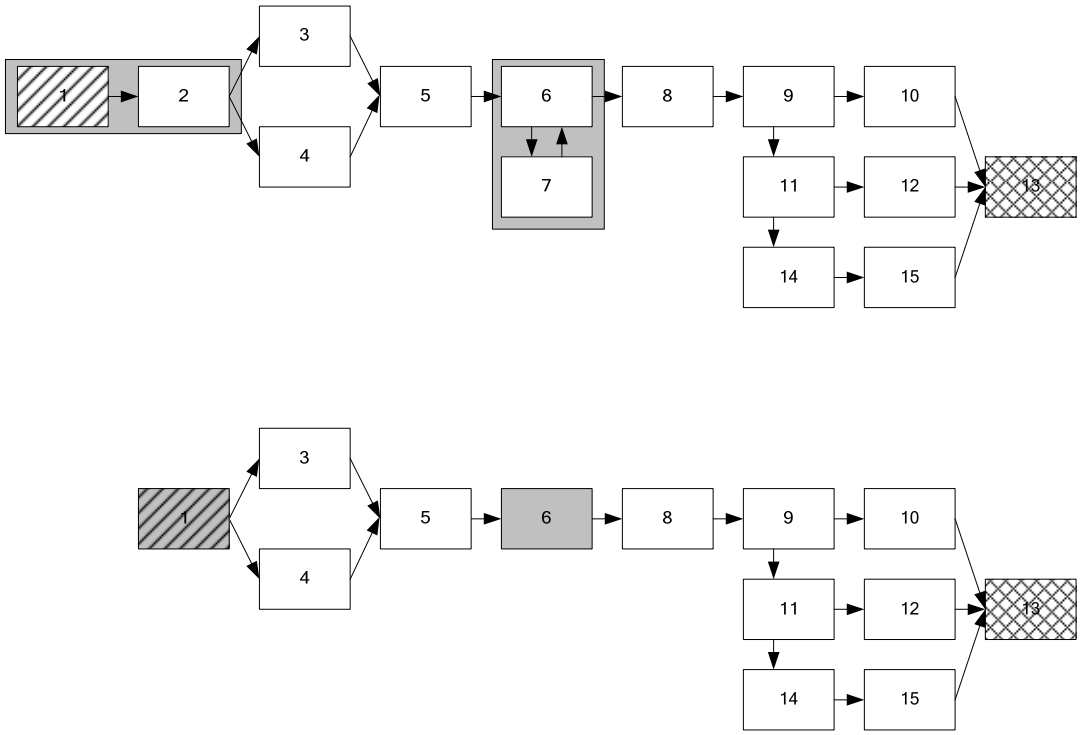


**Abbildung 70: Erster Iterationsschritt**

Analog zum obigen Beispiel erfolgt anschließend Iterationsschritt 2 (Abbildung 71). Die Blöcke 1,3,4 und 5 werden mit der Grundstruktur „Parallele Linie" zusammengefasst. Für die Matrix $C_2$ bedeutet das, dass die Zeilen 3 und 4 sowie die Spalten 3 und 4 ebenso gelöscht werden können wie Zeile 1 und Spalte 5. Zeile 5 wird anschließend in Zeile 1 umbenannt und, entsprechend der Ordnung, nach vorne gezogen. Die Modellbausteine 6 und 8 werden ebenfalls zusammengefasst (Zeile 6 und Spalte 8 löschen, Zeile 8 in Zeile 6 umbenennen). Die resultierende Matrix $C_3$ zeigt Tabelle 19.

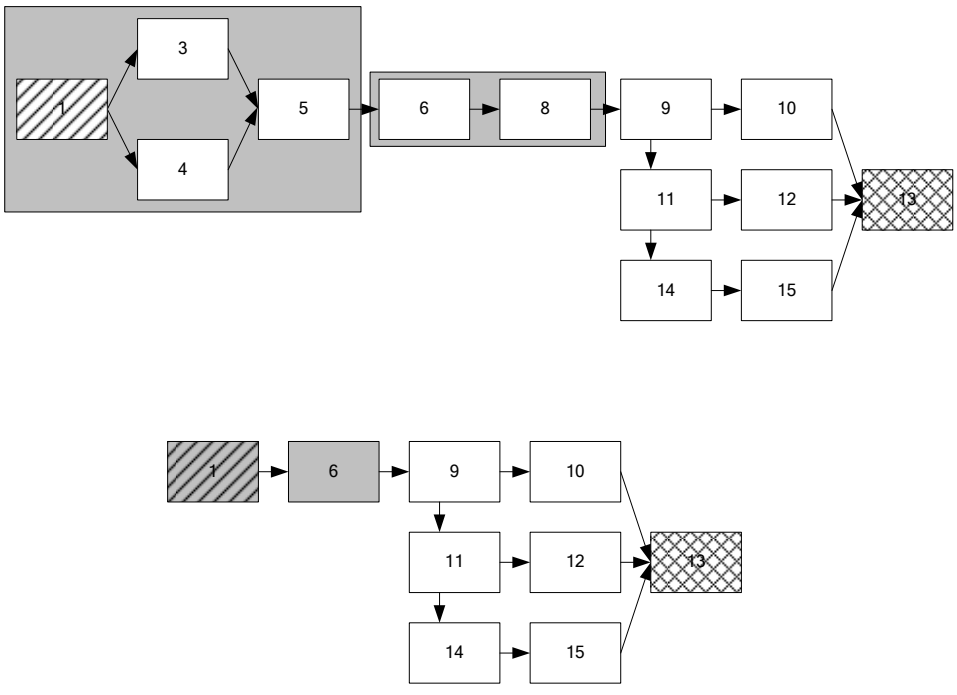


**Abbildung 71: Zweiter Iterationsschritt**

| | 1 | 6 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 100 | | | | | | | |
| **6** | | 0 | 100 | | | | | | |
| **9** | | | 0 | 50 | 50 | | | | |
| **10** | | | | 0 | | | 50 | | |
| **11** | | | | | 0 | 25 | | 25 | |
| **12** | | | | | | 0 | 25 | | |
| **13** | | | | | | | 0 | | |
| **14** | | | | | | | | 0 | 25 |
| **15** | | | | | | | 25 | | 0 |

**Tabelle 19: Bewertungsmatrix $C_3$**

Den letzten Zerlegungsschritt zeigt Abbildung 72. Analog der bereits beschriebenen Vorgehensweise werden in $C_3$ Zeile 1 und Spalte 6 gelöscht, Zeile 6 in Zeile 1 umbenannt und ggf. neu sortiert. Das Resultat zeigt Tabelle 20.

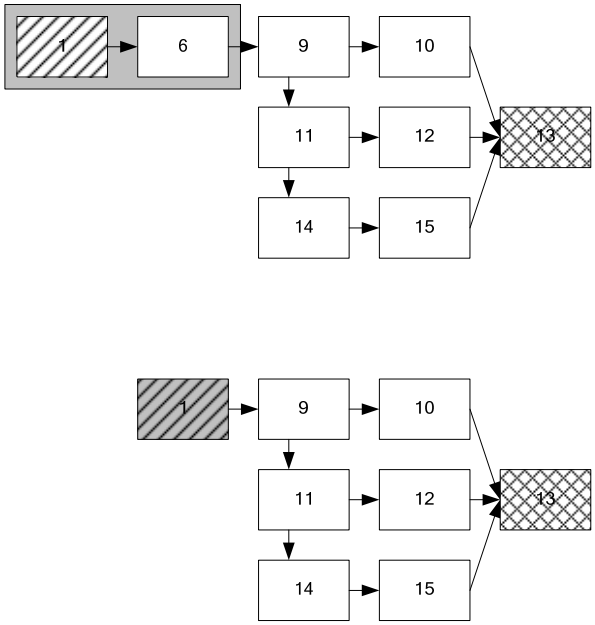


**Abbildung 72: Dritter Iterationsschritt**

| | 1 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 100 | | | | | | |
| **9** | | 0 | 50 | 50 | | | | |
| **10** | | | 0 | | | 50 | | |
| **11** | | | | 0 | 25 | | 25 | |
| **12** | | | | | 0 | 25 | | |
| **13** | | | | | | 0 | | |
| **14** | | | | | | | 0 | 25 |
| **15** | | | | | | 25 | | 0 |

**Tabelle 20: Bewertungsmatrix $C_4$**

Es existiert nun für den resultierenden Restfluss eine Bewertungsmatrix, die ursprünglich mit Hilfe der Vorwärtssimulation erstellt wurde. Sie kann dem Anwender, der den Restfluss umkehren muss, wertvolle Informationen über Quantität und Qualität der abgebildeten Materialflüsse bereitstellen. Wichtiger ist allerdings, dass zu jedem aus einem Iterationsschritt resultierenden Restfluss eine Bewertungsmatrix existiert, aus der

die für den nächsten Zerlegungsschritt benötigten Informationen entnommen werden können. Wie später bei der Umkehrung einzelner Grundstrukturen gezeigt werden wird, können aus den Bewertungsmatrizen verschiedene Informationen extrahiert werden, die die jeweilige Umkehrung erheblich vereinfachen können.

*Events und deren verhaltensspezifische Umkehrung*
Nachfolgend werden die unterschiedlichen Fälle von Verteilungsstrukturen untersucht, die in Modellbausteinen typischerweise auftreten. Dafür werden jeweils Verfahren vorgeschlagen, wie diese Fälle in einer Rückwärtssimulation behandelt werden sollen. Sie werden auf die Grundstrukturen übertragen, um diese eindeutig umkehren zu können. Aus den jeweiligen Fällen werden Empfehlungen für die Modellbildung abgeleitet, sofern sie die Umkehrung im Sinne der Rückwärtssimulation vereinfachen und die Freiheitsgrade des Anwenders nicht einschränken. Grundsätzlich können drei unterschiedliche Funktionen innerhalb eines Modellbausteins unterschieden werden: Das Verstreichen von Zeit, die Veränderung (in der Anwendung Fertigung: Bearbeitung) des Token und die logische Verteilung innerhalb des Materialflusses auf verschiedene Channel. Ferner existieren Quellen und Senken, wobei Kombinationen dieser Fälle existieren. Allein das Verstreichen von Zeit ist den meisten Vorgängen zueigen. Innerhalb dieser drei Typen von Verhaltensweisen sind wiederum verschiedene Aspekte zu unterscheiden.

Der erste Verhaltenstyp beschreibt das reine *Verstreichen von Zeit*, zum Beispiel bei Transport- oder Förderstrecken. Dabei werden weder die Token manipuliert noch der Materialfluss verzweigt. Die Bearbeitungszeit ist für jede Ausführung einer Simulation essentiell. Bearbeitungs- oder Transportzeiten können ihrerseits zeitabhängig sein, beispielsweise durch die Realisierung zeitabhängiger Kapazitäten im Modellbaustein des Simulationsmodells. Hier ist die Bearbeitungs- oder Transportzeit eine zeitabhängige Funktion, die eventuell auf weitere Modellbausteine oder Ereignisse (zum Beispiel Kalender oder Zeittafeln) zugreifen muss. Es ist anzunehmen, dass bei den meisten Ereignissen innerhalb eines Modellbausteins Bearbeitungszeiten auftreten. Deshalb sind diese Zeiten im Idealfall innerhalb eines standardisierten Events zu realisieren, auf das jederzeit unabhängig von anderen Events im Modellbaustein zugegriffen werden kann. Dieses im Folgenden als *Delay-Event* bezeichnete Ereignis beinhaltet im einfachsten Fall nur eine Konstante, kann aber auch jede der angesprochenen Funktionen zur Verzögerung eines Token während eines Simulationslaufs enthalten.

Eine *Veränderung der Token* als zweiter Verhaltenstypus tritt immer dann auf, wenn Token innerhalb eines Modellbausteins in einer beliebigen Form manipuliert werden. Es ändern sich lediglich Eigenschaften oder Parameter des Token. Ein Event, das diese Veränderung eines Token herbeiführt, ist nicht zwangsläufig umkehrbar. Um bei einer Rückwärtssimulation diese Veränderung automatisch rückgängig zu machen, ließe sich die Bewertungsmatrix verwenden. In dieser ist sowohl die Datenstruktur der eingehenden als auch die der ausgehenden Token abgelegt. Ein einfacher Vergleich beider liefert die benötigte Information. Im Gegensatz dazu werden zum Beispiel bei *Montagevorgängen* mehrere unterschiedliche Token zu neuen Token umgewandelt. zunächst irrelevant bleibt, ob die in den Modellbaustein eingehenden Token über einen oder mehrere Input-Channel eintreffen. Analog können auch zwei unterschiedliche, im Modellbaustein neu zu erzeugende Token aus den eintreffenden Token resultieren. Verdeutlichen soll das Abbildung 73. Im linken Fall werden aus drei Token des Typs A

und einem Token des Typs B ein Token C. Im rechten Fall wird aus vier Token A und einem Token B ein Token D (zum Beispiel Tische mit drei oder vier Beinen). Hier kann über die Betrachtung der Ein- und Ausgangsflüsse in der Bewertungsmatrix keine Aussage über die Art des Montagevorgangs getroffen werden. Es wird somit in manchen Fällen nicht möglich sein, das jeweilige Event automatisch umzukehren. Im Sinne der angestrebten Rückwärtssimulation sollen Montagevorgänge oder andere Veränderungen von Token separat durch ein *Transform-Event* realisiert werden. Dieses würde im Fall von Montagevorgängen auch Definitionen der Form C=AAAB oder D=AAAAB enthalten und würde damit eine Umkehrung potentiell ermöglichen.

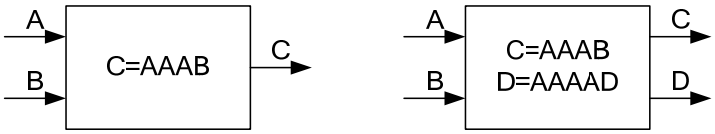


**Abbildung 73: Montagevorgänge**

Die logische Verteilung von Materialflüssen als dritter Typus an Verhaltensweisen beschreibt alle Vorgänge, die strukturell auf den modellierten Materialflusses wirken: Geraden, Verzweigungen und Zusammenführungen. Die Gerade bildet den trivialen Fall, dass die Struktur des Materialflusses nicht beeinflusst wird.
Für alle Verzweigungen und Zusammenführungen gelten für die Umkehrung im Grunde die gleichen logischen Überlegungen. Nach äquivalenten Regeln, wie sich Flüsse verzweigen, können sie auch wieder zusammengeführt werden. Da sich die Problemstellung der Umkehrung des Materialflusses bei Verzweigungen und Zusammenführungen aber unterscheidet, werden sie nachfolgend getrennt betrachtet.

Bei den Verzweigungen kann zwischen drei Unter-Varianten differenziert werden:

- Bei der *Typverzweigung* teilt sich der Materialfluss in Abhängigkeit der ankommenden Token. Dies verdeutlicht Abbildung 74. Unabhängig von der Menge der am Modellbaustein ankommenden Token werden alle Token A auf den oberen Output-Channel und alle Token B auf den unteren Output-Channel verteilt. Umgekehrt wird aus der Verzweigung eine einfache Zusammenführung. Die Reihenfolge, mit der die Token A und B dann in den Output-Channel des rückwärts gerichteten Modellbausteins gelangen, kann teilweise der Bewertungsmatrix der Vorwärtssimulation entnommen werden oder werden mit der gleichen Priorität abgearbeitet.
- Die *Mengenverzweigung* beschreibt einen Verzweigungstyp, bei dem der Materialfluss aus identischen Token mengenmäßig verteilt wird. Dies geschieht in irgendeinem Verhältnis (zum Beispiel 50/50 bei zwei Output-Channeln). Wird diese Verzweigung umgekehrt, können alle ankommenden Token mit der gleichen Priorität abgefertigt werden.
- Eine *last- bzw. kapazitätsabhängige Verzweigung* beschreibt eine Verzweigung, die abhängig von nachgelagerten Kapazitäten des logischen Nachfolgers den Materialfluss verteilt. Hier findet lediglich eine Verfügbarkeitsabfrage an den einzelnen Output-Channeln bei den nachgelagerten Input-Channeln statt, ob eine Weiterleitung an den Nachfolger im Materialfluss möglich ist. Im positiven Fall wird das Token gesendet. Im negativen Fall wird der nächste Output-Channel probiert. Das Vorgehen wiederholt sich analog bei den weiteren Output-Channeln des

Modellbausteins. Sind alle Output-Channel gesperrt, so wird über den ersten frei werdenden Output-Channel gesendet. Diese Verzweigung kann durch Abfertigungsprioritäten umgekehrt werden. Der Output-Channel, der bei der Vorwärtssimulation als erster abgefragt wird, erhält im Rahmen der der Rückwärtssimulation die höchste Priorität. Die anderen Kanäle erhalten entsprechend niedrigere, abgestufte Prioritäten.


**Abbildung 74: Typverzweigung**

Betrachtet man die Gruppe der Zusammenführungen, gelten zunächst analoge Überlegungen wie für Verzweigungen, da in vielen Fällen einer Zusammenführung eine Verzweigung vorangeht. Darüber hinaus existieren Zusammenführungen, denen zwei unterschiedliche Quellen vorangehen. Aus Sicht einer Zusammenführung lassen sich lediglich typabhängige und mengenmäßige Flussarten unterscheiden:

- Analog zu den Verzweigungen ist eine *typabhängige Zusammenführung* eine solche, bei der aus den unterschiedlichen Input-Channeln unterschiedliche Token eingehen. Wird eine Typzusammenführung umgekehrt, so entsteht eine Typverzweigung. Diese benötigt Informationen über die ausgehenden Teilflüsse, die der Bewertungsmatrix zu entnehmen sind.
- Bei einer *Mengenzusammenführung* gehen in die verschiedenen Input-Channel Token des gleichen Typs in bestimmten Mengen ein. Hier muss in einem zweiten Analyseschritt festgestellt werden, ob der Zusammenführung eine Verzweigung oder zwei Quellen vorangehen. Handelt es sich um eine Verzweigung, kann aus Sicht der Zusammenführung nicht unterschieden werden, ob es sich bei dieser Verzweigung um eine Typ-, Mengen- oder um eine last- bzw. kapazitätsabhängige Verzweigung handelt. Dies kann bei der Invertierung des Modellbausteins durch Zugriff auf das Event in der Verzweigung geklärt werden (Werden hier Grundstrukturen der höheren Ordnung identifiziert, liegt diese Verzweigung direkt im Zugriff). Geht beispielsweise eine last- bzw. kapazitätsabhängige Verzweigung voran, so ist das für die Verteilung des Flusses verantwortliche Event aus der Verzweigung für die Umkehrung der Zusammenführung zu verwenden. Dieser Fall scheidet aus, wenn der Zusammenführung zwei Quellen vorausgehen. Hier kann die Zusammenführung nur durch Übernahme der Flussmächtigkeiten aus der Bewertungsmatrix umgekehrt werden.

Um die Umkehrung von Verzweigungen oder Zusammenführungen zu erleichtern, sollen sie analog zu obigem Vorgehen separat durch *Control-Events* realisiert werden, weil damit die für die logische Verteilung des Flusses verantwortlichen Events von anderen Aktionen innerhalb des Modellbausteins abgegrenzt werden können. Für einen standardisierten Modellbaustein ergibt sich damit eine feste innere Grundstruktur, wie sie in Abbildung 75 dargestellt wird. In logischer Reihenfolge des Ablaufs enthält er Input-Event (I), Delay-Event (D), Transform-Event (T), Control-Event (C) sowie Output-Event (O). Alle Event-Typen können mehrfach auftreten, insbesondere Input- und Output-

Events abhängig nach Anzahl der jeweiligen Input- und Output-Channel des Modellbausteins.

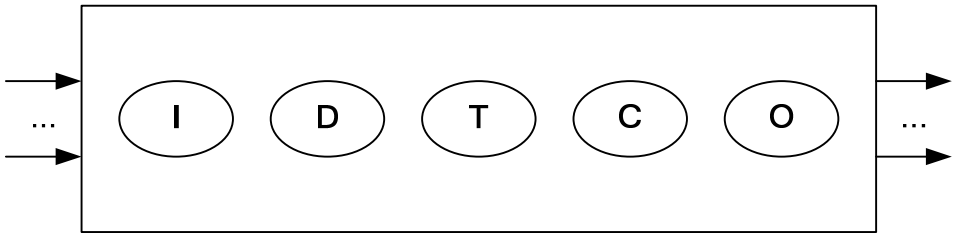


**Abbildung 75: Aufbau eines typischen Modellbausteins**

Durch die Einführung der neuen Event-Typen soll die automatische Invertierung gefördert werden. Darüber hinaus muss die inhaltliche Strukturierung als Modellierungsempfehlung an die Anwender kommuniziert werden, die hinsichtlich der Verhaltensmodellierung in erster Linie alle Freiheiten besitzen. Auch durch das Eigeninteresse des Anwenders gefördert, sollte er sich aber an diese Modellierungsempfehlung gebunden fühlen, um bei der Invertierung des Simulationsmodells wertvolle Modellierungszeit einzusparen. Tabelle 21 fasst die benötigten Erweiterungen der Modellbeschreibung zur Implementierung einer automatischen Invertierung von Modellbausteinen zusammen.

| Bezeichnung | Beschreibung |
|---|---|
| Delay_event | Ereignis zur Berechnung der zeitlichen Verzögerung der Token, ggf. in Abstimmung mit einem gültigen externen Kalender |
| Transform_event | Ereignis zur Manipulation des Token, bzw. zur Komposition/Dekomposition von Token |
| Control_event | Ereignis zur Steuerung des eigentlichen Materialflusses zwischen den verschiedenen Input- und Output-Channeln |

**Tabelle 21: Erweiterung der Modellbeschreibung durch Invertierung**

Zur Vollständigkeit soll an dieser Stelle auch auf die Umkehrung von Quellen und Senken eingegangen werden. Bei einer Rückwärtssimulation werden Quellen zu Senken und umgekehrt. Zu den in den Quellen erzeugten Token müssen im Rahmen der Invertierung aber weitere Überlegungen angestellt werden. Quellen versorgen ein Simulationsmodell mit einem Teil der Eingangsdaten (in Form von Art, Zeit und Quantität des erzeugten Flusses) und Senken können einen Teil der ausgegebenen Daten bereitstellen (in Form von Art, Zeit und Quantität des eingehenden Flusses). Diese Tatsache gilt analog für Vorwärts- wie Rückwärtssimulation. Dieser Fluss wird nach Maßgabe des Entwicklers erzeugt, muss aufgrund der unterschiedlichen Untersuchungszwecke von Vorwärts- und Rückwärtssimulation von der ursprünglichen Quelle aber soweit abweichen, dass diese durch eine neue Quelle und entsprechend durch eine neue Senke modelliert wird, weil sie neben den Eingabedaten auch die Auswertung vor dem Hintergrund eines unterschiedlichen Untersuchungszweckes ändert.

*Anwendung des Verfahrens zur Invertierung der Grundstrukturen*
Nachdem im vorangegangenen Abschnitt Überlegungen zur Typisierung möglicher Events und deren Umkehrung angestellt worden sind, werden die gefundenen

Lösungsalternativen im Folgenden auf die jeweiligen Grundstrukturen adaptiert. Bestimmte Fälle werden durch die Beschaffenheit einzelner Grundstrukturen ausgeschlossen. Die Grundstrukturen werden im Sinne der oben aufgezeigten Punkte untersucht, um geeignete Regeln für die Invertierung abzuleiten. Für alle Grundstrukturen gilt, dass Verzweigungen oder Zusammenführungen an den Input- oder Output-Channeln nicht betrachtet werden müssen, da diese nicht mehr eigentlicher Bestandteil der Grundstrukturen sind. Sie können somit in nachfolgenden Iterationsschritten der Zerlegung Teil anderer Grundstruktur werden.

Grundsätzlich werden bei der Invertierung einer Grundstruktur alle Input-Channel zu Output-Channeln und umgekehrt. Alle direkt dazu hinterlegten Events müssen angepasst werden. Das oder die neuen Input Events müssen dafür sorgen, dass der jeweilige Input-Channel verschlossen wird, wenn ein Token den Modellbaustein erreicht. Außerdem muss der Aufruf eines folgenden Events sichergestellt werden. Jedes Output-Event muss das Senden der Token an den nachfolgenden Input-Channel sicherstellen und ggf. den Aufruf des ReOpen-Events gewährleisten. Um die Invertierung zu ermöglichen, werden alle Input- und Output-Events standardisiert. Diese Ergänzung der Modellierungsempfehlung ist insofern konsequent, weil das logische Verhalten des Modellbausteins in den nachfolgenden Events innerhalb eines Bausteins (Control, Delay, Transform, etc.) gekapselt wird. Darüber hinaus werden Quellen durch Senken ersetzt und umgekehrt. Als nächster Schritt werden an jeder Bausteininstanz der Grundstruktur eingehende und ausgehende Token mit Hilfe der Bewertungsmatrix verglichen. Finden in einem Block zum Beispiel Montagevorgänge statt, so werden die jeweiligen Montageparameter aus dem entsprechenden Transform-Event abgefragt. Veränderungen von Token müssen in der invertierten Grundstruktur implementiert werden. Delay-Events können bei der Rückwärtssimulation zur Laufzeit abgefragt werden und müssen bei der Invertierung nicht verändert werden.

Da bei einer unverzweigten Linie keine Verzweigungen oder Zusammenführungen innerhalb der Struktur auftreten, reicht es, die oben beschriebene Vorgehensweise anzuwenden. Für Verzweigungen deckt sich die Vorgehensweise bei der Invertierung mit den allgemeinen Überlegungen. Typ- und Mengenverzweigungen können mit Hilfe von Bewertungsmatrizen invertiert werden, bei der Invertierung einer last- bzw. kapazitätsabhängigen Verzweigung müssen Abfertigungsprioritäten bei der Zusammenführung eingeführt werden. Das gilt analog für die Überlegung hinsichtlich der Zusammenführung: auch hier können die Vorüberlegungen übernommen werden. Das bedeutet, dass zwischen Mengen- und Typzusammenführungen unterschieden werden muss. Der Fall einer last- bzw. kapazitätsabhängigen Zusammenführung ist nicht relevant, da bei der Grundstruktur „Zusammenführung" vorher keine Verzweigung des Materialflusses stattfindet (sonst würde es sich um eine Grundstruktur höherer Ordnung handeln). Zur Invertierung können die benötigten Flussarten und –mächtigkeiten der Bewertungsmatrix entnommen werden. Die Grundstruktur *Kreuzung* stellt die Kombination der Strukturen Verzweigung und Zusammenführung dar. Daher gelten bei der Invertierung auch die jeweiligen Regeln. Typverzweigungen und -zusammenführungen werden beim bekannten Vorgehen identifiziert. Dieser Fall kann mit Bewertungsmatrizen umgekehrt werden. Ebenfalls mit Hilfe von Bewertungsmatrizen können Mengenverzweigungen oder –Zusammenführungen invertiert werden. Aus bereits bekannten Gründen tritt der last- bzw. kapazitätsabhängige Fall bei der

Zusammenführung nicht auf, da ihm keine Verzweigung vorangeht. Dies gilt jedoch nicht für die Verzweigung. Es kann also ein Mischfall auftreten, bei dem eine Mengenzusammenführung mit einer last- bzw. kapazitätsabhängigen Verzweigung kombiniert ist. Bei der Invertierung müsste dann eine Abfertigungspriorität für die Zusammenführung eingeführt werden. Die Grundstruktur *parallele Linie* (vgl. Abbildung 76) verfügt sowohl über eine Verzweigung als auch über eine Zusammenführung. Bei einer Rückwärtssimulation muss nun an Modellbaustein 3 über die Aufteilung des Materialflusses entschieden werden. Legt man die Vorüberlegungen zu Grunde, so muss bei dieser Struktur nach den drei Arten der Verzweigung unterschieden werden. Vorher kann festgestellt werden, ob es sich an Modellbaustein 1 um eine Typ- Mengen- oder last- bzw. kapazitätsabhängige Verzweigung handelt. Im Fall der Typverzeigung, kann bei Modellbaustein 3 der Materialfluss analog verteilt werden. Im Falle einer Mengenverzweigung oder einer last- bzw. kapazitätsabhängigen Verzweigung bietet es sich ebenfalls an, das entsprechende Control-Event aus Modellbaustein 1 in Modellbaustein 3 zu übertragen.

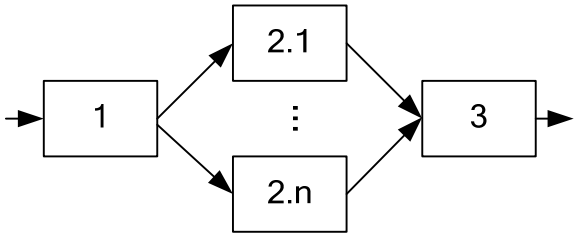


**Abbildung 76: Grundstruktur Parallele Linie**

Die Grundstruktur *Rückkopplung* (Abbildung 77) verbindet Verzweigung und Zusammenführung. Im Gegensatz zur *parallelen Linie* sind diese aber vertauscht: Die Verzweigung befindet sich an Modellbaustein 3, die Zusammenführung an Modellbaustein 1. Die Invertierung dieser Struktur funktioniert analog zur *parallelen Linie*. Handelt es sich bei der Verzweigung an Modellbaustein 3 um eine Typverzweigung, so kann entsprechend der Bewertungsmatrix während der Rückwärtssimulation der Materialfluss an Modellbaustein 1 verzweigt werden. Liegt an Modellbaustein 3 eine Mengen- oder last- bzw. kapazitätsabhängige Verzweigung vor, so muss bei der invertierten Struktur das entsprechende Control-Event aus Modellbaustein 3 in Modellbaustein 1 übertragen werden.

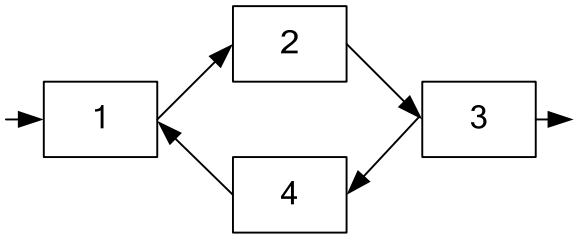


**Abbildung 77: Grundstruktur Rückkopplung**

Bei der Grundstruktur *Stern* (vgl. Abbildung 78) handelt es sich um eine Struktur, die Verzweigungen und Zusammenführungen miteinander verbindet. Im Gegensatz zur *parallelen Linie* sind alle Events, die Verzweigungen oder Zusammenführungen regeln in Modellbaustein 1 enthalten. Im Fall eines einfachen Sterns (in Abbildung 78 Teil a)

können lediglich Typ- oder Mengenverzweigungen auftreten. Diese sind mit Hilfe von Bewertungmatrizen umkehrbar. Komplizierter ist die Invertierung eines Sterns, wie ihn der rechte Teil der Abbildung zeigt (Teil b). Hier können zwei zusätzliche Fälle auftreten: Erstens last- bzw. kapazitätsabhängige Verzweigungen und zweitens eine Bearbeitungsreihenfolge. Im Fall von last- bzw. kapazitätsabhängigen Verzweigungen muss das Control-Event angepasst werden. Tritt eine feste Bearbeitungsreihenfolge auf, so muss diese umgekehrt werden. Dies könnte beispielsweise durch Analyse der Teilflüsse geschehen. Unterscheiden sich die Token nicht voneinander wird empfohlen, festgelegte Reihenfolgen entweder im Vorwärtssimulationsmodell bekannt zu machen, durch ein Transform-Event zu modellieren oder im Rahmen einer ersten Vorwärtssimulation aufzuzeichnen.

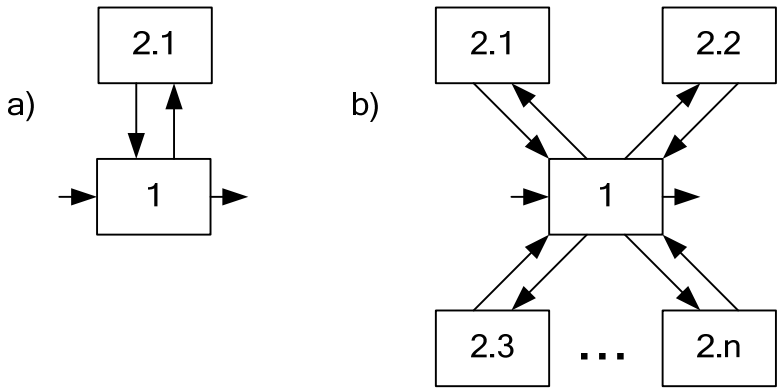


**Abbildung 78: Grundstruktur Stern**

Dieser Abschnitt hat ein Konzept entworfen, gemäß dem die zuvor identifizierten Grundstrukturen unter schwachen Restriktionen umgekehrt werden können. Durch dessen Anwendung ergibt sich eine Modellierungsempfehlung, an die sich der Anwender halten muss, um die automatische Transformation zu erreichen. Der folgende Abschnitt soll darauf basierend das Transformationsmodul skizzieren, das die angesprochene Zerlegung und Invertierung eines Simulationsmodells tatsächlich automatisch ausführt.

**Automatische Transformation von Simulationsmodellen**

Obiger Abschnitt hat eine Methode aufgezeigt, wie komplexe Materialflusssysteme in Grundstrukturen zerlegt und hinsichtlich einer Rückwärtssimulation invertiert werden können. Der folgende Abschnitt will eine automatische Modelltransformation als Untermodul der Modellierungskomponente entwerfen, mit deren Hilfe Grundstrukturen aus dem Simulationsmodellen extrahiert und invertiert werden können, um den Transformationsprozess eines Simulationsmodells von vorwärts in rückwärts gerichtetes Simulationsmodell weitestgehend automatisch zu gestalten. Basis der Invertierung ist stets ein vorliegendes Simulationsmodell, im Idealfall in Verbindung mit einem durchgeführten Simulationslauf und einer zugehörigen Bewertungsmatrix. Die Invertierung erfolgt durch ein spezielles Modul in der Modellierungskomponente, weil dort das geladene Simulationsmodell bereits vollständig vorliegt und zum Anderen aus der Zerlegung und Vereinfachung entstehende Restflüsse direkt durch den Anwender manipuliert werden können. Das resultierende Rückwärtssimulationsmodell kann aus der Modellierungskomponente heraus in der Simulationsdatenbank oder auf dem Dateisystem gesichert werden.

Der Transformationsprozess im Server des Modellierungstool soll wie folgt gestaltet werden: das geladene Simulationsmodell wird im Server in das Untermodul zur Transformation geladen und zunächst auf einen gerichteten Graph reduziert, wobei die eigentlichen Knoten des Graphen, die die Modellbausteine repräsentieren, um Informationen angereichert werden, die auf die Modellbausteine selbst verweisen. Zusätzlich werden, wenn vorhanden, die Ergebnisse eines Simulationslaufs mit diesem Modell und eine zugehörige Bewertungsmatrix in das Modul geladen. Alle weiteren Berechnungen zur Identifikation der Grundstrukturen finden zunächst auf dem gerichteten Graphen statt, werden an den entsprechenden Stellen durch Informationen aus der Bewertungsmatrix ergänzt. Nach dem Start des Transformationsprozesses werden als erster Schritt die Quellen und Senken des Materialflusses identifiziert und für die weitere Bearbeitung markiert. In einem iterativen Prozess wird der Graph in den Folgeschritten immer wieder auf der Suche nach Grundstrukturen durchlaufen. Zunächst sollen die Grundstrukturen einfacher Ordnung (einfache Linie, Verzweigung, etc.) gesucht werden. Erst im Folgeschritt werden Grundstrukturen höherer Ordnung erfasst. Ausnahme ist hier die Grundstruktur „Rückkopplung", die als erstes erkannt werden muss. Andernfalls würden die Bausteine 1 und 2 zu einer einfachen Linie zusammengefasst werden (vgl. Abbildung 51). Nach jeder Identifikation einer Grundstruktur wird diese zusammengefasst und der resultierende Graph (vgl. bspw. Abbildung 57) erneut hinsichtlich aller möglichen Grundstrukturen durchlaufen, bis entweder der gesamte Graph invertiert wurde (Das Simulationsmodell wird in einem einzigen Modellbaustein zusammengefasst) oder ein Restfluss übrig bleibt. Dieser wird an den oder die Clients übertragen und muss manuell invertiert werden. Hier bieten sich dem Anwender zwei Möglichkeiten: entweder er invertiert die verbliebenen Restflüsse manuell oder ergänzt den Restfluss des Simulationsmodells so durch Dummy-Bausteine, dass wiederum Grundstrukturen durch den Algorithmus erkannt werden können. Abbildung 79 zeigt eine kurze Übersicht über den Algorithmus *Merge*, wie er im Rahmen der Erkennung und Zusammenfassung von Grundstrukturen verwendet werden soll.

```
public Map<String, Integer> merge()
{
        Class[] mergers =
        {
        MergeRegeneration.class,
        MergeLine.class,
        MergeBranch.class,
        MergeJoin.class,
        MergeParallel.class,
        MergeStar.class,
        MergeCover.class };

        MergeAll mergeAll = new MergeAll(mergers, this);
        Map<String, Integer> counter = mergeAll.merge(model);
}
```

**Abbildung 79: Übersicht des Merge-Algorithmus**

Nachdem so das gesamte Simulationsmodell in einem Modellbaustein vereinfacht wurde, findet die Invertierung der identifizierten Grundstrukturen nach dem beschriebenen Verfahren statt, wobei zur Fallunterscheidung ggf. die vorliegende Bewertungsmatrix für die entsprechende Grundstruktur erzeugt wird, auf deren Basis die Verzweigungen und Zusammenführungen unterschieden werden können. Darüber hinaus können über die

Verweise in den Knoten des Graphen auch auf die Modellbausteine referenziert werden, wenn der Zugriff auf die strukturierten Events zur Invertierung einer Grundstruktur benötigt wird. Neben der Transformation einzelner Events in andere Modellbausteine einer Grundstruktur muss eventuell die Adaption der implementierten Verteilregeln bei der Aufteilung der Materialflüsse berücksichtigt werden. Prioritätsregeln[28] (z.B. First-In-First-Out) müssen bei der Rückwärtssimulation ggf. durch ihre Pendants ersetzt werden. In den meisten Untersuchungsfällen können die implementierten Regeln jedoch bestehen bleiben. Für die entsprechende Rückwärtssimulation müssen sie aber als Annahmen entsprechend berücksichtigt werden.

In einem letzten Schritt werden die ursprünglichen Quellen und Senken des Simulationsmodells aus dem resultierenden Simulationsmodell gelöscht und durch Platzhalter ersetzt, die im weiteren Verlauf durch den Anwender mit den neuen Quellen und Senken ersetzt werden müssen. Diese können nicht automatisch getauscht werden, weil sich die Untersuchungszwecke von vorwärts und rückwärts gerichteten Simulationsmodellen stark unterscheiden.

Zusammenfassend kann an dieser Stelle festgehalten werden, dass alle Vorraussetzungen für die Konzeption des Modellierungswerkzeugs vorliegen. Basierend auf der Festlegung einer Vorgehensweise zur Modellierung und Simulation wurde in diesem Abschnitt eine Modellbeschreibung entworfen, die alle an das Werkzeug gestellten Anforderungen abbilden kann. Zusätzliche Funktionsmodule zur Unterstützung speziell funktionsorientiert arbeitender Fertigungssysteme und zur weitestgehend automatischen Transformation von Simulationsmodellen in ihre entgegengesetzt gerichteten Pendants auf Basis eines Grundstrukturenkonzeptes wurden entworfen und den entsprechenden Teilmodulen des Werkzeugs zugeordnet. Darüber hinaus wurde die nachrichtenbasierte Kommunikationsschnittstelle zum Datenaustausch zwischen Simulator und Visualisierungskomponenten sowie zwischen den Teilmodulen des Simulators erstellt und steht für die Implementierung des Werkzeugs zur Verfügung. Als nächster Schritt sollen nun die eigentlichen Software-Module des Werkzeugs entwickelt und entsprechend detailliert werden.

## 5.3 Konzeption Modellierungswerkzeug

Im folgenden Abschnitt soll das Werkzeug im Rahmen eines Software-Entwicklungsprozesses konzipiert und modelliert werden. In einem ersten Schritt wird zunächst das System anhand der gestellten Anforderungen grob konzipiert, wobei ein Hauptaugenmerk auf der Strukturierung in Teilmodule besteht. Diese Phase, im Folgenden als Systementwurf bezeichnet, bildet die Basis für die Ausgestaltung der einzelnen Teilmodule.

### 5.3.1 Systementwurf

Die Betrachtung der Anforderungen für das Werkzeug basiert auf dem in 5.1.1 entwickelten Modellierungs- und Simulationsprozess. Bei näherer Betrachtung der

[28] Eine Übersicht und Klassifikation bekannter Prioritätsregeln kann beispielsweise in [FaFG94] oder [Teic98] gefunden werden.

einzelnen Anforderungen ergibt sich mittels der UML erstellte Use-Case-Diagramm, wie es in Abbildung 80 dargestellt ist.

Aus dem in Abbildung 80 dargestellten Use-Case lässt sich ersehen, dass noch keine explizite Unterscheidung zwischen verschiedenen Anwendergruppen gemacht wird. Zunächst werden dem Akteur *Anwender* alle Funktionen der Modellierung und Simulation ermöglicht. Beide Hauptaufgaben folgen bei der Durchführung einer Simulationsstudie hintereinander und können somit auch voneinander losgelöst betrachtet werden. Prinzipiell ergeben sich damit bereits zwei Module; das erste Modul unterstützt speziell den Prozess der Modellierung, das zweite Modul berechnet das Simulationsmodell auf Basis der Eingabewerte. Optional, aber nicht verpflichtend, kann der dynamische Markenfluss des Simulationsmodells visualisiert werden. Um die Visualisierung von der eigentlichen Berechnung im Simulatorkern loszulösen, werden die Aufgaben gemäß dem MVC-Pattern (vgl. Abschnitt 3.5.6) in verschiedene Module verteilt. Das bietet den Vorteil, dass unterschiedliche Visualisierungskomponenten mit dem Simulatorkern verbunden werden können. Neben diesen Hauptaufgaben werden Funktionen zur Administration der Anwender, Benutzergruppen und des Datenbestandes vorgesehen, die prinzipiell auch durch einen Akteur *Administrator* erledigt werden können und eine direkt Manipulation der Daten in der Simulationsdatenbank erlauben.

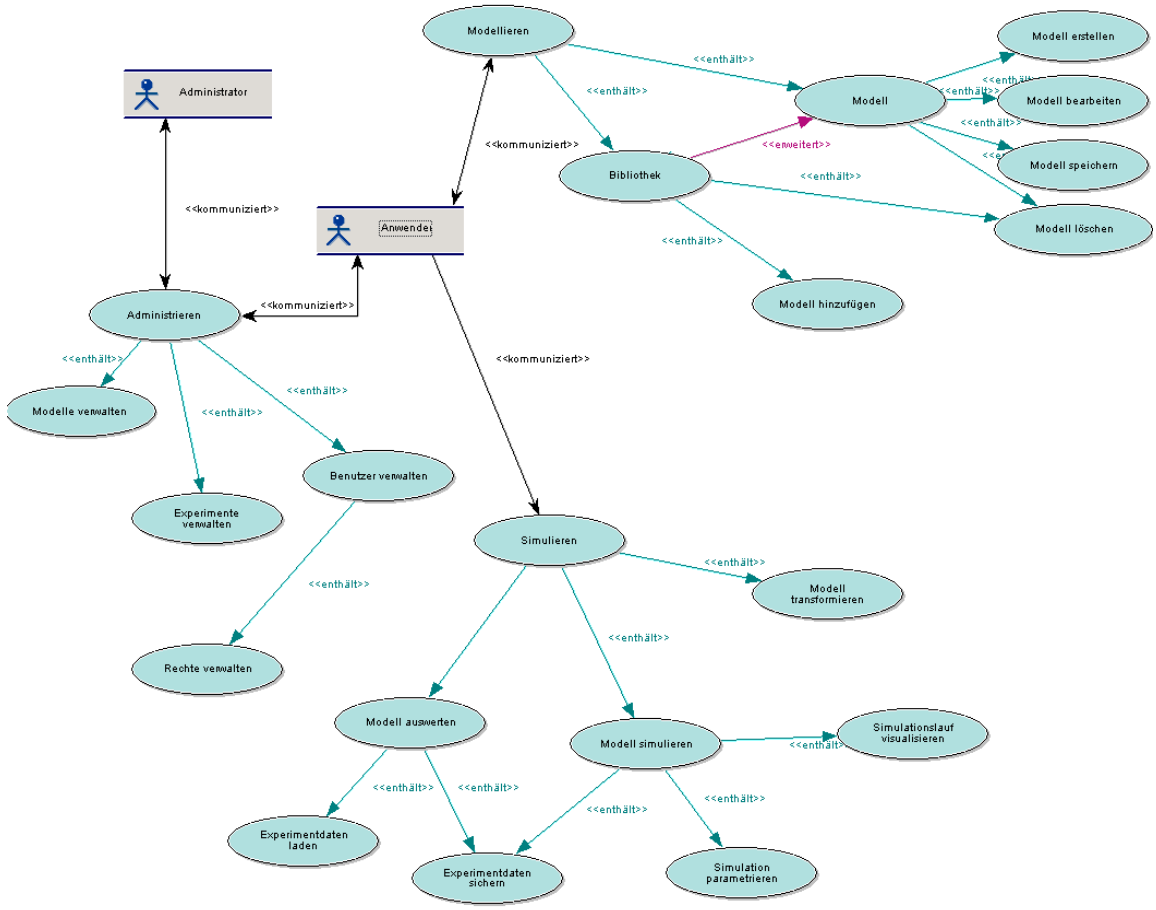


**Abbildung 80: Use-Case-Diagramm des Werkzeugs**

**Use-Case: Modellieren**

In der Modellierungsphase kann zwischen der Modellierung oder Bearbeitung einer Bausteinbibliothek oder der Modellierung oder Bearbeitung eines experimentierfähigen Simulationsmodells unterschieden werden. Da Bibliotheken auch eine geordnete Ansammlung einzelner Simulationsmodelle darstellen, die in einem experimentierfähigen Simulationsmodell verwendet werden können, unterscheidet sich die weitergehende Funktionalität nicht. Innerhalb dieses Use-Cases werden die üblichen Hauptfunktionen wie Erstellen, Bearbeiten, Speichern und Löschen eines Simulationsmodells angeboten. Darüber hinaus wird an dieser Stelle das Verfahren zur automatischen Invertierung von Vorwärtssimulationsmodellen in rückwärts gerichtete Simulationsmodelle in die Modellierungskomponente eingebettet, wie es unter Abschnitt 5.2.4.2 konzipiert wurde.

**Use-Case: Simulieren**
Gemäß Abschnitt 5.1.4 erfolgt vor dem Start der Berechnung des Simulationsmodells im Simulatorkern die Transformation der im XML-Format vorliegenden Modellbeschreibung in ein ausführbares Java-Programm. Anschließend kann die Berechnung des Simulationslaufs erfolgen, der optional durch verschiedene Visualisierungsmodule dargestellt und animiert werden kann. Darüber hinaus erhält der Anwender die Möglichkeit, während der Berechnung des Simulationslaufs interaktiv in die Berechnung einzugreifen und Parameter des Simulationsmodells zu verändern. Gegebenfalls werden die Manipulationsmöglichkeiten durch Voreinstellungen des Modellierers begrenzt. Während oder nach der Berechnung des Simulationsmodells können die gesammelten Informationen des Simulationslaufes ausgewertet und für eine weitere Verwendung gespeichert werden. Sofern die Simulationsmodelle dies unterstützen, führt die Bewegung des Anwenders in der Visualisierungskomponente zu einer dynamischen Detaillierung des Simulationsmodells. Verwendet das Simulationsmodell die automatische Wegberechnung des Motion Planning Moduls, werden diese zur Ausführungszeit eines Simulationslaufs im Simulator berechnet und visualisiert.

**Use-Case: Experimentieren**
In der Experimentierphase einer Simulationsstudie steht dem Anwender ein weiteres Modul des Werkzeugs zur Verfügung mit dem mehrere Simulationsläufe eines dort anzulegenden Simulationsexperimentes mit den entsprechenden Parametern versehen und in einem Stapelverarbeitungs- oder Parallelbetrieb ausgeführt werden. Neben der Darstellung der Eingangsparameter unterstützen spezielle Funktionen des Moduls den Anwender, beispielsweise bei der Variation der Startwerte der Zufallsverteilungen innerhalb der Bausteininstanzen des Simulationsmodells über die verschiedenen Simulationsläufe hinweg. Nach Durchführung aller Simulationsläufe können die generierten Experimentdaten in einer kurzen Darstellung betrachtet und als Simulationsexperiment in der Simulationsdatenbank oder auf dem Dateisystem gespeichert werden.

**Use-Case: Administrieren**
Die Administration der anfallenden Daten und deren Verwaltung soll über ein spezielles Modul erfolgen. Einzelne Funktionen werden nur dann in die Modellierungs-, Simulations- und Visualisierungsmodule integriert, wenn sie den Anwender bei seiner Arbeit direkt unterstützen. Damit können elementare Aufgaben wie beispielsweise die Verwaltung der Anwender, Benutzergruppen, Rechteverwaltung und die Administration der Simulationsdatenbank potentiell auch von Nicht-Simulationsexperten ausgefüllt werden.

**Grobstruktur der Funktionsmodule**

Aus der Darstellung und Entwicklung der Use-Cases ergibt sich das bereits bekannte Schema von Funktionsmodulen, aus denen das Gesamtsystem bestehen soll. Die präzisere Darstellung der Funktionen über die einzelnen Funktionalitäten in den Use-Case-Diagrammen hat die geplante Modularisierung des Werkzeugs nochmals bestätigt. Neben einer Modellierungskomponente und dem Simulatorkern beinhaltet das Werkzeug Module zur Visualisierung von Simulationsläufen, eine Simulationsdatenbank und ein Modul zur Administration der Daten. Abbildung 81 gibt einen Überblick über die Funktionsmodule und deren Beziehungen untereinander.

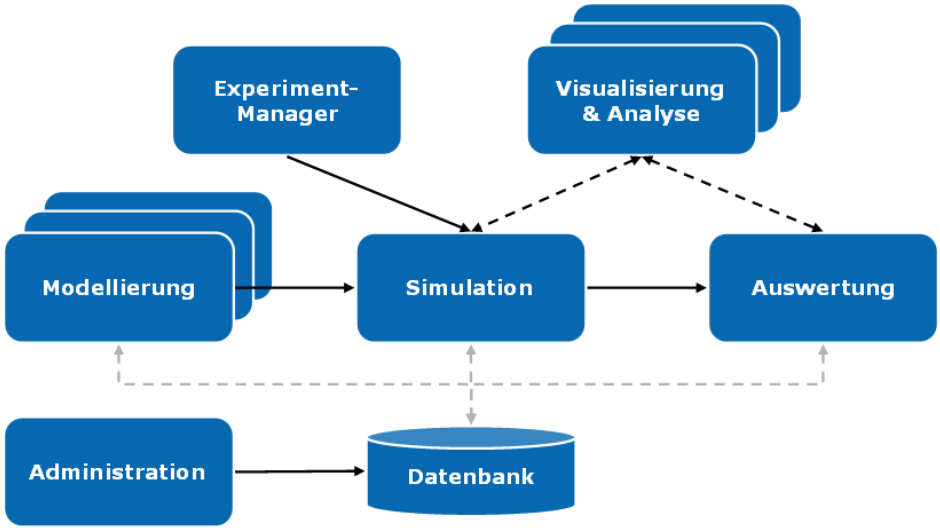


**Abbildung 81: Grobstruktur der Funktionsmodule des Werkzeugs**

Die Darstellung der Funktionsmodule gibt noch keine Auskunft darüber, wie die einzelnen Funktionen mit Bedienoberflächen versehen werden. Hinsichtlich der verschiedenen Aufgabestellungen sind hier unter Umständen alternative Visualisierungsmöglichkeiten zu evaluieren und umzusetzen. Die Grobstrukturierung des Basisprozesses wurde durch den Systementwurf also bestätigt und nur in Teilen erweitert. Im Folgenden sollen die identifizierten Module präziser ausgeplant werden.

## 5.3.2 Entwurf der Funktionsmodule

Im folgenden Abschnitt sollen die einzelnen Funktionsmodule genauer hinsichtlich ihrer Aufgabe und Beschaffenheit spezifiziert werden. Die Reihenfolge des Entwurfs orientiert sich an der Vorgehensweise des Anwenders bei der Arbeit mit dem Werkzeug.

### 5.3.2.1 Modul Modellierung

Zur Umsetzung des Moduls zur Modellierung der Simulationsmodelle nach der in Abschnitt 5.2 beschriebenen Modellbeschreibung wird auf das in Abschnitt 3.5.6 vorgestellte Architekturmuster Client/Server zurückgegriffen, da es sich besonders gut für die Implementierung eines multitaskingfähigen Werkzeugs eignet. In Kombination mit dem Modul Simulationsdatenbank ergibt sich eine 3-Tier-Architektur, bei der auf Seite der Serverschicht zusätzlich der Simulatorkern platziert werden kann.

Die Aufgabe des Servers besteht in der Verwaltung der angeschlossenen Anwender, bzw. Clients sowie der Wahrung der Konsistenz des oder der aktuell bearbeiteten Simulationsmodelle. Zur besseren Kooperations- und Kollaborationsunterstützung kann der Server verschiedene Kommunikationsmöglichkeiten zur Verfügung stellen, damit sich die an der Modellierung beteiligten Anwender während ihrer Arbeit abstimmen können. Darüber hinaus muss der Server eine Möglichkeit vorsehen, die Benutzerverwaltung zu implementieren, so dass sich die Anwender an ihren Clients vor dem Zugriff auf die Simulationsmodelle authentifizieren und damit auch autorisieren müssen. Die Umsetzung der Rechteverwaltung aus Abschnitt 5.1.6 wird angestrebt, so dass der Zugriff auf ein Simulationsmodell auch gestaffelt hinterlegt werden kann. Der Server muss jeweils entscheiden, ob die vom Client angeforderte Interaktion (beispielsweise Hinzufügen eines Bausteins, Löschen, Verschieben, etc.) auf Basis der zugrunde liegenden Rechte ausgeführt werden darf oder nicht. Hierfür sind während der Implementierung zwischen Client und Server geeignete Schnittstellen zu entwerfen. In den Modellierungsserver soll auch das Verfahren zur automatischen Invertierung eines Simulationsmodells integriert werden. Weil hier die aktuell bearbeiteten Simulationsmodelle hinterlegt werden, kann das resultierende Simulationsmodell zur weiteren Bearbeitung für den oder die Anwender direkt zur Verfügung stehen.

Jeder Anwender bearbeitet das Simulationsmodell über einen eigenen Client, der verschiedene Visualisierungsmöglichkeiten anbietet. Neben einer zweidimensionalen Visualisierung soll zusätzlich eine dreidimensionale Modellierungsform angeboten werden. Idealerweise wird diese für weniger leistungsstarke Clients um eine 2½D-Visualisierung ergänzt, die ein realistisches Anordnen der Modellbausteine mittels einer Parallelprojektion von schräg oben in der virtuellen Umgebung ermöglicht. Abbildung 82 zeigt die resultierende Architektur des Modellierungsmoduls.

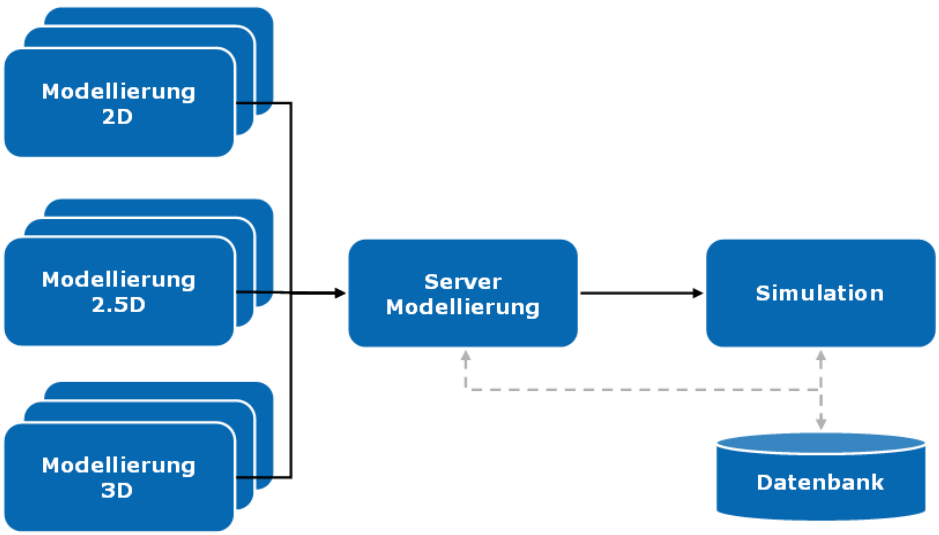


**Abbildung 82: 3-Tier- Architektur des Modellierungsmoduls**

Abbildung 83 beschreibt in einem Sequenzdiagramm die intendierte Bearbeitungs-reihenfolge innerhalb des Modellierungsmoduls.

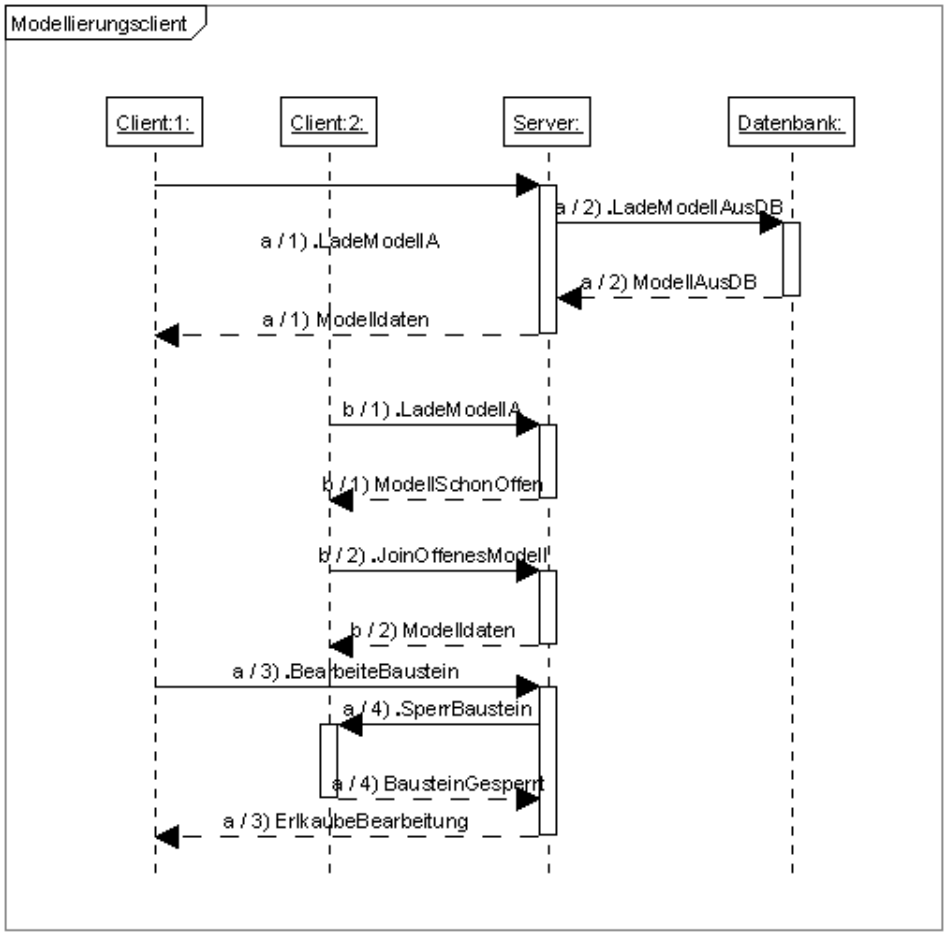


**Abbildung 83: Sequenzdiagramm des Modellierungstools**

Der Anwender kann aus seinem Bearbeitungsclient ein in der Simulationsdatenbank hinterlegtes Simulationsmodell zur Weiterbearbeitung öffnen oder legt ein neues Simulationsmodell an (Analog kann mit der Modellierung von Bausteinbibliotheken verfahren werden). Weitere angeschlossene Anwender können diesem Simulationsmodell beitreten, um gemeinsam an demselben Modell zu arbeiten. Alternativ können sie eine eigene Instanz dieses oder eines anderen Simulationsmodells öffnen. Der Server lädt das Simulationsmodell mit allen Parametern und 2D-, 2½D- und/oder 3D-Repräsentanten der Modellbausteine aus der Simulationsdatenbank und stellt sie den Clients zur Darstellung zur Verfügung. Während der gleichzeitigen Bearbeitung eines Simulationsmodells durch mehrere Anwender synchronisiert der Server die Zugriffe auf der Ebene eines Bausteininstanz, bzw. hinsichtlich der über- wie untergeordneten Hierarchieebenen des selektierten Modellbausteins. Abbildung 84 zeigt schematisch den Sperrmechanismus des Servers.

Der Server verteilt die jeweiligen Informationen über Sperrzustände der einzelnen Bausteininstanzen eines Simulationsmodells an die angeschlossenen Clients, damit diese hier entsprechend dargestellt werden können. Der Server muss nicht zwangsläufig eine grafische Oberfläche besitzen, könnte aber damit zusätzliche Informationen darstellen, die eine Analyse durch die entsprechenden administrativen Stellen ermöglichen. Der Arbeitsprozess der Anwender soll durchgehend mittels Kommunikationsmechanismen wie

Chat[29], UserList[30], User-Awareness[31] und E-Mail unterstützt werden, so dass ein verteiltes, ortsunabhängiges Arbeiten erlaubt wird. Jeder Client kann seine eigene Darstellungsform aus den vorhandenen Möglichkeiten auswählen. Die jeweiligen Darstellungsrepräsentanten werden vom Server an den Client übertragen. Dank der verwendeten Client-Server-Architektur muss diese Informationen nur einmal aus der Simulationsdatenbank ausgelesen werden und kann danach im Server für weitere Anwender vorgehalten werden.

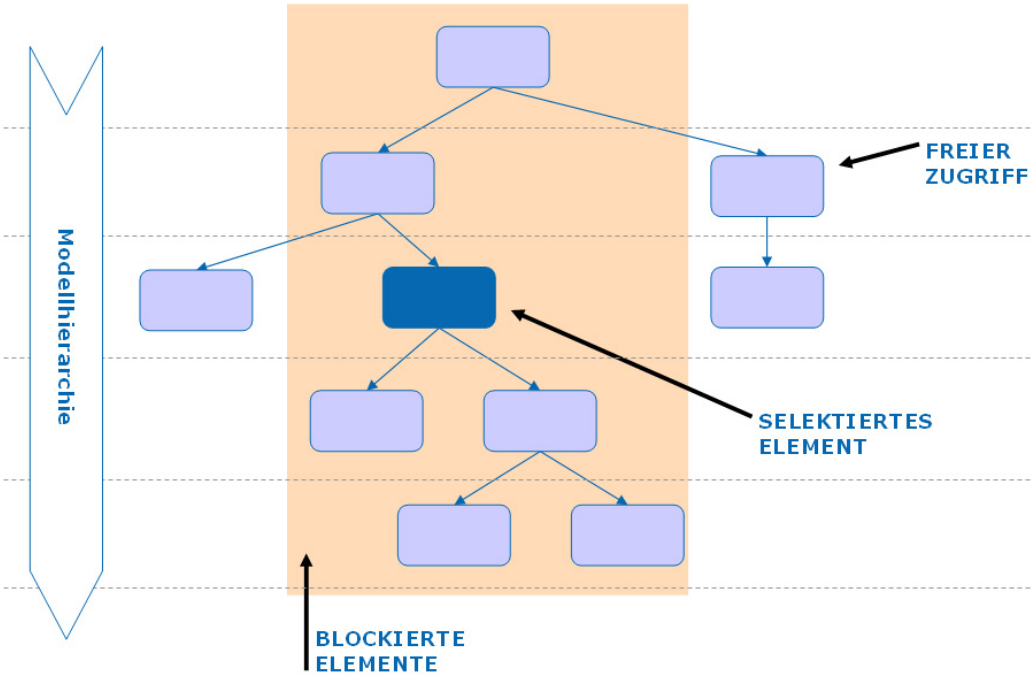


**Abbildung 84: Sperrmechanismus des Modellierungsservers**

Der Anwender kann seine Bearbeitung abschließen, indem er das Simulationsmodell aus seinem Client heraus sichert. Der Server hinterlegt den aktuellen Modellzustand in der Simulationsdatenbank, wo ihm eine aufsteigende Versionsnummer zugewiesen wird. Das Modell wird um Informationen über Anwender, spezielle Kommentare, etc. angereichert und gesichert, so dass jedem Anwender die Version des Simulationsmodells wiederhergestellt werden kann, die er gesichert hat. Die Clients müssen über den Server eine Möglichkeit bereitstellen, den Anwender auf vorhandene, neuere Versionen seines Simulationsmodells hinzuweisen. Zum Schutz vor unbefugtem Zugriff auf die in der Simulationsdatenbank hinterlegten Simulationsmodelle muss der Server eine verschlüsselte Kommunikation mit den angeschlossenen Clients zumindest optional unterstützen. Im Fall einer gemeinschaftlichen Simulationsstudie innerhalb eines

[29] Chat (von engl. to chat) ist die Bezeichnung für eine innerhalb des Internet weit verbreitete Art der interaktiven Kommunikation zwischen zwei oder mehreren Personen in Echtzeit [Balz05].

[30] Die UserList ist meist eine Erweiterung der reinen Chat-Funktion, die es dem Anwender erlaubt, Kollegen und Bekannte in selbst zu definierende Gruppen einzusortieren, um deren Adresse schneller wieder finden zu können.

[31] Chat-Programme erlauben dem Anwender die Angabe eines Anwesenheitsstatus, der allen anderen Anwendern in deren UserList angezeigt wird, damit diese erkennen, ob der Anwender als Kommunikationspartner zur Verfügung steht, stehen kann oder stehen möchte.

Intranets eines abgeschlossenen Unternehmensbereichs kann auf diese zusätzliche Sicherheitsrestriktion verzichtet werden.

Unter Berücksichtigung des Factory-Paterns[32] soll die grafische Benutzeroberfläche des Clients alle vorhandenen Visualisierungsmöglichkeiten in einer Oberfläche bereitstellen. Sie wird dazu zunächst in drei Bereiche aufgeteilt. Abbildung 85 zeigt eine schematische Darstellung der indentierten Benutzeroberfläche mit den Bereichen *menu*, *library* und *workplace*.

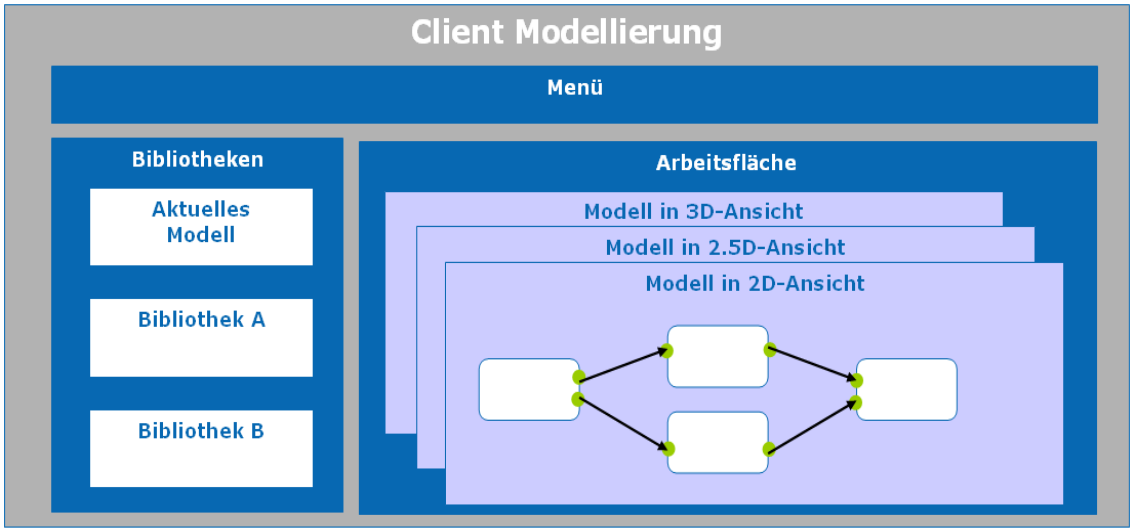


**Abbildung 85: Schematische Darstellung des Modellierungsclients**

Die beiden Bereiche *menu* und *library* sind unabhängig von der gewählten Visualisierungsform zur Darstellung des Simulationsmodells. Lediglich der „workplace", in dem die eigentliche Bearbeitung des Simulationsmodells erfolgt, unterscheidet sich je nach gewählter Visualisierungsform (2D-, 2½D- oder 3D-Darstellung). Durch die Implementierung des *Listener*-Konzepts[33] in Kombination mit der eigentlichen Datenstruktur des Simulationsmodells auf dem Server kann aber jederzeit zwischen der gewählten Visualisierungsform gewechselt werden. Dazu müssen lediglich die entsprechenden grafischen Repräsentanten der Modellbausteine vom Server nachgeladen werden. Die im Client hinterlegten Maus-, Tastatur- oder Menübefehle führen unabhängig von der Visualisierungsform zum Aufruf desselben Menüs. So kann eine einheitliche Bearbeitung unabhängig von der gewählten Darstellung erreicht werden (Bsp.: Ein Menü zur Verwaltung der Variablen eines Modellbausteins erscheint immer gleich und verändert sich nicht, wenn es aus 2D- oder 3D-workplace aufgerufen wird). In den zu entwickelnden Menüs müssen minimal die Parametrierung der Variablen und der Ereignisse des selektierten Modellbausteins ermöglicht werden. Darüber hinaus muss der *workplace* die Möglichkeit anbieten, die aus der Library per Drag&Drop[34] im Modell instanziierten Modellbausteine durch logische Verknüpfungen zwischen den Channeln zu

[32] [LaRa06]

[33] [LaRa06]

[34] Drag & Drop (Ziehen und Fallenlassen) ermöglicht das Verschieben von Daten mittels Ziehen und Fallenlassen und erlaubt dadurch den einfachen und effizienten Datenaustausch … zwischen verschiedenen Anwendungen[Balz05].

verbinden. Die *library* stellt geladene Simulationsmodelle und/oder Bausteinbibliotheken aus der Simulationsdatenbank zur Verfügung, damit der Anwender bei der Erstellung des Modells bereits vorhandene Modellbausteine verwenden kann. Die Verwendung dieser Architektur bietet durch den modularen Aufbau den Vorteil, den Modellierungsclient effizient erweitern zu können. Bei Bedarf können weitere Darstellungsformen ergänzt werden, wobei Änderungen an Menüs oder Eigenschaftsfenstern immer nur an einer zentralen Stelle eingepflegt werden müssen. Für den Anwender ergibt sich darüber hinaus der Vorteil einer einheitlichen Bearbeitung, unabhängig von der gewählten Darstellungsform, zwischen denen fließend gewechselt werden kann.

#### 5.3.2.2 Modul Simulatorkern

Wie in Abschnitt 5.1.2 beschrieben, muss bei dem gewählten Basisprozess vor Ausführung eines Simulationslaufs das im XML-Format vorliegenden Simulationsmodell in ein lauffähiges Java-Programm transformiert und kompiliert werden, um gemeinsam mit den Funktionen des Simulatorkerns ein einziges ausführbares Programm zu bilden. Dieses Vorgehen bietet neben dem erhofften Geschwindigkeitsvorteil bei der Ausführung eines Simulationslaufes weitere Vorteile. Einerseits kann ein Simulationslauf somit ohne Visualisierungsoberfläche berechnet und ausgeführt werden. Einzelne Simulationsläufe können potentiell in einem Netzwerk in Form eines Simulator-Simulationsmodell-Paketes verteilt und mit unterschiedlichen Parametersätzen von Eingabedaten gestartet werden. Andererseits kann das Funktionsmodul Simulatorkern wesentlich schlanker und somit effizienter implementiert werden. Die Anforderung an ein multitaskingfähiges Mehrbenutzersystem kann im Simulatorkern leicht dadurch erfüllt werden, dass ein Ankoppeln mehrerer Visualisierungskomponenten während der Ausführung erlaubt wird. Technisch gesehen entsteht in der Verarbeitung der ankommenden Interaktions-Nachrichten kein Unterschied, ob diese von einem Anwender oder von mehreren Anwendern erzeugt werden. Das Teilmodul zur Nachrichtenverwaltung im Simulator muss dann die Verwaltung der eingehenden Nachrichten nach Anwendern sortiert unterstützen. Kontextbezogen müssen Regeln implementiert werden, die eine inhaltliche Abstimmung der angeschlossenen Anwender unterstützen. Das Kommunizieren von im Simulatorkern berechneten Änderungen am Simulationsmodell über ein Nachrichtenformat kann ähnlich effizient an einen, wie auch an mehrere angeschlossene Visualisierungskomponenten erfolgen. Abbildung 86 zeigt die einzelnen Funktionsbereiche des Simulatorkerns in einem schematischen Diagramm.

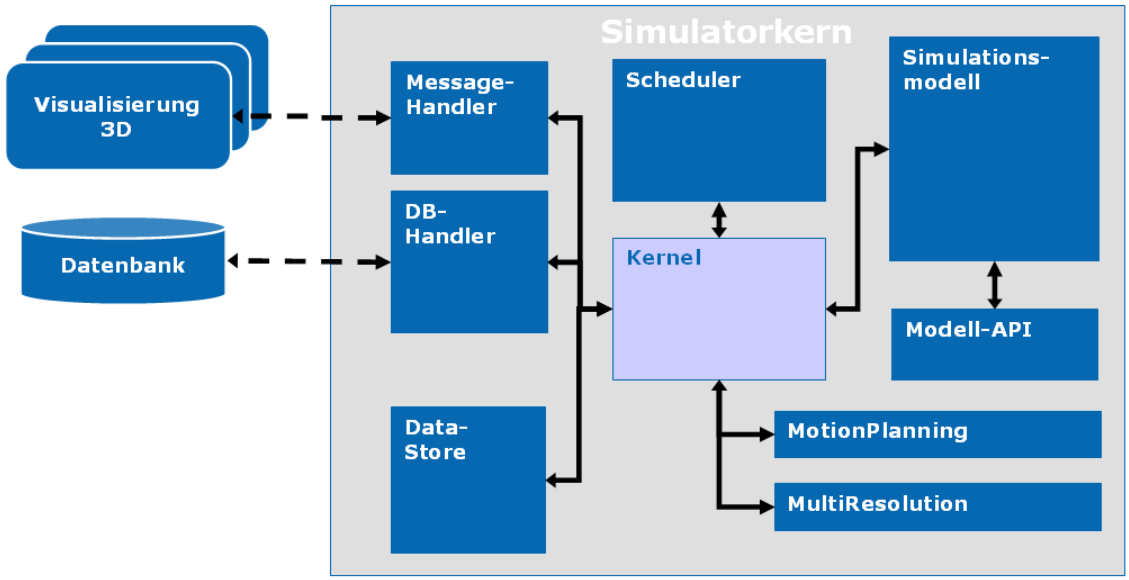


**Abbildung 86: Funktionsbereiche des Simulatorkerns**

Obige Abbildung zeigt, dass sich der Simulatorkern aus mehreren Teilmodulen zusammensetzt, die während der Ausführung eines Simulationslaufs ineinander greifen. Der wesentliche Teil des Simulators ist der *Scheduler* mit der zugehörigen Ereignisliste, in die zu berechnende Ereignisse des Simulationsmodells zur Laufzeit sortiert eingefügt werden (vgl. Abschnitt 3.3). Die Ausführung der diskreten Simulation erfolgt nach den in Abschnitt 2.2 beschriebenen Methoden, wobei die Datenstruktur des parametrierten Simulationsmodells ein objektorientiertes Datenmodell mit Objektklassen und –Instanzen bildet. Diese werden auf Basis eines Interfaces erzeugt, das als eigenständiges Paket im Simulatorkern implementiert werden muss. Hier werden insbesondere die Attribute und Methoden definiert, die der Simulatorkern zur Laufzeit ausführen kann, um die Daten des Simulationsmodells zu manipulieren, weitere Ereignisse zu instanziieren und dem *Scheduler* hinzuzufügen. Zur Verwaltung der erzeugten Datenstrukturen des Simulationsmodells wird im Simulator ein *Datastore* benötigt, an dem sich alle verwendeten Instanzen des Simulationsmodells während ihrer Erzeugung anmelden, um einen Zugriff auf die benötigten Daten aus dem Simulator heraus gewährleisten zu können. Manipulationen aus den Visualisierungskomponenten werden an der Kommunikationsschnittstelle des Simulators, dem *MessageHandler*, verarbeitet. Alle ankommenden Nachrichten werden hier gefiltert, verarbeitet und manipulieren ggf. den *Datastore* oder den *Scheduler*. Für die benötigte Transformation von XML-Modell in eine Java-Objekthierarchie wird zusätzlich ein *Preprocessor* benötigt, der die vorliegenden Modellbeschreibungen in ein Java-Programm parst und das Ergebnis kompiliert.

Die Modellierung und Simulation von zeitorientierten Simulationsmodellen soll mit dem Werkzeug prinzipiell über zwei verschiedene Arten ermöglicht werden:

1. Zeitorientierte Bausteinbibliotheken
   Die Modellierung zeitorientierter Simulationsmodelle erfolgt in diesem Fall über Bausteinbibliotheken, die eine entsprechende Zeitfortschaltung implizit unterstützen. Die zu terminierenden Ereignisse werden *bausteinintern* auf die vorhandenen Zeitinkremente abgebildet. Die Ausführung des Simulationsmodells im Simulatorkern kann dann ereignisorientiert erfolgen, weil die Terminierung der Ereignisse im

*Scheduler* bereits zeitorientiert erfolgt. Jeder Bausteininstanz kann während der Modellierung ein eigener Kalender zugewiesen werden (beispielsweise ein Schichtkalender), wodurch im abgebildeten System die einzelnen Elemente zwar zeitorientiert, aber nach jeweils variablen Intervallen simuliert werden können. Dieses Vorgehen entspricht am ehesten den Sichten in Fertigungsleitständen und kann auf Basis der abstrakteren Simulationsmodelle zur Integration innerhalb der Fertigungslenkung verwendet werden.

2. Zeitorientierte Berechnung im Simulatorkern
   Hier kann die zeitorientierte Ausführung des Simulationsmodells auf Basis eines ereignisdiskreten Simulationsmodells erfolgen. Der Simulator überstützt dies über einen speziellen „Modus Zeitorientierung", bei dem alle auftretenden Ereignisse *kernelintern* auf fixe Zeitinkremente abgebildet werden. Dieses Vorgehen entspricht der unter Abbildung 8 gezeigten Vorgehensweise der Zeitfortschaltung mit fixen Zeitinkrementen. Am Anfang der Simulation kann vom Anwender der Abstand der Zeitinkremente parametriert werden. Vorteil dieser Methode ist die Wiederverwendung der ereignisorientierten Simulationsmodelle, allerdings hängt die Genauigkeit der Simulation stark von den gewählten Zeitinkrementen ab.

Darüber hinaus ergeben sich aus der Aufgabenstellung weitere Teilfunktionen, die als zusätzliche Funktionen in dem Simulator eingebettet werden müssen. Hier sind insbesondere das Motion Planning Modul (vgl. Abschnitt 5.2.1.2) und das Modul zur dynamischen Adaption des Detaillierungsgrades des Simulationsmodells (vgl. Abschnitt 5.2.1.1) zu erwähnen, die jedoch nur dann aktiv dem Simulatorkern hinzugefügt werden sollen, wenn sie in dem zu bearbeiteten Simulationsmodell benötigt werden. Für das Laden und Speichern von Simulationsmodell und Simulationsexperimenten existiert mit dem Experiment-Manager ein separates Modul, das nachfolgend genauer entworfen wird.

#### 5.3.2.3 Modul Experimentmanager

Das Modul Experimentmanager soll den Anwender in Form eines durch die Anwendung leitenden Wizards bei der Durchführung einer Simulationsstudie unterstützen, indem an einer zentralen Stelle alle zur Durchführung eines Simulationsexperimentes benötigten Simulationsläufe parametriert und entweder in einem Stapelverarbeitungs- oder in einem Parallelbetrieb abgearbeitet werden können. Zu Beginn jeder Bearbeitung soll der Anwender ein Simulationsexperiment komplett neu anlegen, auf Basis eines Szenarios neu anlegen oder aus der Simulationsdatenbank laden. Daraufhin sollen die Grundeinstellungen hinsichtlich Simulationszeit- und Dauer, Anzahl der Simulationsläufe, etc. angegeben werden. Für die eigentliche Parametrierung sollen zahlreiche Unterstützungsfunktionen implementiert werden, die beispielsweise die automatische Parametrierung und Verwirbelung der Startparameter aller Zufallszahlen eines Simulationsmodells erledigen. Einzelne oder mehrere Parameter der Bausteininstanzen sollen in einer zentralen Übersicht eingestellt werden, wobei die Ansicht über entsprechende Masken oder Filter auf die relevanten Bereiche eines Simulationsmodells eingegrenzt werden soll. Nach der Einstellung der Ausführungsparameter der einzelnen Simulationsläufe (hintereinander oder parallel auf Rechner 1 bis Rechner N) kann der Anwender das Experiment starten und erhält nach dessen Berechnung eine tabellarische Übersicht über die generierten Ergebnisse. Das Simulationsexperiment inklusive der Ergebnisse kann dann in der Simulationsdatenbank oder auf dem Dateisystem gespeichert werden. Das Protokoll der einzelnen Simulationsläufe kann verwendet

werden, um sich im Nachhinein einzelne Ausschnitte aus den Simulationsläufen visualisieren zu lassen.

#### 5.3.2.4 Modul Visualisierungskomponente

Bei der Konzeption des Visualisierungsmoduls wird auf das bereits im Modellierungsmodul erfolgreich angewendete Architekturpattern Client-Server zurückgegriffen (vgl. Abschnitt 3.5.6 ). Der Simulatorkern übernimmt die eigentliche Serverfunktion. Neben bereits angesprochenen Darstellungsformen (2D-, 2½D- und 3D-Darstellung) soll als weitere Visualisierungsmöglichkeit eine Report-Oberfläche angeboten werden. Unabhängig von der layoutgetreuen Darstellung des Simulationsmodell in den verschiedenen Darstellungsformen soll mit dieser Visualisierungsmöglichkeit eine Analysefunktion bereitgestellt werden, die eine aggregierte Sicht auf die aktuellen Kennzahlen der einzelnen Modellbausteine des simulierten Modells ermöglicht. Es erlaubt eine schnelle Übersicht und damit Einschätzung des aktuellen Modellverhaltens während des Simulationslaufs und soll hauptsächlich im Rahmen der Modellvalidierung und –Verifikation eingesetzt werden. Die Report-Oberfläche ist in der Lage den gesamten Simulationslauf in einem *RecordSet* zu sichern und jederzeit wieder abspielen zu können. Damit wird eine weitere Form der Analyse von Simulationsmodellen ermöglicht, indem verschiedene Anwender ein *RecordSet* gemeinsam analysieren und damit die Entwicklung des Simulationsmodells oder sogar des abgebildeten Fertigungssystems voranzutreiben.

Report-Oberfläche, 2D- und 2½D-Oberfläche können mit wenigen Anpassungen und Erweiterungen in den Client des Modellierungsmoduls integriert werden, so dass eine einheitliche Werkzeugoberfläche zur Modellierung und Simulation zur Verfügung steht. Der Server muss zusätzlich um eine bidirektionale Schnittstelle (*MessageCenter*) zur Verarbeitung der Animationsnachrichten, Kommunikation der Änderungen von Parameterwerten und Interaktionen erweitert werden, um die Animation während der Ausführung eines Simulationsmodells in den Visualisierungskomponenten zu ermöglichen und die Interaktion mit dem Simulatorkern zu unterstützen. Dabei wird das in Abschnitt 5.2.2 entwickelte Nachrichtenformat umgesetzt. Weil der Server bereits die Multitasking-Bearbeitung von Simulationsmodellen unterstützt, muss in diesem Bereich keine weitere Anpassung vorgenommen werden. In den hier abgebildeten Visualisierungsformen nimmt der Anwender keine immersive Rolle ein, sondern betrachtet das abgebildete Fertigungssystem, ähnlich wie in kommerziellen Lösungen, als externer Betrachter. Speziell diese Darstellungsformen können eine gute Immersion nur schwer erzeugen. Deswegen können die Anforderungen hier auf den interaktiven Anteil beschränkt werden. Durch die Unterstützung des Nachrichtenformats in den entsprechenden Erweiterungen ist der Anwender in der Lage, Interaktionen auszuführen, die an den Simulatorkern übertragen und dort verarbeitet werden (vgl. Abbildung 87).

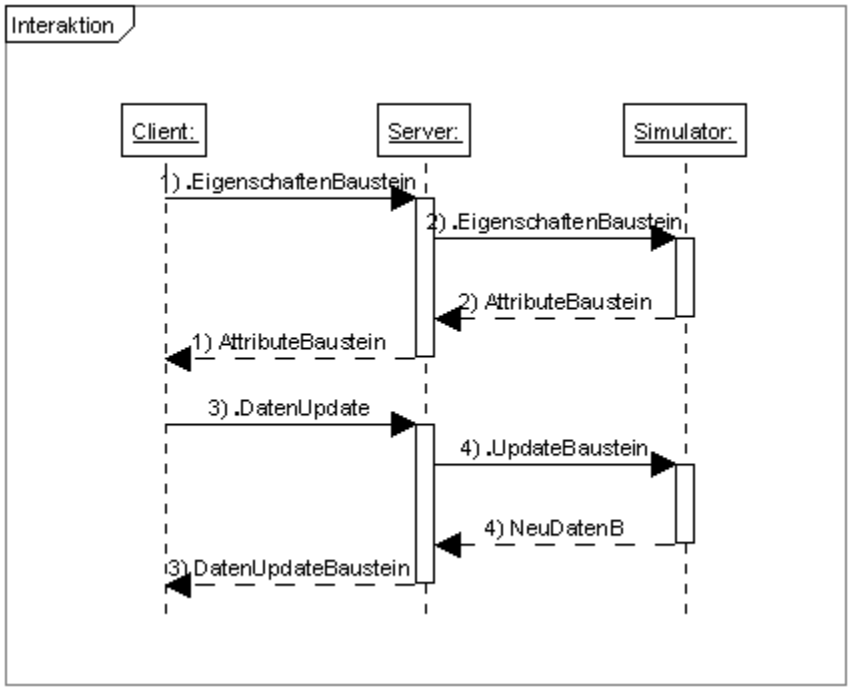


**Abbildung 87: Sequenzdiagramm einer Interaktion während der Simulation**

Hier greift, wie schon bei der Modellierung, die implementierte Benutzerverwaltung, so dass eine Abstufung innerhalb der Interaktionsmöglichkeiten gemäß der assoziierten Rechte umgesetzt werden kann (damit wird auch die Möglichkeit eines reinen Viewers erlaubt, bei dem der Anwender reiner Betrachter ist). Abbildung 88 zeigt die resultierende Architektur nach Erweiterung des Modellierungsmoduls um die beschriebenen Visualisierungsformen.

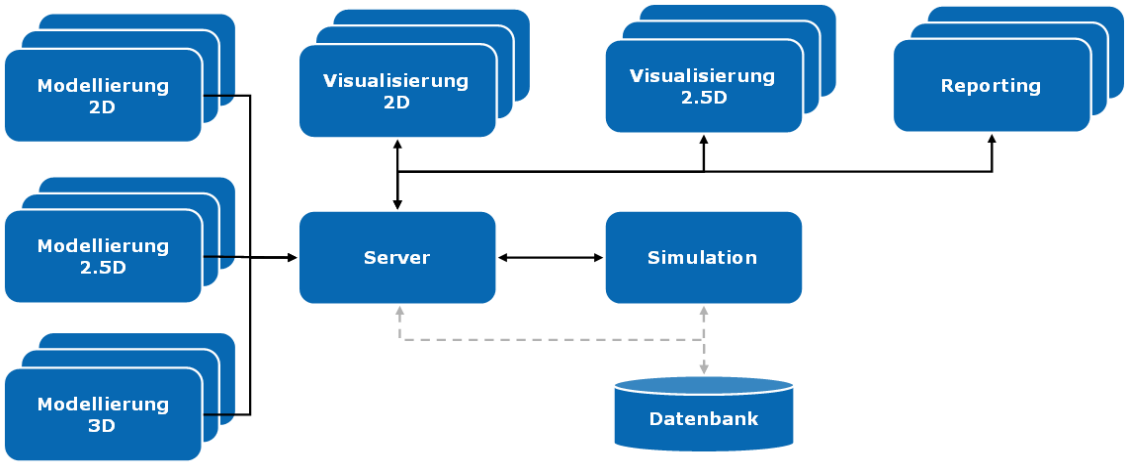


**Abbildung 88: Erweiterung des Modellierungsmoduls um Visualisierungskomponenten**

Der dreidimensionalen Visualisierung kommt während der Implementierung eine spezielle Bedeutung zu, um die realitätsnahe Immersion des Anwenders in die virtuelle Szene des Simulationsmodells zu realisieren. Neben der Anbindung an den Simulatorkern und der Animation des dynamischen Verhaltens des Simulationsmodels ist in einem ersten Schritt die flüssige Visualisierung komplexer 3D-Szenen zu realisieren. Um die Navigation des Anwenders in der virtuellen Umgebung zu unterstützen, sollen für den Simulationsverlauf

besonders relevante Prozesspunkte speziell hervorgehoben werden und die Bewegung zu diesen Punkten unterstützt werden. Neben Integration von Motion Planning zur Wegeplanung und Kollisionsvermeidung der SPMs und Avatar des Anwenders, sollen kürzeste Wege zu diesen signifikanten Prozesspunkten in die virtuelle Umgebung eingeblendet werden. Ebenso wie in anderen Visualisierungsformen muss dem Anwender ein Höchstmaß an Interaktion aus der Darstellung des Simulationslaufs ermöglicht werden. Die 3D-Visualisierung ist ebenfalls als Client umzusetzen, der das gleichzeitige Navigieren mehrerer Anwender über den „Server" Simulatorkern unterstützt. Der *MessageHandler* im Simulator muss um die Verwaltung und Bereitstellung von Anwenderinformationen, Avataren und ihren jeweiligen Positionen erweitert werden. Der Bereich des Moduls zur 3D-Visualisierung wird als Client realisiert werden, der mit dem Simulatorkern bidirektional kommuniziert, gemäß dem unter 5.2.2 entwickelten Nachrichtenprotokoll, um die Interaktion des Anwenders mit dem Simulationslauf zu ermöglichen. Aus Effizienzgründen erhält der 3D-Client eine direkte Schnittstelle zur Simulationsdatenbank, um die teilweise umfangreichen 3D-Repräsentanten einer Bausteininstanz direkt aus der Datenbank laden zu können. Die Informationen, welche 3D-Repräsentanten im Simulationslauf benötigt werden, erhält der Client über den Simulatorkern. Abbildung 89 zeigt den schematischen Aufbau der Gesamtarchitektur unter Berücksichtigung der 3D-Visualisierungskomponente.

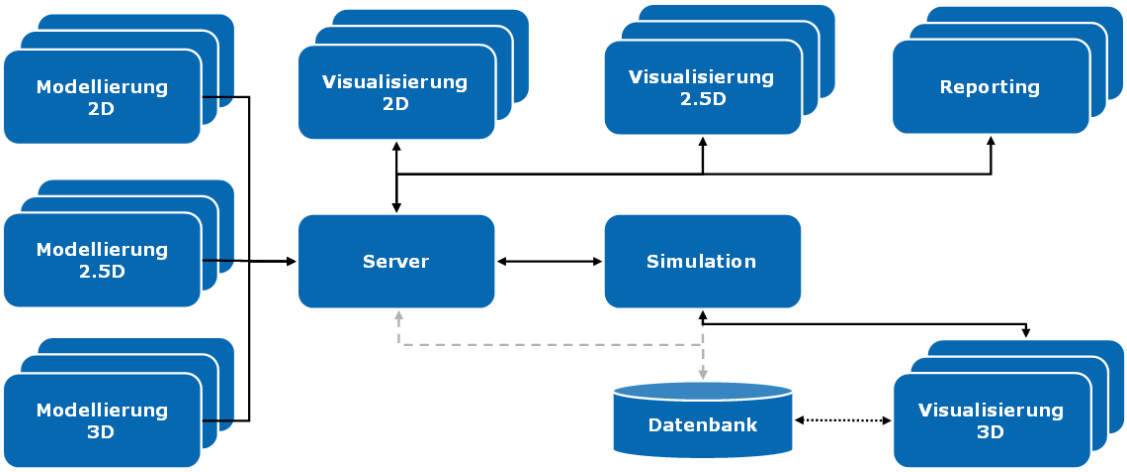


**Abbildung 89: schematischer Aufbau der Gesamtarchitektur**

Der Client der 3D-Visualisierung dient der Darstellung der gerenderten 3D-Szene für den jeweiligen Anwender. Die Navigation wird über das Motion Planning Modul so gestaltet, dass der Anwender in der virtuellen Umgebung nur Wege nehmen kann, die ihm in der realen Umwelt auch zur Verfügung ständen. Neben der Darstellung der virtuellen Umgebung werden dem Anwender in weiteren Fenstern zusätzliche Informationen über einen selektierten Modellbaustein, eine Liste aller im Simulationsmodell vorhandenen Modellbausteine, eine Minimap[35] zur Verbesserung seiner Orientierung und Methoden zur Kommunikationsunterstützung angeordnet werden. Abbildung 90 zeigt den schematischen Aufbau der 3D-Client-Oberfläche.

[35] Unter einer Minimap soll hier eine verkleinerte, maßstabsgetreue Draufsicht auf die Szene verstanden werden, die dem Anwender zusätzlich in den Client eingeblendet wird, um ihm seine Orientierung in der virtuellen Umgebung zu erleichtern.

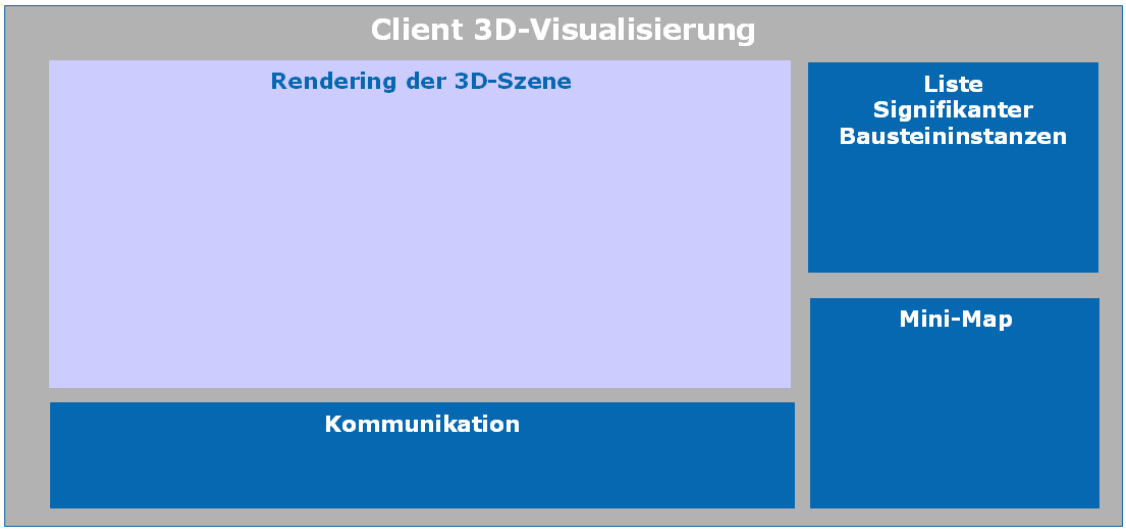


**Abbildung 90: schematischer Aufbau des 3D-Clients**

#### 5.3.2.5 Modul Simulationsdatenbank

Als gemeinsame Datenbasis aller mit der Modellierung und Simulation in Zusammenhang stehenden Daten und Informationen soll eine Simulationsdatenbank dienen. Die anfallenden Daten im Rahmen der Simulation können nach Abschnitt 5.2.3 grob in die Bereiche Eingabedaten, Modelldaten und Ergebnisdaten unterteilt werden. Zusätzlich werden Verwaltungsdaten benötigt, um die Eingangs- und Parametrierdaten sowie die Ergebnisdaten einem Simulationsexperiment und auch einem Anwender zuordnen zu können. Aus den Modelldaten ergibt sich die Struktur der Abläufe und des Materialflusses. Durch die Integration von Layout und Prozessplanung werden auch Änderungen am Layout direkt in das Simulationsmodell übertragen. Diese Aktualität ist wichtig, weil sonst Simulationsergebnisse verfälscht werden können. Die Modellbeschreibung selbst gibt nur die logische Ablaufstruktur vor. Alle Eigenschaften der Modellbausteine (Kapazität, Taktzeit, etc.) können im Bereich der Eingabedaten zu einem Simulationsmodell in der Datenbank gespeichert werden. Die Eingangsdaten der Simulationsmodelle können hinsichtlich ihrer Art weiter unterschieden werden:

- *Statische Parameter* beschreiben die Eigenschaften der Modellbausteine (Größe der Speicher, Zählpunkte, etc.), die zur Eingangsparametrierung (Initialisierung) des Modells dienen. Hier können sowohl konstante Modelldaten als auch Verweise auf Produktivdatenbanken der zu simulierenden Systeme stehen. Letztere Alternative hat den Vorteil, dass Veränderungen am modellierten System automatisch in das Simulationsmodell übertragen werden. Die meisten Eigenschaften ändern sich jedoch selten, sind also über einen längeren Zeitraum konstant. Logischerweise bieten sich diese Parameterverweise nur dann an, wenn die zu modellierenden Systeme bereits in der Realität existieren.
- *Dynamische Parameter* sind Daten eines zugrunde liegenden Produktivsystems, die aktuell für jedes Simulationsexperiment aus den Produktivdatenbanken ausgelesen werden müssen, evtl. aufbereitet für das jeweilige Simulationsmodell. Insbesondere sind dies Daten und Informationen, die den aktuellen Stand der Fertigung wiedergeben (Prozessabbild) und die Bedingungen für den Simulationshorizont darstellen (z.B. geplante Tagesproduktion, Arbeitszeitmodelle, etc.). So kann das

Verhalten des Simulationsmodells dem realen Prozess weitestgehend angepasst werden und damit als Prognoseinstrument für eine laufende Fertigung eingesetzt werden.

Alle genannten Datenbereiche sollen nachfolgend detailliert werden. Daraus ergeben sich die einzelnen Datenbereiche, wie sie in Abbildung 91 skizziert werden und innerhalb der zu entwickelnden Simulationsdatenbank umgesetzt werden müssen.

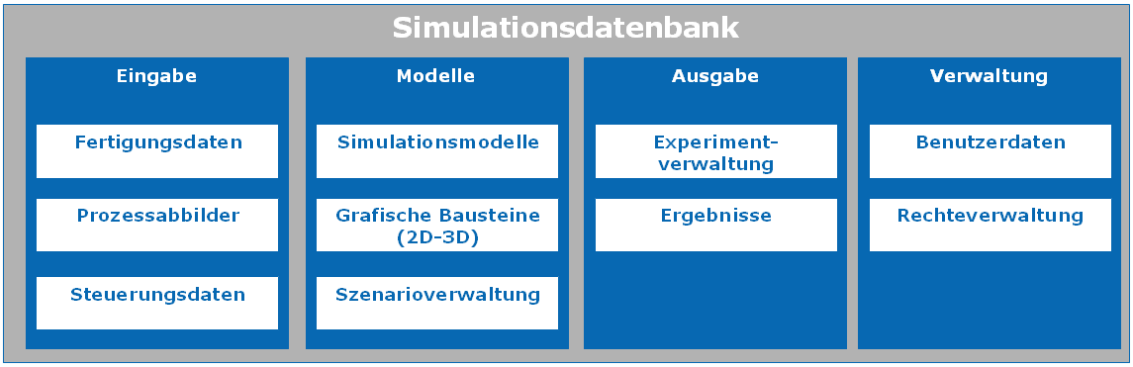


**Abbildung 91: Datenbereiche der Simulationsdatenbank**

Die einzelnen Datenbereiche übernehmen die Sicherung der ihnen zugeordneten Daten in logisch abgetrennten Bereichen der Simulationsdatenbank.

- Fertigungsdaten
  Unter Fertigungsdaten werden die Parametrierungen der einzelnen Bausteininstanzen abgelegt, mit denen die einzelnen Modelle zu Beginn eines Simulationslaufes initialisiert werden. Dabei werden durch die dynamischen Detaillierung unter Umständen einzelne Fertigungsbereiche mehrfach, aber in unterschiedlichen Detaillierungsgraden abgelegt. Abgeleitet aus der abgelegten Hierarchie der Modellbausteine im Simulationsmodell können die einzelnen Fertigungseinheiten in einer Baumstruktur (Top-Down-Hierarchie) abgelegt werden.
- Prozessabbild
  Das Prozessabbild liefert zu einem fest definierten Zeitpunkt (statisch) den Betriebszustand aller im Simulationsmodell abgebildeten Anlagen (Maschinen, Lager, Fördertechnik, etc.) und die Lokalisierung aller Produktionsgüter (innerhalb der Anlage, im Lager, auf Fördertechnik). Jeweils zum gewählten Detaillierungsgrad werden die Fertigungseinheiten und damit die Modellbausteine des Simulationsmodells mit den Produktivdatensätzen initialisiert. Da ein Prozessabbild umfangreiche Datensätze beinhalten kann und zusätzlich für eine Online-Simulation nur einmal benötigt wird, werden nur wesentliche Kennzahlen daraus in der Simulationsdatenbank langfristig abgespeichert, um die wichtigsten Eingangsparameter reproduzieren zu können (Ausnahme: Detailsimulationen, für die ein komplettes Prozessabbild benötigt wird). Die restlichen Daten werden nur temporär in der Simulationsdatenbank gespeichert und nach dem Simulationsexperiment gelöscht. Für Planungssimulationen muss eine Möglichkeit vorgehalten werden, um einzelne Prozessabbilder abzuspeichern und damit eine Simulation mehrerer Experimente auf Basis gleicher Eingangsvoraussetzungen zu starten.

- Steuerungsdaten
  Hier finden sich alle Steuerparameter und Planungsdaten der Simulation und definieren die Rahmenbedingungen für das Prognoseverhalten des Simulationsmodells. Anhand dieser Daten wird das Verhalten der Bausteine des Simulationsmodells über den Zeitraum der Simulation bestimmt. Dies sind beispielsweise Arbeitszeitmodelle und geplante Lose sowie die Histogramme, die Durchlaufzeiten oder Verwirbelungen innerhalb der Modellbausteine anhand von Wahrscheinlichkeitswerten definieren.
- Simulationsmodelle
  In diesem Datenbankbereich werden die Modellbausteine, Bibliotheken und Simulationsmodelle in den verschieden vorliegenden Versionen hinterlegt, angereichert um Meta-Informationen bzgl. des Anwenders und Kommentare zur Identifikation der Unterschiede in den Versionen. Alle Modellbausteine sind mit Standardwerten vorbelegt, soweit diese durch den Modellierer hinterlegt wurden. Die experimentspezifischen Parameter der Modellbausteine hinsichtlich eines Szenarios oder eines Simulationslaufes werden im Bereich Fertigungsdaten hinterlegt und überschreiben ggf. die Standardwerte. Zusätzlich können spezielle Parameter aus den Szenario- und/oder Experimentdaten die Standardwerte der Bausteine überschreiben. Ein Modellbaustein kann auch einen Verweis auf den Bereich der 3D-Daten beinhalten.
- Szenarioverwaltung
  Die Szenarioverwaltung umfasst Einstellparameter für Stör- und Änderungs-szenarien sowie die Zuweisungen zu entsprechenden Simulationsexperimenten (siehe Experimentverwaltung). Die manuell eingestellten Parameter und Störungen des Simulationslaufes oder –Experiments werden in diesem Bereich abgespeichert. Dies können beispielsweise Veränderungen einer Tagesproduktion, des Arbeitszeitmodells oder Störungen im Fertigungsablauf sein.
- 3D-Daten
  Im Bereich der 3D-Daten werden alle anfallenden Informationen und Datenmengen gesichert, die für die grafische Darstellung der Modellbausteine in den Modellierungs- oder Visualisierungskomponenten benötigt werden. Neben 3D-Repräsentanten und höhenabhängigen Draufsichten sind das die relativ angeordnete Anfahrpunkte für das Motion Planning, eventuell vorhandene 2,5D-Darstellungen, Outlines und Informationen über Animationspfade etc.
- Experimentverwaltung
  Hier werden die durchgeführten oder geplanten Experimente mit allen benötigten Daten und Verweisen abgespeichert. Neben dem ausführenden Anwender sind das insbesondere Informationen wie Datum, Beschreibung, Sollzahlen, hauptsächlich Verweise auf eingestellte Veränderungen und Störungen aus der Szenarioverwaltung. Abgespeicherte Experimente können als Vorlage für weiterführende Simulationsexperimente dienen.
- Ergebnisverwaltung
  Zu den durchgeführten Simulationsexperimenten werden in diesem Bereich der Simulationsdatenbank die Ergebnisse der einzelnen Simulationsläufe gespeichert.. Die Ergebnisse spiegeln sich u.a. in einer Reihe von Ergebniskennzahlen wieder, die als Statistiken aufbereitet werden und in Auswertungsmodulen in verschiedenen Darstellungsformen angezeigt werden können. Für das Protokollieren und Sichern

von Experimentdaten eines Simulationslaufs wurde in Abschnitt 5.2.3 eine Beschreibung erstellt, die in diesem Bereich gesichert werden soll.

- Benutzerverwaltung
  Die Benutzerverwaltung enthält alle Benutzerdaten und die Gruppenzugehörigkeit mit den entsprechenden Rechten. Anhand der Benutzergruppe, personifizierten Rechten und den Einstellungen in den einzelnen Modellbausteinen ergeben sich die Rechte für den Zugriff auf die Simulationsmodelle und deren einzelne Modellbausteine. Hier wird zusätzlich das entwickelte Rechtemanagement abgelegt und verwaltet (vgl Abschnitt 5.1.6.).

Für die manuelle Bearbeitung der Daten in der Simulationsdatenbank soll ein einfaches Fron-End entwickelt und benutzt werden, um außerhalb der Modellierungs-, Simulations und Visualisierungskomponenten die Datenbank warten zu können. Wie oben bereits angedeutet, können diese Wartungsaufgaben zumeist auch von Nicht-Simulationsexperten durchgeführt werden.

#### 5.3.2.6 Modul Administration

Das Funktionsmodul Administration dient der wartungsfreundlichen Manipulation vorhandener Datensätze in und aus der Simulationsdatenbank. Wesentliche Anforderungen ergeben sich aus der Pflege der Simulationsmodelle, Experiment- und Szenariodaten sowie der komfortablen Bearbeitung und Pflege aller benötigten 2D, 2.5D oder 3D-Repräsentanten der logischen Modellbausteine. Darüber hinaus wird eine grafische Oberfläche zur Pflege der Benutzerdaten und der vorhandenen Rechteverwaltung benötigt. Abbildung 92 zeigt eine nach Funktionsgruppen sortierte Übersicht aller anfallenden administrativen Aufgaben.

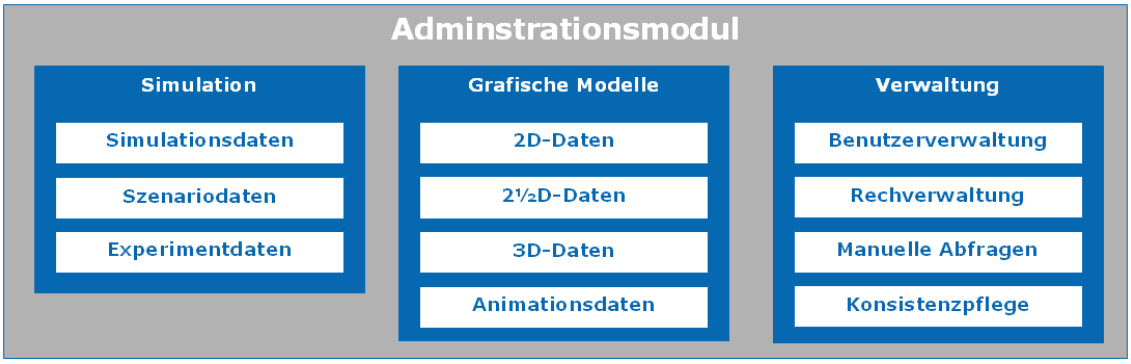


**Abbildung 92: Teilmodule des Administrationsmoduls**

Die einzelnen Anforderungen lassen sich auch durch spezifische, ggf. bereits vorhandene Werkzeuge implementieren und abarbeiten. Im Rahmen der Konzeption eines Gesamtwerkzeugs soll hier aber ein einheitliches Administrationsmodul umgesetzt werden, dass alle erforderlichen administrativen Aufgaben abzuarbeiten ermöglicht. Dazu müssen in den einzelnen Modulteilen entsprechende Funktionen implementiert werden, die eine Umsetzung der dargestellten Use-Cases erlauben.

Auf Basis der in diesem Kapitel getroffenen Design-Entscheidungen hinsichtlich der Modellbeschreibung und der Modularisierung des zu entwickelnden Werkzeuges sollen im folgenden Kapitel die einzelnen Bausteine des Werkzeugs in der Programmiersprache

Java umgesetzt werden. Anschließend wird anhand eines Beispielmodells nachgewiesen, dass das Werkzeug den unter Kapitel 1 gestellten Anforderungen genügt.

# 6 Realisierung

*„Leben heißt Handeln."*

*(Albert Camus)*

Nach der Konzeptionsphase müssen die entwickelten Funktionsmodule der Gesamtanwendung implementiert und in einer Gesamtanwendung zusammengeführt werden. Dieses Kapitel will das Ergebnis der Implementierungsphase beschreiben, indem es anhand eines spezifischen Beispiels den Nachweis führt, dass alle in Abschnitt 2.4 gestellten Anforderungen an Basisprozess, Modellbeschreibung und Werkzeug erfüllt werden können.

Das Kapitel ist nach der Vorgehensweise bei der Durchführung einer Simulationsstudie strukturiert. In den ersten Abschnitten wird daher zunächst Untersuchungsgegenstand und Untersuchungsziel definiert. Die Problemstellung ist so gewählt, dass alle in der Anforderungsbeschreibung aufgenommenen Funktionalitäten ihre Anwendung finden. Sie werden an geeigneter Stelle abgearbeitet, um sowohl Mächtigkeit der Modellbeschreibung als auch Funktion des Werkzeugs zu illustrieren. Auf den Nachweis der Simulationswürdigkeit, der üblicherweise an erster Stelle einer Simulationsstudie steht, wird an dieser Stelle verzichtet.

## 6.1 Definition des Untersuchungsgegenstands

Zu Beginn der Simulationsstudie soll der Untersuchungsgegenstand beschrieben werden, wie er im Folgenden abgebildet wird. Fokus der Untersuchung ist die fiktive Fabrik der PaderKarts GmbH (PaderK), die auf Basis von Rohstoffen und Teilerzeugnissen kundenindividuelle Karts herstellt. Der Leistungserstellungsprozess kann grob in zwei Prozessschritte unterteilt werden: Die Teilefertigung von Zwischenerzeugnissen auf Basis von Rohmaterialien und die kundenindividuelle Montage auf Basis der Zwischenerzeugnisse. Beide Prozesse werden durch ein zentrales Lager entzerrt, bzw. aus diesem Zentrallager mit Faktoren versorgt. Abbildung 93 zeigt das Grobschema der Fertigung der PaderKarts GmbH.

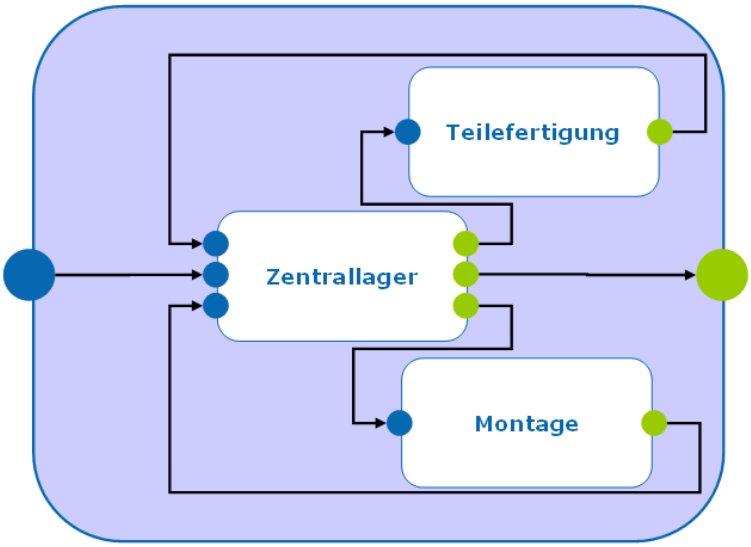

**Abbildung 93: Grobschema der Fertigung von Karts bei der PaderK GmbH**

Alle Teilmodelle der PaderKarts GmbH sollen auf zwei unterschiedlichen Detaillierungsebenen betrachtet werden: Zum Einen eine weniger detaillierte Ebene, in der die drei Teilbereiche der PaderK zeitorientiert und ohne Transportwege modelliert und simuliert werden können; Zum Anderen eine höher detaillierte Betrachtungsebene, die zusätzlich die Transportbeziehungen innerhalb der Teilbereiche abbildet. Nachfolgend sollen die einzelnen Teilbereiche in den jeweiligen Detaillierungen näher beschrieben werden.

### 6.1.1 Zentrallager

Das Lager der PaderK ist zentraler Anlaufpunkt aller Warenflüsse. Wie in Abbildung 93 gezeigt, werden die fremdbeschafften Rohstoffe, alle eingekauften Halbfabrikate, die Teilerzeugnisse der Teilefertigung und kundenindividuellen Erzeugnisse der Montage in einem gemeinsamen Lager verwaltet. Die jeweils ankommenden Waren werden an einem gemeinsamen Wareneingang zwischengelagert und durch Gabelstapler auf die einzelnen Lagerplätze in vier Lagergassen verteilt. Eine weitere Gruppe von Gabelstaplern übernimmt den unternehmensinternen wie -externen Versand der Erzeugnisse, indem sie die Waren aus den Lagerplätzen entnehmen, an einem zentralen Warenausgang kommissionieren und verschicken. Abbildung 94 zeigt die schematische Struktur des Zentrallagers ohne Berücksichtigung der Staplersteuerung und Gabelstapler, also auf der weniger detaillierten Betrachtungsebene. Die Auftragssteuerung generiert die Aufträge für den Warenausgang. Sie werden durch den Wareneingang durchgeschleust und veranlassen die Auslagerung in der jeweiligen Lagergasse, so dass die Erzeugnisse im Warenausgang ankommen. Hier werden sie anschließend versendet.

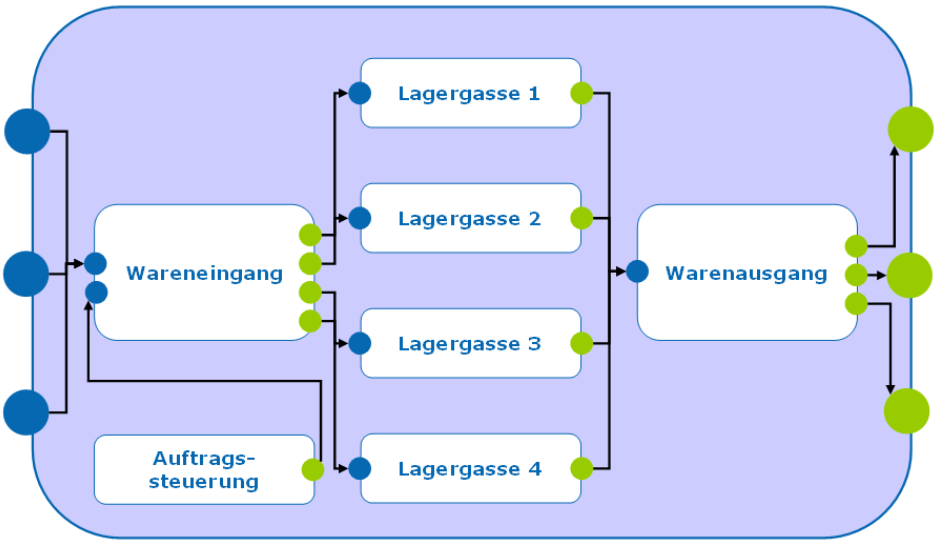


**Abbildung 94: Zentrallager der PaderK in niedriger Detaillierung**

Die detaillierte Betrachtung unter Einbeziehung der beiden Gabelstaplergruppen erweitert das Lager nur leicht. Wie Abbildung 95 zu entnehmen ist, schleust der Wareneingang die ankommenden Einlagerungsaufträge und Waren an die Staplersteuerung weiter, die den Transport mittels der Gabelstapler bis zu den eigentlichen Lagerplätzen in den Lagergassen unter Einbeziehung der kürzesten Wege übernimmt. Die Auftragssteuerung zur Auslagerung ist an die Steuerung der Gabelstapler-Gruppe „Warenausgang" angebunden. Übergebene Auslagerungsaufträge werden mittels der Gabelstapler an den

Warenausgang übergeben, wobei die Transportwege ebenfalls unter Anwendung des Motion Planning berechnet werden sollen. Der Warenausgang übernimmt die Kommissionierung der einzelnen Auslageraufträge zu Versandaufträgen und transportiert die entsprechenden Erzeugnisse anschließend, je nach Bestimmungsort, an die entsprechenden Output-Channel des übergeordneten Modellbausteins *Zentrallager*.

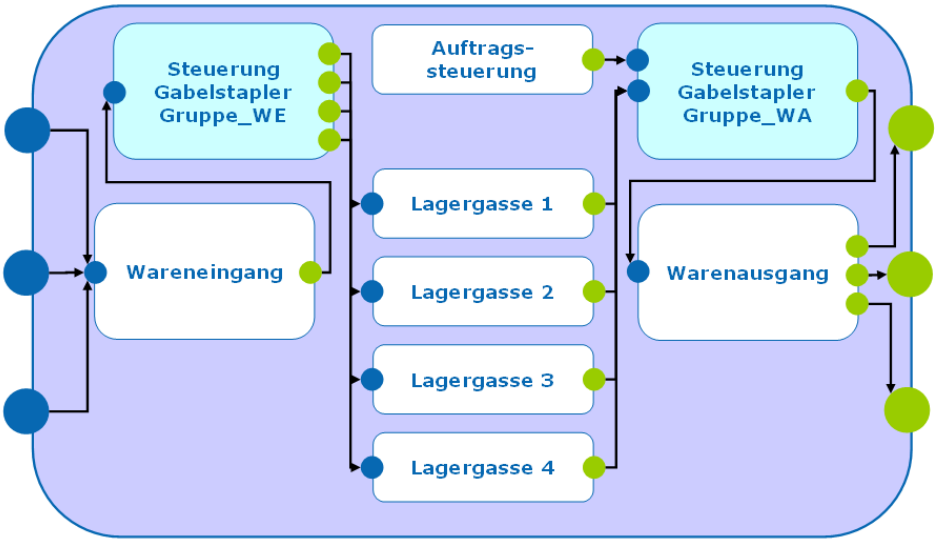


**Abbildung 95:Zentrallager der PaderK in hoher Detaillierung**

Die einzelnen Output-Channel des Modellbausteins *Zentrallager* sind mit den anderen Bereichen des Gesamtmodells verbunden, also dem Eingang des Modellbausteins Teilefertigung, dem Eingang des Modellbausteins Montage oder dem Output-Channel des Gesamtmodells der PaderK. Da die entsprechenden Verknüpfungen jeweils nur logischen Vorgänger/Nachfolger-Beziehungen entsprechen, muss bereits im Modellbaustein *Warenausgang* des Zentrallagers sowohl auf niedriger wie hoher Betrachtungsebene entschieden werden, über welchen Output-Channel des Bausteins der jeweilige Auftrag versandt wird.

### 6.1.2 Teilefertigung

Die von der PaderK eingekauften Rohstoffe und Halberzeugnisse werden in der Teilefertigung verarbeitet. Zwei grundlegende Bearbeitungen können unterschieden werden, die durch eine Gruppenfertigung abgebildet werden: Zum Einen die Bearbeitung von Metallteilen durch Umform-, Biege- und Fügetechniken. Die erstellten und behandelten Metallteile werden in einem Folgeschritt zu Karosseriestrukturen des Karts verschweißt. Zum Anderen die Herstellung von Kunststoffteilen mittels eines Spritzgussverfahrens, das eine umfangreiche Nacharbeit (Entgraten, Schleifen, etc) erfordert. Alle Zwischenerzeugnisse werden in einem Teilelager innerhalb der Teilefertigung zwischengepuffert, anschließend entweder an die Lackierung übergeben oder gehen direkt ins Zentrallager der PaderK. Auf der gröberen Betrachtungsebene, wie sie in Abbildung 96 dargestellt wird, werden die Transporte zwischen den einzelnen Stufen nicht betrachtet.

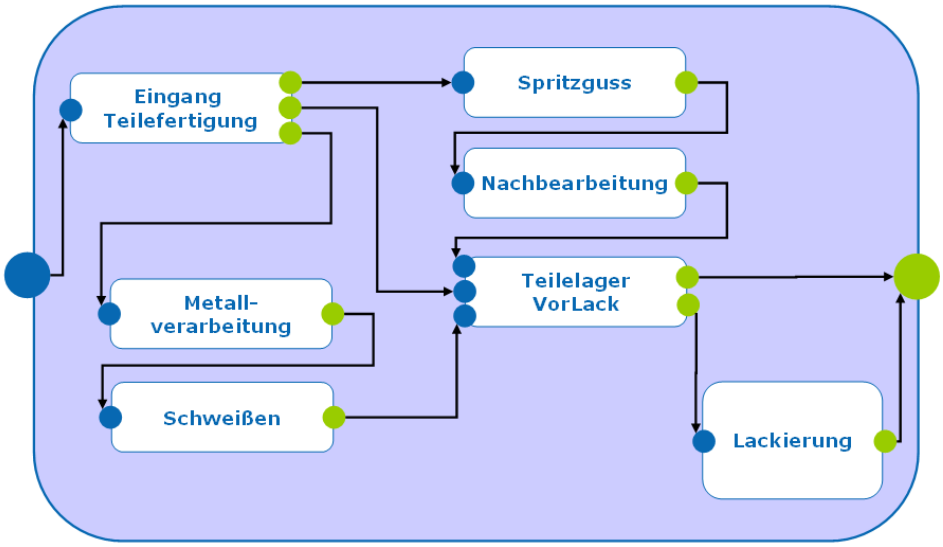


**Abbildung 96: Teilefertigung der PaderK in niedriger Detaillierung**

Der Transport innerhalb der Teilefertigung wird bei der PaderK mittels fahrerloser Transportfahrzeuge (FTF) gewährleistet, die auf festen Wegen die Teilefertigung durchqueren und die Zwischenerzeugnisse zwischen den einzelnen Fertigungsstufen transportieren. Abgebildet wird das fahrerlose Transportsystem (FTS) über eine feste Wegestruktur in Form von Bausteinen, die Wege oder Bahnhöfe repräsentieren, wobei nur an den Bahnhof-Bausteinen Teile aufgenommen und abgeladen werden können. Die Steuerung des FTS erfolgt über einen speziellen Steuerungsbaustein, der alle FTF, ihre Routenpläne und aktuelle Routen verwaltet. Der Weiterversand der bearbeiteten Erzeugnisse ins Zentrallager erfolgt direkt aus der Lackierung oder dem Teilelager VorLack. Die Abbildung erfolgt im Modell der detaillierten Betrachtungsebene durch ein Durchschleifen der Erzeugnisse durch den entsprechenden Bahnhof des Teilelagers.

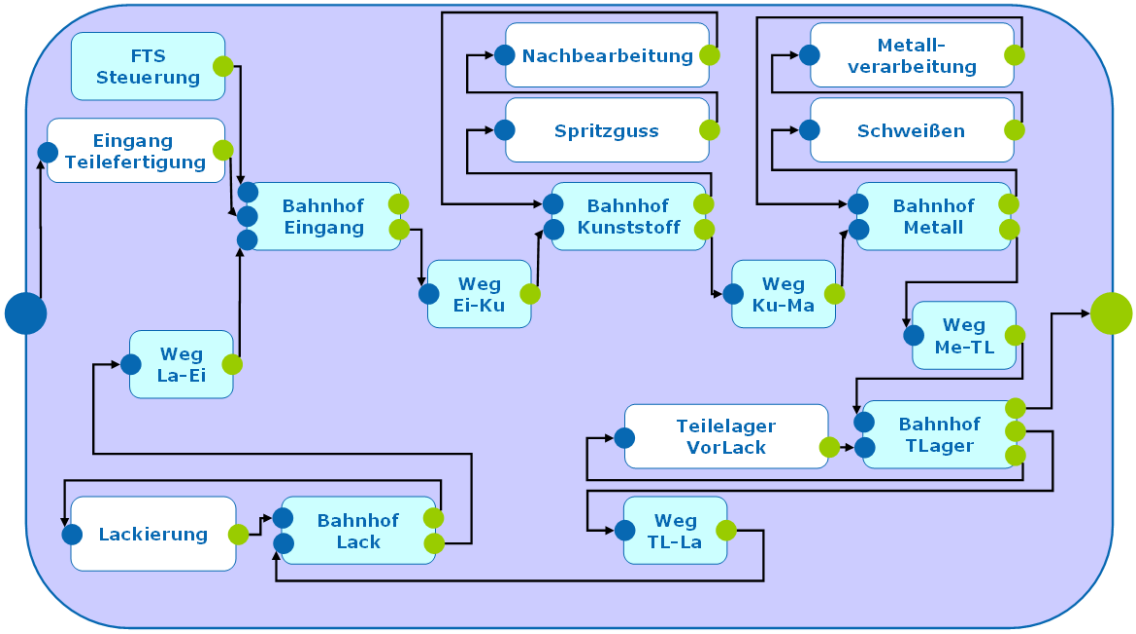


**Abbildung 97: Teilefertigung der PaderK in hoher Detaillierung**

Weitere benötigte Einzelteile zur Herstellung eines kundenindividuellen Karts werden durch die PaderK fremdbeschafft, beispielsweise die verwendeten Motoren und die

Rad/Reifen-Kombinationen. Die Fertigung der Teilefertigung erfolgt auf Basis einer Abschätzung vorhandener und zukünftiger Aufträge und ist weitgehend kundenanonym.

### 6.1.3 Montage

Die einzelnen Karts werden kundenauftragsbezogen in der Montage zusammengebaut. Chassis und Motoren werden am Anfang der Montage sequenzgenau bereitgestellt und als erste Montagestufe verheiratet. Anschließend erfolgt die mehrstufige Montage der Anbauteile und Sonderausstattungen. Die fertig zusammengebauten Karts werden auf zwei parallel betriebenen Linien auf Verarbeitungsqualität und Funktion überprüft und anschließend im Zentrallager bis zur Auslieferung zwischengelagert. Werden im Rahmen der Qualitätsuntersuchungen Mängel an einem Kart festgestellt, so wird das entsprechende Kart aus der Fließfertigung ausgeschleust und der Nacharbeit zugeführt. Nachdem der identifizierte Fehler behoben ist, erfolgt eine erneute Qualitätsprüfung und die Weiterleitung des Karts an das Zentrallager. Auch bei der Abbildung der Montage werden die Transportwege zwischen den einzelnen Montagevorgängen im niedrig detaillierten Modell noch nicht betrachtet.

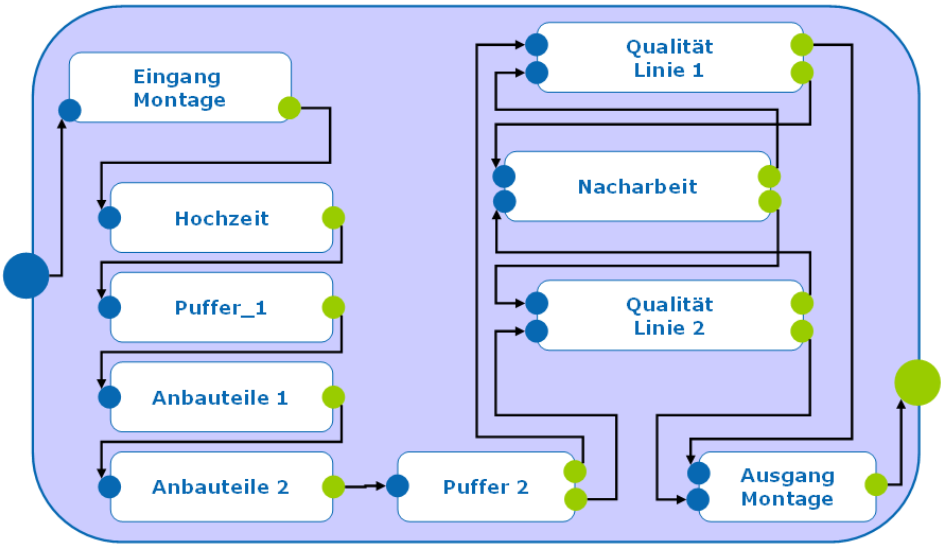


**Abbildung 98: Montage der PaderK in niedriger Detaillierung**

Aus dem höher detaillierten Modell der Montage der PaderK (vgl. Abbildung 99) wird ersichtlich, dass die einzelnen Montagestufen mittels fester Fördertechnik untereinander verbunden sind. Die feste Verkettung der Fließmontage wird nur dadurch teilweise entzerrt, dass einzelne Förderstrecken als Puffer ausgelegt sind. So kann die Förderstrecke 3 einerseits als Ausfallstrecke dienen, wenn Förderstrecke 2 ausfällt oder gewartet werden muss, andererseits können hier Karts zwischengespeichert werden, wenn Qualität oder Nacharbeit die anfallende Arbeit nicht bewältigen können. Zusätzlich ist vor dem Eingang in den Qualitätsbereichen eine Staustrecke installiert worden, auf der weitere Karts zwischengespeichert werden können.

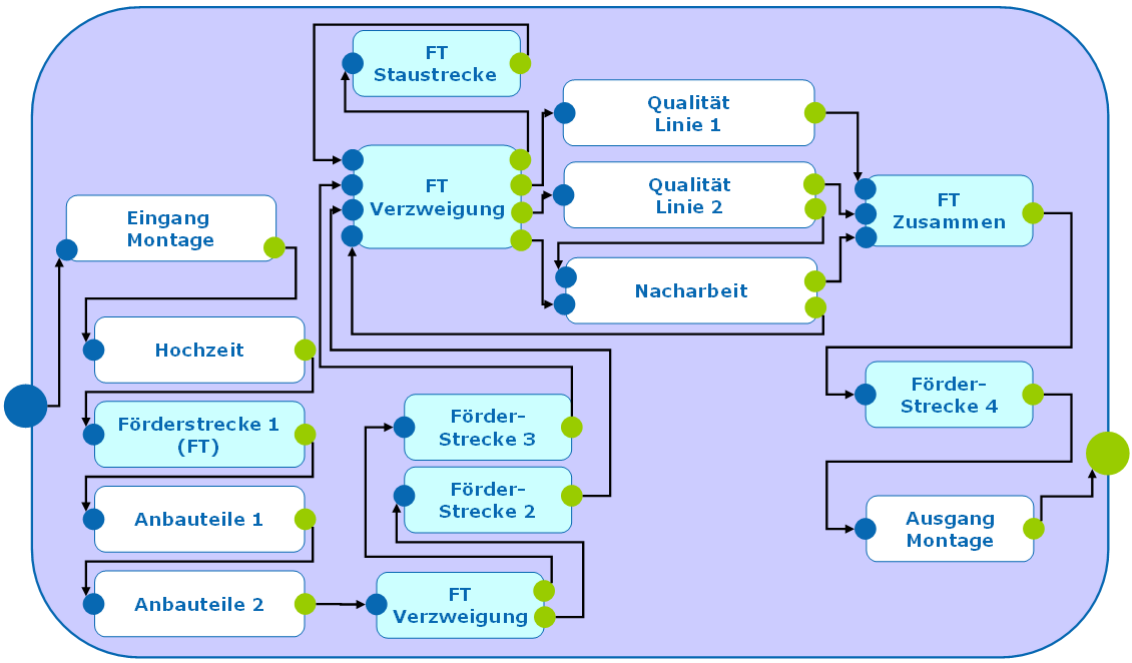


**Abbildung 99: Montage der PaderK in hoher Detaillierung**

Die in den Abbildungen darstellten Bausteine der einzelnen Bereiche der PaderK können ähnlich wie auch die Bereiche Zentrallager, Teilefertigung und Montage an den geeigneten Stellen weiter detailliert werden. So bestehen beispielsweise die Förderstrecken jeweils aus einzelnen Elmenten einer Fördertechnik, oder die Lagergassen des Zentrallagers aus einer Fördertechnik mit den entsprechenden Lagerplätzen. Die dargestellten Detaillierungen sollen aber für die Untersuchungsziele dieser Simulationsstudie genügen.

*Aussagefähigkeit der PaderK hinsichtlich der Aufgabenstellung*
Die Struktur der fiktiven PaderKarts GmbH kann exemplarisch für Fertigungssysteme angesehen werden, die im Rahmen verschiedener Untersuchungen mittels einer Ablaufsimulation betrachtet werden. Die einzelnen Bereiche entsprechenden unterschiedlichen Organisationsformen, die mit den bekanntesten Varianten von Transportsteuerungen verbunden sind. Durch die Modellierung und Simulation der PaderK mit dem hier entwickelten Werkzeug kann somit der Nachweis geführt werden, dass verschiedenartige Fertigungssysteme mit dem Werkzeug abgebildet und analysiert werden können. Im Einzelnen erfüllen die Teilbereiche der PaderK folgenden Zweck hinsichtlich der Aufgabenstellung:

- Zentrallager: Die wesentliche Aufgabe des Zentrallagers vor dem Hintergrund der Anforderungen ist die Implementierung einer Transportsteuerung unter der Verwendung des implementierten Motion Planning Verfahrens auf der höher detaillierten Betrachtungsebene des Modells. Dadurch wird die Funktionsweise des entwickelten Verfahrens zur verbesserten **Abbildung funktionsorientiert gegliederter Fertigungssysteme** nachgewiesen. Im Rahmen der Simulationsstudie hat das Zentrallager die Aufgabe die unterschiedlich arbeitenden Teilbereiche Teilefertigung und Montage zu entzerren, um den Aufbau des Simulationsmodells nicht unnötig zu verkomplizieren. Es erlaubt die Abkopplung der beiden Teilbereiche untereinander, so dass die jeweiligen Simulationsmodelle in den

entsprechenden Szenarien getrennt betrachtet werden können. Als einzige Restriktion, die sich bei diesem Vorgehen ergibt, muss festgehalten werden, dass sich ein Rückstau aus dem Lager nicht auf die einzelnen Teilbereiche auswirkt. Für den Nachweis der Anforderungen, zu dem diese Simulationsstudie dienen soll, kann dies aber in Kauf genommen werden.

- Teilefertigung: Aufgabe der Teilefertigung ist die Abbildung einer funktionsorientiert gegliederten Fertigung. Weil die Suche nach kürzesten Wegen schon durch das Zentrallager abgebildet wird, bietet sich die Teilefertigung als höher detailliertes Simulationsmodell zur Modellierung und Simulation von fixen Transportwegen unter Verwendung von Modellbausteinen an. Durch die Gestaltung der Gruppenfertigung und ihrer Transportwege mittels eines fahrerlosen Transportsystems ist dem bei der Gestaltung des Untersuchungsgegenstandes Rechnung getragen worden.
- Montage: Die Montage dient als Anwendungsbeispiel einer klassischen Fließ- bzw. Serienfertigung von kundenindividuellen Erzeugnissen, wie sie beispielsweise in der Automobilindustrie sehr verbreitet ist. Heutzutage wird die Ablaufsimulation meist vor dem Hintergrund solcher Fragestellungen von **objektorientierten Organisationsformen** eingesetzt. Die Transportwege sind, in der Realität wie auch im gewählten Beispiel, durch eine starke Verkettung (Fördertechnik, Band- oder Kettenförderer) gekennzeichnet, ergänzen und komplettieren dadurch die Abbildung von Transportstrukturen in dem gewählten Beispiel.

Hinsichtlich der Modellbeschreibung kann die Modellierung der PaderK in der oben dargestellten Form, bzw. deren analoge Umsetzung in dem implementieren Werkzeug, zeigen, dass eine **modulare**, **hierarchische** und **objektorientierte** Abbildung durch das Werkzeug ermöglicht wird. Die einzelnen Bausteine werden jeweils mittels der für die Modellbeschreibung geltenden Regeln erzeugt und führen damit auch den Nachweis, dass die Modellierung **beliebiger Detaillierungsgrade** mit dem Werkzeug erfolgen kann.

Der Untersuchungsgegenstand der PaderK soll damit für die Aufgabenstellung hinreichend genau beschrieben sein. Die einzelnen Ziele und Untersuchungsszenarien sollen im folgenden Abschnitt dargestellt und hinsichtlich der Aufgabenstellung begründet werden.

## 6.2 Definition des Untersuchungsziels

Es wird angenommen, dass die PaderKarts GmbH setzt die Methode Ablaufsimulation bisher nicht einsetzt. Im Rahmen der Simulationsstudie sollen daher verschiedene Szenarien durchgeführt werden, die mögliche Einsatzfelder bei der PaderK aufzeigen und evaluieren. Die Szenarien zeichnen sich auch dadurch aus, dass mit ihnen ein planugsphasenübergreifender Einsatz der Ablaufsimulation ermöglicht wird.

**Szenario 1: Zeitorientierte Rückwärtssimulation zur Terminierung von Fertigungsplänen**

Auf Basis der erstellen Fertigungs- und Montageabläufe der Paderk soll durch eine zeitorientierte Rückwärtssimulation des Gesamtmodells eine Grobplanung der Kundenaufträge für Teilefertigung und Endmontage durchgeführt werden. Terminiert durch verfügbare und geplante Kundenaufträge soll als Simulationsergebnis ein initialer Fertigungsplan erstellt werden, der eine Zeit- und Personalplanung für die einzelnen

Teilbereiche ermöglicht, gleichzeitig eine rechtzeitige Bereitstellung der in späteren Fertigungsstufen benötigten Teilerzeugnisse und Zwischenprodukte garantiert. Die einzelnen intralogistischen Transporte sollen in dieser Untersuchung nicht berücksichtigt werden, weshalb an dieser Stelle die Abbildung der einzelnen Teilbereiche auf der niedrigeren Betrachtungsebene ausreicht.

*Aussagefähigkeit des Szenarios hinsichtlich der Aufgabenstellung*
Die Umsetzung des ersten Szenarios durch die Modellierung und Simulation mit dem implementierten Werkzeug dient in erster Linie dem Nachweis einer **zeitorientierten** Modellierung und Simulation. Die gewählte Fragestellung unterstützt den Ansatz eines phasenübergreifenden Einsatzes der Ablaufsimulation durch die Anwendung im Rahmen einer Fertigungslenkungsaufgabe. Darüber hinaus erbringt die rückwärts gerichtete Ausführung der Simulation den Nachweis der benötigten Kernelfunktionalitäten zur Rückwärtssimulation. Die erstellten Modelle dienen als Ausgangsbasis für nachfolgende Szenarien, indem sie durch die implementierte Transformation in vorwärts gerichtete Simulationsmodelle umgewandelt werden können. Die Modellbeschreibung kann unter Verwendung dieses Verfahrens als **richtungsoffen** bezeichnet werden und erfüllt so eine Anforderung der Arbeit. Die daraus resultierenden Simulationsmodelle der weniger detaillierten Betrachtungsebene werden für das zweite Szenario entsprechend erweitert und erfüllen somit die Anforderung nach einem **modularen, erweiterbaren** Aufbau der Simulationsmodelle auf Basis der entwickelten Modellbeschreibung.

**Szenario 2: Ereignisdiskrete Vorwärtssimulation zur Engpass- und Sensitivitätsanalyse der Fertigungsabläufe**
Alle Teilbereiche der PaderK sollen durch eine ereignisdiskrete Vorwärtssimulation einer Engpassanalyse als klassisches Einsatzfeld der Ablaufsimulation unterzogen werden. Aussagen hinsichtlich der maximalen Leistungsfähigkeit der installierten Anlagen unter den gegebenen Randbedingungen (Arbeitszeitmodelle, Materialverfügbarkeiten, etc.) dienen als Ausgangsbasis für eine mögliche Sensitivitätsanalyse, die Abhängigkeiten der jeweiligen Teilsysteme von einzelnen Parametern identifizieren kann. Durch interaktive Analysen des Simulationsmodells soll der Erkenntnisgewinn über die Teilbereiche der PaderK vorangetrieben werden. Um repräsentative Aussagen hinsichtlich des Systemverhaltens der einzelnen Fertigungs- und Montageprozesse zu erlauben, muss diese Simulation im Wesentlichen auf den höher detaillierten Betrachtungsebenen der einzelnen Teilbereiche basieren. Damit kann dann auch gezeigt werden, dass die im ersten Szenario erstellten Fertigungspläne auch unter Berücksichtigung der verschiedenen Transporte eingehalten werden können. Die in diesem Schritt bereits existierenden zeitorientierten Modelle müssen deshalb in ereignisorientierte Modelle umgewandelt und um eine weitere Detaillierungsebene ergänzt werden.

*Aussagefähigkeit des Szenarios hinsichtlich der Aufgabenstellung*
Das zweite Szenario dient in erster Linie dem Nachweis einer ereignisgesteuerten, diskreten Materialflusssimulation mit dem entwickelten Werkzeug. Speziell die Analysen in der dreidimensionalen Darstellung sollen darüber hinaus zeigen, dass eine **interaktive** und bestmöglich **immersive** Betrachtung der dynamischen Verhaltensweisen der Simulationsmodelle mit dem Werkzeug möglich ist. Ein Experimentieren durch mehrere Anwender soll darüber hinaus vorstellen, dass der geforderte **Mehrbenutzer-, bzw. Multitaskingbetrieb** in der Visualisierung ermöglicht werden kann. Der analoge

Nachweis einer Modellierung durch mehrere Anwender soll während der Implementierung des Simulationsmodells in der Modellierungskomponente erfolgen. Durch die Gestaltung des Simulationsmodells mit verschiedenen Detaillierungsgraden kann an dieser Stelle auch gezeigt werden, dass das Verfahren von Mueck (dynamische Detaillierung zur Laufzeit) in dieser Werkzeugimplementierung integriert wurde.

**Szenario 3: Ereignisdirekte Rückwärtssimulation zur Feinplanung der Teilefertigung**

Unter der an dieser Stelle vorweggenommenen Annahme des Gesamtprozess-Engpasses innerhalb der Montage der PaderK, bietet eine zusätzliche Untersuchung in diesem Bereich nur wenig Potential. Als drittes Szenario soll deshalb eine Feinplanung der Aufträge in der weniger ausgelasteten Teilefertigung durch eine ereignisdiskrete Rückwärtssimulation erfolgen. Ziel ist die verbesserte Termin-, Mengen- und Schicht- bzw. Personalplanung in der Teilefertigung hinsichtlich spätester Beginnzeitpunkte, um die Lagerbestände innerhalb des Zentrallagers auf ein Minimum beschränken und dadurch gebundenes Kapitel freisetzen zu können.

*Aussagefähigkeit des Szenarios hinsichtlich der Aufgabenstellung*

Weil die kundenindividuelle Montage der Engpass der PaderK ist, reichen die im zweiten Szenario generierten Ergebnisse nach frühesten möglichen Einplanungen für eine Analyse des Systemverhaltens in der Montage aus. Eine Betrachtung der Teilefertigung unter Verwendung der **ereignisdiskreten Rückwärtssimulation** soll erneut das Potential eines planungsphasenübergreifenden Einsatzes der Ablaufsimulation aufzeigen. Die Planung von Fertigungsplänen kann mittels der Rückwärtssimulation direkt aus dem bestehenden Simulationsmodell erfolgen, wenn Kernel und Modellbeschreibung die anwenderfreundliche Richtungs**transformation** unterstützen.

In Summe ergibt sich aus der Gestaltung des Untersuchungsgegenstandes und der abgeleiteten Untersuchungsziele eine breite Übersicht über die in dem Werkzeug zur Verfügung gestellten Funktionalitäten. Alle aus Kapitel 2.4 aufgenommenen Anforderungen hinsichtlich Modellbeschreibung und Werkzeug werden durch die oben definierten Szenarien überprüft. Wird die Simulationsstudie im Folgenden erfolgreich durchgeführt, kann daraus geschlussfolgert werden, dass alle an diese Arbeit gestellten Anforderungen auch erfüllt werden können.

## 6.3 Datenermittlung und Aufbau eines logischen Modells

Der Bereich der Datenanalyse, -vorbereitung, -sammlung und -bearbeitung ist eine der entscheidenden Teilaufgaben im Rahmen einer Simulationsstudie. Typischerweise ergeben sich hier zeitaufwändige Fragestellungen hinsichtlich Datengenauigkeit, Granularität, Verfügbarkeit und Aktualität, die im Rahmen der Entwicklung eines ersten logischen Modells des abzubildenden Systems erörtert und geklärt werden müssen. Für die hier behandelte Simulationsstudie ist dieser aufwendige Prozess insofern unkritisch, als das durch den Untersuchungsgegenstand einer *fiktiven* Fabrik alle benötigten Daten in der gewünschten Form generiert und bereitgestellt werden können. Die eigentliche Aufgabe in diesem Abschnitt beschränkt sich also auf die Erstellung eines logischen Modells.

Zur Erstellung dieses logischen Ablaufs aller relevanten Prozesse innerhalb der PaderK kann auf die Definition des Untersuchungsgegenstands aus Abschnitt 6.1 zurückgegriffen werden. Die entsprechende Liste der Abbildung 93 bis Abbildung 99 vermitteln einen hinreichend genauen Eindruck über die benötigten Bausteine des Simulationsmodells, ihre Hierarchisierung und gegenseitigen Abhängigkeiten. Wie im obigen Abschnitt angerissen, können einzelne Modellbausteine für die Umsetzung innerhalb des Werkzeugs weiter detailliert werden, beispielsweise die Abbildung einzelner Lagerplätze und eines Regalfahrzeugs innerhalb der Lagergassen 1 bis 4 im Zentrallager. Je nach Szenario müssen in den einzelnen Teilmodellen weitere Quellen, Senken und Auswertungsbausteine hinzugefügt werden, die eine zeit- oder ereignisdiskrete Ablaufsimulation in Vorwärts- oder Rückwärtsrichtung abhängig vom jeweiligen Untersuchungszweck ermöglichen. Die Darstellung der einzelnen Prozessabläufe soll aber mit den oben angegebenen Abbildungen als hinreichend betrachtet werden und erst im Rahmen der eigentlichen Modellierung in dem entwickelten Werkzeug bei Bedarf erweitert werden.

## 6.4 Aufbau eines Simulationsmodells

Im folgenden Abschnitt soll die Modellierung der Simulationsmodelle mittels der Modellierungskomponente des Werkzeugs vorgestellt werden. Für jedes der unter Abschnitt 6.2 vorgestellten Szenarien sollen einige Bausteine der Teilmodelle exemplarisch herausgegriffen und ihre Bearbeitung mit dem Werkzeug detaillierter erläutert werden. Die Vorgehensweise orientiert sich an der Reihenfolge der durchzuführenden Szenarien. Vorab sollen einige Bemerkungen zur Implementierung der Modellbeschreibung und Modellierungskomponente gemacht werden.

**Modellbeschreibung**

Im Rahmen der Implementierung wurden in einem ersten Schritt die Modellbeschreibung und das nachrichtenbasierte Kommunikationsprotokoll in Document Type Definitions (DTD) der XML umgesetzt, die eine formale Struktur der erstellten XML-Dokumente vorgeben. Deren genaue Auflistung finden sich in Anhang A für die Beschreibung der Simulationsmodelle und in Anhang B für die Nachrichtenkommunikation zwischen den Teilmodulen. Die administrativen Arbeiten, die zur Bearbeitung der Szenarien erforderlich sind, sollen erst am Ende dieses Unterabschnittes näher betrachtet werden, auch wenn die Verwendung der Modellierungskomponente das Existieren von Benutzerrechten sowie die Existenz einiger dreidimensionaler Repräsentanten in einer Simulationsdatenbank voraussetzt.

**Modul Modellierung**

Die Modellierungskomponente wurde entsprechend der Entwurfsphase als Client-Server-Architektur umgesetzt. Der Server der Modellierungskomponente übernimmt hierbei die Kommunikation mit den anderen Funktionsmodulen wie Simulatorkern und Simulationsdatenbank und beinhaltet als Teilmodul die Transformation eines Simulationsmodells. Die Simulationsmodelle werden als XML-Datei eingelesen und in die Java-Objektklassenhierarchie geparst, so dass sie durch die Clients manipuliert werden können. Der Client der Modellierung ist die Bearbeitungsoberfläche des Anwenders, in der Simulationsmodelle geladen, bearbeitet, oder transformiert werden können. Über den Server können mehrere Clients an einem gemeinsamen Simulationsmodell arbeiten.

*Modellierungsserver*
Der Server ist zur Speicherung und Datenaufbereitung der Simulationsmodelle notwendig. Er kommuniziert über eine RMI-Socketverbindung mit den angeschlossenen Mainframe-Komponenten. Bei der Datenbankspeicherung werden die einzelnen Modellbausteine als Zeichenfolgen abgelegt. Durch die Einzelspeicherung können einzelne Bausteine in anderen Simulationsmodellen weiter verwendet werden. Damit die Zuordnung der Bausteininstanzen zu ihren Modellbausteinen nicht verloren geht, müssen eindeutige IDs gesetzt werden. Diese IDs werden bei der ersten Speicherung von der Datenbank vergeben und den entsprechenden Modellbausteinen hinzugefügt.

Beim Ladevorgang werden die gespeicherten Zeichenfolgen aus der Datenbank wieder zu einem XML-Dokument zusammengesetzt. Danach wird dieses XML-Dokument von einem Parser durchlaufen, der die Informationen in die Java-Objektklassenhierarchie der Modellbeschreibung überführt. Anschließend sendet der Server das erstellte Simulationsmodell über die Socketverbindung an die angeschlossenen Mainframe-Komponenten.

Angesteuert aus dem Mainframe der Clients, können Simulationsmodelle an das Transformationsmodul übergeben werden und werden dort nach dem beschriebenen Verfahren invertiert. Das Verfahren wurde durch einen Algorithmus implementiert, der auf Basis einer festzulegenden Reihenfolge Grundstrukturen in dem Simulationsmodell identifiziert und das Simulationsmodell vereinfacht. Aus Gründen der Performanz sollten häufig auftretende Grundstrukturen, wie zum Beispiel die unverzweigte Linie möglichst früh erkannt und zusammengefasst werden. Wichtig ist jedoch, dass die Grundstruktur *Rückkopplung* vor der Grundstruktur *unverzweigte Linie* gesucht und zusammengefasst wird, da ansonsten deren Knoten 2 und 3 (vgl. Abbildung 51) als *unverzeigte Linie* erkannt und zusammengefasst werden, wodurch die Grundstruktur der Rückkopplung zerstört und nicht mehr korrekt erkannt werden kann. Die eigentliche Invertierung wird in speziellen Klassen für jede Grundstrukturen einzeln implementiert.

*Modellierung - Mainframe*
Die Mainframe-Komponente stellt den Rahmen für die verschiedenen Modellierungs- und Visualisierungskomponenten dar und besteht aus einem Applikationsfenster, welches ein Menü im oberen und eine Statusleiste im unteren Bereich beinhaltet. In die Komponente eingebettet befindet sich im linken Teil ein internes Fenster, das die Struktur eines geöffneten Simulationsmodells und eventuell geöffneter Bibliotheken objektorientiert in Form eines Baums darstellt. Im rechten Bereich des Hauptfensters werden die Visualisierungs- und Modellierungskomponenten zur Darstellung des aktuell bearbeiteten Simulationsmodells eingebettet. Die Mainframe-Komponente bietet unter anderem Funktionen zum Laden und Speichern von Modellen, wobei in beiden Fällen zwischen Dateisystem- oder Datenbankebene ausgewählt werden kann. Weitere Funktionen sind z.B. die Neuerstellung eines Simulationsmodells oder aber das Hinzufügen eines Modellbausteins zu einer Bilbiothek, die Rechteverwaltung für das bearbeitete Simulationsmodell und Kommunikationsmöglichkeiten mit den anderen angeschlossenen Anwendern.

Das Simulationsmodell wird im linken Fenster als Baum dargestellt. Der Anwender öffnet ein Modell über das Kontextmenü, wobei standardmäßig die 2D-Modellierungsansicht ausgewählt wird. Verfügbar sind darüber hinaus die 2.5D- und 3D-Modellierung. Die gewählte Komponente wird im rechten Bereich des Clients als Fenster geöffnet. Die Baumdarstellung ermöglicht es, einzelne Bausteine einer Bibliothek direkt per Drag&Drop in das Simulationsmodell zu übertragen. Es besteht die Möglichkeit, ein bestimmtes Modell gleichzeitig in mehreren Ansichten zu öffnen. Um die Konsistenz zwischen diesen einzelnen Ansichten zu wahren, wird das Listener-Konzept verwendet. Hierbei registrieren sich die beteiligten Darstellungskomponenten auf den Modellelementen. Alle Darstellungskomponenten werden dann über Änderungen benachrichtigt, die in einer Ansicht vorgenommen wurden. Jede Darstellungsform passt daraufhin ihre Darstellung an. Außerdem bietet die Mainframe-Komponente die Möglichkeit, mehrere Fenster nebeneinander zu öffnen, um beispielsweise verschiedene Komponenten der Modellierung parallel zu betrachten. Abbildung 100 zeigt die laufende Applikation mit zwei in der 2D-Modellierungskomponente geöffneten Simulationsmodellen und veranschaulicht die Baumdarstellung sowie die angesprochenen Bedienelemente.

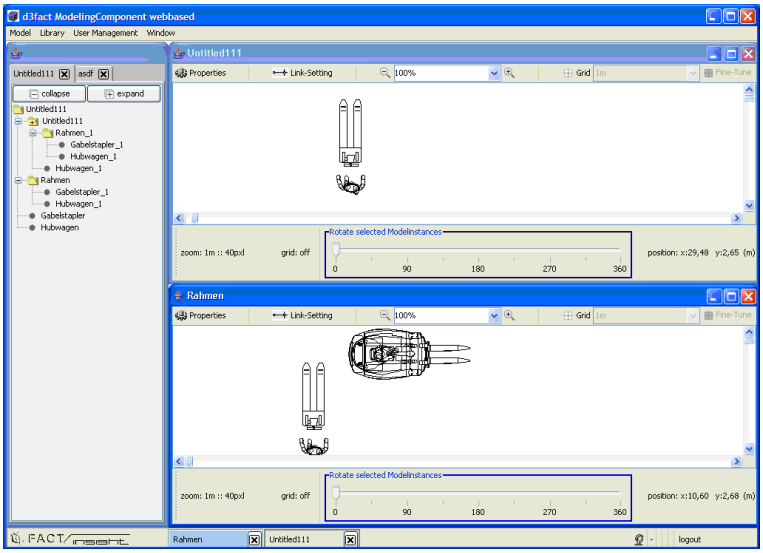


**Abbildung 100: Mainframe der Modellierungskomponente**

*Modellierungsumgebung 2D*

Die 2D-Modellierungskomponente ermöglicht die Erstellung neuer und das Modifizieren bereits vorhandener Simulationsmodelle. Hierbei werden für die einzelnen Bausteininstanzen die in der zentralen Datenbank enthaltenen Polygonzüge ihrer dreidimensionalen Repräsentanten angezeigt, um eine Draufsicht auf das Modell zu erhalten. Abbildung 101 zeigt beispielhaft die Darstellung zweier vorhandener Grafikmodelle.

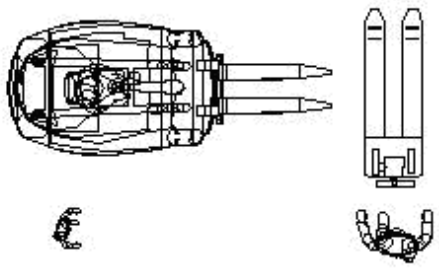

**Abbildung 101: Draufsichten 2D**

Sofern das angezeigte Modell einen Verweis auf einen in der Datenbank gespeicherten grafischen Repräsentanten hat, wird dieser angezeigt. Andernfalls wird ein Rechteck als visuelle Repräsentation einer Bausteininstanz verwendet. Die 2D-Modellierungskomponente ermöglicht das Setzen von Links zwischen den Bausteininstanzen bzw. zwischen den jeweiligen Channeln, um den Materialfluss abzubilden. Über ein Popup-Menü hat der Modellierer die Möglichkeit, das Modell zu rotieren, zu skalieren, zu löschen oder aber dessen Eigenschaften zu verändern. Die Maske zur Modifikation der Eigenschaften des geöffneten Simulationsmodells lässt sich über eine Funktionsleiste erreichen. In dieser befindet sich ebenfalls die Möglichkeit, den Sichtbereich schrittweise per Maus-Klick zu vergrößern bzw. zu verkleinern. Zur genaueren Ausrichtung der Modellbausteine besteht die Möglichkeit, ein Gitternetz einzublenden, das ebenfalls unterschiedlich skaliert werden kann.

*Event-Editor*
Um dem Anwender die Programmierung der einzelnen Events zu erleichtern, wurde der Event-Editor als Submodul des Mainframes konzipiert, mit dem auf alle einzelnen Ereignisse eines Simulationsmodells in der Modellierungsphase zugegriffen werden kann. Mit diesem Modul kann innerhalb einer Event-Programmierung auf eine klassische Programmieroberfläche zurückgegriffen werden, die anwenderfreundliche Funktionen wie die automatische Vervollständigung, Textbausteine, Online-Syntaxkontrolle, Syntax-Highlighting, etc. unterstützt. Einfache Programmierfehler können damit bereits frühzeitig erkannt werden. Bei der Speicherung des modellierten Ereignisses werden die verwendeten Befehle überprüft und ggf. Fehlermeldungen ausgegeben.

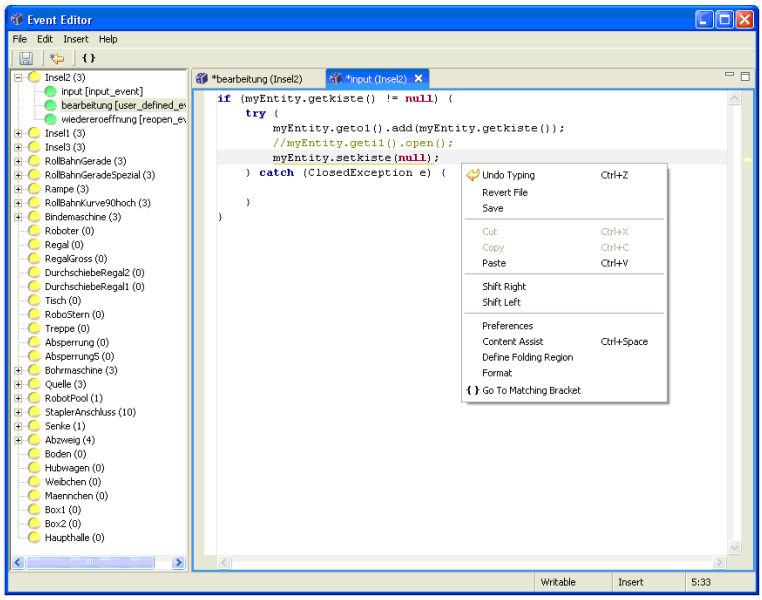


**Abbildung 102: Benutzeroberfläche des Event-Editors**

*Modellierungsumgebung 2.5D*

Ergänzend zur zweidimensionalen Draufsicht wurde in der Implementierungsphase eine 2.5D-Darstellung entwickelt, die das Simulationsmodell in der in Abbildung 103 dargestellten Form abbildet. Aus 8 verschiedenen Projektionsseiten kann das Simulationsmodell von schräg oben betrachtet werden, wobei die grafischen Repräsentanten der Bausteininstanzen Bildern des 3D-Repräsentanten entsprechen. Im Gegensatz zur zwei und dreidimensionalen Darstellung des Simulationsmodells in den entsprechenden anderen Modellierungsumgebungen wird hier der Navigationsbereich des Anwenders eingeschränkt. Im Gegenzug bildet die 2.5D-Darstellung eine schnelle und sehr intuitive Darstellungsform des Simulationsmodells, die zur Kommunikation mit Nicht-Simulationsexperten bereits verwendet werden kann.


**Abbildung 103: Modellierungsumgebung 2.5D**

Um die Effizienz der 2.5D-Darstellung mittels der Ladezeiten zu untersuchen, wurden Performance-Tests mit verschiedenen Verbindungsarten durchgeführt. Die nachfolgende Tabelle zeigt die Ergebnisse der Testdurchläufe.

| | Ladezeit 2D-Outlines | Ladezeit 2.5D-Images | Ladezeit 3D-Objekte |
|---|---|---|---|
| **LAN (1)** | <2 s | <2 s | 4 s |
| **WLAN A (2)** | 16 s | <2 s, 4 s, 10 s | 120 s |
| **WLAN B (3)** | 5 s | <2 s, 2 s , 6 s | 64 s |
| **DSL (4)** | 23 s | <2 s, 6 s , 31 s | 365 s |

**Tabelle 22: Performance-Test : Ladezeiten der 2.5D-Modellierungsumgebung**

Bei dem Vergleich der Werte zeigt sich, dass eine bis zu 60-fach höhere Geschwindigkeit im Vergleich zur 3D-Komponente erzielt werden konnte. Wenn man berücksichtigt, dass bereits Bilddateien im Cache-Ordner vorhanden sein können, fällt dieser Wert noch einmal deutlich höher aus. Die Ladezeiten in 2.5D sind geringer als die der 2D-Komponente.

*Modellierungsumgebung 3D*

Die 3D-Modellierungskomponente ermöglicht das Modellieren von Simulationsmodellen in einer realitätsnahen, dreidimensionalen Umgebung, in der sich der Anwender frei bewegen kann. Sämtliche Bausteininstanzen werden durch ihre 3D- Repräsentanten dargestellt. Über die Buttons „clone" und „remove" können bestimmte Kamerapositionen in der 3D-Szene gespeichert bzw. wieder entfernt werden, um dem Anwender eine zeitaufwendige Navigation zwischen verschiedenen Standardansichten im Simulationsmodell zu ersparen. Zwischen den verfügbaren Ansichten kann hin und her geschaltet werden. Abbildung 104 zeigt einen Screenshot der 3D-Modellierungskomponente und ihrer Bedienelemente.

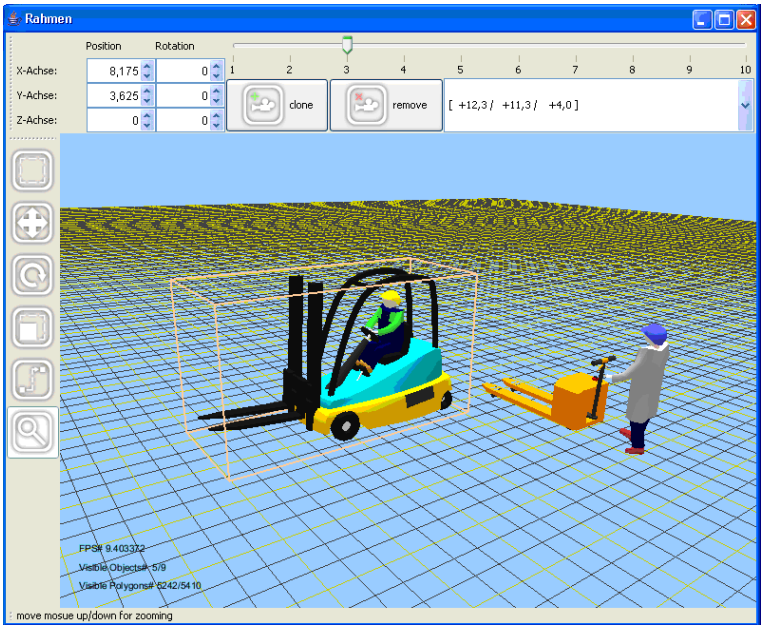


**Abbildung 104: 3D-Modellierungskomponente**

Wie oben aufgezeigt, sind die Ladezeiten der 3D-Modellierungskomponente deutlich höher als beispielsweise die der ähnlich intuitiven Darstellung in 2.5D. Diese

Darstellungsform bietet sich also insbesondere im Rahmen der exakten Layoutausrichtung an und weniger in der alltäglichen Modellierung.

Die vorgestellten Komponenten des Modellierungstools sollen nachfolgend dazu verwendet werden, die einzelnen Simulationsmodelle für die unter Abschnitt 6.2 definierten Szenarien zu erstellen. Dabei wird sich in den einzelnen Szenarien jeweils auf einen repräsentativen Ausschnitt des Simulationsmodells im gewählten Detaillierungsgrad beschränkt. Die Modellierung anderer Teilbereiche erfolgt analog zu der dargestellten Vorgehensweise. In den Szenarien wird sich auf Grund der Vergleichbarkeit auf die zweidimensionale Darstellungsform des Simulationsmodells im Modellierungstool beschränkt.

**Modellierung von Szenario 1:**
In diesem Abschnitt soll der Aufbau eines zeitorientierten Simulationsmodells am Beispiel des Zentrallagers mit der Modellierungskomponente dargestellt werden. Zu Beginn der Modellierung ist keine Bausteinbibliothek vorhanden. Die entsprechenden Modellbausteine des Zentrallagers müssen also in einem ersten Schritt mit dem Modellierungstool erstellt und in der Simulationsdatenbank gesichert werden. Vorraussetzung für eine erfolgreiche Arbeit mit dem Client der Modellierungskomponente ist ein entsprechender Server, der die Verbindung zur Datenbank und die Mehrbenutzerfähigkeit des Werkzeugs herstellt.

Nach dem Start des Client muss sich der Anwender zunächst anmelden, damit seine persönlichen Daten, Einstellungen und Rechte aus der Simulationsdatenbank geladen werden können. Zu Beginn der Bearbeitung wird eine neue Bibliothek *Zentrallager-zeitorientiert* angelegt und in der Simulationsdatenbank gesichert. Ihr werden nachfolgend die einzelnen Modellbausteine hinzugefügt, die für den Aufbau des Simulationsmodells benötigt werden. Für jeden Modellbaustein können seine Attribute durch den Anwender belegt werden. Darüber hinaus können Input- und Output-Channel angelegt und ein 3D-Repräsentant aus der Datenbank zugewiesen werden. Abbildung 105 zeigt die Benutzeroberfläche zum Anlegen eines neuen Modelbausteins in einer Bibliothek.

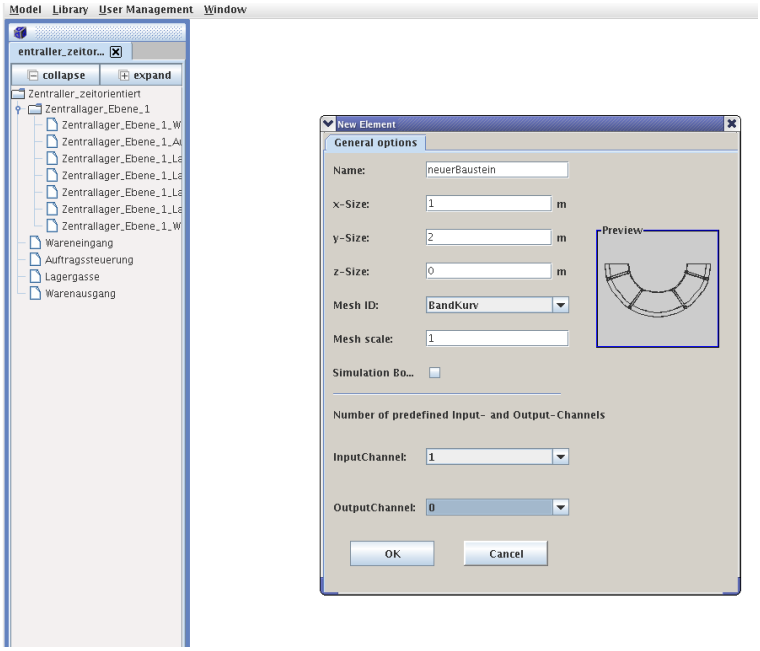


**Abbildung 105: Benutzeroberfläche zum Erstellen eines Modellbausteins**

Der Modellbaustein muss anschließend mit seiner Verhaltenslogik versehen werden. Dazu können in seinen Eigenschaften die entsprechenden Ereignisse angelegt, ggf. einzelnen Channeln zugewiesen und unter Zuhilfenahme des Event-Editors mit dem Programmcode versehen werden (vgl. Abbildung 102). Die inhaltliche Aufteilung der implementierten Verhaltenslogik muss sich an der unter Abschnitt 5.2.4.2 aufgezeigten Form orientieren, um eine spätere Transformation des Simulationsmodells gewährleisten zu können. So darf in den jeweiligen Input-Events eines Modellbausteins nur der Eingang der Token und die Weiterleitung an nachfolgendes Folge-Event erfolgen. Unter Umständen können hier noch Eingangszeiten mitprotokolliert werden, die eine spätere Auswertung hinsichtlich Bearbeitungszeiten etc. erlauben. Abbildung 106 zeigt exemplarisch den Programmcode eines Input-Events im Modellbaustein Warenausgang ( wegen der Rückwärtssimulation entspricht der Input-Channel hier dem Output-Channel des vorwärts gerichteten Modellbausteins, nimmt also die ausgehenden Kundenaufträge als Input in den Warenausgang auf, um deren Rückwärtsterminierung durch das Zentrallager zu ermöglichen.

```
<ichannel name="inchannel">
        <position y="20" x="312"/>
</ichannel>
<event>
        <input_event inchannel="inchannel_one"/>
        <code>
                logger.debug("incoming token in wa");
                Token t = myEntity.getinchannel("inchannel_one").removeToken();
                myEntity.getVariable("space").set(t);
                myEntity.getinchannel().open();

                OutputEvent e = new OutputEvent(myEntity);
                Kernel.schedule (e, 1000);
        </code>
</event>
```

**Abbildung 106: Quellcode des Input-Channels im Modellbaustein Warenausgang**

Die Zeitorientierung soll mittels des *bausteininternen* Verfahrens zur Modellierung erfolgen, also explizit durch den Baustein festgelegt werden. Dazu muss der Bibliothek neben den eigentlichen Modellbausteinen ein Schichtkalender hinzugefügt werden, der dem Modellbaustein zugewiesen werden kann. In dem Schichtkalender können für die abgebildeten Tage entsprechende Schicht- und Pausenzeiten hinterlegt werden. Die Schichten werden wochenweise durchgeführt, für spezielle Datumsangaben können freie Tage und zusätzliche Pausenzeiten hinterlegt werden. Das Zentrallager der PaderK soll nach einem einheitlichen Schichtmodell arbeiten. Allen Modellbausteinen kann also derselbe Schichtkalender zugewiesen werden. Darüber hinaus können in den einzelnen Modellbausteinen alle benötigten Variablen angelegt werden, wobei für die einzelnen Variablentypen umfangreiche Zusatzattribute ausgewählt werden können. Für alle numerischen Datentypen kann beispielsweise eine obere und untere Schranke und ihre späterer Darstellung in den entsprechenden Visualisierungsmodulen angegeben werden

Jeder Baustein beinhaltet eine Listenvariable *nextShifts*, die initial durch die Zeiten der Schichtwechsel aus dem zugewiesenen Schichtkalender befüllt wird. Die Terminierung der einzelnen Ereignisse innerhalb des Modellbausteins kann daraufhin durch eine Zuordnung der Schedulingzeit des Folgeereignisses zum zeitlich nächsten folgenden Schichtwechsel erfolgen, indem direkt auf diesen Schichtwechsel gescheduled wird. Die während der Ausführung einer Simulation auftretenden Ereignisse werden dadurch immer auf die Schichtwechsel terminiert und ermöglichen damit eine zeitorientierte Ausführung des Simulationsmodells. Innerhalb des Programmcodes eines Events wird die Bearbeitungszeit berechnet oder auf Basis einer Variablen des Bausteins belegt und der Bearbeitung folgende Schichtwechsel aus der Liste *nextShifts* ausgewählt. Das Folgeevent wird daraufhin mit diesem Wert im Simulatorkern gescheduled.

Wenn alle benötigten Modellbausteine der Bibliothek mit den benötigten Variablen, Channeln und Ereignissen formalisiert wurden, kann die Bibliothek in der Simulationsdatenbank gesichert werden. Dabei werden alle Modellbausteine hinsichtlich ihrer Vernetzung (Modellbausteine können weitere Modellbausteininstanzen enthalten) und der Syntax der in den Ereignissen beschriebenen Verhaltenslogik überprüft. Nur fehlerfreie Bibliotheken können gesichert werden. Syntax und Modellierungsfehler werden bereits in einer frühen Phase der Modellierung erkannt und erhöhen somit die Qualität der in der Simulationsdatenbank gespeicherten Daten. Der Anwender erstellt daraufhin ein neues Simulationsmodell *Zentrallager_Ebene_1*, zu dessen Modellierung die Bibliothek geladen werden kann. Per Drag & Drop können nun die einzelnen Modellbausteine im Simulationsmodell instanziiert und durch Links verknüpft werden. Die Variablen der Bausteininstanzen können durch den Anwender angepasst werden, wenn sie während der Modellierung der Bibliothek als *public* deklariert wurden und sich ihr Wert von der Standardparametrierung unterscheiden soll. Wenn beispielsweise die Lagergasse 1 des Zentrallagers 40 Lagerfächer mehr Kapazität als die übrigen Gassen hat, kann der Variablenwert *capacity* entsprechend um 40 erhöht werden. Die Instanz *lagergasse1* des Modellbausteins Lagergasse kann damit 40 Token mehr aufnehmen, wenn die entsprechende Verwaltung in dem Programmcode des entsprechenden Events die Kapazität aus dieser Variablen ausliest und daraufhin entscheidet, ob und wie viele Folgetoken im Materialfluss aufgenommen werden können. Abbildung 107 zeigt die Modellierung des Simulationsmodells *Zentrallager_Ebene_1* mittels der Bibliothek *Zentrallager-zeitorientiert*.

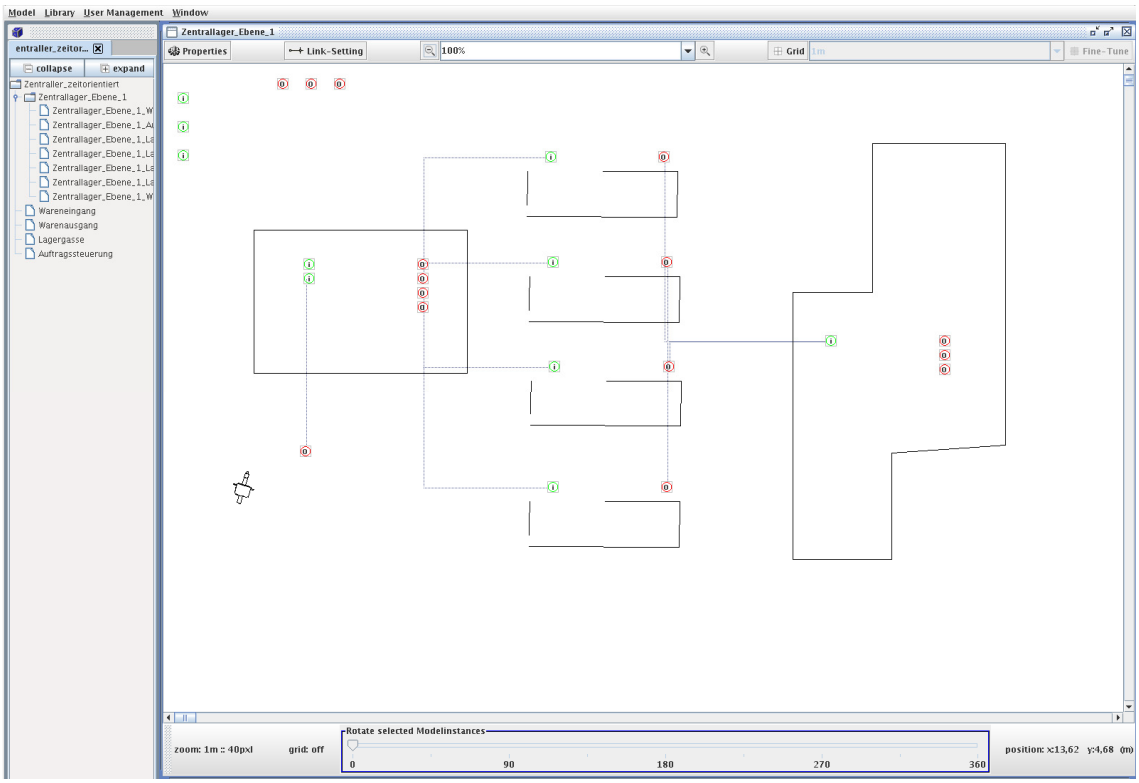


**Abbildung 107: Modellierung des Zentrallagers mit der 2D-Ansicht**

Das Simulationsmodell kann anschließend in der Simulationsdatenbank oder auf dem Dateisystem gesichert werden. Während des Speicherungsprozesses werden die Snytax des Simulationsmodells und die Gültigkeitsbereiche der Parameterwerte der Bausteininstanzen überprüft, sofern das durch das Werkzeug nicht bereits erfolgt ist. Die nachfolgende Phase der Modellverifikation und Verbesserung erfolgt durch erste Testläufe des Simulationsmodells im Simulatorkern. Sie wird in Abschnitt 6.5 beschrieben.

**Modellierung von Szenario 2:**

Im Unterschied zur zeitorientierten Modellierung des ersten Szenarios soll nachfolgend die Modellierung eines ereignisorientierten Simulationsmodells am Beispiel der Montage der PaderK erfolgen. Da sich die grundlegenden Prozesse bei der Modellierung nur unwesentlich unterscheiden, soll sich im Folgenden auf die Darstellung der Unterschiede bei der Modellierung beschränkt werden. Zum Nachweis der Multitasking, beziehungsweise Mehrbenutzerfähigkeit soll die Modellierung der Montage durch zwei Anwender erfolgen.

Die Anmeldung des zweiten Anwenders an der Modellierungskomponente gestaltet sich geringfügig anders, als in obigem Abschnitt beschrieben, weil er während der Autenthifizierung auf bereits existierende Sessions hingewiesen wird, denen er beitreten kann. Abbildung 108 zeigt den Anmeldedialog bei existierenden Sessions.

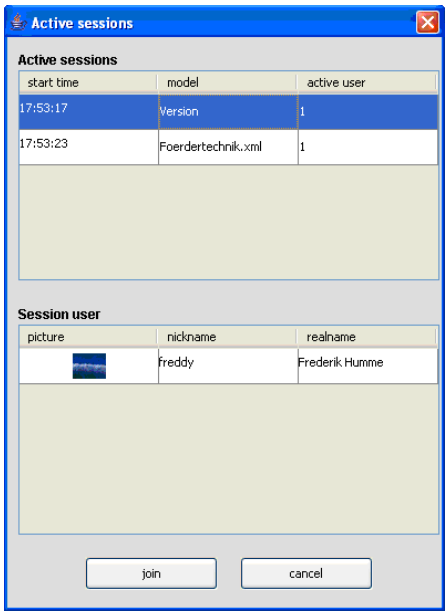


**Abbildung 108: Anmeldedialog bei vorhandenen Sessions**

Die Implementierung der unter Abschnitt 5.1.6 konzipierten Sperrmechanismen erfolgt in der Modellierungskomponente nach dem *Relaxed WYSIWYG*-Prinzip, bei dem das bearbeitete Simulationsmodell oder die Bibliothek auf dem Modellierungsserver verwaltet wird und mittels RMI an die angeschlossenen Clients verteilt wird. Relaxed WYSIWYG meint, dass nicht alle Änderungen sofort an die anderen angeschlossenen Clients übertragen werden, sondern erst, wenn der sperrende, also der aktuelle bearbeitende Anwender seine Aktion beendet hat. Dadurch wird der Nachrichtenaufwand zwischen dem Modellierungsserver und den angeschlossenen Clients erheblich reduziert. Die einzelnen Bausteine, die durch andere Anwender grade bearbeitet werden, werden in dem Client des Anwenders blockiert und farblich hervorgehoben. Abbildung 109 zeigt die Darstellung einer blockierten Bausteinstanz (rechts) im Vergleich zu einer selektierten Darstellung (links).

**Abbildung 109: Darstellung einer blockierten Bausteininstanz**

Das gleichzeitige Arbeiten mehrerer Anwender an einem Modell wird durch mehrere Kommunikationsmöglichkeiten erleichtert. Neben der implementierten Chat-Funktionalität steht die Möglichkeit zur Verfügung, einzelne Bausteininstanzen mit Notizen zu versehen, um anderen Anwendern auch asynchron Informationen zu kommunizieren, beispielsweise über identifizierte Fehler oder durchgeführte Änderungen. Darüber hinaus besteht die Möglichkeit, sich einzelne Dateien direkt über die Modellierungskomponente zuzusenden.

Die Modellierung der einzelnen Modellbausteine für die entsprechende Bibliothek *Montage_Ebene_2* unterscheidet sich nur unwesentlich von der in Szenario 1 beschriebenen Vorgehensweise. Sie ist für dieses Szenario ein wenig leichter, weil die aufwändigere Transformation der Folgeeventterminierung entfällt. Die einzelnen Folgeevents eines Ereignisses können direkt über fixe, variable oder zu berechnende

Variablen festgelegt und anschließend im Kernel gescheduled werden. Da für dieses Szenario die höher detaillierte Betrachtungsebene entscheidend ist, erhöht sich aber die Anzahl der benötigten Bausteine und ihrer Variablen. Abbildung 110 zeigt eine Darstellung des ereignisdiskreten Simulationsmodells der Montage im Modellierungstool.

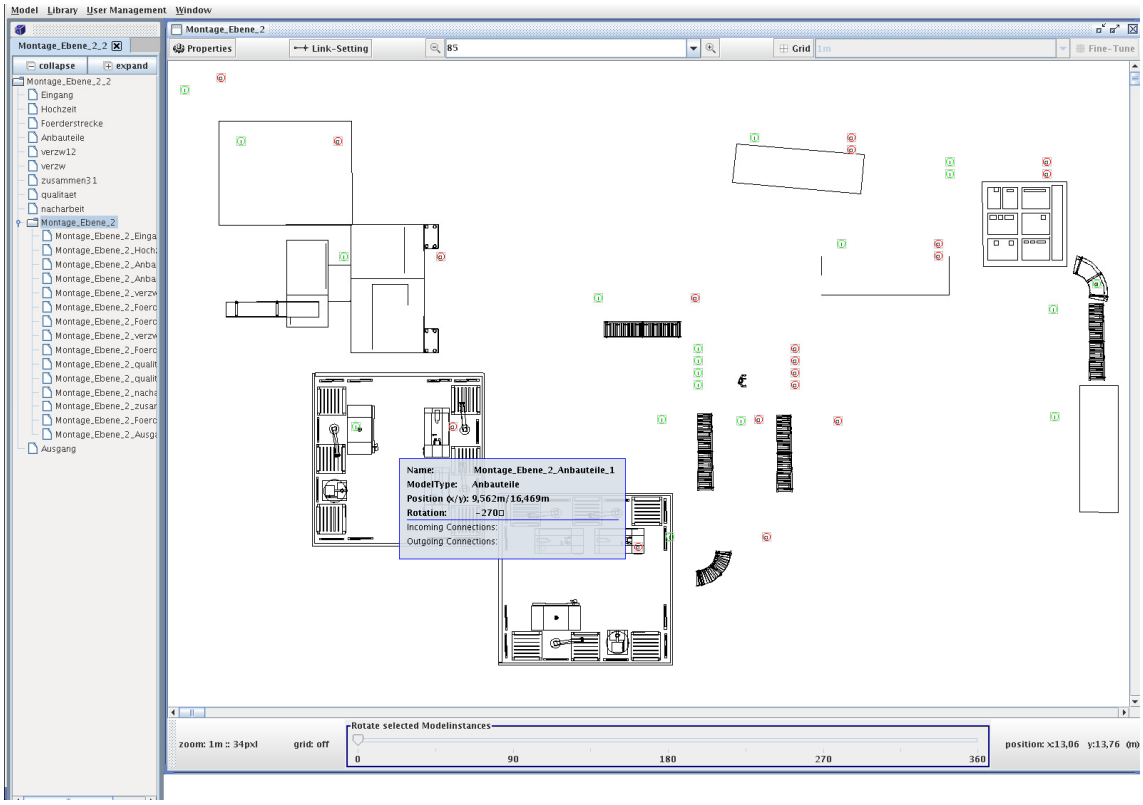


**Abbildung 110: Darstellung der Montage der PaderK im Modellierungstool**

Das analog erstellte, ereignisdiskrete Simulationsmodell der Teilefertigung soll im folgenden Szenario in ein rückwärts gerichtetes Simulationsmodell transformiert werden.

**Modellierung von Szenario 3:**

Weil die Montage der PaderK als Engpass des Gesamtmodells angenommen werden kann, soll die Feinplanung der Teilefertigung in engerer Abstimmung mit der Montage erfolgen. Dazu soll eine ereignisorientierte Rückwärtssimulation der Teilefertigung erfolgen, die als Eingabedaten die Abrufe der Montage zuzüglich der benötigten Verarbeitung im Zentrallager erhält. Das aus Szenario 2 bestehende Simulationsmodell der Teilefertigung mit Betrachtung der höheren Detaillierung soll deshalb mit dem in dem Werkzeug implementierten Mechanismus in ein rückwärts gerichtetes Simulationsmodell transformiert werden (aufgrund der hohen Detaillierung der Betrachtung an dieser Stelle kann nicht auf das Simulationsmodell aus dem ersten Szenario zurückgegriffen werden). Die hohe Detaillierung unter Berücksichtigung der Transporte ist für dieses Untersuchungsszenario wichtig, weil die Einplanung der Fertigungspläne in der Teilefertigung möglichst spät erfolgen soll, um die Kapitalbindung der PaderK durch zu hohe Sicherheitsbestände reduzieren zu können. Eine Betrachtung ohne die benötigten Transportzeiten würde das Ergebnis des Simulationsexperimentes zu sehr verfälschen.

Zur Umkehrung des Simulationsmodells der Teilefertigung muss das Ursprungsmodell zunächst durch den Anwender aus der Simulationsdatenbank in das Modellierungstool geladen werden. Anschließend muss das zu invertierende Simulationsmodell aus der Bibliothek selektiert werden und über das Kontextmenü *SimBack* die automatische Vereinfachung und Invertierung aufgerufen werden. Im Fall der Teilefertigung kann durch die relativ einfache Fertigungsstruktur auf Anhieb das gesamte Modell invertiert werden. Über die Baumansicht kann die Zerlegungsstruktur des Simulationsmodells der Teilefertigung betrachtet werden. Abbildung 111 zeigt die entstehende Vereinfachungsstruktur der Teilefertigung über die Baumvisualisierung des Intervierungsmoduls.

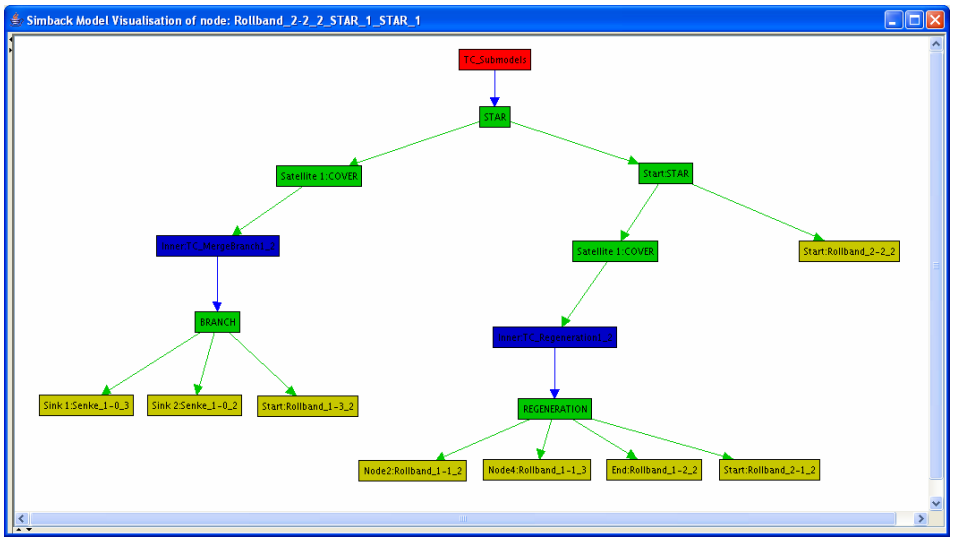


**Abbildung 111: Invertierung des detaillierten Modells der Teilefertigung**

Bevor das Resultat des Invertierungsprozesses als Rückwärtssimulationsmodell der Teilefertigung verwendet werden kann, müssen die Quellen und Senken des Ursprungsmodells durch ihre Pendants zur Rückwärtssimulation ausgetauscht und mit dem Simulationsmodell verknüpft werden. Das daraus resultierende Simulationsmodell kann anschließend in der Simulationsdatenbank gesichert werden. Im Rahmen der Modellverifikation ist aber insbesondere darauf zu achten, die Umkehrung der Verteilregeln zu überprüfen und ggf. anzupassen. Das soll bei der Betrachtung des dritten Szenarios in Abschnitt 6.5 geschehen.

**Administration**

Die Anmeldung eines Anwenders am Modellierungstool, die Modellierung der Bibliotheken, die Verknüpfung der 3D-Repräsentanten mit den neu erstellten Modellbausteinen setzt einen initialen Datenbestand in der Simulationsdatenbank voraus. Um eine Bearbeitung des Datenbestandes in der Simulationsdatenbank unabhängig von den Simulationsexperten ermöglichen zu können, wurde in der Implementierungsphase ein separates Administrationstool erstellt, das den direkten Zugriff auf Datenbereiche der Simulationsdatenbank ohne die Verwendung einer Modellierungs- oder Visualisierungsumgebung erlaubt. Insbesondere zur Wartung der Daten, zum Einpflegen neuer 3D-Modelle, zur Generierung der Outlines und 2.5D-Darstellungen der einzelnen Modelle, zum Löschen von alten oder inkonsistenten Datensätzen und zur Administration der Benutzer und Benutzergruppen kann dieses Tool herangezogen werden.

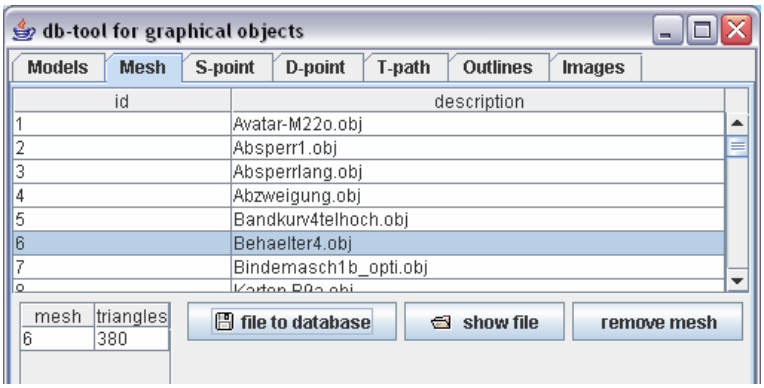


**Abbildung 112: Auszug aus der Oberfläche des Administrationstools**

Das Administrationstool kann unabhängig von den anderen Programmmodulen des entwickelten Werkzeugs arbeiten und erlaubt so die Übernahme von administrativen Funktionen durch Nicht-Simulationsexperten, beispielsweise der IT-Abteilung.

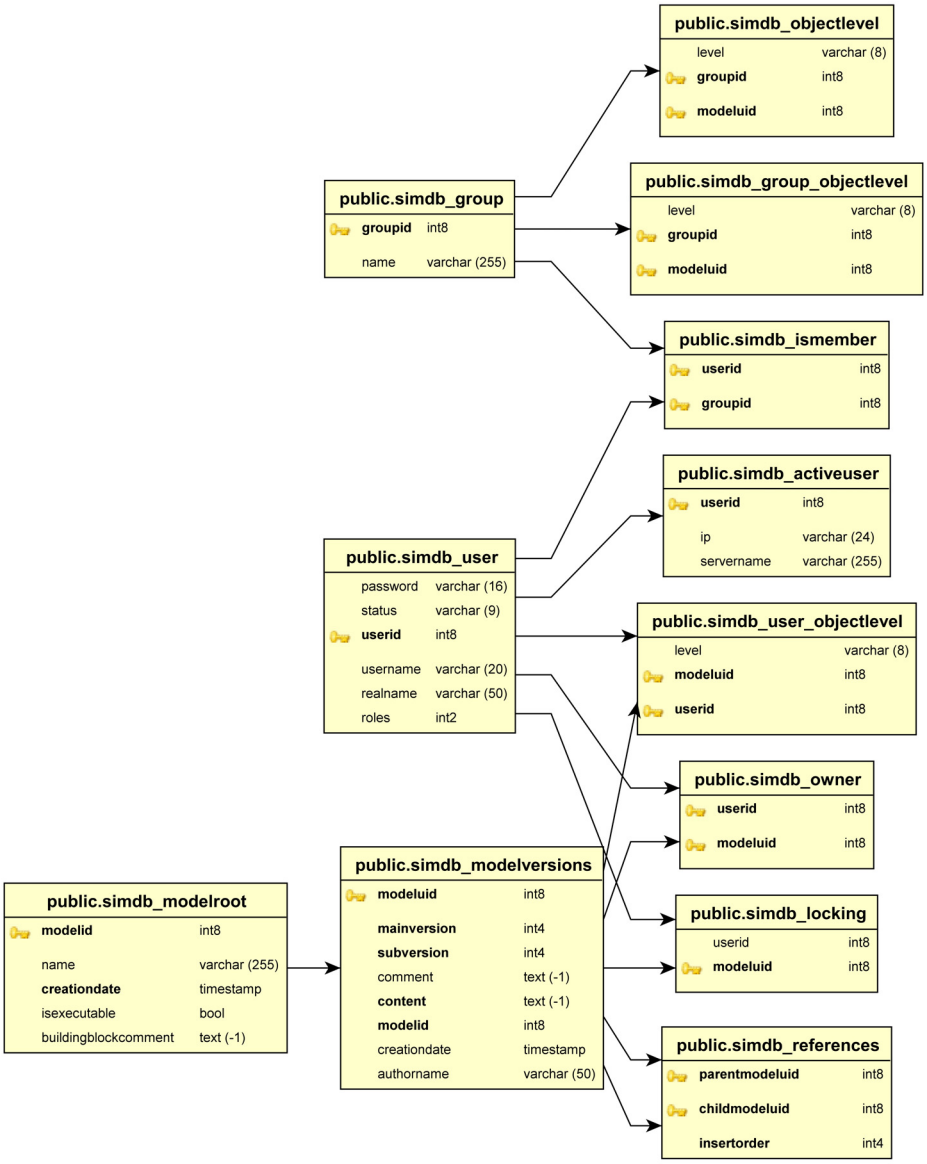


**Abbildung 113: Auszug aus dem Datenbankschema**

Die Simulationsdatenbank selbst wurde entsprechend der Festlegungen der Konzeptionsphase auf Basis eines PostgresSQL-Datenbanksystems implementiert. Abbildung 113 zeigt einen Teilbereich der Benutzerverwaltung als Auszug aus dem Datenbankschema der Simulationsdatenbank, in dem alle benötigten Tabellen, Primär und Fremdschlüsselbeziehungen hinterlegt sind. Durch die Implementierung des Rechtemanagements wird zwischen Gruppen und Personen, bzw. Gruppen- und Personenrechten unterschieden. Über die Zuordnungstabelle public_simdb.ismember werden beispielsweise einzelne Anwender bestimmten Gruppen zugewiesen. Die anderen Bereiche der Konzeption wurden entsprechend umgesetzt und bieten somit die Möglichkeit, Simulationsmodelle, ihre Eingabe- und Ausgabedaten, Experimente und grafische Repräsentanten in einer gemeinsamen Datenbank zu hinterlegen.

Die für die definierten Untersuchungsziele benötigten Simulationsmodelle konnten mit der Modellierungskomponente erstellt werden. Als wesentlicher, folgender Schritt innerhalb der Simulationsstudie müssen die einzelnen Simulationsmodelle im Folgenden verifiziert und verbessert werden. Gegebenfalls müssen für einzelne Untersuchungszwecke darüber hinaus Modellalternativen generiert werden, um Aussagen hinsichtlich spezieller Fragestellungen zu ermöglichen.

## 6.5 Modellverifikation und –Verbesserung

Die Verifikation der erstellten Simulationsmodelle kann in dem implementierten Werkzeug über verschiedene Module erfolgen, die sich gegenseitig ergänzen. Wesentliches Element ist aber der Simulatorkern, durch den die Ausführung des kompilierten Simulationsmodells gewährleistet wird. Darüber hinaus können in dieser Phase die zweidimensionale Visualisierung, das Reportingtool und die Debugging-Funktionalität der Modellierungskomponente eingesetzt werden. Die nachfolgenden Abschnitte sollen diese Module kurz darstellen, ehe die einzelnen Szenarien damit bearbeitet werden sollen. In jedem Szenario soll jeweils eine der drei aufgezeigten Möglichkeiten zum Einsatz kommen.

**Modul Simulatorkern**

Bei der Implementierung des Simulatorkerns wurde darauf geachtet, dass das Modul sowohl als eigenständige Applikation, als auch als integrierter Teil einer Applikation gestartet werden kann, jeweils mit oder ohne eigene grafische Benutzeroberfläche. In jedem Fall startet der Simulatorkern mit einem Preprocessing-Prozess, in der das übergebene Simulationsmodell vom XML-Format in ein lauffähiges Java-Programm geparst wird. Werden in dieser Phase keine Fehler gefunden, steht der Simulatorkern für die Ausführung des Simulationslaufs zur Verfügung. Im Fall der grafischen Oberfläche kann die Simulation nun einfach mit einem Maus-Klick gestartet werden (vgl. Abbildung 114). Im Fall einer Einbettung des Simulators kann die Simulation über eine Nachricht an den Simulator gestartet werden.

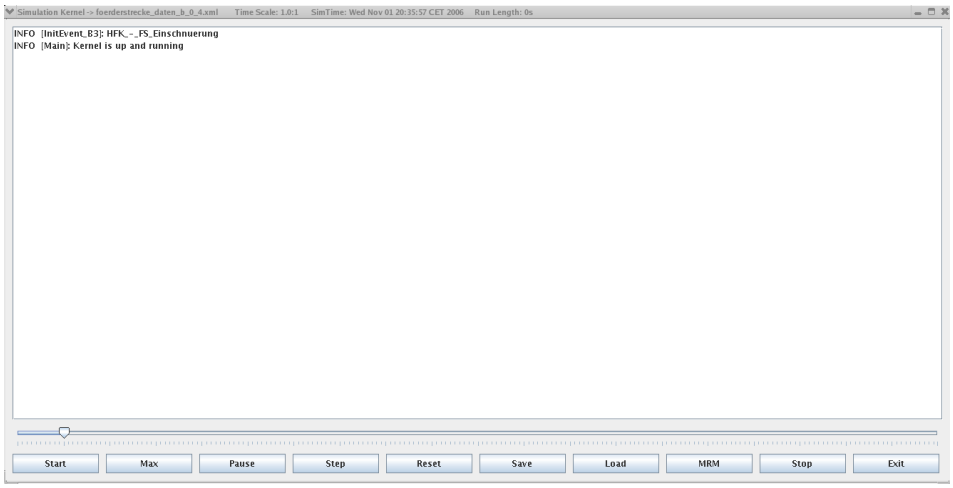


**Abbildung 114: Benutzeroberfläche des Simulators**

Je nach Modellierung der einzelnen Modellbausteine des Simulationsmodells werden die entsprechenden Module zum Motion Planning und zur dynamischen Detaillierung von Simulationsmodellen mit in das laufende Gesamtpaket integriert. Beim Aufbau der objektorientierten Klassenhierarchie der einzelnen Modellobjekte wird erkannt, welche Kernelfunktionen von den Bausteinen aufgerufen werden. Beziehen sich diese zumindest teilweise auf Funktionen aus den beiden genannten Teilmodulen, so werden diese dem Gesamtpaket hinzugefügt. Dadurch wird die Instanz des Simulatorkerns so schlank wie möglich gehalten, was in verschiedener Hinsicht von Vorteil ist. Zum Einen wird der Speicherbedarf des Simulators reduziert, zum Anderen sollen solche „Simulatorpakete" auf mehrere Rechner verteilt werden, um eine parallele Abarbeitung eines Simulationsexperimentes mit mehreren Simulationsläufen zu ermöglichen. Diese Verteilung kann umso schneller erfolgen, je kleiner das zur Verteilung bestimmt Paket ist.

**Modul Visualisierungskomponente**

Im Rahmen der Entwicklung von Simulationsmodellen spielt vor der eigentlichen Experimentierphase die Modellvalidierung und Verifikation eine große Rolle. Außer den generierten Daten ist in dieser Phase insbesondere die Visualisierung der dynamischen Abläufe von großer Bedeutung. Vor diesem Hintergrund wurden in der Implementierungsphase verschiedene Visualisierungsformen umgesetzt, die im Folgenden kurz erläutert werden.

*Visualisierungsumgebung 2D*

Die Darstellung der 2D-Visualisierungsumgebung erfolgt analog der Darstellung der 2D-Modellierung, jedoch angereichert um das dynamische Verhalten des Simulationsmodells. Während der Modellierung kann der Anwender das erstellte Modell mit einem integrierten Kernel übersetzen und sich das Verhalten in der bekannten Draufsicht animieren lassen. Die einzelnen Token werden durch Pakete visualisiert (vgl. Abbildung 115), wenn sie keine eigene 3D-Darstellung besitzen. Ansonsten werden die Draufsichten ihrer 3D-Repräsentanten verwendet.

**Abbildung 115: Animation von Token in der 2D-Ansicht**

Im Rahmen der Modellerstellung kommt es immer wieder zu logischen Fehlern bei der Modellierung der einzelnen Bausteine oder zu Programmierfehlern bei der Spezifikation der Verhaltenslogik innerhalb eines Modellbausteins oder einer übergeordneten Steuerung. Das visualisierte Modell verhält sich in diesem Fall nicht wie vom Anwender vorgesehen. Zur besseren Unterstützung des Anwenders bei der Fehlersuche wurde in den Mainframe des Modellierungs- und Visualisierungstools eine Funktionalität zum Debuggen der Simulationsmodelle integriert. Nach der Übersetzung des Simulationsmodells in ein lauffähiges Programm durch den Simulator wird das erzeugte Programmpaket mit der Debugging-Oberfläche verknüpft, aus der die weitere Steuerung erfolgt. Sowohl der Aufruf einzelner Funktionen, wie auch die Veränderung der Parameterwerte können jetzt erfolgen, indem in der Oberfläche an den relevanten Stellen entsprechende Haltepunkte gesetzt werden. Beispielsweise wird die Simulation beim Aufruf eines speziellen Input-Events einer Bausteininstanz angehalten, bei der der Anwender einen Fehler in der Programmierung vermutet. Abbildung 116 zeigt links die Oberfläche des Debugging-Tools und im rechten Bereich die entsprechende Steuerung des Simulators aus der 2D-Visualisierungsoberfläche heraus. Neben der Ausführungsgeschwindigkeit werden die wichtigsten Steuerungsbefehle implementiert, um eine Simulation zu pausieren oder zu beenden. Über spezielle Funktionen kann der Nachrichtenfluss zwischen Simulator und Visualisierungskomponente nachvollzogen werden. Fehlermeldungen werden identifiziert und aktuelle Wertebelegungen von Variablen betrachtet. Alternativ zur Durchführung eines Simulationsexperiments können somit auch die Ergebnisse eines einzelnen Simulationslaufs zur Weiterverarbeitung zur Verfügung gestellt werden

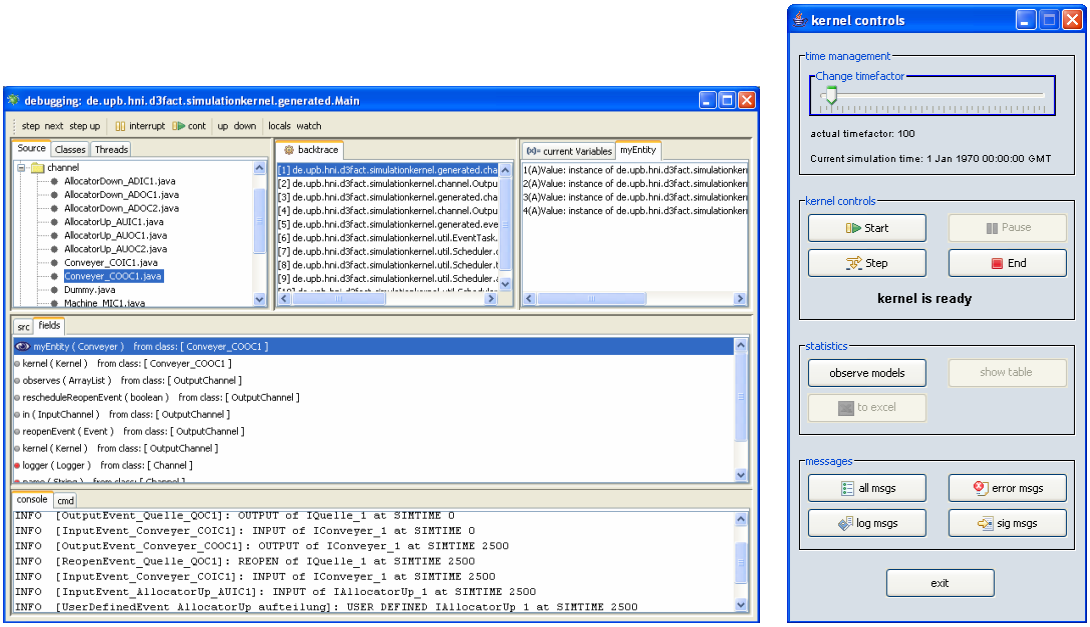


**Abbildung 116: Debugging und Simulator-Steuerung der 2D-Visualisierung**

Durch die Implementierung innerhalb des Mainframes stehen den Anwendern auch bei der Visualisierung von Simulationsläufen in der 2D, bzw. 2.5D-Darstellung die gleichen Kommunikationsmöglichkeiten zur Verfügung, wie sie aus dem Modellierungsbereich bekannt sind. Darüber hinaus können einzelne Daten zwischen den Anwendern über

einen direkten Dateiversand verschickt werden, um möglichst unkompliziert Arbeitsinhalte austauschen zu können.

*Visualisierungsumgebung 2.5D*

Die Visualisierungskomponente in 2.5D ist das Pendant zur entsprechenden Modellierungskomponente. Dadurch ergeben sich bei der Verwendung dieselben Vor- und Nachteile, die aus der Modellierung bekannt sind. Einer intuitiveren Betrachtung des Simulationsmodells steht eine Einschränkung des Anwenders in Bezug auf die Freiheitsgrade der Navigation gegenüber. Die Funktionsweise ist analog der 2D-Visualisierungskomponente.

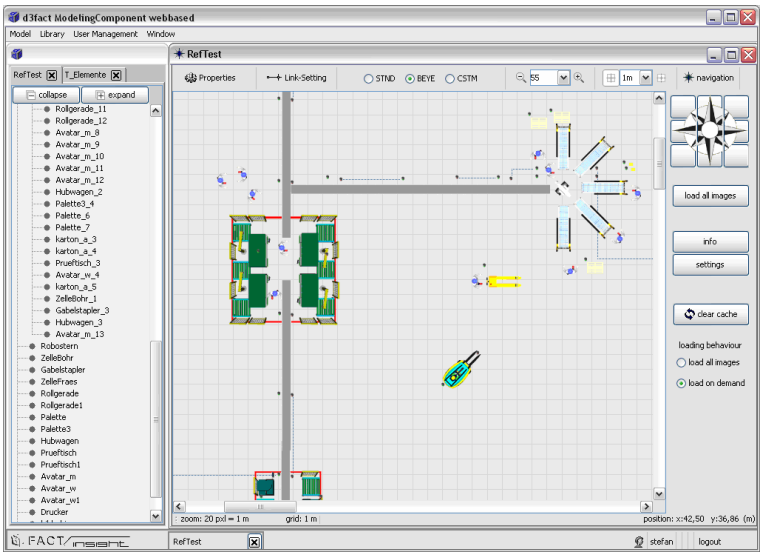


**Abbildung 117: Visualisierungsumgebung 2.5D**

Wie Abbildung 117 zeigt, ist die Darstellung des Simulationsablaufes in der 2.5D-Umgebung wenig immersiv, wenn auch das grobe Verhalten des Simulationsmodells dargestellt wird. Vor diesem Hintergrund erfährt die dreidimensionale Visualisierungskomponente hinsichtlich der Anforderungserfüllung eine steigende Bedeutung. Grundlegende Interaktionen werden aber auch aus den bereits dargestellten Visualisierungsumgebungen ermöglicht. So ist es möglich, die Steuerung des Simulationsmodells zu handhaben und einzelne Parameter während der Simulationsdurchführung interaktiv zu verändern.

*Reporting*

In Anlehnung an die Darstellung von Leitständen zur Fertigungsplanung und –Steuerung wurde zusätzlich eine Reporting-Oberfläche geschaffen, die sowohl als integrierte Applikation in den Mainframe des Modellierungs- und Visualisierungsclients, als auch als eigenständige Applikation verwendet werden kann. Sie bietet eine alternative Sicht auf das dynamische Verhalten eines Simulationsmodells, indem verschiedene Attribute der Bausteininstanzen durch einfache Symbole dargestellt werden. Je nach festgelegtem Auswertungstyp der Variablen in den Modellbausteinen kann somit automatisch eine Visualisierung generiert werden, wie sie Abbildung 118 zeigt. Darüber hinaus können einzelne Kennzahlen der Simulation dynamisch visualisiert werden, beispielsweise die

Ablaufstruktur aller zur Laufzeit existierenden Token, die Simulationszeit, der Nachrichtenverkehr des Simulators mit den Visualisierungskomponenten, etc. Auswertungsbausteine und ihre generierten Datentabellen werden dynamisch angezeigt und bieten so vorab eine gute Einschätzung der Qualität des Simulationslaufs. Für verschiedene Kennziffern, die in eingegrenzten Bereichen streuen, können dynamisch statistische Auswertungen visualisiert werden. Zum Start eines Simulationslaufs genügt die Auswahl eines Simulationsmodells. Die Übersetzung und Ausführung erfolgt danach durch einen integrierten Simulatorkern. Die Reporting-Oberfläche ist in der Lage, bereits vorhandene Simulationsläufe auf Basis des Simulationsmodells und dem gespeicherten Nachrichtenstrom erneut zu visualisieren, ohne sie erneut simulieren zu müssen. Dadurch wird eine nachgelagerte Analyse eines Simulationslaufs ermöglicht, wenn aus einer Reihe von Simulationsläufen beispielsweise die Extremata hinsichtlich ihrer Ursachen nachträglich untersucht werden sollen. Dadurch kann der Anwender abschätzen, ob der betrachtete Simulationslauf spezielle Bedingungen erfüllt hat, die bei der bisherigen Planung nicht berücksichtigt oder erkannt wurden.

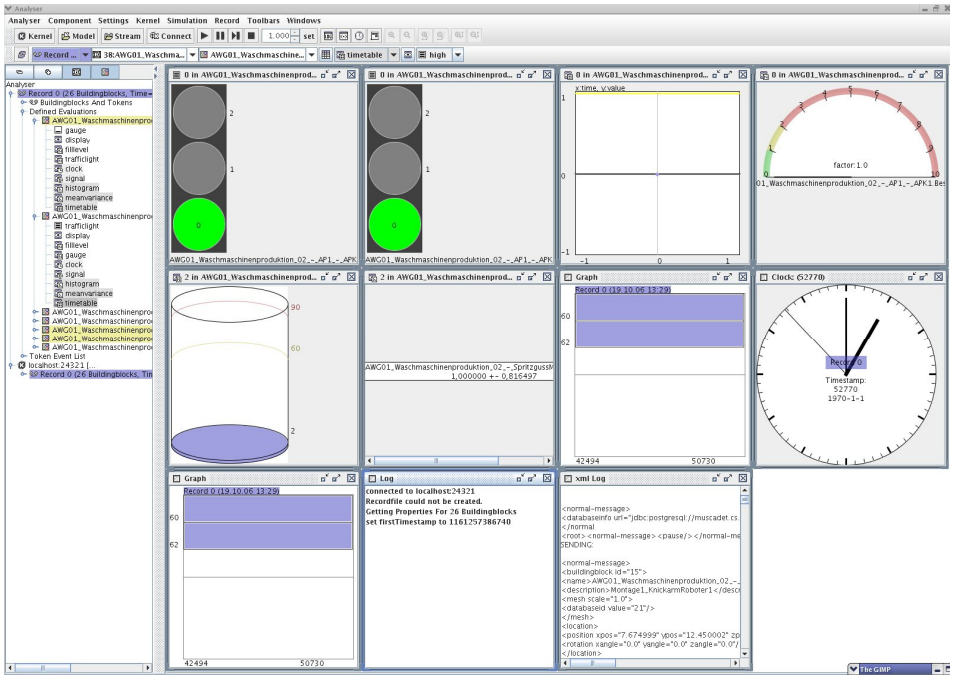

**Abbildung 118: Benutzeroberfläche des Reportingtools**

Die mittels dem Reporting-Modul erstellen Simulationsläufe können als Experiment in der Simulationsdatenbank gesichert werden.

### Szenario 1: Modellverifikation des rückwärts gerichteten, zeitorientierten Simulationsmodells des Zentrallagers mit der Debuggingfunktionalität

Die Funktionsweise des rückwärts gerichteten, zeitorientierten Simulationsmodells des Zentrallagers soll mittels der Debugging-Funktionalität der Modellierungskomponente überprüft werden. Dabei wird für das aktuell geladene Simulationsmodell direkt der Preprocessor des Simulators aufgerufen, der das Simulationsmodell kompiliert und in einer Debugging-Oberfläche startet, wie sie aus Entwicklungsumgebungen bei der Programmierung von Softwaresystemen bekannt ist (vgl. Abbildung 116). Die entsprechenden Einstellungen zum Starten des Preprocessors werden dabei soweit wie

möglich automatisch generiert. Alle anwenderabhängigen Startparameter (Start- und Endzeit der Simulation, Ausführungsgeschwindigkeit, Speicherort der temporären Dateien, etc.) können über ein Untermenü der Modellierungskomponente parametriert werden. Die gestartete Debugging-Oberfläche bietet verschiedene Funktionen zur Kernelsteuerung an und erlaubt darüber hinaus die Ansicht aller aktuell laufenden Prozesse, Variablenwerte, Bausteininstanzen usw. Der Programmcode der übersetzen Ereignisse kann eingesehen werden; durch das Setzen von Breakpoints kann die Simulation durch den Debugger beim Erreichen der Codestellen angehalten werden. Zusätzlich können an alle Objekte und deren Variablen Watchpoints gesetzt werden, die die Ausführung der Simulation im Simulator genau dann pausieren, wenn auf die markierten Objektinstanzen zugegriffen wird.

Für den Anwender ergibt sich damit die Möglichkeit einer einfachen und individuellen Modellanalyse und Überprüfung der verschiedenen Objektzustände zu ausgewählten Zeitpunkten der Simulation. Einzelne Objekte können in ihrer Bewegung durch das Simulationsmodell nachverfolgt und damit der Ablauf der Simulation überprüft werden. Die einzelnen Ereignisse und Funktionsmethoden können hinsichtlich ihrer Korrektheit überprüft werden, weil nach Erreichen eines Break- oder Watchpoints der Debugger das Voranschreiten der Simulation nicht nur „Ereignis für Ereignis", sondern „Codezeile für Codezeile" ermöglicht. Dadurch kann eine sehr detaillierte Fehlersuche erfolgen, die den Gesamtprozess der Modellverifikation beschleunigt. Für fehlerfrei ausführbare Simulationsmodelle bietet sich die Reporting-Oberfläche zur Modellanalyse an, mit der eine anwenderspezifische Übersicht über einzelne Werteverläufe und Variablenzustände während der Simulation realisiert wird.

Für das zeitorientiert simulierte Simulationsmodell des Zentrallagers der PaderK kann mit dieser Methode effektiv überprüft werden, wie sich die einzelnen Parameter während eines Schichtwechsels schrittweise verändern und ob die Abläufe richtig in den entsprechenden Events formalisiert wurden. Darüber hinaus kann der Scheduler des Simulatorkerns selbst eingesehen werden, um zu überprüfen, ob die Ereignisse in der korrekten Reihenfolge eingepflegt beziehungsweise aufgerufen werden. Darüber hinaus kann beispielsweise während der Initialisierung des Simulationsmodells überprüft werden, ob die zugewiesenen Schichtwechsel des Kalenders korrekt in die vorgesehenen Listen eingetragen werden, damit die Folgereignisse entsprechend korrekt terminiert werden können.

**Szenario 2: Modellverifikation des ereignisorientierten Simulationsmodells der Montage mit dem Reportingtool**

Nachdem die grundlegende Funktionsweise eines Simulationsmodells durch den Anwender überprüft wurde, beispielsweise durch die Überprüfung mittels der beschriebenen Debugging-Funktionalität, müssen in einem Folgeschritt die einzelnen Parameter der Bausteininstanzen überprüft und mit der Realität oder den Plandaten verglichen werden. Daraufhin kann in einem weiteren Schritt ein erster Simulationslauf durchgeführt werden, an dem das grundlegende, dynamische Verhalten des Simulationsmodells überprüft werden soll. Neben der später verwendeten 2D-Visualisierung kann hierzu das Reporting-Modul des implementierten Werkzeugs verwendet werden.

Das ereignisorientierte Simulationsmodell der Montage aus dem Szenario 2 kann in der Applikation Reporting aus der Simulationsdatenbank geladen und an einen eingebetteten Simulator übergeben werden. Es wird daraufhin übersetzt, kompiliert und initialisiert, wobei über das Kommunikationsprotokoll alle Parameter und Initialwerte der Bausteininstanzen an das Reportingtool übertragen werden. Es erlaubt eine komfortable Steuerung des Simulators und bietet die Möglichkeit, sich individuell die einzelnen Variablen der Bausteininstanzen der Montage grafisch auswerten zu lassen. Darüber hinaus kann der Fluss aller durch das Simulationsmodell laufenden Token, der Nachrichtenverkehr zwischen Simulator und Visualisierungskomponente, die Simulationszeit und weitere Eigenschaften der Simulation angezeigt werden. Die Bedienoberfläche kann dabei durch den Anwender frei mit den einzelnen Auswertungsfenstern belegt werden, so dass eine modular aufgebaute „Leitstand"-ähnliche Sicht auf das Simulationsmodell erlaubt wird.

Für das ereignisdiskrete Simulationsmodell der Montage der PaderK kann unter Verwendung des Reportingtools die Auslastung aller Fördertechnik-Puffer gleichzeitig beobachtet und in Abhängigkeit vom Simulationsverlauf analysiert werden. Darüber hinaus können die kundenindividuellen Aufträge, die als Token die Montage durchlaufen, nachverfolgt und ihr spezifischer Weg durch die Montage verfolgt werden. Als zusätzliche intuitive Darstellung des Flusses der Token durch das Simulationsmodell während der Simulation kann die 2D-Visualisierung.

**Szenario 3: Modellverifikation des rückwärts gerichteten, ereignisorientierten Simulationsmodells mit der 2D-Visualisierung der Modellierungskomponente.**
Die anwenderfreundlichste Darstellung des simulierten Materialflusses kann innerhalb der Modellverifikationsphase durch die 2D-Visualisierungskomponente erfolgen. Zwar ist die Darstellung in der 3D-Visualisierungskomponente generell auch möglich, die zweidimensionale Darstellung ist jedoch direkt in die Modellierungskomponente integriert. Analog zum Starten, Übersetzen und Kompilieren eines Simulationsmodells über die Debugging-Funktionalität wird das geladene Simulationsmodell durch einen internen Simulatorkern übersetzt und die Kommunikationsnachrichten durch die 2D-Visualisierung entsprechend interpretiert. Der Fluss der Token durch das Simulationsmodell kann dadurch auf Basis des zweidimensionalen Layouts nachvollzogen werden (vgl. Abbildung 115) und erlaubt somit eine schnelle Analyse der modellierten Verkettungen und Verteilregeln in den einzelnen Bausteininstanzen. Die einzelnen Token werden dabei durch Marken dargestellt, können aber nicht individuell parametriert werden. Zusätzlich ermöglicht die 2D-Visualisierungskomponente aber bereits eine einfache Manipulation der Bausteinparameter, um identifizierte Schwachstellen und deren Behebung direkt in der Simulation überprüfen zu können.

Zur Modellverifikation des rückwärts gerichteten Simulationsmodells der Teilefertigung im Szenario 3 bietet sich diese Darstellungsform insbesondere an, um eine schnelle Überprüfung der Invertierung des Simulationsmodells realisieren zu können. Fragestellungen wie „Wurden alle Verteilregeln entsprechend in die richtigen Bausteininstanzen transformiert?", „Wurden alle Prioritätsregeln korrekt durch ihr jeweiliges Pendant ersetzt?" können schnell beantwortet werden, weil die Dynamik des Simulationsmodells in einer intuitiven Form dargestellt wird.

Auf die Entwicklung von alternativen Simulationsmodellen für weitergehende Fragestellungen soll an dieser Stelle verzichtet werden, weil sie für den Nachweis der Funktionalitäten des implementierten Werkzeugs nicht erforderlich sind. Im nachfolgenden Abschnitt sollen die einzelnen Untersuchungsszenarien durch die entsprechenden Module des Werkzeugs simuliert werden, indem zum Einen einzelne Experimentreihen mit dem Experimentmanager definiert und ausgeführt werden und zum Anderen Simulationsläufe einer interaktiven Analyse mit der dreidimensionalen Visualisierungskomponente zugeführt werden.

## 6.6 Simulationsexperiment

Die im vorigen Abschnitt verifizierten Simulationsmodelle stehen nun für die Experimentierphase der Simulationsstudie zur Verfügung. Die Simulationsexperimente können hinsichtlich zweier unterschiedliche Strategien durchgeführt werden. Zum Einen können Simulationsläufe ohne angeschlossene Visualisierungskomponenten möglichst schnell berechnet werden, um die Experimentdaten zu generieren; zum Anderen kann das dynamische Verhalten des abgebildeten Systems interaktiv mit einer Visualisierungskomponente analysiert werden. In dem implementierten Werkzeug stehen für beide Aufgaben jeweils ein Modul zur Verfügung, die nachfolgend kurz beschrieben werden sollen.

**Modul Experimentmanager**

Für die Parametrierung eines kompletten Simulationsexperimentes wurde der Experimentmanager implementiert. In Anlehnung an einen Wizard kann Anwender hier schrittweise ein Simulationsexperiment anlegen oder laden, konfigurieren, durchführen und eine Übersicht über die Ergebnisse erhalten. Der Simulatorkern ist in den Experimentmanager eingebettet, so dass aus einer einheitlichen Oberfläche die entsprechenden Versuchsreihen parametriert, durchgeführt und in der Simulationsdatenbank gespeichert werden können. Nach dem Start legt der Anwender dazu ein neues Simulationsexperiment an oder lädt ein vorab definiertes Szenario aus der Simulationsdatenbank. Danach kann er Start- und Endzeit der Simulation und Anzahl der benötigten Simulationsläufe festlegen (vgl. Abbildung 119 links). Auf diesen Angaben basierend erhält der Anwender eine Liste aller zu parametrierender Variablen, die nach der Struktur des Simulationsmodells in einer Baumstruktur geordnet sind und durch verschiedene Filter nach Datentypen oder Bausteinen eingegrenzt werden können (vgl. Abbildung 119 rechts). Für alle Startwerte der Zufallszahlen dieses Experimentes steht eine Funktion zur Verfügung, um die entsprechenden Werte zu Beginn jedes Simulationslaufs zu verwirbeln (Beispielsweise in der Form: Lauf 1: 1-2-3, Lauf 2: 2-3-1, Lauf 3: 3-1-2, etc.).

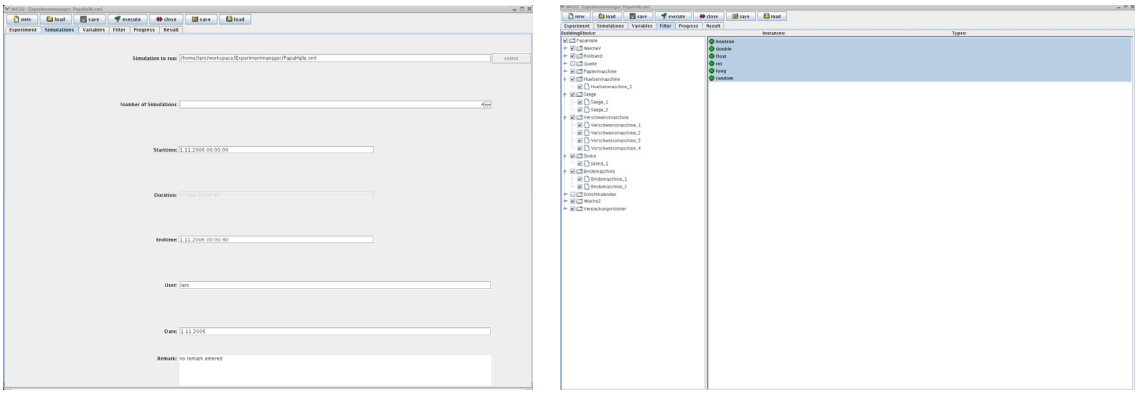

**Abbildung 119: Ansichten des Experimentmanagers**

Zusätzlich zur Parametrierung der einzelnen Attribute der Modellbausteine kann der Anwender diejenigen Attribute auswählen, deren Werteveränderungen über einen Simulationslauf protokolliert werden sollen. Neben Auswertungsbausteinen etc, können so auch weitere interessante Attribute einzelner Bausteine verfolgt werden. Nach der vollständigen Parametrierung können die einzelnen Simulationsläufe als Stapelverarbeitung gestartet werden. Alternativ bietet sich die Verteilung auf einem Rechencluster an, für die entsprechende zusätzliche Attribute angegeben werden müssen. Die Parallelisierung bezieht sich ausschließlich auf die Verteilung der einzelnen Simulationsläufe, das heißt, ein einzelner Lauf kann nicht auf mehreren Rechnern verteilt werden. Nach dem Durchlauf aller Simulationsläufe werden die entsprechend ausgewerteten Kennziffern dargestellt. Sie dienen einer ersten Übersicht der generierten Datenmenge und lassen sich zwischen den verschiedenen Simulationsläufen vergleichen. Das Simulationsexperiment kann wieder in der Simulationsdatenbank gespeichert werden und steht für spätere, umfangreichere Analysemethoden zur Verfügung.

**Visualisierungskomponente**

*Visualisierungsumgebung 3D*

Die dreidimensionale Visualisierungskomponente wurde als umfangreichstes Analysemodul in Form einer eigenständigen Applikation implementiert, die sich direkt an einen Simulatorkern ankoppelt. Zur Darstellung besonders großer und damit auch komplexer Szenen wurden spezielle Grafikalgorithmen implementiert, die eine echtzeitfähige Analyse großer Simulationsmodelle ermöglichen. Zur Implementierung konnte hier auf umfangreiche Arbeiten der Fachgruppe „Algorithmen und Komplexität" des Heinz Nixdorf Instituts zurückgegriffen werden. Neben der dynamischen Visualisierung der simulierten Abläufe kann diese mehrbenutzerfähige Visualisierungs-komponente den Anwender zu ausgewählten Prozesspunkten führen und bietet neben Mini-Map und dreidimensionaler Darstellung weitere Unterstützungsfunktionalitäten für den Anwender an. Analog zur Modellierungskomponente wurden auch hier Kommunika-tionsmechanismen wie eine Chatfunktion implementiert, um eine gemeinsame Analyse von Simulationsmodellen auch dann zu erlauben, wenn die beteiligten Anwender nicht an einem Ort sind.

Die dreidimensionale Visualisierungskomponente erlaubt eine interaktive Manipulation des berechneten Simulationslaufs, indem einzelne Parameter der Bausteininstanzen ausgewählt und innerhalb der vom Modellierer festgelegten Grenzen manipuliert werden können. Dem Anwender wird zusätzlich auch dadurch eine möglichst immersive

Umgebung präsentiert, dass sein „virtuelles Ich", der Avatar, sich nur auf denjenigen Wegen durch die virtuelle Fertigung bewegen kann, wie es dem Anwender auch in der Realität möglich wäre. Unter Verwendung des Motion Planning Moduls im Simulatorkern, werden Kollisionen mit vorhandenen SPMs und Avataren verhindert.

Abbildung 120 zeigt einen Screenshot der Benutzeroberfläche des 3D-Clients, in der die einzelnen Funktionsfenster gut zu erkennen sind. Mit den hier vorgestellten Werkzeugmodulen sollen die Untersuchungsziele mittels verschiedener Simulationsexperimente erarbeitet werden. Die folgenden Abschnitte beschreiben dazu jeweils die Vorgehensweise.

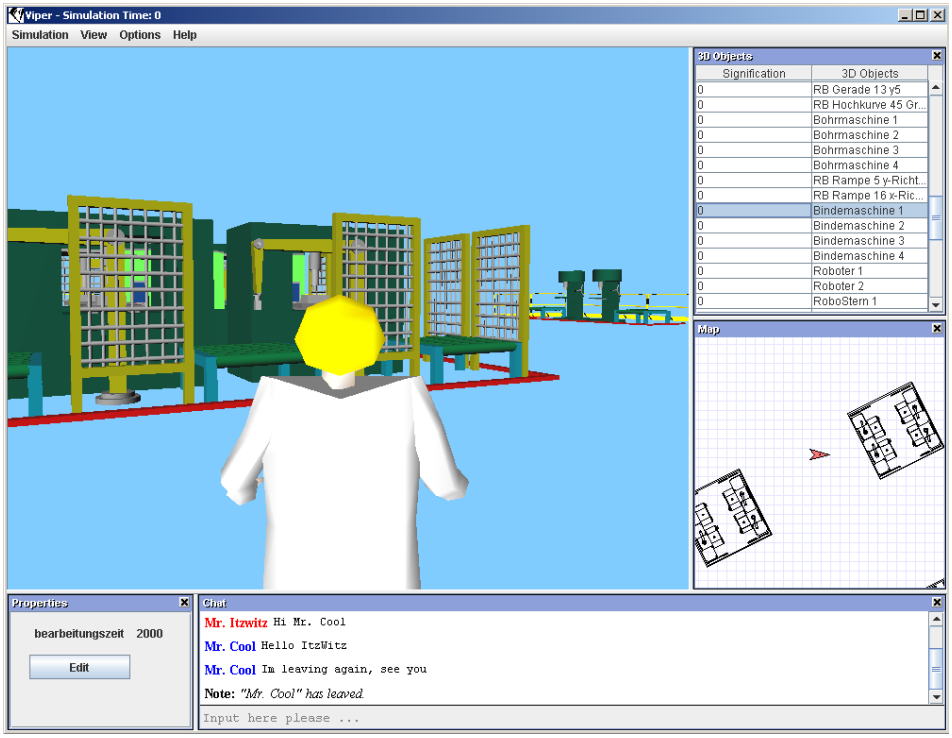


**Abbildung 120: Benutzeroberfläche des 3D-Clients**

**Szenario 1: Zeitorientierte Rückwärtssimulation zur Terminierung von Fertigungsplänen**

Die Parametrierung der Experimentreihe zur zeitorientierten Rückwärtssimulation erfolgt mit Hilfe des Experimentmanagers. Auf der hier abgebildeten, weniger detaillierten Betrachtungsebene sind nur wenige Variablen zufallsverteilt, so dass nach dem Laden des entsprechenden Simulationsmodells nur die Anzahl der Simulationsläufe, Start- und Endzeiten der Simulation und zusätzliche Kommentare durch den Anwender angegeben werden müssen. Die jeweiligen Parameterwerte der Bausteininstanzen können in der Übersicht schnell überprüft werden. Anschließend werden die einzelnen Simulationsläufe als Stapelverarbeitung gestartet und die Simulationsdaten generiert.

Nach der Simulation bietet der Experimentmanager eine Übersicht der in den Simulationsläufen generierten Daten, so dass der Anwender eine erste Analyse durchführen kann. Die Ergebnisse des Simulationsexperimentes können dann in der Simulationsdatenbank gesichert werden und stehen für eine spätere grafische Auswertung zur Verfügung.

**Szenario 2: Ereignisdiskrete Vorwärtssimulation zur Engpass- und Sensitivitätsanalyse der Fertigungsabläufe**

Neben der Durchführung von Experimentreihen, in denen die Simulationsdaten so schnell wie möglich berechnet werden sollen, soll im zweiten Szenario eine interaktive Sensitivitätsanalyse mit dem implementierten Werkzeug durchgeführt werden. Dazu wird das entsprechende Simulationsmodell, hier der Montage der PaderK, in den Simulator geladen und die dreidimensionale Visualisierungskomponente gestartet. Die dreidimensionale Visualisierung der Montagehalle wird geladen, ihre dynamische Verhaltensweise wird animiert und kann durch den Anwender manipuliert werden.

Zur Sensitivitätsanalyse kann der Anwender nun durch die virtuelle Montage navigieren und die Dynamik des abgebildeten Systems verstehen. Dafür ist es wichtig, dass der Faktor der Ausführungsgeschwindigkeit der Simulation nahe der Echtzeit liegt, um die realistische Darstellung der Fertigungsabläufe darstellen zu können. Einzelne Parameter einer Bausteininstanz können jetzt durch den Anwender manipuliert und die Auswirkungen in der Simulation direkt beobachtet werden. Löst der Anwender in der virtuellen Umgebung beispielsweise in der Bausteininstanz *Förderstrecke 2* eine Störung aus, so kann beobachtet werden, wie der Transport in den Qualitätsbereich ausschließlich über die Förderstrecke 3 erfolgt. Wird analog dazu durch den Anwender beispielsweise der Status vieler durch das System laufender Karts (abgebildet durch die Token) auf *fehlerhaf*t gesetzt, so werden diese Karts in den Nacharbeitsbereich ausgeschleust. Der daraus resultierende Rückstau kann direkt in der Visualisierungskomponente beobachtet werden. Unter Umständen können weitere Maßnahmen in das Simulationsmodell eingepflegt werden, die eine realistische Reaktion auf diesen Fertigungsablauf darstellen.

Neben der Sensitivitätsanalyse muss eine ereignisorientierte Vorwärtssimulation des Gesamtmodells erfolgen, die über den Experimentmanager eingestellt werden kann. Weil in der detaillierten Betrachtungsweise viele Parameter einer Zufallsverteilung unterliegen, kann hier die Unterfunktion des Experimentmanagers angewendet werden, die eine Umverteilung der Startwerte aller Zufallsverteilungen für jeden Simulationslauf automatisch durchführt. Nach dem Simulationsexperiment zeigt die Analyse im Experimentmanager, dass die Auslastung der einzelnen Fertigungsstufen der Montage durchgehend hoch ist (vgl. Abbildung 121), wohingegen Zentrallager und insbesondere die personalintensive Teilefertigung eine geringere Auslastung zeigen. Zur verbesserten Fertigungsplanung soll im dritten Szenario eine ereignisorientierte Rückwärtssimulation durchgeführt werden, um die spätesten Beginn-Zeitpunkte der Fertigungsaufträge zu bestimmen und daraus eine verbesserte Planung der Teilefertigung zu ermöglichen.

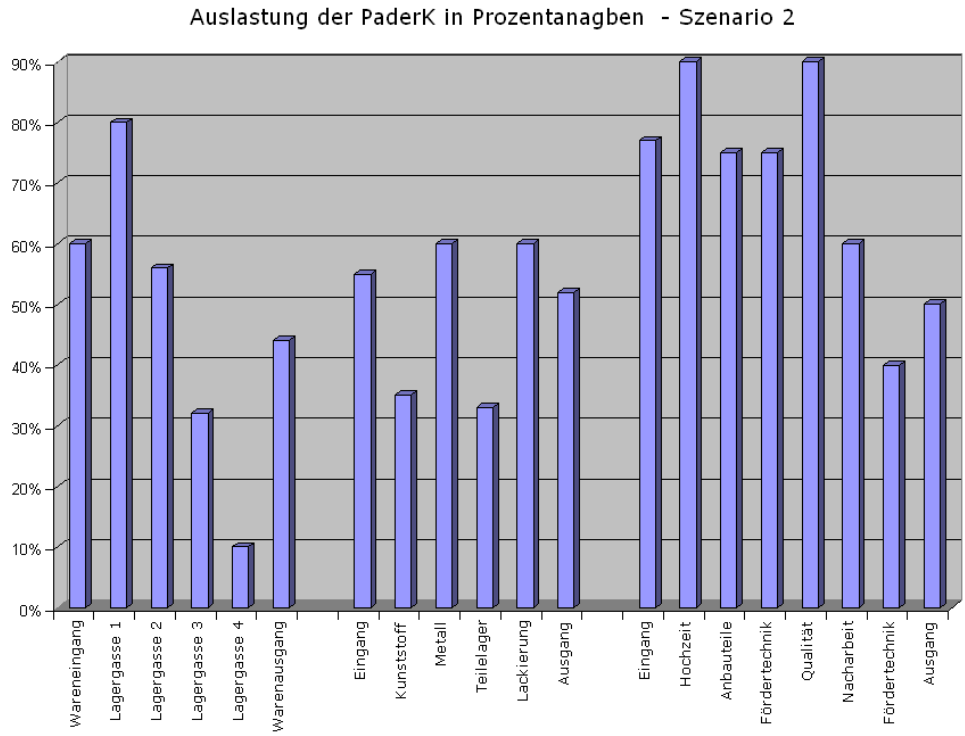


**Abbildung 121: Auslastung der Teilbereiche bei der Vorwärtssimulation**

Auch die Ergebnisdaten der Engpassanalyse können als Experiment in der Simulationsdatenbank gesichert werden und stehen für eine weitere Datenauswertung zur Verfügung (vgl. Abschnitt 6.7).

**Szenario 3: Ereignisdirekte Rückwärtssimulation zur Feinplanung der Teilefertigung**

Die Parametrierung des Experimentmanagers für das dritte Szenario gestaltet sich analog zu der oben dargestellten Vorgehensweise. Auch hier wird das Simulationsmodell auf der hohen Detaillierungsstufe betrachtet, wodurch sich eine hohe Anzahl an zufallsverteilten Bausteinparametern ergibt, die wiederum zu einer höheren Anzahl an Simulationsläufen führen. Alle wesentlichen Unterschiede zum vorwärts gerichteten Simulationsmodell sind in der Modellierungsphase berücksichtigt und in der Modellverifikation überprüft worden. Die generierten Simulationsdaten werden ebenfalls in der Simulationsdatenbank gesichert und stehen zur Datenauswertung zur Verfügung.

## 6.7 Datenauswertung

Die Aufbereitung und Interpretation der generierten Simulationsdaten durch verschiedene Methoden der Datenauswertung erfolgt als letzter Schritt einer Simulationsstudie. Das implementierte Werkzeug bietet in den verschiedenen Teilmodulen jeweils unterschiedliche Möglichkeiten, Simulationsdaten nicht nur in der Simulationsdatenbank zu sichern, sondern auch in das Dateisystem zu exportieren. Außerdem können durch die Möglichkeiten von Java auch während der Simulation aus den Modellbausteinen Simulationsdaten direkt ins Dateisystem geschrieben werden, beispielsweise in Form von kommagetrennten Wertelisten (CSV-Dateien), die in kommerziellen Tabellenkalkulationen wieder importiert und aufbereitet werden können.

Die Modellierungskomponente erlaubt aus der zweidimensionalen Visualisierung das „Abonnement" von einzelnen Parametern über den Simulationslauf und deren anschließenden Export in eine solche Excel-Schnittstelle. Darüber hinaus können einzelne Zustände der Simulation von der Oberfläche abfotografiert und als Bilddateien im Dateisystem gesichert werden. Zur Kommunikation von Modellfehlern bietet sich dieses Vorgehen insbesondere dann an, wenn die Generierung des fehlerhaften Zustandes eine lange Simulationszeit erfordert. Die Zustände können so unabhängig vom Simulationslauf gesichert und später durch die beteiligten Anwender besprochen werden.

Die Reporting-Oberfläche erlaubt neben der Speicherung des Simulationsexperimentes auch die Sicherung des auftretenden Nachrichtenverkehrs, so dass der Simulationslauf im weiteren Verlauf auch ohne erneute Simulation nachvollzogen werden kann. Alle Zwischenzustände eines Simulationslaufs können damit wieder generiert werden und erlauben eine individuelle Nachbetrachtung durch den Anwender. Die aus der Simulation entstehenden grafischen Auswertungen der einzelnen Parameter eines Modellbausteins können als einzelne Bilder auch direkt ins Dateisystem exportiert werden.

Die abgesicherten Experimentdaten können durch den Experimentmanager wieder eingelesen und dargestellt werden. Hier bietet sich die Möglichkeit, die entsprechenden Simulationsdaten, bzw. einzelne Wertereihen wieder ins Dateisystem zu exportieren. Die Generierung weiterer grafischer Auswertungen, die beispielsweise eine Nachbearbeitung der Auswertedaten benötigen, kann dadurch entweder in kommerziellen Tabellen-kalkulationen (MS Excel, OpenOffice, etc) oder durch Anwendung individueller Bibliotheken (GNUPlot, etc.) erfolgen.

Zusammenfassend kann an dieser Stelle festgehalten werden, dass alle Fragestellungen der PaderK mit dem implementierten Werkzeug beantwortet werden konnten. Durch die Abbildung der verschiedenen Fertigungsarten innerhalb der PaderK und durch die Wahl der entsprechenden Szenarien ergibt sich daraus, dass die in Abschnitt 2.4 gestellten Anforderungen an das zu entwickelnde Werkzeug mit der implementierten Lösung gelöst wurden. Anforderungen und Lösung sollen dazu nachfolgend noch einmal zusammengefasst werden und mögliche weitere Schritte aus der existierenden Lösung abgeleitet werden.

# 7 Ausblick

*„Das Bessere ist*
*der Feind des Guten"*

*(Voltaire)*

## 7.1 Zusammenfassung

Aufgabe der hier vorliegenden Arbeit war die Konzeption und Implementierung einer Modellierung und Ablaufsimulation von Fertigungssystemen hinsichtlich eines erweiterten Einsatzgebietes. Basis der Entwicklung war die Festlegung eines entsprechenden Arbeitsprozesses und einer Modellbeschreibung, auf deren Basis die eigentlichen Simulationsmodelle mit dem Werkzeug erstellt und ausgeführt werden können.

Inhaltlicher Schwerpunkt war die Erfüllung folgender neuer Szenarien zum Einsatz der Ablaufsimulation:

***Synchronisierte, ortsunabhängige Mehrbenutzerunterstützung bei der Modellierung und Simulation von Materialflussmodellen in einer interaktiven, immersiven und virtuellen Umgebung***

Der steigenden Komplexität von Planungsprojekten im Bereich der Ablaufsimulation sollte dadurch Rechnung getragen werden, dass in dem entwickelten Werkzeug mehrere Anwender in einer gemeinsamen Umgebung Simulationsmodelle erstellen und ausführen können. Insbesondere die Kommunikation mit Nicht-Simulationsexperten innerhalb des Planungsteams erfordert dazu eine möglichst immersive Darstellung, die das Verhalten des Simulationsmodells bestmöglich darstellt und erklärt. Zur Visualisierung der Modelle und deren dynamischen Verhaltensweisen dient eine virtuelle Umgebung, in der das Simulationsmodell dreidimensional dargestellt wird. Der Anwender selbst ist mehr als ein passiver Betrachter, sondern nimmt auf den Fortlauf eines Simulationslaufes interaktiv Einfluss. Die Qualität der Gesamtplanung soll dadurch verbessert werden, dass die entsprechende Modellierung der Fertigungssysteme layoutgerecht ausgeführt wird.

***Planung, Evaluierung und fortlaufende Verbesserung der Fertigungsprozesse über alle Planungs- und Ausführungsphasen bis zur Rückkopplung in die Fertigungslenkung***

Der Einsatz der Ablaufsimulation soll über alle Planungs- und operativen Phasen eines Fertigungsprozesses hinweg erfolgen. Neben Machbarkeitsstudien, Variantenplanungen oder quantitativen Fragestellungen dient die Simulation auch der Planung von Fertigungsprogrammen. Ein bestehendes Simulationsmodell kann über alle Phasen der Strukturplanung, Mengen-, Kapazitäts- und Programmplanung bis hin zur Prognose und der laufenden Verbesserung vorhandener Fertigungsprozesse eingesetzt werden. Durch eine richtungsoffene Simulation können auch kundenorientierte Fertigungsprozesse abgebildet werden. Darüber hinaus wird eine zeitorientierte Ausführung in Vor- oder Rückwärtsrichtung unterstützt, um die Integration in bestehende Leitstandsysteme zu erleichtern.

***Kooperative Planung innerhalb von Unternehmen, Unternehmensverbünden oder Supply-Chain-Netzwerken***
Großunternehmen, virtuelle Unternehmen oder Supply-Chain-Netzwerke fertigen an unterschiedlichen Standorten. Der Lieferfähigkeit kommt innerhalb einer Supply-Chain steigende Bedeutung zu. Mit dem implementierten Werkzeug können die Planungen der Supply-Chain Partner enger aufeinander abgestimmt und überwacht werden. Die Mehrbenutzerfähigkeit der angestrebten Ablaufsimulation ermöglicht eine kooperative Planung mehrerer Simulationsexperten an einem gemeinsamen, dynamisch detaillierenden Simulationsmodell des Fertigungsnetzwerks unabhängig vom Standort der jeweiligen Experten. Da die verschiedenen Partner innerhalb solcher unternehmens-internen wie –externen Fertigungsnetzwerke nach unterschiedlichen Fertigungs-ablaufarten produzieren können, wurde mit der automatischen Wegberechnung ein spezielles Verfahren entwickelt, welches die Modellierung und Simulation von funktional gegliederten Fertigungssystemen bzw. deren Mischformen erlaubt.

## 7.2 Grenzen der Arbeit

Diese Arbeit beschränkt sich auf den erweiterten Einsatz der Methode Ablaufsimulation und versucht, Schnittstellen zu angrenzenden Planungsschritten möglichst konstruktiv und praxistauglich zu gestalten. In der Praxis der Unternehmenslandschaften existieren zahlreiche Bestrebungen hin zu der Vision einer „Digitalen Fabrik". Dieser Ansatz ist von seiner Grundidee sicherlich ganzheitlicher, wird aber von den jeweiligen Unternehmen oftmals nur hinsichtlich der jeweils eingesetzten Technologien verfolgt. Ziel ist ebenso wie bei der hier vorliegenden Arbeit eine ganzheitliche Gestaltung der Produkt- und Prozessplanung einer Unternehmung. Die wesentliche Herausforderung in der Praxis ist hierbei jedoch zumeist die Integration der existierenden Datenformate und fokussiert nicht so sehr auf eine ganzheitliche Betrachtung der Methoden und Ziele. Insofern passen die hier genannten Bestrebungen in den Kontext der Digitalen Fabrik; die Digitale Fabrik fördert aber noch mehr auch die Integration von Produkt- und Prozessplanung, als das in dieser Arbeit der Fall ist.

Die Summe der Einzelziele zur Erweiterung des Einsatzgebietes der Ablaufsimulation erhebt keinen Anspruch auf Vollständigkeit, sondern will vielmehr mögliche nächste Schritte aufzeigen, die vor dem Hintergrund bereits heute existierender Projekte auftreten. Dennoch scheint eine engere Anbindung von Fertigungsplanung und –Lenkung, die Kopplung der Methode Ablaufsimulation mit den Planungsszenarien einer PPS-Steuerung als sinnvoll und wurde in Ansätzen heute bereits realisiert.

Des Weiteren ergeben sich auch aus den Ergebnissen der hier vorliegenden Arbeit weitere Erkenntnisse für mögliche, zukünftige Schritte, die den Themenbereich der Ablaufsimulation vorantreiben können, ggf. eingebettet in den Kontext der Digitalen Fabrik. Der nachfolgende Absatz will einige Ideen hierzu liefern.

## 7.3 Ausblick

Die ersten Arbeiten mit dem entwickelten Werkzeug zeigen erfolgreiche Durchführungen und Ergebnisse hinsichtlich einer Kopplung der Methode Ablaufsimulation mit den Planungs- und Steuerungsalgorithmen der Fertigungslenkung. Die Arbeit könnte an dieser Stelle vorangetrieben werden, indem weitere Lösungsmöglichkeiten in diesem

Problembereich einer möglichst optimalen Einsteuerung der Fertigungsaufträge in das Fertigungssystem, beispielsweise durch Ankopplung von Optimierungsalgorithmen, Neuronalen Netzen oder Genetischen Algorithmen entwickelt werden. Erste Aussagen über das Verhalten eines dynamischen Fertigungssystems könnten schneller generiert werden und die eigentlichen Simulationsläufe müssten erst später und mit optimierten Parametern gestartet werden.

Ein weiteres sinnvolles Arbeitsgebiet ist eine erweiterte Unterstützung der Anwender bei der Modellierung von Simulationsmodellen. Vielfach sind einzelne Maschinendaten, Fertigungsabläufe, etc. schon in den entsprechenden ERP-Systemen vorhanden. Der Anwender könnte durch spezielle Methoden zur (semi-)automatischen Modellgenerierung auf Basis dieser ERP-Daten bei der Erstellung der Simulationsmodelle deutlich besser unterstützt werden. Insbesondere der Prozess der Modellgenerierung zu einer Ausgangslösung, die durch erste Simulationsläufe validiert werden kann, würde dadurch erheblich beschleunigt. Ausschließlich die speziellen Steuerstrategien und Soll-Konzepte müssten manuell in das Simulationsmodell nachgepflegt werden.

Die Visualisierung einzelner Simulationsläufe ist immer nur eine mögliche Ausprägung des dynamischen Verhaltens des modellierten Fertigungssystems und basiert in großen Teilen auch auf den entsprechenden Starparametern der stochastisch verteilten Modellvariablen. Inwiefern der entsprechende Simulationslauf und seine Visualisierung also typisch für das modellierte System sind, kann zunächst nicht vom Anwender beantwortet werden. Klassischerweise wird das Problem in der Experimentierphase dadurch umgangen, dass mehrere Simulationsläufe mit unterschiedlichen Startwerten der stochastisch verteilten Modellvariablen simuliert werden. Um dieses Problem bereits während der Modellierungsphase zu bewältigen, könnte man den Versuch aus den entsprechenden Simulationsläufen parallel simulieren und in einer Visualisierungskomponente darstellen. Für jeden einzelnen Parameter einer Bausteininstanz würde in dieser Visualisierungsform eine Streubreite angezeigt, die den Anwender einschätzen lässt, wie typisch die angezeigte Visualisierung für das dynamische Verhalten des Systems ist. Wird die Visualisierung um eine Möglichkeit angereichert, zwischen der Visualisierung der einzelnen Simulationsläufe zu wechseln, könnte der Anwender gleichzeitig auch zu typischeren oder extremeren Simulationsläufen wechseln. Darüber hinaus könnten in diesem Kontext Visualisierungsmöglichkeiten konzipiert werden, die dem Anwender mehrere Simulationen in einer Oberfläche darstellen und analysieren lassen.

# Quellenverzeichnis

[AlIs77] Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S.: A pattern language, Oxford University Press, New York, 1977

[Arno95] Arnold, D.: Materialflusslehre. Vierweg Verlag, Braunschweig, Wiesbaden, 1995.

[Balz05] Balzert, H.: Lehrbuch Grundlagen der Informatik, Spektrum Akademischer Verlag; Auflage: 2., Aufl.,2005

[Balz82] Balzert, H.: Die Entwicklung von Software-Systemen: Prinzipien, Methoden, Sprachen Werkzeuge, Bibliographisches Institut, 1982

[Baum03] Baumgärtner, T.: Wenn die Computer die Fabrik von morgen testen, Industrieanzeiger 51-52, 2003

[BeBr03] Bender, M., Brill, M.: Computergrafik – Ein anwendungsorientiertes Lehrbuch, Hanser Verlag, München [u.a.], 2003

[Bega94] Begault, Durand R.: 3-D sound for virtual reality and multimedia, Academic Press, Boston [u.a.], 1994

[Berg02] Bergbauer, J.: Entwicklung eines Systems zur interaktiven Simulation von Produktionssystemen in einer Virtuellen Umgebung, Shaker Verlag, Aachen, 2002

[Birt82] Birtwistle, G. M., Luker, P.: Discrete event simulation with Demos, in: Proceedings of the 14th conference on Winter Simulation, San Diego, California, 1982

[BKLP01] Bowman, D.; Kruijff, E.; LaViola, J. J. Jr. und Poupyrev, I.: An Introduction to 3-D User Interface Design, In: Durlach, N. I. und Slate, M. (Hrsg.), Presence, Volume 10, Number 1, S. 96-108, MIT-Press, 2001

[Booc94] Booch, G.: Objekorintierte Analyse und Design: Mit praktischen Anwendungsbeispielen, Addison-Westley, 1994

[Borm94] Bormann, S.: Virtuelle Realität – Genese und Evaluation, Addison Westley Publishing Company, Bonn [u.a.], 1994

[Boss92] Bossel, Hartmut: Modellbildung und Simulation, Vieweg-Verlag, 1992

[Brac02] Bracht, U.: Ansätze und Methoden der Digitalen Fabrik, In: Schulze, T.; Schlechtweg, S. und Hinz, V. (Hrsg.): Simulation und Visualisierung 2002, S. 1-11, SCS-Europe BVBA, Gent, Belgien, 2002

[BrDu04] Brügge, B., Dutoit, A.H.: Objektorientierte Softwaretechnik – mit UML, Entwurfsmustern und Java, Pearson Education Deutschland, 2004

[Brey05] Breymann, U.: C++ , Hanser Fachbuchverlag; Auflage: 8., Aufl., 2005

[BrFa01] Bracht, U. / Fahlbusch, M. W.: Einsatz von Virtual Reality – Systemen in der Fabrik- und Anlagenplanung, Zeitschrift für wirtschaftliche Fachbetriebe (ZWF), 2001

[BuMe96] BUschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-oriented Software Architecture – A System of Patterns, John Wiley & Sons, Chichester [u.a.], 1996

[Buur05] Buurman, Gerhard M. (Hrsg.): Total Interaction. Theory and practice of a new paradigm for the design disciplines. Birkhäuser. Basel, Wien, New York, 2005

[Chis92] Chisman, J.A.: Introduction to Simulation Modeling Using GPSS/PC, Prentice Hall, Englewood Cliffs, N.J., 1992

[CoDo02] Coulouris G. F., Dollimore J., Kindeberg T.: Verteilte Systeme: Konzepte und Design; München u. a. 2002; Pearson Studium

[Corm90] Cormen, Thomas H.: Introduction to algorithms, Cambridge, MIT Press, 1990

[Dang03] Dangelmaier W.: Skript zur Vorlesung: Grundlagen der Informationstechnik von Produktions- und Logistiksystemen; Universität Paderborn 2003

[Dang99] Dangelmaier, W.: Fertigungsplanung: Planung von Aufbau und Ablauf der Fertigung, Springer, Berlin, 1999

[DaWa97] Dangelmaier W., Warnecke H.: Fertigungslenkung: Planung und Steuerung des Ablaufs der diskreten Fertigung; Berlin u. a. 1997; Springer

[Delm-ol] Delmia Quest, http://www.delmia.com/gallery/pdf/DELMIA_QUEST.pdf (zuletzt abgefragt: November 2006

[DFAB98] Dix, A.; Finlay, J.; Abowd, G. und Beale, R.: Human-Computer Interaction, 2. Auflage, Prentice Hall Europe, London u.a., 1998

[DMD03] Dangelmaier W. (Hrsg.), Dittmann N., Mueck B.: Marktanalyse: Materialfluss–Simulatoren; Paderborn 2003; ALB-HNI-Verlagsschriftenreihe

[EDFa-ol] ED-Falcon, Incontrol Enterprise Dynamics, http://www.taylorii.com (zuletzt abgefragt: November 2006)

[Evan88] Evans, J. B.: Structures of discrete event simulation: an introduction to engagement strategy, Ellis Horwood Limited, Chichester, England, 1988

[FaFG94] Fandel, G., Francois, P., Gubitz, K.-M.: PPS-Systeme, Grundlagen, Methoden, Software, Martkanalysen, Springer Verlag, Berlin [u.a.], 1994

[FaHa94] Faßler, M., Halbach, W. (Hrsg.): Cyberspace. Gemeinschaften, Virtuelle Kolonien, Öffentlichkeiten. Fink Verlag, München, 1994

[FiHe00] Fischer, J., Herold, W., Dangelmaier, W., Nastansky, L., Suhl, L.: Bausteine der Wirtschaftsinformatik, 2. überarbeitete und erweiterte Auflage, Erich Schmidt Verlag, Berlin, 2000

[GaHe01] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Entwurfsmuster– Elemente wiederverwendbarer objektorientierter Software, Addison-Wesley, München [u.a.], 2001

[GiVa03] Girault, C. und Valk, R.: Petri Nets for Systems Engineering – A Guide to Modeling, Verification, and Applications, Springer Verlag, Berlin u.a., 2003

[GrBo04] Graupner, T.; Bornhäuser, M.; Sihn, W.: Backward Simulation In Food Industry For Facility Planning And Daily Scheduling; Proceedings 16th European Simulation Symposium, 2004

[Hack89] Hackstein, R.: Produktionsplanung und –Steuerung, 2.Auflage, VDI-Verlag, Düsseldorf, 1989

[Henk97] Henkel, S.: Ein System von Software-Entwurfsmustern für die Propagation von Ereignissen in Werkzeugen zur kooperativen Fabrikmodellierung, HNI-Verlagsschriftenreihe, Paderborn, 1997

[Holm-ol] Holmevik, J.R.: Compiling Simula, http://heim.ifi.uio.no/~cim/sim_history.html (zuletzt abgefragt November 2006)

[JaCh97] Jain, S., Chan, S.: Experiences with Backward Simulation Based Approach for Lot Release Planning, Winter Simulation Conference, 773-780, 1997

[Joha91] Johansen, R., " Teams for tomorrow". In Proc. 24th IEEE Hawaii Intl Conf. On System Science, S. 520-534. IEEE Comp. Soc. Press, Los Alamitos, 1991 ASIM

[Kech05] Kecher,Christian: UML2.0, GallileoComputing, 2005

[Kern79] Kern, W.: Handwörterbuch der Produktionswirtschaft. Sp. 1481, Poeschel, Stuttgart, 1979

[KlBu71] Klaus, G. und Buhr, M.: Philosophisches Wörterbuch, VEB Verlag Enzyklopädie Leipzig, 8. Auflage, 1971

[KlKr02] Klein, J.; Krokowski, J.; Fischer, M.; Wand, M.; Wanka, R.; Meyer auf der Heide, F.: The Randomized Sample Tree: A Data Structure for Interactive Walkthroughs in Externally Stored Virtual Environments; In: ACM Symposium on Virtual Reality Software and Technology (VRST '2002); 2002; S. 137 – 146

[Klos06] Klose, M.: Betreibersimulation Werk Leipzig - Ein webbasiertes und online gekoppeltes Prognosetool zur Unterstützung der Produktionssteuerung, In: Dangelmaier, Wilhelm; Laroque, Christoph; Döring, Andre (Hrsg.) Die Supply-Chain von morgen – Lieferfähigkeit im globalen Unternehmen, ALB-HNI-Verlagsschriftenreihe, Band 14, Paderborn, 2006

[LaKe00] Law A. M., Kelton W. D.: Simulation Modeling and Analysis; Boston 2000; Third Edition; McGRAW-Hill International Series

[Lani91] Lanier, J.: Was heißt „virtuelle Realität", In: Cyberspace. Ausflüge in virtuelle Wirklichkeiten, Rowohlt Verlag, Reinbeck, 1991

[LaRa06] Lahres, B., Raýman, G.: *Praxisbuch Objektorientierung*, Galileo Computing, 2006

[LeEg88] Leszak, M. und Eggert, H.: Petri-Netz-Methoden und –Werkzeuge – Hilfsmittel zur Entwurfsspezifikation und –validation von Rechensystemen, Springer-Verlag, Berlin u.a., 1988

[Lock93] Lockemann, P.C., Krüger, G., Krumm, H.: Telekommunikation und Datenhaltung, Hanser, 1993

[Muec05] Mueck, B.: Eine Methode zur benutzerstimulierten detaillierungsvarianten Berechnung von diskreten Simulationen von Materialflüssen, HNI-Verlagsschriftenreihe, Paderborn, 2005

[OhHa04] Ohkawa, T.; Hata, S.; Komoda, N.: Backward Qualitative Simulation Of Structural Model; Japan, 1996

[OOSE] OOSE – innovative Informatik, http://www.oose.de (zuletzt abgefragt: November 2006)

[PaKr05] Page, B., Kreutzer, W.: The Java Simulation Handbook – Simulating Discrete Event Systems, Shaker Verlag, Aachen, 2005

[RDBa01] Rolland, J. P.; Davis, L. D.; Baillot, Y.: A Survey of Tracking Technologies for Virtual Environments. In: In: Barfield, W.; Caudell, T. (Hrsg.): Fundamentals of wearable Computers and Augmented Reality. Lawrence Erlbaum Associates, Publishers, Mahwah, London, 2001. TRACKING

[Rock00] Rockwell, W.: XML, XSLT, Java und JSP – Professionelle Web-Applikationen entwickeln, Galileo Press GmbH, Bonn, 2000

[Rose92] Rosenberg, O.: Potentialfaktorwirtschaft. Vorlesungsumdruck. UNI-GH Paderborn, 1992

[Rose93] Rosemann, M.: Design of a Real-Time Groupware-Toolkit, University of Calgary, 1993

[Schm92] Schmidt, C.: Petri-Netze: Ein Instrument zur Lösung logistischer Probleme im CIM-Bereich, In: Wirtschaftsinformatik, 34, S. 66-75, 1992

[Schn96] Schneider, U.: Ein formales Modell und eine Klassifikation für die Fertigungssteuerung, Band 16, HNI-Verlagsschriftenreihe, Paderborn, 1996

[ScWe00] Schumacher, R. und Wenzel, S.: Der Modellbildungsprozeß in der Simulation, In: Wenzel, S. (Hrsg.): Referenzmodelle für die Simulation in Produktion und Logistik, S.5-11, SCS-Europe BVBA, Gent, Belgien, 2000

[Sims-ol] CACI Process Company
http://www.simprocess.com (zuletzt abgefragt: November 2006)

[Simu03] Programminternes Handbuch: Simul8 Manual and Simulation Guide, Simul8 Corporation, 2003

[Somm92] Sommerville, I.: Software Engineering, Addison-Westley, 1992

[Star90] Starke, P. H.: Analyse von Petri-Netz-Modellen, Teubner, Stuttgart, 1990

[Stor00] Storck, A.: Effiziente 3D-Interaktions und Visualisierungstechniken für benutzerzentrierte Modellierungssysteme, Dissertation, Technische Universität Darmstadt, 2000

[StVö05] Stahl,T. & Völter,M.:Modellgetriebene Softwareentwicklung, dpunkt,1. Auflage,2005.

[SuMe01] Suhl, L., Mellouli, T.: Optimierungssysteme, Skript zur Vorlesung, uni Paderborn, 2001

[Tane03] Tanenbaum A. S.: Computerarchitektur: Strukturen, Konzepte, Grundlagen; München 2003; Pearson Studium

[TaSt03] Tanenbaum, A. S., Steen M.: Verteilte Systeme: Grundlagen und Paradigmen; München u. a. 2003; Pearson Studium

[Teic98] Teich, T.: Optimierung von Maschienenbelegungsplänen unter Benutzung heuristischer Verfahren, Josef Eul Verlag, Lohmar, 1998

[UGS-ol] UGS TEcnomatix Plant Simulation,
http://em-plant.de (zuletzt abgefragt: November 2006)

[Ulle05] Ullenboom, C.: Java ist auch eine Insel. Programmieren mit der Java Standard Edition Version 5, Galileo Press; Auflage: 5., 2005

[VDI3300] VDI-Richtlinie 3300: Materialfluß-Untersuchungen, Beuth Verlag, Berlin,

[VDI3633] VDI-Richtlinie 3633: Simulation von Logistik-, Materialfluss- und Produktionssystemen - Grundlagen; Düsseldorf 1993; VDI Verlag Düsseldorf

[VDI4499] VDI-Richtlinie 4499: Digitale Fabrik - Grundlagen; Düsseldorf 2006; VDI Verlag Düsseldorf

[Völt-ol] Völter,Markus: ModellgetriebeneSoftwareentwicklung, 2004,
http://www.voelter.de (zuletzt abgefragt: November 2006)

[W3C] World Wide Web Consortium,
http://w3c.org (zuletzt abgefragt: November 2006)

[WaFi01] Wand, M.; Fischer, M.; Peter, I.; Meyer auf der Heide, F.; Strasser, W.: *The Randomized z-Buffer Algorithm: Interactive Rendering of Highly Complex Scenes*; In: Computer Graphics (SIGGRAPH 01 Conference Proceedings); 2001; S. 361 – 370

[WaMe97] Watson, E. F., Medeiros, D. J., Sadowski, R. P.: A simulation-based backward planning approach for order-release, Proceedings of the 29th Conference on Winter Simulation, Atlanta, Georgia, ACM Press, New York, NY, 765-772. 1997

[Woeh90] Wöhe, G.: Einführung in die Allgemein Betriebwirtschaftsaftslehre, Vahlen, 17. überarb. Auflage, 1990

[YiCl94] Ying, C. C., Clark, G. M.: Order release planning in a job shop using a bi-directional simulation algorithm, Proceedings of the 26th Conference on Winter Simulation, Orlando, Florida, Society for Computer Simulation International, 1008-1012, 1994
1973

# Anhang A - DTD zur Modellbeschreibung

```
<?xml version="1.0" encoding="iso-8859-1"?>

<!--
Fixe DTD fuer Modellbeschreibung
Koordinatensystem: Linkshaendig, Daumen: x, Zeige: y, Mittel: z, z auf den Betrachter zu
-->

<!ELEMENT model (library, main, database, comment?)>

<!ELEMENT database EMPTY>
<!ATTLIST database
        dburl CDATA #REQUIRED
        username CDATA #REQUIRED
        password CDATA #REQUIRED
>

<!ELEMENT library (buildingblock*, comment?)>

<!ELEMENT buildingblock ( dockingpoints*, tokenpath*, subblock*, ichannel*, ochannel*, variable*,
   event*, link*, moredetailed?, lessdetailed?, comment?, calendarRowShifts*, calendarRowFreeDays*,
breakdown*)>
<!ATTLIST buildingblock
        id CDATA #REQUIRED
        name CDATA #REQUIRED
        id_mesh CDATA #IMPLIED
        meshscale CDATA "1"
        significance CDATA "0"
        x-size CDATA #IMPLIED
        y-size CDATA #IMPLIED
        z-size CDATA #IMPLIED
        color CDATA #IMPLIED
        simulationBoundary (true|false) "false"
        sortOfModel CDATA "normal"
        calendar CDATA #IMPLIED
        shiftTableActive (true|false) "true"
        freeDaysTableActive (true|false) "true"
        simStartTime CDATA #IMPLIED
        simEndTime CDATA #IMPLIED
>

<!ELEMENT calendarRowShifts EMPTY>
<!ATTLIST calendarRowShifts
        name CDATA #REQUIRED
        type CDATA "regular"
        from CDATA #REQUIRED
        to CDATA #REQUIRED
        breaks CDATA #IMPLIED
        mo (true|false) "false"
        di (true|false) "false"
        mi (true|false) "false"
        do (true|false) "false"
        fr (true|false) "false"
        sa (true|false) "false"
        so (true|false) "false"
        dates CDATA #IMPLIED
>
```

```
<!ELEMENT calendarRowFreeDays EMPTY>
<!ATTLIST calendarRowFreeDays
        date CDATA #REQUIRED
        reason CDATA #IMPLIED
        reduceTo CDATA #IMPLIED
>

<!ELEMENT breakdown ( mttfdist?, mttrdist?, firstTTFdist?, breakdownstart_event?, breakdownend_event?)>
<!ATTLIST breakdown
        name CDATA #REQUIRED
        hasMTTFandMTTR (true|false) "false"
        availability CDATA #IMPLIED
        mttrAvailability CDATA #IMPLIED
        generateStartEvents (true|false) "true"
        generateEndEvents (true|false) "true"
        firstTTFAutomatically (true|false) "true"
>

<!ELEMENT firstTTFdist EMPTY>
<!ATTLIST firstTTFdist
        type CDATA #REQUIRED
        stream CDATA #IMPLIED
        lowerbound CDATA #IMPLIED
        upperbound CDATA #IMPLIED
        haslowerbound CDATA #IMPLIED
        hasupperbound CDATA #IMPLIED
        alpha CDATA #IMPLIED
        beta CDATA #IMPLIED
        mean CDATA #IMPLIED
        variance CDATA #IMPLIED
        leftpoint CDATA #IMPLIED
        midpoint CDATA #IMPLIED
        rightpoint CDATA #IMPLIED
        constvalue CDATA #IMPLIED
>

<!ELEMENT mttfdist EMPTY>
<!ATTLIST mttfdist
        type CDATA #REQUIRED
        stream CDATA #IMPLIED
        lowerbound CDATA #IMPLIED
        upperbound CDATA #IMPLIED
        haslowerbound CDATA #IMPLIED
        hasupperbound CDATA #IMPLIED
        alpha CDATA #IMPLIED
        beta CDATA #IMPLIED
        mean CDATA #IMPLIED
        variance CDATA #IMPLIED
        leftpoint CDATA #IMPLIED
        midpoint CDATA #IMPLIED
        rightpoint CDATA #IMPLIED
        constvalue CDATA #IMPLIED
>

<!ELEMENT mttrdist EMPTY>
<!ATTLIST mttrdist
        type CDATA #REQUIRED
        stream CDATA #IMPLIED
        lowerbound CDATA #IMPLIED
        upperbound CDATA #IMPLIED
```

```
        haslowerbound CDATA #IMPLIED
        hasupperbound CDATA #IMPLIED
        alpha CDATA #IMPLIED
        beta CDATA #IMPLIED
        mean CDATA #IMPLIED
        variance CDATA #IMPLIED
        leftpoint CDATA #IMPLIED
        midpoint CDATA #IMPLIED
        rightpoint CDATA #IMPLIED
        constvalue CDATA #IMPLIED
>

<!ELEMENT breakdownstart_event ( codeBS)>

<!ELEMENT codeBS (#PCDATA)>

<!ELEMENT breakdownend_event ( codeBE)>

<!ELEMENT codeBE (#PCDATA)>

<!ELEMENT tokenpath EMPTY>
<!ATTLIST tokenpath
        iddatabase      CDATA #REQUIRED
        name CDATA #IMPLIED
>

<!ELEMENT dockingpoints (dockingpointfromdb)+>

<!ELEMENT dockingpointfromdb EMPTY>
<!ATTLIST dockingpointfromdb
        id CDATA #REQUIRED
        name CDATA #IMPLIED>

<!ELEMENT subblock (dockingpoints*, tokenpath*, position, relativePosition?, variable_value*,
calendarRowShifts*, calendarRowFreeDays*, breakdown*, subblock*, comment?)>
<!ATTLIST subblock
        type CDATA #REQUIRED
        name CDATA #REQUIRED
        meshScale CDATA #IMPLIED
        id_mesh CDATA #IMPLIED
        srcid CDATA #REQUIRED
        color CDATA #IMPLIED
        layer CDATA "0"
        significance CDATA "0"
        sortOfModel CDATA "normal"
        calendar CDATA #IMPLIED
        shiftTableActive (true|false) "true"
        freeDaysTableActive (true|false) "true"
>

<!ELEMENT position EMPTY>
<!ATTLIST position
        x CDATA #REQUIRED
        y CDATA #REQUIRED
   z CDATA "0"
   rotx CDATA "0"
   roty CDATA "0"
   rotz CDATA "0">

<!ELEMENT relativePosition EMPTY>
```

```
<!ATTLIST relativePosition
        x CDATA #REQUIRED
        y CDATA #REQUIRED>

<!ELEMENT variable_value EMPTY>
<!ATTLIST variable_value
        name CDATA #REQUIRED
        value CDATA #IMPLIED
        meanvalue CDATA #IMPLIED
        variance CDATA #IMPLIED
        lowerbound CDATA #IMPLIED
        upperbound CDATA #IMPLIED
        streamNumber CDATA #IMPLIED
>

<!ELEMENT ichannel (position?, comment?)>
<!ATTLIST ichannel
        name CDATA #REQUIRED
>

<!ELEMENT ochannel (position?, comment?)>
<!ATTLIST ochannel
        name CDATA #REQUIRED
>

<!ELEMENT link (comment?)>
<!ATTLIST link
        from CDATA #REQUIRED
        from_channel CDATA #REQUIRED
        to CDATA #REQUIRED
        to_channel CDATA #REQUIRED
>

<!ELEMENT moredetailed (comment?)>
<!ATTLIST moredetailed   type CDATA #REQUIRED>

<!ELEMENT lessdetailed (comment?)>
<!ATTLIST lessdetailed type CDATA #REQUIRED>

<!ELEMENT variable ((int | long | double | float | time | arraylist | string | boolean | enum | token | hashmap |
table | random ), comment?)>
<!ATTLIST variable
        public (yes | no) #REQUIRED
        vrvisibility (none | readable | writeable) #REQUIRED
        name CDATA #REQUIRED
>

<!ELEMENT long (standardvalue, lowerbound?, upperbound?, evaluationtypeno?)>

<!ELEMENT int (standardvalue, lowerbound?, upperbound?, evaluationtypeno?)>

<!ELEMENT lowerbound (#PCDATA)>

<!ELEMENT upperbound (#PCDATA)>

<!ELEMENT standardvalue (#PCDATA)>

<!ELEMENT double (standardvalue, lowerbound?, upperbound?, precision?, evaluationtypeno?)>

<!ELEMENT float (standardvalue, lowerbound?, upperbound?, precision?, evaluationtypeno?)>
```

```
<!ELEMENT precision (#PCDATA)>

<!ELEMENT arraylist (variable*, evaluationtypelist?)>

<!ELEMENT string (standardvalue, length, evaluationtypestring?)>

<!ELEMENT length (#PCDATA)>

<!ELEMENT boolean (standardvalue, evaluationtypebool?)>

<!ELEMENT time (#PCDATA)>

<!ELEMENT enum (standardvalue, enummember+)>

<!ELEMENT enummember (#PCDATA)>
<!ATTLIST enummember
        description CDATA #IMPLIED
>

<!ELEMENT token EMPTY>

<!ELEMENT hashmap EMPTY>

<!ELEMENT table (evaluationtypetable?)>
<!ATTLIST table
        columns CDATA #REQUIRED
>

<!ELEMENT random ((uniform | normal | triangular | exponential), streamNumber, evaluationtypeno?)>

<!ELEMENT uniform (lowerbound, upperbound)>

<!ELEMENT normal (meanvalue, variance)>

<!ELEMENT triangular (meanvalue, lowerbound, upperbound)>

<!ELEMENT exponential (meanvalue)>

<!ELEMENT meanvalue (#PCDATA) >

<!ELEMENT variance (#PCDATA) >

<!ELEMENT streamNumber (#PCDATA)>

<!ELEMENT evaluationtypeno (trafficlight | gauge | display)>
<!ATTLIST evaluationtypeno
        evaluationprio (high | medium | low)  #REQUIRED
>

<!ELEMENT evaluationtypebool (signal | display)>
<!ATTLIST evaluationtypebool
        evaluationprio (high | medium | low)  #REQUIRED
>

<!ELEMENT evaluationtypelist (meanvariance | histogram | display)>
<!ATTLIST evaluationtypelist
        evaluationprio (high | medium | low)  #REQUIRED
>
```

```
<!ELEMENT evaluationtypestring (display)>
<!ATTLIST evaluationtypestring
        evaluationprio (high | medium | low)  #REQUIRED
>

<!ELEMENT evaluationtypetable (meanvariance | timetable | histogram | display)>
<!ATTLIST evaluationtypetable
        evaluationprio (high | medium | low)  #REQUIRED
>

<!ELEMENT meanvariance (valuecolumn?)>

<!ELEMENT valuecolumn (#PCDATA)>

<!ELEMENT timetable (yline*, timecolumn, valuecolumn)>
<!ATTLIST timetable
        dots (yes | no) #REQUIRED
        color CDATA #IMPLIED
>

<!ELEMENT yline (#PCDATA)>

<!ELEMENT timecolumn (#PCDATA)>

<!ELEMENT display (#PCDATA)>

<!ELEMENT trafficlight (thresholdyellow, thresholdred)>

<!ELEMENT gauge (thresholdyellow?, thresholdred?)>

<!ELEMENT thresholdyellow (#PCDATA)>

<!ELEMENT thresholdred (#PCDATA)>

<!ELEMENT threshold (#PCDATA)>

<!ELEMENT histogram (threshold*, valuecolumn?)>

<!ELEMENT signal (greentrue | greenfalse)>

<!ELEMENT greentrue (#PCDATA)>

<!ELEMENT greenfalse (#PCDATA)>

<!ELEMENT event
((input_event|output_event|reopen_event|sub_output_event|sub_input_event|sub_reopen_event|init_event|u
ser_defined_event|switch_event|final_event), code, comment?)>
<!ATTLIST event
        name CDATA #IMPLIED
>

<!ELEMENT input_event EMPTY>
<!ATTLIST input_event
        inchannel CDATA #REQUIRED
>

<!ELEMENT output_event EMPTY>
<!ATTLIST output_event
        outchannel CDATA #REQUIRED
>
```

```
<!ELEMENT reopen_event EMPTY>
<!ATTLIST reopen_event
        channel CDATA #REQUIRED
>

<!ELEMENT sub_output_event EMPTY>
<!ATTLIST subout_event
        outchannel CDATA #REQUIRED
>

<!ELEMENT sub_input_event EMPTY>
<!ATTLIST sub_input_event
        inchannel CDATA #REQUIRED
>

<!ELEMENT sub_reopen_event EMPTY>
<!ATTLIST sub_reopen_event
        inchannel CDATA #REQUIRED
>

<!ELEMENT init_event EMPTY>

<!ELEMENT user_defined_event EMPTY>
<!ATTLIST user_defined_event name CDATA #REQUIRED>

<!ELEMENT switch_event EMPTY>
<!ATTLIST switch_event name CDATA #REQUIRED>

<!ELEMENT final_event EMPTY>

<!ELEMENT code (#PCDATA)>

<!ELEMENT comment (#PCDATA)>

<!ELEMENT main EMPTY>
<!ATTLIST main
        modeltorun CDATA #IMPLIED
>
```

# Anhang B - DTD zum Nachrichtenaustausch

```
<!--
        ============================================================
   DTD for XML-communication between simulation and visualisation
        ============================================================

        $Id: messages.dtd,v 1.44 2006/02/20 16:03:00 shampoo Exp $
-->

<!--
        ========
        Entities
        ========
-->

<!ENTITY % ID "CDATA">
<!ENTITY % Int "CDATA">
<!ENTITY % Long "CDATA">
<!ENTITY % Float "CDATA">
<!ENTITY % Double "CDATA">
<!ENTITY % String "CDATA">
<!ENTITY % Boolean "CDATA">

<!ENTITY % Timestamp "
        timestamp %Long; #REQUIRED
">

<!ENTITY % Message1 "(normal-message | request-message | reply-message)">
<!ENTITY % Message2 "(start | pause | stop| save | timefactor | databaseinfo |
                              dockingpoint | tokenpath | buildingblock | robot | token |
                              robotpool | subscribe-properties | unsubscribe-properties |
                              object-properties | properties-changed | significance |
                              include-token | exclude-token | animate-token | remove-token |
                              remove-robot | move-robot | move-poolmember | check-arrival |
                              free-robot |endOfInitialization | mp-prefs | timestamp
                              | avatar | mrm-quantifier | unlockMRM | lockMRM | maxspeed | normalspeed)">
<!ENTITY % Message3 "(error | no-error | expected-arrival-time | arrival | still-computing-path | object-
properties)">

<!--
   ====
        ROOT
        ====
-->

<!ELEMENT root (%Message1;)*>

<!--
        ==============
        Callback-Layer
        ==============

   A Message can be of three types:
     - normal message
     - request message
     - reply message
-->
```

```
<!ELEMENT normal-message %Message2;>

<!ELEMENT request-message %Message2;>
<!ATTLIST request-message
        cbid      %ID;    #REQUIRED
>

<!ELEMENT reply-message %Message3;>
<!ATTLIST reply-message
        cbid      %ID;    #REQUIRED
>

<!--
        ==============
        Avatar-Message needed for switch procedure
        Consits of "Front-Vector" and Avatar-Vector
        At this stage of development just 2-D
        ==============
-->
<!ELEMENT avatar EMPTY>
<!ATTLIST avatar
        xfront    %Float; #REQUIRED
        yfront    %Float; #REQUIRED
        zfront %Float; #IMPLIED
        xavatar %Float; #REQUIRED
        yavatar %Float; #REQUIRED
        zavatar %Float; #IMPLIED
>

<!--
        ==============
        Message for adjusting the weighting of the mrm indicators.
        The sum of distance and lineOfSight must be 1.
        ==============
-->
<!ELEMENT mrm-quantifier EMPTY>
<!ATTLIST mrm-quantifier
        distance %Float; #REQUIRED
        lineOfSight %Float; #REQUIRED
        relations (0 | 1 | 2) #REQUIRED
>

<!ELEMENT lockMRM EMPTY>

<!ELEMENT unlockMRM EMPTY>

<!--
        ==============
        Administration
        ==============
-->

<!--
This message can be sent in initialisation phase. The computationtime is the time a path computation will take.
Simulation will send a move-robot message and, after waiting for computationtime simulation time, send a
check-arrival message. As simulation time is ahead of server time, timeprecision limits this inaccuracy by
blocking simulation until the difference between simulation and server time is less then timeprecision. Note:
The worst thing which can happen is that simulation gets to know timperecision later that a robot arrived.
Setting timeprecision to a reasonable value (e.g. 100) makes it easier for visualisation to produce smooth
pictures. On the other hand,  setting both values to 0 guarantees accurate simulation results.
```

```
-->
<!ELEMENT mp-prefs EMPTY>
<!ATTLIST mp-prefs
        computationtime %Int;    #REQUIRED
        timeprecision     %Int;    #REQUIRED
>

<!ELEMENT start EMPTY>
<!ELEMENT pause EMPTY>
<!ELEMENT stop  EMPTY>
<!ELEMENT save EMPTY>
<!ATTLIST save
        pathname   %String;      #REQUIRED
>

<!ELEMENT maxspeed  EMPTY>

<!ELEMENT normalspeed  EMPTY>

<!ELEMENT timestamp EMPTY>
<!ATTLIST timestamp
        %Timestamp;
>

<!ELEMENT timefactor EMPTY>
<!ATTLIST timefactor
        %Timestamp;
        value     %Int;    #REQUIRED
>


<!--
        ===============
        Initialization
        ===============
-->

<!--
If a visualisation server connects to the simulation, the simulation needs switch to pause state. Afterwards, all
information about the current simulation status is transferred to the server using the messages in this section.
Note: Some messages have to be sent in a special order. If all messages are sent, simulation sends a message
of this type to the server. After processing the input, the server will send a start message to start the
simulation.
-->
<!ELEMENT endOfInitialization EMPTY>

<!--
        Simulation specifies the database location etc. Encryption needed!
-->
<!ELEMENT databaseinfo EMPTY>
<!ATTLIST databaseinfo
        url             %String;        #REQUIRED
        username        %String;        #REQUIRED
        password        %String;        #REQUIRED
>

<!--
Dockingpoints belong to a Buildingblock. A Buildingblock can have Dockingpoints. Robots can be sent by
specifying id of Buildingblock and id of Dockingpoint, or by just specifying the Buildingblock (in this case
Motionplanning chooses a Dockingpoint). Usually Dockingpoints are saved in the db together with the 3D model
```

```
they belong to, but they can also be added by specifying the Buildingblock and a position relative to the
position of the Buildingblock
-->
<!ELEMENT dockingpoint (databaseid | (description?, location))>
<!ATTLIST dockingpoint
        id              %Int;   #REQUIRED
        idbb            %Int;   #REQUIRED
>

<!--
Tokens can be moved automatically along a specified path. (e.g. a box moved along a production line). A Token
path describes such a path. A movement of a token can be moved by sending an AnimateToken message.
-->

<!ELEMENT tokenpath (databaseid | (description?, location, location+))>
<!ATTLIST tokenpath
        id              %Int;   #REQUIRED
        idbb            %Int;   #REQUIRED
>

<!--
Buildingblocks represent static objects, e.g. machines. They must have an associated mesh which is saved in
the db. Visualisation will get the Dockingpoints and Tokenpathes out of the database, but additional ones can
be specified. If the bulding block mesh contains other building block meshes (e.g. if it is a hall),
the optional simulationBoundary flag has to be set.
-->

<!ELEMENT buildingblock (description?, mesh, location, significancepoint?)>
<!ATTLIST buildingblock
        id                      %Int;           #REQUIRED
        simulationBoundary      %Boolean;       "false"
>

<!--
A Robot is an object for which movements can be ordered with a MoveRobot or a MovePoolMember message.
Initial position, maxSpeed and radius must be specified. NOTE: All Dockingpoints, Tokenpathes and
Buildingblocks have to be transmitted before the first Robot!
-->

<!ELEMENT robot (description?, mesh, location)>
<!ATTLIST robot
        type    (portal2d | portal3d | forklifter | worker)    #REQUIRED

        id              %Int;   #REQUIRED
        idpool  %Int;   #IMPLIED

        speed   %Float; #REQUIRED
        radius  %Float; #REQUIRED
        accel   %Float; #IMPLIED
        decel   %Float; #IMPLIED
>

<!--
A Token is an object which can be displayed at a fixed position
or moved along a Buildingblock via a Tokenpath.
-->
<!ELEMENT token (description?, mesh)>
<!ATTLIST token
        id              %Int;   #REQUIRED
>
```

```
<!--
Robots don't have to, but should belong to Pools, which are introduced with messages of this type. Movements
of Robots should be ordered by using a MovePoolMember message (Simulation can't know how long it takes a
special Robot to go from A to B!)
-->

<!ELEMENT robotpool (description?)>
<!ATTLIST robotpool
        id                  %Int;    #REQUIRED
>


<!--
        ==========
        Properties
        ==========
-->

<!--
If the Visualisation is interested in the properties of a special Buldingblock, it sends a message of this type to
the Simulation. Simulation answers with a PropertyMessage immediately, and sends a
PropertyChangedMessage as soon as one or more properties have changed, but no more often than every
interval milliseconds. A subscription can be cancelled by Visualisation with an UnsubscribeObjectProperties
message.
-->

<!ELEMENT subscribe-properties EMPTY>
<!ATTLIST subscribe-properties
        idbb                %Int;    #REQUIRED
        interval  %Int;    #REQUIRED
>

<!--
Cancels a subscription of properties of the given object.
-->
<!ELEMENT unsubscribe-properties EMPTY>
<!ATTLIST unsubscribe-properties
        idbb                %Int;    #REQUIRED
>

<!--

Contains a list of all properties of an object, including the current values of those properties.
-->
<!ELEMENT object-properties (property*)>
<!ATTLIST object-properties
        %Timestamp;
        idbb      %Int;    #REQUIRED
>

<!ELEMENT property (description?, (simple-type | enum-type | list| table | random), evaluationtype?)>
<!ATTLIST property
        name        %String;        #REQUIRED
        readonly(yes|no)    "yes"
>

<!ELEMENT simple-type EMPTY>
<!ATTLIST simple-type
        type      (long|double|bool|string|time)      #REQUIRED
```

```
        value       %String;    #REQUIRED
        lowerbound      %String;        #IMPLIED
        upperbound      %String;        #IMPLIED

>

<!ELEMENT enum-type (item+)>

<!ELEMENT item EMPTY>
<!ATTLIST item
        value    %String;        #REQUIRED
>

<!--
a list can have serveral items
-->

<!ELEMENT list (simple-type*)>


<!--
erlaubt nur gleichmaessige tabellen
-->
<!ELEMENT table  (content*)>
<!ATTLIST table
         rowsize  %Int;   #REQUIRED
         columnsize     %Int;     #REQUIRED
>

<!ELEMENT content  (simple-type)>
<!ATTLIST content
         row      %Int;   #REQUIRED
         column  %Int;   #REQUIRED
>



<!ELEMENT random EMPTY>
<!ATTLIST random
        type     (exponential | normal | triangular | uniform)        #REQUIRED
        value           %String;        #REQUIRED
        streamnumber    %Int;           #REQUIRED
        meanvalue               %Float;         #IMPLIED
        variance        %Float;         #IMPLIED
        lowerbound              %Float;         #IMPLIED
        upperbound              %Float;         #IMPLIED
>


<!--
Contains a list of all properties which have changed since the last object-properties or properties-changed
message has been sent. If sent by Visualisation, then a user wants to adjust a setting. It's up to the simulation
to decide if it allows that. If it decides to change the property, a properties-changed message has to be sent as
usual. The timestamp is only needed if the message is sent from Simulation to Visualisation.
-->

<!ELEMENT properties-changed (simple-update*|enum-update* |list-update*|table-update*|random-
update*)>
<!ATTLIST properties-changed
        %Timestamp;
```

```
        idbb      %Int;    #REQUIRED
>

<!ELEMENT list-update (remove-first?,add-last?)>
<!ATTLIST list-update
        name       %String;       #REQUIRED
>

<!ELEMENT remove-first EMPTY>
<!ATTLIST remove-first
        quantity    %Int;          #REQUIRED
>

<!ELEMENT add-last (simple-type+)>

<!ELEMENT table-update (content+)>
<!ATTLIST table-update
        name    %String;          #REQUIRED
>

<!ELEMENT random-update EMPTY>
<!ATTLIST random-update
        name    %String;          #REQUIRED
        type     (exponential | normal | triangular | uniform)         #IMPLIED
        value              %String;          #IMPLIED
        streamnumber    %Int;             #IMPLIED
        meanvalue                %Float;           #IMPLIED
        variance          %Float;           #IMPLIED
        lowerbound               %Float;           #IMPLIED
        upperbound               %Float;           #IMPLIED
>

<!ELEMENT simple-update EMPTY>
<!ATTLIST simple-update
        name    %String;                                                  #REQUIRED
        type     (long|double|bool|string|time)     #IMPLIED
        value    %String;                                                  #REQUIRED
>

<!ELEMENT enum-update (item+)>
<!ATTLIST enum-update
        name    %String;          #REQUIRED
>


<!--
        ===============
        Evaluationtypes
        ===============
-->

<!ELEMENT evaluationtype (meanvariance | timetable | display | trafficlight | gauge | histogram | signal)>
<!ATTLIST evaluationtype
        priority  (high | medium | low)  #REQUIRED
>
<!--
Mittelwert und Standartabweichung (z.B. 5 +- 2)
-->
<!ELEMENT meanvariance (valuecolumn?)>
```

```
<!--
Bei Tabellen: Aus welcher Spalte sollen die Daten genommen werden
-->
<!ELEMENT valuecolumn (#PCDATA)>

<!--
Diagramm mit f(timecolumn(i)) = valuecolumn(i) Es kann angegeben werden, ob Datenpunkte als dicke Punkte
dargestellt werden sollen und welche Farbe der Graph haben soll
-->

<!ELEMENT timetable (yline*, timecolumn, valuecolumn)>
<!ATTLIST timetable
        dots (true | false) #REQUIRED
        color CDATA #IMPLIED
>
<!--
Reele Zahl. f(x)=yline. Waagerechte Linie im Diagramm.
-->
<!ELEMENT yline (#PCDATA)>

<!--
Wert 1..N Nummer der Spalte, in der die Zeitwerte stehen (x-Achse)
-->
<!ELEMENT timecolumn (#PCDATA)>

<!--
Einfache Anzeige eines Wertes.
-->
<!ELEMENT display (#PCDATA)>

<!--
Eine Ampel. thresholdyellow gibt den Anfangswert für den gelben Bereich, thresholdred den für den roten
Bereich an.
-->
<!ELEMENT trafficlight (thresholdyellow, thresholdred)>

<!--
Ein Drehzahlmesser. thresholdyellow gibt den Anfangswert für den gelben Bereich, thresholdred den für den
roten Bereich an. Anordnung(im Uhrzeigersinn): gruen, gelb, rot
-->
<!ELEMENT gauge (thresholdyellow?, thresholdred?)>

<!--
Ein Wert aus der Menge der reellen Zahlen
-->
<!ELEMENT thresholdyellow (#PCDATA)>

<!--
Ein Wert aus der Menge der reellen Zahlen
-->
<!ELEMENT thresholdred (#PCDATA)>

<!--
Ein Wert aus der Menge der reellen Zahlen
-->
<!ELEMENT threshold (#PCDATA)>

<!--
```

```
Ein Histogramm ist ein Balkendiagramm, bei dem jeder Balken einen Wertebereich darstellt. Es werden nur die
Ereignisse gezählt, wenn ein Wert in ein Intervall passt. Diese Intervalle werden durch die threshold-Wert
angegeben (z.B. 5,10,20,30 -> Werte <= 5 in der ersten, Werte >5 und <= 10 in der zweiten, etc.)
-->

<!ELEMENT histogram (threshold*, valuecolumn?)>

<!--
Einfache Darstellung eines grünen oder roten Kreises. Es kann ausgewählt werden,
        ob true gut(gruen) oder schlecht(rot) ist.
-->
<!ELEMENT signal EMPTY>
<!ATTLIST signal
    truegreen    (true|false)    #REQUIRED
>




<!--
        ============
        Significance
        ============
-->

<!--
Changes the significance of the given object. This causes the object to be displayed in a more detailed way. If
the significance of an object gets higher then a threshold which can be chosen by each visualisation user, a
marker will be displayed at the position of the object.
-->

<!ELEMENT significance EMPTY>
<!ATTLIST significance
        %Timestamp;
        idbb     %Int;              #REQUIRED
        value    %Float;            #REQUIRED
        message          %String;          #IMPLIED
>


<!--
        ======
        Tokens
        ======
-->

<!--
This message can be used to put Tokens into other Tokens (e.g. shoes in a box). It can also be used to put a
Token onto a Robot. Each Token must have a location which is relative to the reference point of the Token
which it is put into.
-->

<!ELEMENT include-token (include+)>
<!ATTLIST include-token
        %Timestamp;
        idparent %Int;    #REQUIRED
>

<!ELEMENT include (location)>
<!ATTLIST include
```

```
        idchild   %Int;    #REQUIRED
>

<!--
To remove Tokens from a Token or a Robot
-->
<!ELEMENT exclude-token (exclude+)>
<!ATTLIST exclude-token
        %Timestamp;
        idparent %Int;    #REQUIRED
>

<!ELEMENT exclude EMPTY>
<!ATTLIST exclude
        idchild   %Int;    #REQUIRED
>

<!--
This message is used to animate a Token. The Token will start a the first node of the path and will move along
all the path nodes. The movement will take the specified time. At the end, the token will be displayed at the
location of the last node of the path, or hidden if the hide flag is set to true.
-->

<!ELEMENT animate-token EMPTY>
<!ATTLIST animate-token
        %Timestamp;
        idtoken           %Int;     #REQUIRED
        idpath            %Int;    #REQUIRED
        starttime         %Long;    #REQUIRED
        endtime           %Long;    #REQUIRED
        startpos %Float;   "0"
        endpos            %Float;   "1"
        hide              %Boolean; "true"
        bbName                  %String;  #IMPLIED
        ichannel          %String;  #IMPLIED
        ochannel          %String;  #IMPLIED
>

<!--
To remove a token from the visualisation.
-->

<!ELEMENT remove-token EMPTY>
<!ATTLIST remove-token
        %Timestamp;
        idtoken  %Int;    #REQUIRED
        bbName                  %String;  #IMPLIED
        ichannel          %String;  #IMPLIED
        ochannel          %String;  #IMPLIED
>


<!--
        ======
        Robots
        ======
-->

<!--
Removes a robot from the Visualisation
```

```
-->

<!ELEMENT remove-robot EMPTY>
<!ATTLIST remove-robot
        %Timestamp;
        idrobot  %Int;    #REQUIRED
>

<!--
Used to order a movement of a Robot. See move-poolmember message.
-->

<!ELEMENT move-robot EMPTY>
<!ATTLIST move-robot
   %Timestamp;
        idrobot                 %Int;    #REQUIRED
        iddestination           %Int;    #REQUIRED
        idorder                 %Int;    #IMPLIED
>

<!--
Used to order a movement of a Robot. Motion planning will choose the Robot which is nearest to the target
Buldingblock and member of the specified pool. If the given targetid identifies a building block, motion planning
will choose a dockingpoint. If the id identifies a docking point, that point will be the destination. Motion planning
needs some time to compute the path. To avoid blocking the simulation kernel for the whole time, motion
planning will send an still-computing-path immediately. After waiting for the specified period of time, simulation
kernel uses a check-arrival message. The answer will be a still-computing-path message (if motion planning is
still computing) or an expected-arrival-time message (if motion planning has finished the computation. The
expected-arrival-time message will contain the robot and dockingpoint chosen. Simulation will ask  at the
expected time if the robot has reached its destination. If this is the case, motion planning will send an Arrival
message, else an ExpectedArrivalTime message containing the new expected arrival time. If the ordered Robot
movement is not possible, an error message will be sent.
-->

<!ELEMENT move-poolmember EMPTY>
<!ATTLIST move-poolmember
   %Timestamp;
        idpool                  %Int;    #REQUIRED
        iddestination           %Int;    #REQUIRED
>

<!--
See move-poolmember message.
-->

<!ELEMENT still-computing-path EMPTY>
<!ATTLIST still-computing-path
        idorder %Int;     #REQUIRED
        waitFor  %Int;    #REQUIRED
>

<!--
Sent to check if the given Robot has reached its destination.
-->

<!ELEMENT check-arrival EMPTY>
<!ATTLIST check-arrival
        %Timestamp;
        idorder  %Int;    #REQUIRED
>
```

```
<!--
Contains the expected arrival time of the order with the given id.
-->

<!ELEMENT expected-arrival-time EMPTY>
<!ATTLIST expected-arrival-time
        idorder         %Int;    #IMPLIED
        idrobot         %Int;    #IMPLIED
        arrivaltime     %Long;  #REQUIRED
>

<!--
        Sent if the robot has reached its destination.
-->

<!ELEMENT arrival EMPTY>

<!--
Sent if the robot isn't needed any more. Note that a robot is reserved by motion planning. Simulation must free
it if it's no longer needed. See move-poolmember message.
-->

<!ELEMENT free-robot EMPTY>
<!ATTLIST free-robot
        %Timestamp;
        idorder         %Int;    #REQUIRED
        idrobot         %Int;    #REQUIRED
>

<!ELEMENT no-error (description?)>

<!ELEMENT error (description?)>

<!--
   Multi resolution support
-->

<!ELEMENT buildingblocks-in-view (objectid+)>

<!ELEMENT buildingblocks-out-of-view (objectid+)>


<!--
   Some misc. stuff.
-->

<!ELEMENT description (#PCDATA)>

<!ELEMENT databaseid EMPTY>
<!ATTLIST databaseid
        value    %Int;    #REQUIRED
>

<!ELEMENT objectid EMPTY>
<!ATTLIST objectid
        value  %Int;    #REQUIRED
>

<!ELEMENT position EMPTY>
```

```
<!ATTLIST position
        xpos     %Float; #REQUIRED
        ypos     %Float; #REQUIRED
        zpos     %Float; #REQUIRED
>

<!--
NOTE: x- and y-angle are not supported yet, they're just ignored.
-->
<!ELEMENT rotation EMPTY>
<!ATTLIST rotation
        xangle  %Float; "0"
        yangle   %Float; "0"
        zangle   %Float; "0"
>

<!ELEMENT mesh (databaseid)>
<!ATTLIST mesh
        scale     %Float; "1"
>

<!ELEMENT significancepoint (position)>

<!ELEMENT location (position, rotation?)>

<!--
        EOF
-->
```