# Softwarearchitektur für die interaktive Simulation mobiler Arbeitsmaschinen in virtuellen Umgebungen

Von der Fakultät Maschinenwesen der
Technischen Universität Dresden angenommene

## Dissertation

zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
von

Timo Penndorf
geboren am 15.03.1977 in Bautzen.

| | |
|---|---|
| Gutachter : | Prof. Dr.-Ing. habil. Günter Kunze |
| | Prof. Dr. rer. nat. habil Uwe Aßmann |
| Prüfer : | Prof. Dr.-Ing. habil. Günter Kunze |
| | Prof. Dr.-Ing. Michael Beitelschmidt |

| | |
|---|---|
| Tag der Einreichung : | 15.06.2012 |
| Tag der mündlichen Prüfung : | 01.02.2013 |

*The nice thing about standards is that there are so many to choose from.*

Tanenbaum, A. S.: Computer Networks. Prentice Hall Ptr, 1996.

# Vorwort

Der wesentliche inhaltliche Teil dieser Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter an der Professur für Baumaschinentechnik der Technischen Universität Dresden.
Mein besonderer Dank gilt Herrn Prof. Dr.-Ing. Günter Kunze für die Bereitstellung einer extrem interessanten und anspruchsvollen Aufgabe in Form eines Fahrsimulators sowie die Unterstützung während und nach meiner Zeit an der Professur. Herrn Prof. Dr. rer. nat. Uwe Aßmann danke ich für die Durchsicht und Begutachtung meiner Arbeit. Weiterhin möchte ich Herrn Prof. Dr.-Ing. Michael Beitelschmidt für die Übernahme der Nebenfachprüfung danken.
Den Kolleginnen und Kollegen der Professur für Baumaschinentechnik danke ich für das persönlich und fachlich herausragende Arbeitsumfeld sowie die spannenden Diskussionen und inhaltlichen Anregungen. Besonders hervorheben möchte ich an dieser Stelle Frau Dr.-Ing. Ines Gubsch und Herrn Dr.-Ing. Miguel Mothes.
Den Herren Christian Schubert, Jens Frenkel und Sebastian Voigt danke ich für die Weiterführung meiner Arbeit, vor allem für die in diesem Zusammenhang geführten Fachdiskussionen. Herrn Prof. Dr. rer. nat. Markus Wacker und Herrn Martin Großer danke ich für die intensive Zusammenarbeit über die Grenzen der Institutionen hinweg.
Der schriftliche Teil dieser Arbeit entstand im Anschluss an meine Tätigkeit in Dresden an der Universitätsbibliothek Tübingen. Für die Ausstattung und freie Benutzung der Ressourcen danke ich dem Land Baden-Württemberg.
Mein ganz besonderer Dank gilt meiner Familie. Meiner Frau Sandra danke ich für die Unterstützung, das Korrekturlesen und die Geduld speziell während der Phase des Schreibens dieser Arbeit.

Aichtal, Juni 2012

Timo Penndorf

Im Rahmen dieser Arbeit wird der Begriff des Bedieners bzw. Fahrers sowie des Anwenders in männlicher Form verwendet. Alle Ausführungen gelten sowohl für weibliche als auch für männliche Bediener, Fahrer und Anwender.

# Inhaltsverzeichnis

# Symbole und Abkürzungen

## Symbole

| | |
|---|---|
| $\mathcal{O}(n^2)$ | Problemordnung (quadratische Abhängigkeit) |
| $x$ | Allgemeine skalare Zustandsgröße |
| $t$ | Zeit |
| $f(x,t)$ | Skalare Funktion über einer skalaren Zustandsgröße und der Zeit |
| $x_1, x_2$ | Koordinaten der Massen des Zweimassenschwingers |
| $v_1, v_2$ | Geschwindigkeiten der Massen des Zweimassenschwingers |
| $d_1, d_2$ | Dämpfungskonstanten des Zweimassenschwingers |
| $c_1, c_2$ | Federsteifigkeiten des Zweimassenschwingers |
| $m_1, m_2$ | Einzelmassen des Zweimassenschwingers |
| $F\sin(\Omega t)$ | Periodische Anregung des Zweimassenschwingers |
| $\mathbf{y}$ | Allgemeiner Vektor von Zustandsgrößen |
| $\mathbf{f}(\mathbf{y},t)$ | Vektorfunktion über einem Zustandsvektor und der Zeit |
| $\mathbf{H}$ | Massematrix |
| $\mathbf{q}$ | Vektor der generalisierten Koordinaten |
| $\mathbf{q}_c$ | Vektor der Corioliskräfte |
| $\mathbf{q}_g$ | Vektor der Gewichtskräfte |
| $\mathbf{q}_f$ | Vektor der Aktuatorkräfte und äußeren Kräfte |
| $l_1, l_2$ | Längen in viergliedrigem Mechanismus |
| $\phi, \psi$ | Winkel in viergliedrigem Mechanismus |
| $p$ | Zustandsgröße Druck in einem diskreten Ölvolumen |
| $V$ | Volumenmaß eines diskreten Ölvolumens |
| $K$ | Kompressionsmodul eines diskreten Ölvolumens |
| $\alpha$ | Blendenkoeffizient |
| $\mathbf{z}$ | Zustandsvektor des Modells für einen Hydraulikzylinder |
| $z_1, z_2, z_3, z_4$ | Komponenten des Zustandsvektors $\mathbf{z}$ |
| $A_1, A_2$ | Kolben- und Ringfläche |

| | |
|---|---|
| $F$ | Kraft auf Hydraulikzylinder |
| $m_K$ | Masse des Kolbens des Hydraulikzylinders |
| $p_1, p_2$ | Druck auf der Kolben- und Stangenseite |
| $Q_1, Q_2$ | Volumenstrom auf der Kolben- und Stangenseite |
| $Q_{12}$ | Leckagevolumenstrom |
| $l_0, l_1$ | Kolben- und Zylinderlänge |
| $h_K$ | Dicke des Kolbens |
| $m$ | Masse der Gesamtmaschine |
| $v$ | Translatorische Geschwindigkeit der Gesamtmaschine |
| $M$ | Antriebsdrehmoment |
| $\phi, \dot{\phi}$ | Bewegungszustand (Winkel, Winkelgeschwindigkeit) des Rades |
| $P$ | Arbeitspunkt des Reifenmodells (Aufstandspunkt) |
| $h(x, y)$ | Funktion zur Abfrage des Geländeprofils |
| $F_x, F_y$ | Reaktionskräfte des ebenen Reifenmodells |
| $n_f$ | Freiheitsgrad des Starrkörpermodells des Radladers |
| $n_k$ | Anzahl der starren Körper des Radladermodells |
| $n_F$, $n_T$ | Anzahl der äußeren Kräfte und Momente |
| $\mathbf{g}$ | Vektor der Erdbeschleunigung in globalen Koordinaten |
| $\mathbf{S}_i$ | Globale Koordinaten des Schwerpunktes des Körpers $i$ |
| $\mathbf{R}_i$ | Rotationsmatrix des Körpers $i$ (bezogen auf globales Koordinatensystem) |
| $\mathbf{J}_{Ti}$ | Translatorische JACOBI-Matrix (bezogen auf $\mathbf{S}_i$) |
| $\mathbf{J}_{Ri}$ | Rotatorische JACOBI-Matrix (bezogen auf $\mathbf{R}_i$) |
| $\tilde{\mathbf{J}}_{Tk}$ | Translatorische JACOBI-Matrix (beliebiger Punkt $k$) |
| $\tilde{\mathbf{J}}_{Rk}$ | Rotatorische JACOBI-Matrix (beliebiges Koordinatensystem $k$) |
| $m_i$ | Masse des Körpers $i$ |
| $\mathbf{I}_i$ | Trägheitstensor des Körpers $i$ in lokalen Koordinaten |

## Abkürzungen

| | |
|---|---|
| API | Application Programming Interface |
| CAN | Controller Area Network |
| CAVE | Computer Aided Virtual Environment |
| COM | Component Object Model |

| | |
|---|---|
| CORBA | Common Object Request Broker Architecture |
| CFD | Computational Fluid Dynamics |
| DAE | Differential Algebraic Equation |
| FMI | Functional Mockup Interface |
| GUI | Graphical User Interface |
| GTK | Gimp Toolkit |
| HTTP | Hypertext Transfer Protocol |
| IDL | Interface Definition Language |
| IO | Input & Output |
| Lang | Language |
| MKS | Mehrkörpersystem |
| ODE | Ordinary Differential Equation |
| OSG | OpenSceneGraph |
| RPC | Remote Procedure Call |
| STL | Standard Template Library |
| UDP | User Datagram Protocol |
| VLC | Video Lan Client |
| VR | Virtuelle Realität (Virtual Reality) |
| VRJ | VRJuggler |
| WCET | Worst Case Execution Time |
| XML | Extensible Markup Language |

# Tabellenverzeichnis

# Abbildungsverzeichnis

# Kurzfassung

Die numerische Simulation ist ein unverzichtbares Werkzeug bei der Produktentwicklung geworden. Bereits in den frühen Phasen von Studien und Konzepten können unterschiedliche Lösungsansätze für eine Aufgabenstellung bewertet werden. In diesem Zusammenhang wird von *virtuellen Prototypen* gesprochen. Bei der Simulation mobiler Arbeitsmaschinen erfordert das die Einbindung des Bedieners. Die technologische Leistungsfähigkeit wird wesentlich durch die Interaktion zwischen Bediener und Maschine geprägt. Die Beschreibung des Bedieners durch mathematische Modelle ist bis zum gegenwärtigen Zeitpunkt nicht mit belastbaren Resultaten erfolgt.
Die Entwicklungen im Bereich der Simulationstechnologie und der Computergrafik ermöglichen die Durchführung interaktiver Simulationen in komplexen virtuellen Welten. Damit lässt sich der Bediener direkt in die Simulation einbinden und es können zusätzliche Potenziale, wie z. B. bei der Untersuchung der Mensch-Maschine-Interaktion erschlossen werden. Durch die interaktive Simulation in Virtual-Reality-Systemen werden neue Anforderungen an die Simulationssoftware gestellt. Zur Interaktion mit dem Bediener müssen die Eingaben aus den Bedienelementen in Echtzeit verarbeitet und audiovisuelle Ausgaben generiert werden. Dabei sind neben den mathematisch-physikalischen Aspekten der Simulation auch Problemfelder wie Synchronisation, Kommunikation, Bussysteme und Computergrafik zu behandeln. Die Anpassung des Simulationssystems an unterschiedliche Aufgabenstellungen erfordert ein flexibel konfigurierbares Softwaresystem.
Als Lösung dieser Aufgabenstellung wird eine Softwarearchitektur vorgestellt, welche die unterschiedlichen Problemfelder durch klar voneinander abgegrenzte Komponenten mit entsprechenden Schnittstellen behandelt. Das entstandene Softwaresystem ist flexibel und erweiterbar. Die Simulationsaufgabe wird durch die Konfiguration des Komponentensystems spezifiziert. Die entstehenden Konfigurationsdateien bilden die Anwendungslogik ab und stellen daher einen der wesentlichen Kostenfaktoren bei der Realisierung interaktiver Simulationen dar. Zur Erhöhung der Wiederverwendbarkeit bestehender Konfigurationsfragmente wird ein kompositionsbasierter Ansatz auf der Basis von Skriptsprachen gewählt.

# Preface

Numerical simulation has evolved into an indispensable tool in modern product development. Even in the early design phases of studies and concepts several different approaches for one individual task can be evaluated. In this context the term *virtual prototypes* can be used. For effective simulation of mobile construction site machinery incorporation of the operator's influence is required. The technological performance of the machinery is essentially influenced by the interaction between the operator and the machine. Currently, there are no known mathematical models describing the operator's behaviour, which give substantiated results.

The latest developments in computing technology and computer graphics facilitate interactive simulations in complex virtual worlds. This allows not only the operator to be linked to the simulation but also the investigation of additional research areas such as human-machine-interaction. The application of interactive simulation in virtual reality systems places new demands on the simulation software. Due to the interaction not only input signals from the instruments have to be processed but also audio and visual output has to be generated in real time. In addition to the mathematical and physical aspects of simulation, problems in the areas of synchronisation, communication, bus systems and computer graphics also have to be solved. The adaption of the simulation system to new tasks requires a flexible and highly configurable software system.

As a response to these demands, a software architecture is presented which partitions the various problems into finite components with corresponding interfaces. The partitioning results in a flexible and extendable software system. The simulation task is specified by the configuration of the component system. The resulting configuration files reflect the application logic and therefore represent one of the main cost factors in the realisation of the interactive simulations. A composition-based approach is chosen as it raises the level of reuse of existing configuration fragments. This approach is based on scripting languages.

# 1 Einleitung, Motivation und Ansatz

Der zunehmende Wettbewerb unter den Herstellern mobiler Arbeitsmaschinen führt zu immer komplexeren Produkten, welche den ständig steigenden Anforderungen der globalen Märkte gerecht werden müssen. Die daraus resultierende Zunahme des Entwicklungsaufwandes bei gleichzeitig immer kürzer werdenden Entwicklungszyklen erfordert bereits in den frühen Phasen der Produktentwicklung eine Absicherung von Entscheidungen.
Die numerische Simulation hat sich auf diesem Gebiet als Werkzeug etabliert. Mit Hilfe der in den Simulationsmodellen abgebildeten *virtuellen Prototypen* lassen sich komplexe Fragestellungen effizient und unter reproduzierbaren Bedingungen analysieren. Durch die einfache Anpassung der Modelle lassen sich umfangreiche Parameterstudien mit vertretbarem Aufwand durchführen. Der Einsatz realer Prototypen für die Analyse komplett neuartiger Konzepte ist allein durch die Verfügbarkeit entsprechender Komponenten und Teilsysteme begrenzt. Durch die Abstraktion in Rechenmodelle können zudem auch Teile des zukünftigen technischen Systems durch mathematische Zusammenhänge abgebildet werden, welche noch nicht vollständig entwickelt sind.
Mobile Arbeitsmaschinen sind dadurch charakterisiert, dass sie für die Verrichtung einer Arbeitsaufgabe (Prozess) konzipiert sind und durch einen Menschen bedient werden. Die Maschine ist somit das Bindeglied zwischen dem Bediener und dem Prozess ([Ras86]). Der Bediener nimmt über die menschlichen Sinneskanäle Informationen aus der Umgebung, von der Maschine und dem Prozess auf. Die resultierenden Handlungen beeinflussen direkt die Maschine und wirken durch die Arbeitswerkzeuge indirekt auf den Prozess und damit auf die Umgebung (Abbildung 1.1).
Für die Entwicklung mobiler Arbeitsmaschinen ergeben sich daraus je nach Aufgabenstellung wesentliche Konsequenzen. Die Simulation unterschiedlicher Konzepte für Antriebe, Tragstrukturen und Arbeitswerkzeuge erfordert neben den eigentlichen Rechenmodellen für das technische System auch ein geeignetes Bedienermodell. Liegt ein solches Modell nicht vor, so muss der Bediener in die Simulation interaktiv einbezogen werden. Für Aufgabenstellungen mit dem Ziel der Bewertung des ganz-

heitlichen Systems aus Bediener, Maschine und Prozess ist die Einbeziehung des Bedieners zwingend notwendig. Gegenwärtige Entwicklungstendenzen zeigen, dass die technologischen Potenziale einer Maschine nur dann ausgenutzt werden, wenn die Schnittstellen zwischen Mensch und Maschine entsprechend leistungsfähig sind.

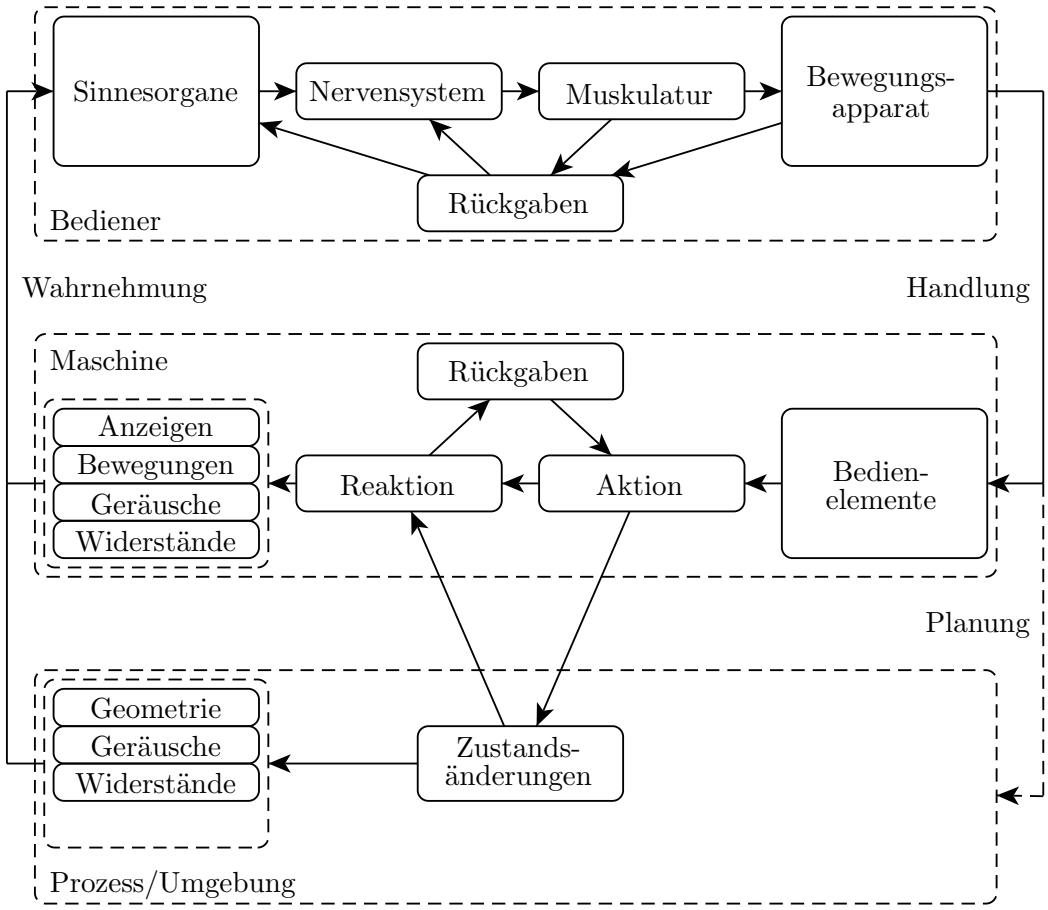


Abbildung 1.1: Bediener im Kontext von Maschine, Prozess und Umgebung

Die Kombination bekannter Algorithmen der numerischen Simulation mit den Methoden der virtuellen Realität (VR) ermöglicht die Einbeziehung des Bedieners in eine Maschinensimulation. Die Handlungen des Bedieners im Umgang mit dem virtuellen Prototypen ersetzen das Bedienermodell.

Die Verbindung von Simulationstechnologie und virtueller Realität ergeben neue Fragestellungen vor allem aus Sicht der eingesetzten Softwaresysteme. Ein Ansatz zur ganzheitlichen Abbildung der Maschine ist die domänenübergreifende Kopplung von Simulationswerkzeugen ([Dro04; GKL06; GGM10]). Aufgrund fehlender Standards

entsteht jedoch ein hoher Aufwand für die Implementation und Pflege der entsprechenden Schnittstellen. Dieser Aufwand wird durch die Integration von VR-Systemen deutlich erhöht, da diese ebenfalls nicht standardisiert sind. Im Ergebnis entsteht ein System, das im Wesentlichen von spezialisierten Schnittstellen zwischen einzelnen Werkzeugen geprägt ist (Abbildung 1.2). Die Anzahl der Schnittstellen ist mit $\mathcal{O}(n^2)$ abhängig von der Anzahl der eingesetzten Softwarewerkzeuge.

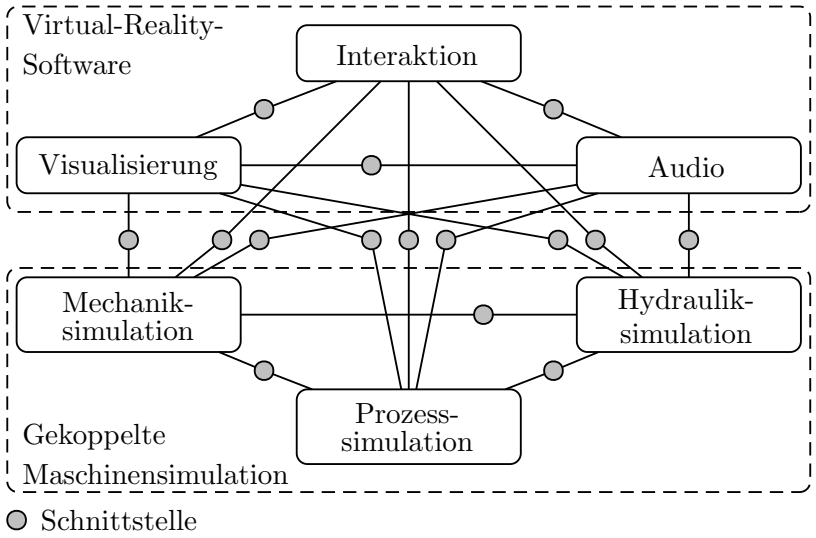


Abbildung 1.2: Schnittstellen bei Werkzeugkopplung

Das hohe Entwicklungsrisiko eines solchen Ansatzes besteht darin, dass jedes neue Werkzeug mit den entsprechenden Schnittstellen versehen werden muss. Die Veränderung eines Werkzeuges bedingt die Anpassung aller relevanten Schnittstellen. Vorhandene Ansätze mit generischen Plattformen für gekoppelte Simulationen erfordern die Implementation und Pflege der zugehörigen Adapter ([Bay+12]) bzw. die Fähigkeit des Werkzeuges, Modelle und Daten für die jeweilige Plattform aufzubereiten ([RG11]).

Die interaktive Simulation erfordert eine Simulation in Echtzeit. Diese unterscheidet sich von den klassischen Anwendungsfällen der Simulationswerkzeuge dadurch, dass z. B. automatische Schrittweitensteuerungen nur bedingt für Echtzeitberechnungen geeignet sind. Die Schnittstellen der Simulationswerkzeuge müssen neben dem Austausch der entsprechenden numerischen Daten auch die notwendige Synchronisation mit dem VR-System ermöglichen.

Neben der offenen Diskussion der numerischen Effekte bei der Kopplung von Simulationswerkzeugen ([AS09; TF11]) stellen sowohl die Verfügbarkeit der einzelnen Werkzeuge als auch die beschriebene Schnittstellenproblematik hohe Risiken für die Anwendung interaktiver Simulationen in VR-Systemen dar.
Vorhandene Software für VR-Systeme ist in der Lage, physikalische Modelle abzubilden ([BW98; Coh+95; YFR06; Wag01]). Dabei werden im Wesentlichen die Effekte berücksichtigt, welche für eine ansprechende Visualisierung notwendig sind. In ganzheitlichen Maschinenmodellen sind zusätzliche, nicht sichtbare Effekte (z. B. aus Steuerung und Regelung) abzubilden. Die durch VR-Systeme gegebenen Möglichkeiten der Modellbildung und Simulation sind daher nicht ausreichend.
Auf dem Gebiet der Fahrsimulatoren wird die Verbindung von numerischer Simulation und VR-Technologie bereits angewendet. Die Anwendung reicht von der Bewertung von Assistenzsystemen ([The07; Neg07]) bis zu fahrdynamischen Untersuchungen ([Col+07]). Aufgrund der Spezifik der Aufgabenstellung sind alle etablierten Systeme als monolithische Spezialanwendungen konzipiert ([OCS03; Slo08]). Allgemeine Anforderungen hinsichtlich der technischen Parameter von Fahrsimulatoren sind dokumentiert ([Neg07; Wan+07]). Betrachtungen zur Softwaretechnologie sowie die Definition allgemeingültiger Schnittstellen und Bibliotheken sind nicht Gegenstand der vorliegenden Arbeiten.

In dieser Arbeit wird eine Softwarearchitektur für die interaktive Simulation mobiler Arbeitsmaschinen in virtuellen Umgebungen (SARTURIS) definiert. Dabei werden sowohl die klassischen Aspekte der Simulation (Modellbildung und numerische Berechnung) und VR-Systeme (Visualisierung, Tracking, Ein- und Ausgabe) als auch die zusätzlichen Anforderungen aus der Kombination dieser beiden Technologien betrachtet.
Die Analyse der gegebenen Problemdomäne führt zu einer Verteilung der Aufgabenstellung in klar abgegrenzte Bereiche auf den Gebieten der Mathematik, Physik und Informatik sowie zur Definition intuitiver Schnittstellen. Dadurch lässt sich die komplexe Problemstellung mit Hilfe einer Komponentenarchitektur abbilden. Diese ermöglicht die getrennte Bearbeitung der Teilaufgaben durch entsprechende Experten auf dem jeweiligen Gebiet.
Die Komponenten und Schnittstellen bilden ein Framework. Dieses kann durch Hinzufügen von Komponenten leicht erweitert werden, ohne bestehende Elemente zu beeinflussen. Ein wesentliches Merkmal des Frameworks ist die Trennung der

Implementation der Komponenten und ihrer Verwendung für konkrete Simulationsanwendungsfälle. Diese werden durch Konfigurationen des Komponentensystems in XML oder einer Skriptsprache wie PYTHON beschrieben.
Im Rahmen der Referenzimplementation von SARTURIS wurden sowohl Komponenten für die Modellbildung und Simulationen mobiler Arbeitsmaschinen entwickelt als auch Komponenten für deren Anwendung in verschiedenen VR-Systemen. Dabei wurden folgende Aspekte berücksichtigt:

- Modellbildung technischer Systeme
- Transformation der Gleichungen in Quellcode
- Numerische Berechnung der Modelle
- Visualisierung, Audio und Bewegungssteuerung
- Synchronisation, Kommunikation und Echtzeit
- Bedienelemente und Bussysteme
- Grafische Benutzeroberflächen.

Die Komponenten sind in C++ unter Verwendung frei verfügbarer Bibliotheken und Standards implementiert. Dadurch wird Plattformneutralität auf der Quellcodeebene erreicht. Bei der Implementation von Komponenten werden deren Schnittstellen durch eine Spezifikation im XML-Format beschrieben. Durch automatisierte Abläufe während des Übersetzens der Komponente werden Codefragmente zur typsicheren Integration in das Framework und zur Konfiguration der Komponente generiert. Dadurch wird der Aufwand für die Erstellung einer Komponente im Wesentlichen auf die Implementation der Funktionalität in C++ und die Spezifikation der Schnittstellen in XML begrenzt.
Im Allgemeinen ist es für die Anwendung von SARTURIS nicht notwendig C++ zu nutzen. Die Anpassung der Simulationsanwendung an die konkrete Aufgabenstellung erfolgt in der Konfigurationsebene unter Anwendung von XML bzw. PYTHON.
Als Referenzanwendung für das Softwaresystem wurde die Simulation eines Radladers in einem interaktiven Fahrsimulator (Abbildung 1.3) realisiert. Diese Referenzanwendung verknüpft das vorhandene Framework mit der speziellen Komponente zur Simulation des Radladers, welche durch Codeexport aus einem Computer-Algebra-System erstellt wird. Die Darstellung des Radladers und der virtuellen Umgebung

im VR-System (Visualisierung und Audio) wird durch die vorhandenen Standardkomponenten sowie deren Konfiguration realisiert.
Die Interaktion zwischen dem Bediener und dem virtuellen Radlader erfolgt durch eine Originalkabine mit den entsprechenden Stellteilen und Anzeigen. Diese werden ebenfalls durch Standardkomponenten angesteuert und durch die Konfiguration mit dem Radladermodell verbunden.

Abbildung 1.3: Interaktiver Simulator mit der Fahrsimulation eines Radladers

Durch den komponentenbasierten Ansatz wird das Softwaresystem in übersichtliche Teilbereiche gegliedert. Dadurch wird der Grad der Wiederverwendung einzelner Codefragmente deutlich erhöht. Die Fehleranfälligkeit wird reduziert und die Wartung des Codes vereinfacht. Zusätzlich ist die Möglichkeit der partiellen Erweiterung des Komponentensystems ein wesentlicher Vorteil. Alternative Methoden der Modellbildung und Simulation ([Fre+09; Fre10; SBK10; SNK11]) können bei der Erstellung von Komponenten ebenso eingesetzt werden wie erweiterte Visualisierungsbibliotheken ([Gro08; GW11]) oder neue Kommunikationstechnologien. Die Nutzung der Refe-

renzanwendung für Untersuchungen auf dem Gebiet der Mensch-Maschine-Interaktion dokumentiert die Leistungsfähigkeit des Ansatzes ([Bür+11; Hos+12]).

Die vorliegende Arbeit ist in 8 Kapitel gegliedert. Nach der Diskussion der Grundlagen auf den Gebieten der numerischen Simulation mobiler Arbeitsmaschinen und der virtuellen Realität (Kapitel 2) erfolgt die Ableitung der entsprechenden Softwarearchitektur (Kapitel 3). Anschließend werden die Realisierung durch ein komponentenbasiertes Framework (Kapitel 4) und eine Referenzanwendung (Kapitel 5) beschrieben. Aus der Anwendung des Frameworks resultiert die Forderung nach einem leistungsfähigen Konfigurationssystem. Dieses wird durch den Einsatz von Skriptsprachen realisiert (Kapitel 6). Nach der Diskussion der interaktiven Simulation im Prozess der Produktentwicklung (Kapitel 7) folgt die Zusammenfassung und der Abschluss der Arbeit (Kapitel 8).

# 2 Grundlagen der interaktiven Simulation in virtuellen Umgebungen

## 2.1 Simulation technischer Systeme

Die Aufgabenstellung bei der Simulation technischer Systeme ist unterteilt in die Modellbildung sowie deren Berechnung. Unter Modellbildung wird die Abstraktion des technischen Systems in berechenbare Gleichungen verstanden. Abhängig von der Fragestellung und den zu untersuchenden Effekten müssen zu einem technischen System unterschiedliche Modelle formuliert werden. Je nach Ansatz entstehen Systeme aus algebraischen Gleichungen und gewöhnlichen oder partiellen Differentialgleichungen ([Bos94; CK06]). Die Berechnung dieser Gleichungen erfolgt im Allgemeinen durch numerische Methoden, da analytische Lösungen der Modellgleichungen aufgrund ihres Charakters nicht vorliegen.
Die Menge an notwendigen Rechenoperationen erfordert die Berechnung auf einem Automaten. Als Automat wird im Kontext dieser Arbeit ein Softwaresystem verstanden, welches in der Lage ist, die Simulation durchzuführen. Diese allgemeine Einordnung umfasst sowohl spezielle Simulationssoftware als auch mathematisch-technische Software, wie z. B. MATLAB oder MAPLE. Im einfachsten Fall kann ein Computer mit Betriebssystem als Automat benutzt werden, wenn die Simulation durch Codeexport in ein eigenständiges Programm transformiert wurde.
Die Anwendung des intuitiven Algorithmenbegriffs erlaubt die Strukturierung von Modellbildung und Berechnung im Sinne einer Abbildung in Software. Unter einem Algorithmus wird eine allgemeine Lösungsvorschrift verstanden, die auf eine bestimmte Problemklasse angewendet werden kann. Diese Vorschrift muss notierbar und auf einem Automaten ausführbar sein. Dies setzt voraus, dass der Automat die gegebene Notation interpretieren kann.

Im Sinne des intuitiven Algorithmenbegriffs ist ein Modell die Menge an Gleichungen, die notwendig ist, um die in der Simulation abzubildenden Effekte zu beschreiben. Dabei ist es unerheblich, mit welchen Methoden die Modellgleichungen gewonnen werden. Als Notation sind sowohl mathematische Formulierungen in Gleichungsform als auch Programmiersprachen geeignet. Kommerzielle Simulationsprogramme verwenden in der Regel grafische Notationen und eigene Formate für die Modellbeschreibung.
Numerische Verfahren beschreiben ebenfalls eine Lösungsvorschrift. Die zugehörige Problemklasse wird durch das zu berechnende System von Gleichungen und Differentialgleichungen definiert. Typische Problemklassen sind Anfangswertaufgaben oder Randwertprobleme. Die Notation erfolgt analog zu den Modellgleichungen in Gleichungsform bzw. Programmiersprachen.
Die allgemeine Anwendbarkeit numerischer Verfahren führt zur Abstraktion des Modells in eine Gleichung $g(x)$. Das numerische Verfahren ist eine Funktion $f\left(g(x)\right)$. Die Durchführung einer Simulation entspricht dann der Berechnung der Funktion $f$ mit dem Simulationssystem. Diese erfordert die Übersetzung von Modell und numerischer Methode aus der gegebenen Notation in eine durch den verwendeten Automaten gegebene Sprache.
Aufgrund der Vielfalt der Verwendung des Automaten-Begriffs wird im Weiteren der Begriff Simulationssystem verwendet. Dieser fasst die zur Durchführung der Berechnung notwendige Hard- und Software zusammen.
Die Interaktion von Modell und Löser lässt sich am Beispiel eines Zweimassenschwingers (Abbildung 2.1) darstellen. Die Modellbildung mit den grundlegenden Methoden der Maschinendynamik führt auf eine Anfangswertaufgabe (Gleichung 2.1).

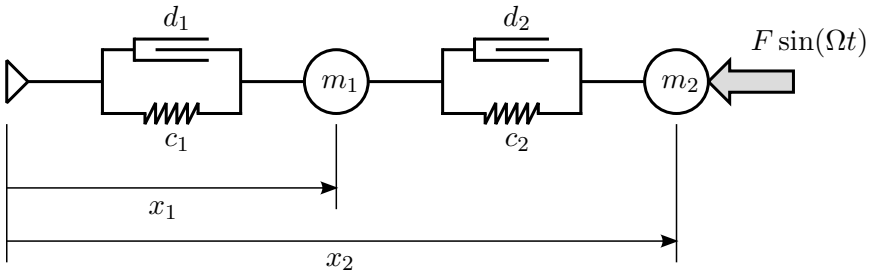


Abbildung 2.1: Zweimassenschwinger

$$\begin{aligned} \dot{x} &= f(x,t) \\ x(t=0) &= x_0 \end{aligned} \tag{2.1}$$

Das Modell ist durch ein System gewöhnlicher Differentialgleichungen beschrieben (Gleichung 2.2). Als Löser eignet sich z. B. das RUNGE-KUTTA-Verfahren 4. Ordnung. Zur Ausführung auf einem einfachen Simulationssystem (Standardcomputer mit Betriebssystem) werden sowohl die Modellgleichungen als auch der Löser in einem C++-Code (Abbildung 2.2 und 2.3) notiert. Bei der Implementation des Lösers ist die Abstraktion des Modells durch die anonyme Funktion $f(x,t)$ zu erkennen. Der Code für die Berechnung der Gleichungen des Zweimassenschwingers zeigt den hohen Abstraktionsgrad bei der Notation der Modelle in C++-Quellcode.

$$\begin{aligned}
\dot{x}_1 &= v_1 \\
\dot{x}_2 &= v_2 \\
\dot{v}_1 &= \frac{d_2\left(v_2 - v_1\right) + c_2\left(x_2 - x_1\right) - c_1 x_1 - d_1 v_1}{m_1} \\
\dot{v}_2 &= \frac{-F\left(t\right) + d_2\left(v_1 - v_2\right) + c_2\left(x_1 - x_2\right)}{m_2}
\end{aligned} \tag{2.2}$$

```
1 Vector f(const Vector& x, double t)
2 {
3   Vector y(4);
4   y(0) = x(2);
5   y(1) = x(3);
6   y(2) = (d2*(x(3)-x(2))+c2*(x(1)-x(0))-c1*x(0)-d1*x(2))/m1;
7   y(3) = (-F*sin(om*t)+d2*(x(2)-x(3))+c2*(x(0)-x(1)))/m2;
8   return y;
9 }
```

Abbildung 2.2: Implementation des Zweimassenschwingers

```
1 void step(Vector& x, double& t, double h)
2 {
3   Vector k1 = h * f(x       , t       );
4   Vector k2 = h * f(x+0.5*k1, t+0.5*h);
5   Vector k3 = h * f(x+0.5*k2, t+0.5*h);
6   Vector k4 = h * f(x+k3    , t+h     );
7   x = x+(k1+2*k2+2*k3+k4)/6.0;
8   t+=stepwidth;
9 }
```

Abbildung 2.3: Implementation des Runge-Kutta-Verfahrens 4. Ordnung

Die Trennung von Modell und Löser (Abbildung 2.4) bildet die Grundlage für ein modulares Simulationssystem. Die Darstellung am Beispiel des Zweimassenschwingers zeigt, dass dadurch die Modellbildung von der eigentlichen Berechnung getrennt werden kann. Das Simulationssystem benötigt nur Zugriff auf den Löser, um die Berechnung durchzuführen. Das Modell selbst ist für das Simulationssystem transparent. Die Menge an numerischen Methoden ist prinzipiell erweiterbar und kann an die Anforderungen neuer Problemklassen angepasst werden.
Die Darstellung am Beispiel von Systemen gewöhnlicher Differentialgleichungen schränkt die Aussagen hinsichtlich der Trennung von Modell und Löser nicht ein. Für die Abbildung und Berechnung von Modellen, welche nicht durch gewöhnliche Differentialgleichungen beschrieben werden, lassen sich gleichwertige Beziehungen formulieren. Die dabei entstehenden Schnittstellen und Interaktionsmuster sind komplexer als bei gewöhnlichen Differentialgleichungen.

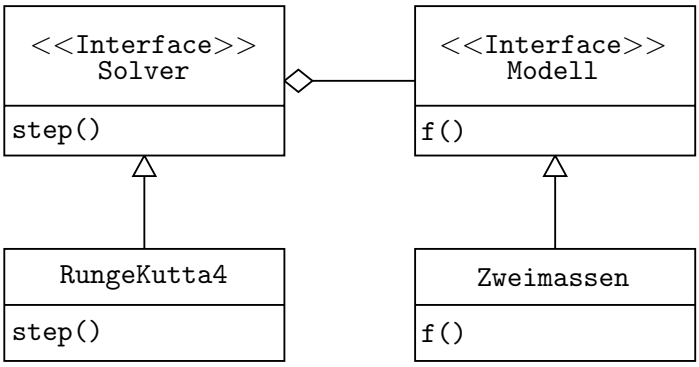


Abbildung 2.4: Beziehung zwischen Modell und Löser

Die Notation der Algorithmen für die numerischen Verfahren und die Modelle in einer standardisierten Programmiersprache reduziert die Anforderungen an das Simulationssystem. Dieses besteht im grundlegenden Fall aus der entsprechenden Hardware und einem Betriebssystem. Weiterhin ist die Formulierung der Modelle nicht an spezifische Werkzeuge gebunden. Prinzipiell eignet sich jedes Werkzeug bzw. jede Werkzeugkette, welche die mathematischen Zusammenhänge der Modellbildung in einen Quellcode transformieren kann. In diesem Zusammenhang haben sich Codegeneratoren etabliert. Diese ermöglichen die Notation der Modelle in einer weniger abstrakten Form.

## 2.2 Modelle mobiler Arbeitsmaschinen

### 2.2.1 Mechanische Strukturen

Je nach Aufgabenstellung lassen sich Modelle für mechanische Strukturen in zwei Gruppen einteilen. Für die Analyse des Tragverhaltens einer Struktur spielt die Verteilung der eingeleiteten Lasten eine entscheidende Rolle. Die Berechnung erfolgt aufgrund lokaler und globaler statischer Unbestimmtheiten häufig über einen Verformungsansatz mit Rand- und Übergangsbedingungen. Eine Verallgemeinerung dieses Ansatzes wird in der Methode der finiten Elemente realisiert. Entsprechende Modelle sind notwendig, wenn lokale Effekte wie Spannungen und Verformungen oder durch die Masseverteilung der Struktur bedingte Schwingungen berechnet werden sollen. Der Aufwand für die Modellbildung und Berechnung ist sehr hoch, so dass dieser Ansatz im Rahmen einer ganzheitlichen Systemsimulation eher ungeeignet erscheint. Wesentlich geeigneter ist die Methode der Mehr- bzw. Starrkörpersimulation. Das mechanische System wird in eine Menge aus starren Körpern und Koppelelementen zerlegt. Diese Koppelelemente können Randbedingungen (Gelenke) oder Übertragungsglieder (Aktuatoren) abbilden. Die Effekte der Verformung großer und schlanker Strukturen, die bei mobilen Arbeitsmaschinen auftreten, können durch Ansätze flexibler Mehrkörpersysteme ([GDI04; Jan10; Sch99; SW99; WN03]) hinreichend genau abgebildet werden.
Für die Beschreibung von Mehrkörpersystemen existieren zahlreiche etablierte Methoden ([Arn+11; EF98; FO00; Fea08; RS10]). Diese führen entweder auf Systeme gewöhnlicher Differentialgleichungen (Gleichung 2.3) oder auf differential-algebraische Gleichungen (Gleichung 2.4). Die wesentlichen Unterschiede bei der Modellbildung existieren im Umgang mit geschlossenen kinematischen Schleifen. Diese sind bei mobilen Arbeitsmaschinen Standard für die Übertragung von Bewegungen unter Last (Arbeitsausrüstungen).

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, t) \tag{2.3}$$

$$\begin{aligned} \dot{\mathbf{y}} &= \mathbf{f}(\mathbf{y}, \mathbf{z}, t) \\ 0 &= \mathbf{g}(\mathbf{y}, \mathbf{z}, t) \end{aligned} \tag{2.4}$$

Ansätze, welche ausschließlich gewöhnliche Differentialgleichungen beschreiben, behandeln die kinematischen Schleifen als Mechanismen und benutzen deren Übertragungsfunktionen für die Berechnung von Energieanteilen bzw. Schnittkräften.

Nachteilig bei dieser Methode ist die notwendige kinematische Analyse, die im Allgemeinen nicht durch einen Rechner erfolgen kann. Der wesentliche Vorteil ergibt sich durch das System gewöhnlicher Differentialgleichungen, welches sehr robust ist und durch vergleichsweise einfache numerische Verfahren gelöst werden kann.
Andere Methoden beschreiben Mehrkörpersysteme mit Hilfe einfacher Differentialgleichungen für Baumstrukturen und zusätzlicher algebraischer Gleichungen für die Zwangsbedingungen bei geschlossenen Schleifen. Dieser Ansatz ist automatisierbar, d. h. eine analytische Ermittlung der kinematischen Beziehungen der mobilen Arbeitsmaschine ist nicht erforderlich. Im Ergebnis entsteht ein System algebraischer Differentialgleichungen mit dem Index 3. Die Schwierigkeiten bestehen in der Indexreduktion ([Arn98; EF98; Füh88; HW10; Han91; SFR91]) bzw. Stabilisierung der numerischen Integration ([ACR93; Bau72]).

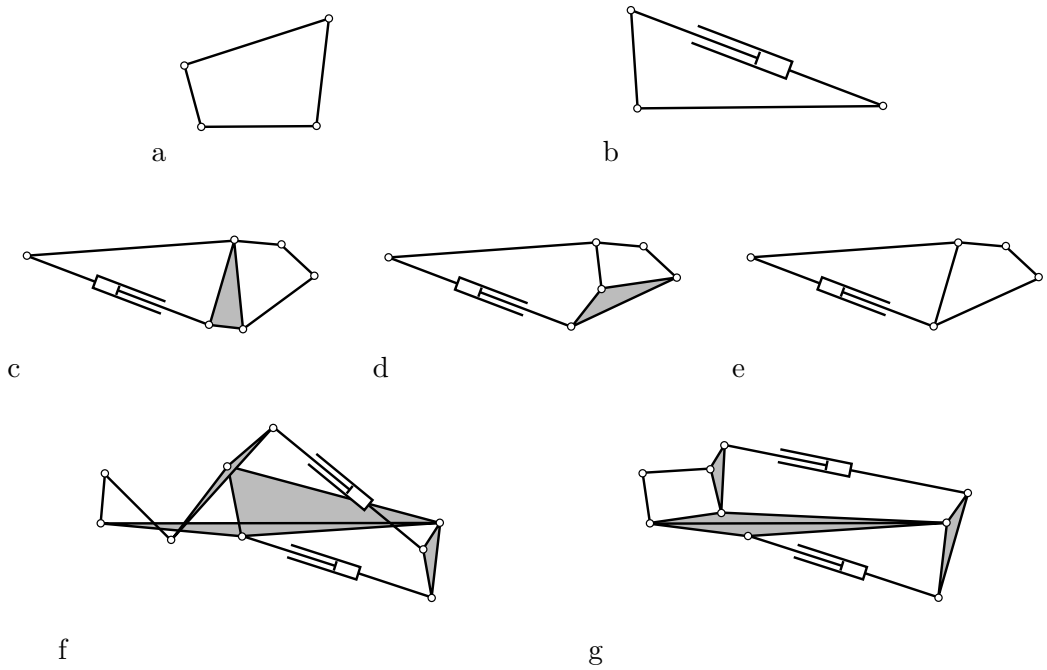


Abbildung 2.5: Typische elementare Mechanismen mobiler Arbeitsmaschinen

Eine Analyse der gängigen Konzepte für Arbeitsausrüstungen mobiler Arbeitsmaschinen zeigt, dass diese häufig aus der Kombination elementarer ebener Mechanismen (Abbildung 2.5) aufgebaut sind. Räumliche Bewegungen werden durch die entsprechende Anordnung unterschiedlicher Mechanismenebenen ermöglicht.

Bei Baggern, Ladekranen und Betonverteilermasten werden in der Regel vier- bzw. sechsgliedrige Übertragungsgetriebe mit einem Hydraulikzylinder als Antrieb angewendet (Abbildung 2.5a - e). Für Parallelführungen bei Lademaschinen werden neungliedrige Mechanismen mit dem Freiheitsgrad 2 eingesetzt (Abbildung 2.5f und Abbildung 2.5g). Für diese Mechanismen ist eine kinematische Analyse durchführbar. Mit Hilfe einer entsprechenden Mechanismenbibliothek kann die Beschreibung der Starrkörpermodelle mobiler Arbeitsmaschinen deutlich vereinfacht werden.
Beispiele neuer Entwicklungen wie die Adaption einer Stewart-Plattform an mobile Arbeitsmaschinen ([Fed04; Rud05]) zeigen, dass auch die kinematische Analyse dieser Mechanismen möglich ist und im Rahmen der Entwicklung für Aufgaben der Mechanismensynthese auch durchgeführt wird. Für die Formulierung der entsprechenden Starrkörpermodelle sind diese Ergebnisse direkt verwendbar.
Für die Berechnungsvorschrift können Methoden ([HHS97; SBK10]) angewendet werden, welche Systeme gewöhnlicher Differentialgleichungen erzeugen (Gleichung 2.5).

$$\mathbf{H}\left(\mathbf{q}\left(t\right)\right)\ddot{\mathbf{q}}(t)+\mathbf{q}_c\left(\mathbf{q}\left(t\right),\dot{\mathbf{q}}\left(t\right)\right)+\mathbf{q}_g\left(\mathbf{q}\left(t\right)\right)=\mathbf{q}_f\left(\mathbf{q}\left(t\right),\dot{\mathbf{q}}\left(t\right)\right) \tag{2.5}$$

Das Prinzip mittels Übertragungsfunktionen geschlossene Schleifen aufzulösen, kann am Beispiel des viergliedrigen Mechanismus mit Hydraulikzylinder demonstriert werden (Abbildung 2.5b und Abbildung 2.6). Der Zylinder (Körper 2 und 3) bildet mit dem Gestell (Körper 0) und dem Abtrieb (Körper 1) eine geschlossene Schleife. Als generalisierte Koordinate wird der Winkel $\phi$ zwischen Gestell und Abtrieb definiert. Durch die Berechnung des Winkels $\psi$ (Gleichung 2.6) wird die koaxiale Bedingung zwischen Zylinderrohr und Kolben erfüllt. Die Koordinatentransformation für das Zylinderrohr erfolgt dabei im Punkt $B$ relativ zum Gestell. Die Transformation des Kolbens erfolgt relativ zum Abtrieb im Punkt $C$.

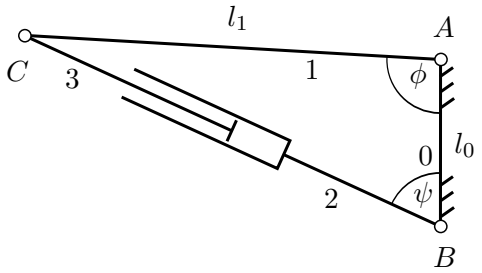


Abbildung 2.6: Viergliedriger Mechanismus mit kinematischer Schleife

$$\psi = \arcsin\left(\frac{l_1 \sin\phi}{\sqrt{l_1^2 + l_0^2 - 2 l_1 l_0 \cos\phi}}\right) \tag{2.6}$$

Diese Vorgehensweise ist prinzipiell für alle elementaren Mechanismen mobiler Arbeitsmaschinen (Abbildung 2.5) anwendbar. Dabei kann die Komplexität der resultierenden Gleichungen wesentlich durch die Wahl der generalisierten Koordinaten beeinflusst werden.

Die durch die kinematischen Beziehungen verbundenen Massen beschreiben nur einen Teil des Starrkörpermodells ($\mathbf{H}$, $\mathbf{q}_c$ und $\mathbf{q}_g$ in Gleichung 2.5). Der wesentliche zweite Teil wird durch Aktuatoren und Koppelelemente (z. B. Federn und Dämpfer) abgebildet. Diese beschreiben die inneren und äußeren Kräfte des Modells ($\mathbf{q}_f$ in Gleichung 2.5).

Häufig wird dabei auf Elemente mit einer linearen Formulierung des Zusammenhangs zwischen Kraft und Weg bzw. Geschwindigkeit zurückgegriffen. Für komplexere Elemente, wie z. B. Kabinenlagerungen oder hydro-pneumatische Federungen, sind entsprechende nichtlineare Gleichungen zu formulieren. Diese verursachen im Allgemeinen einen höheren Aufwand bei der Modellbildung und numerischen Berechnung, welcher durch eine entsprechend höhere Ergebnisgüte gerechtfertigt werden kann.

Komplexe Modelle für Aktuatoren besitzen eigene Zustandsgrößen und werden in der gegebenen Systematik den Modellen für Antriebe zugeordnet. Die Repräsentation der Aktuatoren durch Kraftelemente bildet die Schnittstelle zwischen dem Starrkörpermodell und dem Antriebsmodell. Die Modelle für Arbeitsprozesse werden analog zu den Aktuatoren durch äußere Kraftelemente mit dem mechanischen Modell gekoppelt.

### 2.2.2 Modelle für Antriebe

Mobile Arbeitsmaschinen werden in der Regel durch hydraulische Systeme angetrieben. Als Primärenergiequelle dient ein Dieselmotor. Die elektrische Übertragung der Leistung vom Motor zu den einzelnen Verbrauchern wurde bisher nicht realisiert, da vor allem auf dem Gebiet der Linearaktuatoren keine geeigneten elektrischen Komponenten für die geforderten Leistungen und Energiedichten verfügbar sind. Durch die Bestrebungen, Energie vor allem aus Hubbewegungen zurückzugewinnen und dadurch die Effizienz der mobilen Arbeitsmaschine zu steigern, rücken elektrische Antriebe verstärkt in den Fokus ([Göt11; Moh11; Vah09; Tuu09]). Brauchbare Ergebnisse werden aber vor allem auf dem Gebiet der hydraulisch-mechanischen Leistungsübertragung erziehlt ([TG07; Win06; Win09]).

Zur Modellierung hydraulischer Antriebe sowie deren Steuerungen und Regelungen gelten die Druckaufbaugleichung (Gleichung 2.7) und die Blendengleichung (Gleichung 2.8). Mit Hilfe dieser Grundgleichungen lassen sich hydraulische Netzwerke modellieren ([Bea99; BO03; Sto+07; Wat08; WSG11]). Dabei beschreibt die Druckaufbaugleichung wie sich der Druckzustand in Abhängigkeit des Ölvolumens verändert (Abbildung 2.7a). Die Blendengleichung beschreibt den ausgleichenden Volumenstrom infolge eines Druckunterschiedes benachbarter Elemente (Abbildung 2.7b).


a b

Abbildung 2.7: Modell für Ölvolumen

$$\dot{p} = \dot{V} \cdot \frac{K}{V} \tag{2.7}$$

$$Q = \alpha \cdot \operatorname{sign}(\Delta p) \sqrt{|\Delta p|} \tag{2.8}$$

Analog zu den Starrkörpermodellen werden mit dieser Methode der Modellbildung Energie und Impuls gleichmäßig über das diskretisierte Ölvolumen verteilt. Lokale Effekte, wie z. B. Kavitation, lassen sich durch die beschriebenen Modelle hydraulischer Netzwerke nicht abbilden. Die dafür notwendigen CFD-Modelle führen auf Systeme partieller Differentialgleichungen mit entsprechendem Berechnungsaufwand. Der Fokus der interaktiven Simulation in virtuellen Umgebungen liegt auf Modellen, die das Systemverhalten der mobilen Arbeitsmaschine mit der durch die Aufgabenstellung geforderten Genauigkeit und Abbildungstiefe beschreiben. Im Allgemeinen genügen Modelle hydraulischer Netzwerke, die mit Hilfe der Druckaufbau- und Blendengleichung beschrieben sind, diesen Anforderungen.
Die Grundgleichungen hydraulischer Netzwerke sind nichtlinear und bilden steife Systeme gewöhnlicher Differentialgleichungen. Aufgrund des Netzwerkcharakters interagieren nur benachbarte Elemente, was die Komplexität des resultierenden Gleichungssystems deutlich reduziert. Der numerische Aufwand bei der Berechnung entsteht im Wesentlichen durch die steifen Differentialgleichungssysteme ([BP00; HW10; SB05]).

Die Verbindung zum Modell für mechanische Strukturen erfolgt an den Aktuatoren. Diese bilden sowohl Gleichungen für die Bewegung des Aktuators infolge der Käfte- und Momentenbilanz als auch für die zugehörigen Systemdrücke und Volumenströme. Dies soll am Beispiel eines Hydraulikzylinders (Abbildung 2.8) dargestellt werden. Modelliert werden die Ölvolumen auf der Kolbenseite ($V_1$) und Stangenseite ($V_2$) durch die Druckaufbaugleichung. Die Leckage wird durch die Blendengleichung zwischen $V_1$ und $V_2$ abgebildet. Die Beschleunigung der Masse des Kolbens $m_K$ mit einer Bewegungsgleichung entspricht einem trivialen Starrkörpersystem. Es ist leicht vorstellbar, dass anstelle der Bewegungsgleichung des Kolbens eine deutlich komplexere Berechnungsvorschrift mit entsprechend reduzierten und nicht konstanten Massen eingesetzt werden kann. Das gemeinsame Modell beschreibt ein System aus vier gewöhnlichen Differentialgleichungen (Gleichung 2.9) und kann mit den beschriebenen Methoden formuliert und berechnet werden (siehe Abschnitt 2.1).

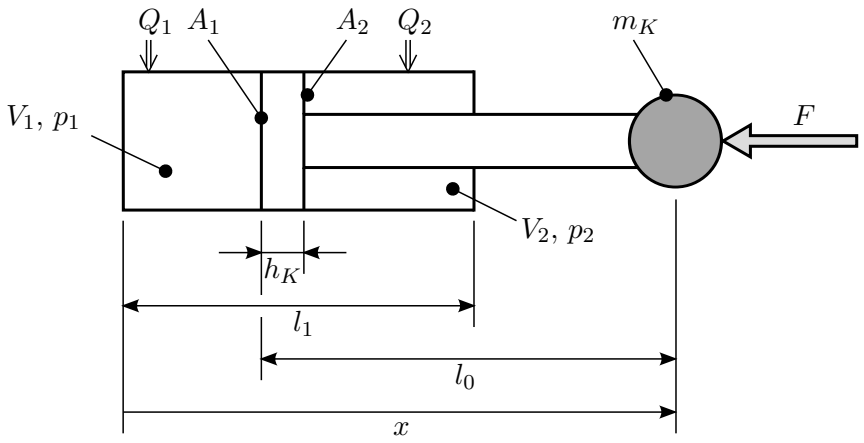


Abbildung 2.8: Modell für einen Hydraulikzylinder

$$
\begin{aligned}
\mathbf{z} &= [x, \dot{x}, p_1, p_2]^{\mathrm{T}} \\
\dot{z}_1 &= z_2 \\
\dot{z}_2 &= \frac{(z_3 A_1 - z_4 A_2 - F)}{m_K} \\
\dot{z}_3 &= \frac{Q_1 - Q_{12} - A_1 z_2}{A_1 (z_1 - l_0)} \\
\dot{z}_4 &= \frac{Q_{12} - Q_2 - A_2 z_2}{A_2 (l_1 + l_0 - h_K - z_1)} \\
Q_{12} &= \alpha_{12} \cdot \operatorname{sign}(z_3 - z_4) \sqrt{|z_3 - z_4|}
\end{aligned}
\tag{2.9}
$$

Die Verallgemeinerung der Druckaufbau- und Blendengleichung führt auf die Abbildung von Potenzial- und Flussgrößen ([EM97; FE98]). Die Anwendung dieses Konzeptes auf die elektischen Größen Strom und Spannung ermöglicht die Abbildung elektrischer Netzwerke in Antriebsmodellen sowie die Verbindung zu mechanischen Strukturen an elektrischen Aktuatoren. Die Sprache MODELICA, welche im Bereich der ganzheitlichen Simulation technischer Systeme einen Standard darstellt, basiert im Wesentlichen auf diesem Konzept ([Fri04; Fri11]).

### 2.2.3 Bodenkontakt-Modelle

Der Kontakt zwischen mobilen Arbeitsmaschinen und dem Boden erfolgt im Bereich der Fahrwerke, Abstützungen und Werkzeuge (bei Erdbaumaschinen). Der Kontakt im Bereich der Werkzeuge wird speziell durch das Prozessmodell beschrieben (siehe Abschnitt 2.2.4). In diesem Abschnitt wird der für die Berechnung der Fahrdynamik und Standsicherheit elementare Bodenkontakt im Bereich der Fahrwerke und Abstützungen behandelt. Dieser kann durch drei wesentlichen Systeme charakterisiert werden:

- **Radfahrwerke ohne Abstützung:** Die Ortsveränderung der Maschine ist wesentlicher Bestandteil der Arbeitsaufgabe (z. B. bei Radladern). Der Kontakt zum Boden wird durch die Reifen hergestellt. Fahrdynamik und Fahrstabilität werden wesentlich durch die Interaktion von Reifen und Boden bestimmt.
- **Radfahrwerke mit Abstützung:** Die Ortsveränderung ist nicht unmittelbar Bestandteil der Arbeitsaufgabe (z. B. bei Mobilbaggern). Während der Ausführung der Arbeitsaufgabe wird die Maschine auf den Abstützelementen gelagert. Für die Standsicherheit ist die Abstützgeometrie sowie deren Interaktion mit dem Boden von Bedeutung. Während des Fahrbetriebes definieren die Reifen die fahrdynamischen Eigenschaften.
- **Kettenlaufwerke:** Die Ortsveränderung ist sekundäre Aufgabe der Arbeitsmaschine (z. B. bei Raupenbaggern). Die Analyse der Fahrdynamik und Fahrstabilität ist in der Regel nicht Gegenstand der Maschinenentwicklung. Die Standsicherheit und die dynamischen Reaktionen der Maschine während der Arbeitsaufgabe werden wesentlich durch die Geometrie der Laufwerke und den Boden bestimmt.

Die aus diesen vielfältigen Gestaltungsmöglichkeiten für Fahrantriebe resultierende Komplexität bei der Abbildung der Interaktion mit dem Boden ist nicht Gegenstand

dieser Arbeit. Im Vordergrund stehen Maschinen mit gummibereiften Radfahrwerken. Für die Abbildung von Abstützungen und Kettenlaufwerken sei auf die Literatur verwiesen ([KGJ09; Sch94]).
Radfahrwerke mobiler Arbeitsmaschinen verfügen in der Regel nicht über zusätzliche Feder- und Dämpferelemente. Die Reifen tragen die komplette ungefederte Masse einschließlich der Nutzlast und haben somit einen signifikanten Einfluss auf die Fahrstabilität und den Fahrkomfort ([Fer09; Har02; HB03]). Die aus der Kraftfahrzeugtechnik bekannten Reifenmodelle ([Gip99; HRW03; HRW07; PB93; Pac07; Ril05; RC07]) sind aufgrund der Unterschiede in den Radkräften, Umfangsgeschwindigkeiten und Radgrößen nicht ohne Weiteres auf mobile Arbeitsmaschinen übertragbar. Geeignete Modelle für Offroadfahrten befinden sich in Entwicklung ([Har02; HB03; HKN08; Ruf97]).
Je nach Komplexität des Modells werden Elastizität und Dämpfung der Reifen durch interne Zustandsgrößen des Reifenmodells abgebildet. Profilierung und Bodenbeschaffenheit werden durch entsprechende Parameter bei der Berechnung der Reaktionskräfte berücksichtigt.
Das Zusammenwirken zwischen dem Reifenmodell und den Modellen für den Fahrantrieb sowie die mechanische Struktur soll anhand der verallgemeinerten Darstellung in Abbildung 2.9 gezeigt werden. Unabhängig von der Formulierung des Reifenmodells lässt sich ein Arbeitspunkt $P$ definieren, in dem die Reaktionskräfte angetragen werden. Die Formulierung der Bewegungsgleichungen für das dargestellte System zeigt die Wirkung der Reaktionskräfte durch die Radnabe auf das Gesamtfahrzeug. Die Betrachtung des Rotationsfreiheitsgrades des Rades (Zustandsgrößen $\phi$ und $\dot{\phi}$) zeigt die Rückwirkung auf den Antrieb durch das Momentengleichgewicht um die Radachse.

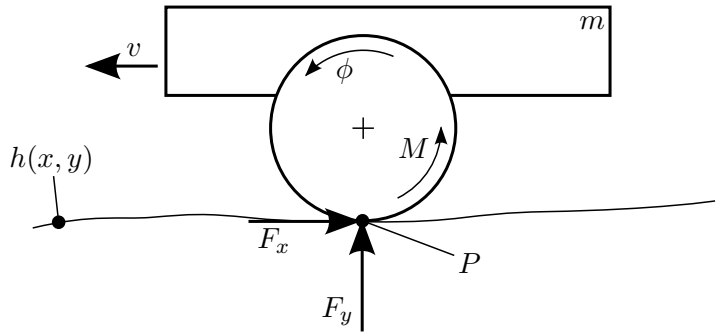


Abbildung 2.9: Verallgemeinerung des Reifenmodells

Neben der echtzeitfähigen Implementation der Gleichungen des Reifenmodells ist die Abfrage des Geländeprofils $h(x, y)$ in der Umgebung des Arbeitspunktes eine wesentliche Aufgabe. Da bei einer interaktiven Simulation keine Annahmen über die Fahrtrajektorie getroffen werden können, ist zu jedem Berechnungsschritt des Reifenmodells die Höhe des Geländes und die Geländeeigenschaften an der jeweiligen Position abzufragen. Die Abfrage muss mit dem in der VR-Umgebung dargestellten Modell korrelieren, andernfalls wird die Simulation für den Bediener unbenutzbar.

### 2.2.4 Prozessmodelle

Aufgrund der Vielfalt der Arbeitsaufgaben mobiler Arbeitsmaschinen sind kaum geeignete Prozessmodelle formuliert. Bekannte Modelle für Arbeitsprozesse sind sehr speziell und bisher nicht allgemeingültig abgeleitet. Sie reichen von analytischen Schnittkraftformeln für die Erdgewinnung ([PPG11; ES00; KGJ09; Mot03; Ree64; Xia08; ZK95]) bis zu komplexen Diskretisierungsverfahren bei der Modellierung von Schüttgutflüssen ([CS02; Coe09; CS79]). Von besonderem Interesse ist die Kopplung der Prozessmodelle mit dem Maschinenmodell ([GKK10; KKG11]). Im Allgemeinen interagieren die Zustandsräume über Kraft- und Momentvektoren, deren Betrag und Richtung sich aus dem Bewegungszustand des Werkzeuges ableitet.
Bei der ganzheitlichen Simulation mobiler Arbeitsmaschinen werden Prozessmodelle bisher nur am Rande behandelt ([BO03; EAB09; ES00; Mot03]). Im Rahmen dieser Arbeit steht das Fahren im Gelände als Arbeitsaufgabe im Fokus. Die interaktive Simulation wird auf Maschinen begrenzt, welche einen wesentlichen Teil der Arbeitsaufgabe mit dem Fahrantrieb verrichten. Die Gültigkeit der Aussagen wird dadurch nicht beeinflusst, da die Untersuchung des Fahrverhaltens mobiler Arbeitsmaschinen Gegenstand aktueller Arbeiten ist ([Böh01; Fer09; HKN08; HB03; Sch06; Ruf97]).

### 2.2.5 Fahrermodelle

Sowohl die Ortsveränderung als auch die Arbeitsaufgabe werden bei mobilen Arbeitsmaschinen nicht automatisiert durchgeführt. In beiden Situationen wird die Maschine durch einen Bediener geführt. Dieser ist aufgrund seiner kognitiven Fähigkeiten und Ausbildung in der Lage, die Maschine mit Hilfe der zur Verfügung stehenden Bedienelemente zu steuern ([Ras86]) . Die dabei entstehenden Systemzustände (Drücke, Kräfte, Geschwindigkeiten, usw.) sind wesentlich vom Zusammenspiel zwischen dem Bediener und der Arbeitsmaschine gekennzeichnet.

Während für die Maschine sowie deren Interaktion mit der Umgebung unterschiedliche Modelle und Modellbildungsmethoden zur Verfügung stehen, ist die Abbildung geeigneter mathematischer Modelle für den Bediener eine aktuelle Problemstellung bei der Simulation mobiler Arbeitsmaschinen und Fahrzeuge ([FEP05; Fil09; PKH01; Sch+05; Voi09]).
Die einfachste Arbeitsaufgabe, das Fahren im Gelände, zeigt im Vergleich zur Fahrzeugsimulation die Komplexität der Problemstellung. Bereits der Spurwechsel auf Autobahnen oder die Anpassung der eigenen Geschwindigkeit an eine spezielle Verkehrssituation sind durch Modelle nur schwer zu beschreiben ([Mac03; Mic85; OC04]). Der dabei zu erfassende Zustandsraum der Bedienelemente aus Lenkwinkel und Gaspedal ist klein gegenüber den komplexen Stellteilen mobiler Arbeitsmaschinen, die zum Erfüllen einer Arbeitsaufgabe notwendig sind ([Zie+02; Züh02]).
Der Zusammenhang aus den kognitiven und sensomotorischen Fähigkeiten des Bedieners, seiner Ausbildung und dem technischen System der mobilen Arbeitsmaschine konnte bisher nicht durch geeignete Modelle beschrieben werden.
An einer realen Maschine aufgezeichnete Bedienhandlungen sind nicht für die Simulation virtueller Prototypen geeignet. Analog zur Beschreibung der Interaktion zwischen einem Fahrer und einem Straßenfahrzeug ([Mic85]) kann die Interaktion zwischen einem Bediener und einer mobilen Arbeitsmaschine durch drei hierarchische Ebenen beschrieben werden (Abbildung 2.10).

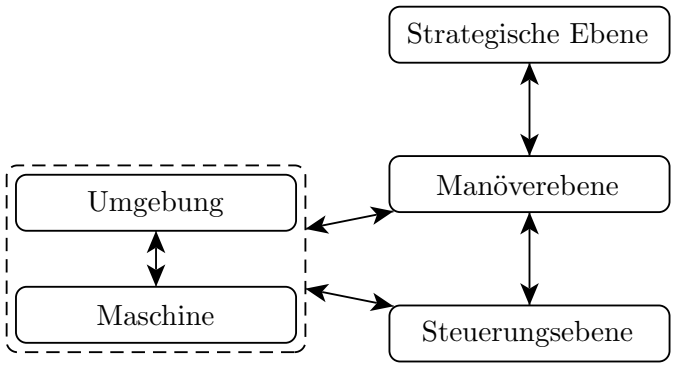


Abbildung 2.10: Hierarchien der Arbeitsaufgabe ([Mic85])

Die generellen Ziele zur Erfüllung der Arbeitsaufgabe werden in der strategischen Ebene beschrieben. Dabei werden vor allem Informationen verarbeitet, die der Planung der Handlungen dienen. Die Erfahrungen über die Maschine und den Prozess fließen in Form technologischer Parameter und Randbedingungen in die Planung ein.

Die Reaktionen des Bedieners auf konkrete Situationen werden in der Manöver- und der Steuerungsebene abgebildet. Dabei repräsentiert die Manöverebene konkrete zur Arbeitsaufgabe gehörende Handlungsmuster. Diese werden durch den Bediener aufgrund seiner Ausbildung und Fähigkeiten kontrolliert ausgeführt. Dabei ist der Bediener mit dem Maschine-Prozess-System verbunden. Die Ausführung der entsprechenden Handlungsmuster wird automatisch an die konkrete Situation angepasst. In der Steuerungsebene werden die Handlungen abgebildet, welche in direkter Rückwirkung auf die Wahrnehmung des aktuellen Zustandes von Maschine und Umgebung ausgeführt werden. Diese Handlungen werden nicht im Kontext von Manövern abgerufen, sondern liegen beim Bediener als automatisierte Handlungsmuster vor.
Bei der Benutzung aufgezeichneter Bedienhandlungen ist die Rückkopplung in der Manöver- und Steuerungsebene nicht gegeben, da die Aufzeichung der Bedienung einer konkreten Maschine keine Schlüsse auf die Reaktionen des Bedieners in einem veränderten technischen System zulässt. Auf der strategischen Ebene können eventuell aus genormten Zyklen (z. B. Y-Zyklus beim Radlader) standardisierte Eingaben für Simulationsmodelle abgeleitet werden. Diese können für vergleichende Aussagen bei der Analyse unterschiedlicher Maschinenkonzepte dienen. Sie sind aber begrenzt, sobald Veränderungen am technischen System Auswirkungen auf die Manöver- und die Steuerungsebene haben. Sehr spezifische Untersuchungen zu Fahrkomfort, Fahrstabilität und Assistenzsystemen betreffen in jedem Fall die Steuerungsebene und können deshalb nicht mit aufgezeichneten Bedienhandlungen durchgeführt werden.
Da sowohl relevante mathematische Modelle für den Bediener fehlen als auch Aufzeichnungen an realen Maschinen ungeeignet für die ganzheitliche Simulation mobiler Arbeitsmaschinen sind, ist der Bediener in die Simulation einzubeziehen. Dies erfolgt durch die Kombination der numerischen Simulation mit einer realitätsnahen Darstellung von Maschine, Umgebung und Arbeitsprozess innerhalb einer *virtuellen Welt*.

## 2.3 Virtuelle Realität und Fahrsimulatoren

### 2.3.1 Mensch-Maschine-Systeme und Informationsverarbeitung

Mobile Arbeitsmaschinen werden zum Zweck der Erfüllung einer Arbeitsaufgabe eingesetzt. Sowohl die reale Maschine als auch deren Abbild in einer virtuellen Umgebung ist das Bindeglied zwischen dem Bediener und dem Arbeitsprozess. Das

Verständnis der Mensch-Maschine-Interaktion ist aus den folgenden Gründen von hoher Bedeutung für die Anwendung der interaktiven Simulation.
Das Fehlen abstrahierter Bedienermodelle erfordert die Integration des Menschen in die Simulation. Die Gültigkeit von Aussagen auf der Basis interaktiv durchgeführter Berechnungen ist wesentlich davon abhängig, ob eine Übertragbarkeit der Bedienhandlung auf eine reale Maschine möglich ist. Diese Übertragbarkeit ist nicht pauschal gegeben, sondern muss im Anschluss an Experimente in virtuellen Umgebungen nachgewiesen werden ([EGD05; KP03]).
Je nach Aufgabenstellung können die Simulationsergebnisse durch objektive Größen bewertet werden, die sich aus den Simulationsmodellen ableiten lassen. Diese Größen sind entweder direkt berechnete Zustandsgrößen (z. B. Kräfte, Drücke, Geschwindigkeiten) oder daraus abgeleitete Werte (z. B. Zykluszeiten, Energiebilanzen). Die Beurteilung der Mensch-Maschine-Schnittstelle erfolgt dagegen hauptsächlich auf der Basis der subjektiven Eindrücke der Probanden ([Cro+98; HB02; KSG06]). Für verlässliche Aussagen im Entwicklungsprozess müssen diese Aussagen durch reproduzierbare Experimente objektiviert werden ([Abe+09; JMS96]). Objektive Messgrößen werden aus dem Fahrverhalten abgeleitet ([BN05; FGH05; SS07]).
Im Kontext der Mensch-Maschine-Interaktion repräsentiert das VR-System selbst eine Maschine. Dabei ist der vereinfachende Charakter der Modellbildung (siehe Abschnitt 2.1) ebenso zu berücksichtigen ([Neg07]) wie die gegebenen physikalischen und technologischen Grenzen der Hard- und Software ([Aki+03; Aki+05; ECE00; Kim05; LPP96; Sha+08]). Anforderungen an VR-Systeme hinsichtlich der Präsentation der virtuellen Umgebung sowie der Genauigkeit und Abbildungstiefe der verwendeten Modelle müssen im Kontext mit den kognitiven und motorischen Fähigkeiten des Menschen definiert werden ([Ant+09; KM08; SPB06; Sta95; SMK98; DKE10; Wal+07]).
Der Informationsfluss im Mensch-Maschine-System ([Bub77]) unterteilt sich in die Phasen der Aufnahme und Verarbeitung der Informationen sowie die anschließende Reaktion ([The07]). Die Informationsaufnahme erfolgt durch die Sinnesorgane des Bedieners. Die Verarbeitung der Informationen findet in den drei Ebenen des wissens-, regel- und fertigkeitsbasierten Handelns ([Ras95]) statt. Dabei vergleicht der Bediener die aufgenommenen Informationen mit Modellen, die eine eigene Erwartungshaltung repräsentieren. Besonders bei trainierten Bedienern ist das fertigkeitsbasierte Handeln auf der Basis einer klaren Erwartungshaltung an die Maschine maßgebend für die Leistungsfähigkeit des Mensch-Maschine-Systems ([End88]).

Die kognitive Schleife der Informationsverarbeitung (Abbildung 2.11) erweitert den Informationsfluss im Mensch-Maschine-System um die Anpassung des Wissens und der Fertigkeiten des Bedieners. Die Erfüllung einer Arbeitsaufgabe führt zu Übungseffekten, welche für den Bediener in Form von Handlungsmustern wieder abrufbar sind ([Str05; Str09]).
Dies hat Konsequenzen sowohl für die Anforderungen an eine virtuelle Welt als auch für die Bewertung der Ergebnisse ([Neg07; The07]). Der Trainingseffekt betrifft vor allem die Vereinfachungen der Modelle und technischen Grenzen der VR-Systeme. Insbesondere geübte Bediener werden Differenzen zwischen den erwarteten Reaktionen der Maschine und der dargestellten Simulation feststellen. Je nach Einfluss dieser Differenz auf die Immersionsfähigkeit des Bedieners ist seine Handlung unter Umständen in Frage zu stellen ([EGD05]).

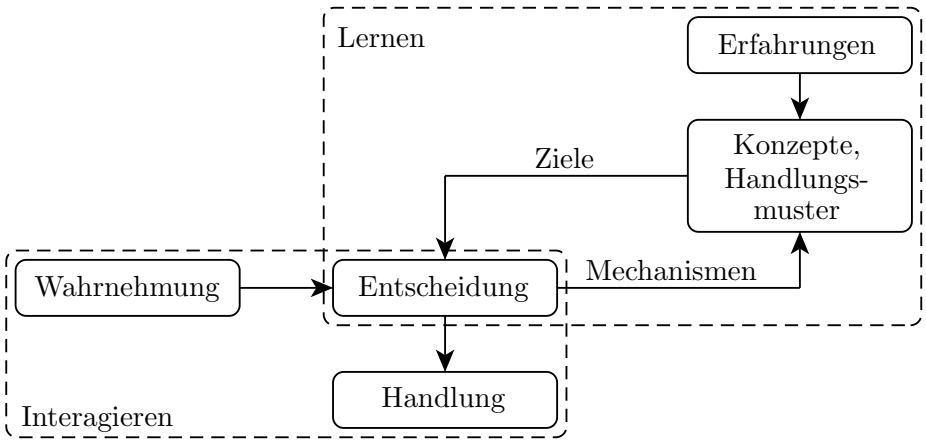


Abbildung 2.11: Kognitive Schleife der Informationsverarbeitung ([Str09])

Der Trainingseffekt findet analog zur realen Welt auch in der virtuellen Umgebung statt. Probanden für Experimente in virtuellen Welten werden immer auch auf den Umgang mit den systematisch gegebenen Unterschieden zur realen Welt trainiert. Eine zu starke Adaption an das VR-System kann dazu führen, dass die beobachtete Bedienhandlung nicht repräsentativ für Aussagen zur realen Welt ist ([HB06]).

### 2.3.2 VR-Systeme

Virtuelle Realität wird als eine künstliche durch Computer simulierte Welt verstanden, in die Personen mit Hilfe von Computern interaktiv eingebunden sind ([Bri09]).

Diese Interaktion ist das wesentliche Ziel des Aufbaus einer virtuellen Umgebung. Die Handlungen der eingebundenen Personen sollen sich von denen in der Realität nicht unterscheiden. Dieser Zustand wird als Immersion bezeichnet. Der erreichte Grad der Immersion ist von den darstellbaren visuellen, akustischen und taktilen Reizen abhängig und für die Sicherheit von Aussagen auf der Basis des Verhaltens der Personen innerhalb der virtuellen Umgebung ausschlaggebend. Hochgradig immersive virtuelle Umgebungen (Abbildung 2.12) können durch den Einsatz interaktiver Virtual-Reality-Systeme erzeugt werden ([Cru+92]).
Die Randbedingungen für den erreichbaren Grad der Immersion werden durch die physiologischen und psychologischen Grenzen des Menschen sowie die technischen Grenzen des VR-Systems definiert ([Neg07]). Für den Aufbau von virtuellen Welten für die interaktive Simulation mobiler Arbeitsmaschinen ist nicht das Erreichen eines vollkommen perfekten Abbildes der Realität das Ziel sondern vielmehr die Akzeptanz der virtuellen Welt durch den Bediener der Maschine ([Ell89; Rie03; Wal+07]).


Abbildung 2.12: Computer Aided Virtual Environment - CAVE[1]

Zur Abbildung interaktiver Simulationen müssen die klassischen VR-Systeme entsprechend erweitert werden. Der Bediener einer VR-Anwendung übernimmt im

[1]Quelle: Lehrstuhl Konstruktionstechnik/CAD, Technische Universität Dresden

Allgemeinen die Funktion eines Navigators (Abbildung 2.13a). Mit Hilfe der Ein- und Ausgabeschnittstellen des VR-Systems manipuliert er seine Position relativ zur virtuellen Welt. Die visuellen Eindrücke sind bereits ausreichend für eine erfolgreiche Navigation, da diese einen hohen Anteil an der menschlichen Wahrnehmung haben ([KM08; Rie03]).
Wird das VR-System durch die interaktive Simulation ergänzt, so ist zwischen dem Navigator, der sich weiterhin mit den klassischen Mechanismen in der virtuellen Welt bewegt und dem Bediener der Maschine zu unterscheiden. Dieser interagiert mit der Simulation durch eigene Schnittstellen (Abbildung 2.13b). In der Regel sind diese durch Originalbedienelemente oder entsprechende Nachbildungen gegeben. Die Navigation innerhalb der virtuellen Welt erfolgt durch die Veränderung der Koordinaten des Maschinenmodells.

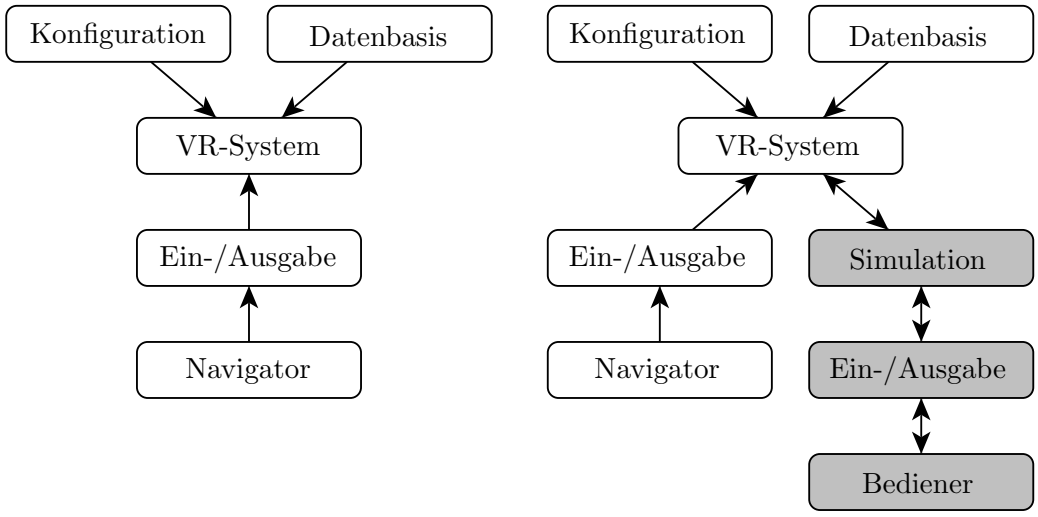


Abbildung 2.13: Navigator und Bediener in VR-Systemen

Für die Benutzer der virtuellen Umgebung ist die Erweiterung durch interaktive Simulationen nicht sichtbar. Auf der Funktionsebene interagieren die Benutzer mit dem System entweder als Navigator oder als Bediener des Maschinenmodells (Abbildung 2.14). Für die Implementation des VR-Systems bedeutet die Erweiterung, dass neben den klassischen Aspekten der Datenbasis, Kommunikation, Synchronisation und Konfiguration die numerische Simulation (siehe Abschnitt 2.1) berücksichtigt werden muss.

Die frühzeitige Einbindung des Bedieners in die Fahrzeugsimulation hat zu einer Etablierung von Fahrsimulatoren (Abbildung 2.15) bei der Fahrzeugentwicklung geführt. Fahrsimulatoren stellen in diesem Zusammenhang eine spezialisierte Form von VR-Systemen dar. Verfügen die Simulatoren über Bewegungssysteme, werden zusätzlich zu den visuellen, akustischen und taktilen Informationen vestibuläre Reize übertragen ([For03; TF01]). Diese sind insofern von Bedeutung, da bei der Fahrsimulation die Navigation des Bedieners innerhalb der virtuellen Welt indirekt über die Steuerung der Fahrzeugbewegung erfolgt. Die generierten Eindrücke über Geschwindigkeiten, Beschleunigungen und Neigungswinkel sind wesentlich für die Immersion des Bedieners ([Kun03; Neg07]).

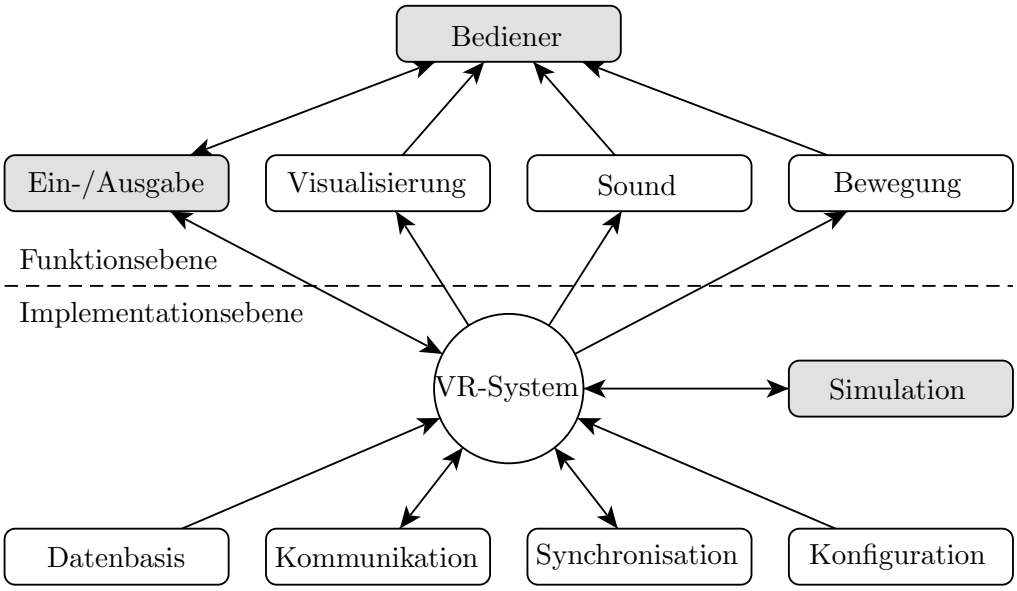


Abbildung 2.14: VR-System mit interaktiver Simulation

Die Anwendungsgebiete von Fahrsimulatoren reichen vom Training über die generelle Analyse des menschlichen Bedienverhaltens und die Bewertung der Mensch-Maschine-Interaktion bis zur Auslegung konkreter technischer Systeme ([Neg07; The07]).
Der Aufbau von Fahrsimulatoren lässt sich verallgemeinern ([Neg07]). Die wesentlichen Elemente der Software sind die Visualisierung und Akustik, die Bewegungssimulation sowie die Simulation eines entsprechenden Szenarios. Die unmittelbare Bedienumgebung aus Anzeigen und Instrumenten wird meistens durch Originalteile bzw. -kabinen dargestellt ([HBB05; Neg07; Sch+03; Slo08; Wan+07]).

Die Anforderungen an Fahrsimulatoren variieren je nach Einsatz und Aufgabenstellung. Simulatoren mit dem Ziel des Trainings und der Untersuchung der Mensch-Maschine-Interaktion benötigen in der Regel keine detaillierten physikalischen Modelle der Fahrzeuge. Der Fokus liegt auf einer reproduzierbaren Abbildung einer Arbeitsaufgabe und der entsprechend konsistenten Simulation der Umgebung.
Im Gegensatz dazu verlangen typische Aufgabenstellungen der Fahrzeugentwicklung detaillierte Abbildungen der zugrundeliegenden physikalischen Effekte. Es entsteht der Anspruch, das Fahrzeug als digitalen Prototypen in Form geeigneter Simulationsmodelle abzubilden ([Col+07; Kor09]).

Abbildung 2.15: Fahrsimulator mit der Kabine einer mobilen Arbeitsmaschine

Die softwaretechnologischen Aspekte spielen bei der Implementation der meisten Fahrsimulatoren eine eher untergeordnete Rolle. Aufgrund der spezialisierten Problemdomäne bilden die meisten realisierten Systeme monolithische Architekturen ab ([OCS03]). Gemeinsamkeiten in der Struktur der eingesetzten Software sind dokumentiert ([Kor09; Neg07; Slo08; Wan+07]). Gemeinsame Module, Bibliotheken und Schnittstellen sind nicht Gegenstand der vorliegenden Arbeiten.

### 2.3.3 Echtzeitsimulation

Unter dem Begriff Echtzeitverhalten wird die Fähigkeit eines Systems verstanden, Daten innerhalb definierter Antwortzeiten zu verarbeiten ([Gha00; Kop11]). Für die Bewertung der Qualität eines Echtzeitsystems sind daher nicht nur die Ergebnisse der Datenverarbeitung relevant sondern auch das Einhalten der vorgegeben Zeitschranken. Die Zeitspanne, die dem System zur Verarbeitung der Daten zur Verfügung steht, ist dabei nicht relevant. Das System muss prinzipiell so dimensioniert sein, dass entsprechende Kapazitäten zur Verfügung stehen. Wesentlich für die Dimensionierung eines Echtzeitsystems ist dabei der Begriff der *Worst Case Execution Time*. Dieser beschreibt die maximal notwendige Zeitspanne zur Verarbeitung der Daten. Zur Bestimmung dieser Zeitspanne sind für nicht deterministische Effekte (z. B. durch Caches) entsprechende Annahmen zu treffen.
Grundsätzlich werden Echzeitsysteme hinsichtlich ihrer Anforderungen in harte und weiche Systeme unterteilt ([Ben94; Gha00; Kop11]). Harte Echtzeitanforderungen bedeuten, dass alle Zeitschranken eingehalten werden müssen. Andernfalls entstehen hohe Risiken für den Anwender des Systems. Beispiele für harte Echtzeitsysteme sind Steuerungen autonomer Fahrzeuge oder Materialflusssysteme.
Bei weichen Echtzeitsystemen wird das Überschreiten einzelner Zeitschranken toleriert. Die Qualität des Systems wird durch die Anzahl der Überschreitungen in einem bestimmten Bezugsintervall definiert. Diese Toleranz ist typisch für Multimediaanwendungen und damit auf VR-Systeme übertragbar. Vereinzelte Verzögerungen bei der Darstellung virtueller Welten werden im Allgemeinen nicht wahrgenommen. Erst bei einer gewissen Häufigkeit dieser Verzögerungen ensteht der Eindruck des „Ruckelns“ und das VR-System wird damit unbrauchbar.
Werden Echtzeitsysteme zur Ausnutzung zusätzlicher Ressourcen auf mehrere parallele Prozesse verteilt, ist zu prüfen, welche Anforderungen an die Abbildung einer gemeinsamen Zeitachse bestehen. Jeder Prozess erhält seine Zeitinformationen von der lokalen Uhr des ausführenden Rechners. Sind die Prozesse physisch auf mehrere Computer verteilt, müssen deren Uhren synchronisiert werden, um ein gemeinsames Zeitsignal für alle Prozesse des Systems abzubilden. In der Praxis stellt die Synchronisation der Uhren über ein Netzwerk häufig ein nicht triviales Problem dar, da die verbreiteten Netzwerktechnologien keine exakten Aussagen über die Dauer einer Nachrichtenübertragung zulassen ([Hor04; Krz]).
Eine wesentliche Vereinfachung wird erziehlt, wenn nicht der absolute Zeitpunkt von Ereignissen (z. B. Reaktionen des Bedieners auf bestimmte Signale) von Inter-

esse ist, sondern nur deren korrekte Reihenfolge. In diesem Fall genügt z. B. die Implementierung einer *Lamport Clock* ([Ben+08; Lam78]).
Der Einsatz eines Echtzeitsystems ist keine Garantie für das Erreichen von Zeitschranken. Vielmehr wird die durch die Funktion der Hardware und des Betriebssystems bedingte Streuung (Abbildung 2.16a) der *Worst Case Execution Time* reduziert ([Edg02; Wil+08]). In einem idealen Echtzeitsystem wird eine diskrete Verteilung mit einem Element erreicht (Abbildung 2.16b).
Zur Bewertung eines weichen Echtzeitsystems ist aus der Verteilung der *Worst Case Execution Time* der Anteil an Rechenschritten zu bestimmen, bei dem die Ausführungszeit oberhalb der durch das Simulationssystem gegebenen Schranke liegt. In einem System mit harten Echtzeitanforderungen ist die Schranke so zu wählen, dass die *Worst Case Execution Time* aller Rechenschritte kleiner als die Schranke ist.

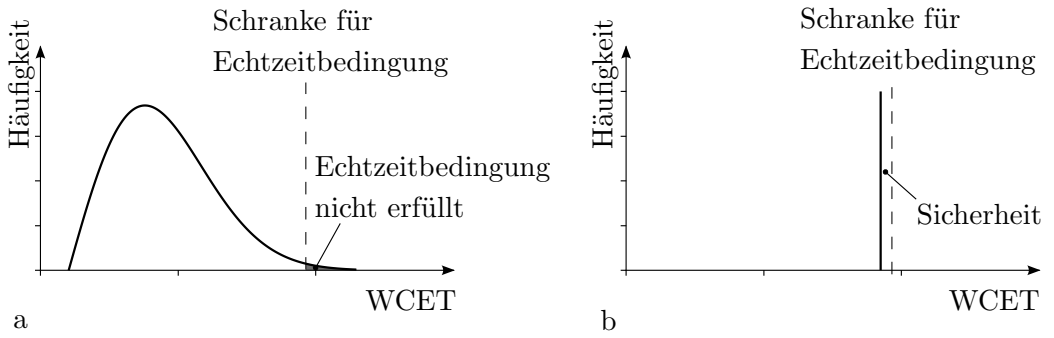


Abbildung 2.16: Verteilung der *Worst Case Execution Time*

Im Bereich der interaktiven Simulation ergeben sich aus der Echtzeitbedingung nach dem gegenwärtigen Stand der Technik hohe Anforderungen an die Algorithmen der numerischen Berechnung. Die notwendige Zeit zur Berechnung eines Simulationsschrittes wird wesentlich durch die Effizienz der Modellberechnung (siehe Abschnitt 2.1) definiert. Dies führt zu Restriktionen bei der Komplexität der einsetzbaren Modelle. So können z. B. hochfrequente Schwingungen in einem Fahrantrieb momentan nicht in Echtzeit berechnet werden. Es zeigt sich aber, dass es eine Reihe von Anwendungsfällen gibt, in denen echtzeitfähige Modelle formuliert und interaktiv berechnet werden können.
Einen allgemein gültigen Weg zur Ableitung von Echtzeitmodellen kann die vorliegende Arbeit nicht geben. Die Begrenzung des Rechenaufwandes auf das Notwendige ist

Aufgabe der Modellbildung. Entsprechend der Aufgabenstellung ist das technische System zu abstrahieren und durch geeignete Gleichungen zu beschreiben. Dabei besteht das Risiko, dass die entstehenden Modelle zu umfangreich für eine Berechnung in Echtzeit sind. In dem Fall ist die Anwendung alternativer Modelle ([RC07; PR03]) bzw. spezieller Modellbildungsmethoden ([BP00; MB11; PK06; RP05; Tob04; SBK10; Uch11]) notwendig.
Eine klassische Methode zur Minimierung des Risikos bei der Bereitstellung von echtzeitfähigen Simulationsmodellen ist der Codeexport ([Dro04; PK06; Pen06]). Die Modellgleichungen werden dabei durch das Modellierungswerkzeug in eine Programmiersprache (z. B. C++) überführt und anschließend in ausführbare Teile des Simulationssystems übersetzt. Aufgrund des Charakters des generierten Codes können Schleifen und Sprünge durch den Compiler aggressiver optimiert werden als bei allgemeinen Berechnungsvorschriften. Durch die Analyse der Modellstruktur ist der Codegenerator in der Lage Rechenoperationen zu identifizieren, die für das spezielle Modell nicht notwendig sind. Der generierte modellspezifische Code ist dadurch effizienter als eine allgemeingültige Berechnungsvorschrift.

# 3 Architektur des Simulationssystems

## 3.1 Anforderungen an das Simulationssystem

### 3.1.1 Anwendungsfälle

Das Konzept der interaktiven Simulation bei der Entwicklung mobiler Arbeitsmaschinen dient neben der Abbildung der virtuellen Prototypen auch der kontinuierlichen Weiterentwicklung der angewendeten Methoden. Unter Berücksichtigung dieser Anforderung ergibt die Analyse bestehender Fahrsimulatoren und VR-Systeme fünf wesentliche Anwendungsfälle für ein entsprechendes Simulationssystem.

- **Maschinenentwicklung aus technischer Sicht:** Unterschiedliche virtuelle Prototypen werden durch Modelle beschrieben und in Echtzeit interaktiv simuliert. Die Analyse der Ergebnisse erlaubt die Bewertung von Konzepten, Komponenten und Systemen. Auf dieser Basis lassen sich Entwicklungsentscheidungen begründen.
  Die Anwender des Simulationssystems sind Experten in der Domäne der mobilen Arbeitsmaschinen. Notwendig ist die Abstraktion des technischen Systems in ein Maschinenmodell unter Anwendung spezifischer Modellierungswerkzeuge. Die Integration von Originalbedienteilen setzt die Kenntnis der entsprechenden Protokolle voraus. Die Daten für die Grafik und Akustik des VR-Systems werden durch generische Werkzeuge erzeugt.

- **Maschinenentwicklung aus Sicht der Mensch-Maschine-Interaktion:** Unterschiedliche Komponenten und Systeme der Mensch-Maschine-Schnittstelle werden mit identischen Modellen simuliert. Zur Reduzierung des Aufwandes bei der Modellbildung und Parameterbestimmung sind diese Modelle soweit vereinfacht, dass sie der Erwartungshaltung der Probanden entsprechen. Für technische Aussagen anhand der Berechnung physikalischer Phänomene sind die Modelle eher ungeeignet. Eine objektive Bewertung des Fahrverhaltens auf der Basis der Simulationsergebnisse setzt dennoch eine entsprechende Modellgüte

voraus.
In diesem Anwendungsfall wird das Simulationssystem durch Experten auf dem Gebieten der Ergonomie, Ingenieurpsychologie und Anwendung mobiler Arbeitsmaschinen benutzt. Veränderungen am System finden hauptsächlich durch Variation der Mensch-Maschine-Schnittstelle statt.

- **Modellentwicklung:** Die Entwicklung der Rechenmodelle definiert einen weiteren wesentlichen Anwendungsfall. Bereits vorhandene Modelle sind je nach Aufgabenstellung nicht ausreichend. Der flexible Austausch unterschiedlicher Modelle für die Arbeitsmaschine (mechanische Struktur, Antrieb) und ihrer Interaktion mit der Umgebung (Reifen, Arbeitsprozess) soll durch die dargestellte Softwarearchitekur ermöglicht werden.
  Die Entwicklung neuer Rechenmodelle setzt umfangreiche Kenntnisse über die physikalischen Effekte auf den entsprechenden Gebieten voraus. Aus dem intuitiven Algorithmenbegriff (siehe Abschnitt 2.1) folgt die Notwendigkeit, die resultierenden Modelle in einer geeigneten Notation abzubilden. Das Simulationssystem muss in der Lage sein, diese Daten zu verarbeiten.

- **Methodenentwicklung:** Sowohl die Modellbildung als auch die Umsetzung in VR-Systemen profitieren von der permanenten Weiterentwicklung im Bereich der technischen Mechanik, Mechatronik und Informationstechnologie. Die Integration neuer Methoden sowie die Anpassung des Softwaresystems an neue Methoden soll mit wenig Aufwand möglich sein.
  Die Erweiterung des Simulationssystems durch neue Methoden erfordert Kenntnisse der Systemarchitektur und Implementation.

- **Systementwicklung:** Die Entwicklungen vor allem im Bereich der Computerhardware erlauben den permanenten Ausbau der VR-Technologie. Dieser Ausbau ist durch die Weiterentwicklung und Pflege der Software zu unterstützen. Für diese Aufgaben werden die entsprechenden Kenntnisse auf dem Gebiet der VR-Technologie sowie im Bereich der Programmierung des Simulationssystems benötigt.

### 3.1.2 Allgemeine Anforderungen an die Hard- und Software

Das Simulationssystem (siehe Abschnitt 2.1) besteht im Wesentlichen aus der Hardware, den Betriebssystemen und dem Softwaresystem. Hardware und Betriebssysteme

werden nicht durch die Softwarearchitekur beeinflusst, sondern stellen Randbedingungen dar. Es wird auf die Verwendung von Standardsystemen orientiert, d. h. es werden handelsübliche Personalcomputer und verbreitete Betriebssysteme (Linux, Windows) eingesetzt.
Die Anwendung spezieller Echtzeitsysteme ist für normale VR-Anwendungen nicht notwendig. Die vorliegenden weichen Echtzeitanforderungen (siehe Abschnitt 2.3.3) werden durch Standardbetriebssysteme erfüllt (siehe Videoplayer). Entstehen aufgrund spezieller Anwendungsfälle harte Echtzeitanforderungen, so sind diese durch geeignete problembezogene Erweiterungen des Systems abzubilden.
Die allgemeinen Anforderungen an das Softwaresystem werden im Wesentlichen durch die Erweiterbarkeit und Flexibilität sowie den Aufwand für die Wartung und Pflege definiert. Die Implementation der Software erfolgt in einer standardisierten Programmiersprache (C++). Dabei wird eine Plattformneutralität auf der Quellcodeebene gefordert. Diese Forderung kann durch den Einsatz entsprechender Bibliotheken erfüllt werden. Dabei wird auf die Anwendung freier und im Quellcode verfügbarer Bibliotheken orientiert.
Für die Benutzung des Simulationssystems im Rahmen von Maschinenentwicklungen sind Programmierkenntnisse in C++ keine Voraussetzung. Sowohl die virtuelle Umgebung als auch die Modelle der Maschine werden durch spezielle Modellierwerkzeuge abgebildet. Die Integration in das Simulationssystem erfolgt durch entsprechende Konfiguration der Software. Dabei wird die Konfiguration in einem Standardformat (XML) notiert. Dadurch können Konfigurationen mit elementaren Werkzeugen (Texteditor) modifiziert und erweitert werden, so dass die Anpassung einer Simulationsanwendung sehr einfach durchgeführt werden kann.
Die Einsatzfälle der Modell- und Methodenentwicklung betreffen einzelne, klar abgegrenzte Bereiche der gesamten Problemdomäne der interaktiven Simulation. Die Ableitung und Implementation entsprechender Algorithmen sowie die Tests der Methoden und Modelle finden aus Gründen der Effizienz und Übersichtlichkeit an Teilen des Simulationssystems statt. Insbesondere bei der Modellentwicklung können Parameterstudien ohne Echtzeitanforderungen durchgeführt werden.
Eine Entwicklung von Modellen und Methoden ohne Anwendung des Simulationssystems ist nicht sinnvoll, da bestehende und getestete Fragmente (z. B. numerische Methoden) neu zu implementieren wären. Die Berücksichtigung der vorhandenen Schnittstellen bei der Implementation neuer Modelle und Methoden erspart die spätere Anpassung bei der Integration in das Simulationssystem.

Die Weiterentwicklung des VR-Systems erfolgt parallel zur Entwicklung der zugrundeliegenden Technologien. Der Einsatz freier und im Quellcode vorliegender Bibliotheken erlaubt die Anpassung des Simulationssystems an die entsprechenden Release-Zyklen. Jede Bibliothek implementiert einen speziellen Teil der gesamten Problemdomäne. Für die Standardelemente von VR-Systemen (siehe Abschnitt 2.3.2, Abbildung 2.14) sind im Allgemeinen mehrere unabhängige Bibliotheken verfügbar. Bei einem notwendigen Wechsel der Implementation entsteht sowohl Aufwand bei der Implementation der Funktionalität als auch bei der Integration der neuen Bibliotheken. Ist der Austausch einer Bibliothek nicht zu vermeiden, muss das entstehende Risiko für das gesamte Softwaresystem minimiert werden.
Die Diskussion dieser Aspekte zeigt, dass ein modulares Softwaresystem notwendig ist, um die Anforderungen zu erfüllen. Die Problemdomäne der interaktiven Simulation mobiler Arbeitsmaschinen beinhaltet die klassischen Fachgebiete der Modellbildung und Simulation sowie der VR-Technologie. Die Vielfalt der daraus resultierenden Teilaufgaben kann nur begrenzt durch monolithische Softwaresysteme abgebildet werden. Ein einzelnes Programm für die interaktive Simulation mobiler Arbeitsmaschinen ist aufwendig zu implementieren und nur mit großem Aufwand zu warten. Bei der Erweiterung einzelner Teilbereiche muss der Einfluss auf das Gesamtsystem beachtet werden. Dadurch werden Gestaltungsmöglichkeiten begrenzt und Ressourcen gebunden. Das Risiko der Verfügbarkeit und der mangelnden Weiterentwicklung der benutzten Bibliotheken betrifft die gesamte Anwendung.
Im Gegensatz dazu erlaubt die Verteilung der Funktionalität des Softwaresystems auf klar abgegrenzte Bereiche eine auf die entsprechenden Problemfelder fokussierte Entwicklung. Die Integration zu einem Gesamtsystem erfolgt auf der Basis klar definierter Schnittstellen. Diese spiegeln die gemeinsamen Daten und Funktionen der Problemdomäne wieder und sind fachübergreifend intuitiv. Eine formale Beschreibung der Schnittstellen garantiert die Austauschbarkeit der einzelnen Elemente und bildet somit die Basis für die Unterteilung und Abgrenzung der einzelnen Bereiche.
Die Teilaufgaben können durch Experten der jeweiligen Domänen bearbeitet werden. Der Einsatz von Bibliotheken erfolgt ausschließlich innerhalb einer definierten Teilaufgabe. Die Risiken bei der Entwicklung, Wartung und Pflege des Simulationssystems werden auf die einzelnen Bereiche verteilt. Die in diesem Zusammenhang entwickelte Methode der komponentenorientierten Softwareentwicklung ([Sam97; Sch10; SGM02; WQ05]) wird bei der Entwicklung des Softwaresystems für die interaktive Simulation mobiler Arbeitsmaschinen angewendet.

## 3.2 Komponentenarchitektur und Framework

### 3.2.1 Begriffsdefinitionen

Eine Softwarekomponente beschreibt eine abgegrenzte eigenständige Funktionseinheit eines Softwaresystems mit klar definierten Schnittstellen ([Sam97; SGM02]). Die Komponenten sind innerhalb des Softwaresystems austauschbar und werden durch die Verbindung mit anderen Komponenten nicht verändert. Das Softwaresystem und die Komponenten werden durch ein gemeinsames Komponentenmodell beschrieben. Durch die Anwendung von Komponenten wird die Gesamtfunktionalität eines Softwaresystems verteilt. Jede Komponente bildet nur den Teil ab, für den sie spezifiziert ist. Dadurch werden die Softwareelemente einer Problemdomäne in eigenständigen Einheiten gekapselt, welche getrennt voneinander implementiert, getestet und gewartet werden können. Dabei ist es unerheblich, welchen Umfang an Code eine Komponente beinhaltet bzw. welche externen Bibliotheken sie benutzt. Der Anteil an wiederverwendbaren Fragmenten steigt mit zunehmender Spezialisierung der Komponenten ([SGM02]). Die notwendigen Basisbibliotheken sowie die Schnittstellen und Komponenten (Abbildung 3.1) bilden ein *Framework* ([Pre97; SGM02]).

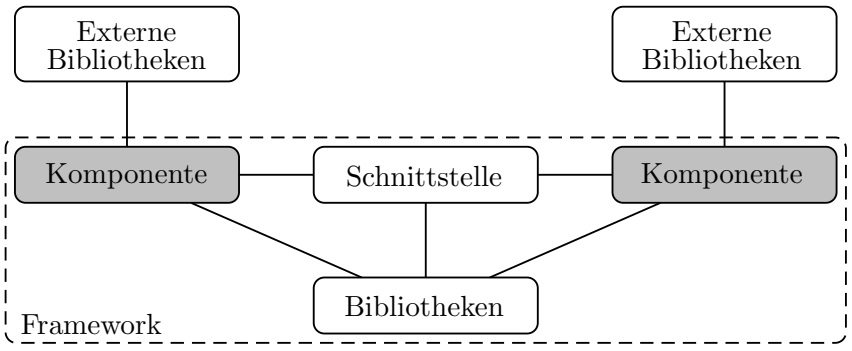


Abbildung 3.1: Komponenten, Framework und externe Bibliotheken

Im Unterschied zu einer Bibliothek mit implementierten Klassen und Funktionen werden durch ein Framework die wesentlichen Funktionselemente einer Problemdomäne durch Schnittstellen und Abstaktionen vorgegeben. Die Anpassung des Frameworks an eine konkrete Aufgabenstellung erfolgt durch Komposition der Komponenten. Durch die Implementation zusätzlicher Komponenten wird das Framework um neue Funktionen erweitert. Dabei ist die Anwendungslogik durch das Framework vorgege-

ben. Dem Prinzip der *Inversion of Control* folgend wird die einzelne Komponente vollständig durch diese externe Logik kontrolliert.
Die aus der objektorientierten Softwareentwicklung bekannten Objekte und Klassen bilden die Basis für die Implementation einer Komponente (Abbildung 3.2). Dabei wird die Funktionalität der Komponente durch die entsprechenden Instanzen abgebildet. Formal kann das Softwaresystem ausschließlich aus den Objekten und Klassen gebildet werden. Die resultierende Komplexität ist schwer zu beherrschen. Die Einführung von Komponenten definiert dafür den organisatorischen Rahmen.

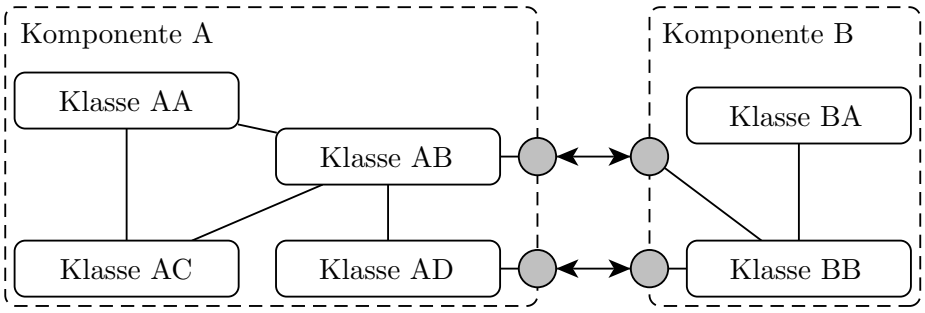


Abbildung 3.2: Komponenten und Klassen

Neben den Vorteilen bei der Implementation, Wartung und Pflege des Codes vereinfacht die Aufteilung der Funktionen auf die Komponenten die Anpassung des Softwaresystems an veränderte Aufgabenstellungen bzw. andere VR-Systeme. Durch die Definition von Schnittstellen wird die einzelne Komponente logisch vom Rest des Systems getrennt. Die Umgebung einer Komponente kann verändert werden, ohne dass die Komponente selbst angepasst werden muss.
Dieses Konzept ist bei der Entwicklung des Softwaresystems für die interaktive Simulation mobiler Arbeitsmaschinen sehr hilfreich. Der hauptsächliche Teil der Entwicklungsarbeit findet in der Regel nicht in der VR-Umgebung selbst statt, da die Verfügbarkeit der Systeme zeitlich begrenzt ist. Zusätzlich ist die Produktivität der Entwicklung in einem solchen System geringer, da erhebliche Aufwände für den Umgang mit der VR-Umgebung selbst enstehen.
Die Entwicklung einer einzelnen Komponente erfolgt daher in einer reduzierten Umgebung, die alle zur Entwicklung notwendigen Elemente des Simulationssystems beinhaltet. Der Transfer in das vollständige VR-System erfolgt durch die Erweiterung und Anpassung der Komponentenstruktur. So kann z. B. eine Modellkomponente mit entsprechenden Ein- und Ausgaben an einem Einzelrechner entwickelt werden

(Abbildung 3.3a). Im vollständigen VR-System werden Eingabe, Berechnung und Ausgabe auf verschiedenen Rechnern ausgeführt. Dieses wird durch den Einsatz entsprechender Netzwerkkomponenten realisiert (Abbildung 3.3b). Das Modell sowie die Ein- und Ausgabekomponenten sind invariant gegenüber der Veränderung der Komponentenstruktur.

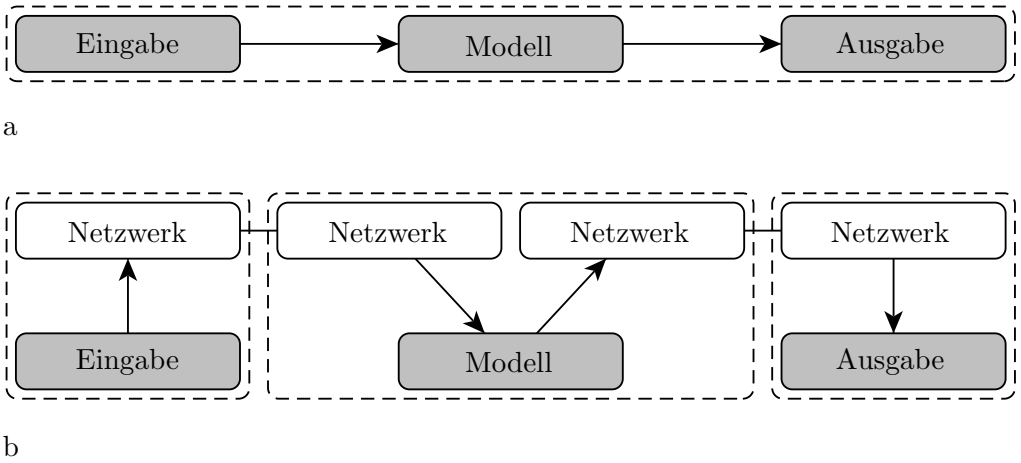


Abbildung 3.3: Einzelrechner- und Netzwerkkonfiguration

### 3.2.2 Architektur des Softwaresystems

Die Verknüpfung der Komponenten zu einer Anwendung wird als Komposition bezeichnet ([Aßm03; Sam97; SGM02]). Im Kontext dieser Arbeit ist der Begriff der Komposition identisch mit der Konfiguration einer Simulationsanwendung (siehe Abschnitt 3.1.1). Ein Kompositionssystem setzt neben der Existenz eines Komponentenmodells eine Kompositionssprache ([Aßm03]) voraus. Diese ist die Sprache für die Notation der Konfigurationen (XML). Das Komponentenmodell beinhaltet den Begriff der Plattform (Abbildung 3.4), welche die grundlegenden Funktionen zur Speicherverwaltung, Identifikation und Typprüfung der Komponenten zu Verfügung stellt.

Für die unterschiedlichen Aspekte der interaktiven Simulation in VR-Umgebungen werden einzelne Komponentenbereiche definiert (Abbildung 3.4). Diese Bereiche strukturieren die Menge der Komponenten nach Problemfeldern und werden mit vorhandenen, frei verfügbaren Bibliotheken und Standards verknüpft.

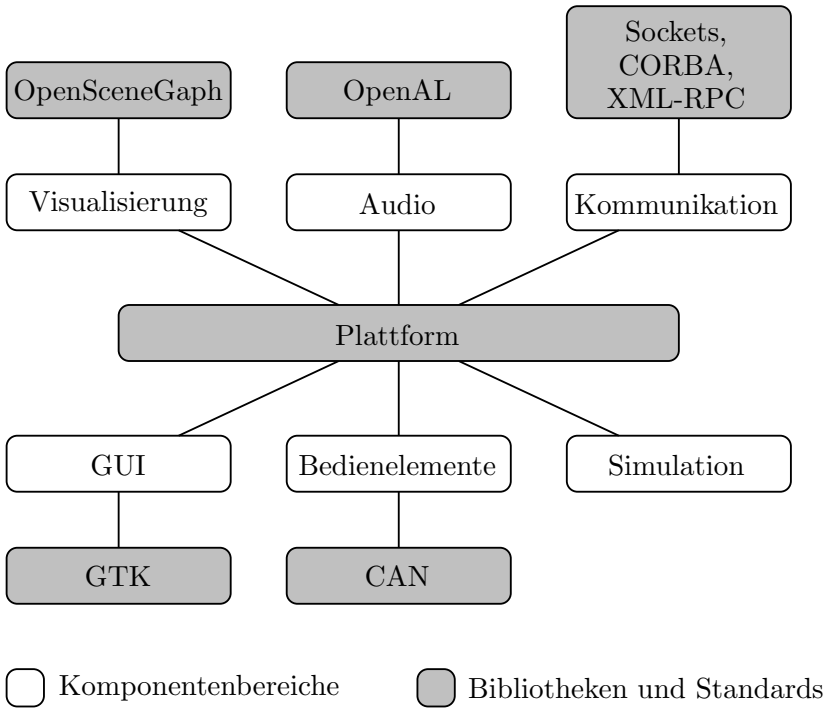


Abbildung 3.4: Architektur des Softwaresystems

Generische Komponentenmodelle und Plattformen sind Stand der Technik ([And04; SGM02; WQ05]). Vertreter dieser Technologie sind z. B. COM, CORBA und JAVABEANS. Die zugehörigen Komponentenmodelle sind nicht an eine Problemdomäne gebunden sondern definieren einen generischen Rahmen für die Entwicklung spezifischer Komponenten.

Das im Rahmen dieser Arbeit entwickelte Komponentenmodell ist spezifisch für die Problemdomäne der interaktiven Simulation mobiler Arbeitsmaschinen. Die generischen Komponentenmodelle zeichnen sich insbesondere durch die zur Verfügung gestellten Mechanismen der Interaktion (Name Services, Object Request Services, etc.) zwischen den Komponenten aus ([SGM02; WQ05]).

Aufgrund des Charakters der VR-Umgebung können diese Dienste wesentlich vereinfacht werden. So erfolgt die Identifikation von Komponenten innerhalb der Konfiguration anhand von eindeutigen Bezeichnern. Ein spezifischer Namensdienst ist nicht erforderlich. Die Simulationsanwendung wird während der Laufzeit nicht umkonfiguriert. Dadurch reduzieren sich die Anforderungen an die Flexibilität der Komponentenverbindungen. Die Kommunikation der Komponenten untereinander erfolgt lokal innerhalb des VR-Systems. Dementsprechend können die eingesetzten

Netzwerktechnologien vereinfacht werden. Mechanismen wie Transaktionen oder Verschlüsselung sind nicht erforderlich.
Aufgrund der reduzierten Anforderungen kann ein sehr einfaches Komponentenmodell benutzt werden. Dieses benutzt den XML-Standard zum Auswerten der Konfigurationen. Zum Instanziieren und Verknüpfen der Komponenten werden die Funktionen des Betriebssystems zum dynamischen Laden von Bibliotheken benutzt.
Für die Gestaltung der Komponenten gelten folgende grundlegende Annahmen:

- **Granularität:** Der Funktionsumfang einer Komponente wird durch den Entwurf spezifiziert. Dabei werden die geforderten Funktionen gegebenenfalls auf mehrere interagierende Komponenten verteilt ([CCK04]). Die wesentlichen Aspekte sind der Grad der Abstraktion, die Lokalität der Daten, die Fehlertoleranz und der Wartungsaufwand ([SGM02]). Dabei wird zwischen Komponenten mit wenig Funktionsumfang (*Fine-grained*) und komplexen Komponenten (*Coarse-grained*) unterschieden ([Sam97]). Komplexe Komponenten kapseln sehr viel Funktionalität. Eine Simulationsanwendung kann durch wenige komplexe Komponenten definiert werden. Dies gilt insbesondere bei domänenspezifischen Komponentenmodellen, da die wesentlichen Interaktionsmuster durch die Domäne intuitiv gegeben sind.
  Die Wiederverwendbarkeit wird durch die beschriebene Kapselung der Funktionalität deutlich eingeschränkt. Dies führt zu identischen Implementationen gleicher Codefragmente in unterschiedlichen Komponenten. Trotz der klar abgegrenzten Problemdomäne wird daher auf die Unterteilung in weniger spezialisierte Komponenten orientiert.

- **Nebenläufigkeit:** Das Softwaresystem ist grundsätzlich als nebenläufig anzusehen. Dies bedeutet, dass im Adressraum der Simulationsanwendung mehrere Prozesse parallel ablaufen können. Kritische Abschnitte und Synchronisationsmechanismen (z. B. Erzeuger-Verbraucher-Probleme) werden nicht durch die Konfiguration abgebildet sondern müssen durch die Implementation der jeweiligen Komponenten berücksichtigt werden.

- **Parallelisierung:** Parallelisierte Algorithmen zur Berechnung der Differentialgleichungen und numerischen Integration werden vollständig durch die entsprechenden Komponenten gekapselt. Es existieren keine Mechanismen zur Abbildung der Parallelisierung in der Konfiguration.

- **Automatisiertes Verhalten:** Ein wesentlicher Aspekt der Interaktion der Komponenten ist der Datenaustausch. Dabei kann jede Komponente direkt auf den Empfang neuer Daten reagieren und automatisch ihren Zustand aktualisieren. Für typische VR-Anwendungen ist dieses synchrone Verhalten jedoch nicht sinnvoll. Durch den Empfang der Daten wird die Berechnung der Modelle und Visualisierung blockiert. Die Reihenfolge der Aktualisierung der Komponenten wird durch die Kommunikation bestimmt. Dies führt zu nichtdeterministischem Verhalten, wenn entsprechend einfache Protokolle (z. B. UDP) zum Datentransfer eingesetzt werden.
  Durch die Definition spezieller Komponenten und Schnittstellen zur Synchronisation können sowohl Kommunikationsintervalle als auch die Reihenfolge der Aktualisierung der Komponenten konfiguriert werden.

### 3.2.3 Komponenten zur Berechnung der Maschinenmodelle

Einen Sonderfall bilden die Komponenten zur Berechnung der Maschinenmodelle für die Simulation. Die Interaktion zwischen der numerischen Integration und der dafür notwendigen Berechnung der Differentialgleichungssysteme (siehe Abschnitt 2.1) ist durch das Komponentenmodell und entsprechende Schnittstellen abgebildet. Die zur Verfügung stehenden numerischen Methoden können direkt in C++ implementiert und durch entsprechende Komponenten abgebildet werden.
Eine Notation der Maschinenmodelle in C++ ist dagegen nicht sinnvoll. Der hohe Grad der Abstraktion ist bereits am einfachen Beispiel des Zweimassenschwingers erkennbar (Abbildung 2.2). Zusätzlich sind Problemstellungen wie Speicherverwaltung und kritische Abschnitte zu beachten, die den Fokus bei der Entwicklung der Komponente vom eigentlichen Modell lösen.
Als Alternative zu der beschriebenen Notation in Differentialgleichungsform existiert die Benutzung entsprechender Objektbibliotheken für Mehrkörpersysteme und Antriebe. Eine mit den Elementen der Bibliotheken gebildete Berechnungsvorschrift ist prinzipiell in der Lage, die Funktion $f(x, t)$ abzubilden. Die Anwendung der Bibliotheken erfolgt dabei auf einer generischen Ebene. Problemspezifische Optimierungen können nicht durchgeführt werden, da der Code in den Bibliotheken allgemein gültig ist.
Im Bereich der Simulation technischer Systeme ist der Einsatz von Codegeneratoren eine verbreitete Methode ([Dro04; RP05; Mor07]). Je nach Anwendungsfall wird das Modell allein oder in Verbindung mit einer numerischen Methode exportiert.

Im Allgemeinen entsteht eine eigenständig lauffähige Anwendung, welche z. B. auf speziellen Echtzeitplattformen eingesetzt werden kann ([BDL09]).
Die Mechanismen für die Kommunikation und Synchronisation der Modellberechnung mit dem umgebenden Simulationssystem sind nicht standardisiert. Da die entsprechenden Funktionsaufrufe im generierten Code enthalten sein müssen, sind der Codegenerator und das Softwaresystem aufeinander abzustimmen. Dies führt im praktischen Einsatz zu erheblichen Einschränkungen, da die Codegeneratoren feste Bestandteile der Modellierwerkzeuge sind.

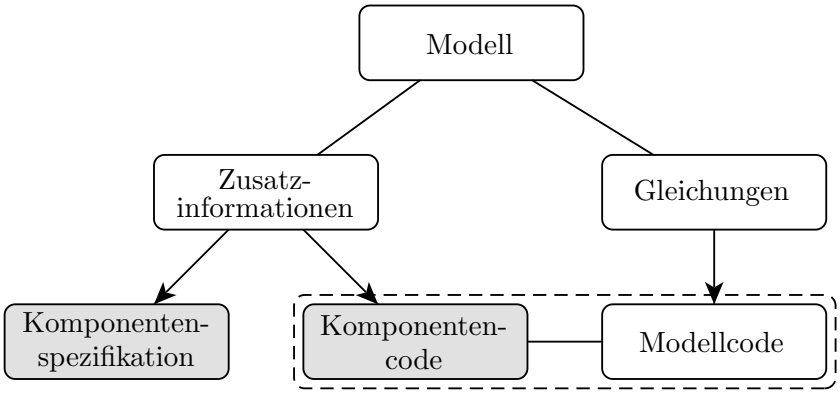


Abbildung 3.5: Komponenten aus Codegeneratoren

Der komponentenbasierte Ansatz stellt eine geeignete Lösungsmöglichkeit für diese Problemstellung dar. Der generierte Code zur Abbildung der Differentialgleichungssysteme kann auf die Berechnung der Funktion $f(x, t)$ (siehe Abschnitt 2.1) eingegrenzt werden. Dies entspricht intuitiv der Schnittstelle der entsprechenden Komponenten. Die Funktion $f(x, t)$ ist unabhängig vom Modellierwerkzeug im generierten Code zu finden ([Fre09; Gol10]).
Die Komponentenschnittstelle kann auf der Basis von Zusatzinformationen automatisch erzeugt werden (Abbildung 3.5). Dabei erfolgt die Adaption an die Signatur des generierten Codes durch entsprechende Vorlagen ([Gol10]) bzw. aus der Analyse des generierten Codes ([Fre09]). Dadurch können unterschiedliche Modellierwerkzeuge zur Abbildung von Maschinenmodellen eingesetzt werden. Die integrierten Codegeneratoren müssen nicht an das Simulationssystem angepasst werden.

# 4 Implementation

## 4.1 Framework

### 4.1.1 Hauptanwendung, Komponenten, Module und Packages

Entsprechend der definierten Komponentenarchitektur besteht die Hauptanwendung im Wesentlichen aus dem Komponentenlader. Dieser liest die Konfiguration der Simulationsanwendung, instanziiert die Komponenten und steuert den Anwendungsverlauf in der Anwendungshauptschleife (Abbildung 4.1). Dabei wird auf spezielle Komponenten zurückgegriffen, welche den sogenannten Anwendungskontext (siehe Abschnitt 4.1.2) bilden. Die restliche Komponentenstruktur ist transparent für die Hauptanwendung.

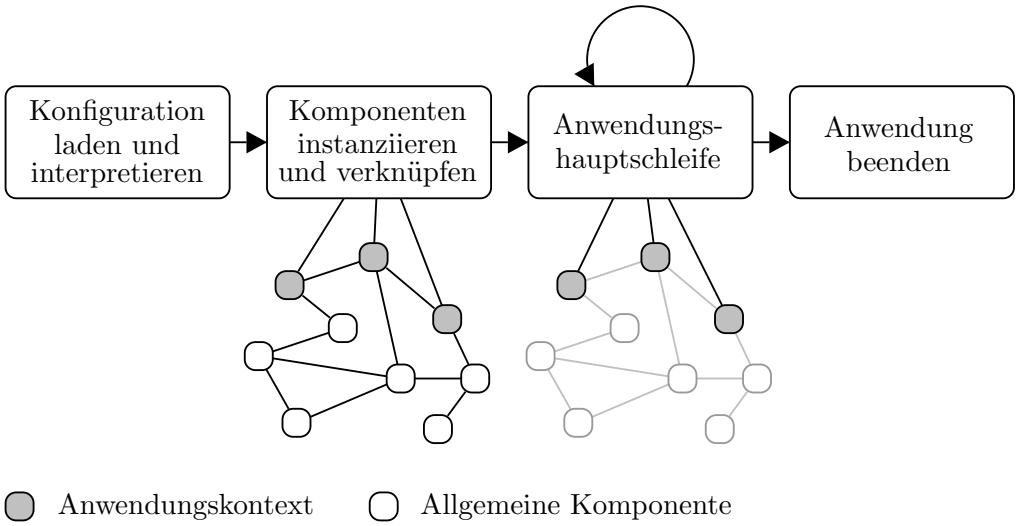


Abbildung 4.1: Ablauf einer SARTURIS-Anwendung

Die Funktionalität der Simulationsanwendung wird ausschließlich durch die verwendeten Komponenten sowie deren Verknüpfung definiert. Die Komponenten werden in Modulen organisiert. Diese Module sind dynamisch ladbare Bibliotheken mit

einer standardisierten Schnittstelle (Plugins). Zum Laden der Module wird auf Standardfunktionen des Betriebssystems bzw. der C-Bibliothek zurückgegriffen.
Das Modul selbst spielt bei der Konfiguration einer SARTURIS-Anwendung eine untergeordnete Rolle, da der Lader anhand der benutzten Komponente in der Lage ist, das Modul zu identifizieren, welches den Komponentencode zur Verfügung stellt. Module deren Komponenten den gleichen Anwendungshintergrund haben, lassen sich in Packages organisieren. Dadurch werden die Abhängigkeiten von externen Bibliotheken strukturiert. Das Simulationssystem für eine spezielle VR-Umgebung muss nur aus den für die geforderte Funktionalität notwendigen Packages bestehen. Dies erhöht die Flexibilität, Kompatibilität und Portabilität des Gesamtsystems. Die Organisation der Komponenten in Module und Packages bildet eine standardisierte Integrationsmöglichkeit für externe Bibliotheken (siehe Tabelle 4.1).

| Package | Funktionalität | externe Abhängigkeiten |
|---|---|---|
| sarturis | Grundfunktionen | XML-Bibliothek |
| sarturis-can | CAN-Bus-Integration | Geräte-Treiber-APIs |
| sarturis-corba | CORBA-Schnittstelle | CORBA-Impl. |
| sarturis-diffeqn | Integratoren und DGL-Systeme | SUNDIALS [Hin+05] |
| sarturis-gtk | Fenstersystem | GTK [GTK] |
| sarturis-hdf5 | Aufzeichnen und Wiedergeben von Simulationsdaten | HDF5 [HDF] |
| sarturis-osg | Szenegraph und Visualisierung | OSG [OSG] |
| sarturis-python | PYTHON als Skriptsprache | PYTHON [PYT] |
| sarturis-simpack | Integration von SIMPACK-Modellen | SIMPACK |
| sarturis-simulator | Simulatorspezifische Komponenten und Module | |
| sarturis-vlc | Videoein- und ausgabe | VLC [VLC] |
| sarturis-xmlrpc | XML-RPC-Schnittstelle | XML-RPC-Impl. |

Tabelle 4.1: Packages und externe Abhängigkeiten

Die Struktur der Packages (Abbildung 4.2) zeigt, dass nur einige wenige Basispackages existieren, von denen andere Packages abhängig sind. Die meisten Packages bilden Terminierungen im Abhängigkeitsbaum, d. h. sie sind keine Voraussetzung für andere

Packages. Die Basispackages bilden ein Grundgerüst an Funktionalität und sind Voraussetzung jeder Simulationsanwendung. Die dafür notwendigen Bibliotheken sind in allen Zielplattformen (siehe Abschnitt 3.1.2) für das Simulationssystem verfügbar. Der gesamte Funktionsumfang, welcher durch die Integration der Komponenten erreicht wird, wäre in einer monolithischen Anwendung nur schwer zu realisieren und kaum zu warten. Durch die Zerlegung der gesamten Anwendung in die einzelnen Problemfelder können diese unabhängig voneinander an jeweils neue Technologien angepasst werden. Dabei notwendige Überarbeitungen des Codes betreffen nur das entsprechende Fachgebiet und beeinflussen den Rest des Systems nicht.

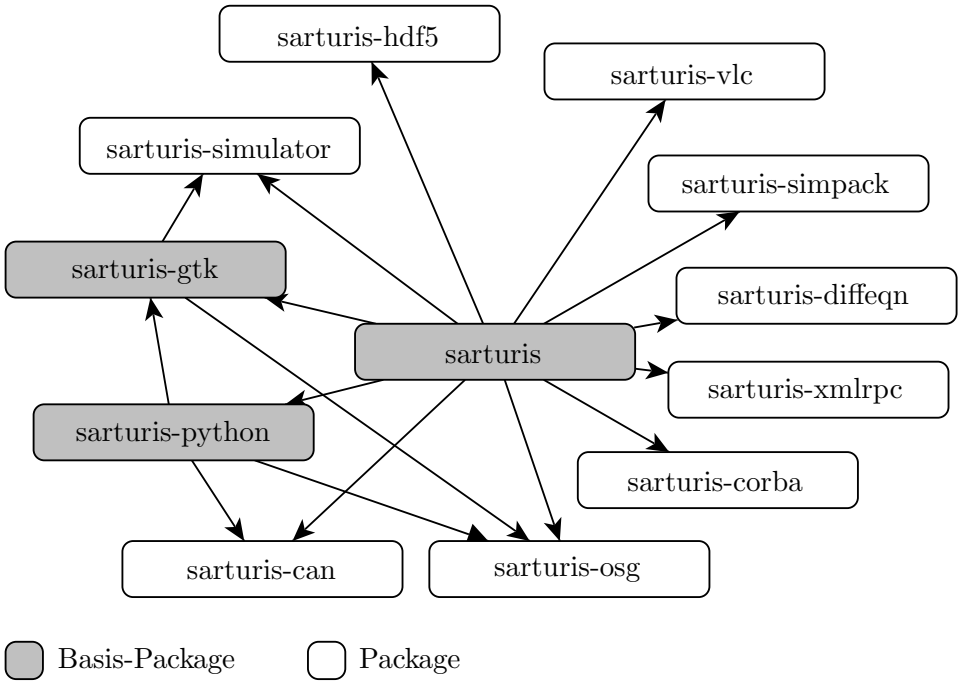


Abbildung 4.2: Packages und Abhängigkeiten

### 4.1.2 Schnittstellen und Synchronisation

Durch die Analyse der Problemdomäne der interaktiven Simulation mobiler Arbeitsmaschinen lassen sich die grundlegenden Schnittstellen zwischen den Komponenten ableiten. Die Menge der Schnittstellen ist prinzipiell unbegrenzt. Die innerhalb eines Packages definierten Schnittstellen sind in der Regel an den Funktionsumfang des Packages gebunden und kapseln häufig Schnittstellen externer Bibliotheken.

Schnittstellen, die nicht zu einem speziellen Package gehören, beschreiben die allgemein zur interaktiven Simulation in VR-Umgebungen notwendigen Beziehungen zwischen den Komponenten. Die benutzten Datenstrukturen sind eher flach und aus den grundlegenden Datentypen von C++ und der STL zusammengesetzt. Dadurch entstehen keine Abhängigkeiten zu weiteren externen Bibliotheken.
Alle Schnittstellen werden in der Implementation durch vollständig abstrakte Klassen beschrieben. Die Anwendung der Mehrfachvererbung ([Car84; Str89]) ermöglicht die Implementation mehrerer Schnittstellen durch eine Komponente. Die Typsicherheit der Komponenten wird innerhalb der Laufzeitumgebung von SARTURIS durch dynamisches Casten überprüft.
Eine der grundlegenden Schnittstellen ist der `ApplicationContext`. Dieser definiert hauptsächlich Methoden für die Initialisierung, die Hauptschleife und das Beenden der Anwendung. Dabei werden die konfigurierten Komponenten, die diese Schnittstelle implementieren, nacheinander mit den entsprechenden Methoden aufgerufen. Eine Priorisierung erfolgt nicht. Die Hauptschleife setzt kooperatives Verhalten der Komponenten voraus, d. h. die Funktionsaufrufe müssen terminieren ([NS01]).
Für die Synchronisation der Anwendung existiert die Schnittstelle `Executable`. Diese beschreibt einen ausführbaren Codeblock, der z. B. periodisch durch einen Timer oder als Reaktion auf eine Eingabe ausgeführt wird. Als Spezialisierung fungieren die Schnittstellen `Updateable` und `Resettable`. Diese spezialisieren den Kontext des ausführbaren Codeblocks auf die konkreten Aktionen zum Aktualisieren oder Zurücksetzen eines Objektes.
Für nebenläufige Anwendungen ist die Schnittstelle `Thread` definiert. Den entsprechenden Komponenten ist eine Implementation der Schnittstelle `Executable` zugeordnet, welche in dem entsprechenden Prozess des Betriebssystems ausgeführt wird. Eine Komponente kann mehreren Threads zugeordnet werden. Der Schutz kritischer Abschnitte muss durch die Implementation der Komponente sichergestellt werden. Sind externe Bibliotheken nicht für die Benutzung in nebenläufigen Anwendungen geeignet, so muss dies bei der Implementation von abgeleiteten Komponenten berücksichtigt werden.
Wesentlich für die interakive Simulation ist der Datenaustausch zwischen den Komponenten. Die dabei zu berücksichtigenden Datentypen bzw. -strukturen sind:

- Digitale Signale für Ein- und Ausgaben
- Analoge Skalare für eindimensionale Zustände, Ein- und Ausgaben

- Analoge Vektoren für Zustandsräume
- Koordinatentransformationen zur Positionierung von Objekten im Raum
- Text zur Anzeige auf Instrumenten oder der Projektion
- Winkel, Geschwindigkeiten und Beschleunigungen zur Bewegungssteuerung
- Objektdaten zur Darstellung von Zusatzinformationen
- Daten zur Steuerung der Simulationsanwendung.

Für jeden dieser Datentypen existiert eine Eingabeschnittstelle (`Input`) und eine Ausgabeschnittstelle (`Output`). Der Datenaustausch zwischen zwei Komponenten erfolgt prinzipiell auf zwei unterschiedlichen Wegen (Abbildung 4.3). Im ersten Fall implementiert die Komponente B die Schnittstelle `Input` und die Komponente A liest die Daten. Im zweiten Fall implementiert A die Schnittstelle `Output` und B schreibt die Daten. Beide Fälle erfordern den gleichen Aufwand bei der Implementation der Funktionalität und Konfiguration der Struktur und Synchronisation. Eine Präferenz für eine der beiden Varianten kann pauschal nicht gegeben werden. Die Entscheidung, ob der Datenaustausch zwischen zwei Komponenten lesend oder schreibend erfolgt, ist an den Entwurf der entsprechenden Komponenten gebunden.

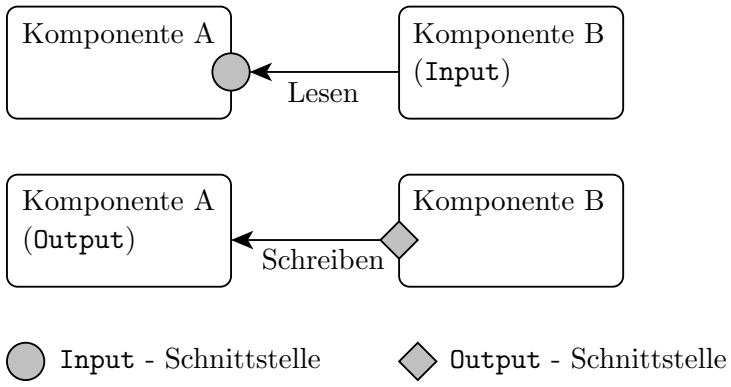


Abbildung 4.3: Möglicher Datenaustausch zwischen zwei Komponenten

Auf der Basis der definierten Datenschnittstellen lassen sich einfache Grundkomponenten implementieren, welche die Synchronisation der Daten zwischen den Komponenten abbilden. Diese Grundkomponenten sind Speicher (`Duplex`), Datenaustauschelemente (`Chain`) und Konvertierelemente (`Mux, Demux, Converter`).

Von besonderer Bedeutung sind Speicher. Diese implementieren die Eingabe- und die Ausgabeschnittstelle eines Datentypen unter Berücksichtigung kritischer Abschnitte. Durch die Anwendung generischer Programmiertechnologien kann der Aufwand für die Implementation deutlich reduziert werden.
Der Einsatz von Speichern entkoppelt die Simulationsanwendung. Unterschiedliche nebenläufige Bereiche der Simulation verarbeiten oder generieren Daten mit unterschiedlicher Frequenz. Eine Synchronisation im Sinne eines Erzeuger-Verbraucher-Problems ist in der Regel nicht notwendig. Entsprechend den weichen Echtzeitanforderungen des VR-Systems (siehe Abschnitt 2.3.3) benötigt jede Komponente gültige Daten zum „aktuellen“ Zeitpunkt der Simulation. Dieser asynchrone Datenaustausch mit unterschiedlicher Frequenz wird durch den Einsatz von Speichern realisiert (Abbildung 4.4).

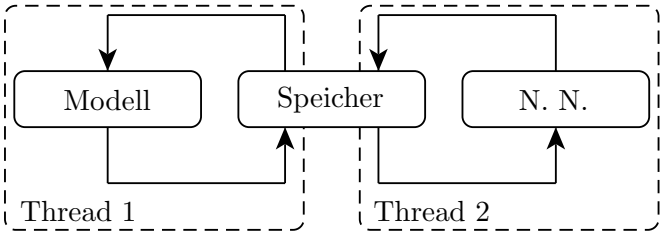


Abbildung 4.4: Datenaustausch mit externen Speichern

Ein weiterer wichtiger Anwendungsfall für Speicher ist die Adaption der `Input`- und `Output`-Schnittstellen von Komponenten. Die sinnvolle Aufteilung der Funktionalität des Gesamtsystems in klar abgegrenzte Komponenten beinhaltet das Risiko inkompatibler Teilentwürfe. Soll eine Komponente A, der eine Eingabeschnittstelle zugeordnet ist, mit einer Komponente B verbunden werden, die eine Ausgabeschnittstelle erwartet, so kann diese Verbindung nur mit Hilfe eines zusätzlichen Speichers realisiert werden (Abbildung 4.5).

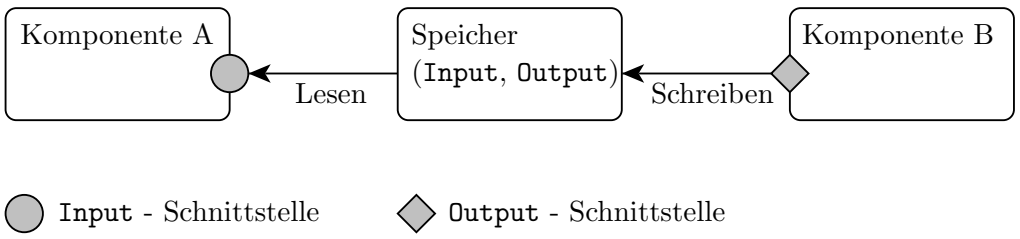


Abbildung 4.5: Adaption von Ein- und Ausgabeschnittstellen durch Speicher

## 4.2 Komponentenspezifikation

### 4.2.1 Aufbau, Notation und Typsystem

Die Komponentenspezifikation wird in SARTURIS auch als Typdefinition bezeichnet. Diese Spezifikation ist in XML notiert und beschreibt Parameter, Schnittstellen und Abhängigkeiten der Komponente mit den folgenden Elementen:

- **Basistypen** definieren die durch die Komponente implementierten Schnittstellen. Definiert werden diese durch das `Base`-Tag.
- **Attribute** sind einfache Parameter für die Instanz. Mögliche Attribut-Typen sind Zeichenketten, Zahlen, boolesche Ausdrücke sowie Aufzählungen. Attribute werden durch das Tag `Attribute` definiert und sind Pflichtelemente, denen in der Typdefinition ein Standardwert zugewiesen werden kann.
- **Referenzen** bilden Verweise auf andere Instanzen ab. Der Verweis erfolgt durch eine Zeichenkette mit dem Namen der anderen Instanz. Die Auflösung des Namens sowie die Typprüfung erfolgt durch die Laufzeitumgebung. Das Tag `Instance` innerhalb des `Reference`-Tags definiert den Typen der anderen Instanz. Zusätzlich können weitere Attribute definiert werden. Diese parametrisieren die Verbindung zwischen den beiden Instanzen.
- **Konfigurationen** beschreiben komplexere Datenstrukturen für die Konfiguration einer Instanz. Dazu kann das Tag `Configuration` beliebig oft geschachtelt und mit Attributen kombiniert werden.

Abbildung 4.6 zeigt die Typdefinition für eine Komponente zur Berechnung der Modellgleichungen eines Radladers. Die Komponente implementiert mehrere Schnittstellen. Namensräume dienen der Strukturierung des Komponentensystems und werden durch die Zeichenkette „::“ getrennt. Das Attribut `Model` bezeichnet die Modellbezeichnung des Radladers und dient eher untergeordneten Zwecken.
Externe Referenzen sind die Bedienereingaben (z. B. das Fahrpedal) sowie die Ausgabeschnittstellen für die im Modell berechneten Werte (z. B. die Fahrgeschwindigkeit). Die Eingabeschnittstellen sind in der Regel obligatorisch (siehe Attribute `Min` und `Max`). Die berechnete Fahrgeschwindigkeit wird in eine beliebige Anzahl von Schnittstellenobjekten geschrieben und dabei je nach gewünschter Maßeinheit umgerechnet. Alle Schnittstellenobjekte benutzen die in Abschnitt 4.1.2 behandelten Standardtypen.

Die Parametrisierung des Modells erfolgt durch komplexe Datenstrukturen, welche z. B. durch das `Configuration`-Element `Structure` repräsentiert werden.
Durch die Benennung der Komponenten wird ein Typsystem definiert, welches durch Namensräume organisiert ist. Die Anwendung dieses Konzepts erfolgt analog zur Implementierung der Komponenten in C++. Dadurch werden Konflikte mit gleichnamigen Objekten unterschiedlicher Problemdomänen vermieden.

```
1  <Type Name="sarturis::models::WheelLoader">
2    <Base Type="sarturis::Model"/>
3    <Base Type="sarturis::vehicles::FourWheeler"/>
4    ...
5    <Attribute Name="Model" Type="string"/>
6    ...
7    <Reference Name="DrivePedal" Min="1" Max="1">
8      <Instance Type="sarturis::AnalogInput"/>
9    </Reference>
10   <Reference Name="Speed" Min="0" Max="-1">
11     <Instance Type="sarturis::AnalogOutput"/>
12     <Attribute Name="Unit" Type="enum:MPH:KMH/>
13   </Reference>
14   ...
15   <Configuration Name="Structure" Min="1" Max="1">
16     <Configuration Name="FrontAxle" Min="1" Max="1">
17       <Configuration Name="LeftWheel">
18         <Attribute Name="Stiffness" Type="double"/>
19         <Attribute Name="Damping" Type="double"/>
20       </Configuration>
21       ...
22       <Attribute Name="Mass" Type="double"/>
23     </Configuration>
24   </Configuration>
25 </Type>
```

Abbildung 4.6: Spezifikation einer Komponente

### 4.2.2 Konfiguration

Die Konfiguration einer konkreten Simulationsanwendung erfolgt ebenfalls in XML. Die Hauptanwendung liest und interpretiert die Konfiguration, welche auf mehrere Dokumente verteilt werden kann. Die Prüfung der Korrektheit der Konfiguration erfolgt durch den Vergleich mit der Komponentenspezifikation.

```
1  <Instance xsi:type="sarturis__models__WheelLoader"
2            Name="WheelLoader"
3            Model="WL612X">
4
5    <DrivePedal Reference="Pedal1" />
6    <Speed Reference="SpeedOut1" Unit="KMH" />
7    ...
8    <Structure>
9      <FrontAxle Mass="326">
10       <LeftWheel Stiffness="56.1" Damping="3.4" />
11       ...
12     </FrontAxle>
13     ...
14   </Structure>
15 </Instance>
```

Abbildung 4.7: Konfiguration einer Instanz der Radladerkomponente

Dabei wird die Struktur der Attribute, Referenzen und Konfiguration durch Validierung gegen ein XML-Schema überprüft. Der wesentliche Vorteil dieser Methode ist, dass die Prüfung der Korrektheit ohne die Sarturis-Anwendung durch eine standardisierte XML-Technologie erfolgen kann. Nachteilig wirkt sich die notwendige Synchronisation des Schemas und der Komponentenspezifikation aus.
Abbildung 4.7 zeigt die Konfiguration einer Instanz der in Abschnitt 4.2.1 spezifizierten Radladerkomponente. XML-Schema erlaubt die Abstraktion aller Instanzen durch das Tag `Instance` und die Spezifikation der Komponente durch das standardisierte Tag `xsi:type`. Die Trennung der Namensräume kann in XML-Schema nicht durch die Zeichenkette „::“ erfolgen, da der Doppelpunkt im XML-Standard reserviert ist. Als Ersatz wird die Zeichenkette „__“ verwendet.
Schnittstellenobjekte werden durch ihre Namen referenziert. Die Typsicherheit kann nicht bei der Interpretation der Konfiguration überprüft werden, da durch die generische Angabe einer Zeichenkette keine Typinformationen vorliegen. Durch die identische Abbildung des Typsystems zwischen der Komponentenspezifikation und der Implementation in C++ erfolgt die Prüfung der Typsicherheit bei der Instanziierung und Verknüpfung der Komponenten. Dafür ist die Implementierung eines Namensdienstes notwendig. Dieser ermittelt die Komponente anhand des `Reference`-Attributes und führt die entsprechende Typprüfung durch. Die Reihenfolge der Instanziierung ist somit von Bedeutung und wird durch den Aufbau und die Abarbeitung eines gerichteten Graphen sichergestellt.

### 4.2.3 Automatisierte Komponenten- und Schemaerstellung

Sowohl das für die Prüfung der Konfigurationsstruktur notwendige Schema als auch die Abbildung des Typsystems bei der Implementierung der Komponenten stellen Redundanzen in der Komponentenarchitektur von SARTURIS dar. Die Deklaration der Komponentenklasse (Abbildung 4.8) und das Schema (Abbildung 4.9) lassen sich mit den Informationen der Komponentenspezifikation notieren.

```
1  namespace sarturis
2  {
3    namespace models
4    {
5      class WheelLoader : public sarturis::Model
6      {
7        public:
8          WheelLoader(const std::string& Model,
9                      const DrivePedal_Ref& DrivePedal,
10                     const Speed_Ref& Speed,
11                     const Structure_Conf& Structure);
12     };
13   }
14 }
```

Abbildung 4.8: Klassendefinition der Radladerkomponente

Die SARTURIS-Module verfügen über eine standardisierte Schnittstelle zur Instanzierung und Verknüpfung der Komponenten. Dabei kann die Typprüfung mit Hilfe des dynamischen Casts von C++ erfolgen. Dies gilt sowohl für die Vererbungshierarchie der Komponente selbst (`Base`) als auch für die referenzierten Komponenten (`Reference`). Der dafür notwendige Code kann auf der Basis der Komponentenspezifikation automatisch generiert werden. Die Transformation der Komponentenspezifikation in C++-Code und in XML-SCHEMA ist in SARTURIS Bestandteil der Werkzeugkette bei der Modulerstellung (Abbildung 4.10).
Das übersetzte Modul und das Schema bilden die Informationen aus der Komponentenspezifikation redundant in verschiedenen Formaten ab. Fehler entstehen, wenn die Struktur des Schemas nicht zu den binären Daten des Moduls passt. Aus diesem Grund wird aus dem Quelltext der Komponentenspezifikation eine Checksumme gebildet, welche im Schema und im Code des Moduls abgelegt wird. Durch

den Vergleich der Checksummen kann der Lader sicherstellen, dass die erwartete Konfigurationsstruktur mit dem Code des Moduls kompatibel ist.

```
<xsd:complexType name="sarturis__models__WheelLoader">
  <xsd:sequence>
    <xsd:element name="DrivePedal" minOccurs="1" maxOccurs="1">
      ...
    </xsd:element>
    <xsd:element name="Speed" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:complexContent>
          <xsd:extension base="Referenced">
            <xsd:attribute name="Unit" type="xsd:string"/>
          </xsd:extension base="Referenced">
        </xsd:complexContent>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="Model" type="xsd:string" use="required"/>
</xsd:complexType>
```

Abbildung 4.9: Schema der Radladerkomponente

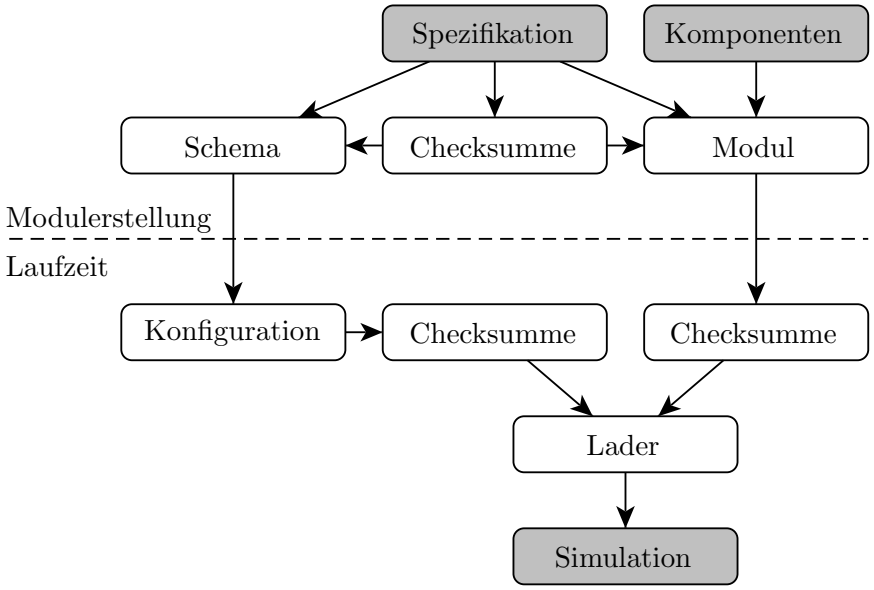


Abbildung 4.10: Typdefinition und Modulerstellung

### 4.2.4 Reflektion

Reflektion bezeichnet die Fähigkeit von Objekten, Informationen über ihre Struktur zur Verfügung zu stellen ([MJD96]). Dies geschieht im Rahmen der objektorientierten Programmierung durch sogenannte Typobjekte. Diese speziellen Klassen erlauben eine tiefergehende Analyse und Manipulation der Funktionsstruktur eines Programmes zur Laufzeit.
Im Rahmen des Komponentensystems von SARTURIS kann Reflektion unter anderem benutzt werden, um Parameter einer Komponente während der Laufzeit zu ändern. Anwendungsfälle einer solchen Parameteränderung sind z. B. die Analyse numerischer Methoden oder die Konfiguration grafischer Algorithmen. Die Parameteränderung ist dabei Bestandteil der Konfiguration einer Simulationsanwendung.
Die Alternative zum Einsatz von Reflektion wäre die Verbindung jedes Parameters mit einer Schnittstelle (siehe Abschnitt 4.1.2). Die Menge der Parameter der resultierenden Komponente wird dann um die einstellbaren Parameter erweitert. Diese Einordnung wird durch den Entwickler getroffen und ist für den Anwender der Komponente fixiert. Je nach Komponente und Einstellmöglichkeiten erhöht sich die Anzahl der Schnittstellen um ein Vielfaches. Für eine korrekte Konfiguration ist zwischen Schnittstellen, die für die Funktion der Komponente von Bedeutung sind und denen die Parameter einstellen, zu unterscheiden. Dadurch wird die Übersichtlichkeit für den Anwender der Komponente reduziert. Die Anpassung von Parametern einer Komponente durch Reflektion trennt diese von den Schnittstellen und damit von der Verwendung in einer Simulationsanwendung. Dadurch bleibt die Übersichtlichkeit der Konfiguration erhalten.
Reflektion ist eine Eigenschaft der eingesetzten Programmiersprache. Mit Ausnahme von Laufzeit-Typinformationen bietet C++ keine weiteren Technologien zur Reflektion ([AC01; CKW98]). Ein Ansatz ist das Generieren von Metaobjekten aus den gegebenen C++-Klassen ([Chi95; CKW98; DCL07]). Dafür ist ein geeigneter Generator notwendig, der die entsprechenden Klassendefinitionen analysiert und die Metaobjektklassen generiert.
Für die Manipulation einer SARTURIS-Komponente zur Laufzeit ist dieser Aufwand nicht notwendig, da die innere Struktur einer Komponente unabhängig von den Parametern und Schnittstellen ist (siehe Abschnitt 3.2, Abbildung 3.2). Alle notwendigen Informationen sind in der Komponentenspezifikation beschrieben.
Analog zur Generierung des Schemas und des Modulcodes (siehe Abschnitt 4.2.3) können anhand der Komponentenspezifikation Typobjekte generiert werden. Diese

Typobjekte verfügen über generische Schnittstellen zur Beschreibung der Struktur der Komponente und zur Manipulation einer entsprechenden Instanz (Abbildung 4.11).

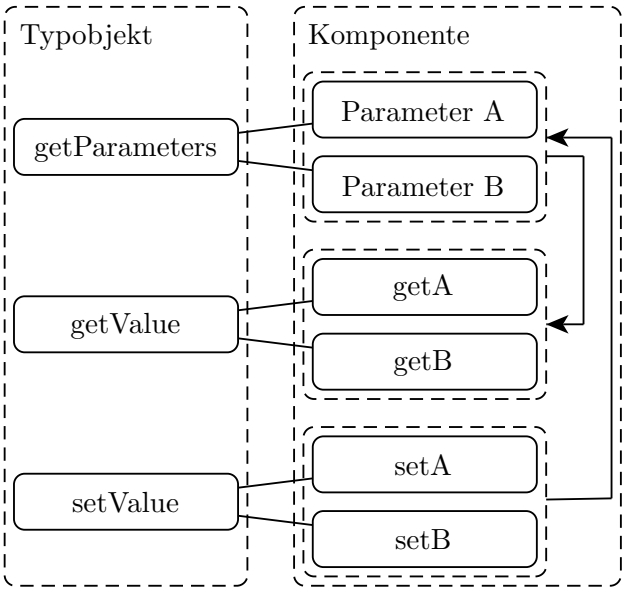


Abbildung 4.11: Reflektion, Typobjekt und Instanz einer Komponente

Innerhalb der generischen Methoden der Typobjekte ist der Aufruf der spezialisierten Methoden der Komponenten notwendig. Diese führen die eigentliche Manipulation des Parameters inklusive der notwendigen Veränderungen innerhalb der Komponente durch. Zum Aufruf der spezialisierten Methoden werden die generisch als Zeichenkette übergebenen Parameterwerte in die spezialisierten Datentypen aus der Komponentenspezifikation konvertiert. Anschließend wird anhand des Parameternamens der Name der spezialisierten Methode berechnet und diese Methode in einer Verzweigung aufgerufen. Der entsprechende Code der Typobjekte wird anhand der Komponentenspezifikation generiert (Abbildung 4.12).
Als notwendige Voraussetzung muss die Komponente die entsprechenden Methoden zur Manipulation von Parametern implementieren. Diese Anforderung wird während des Übersetzens der Komponente direkt durch den Compiler typsicher überprüft und stellt damit kein Risiko für den Komponentenanwender dar. Die konkrete Implementation einer Komponente wird durch die Abbildung der Typobjekte nicht beeinflusst. Es sind keine Erweiterungen des C++-Quellcodes notwendig. Spezielle Vererbungshierarchien werden nicht vorausgesetzt. Dadurch müssen auch keine exter-

nen Bibliotheken angepasst werden. Die Manipulation von Komponenten zur Laufzeit kann z. B. mit einer grafischen Benutzeroberfläche erfolgen (Abbildung 4.13).

```
1  void ComponentType::SetValue(const std::string& Name,
2                               const std::string& Value)
3  {
4    // Verzweigung und Typkonvertierung nach Parametername
5    if (Name=="A") comp->setA(strtoint(Value));
6    else if (Name="B") comp->setB(strtodouble(Value));
7    ...
8
9    // Unbekannter Parameter
10   else throw UnknownParameter(Name);
11 }
```

Abbildung 4.12: Spezialisierung der Methodenaufrufe bei Reflektion

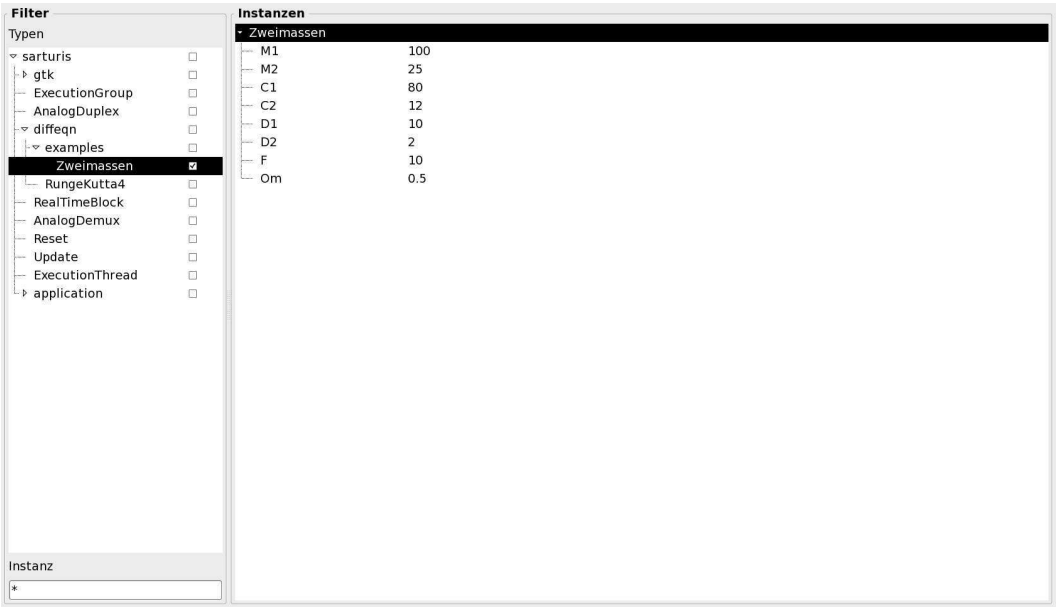


Abbildung 4.13: Grafische Benutzeroberfläche zur Manipulation einer Komponente

Die Adaption an die standardisierten Datenschnittstellen (siehe Abschnitt 4.1.2) erfolgt durch generische Adapterkomponenten (Abbildung 4.14). Damit wird die Parameteranpassung in das Schnittstellensystem von SARTURIS integriert und die

Eigenschaften von Objekten können z. B. mit den Bedienelementen des Fahrzeuges während der Simulation angepasst werden.

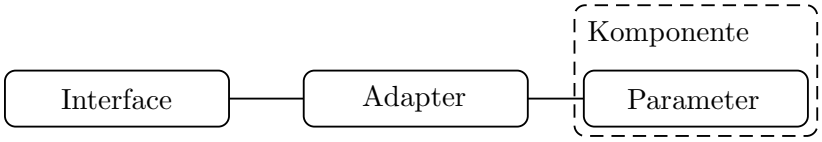


Abbildung 4.14: Adapter zur Manipulation von Komponenten

## 4.3 Komponenten der numerischen Simulation

Die Komponenten der numerischen Simulation bilden den wesentlichen Kern des SARTURIS-Frameworks. Durch diese Komponenten werden die Maschinenmodelle und die mathematischen Verfahren zu deren interaktiver Berechnung implementiert. Die Aspekte der Modellbildung sowie der Einsatz von Codegeneratoren zur Transformation von Modellen in ausführbaren Code finden bei der Erstellung der Komponenten für die Maschinenmodelle Berücksichtigung (siehe Abschnitt 3.2.3). Die entsprechenden numerischen Verfahren werden als manuell implementierte Komponenten bereitgestellt.

Die Problemstellung der Simulation ist durch die numerische Lösung einer Anfangswertaufgabe charakterisiert (siehe Abschnitt 2.1). Das konkrete Maschinenmodell wird durch ein System gewöhnlicher Differentialgleichungen oder durch ein differential-algebraisches Gleichungssystem beschrieben. Zur Abstraktion dienen die Schnittstellen `OdeSystem` und `DaeSystem`. Die Komponenten, welche die Maschinenmodelle repräsentieren, enstehen im Allgemeinen durch Codeexport (siehe Abschnitt 3.2.3). Dabei werden je nach Ansatz die Schnittstellen `OdeSystem` bzw. `DaeSystem` implementiert. Eine weitere Unterteilung ist aufgrund der abgegrenzten Problemstellung nicht notwendig.

Der Funktionsaufruf zum Berechnen der Modellgleichungen (Gleichung 2.3 bzw. Gleichung 2.4) wird durch einen entsprechenden Methodenaufruf abstrahiert. Diese Abstraktion wird von den Komponenten der numerischen Methoden zur Berechnung eines Integrationsschrittes benutzt (siehe Abschnitt 2.1, Abbildung 2.4).

Die Löser werden analog zu den Modellgleichungen in Methoden für Systeme gewöhnlicher Differentialgleichungen und differential-algebraischer Gleichungen unterschieden. Die Synchronisation der Berechnung erfolgt durch die Implementation der

Schnittstellen `Executable` und `Resettable` (Abbildung 4.15). Diese abstrahieren die Ausführung eines Integrationsschrittes und das Zurücksetzen der Simulation in den konfigurierten Anfangszustand.

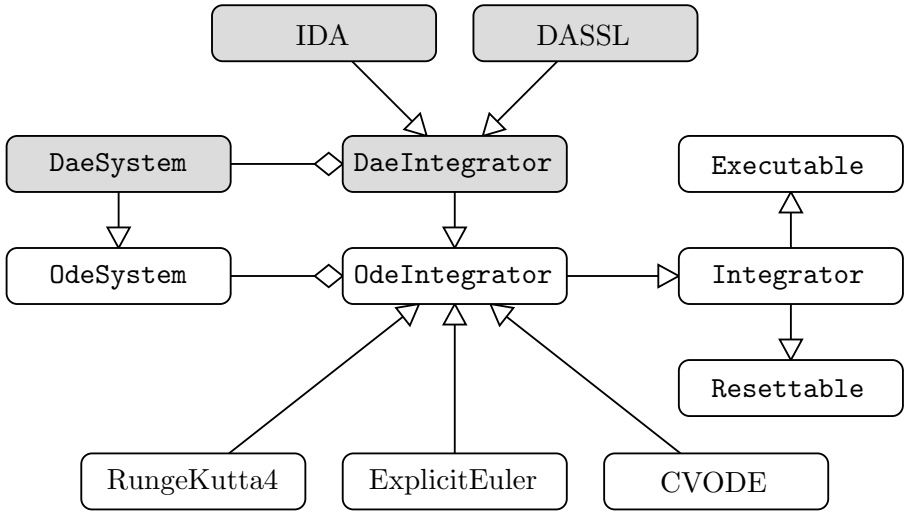


Abbildung 4.15: Komponenten und Schnittstellen für Modellbildung und Simulation

Jeder Integrator liegt als eigenständige Komponente vor. Die Spezialisierung von `OdeSystem` und `OdeIntegrator` zu `DaeSystem` und `DaeIntegrator` erlaubt prinzipiell die Berechnung eines Systems gewöhnlicher Differentialgleichungen mit einer numerischen Methode, die auch zur Berechnung differential-algebraischer Gleichungen geeignet ist.
Neben der Implementation des expliziten Euler-Verfahrens sowie des Runge-Kutta-Verfahrens 4. Ordnung (siehe Abschnitt 2.1, Abbildung 2.3) wurden noch die Integratoren DASSL ([BCP95]) sowie CVODE ([CH96]) und IDA ([Hin+05]) implementiert. Diese beruhen auf komplexeren Integrationsvorschriften und erfordern einen höheren Implementationsaufwand. Die Integration dieser Verfahren in Sarturis verdeutlicht die Vorteile der Komponentenarchitektur.
CVODE und IDA sind in der Sundials-Bibliothek implementiert ([Sun]). Diese wird durch das entsprechende Sarturis-Modul benutzt. Die Details der Implementierung betreffen nur dieses spezielle Modul. Die Schnittstelle der genutzten Bibliothek ist aufgrund des abgegrenzten Funktionsumfangs sehr stabil, so dass der Aufwand für die Pflege der Sarturis-Komponenten und des Moduls minimiert wird.

Der DASSL-Algorithmus liegt öffentlich als FORTRAN-Code vor ([Net]). Dieser kann automatisch in C-Code umgewandelt werden (siehe [Dro04]). Das Ergebnis dieser Umwandlung ist aufgrund der Codestruktur sehr schwierig zu handhaben und damit als Bestandteil größerer Softwareprojekte ungeeignet. Im Rahmen der Komponentenarchitektur treten die Nachteile dieser Vorgehensweise nur innerhalb der Komponente und des zugehörigen Moduls auf. Dadurch wird das Problem der Integration des umgewandelten C-Codes überschaubar.

## 4.4 Komponenten des VR-Systems

### 4.4.1 Ein- und Ausgabegeräte

Bereits mit Joysticks, Gamepads und anderen Ein- und Ausgabegeräten aus dem Bereich des Consumer- und Spielemarktes können preiswerte und flexible Bedienumgebungen für die interaktive Simulation mobiler Arbeitsmaschinen aufgebaut werden. Dabei kann die Hardware sowohl als Ersatz für Originalbedienelemente zur Interaktion mit dem Modell genutzt werden als auch z. B. zur Navigation innerhalb der virtuellen Umgebung.
Die Consumerhardware wird durch entsprechende Bibliotheken des Betriebssystems bzw. erweiterte APIs (z. B. DIRECTX) angesteuert. Die Komponenten enthalten für jede Plattform in der Regel einen eigenen Codepfad, bieten aber jeweils neutrale Schnittstellen. Ein charakteristisches Merkmal ist, dass jedes physische Gerät durch eine Instanz repräsentiert wird. Diese besitzt eine eigene Menge von analogen und digitalen Schnittstellen, welche direkt mit anderen Objekten verbunden werden können (Abbildung 4.16).
Für die Integration realer Bedienumgebungen in ein VR-System ist die Adaption industrieller Bussysteme (z. B. CAN, Interbus, Profibus) notwendig. Diese gestatten die Ansteuerung und Abfrage von entsprechenden Bedienteilen und Anzeigen realer Maschinen. Als Beispiel für die Adaption eines solchen Bussystems wurden in SARTURIS entsprechende Komponenten für den CAN-Bus implementiert. Die grundsätzlichen Überlegungen gelten für alle nachrichtenbasierten Feldbussysteme.
Im Unterschied zur Consumerhardware werden die Geräte nicht als einzelne gekapselte Einheiten angesteuert, sondern durch Nachrichten auf dem entsprechenden Bus. Dabei wird zwischen einer geräteorientierten und inhalts- bzw. objektorientierten Adressierung unterschieden ([SW06; ZS07]). Der Bus ist durch spezielle Hardware mit dem Rechner verbunden. Die notwendigen Gerätetreiber werden durch entsprechende

Programmierschnittstellen angesteuert. Diese wiederum bilden die Grundlage für die entsprechenden SARTURIS-Komponenten (Abbildung 4.17).

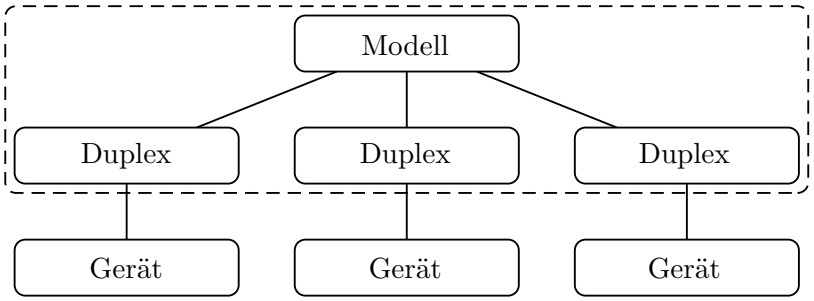


Abbildung 4.16: Direkte Anbindung an IO-Schnittstellen

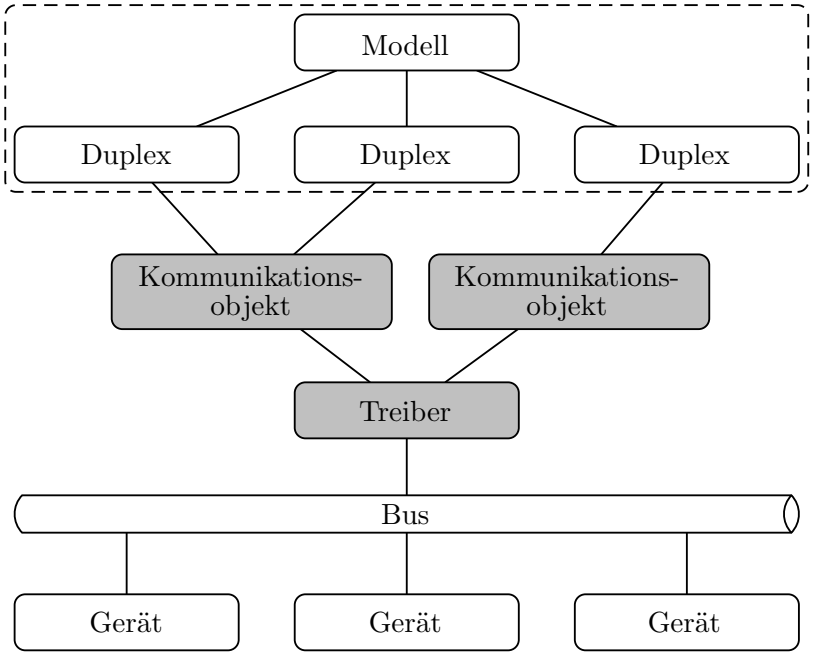


Abbildung 4.17: Indirekte Anbindung an IO-Schnittstellen

Bei einer inhaltsorientierten Nachrichtenadressierung sind die Nachrichten konzeptbedingt nicht einem reellen Gerät zuzuordnen. Jedes angeschlossene Gerät erhält jede Nachricht. Die Verarbeitung der Nachrichten erfolgt durch die implementierte Gerätesoftware entsprechend des aktuellen Systemzustands. Im einfachsten Fall wird eine Nachricht einfach verworfen.

Dieses Konzept wird in SARTURIS durch Kommunikationskomponenten abgebildet. Diese werden über die Schnittstellen `ReceptionHandler` und `TransmissionHandler` mit dem durch eine Treiberkomponente repräsentierten physischen Bus verbunden (Abbildung 4.18). Somit können Objekte implementiert werden, welche ihren Zustand auf der Basis mehrerer Nachrichten (die z. B. ein herstellerspezifisches Protokoll definieren) verändern. Ebenso können beliebig viele Nachrichten auf den Bus geschrieben werden, um analoge oder digitale Zustände anzuzeigen oder anderweitig zu verarbeiten.

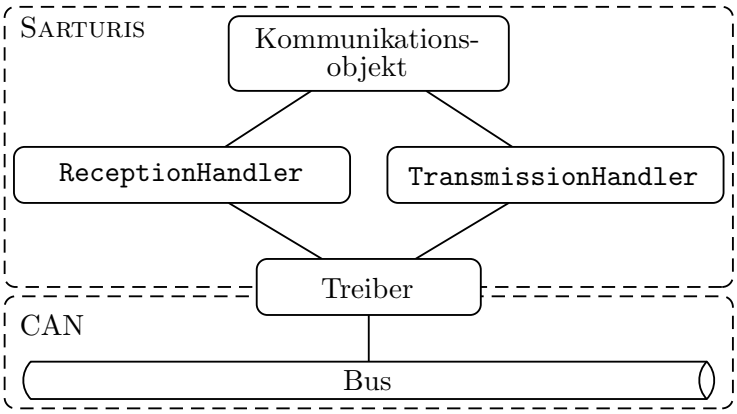


Abbildung 4.18: CAN-Bus, Schnittstellen

Auf diesem Weg lassen sich sowohl einzelne Geräte wie Anzeigen (Tacho, Drehzahlmesser, usw.) oder Bedienelemente (Joysticks, Tasten, usw.) ansteuern als auch Kommunikationskomponenten für ganze Kabinen entwickeln. Dies ist von Bedeutung, da aus dem Zustandsraum des Modells nur simulationsrelevante Informationen (Motordrehzahl, Fahrgeschwindigkeit, usw.) abgeleitet werden können. In der Regel benötigt die Steuerungssoftware einer kompletten Maschinenkabine aber noch zusätzliche Informationen, wie z. B. den Füllstand des Tanks oder die Außentemperatur. Da die Kabine für die interaktive Simulation in einer VR-Umgebung nicht mit den entsprechenden Sensoren verbunden ist, fehlen diese Informationen. Dies führt in der Steuerungssoftware der Kabine in der Regel zu Fehlerzuständen. Eine Anpassung der Steuerungssoftware ist nicht akzeptabel, so dass die fehlenden Sensoren durch interne Funktionen der Kommunikationskomponenten emuliert werden müssen.

### 4.4.2 Fenstersystem

Zur maus- und bildschirmgesteuerten Interaktion mit dem Benutzer aber auch zur Anzeige von Zuständen wird eine grafische Benutzerschnittstelle (GUI) eingesetzt. In den Anwendungsfällen der Modell- und Methodenentwicklung ist diese von zentraler Bedeutung, da hier in der Regel nicht in der VR-Umgebung entwickelt wird. Dadurch stehen die entsprechenden (Hardware-) Benutzerschnittstellen zur Interaktion mit dem Modell nicht zur Verfügung und müssen durch Ersatzelemente der grafischen Benutzerschnittstelle abgebildet werden (Abbildung 4.19).
Zusätzlich werden bei der Modell- und Methodenentwicklung erweiterte Möglichkeiten der Parametrisierung und Abstimmung der Algorithmen von Modell und Löser benötigt. Dies kann während der Simulation mit Hilfe von entsprechenden Anzeigen und Eingabefeldern durchgeführt werden.

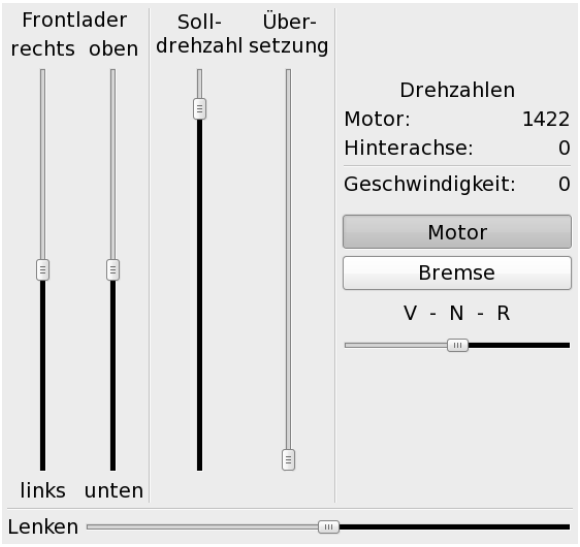


Abbildung 4.19: Bedienumgebung mit GUI-Elementen

Bei der Entwicklung von Sarturis wurde für die grafische Benutzeroberfläche das Gimp Toolkit (GTK) eingesetzt. Dieses ist frei verfügbar ([GTK]), entsprechend dokumentiert und portabel. Jedes Element der grafischen Oberfläche wird als Widget bezeichnet. Diese Widgets sind gegenseitig austauschbar, so dass komplexe grafische Oberflächen sehr strukturiert aufgebaut werden können ([War03]).
GTK selbst erfüllt die Anforderungen an eine Komponentenarchitektur. Das entsprechende Metamodell findet sich aber nicht in einer computergeeigneten Notation

wieder, so dass die Abbildung auf SARTURIS-Komponenten nicht automatisiert erfolgen kann.

Layoutelemente (Abbildung 4.20) wie Tabellen, Boxen oder Panels sind rekursiv aufgebaut und erlauben somit die Strukturierung der grafischen Oberfläche. Inhaltlich zusammengehörige Elemente können gruppiert werden, so dass eine Wiederverwendung komplexer Strukturen möglich ist. Dieser Ansatz unterstützt die Anwendung eines sehr feingranularen Komponentenkonzeptes mit hoher Flexibilität in der Anwendung, da die sich wiederholenden Beziehungen zwischen den Komponenten der grafischen Benutzeroberfläche durch die Konfiguration der Simulationsanwendung abgebildet werden können.

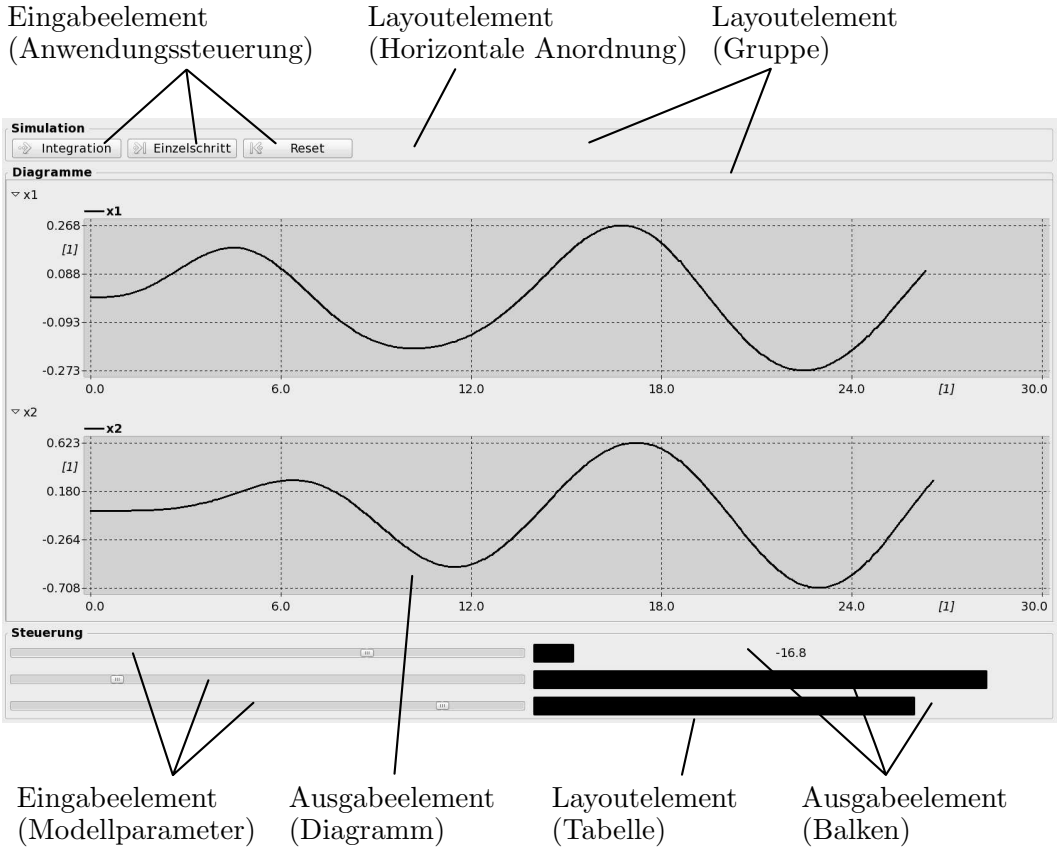


Abbildung 4.20: Elemente der grafischen Benutzerschnittstelle

Ein- und Ausgabeelemente wie Schalter, Schieberegler, Verlaufsbalken oder Diagramme implementieren neben der `Widget`-Schnittstelle zum Aufbau der grafi-

schen Benutzeroberfläche auch die entsprechenden Ein- und Ausgabeschnittstellen (Abbildung 4.21), so dass die Austauschbarkeit im Sinne der Komponentenarchitektur gegeben ist.

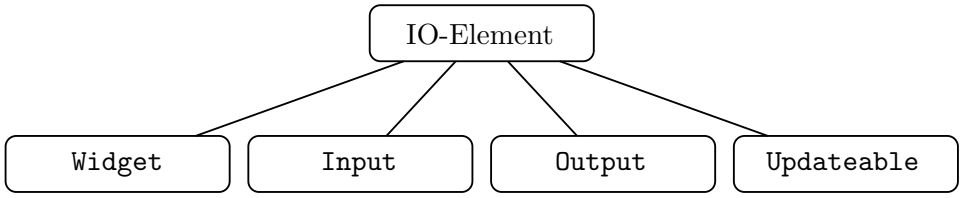


Abbildung 4.21: Implementierte Schnittstellen bei Ein- und Ausgabeelementen

Die Synchronisation spielt bei der Implementation der grafischen Benutzeroberfläche eine wesentliche Rolle. Die Aktualisierung mehrerer Diagramme, Verlaufsbalken, usw. ist sehr rechenintensiv und sollte daher nur mit der maximal notwendigen Frequenz erfolgen. Zusätzlich dazu erfordert die Aktualisierung von GTK-Elementen aus nebenläufigen Anwendungskontexten einen erhöhten Implementationsaufwand. Deshalb wurde für alle Ein- und Ausgabekomponenten der grafischen Benutzeroberfläche eine asynchrone Implementierung mit einem zusätzlichen Speicher und der Schnittstelle `Updateable` gewählt (Abbildung 4.22).

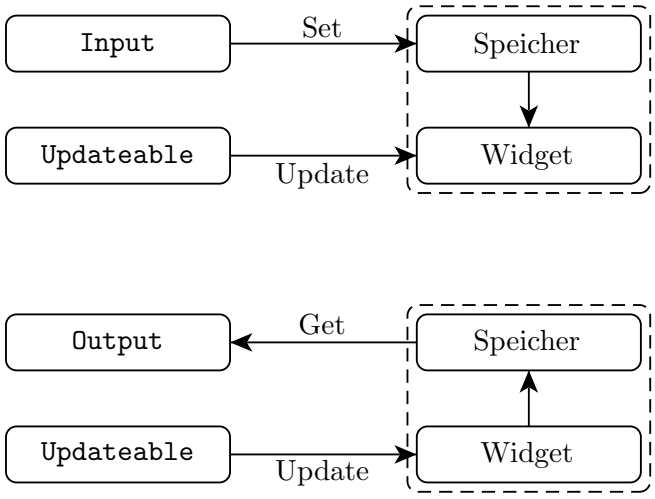


Abbildung 4.22: Synchronisation der grafischen Benutzerschnittstelle

Dieser Speicher ist mit der entsprechenden IO-Schnittstelle verbunden und kann durch assoziierte Komponenten gelesen und geschrieben werden. Dies kann auch in

parallelisierten Anwendungen erfolgen. Der notwendige Schutz der dabei auftretenden kritischen Abschnitte erfolgt komponentenintern.
Die Synchronisation des Speichers mit der grafischen Oberfläche wird durch die Implementation des Interfaces `Updateable` realisiert. Dadurch kann der Zeitpunkt der Aktualisierung durch die Konfiguration beeinflusst werden und ist von der Simulation unabhängig.

### 4.4.3 Szenegraph und Visualisierung

Die Visualisierung in VR-Systemen wird gewöhnlich durch Szenegraphen realisiert. Dies sind Datenstrukturen, welche die Ausprägung (Geometrie, Textur, Beleuchtung) und Position aller Objekte einer Szene beschreiben. Perspektive, Position und Blickrichtung des Betrachters werden ebenfalls in entsprechenden Objekten gespeichert. Die Darstellung der Objekte erfolgt durch Traversieren des Baumes mit sogenannten Visitoren. Diese analysieren die Datenstruktur und benutzen die eigentlichen Grafik-API (z. B. OpenGL) zur Darstellung der Szene. Der Szenegraph stellt somit eine objektorientierte High-Level-API zur Verfügung, mit deren Hilfe der Anwender ohne tiefere Kenntnis der Low-Level-Grafik-API qualitativ hochwertige Visualisierungslösungen erstellen kann.
Strukturiert wird der Szenegraph als einseitig gerichteter Graph (Abbildung 4.23). Die Objekte werden als Knoten bezeichnet und besitzen eine klar abgegrenzte Funktionalität. Knoten lassen sich gegenseitig austauschen. Jeder Graph kann Teil eines anderen werden. Damit erfüllt die Architektur von Szenegraphen allgemein die Anforderungen an ein Komponentensystem.
Aufgrund der Verfügbarkeit und des umfangreichen Leistungsumfangs ([See04]) bietet die freie Bibliothek OpenSceneGraph (OSG) die Basis für die Implementierung der folgenden Szenegraph- und Visualisierungskomponenten:

- **Geometrieknoten** speichern die in Form von Polygonen gegebenen Facetten eines geometrischen Objektes in dessen lokalem Koordinatensystem. Dabei werden für jeden Polygon-Eckpunkt sowohl kartesische Koordinaten als auch Informationen über Texturen und Beleuchtung gespeichert.
  Das geometrische Objekt ist entweder ein Bauteil der simulierten Maschine (Abbildung 4.23) oder Bestandteil der virtuellen Umgebung. Die Erstellung der geometrischen Objekte erfolgt in der Regel aus CAD-Geometrien, welche durch entsprechende Werkzeuge für den Einsatz in virtuellen Umgebungen aufbereitet werden müssen ([PMT03]). Die resultierenden Geometrien werden

in standardisierten Dateiformaten gespeichert und durch die Geometrieknoten in SARTURIS aus dem Dateisystem geladen.

- **Transformationsknoten** dienen der Positionierung von Objekten in einer Szene. Dabei werden die lokalen Koordinaten in das globale Koordinatensystem transformiert. Die Trennung von Geometrie und Transformation erlaubt die mehrfache Verwendung eines Geometrieknotens für Exemplare des gleichen Bauteils (Abbildung 4.23). Dadurch reduzieren sich Speicherbedarf und Berechnungsaufwand der kompletten Szene.
  Die Transformationsknoten stehen in Beziehung mit einem `PositionInput`, welches die Koordinatentransformation in Form einer Translation und Rotation bereitstellt.

- **Gruppenknoten** fassen Knoten zusammen und gestatten die Strukturierung des Szenegraphen. Durch Mehrfachverwendung der Knoten können Rechenleistung und Aufwand bei der Konfiguration des Szenegraphen eingespart werden.

- **Kameraknoten** speichern das Sichtfeld und die Perspektive des Betrachters in einem lokalen Koordinatensystem. Durch die Zuordnung des Kameraknotens an einen Transformationsknoten kann der Betrachter durch die Szene navigieren. Erfolgt diese Navigation im Kontext des Maschinenbedieners, so kann der Kameraknoten mit einer durch die Maschinenbewegung gesteuerten Transformation verbunden werden.
  Die Abbildung einer zusätzlichen Überlagerung z. B. aus einem Head-Tracking-System ([BR96; CSD93; MJR03]) ist aufgrund der abgebildeten rekursiven Komponentenstruktur durch Konfiguration der Simulationsanwendung möglich.

- **Supportknoten** werden für das Einrichten der virtuellen Umgebung sowie die Fehlersuche benötigt. Diese Knoten stellen keine Geometrie im eigentlichen Sinne dar, sondern symbolisieren z. B. die Kameraeinstellungen oder die Position der Projektionsflächen.

- **Dynamische Knoten** repräsentieren geometrische Objekte mit Zusatzinformationen aus der Simulation. Beispiele für diese Objekte sind Pfeile für die Darstellung von Vektoren (z. B. für Geschwindigkeiten oder Kräfte) oder in das Sichtfeld projizierte Anzeigen (z. B. für Drücke, Drehzahlen oder Temperatu-

ren). Bei diesen Knoten ist die Geometrie während der Simulation permanent zu aktualisieren.

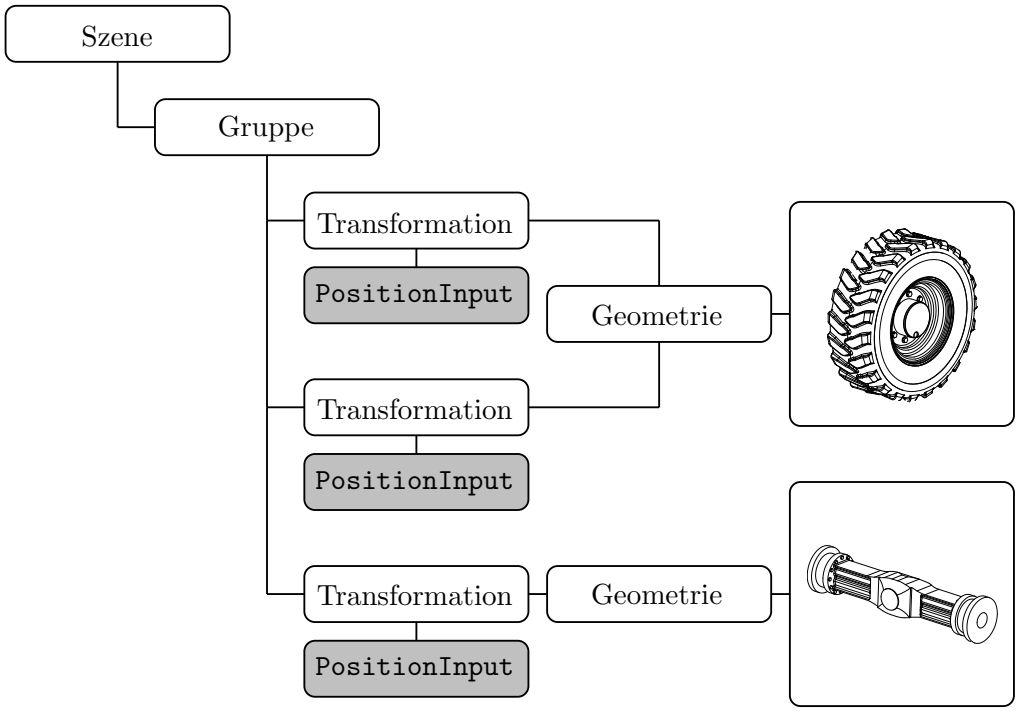


Abbildung 4.23: Szenegraphkomponenten

Analog zu der in Abschnitt 4.4.2 beschriebenen asynchronen Ansteuerung der Elemente der grafischen Benutzeroberfläche erfolgt die Aktualisierung des Szenegraphen durch die entsprechende Implementation des Interfaces `Updateable` an den Transformations-, Kamera- und dynamischen Knoten. Auf diese Weise sind insbesondere Effekte der Reihenfolge der Aktualisierung der einzelnen Knoten abhängig von der Konfiguration der Simulationsanwendung und nicht durch die spezifische Implementation einer speziellen Komponente fixiert.
Ein Aspekt bei der grafischen Darstellung von Mehrkörpersystemen ist die Unterscheidung von Absolut- und Relativkinematik. Bei der Anwendung der Absolutkinematik wird die Transformation für jedes Objekt in absoluten Koordinaten angegeben. Im Szenegraphen existieren keine Hierarchiebeziehungen zwischen den Objekten. Alle dem Mehrkörpersystem zugeordneten Transformationsknoten befinden sich auf einer Ebene (Abbildung 4.23).

Alternativ dazu wird bei der Relativkinematik ausgenutzt, dass Mehrkörpersysteme mobiler Arbeitsmaschinen häufig in Baumstrukturen abgebildet bzw. in solche überführt werden können ([Sch88; FS96; WH82]). Die Beschreibung der Bewegung eines Objektes erfolgt dann relativ zum übergeordneten Objekt. Die Transformation beschreibt relative anstelle absoluter Koordinaten. Szenegraphen bilden dieses Konzept aufgrund ihrer Strukturierung sehr gut ab. Da Geometrie- und Transformationsknoten gleichberechtigt einem Knoten zugeordnet werden können, eignen sich nacheinander angeordnete Transformationsknoten (Abbildung 4.24) zur Abbildung einer Relativkinematik.

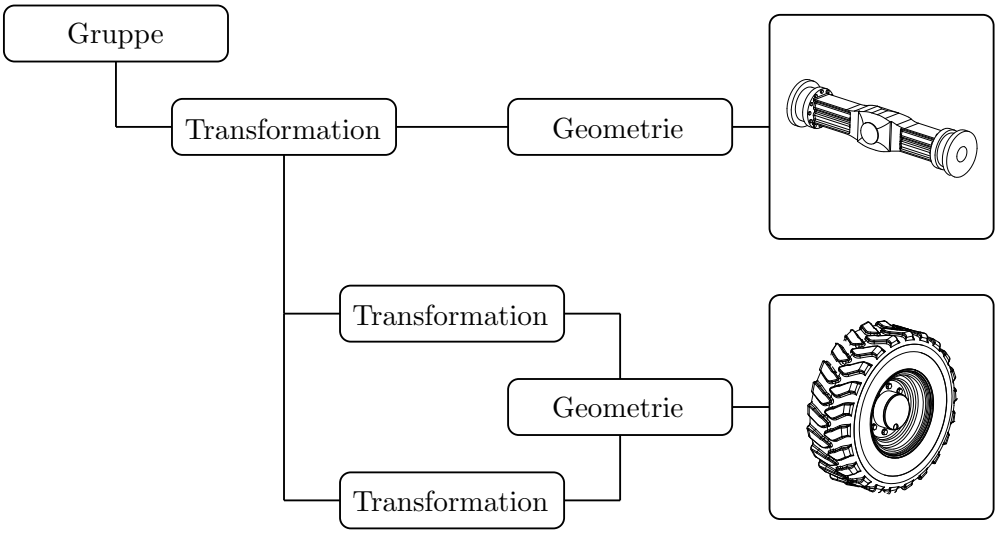


Abbildung 4.24: Relativkinematik im Baum des Szenegraphen

Der Vergleich beider Konzepte hinsichtlich der Visualisierung zeigt weder signifikante Vorteile für die Absolut- noch für die Relativkinematik. Die Anzahl der Transformationsknoten ist gleich, da in beiden Fällen jedem Bauteil ein solcher Knoten zugeordnet ist unabhängig davon, ob die Informationen eine absolute oder relative Bauteilposition beschreiben. Die Anzahl der notwendigen Rechenoperationen ist praktisch identisch, da die Transformationsknoten generisch sind. Potenzielle Einsparungen, die sich bei der Relativkinematik daraus ergeben, dass Bewegungen meist entlang einer Koordinatenachse definiert sind, werden prinzipbedingt (bei der Visualisierung) nicht erzielt.
Die Anwendung der Relativkinematik setzt eine strukturelle Übereinstimmung des Szenegraphen und des MKS-Modells voraus. Nur dann beschreiben die lokalen Trans-

formationen im MKS-Modell auch die relativen Transformationen im Szenegraphen. Diese Forderung führt dazu, dass bei der Erstellung und Modifikation des MKS-Modells die Struktur der Visualisierung beachtet werden muss. Diese Forderung besteht bei der Absolutkinematik nicht, was die Flexibilität der Anwendung deutlich erhöht. Aus diesem Grund wurde dieses Konzept in SARTURIS implementiert.
Die endgültige Darstellung der Visualisierung wird durch das Fenstersystem realisiert. Dafür wird eine `Viewer`-Komponente implementiert, welche die Packages `sarturis-gtk` und `sarturis-osg` integriert. Die dabei entstehende zusätzliche Abhängigkeit kann in beiden Richtungen abgebildet werden. In der vorliegenden Implementation wurde das Package `sarturis-gtk` als grundlegender bewertet (siehe Abbildung 4.2).
Bei der Anwendung der Komponente zeigt sich, dass zwischen einer Vollbildvisualisierung, wie sie typischerweise in VR-Umgebungen eingesetzt wird und einer Fenstervisualisierung auf einem Desktop unterschieden werden muss. Die Vollbildvisualisierung in der VR-Umgebung erfolgt mit einer durch die Visualisierungshardware (Projektoren) vorgegebenen sehr hohen Auflösung. Die Anforderungen an die Qualität der Darstellung und Effekte sind maximal. Im Gegensatz dazu wird die Fenstervisualisierung häufig bei der Modell- und Methodenentwicklung eingesetzt. Die Auflösung kann deutlich niedriger sein als in einer VR-Umgebung. Anforderungen an grafische Effekte sind untergeordnet. Vielmehr werden Abstraktionen wie Drahtgittermodelle oder die Darstellung von wirkenden Kräften oder resultierenden Geschwindigkeitsvektoren benötigt.
Diese grundsätzlich verschiedenen Anforderungen spiegeln sich im Entwurf und der Implementation der Integration von OSG und GTK wieder. Es ist ohne weiteres möglich, ein GTK-Fenster ohne Dekoration im Vollbild-Modus darzustellen. In dem Fall implementiert der `GtkViewer` das Interface `gtk::Widget` und kann einem Fenster zugeordnet werden (Abbildung 4.25a). Eine Zuordnung zu einem Layoutelement im Sinne einer Integration in eine komplexe grafische Benutzeroberfläche ist ebenso denkbar. Dieser Entwurf ist sowohl für die Anwendung auf einem Desktop-PC als auch für den Einsatz in der VR-Umgebung geeignet, da die Vollbildfunktion durch GTK realisiert wird.
Es zeigt sich aber, dass bei höheren Auflösungen die Leistungsfähigkeit nicht ausreichend ist, um eine akzeptable Bildwiederholrate zu erzielen. Offensichtlich führt GTK bei der Darstellung von OPENGL-Inhalten Zusatzoperationen aus, welche die Grafikleistung negativ beeinflussen. Aus diesem Grund wird für die Vollbilddar-

stellung auf den nativen `osg::Viewer` von OSG zurückgegriffen (Abbildung 4.25b). Dieser ist wesentlich leistungsfähiger als ein GTK-Fenster mit Vollbilddarstellung und läuft als eigenständiger Anwendungskontext in einem eigenen Thread.

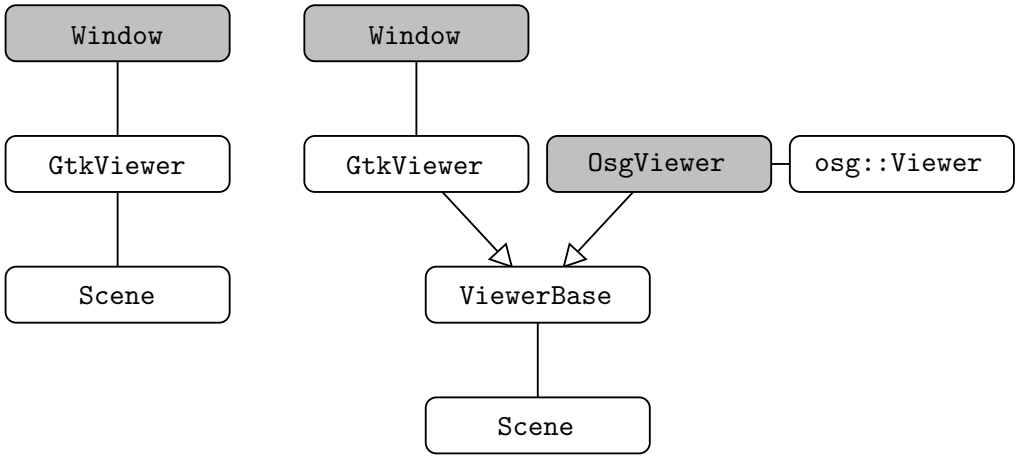


Abbildung 4.25: Integration des Szenegraphen in das Fenstersystem

Die im Rahmen dieser Arbeit präsentierte Visualisierung ist aus computergrafischer Sicht auf die Grundfunktionen begrenzt, welche man von einem Szenegraphen erwartet. Diese erfüllen die Mindestanforderungen an die grafische Darstellung innerhalb eines VR-Systems. Die flexible Komponentenarchitektur von Sarturis erlaubt die Integration zusätzlichen Expertenwissens ([Gro08; GW11]) in das Gesamtsystem.
Die Problemdomäne wird auf die Implementation eines Szenegraphen unter Ausnutzung des aktuellen Standes der Technik eingegrenzt. Zusätzlich muss die Integration in das Komponentensystem von Sarturis erfolgen. Diese ist weitestgehend automatisiert (Abschnitt 4.2.3) und erfordert neben der Spezifikation der Komponenten keinen Zusatzaufwand. Die Schnittmenge zum Rest des Simulationssystems wird durch die Implementation bzw. Nutzung der gegebenen Schnittstellen gebildet. Die in Abschnitt 4.1.2 definierten Schnittstellen zur Synchronisation und zum Datenaustausch sind bereits Bestandteil der Problemdomäne des Szenegraphen und erfordern keinen zusätzlichen Implementationsaufwand.
Im Ergebnis dieser Arbeiten konnten deutlich leistungsfähigere Visualisierungskomponenten implementiert werden (Abbildung 4.26). Die Implementation dieser Komponenten benutzt weiterhin die freie Bibliothek OpenSceneGraph unter Anwendung aktueller Technologien auf dem Gebiet der Computergrafik (Shader, etc.). Die

wesentlichen Schnittstellen für die Integration der Visualisierung in die Simulationsanwendung (Positionierung der Bauteile, Integration in das Fenstersystem, usw.) bleiben erhalten. Die Entwicklung der leistungsfähigeren Visualisierungskomponenten erfolgte autark mit eigenen Testfällen. Die Übertragbarkeit in das Gesamtsystem ist durch den komponentenbasierten Entwicklungsansatz gegeben.

Abbildung 4.26: Visualisierung in SARTURIS, 2008 und 2011

### 4.4.4 Soundsystem

Für die Qualität der Gesamtsimulation spielt die akustische Wahrnehmung eine wichtige Rolle ([Neg07]). Unmittelbar mit der Arbeitsaufgabe und Maschinensimulation verbundene Geräusche, wie z. B. das drehzahlabhängige Motorgeräusch sowie entsprechende Rückmeldungen beim Erreichen mechanischer Anschläge sind wesentlich für die Immersion des Bedieners. Dabei ist nicht die absolute Qualität der Geräusche entscheidend als vielmehr deren Vorhandensein überhaupt. Die aktuelle Implementation am vorhandenen Fahrsimulator zeigt, dass bereits einfache, durch in die Fahrerkabine integrierte Lautsprecher abgespielte Geräuschaufzeichnungen ausreichen, um den gewünschten Eindruck zu erzielen. Die Nachbildung der physikalischen Ursache der Geräusche (z. B. Körperschallabstrahlungen der mechanischen Struktur) ist nicht notwendig.

Die Implementation des Soundsystems gestaltet sich daher entsprechend einfach. Als plattformneutrale Basis wurde OPENAL verwendet. Dessen Ansatz ist mit OPENGL für die Visualisierung vergleichbar. Aufgrund der deutlich geringeren Komplexität des Soundsystems ist ein dem Konzept des Szenegraphen vergleichbarer Ansatz nicht notwendig.

Geräuschquellen werden durch ein Sample dargestellt (Abbildung 4.27). Die Position der Geräuschquelle wird durch ein zugeordnetes `PositionInput` beschrieben. Die Position des Zuhörers (Bedieners) ist ebenfalls variabel, so dass die Entfernung zur Geräuschquelle bei der Berechnung der Lautstärke berücksichtigt wird. Nicht berücksichtigt wird der Doppler-Effekt. Dieses ist aufgrund der mit mobilen Arbeitsmaschinen erreichbaren Geschwindigkeiten und den Zielen der interaktiven Simulation legitim.

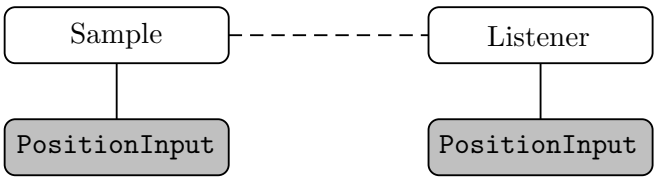


Abbildung 4.27: Geräuschquelle und Zuhörer

### 4.4.5 Bewegungssteuerung

Der als Referenzsystem implementierte Fahrsimulator verfügt über ein, nach dem Prinzip der *Stewart-Plattform* aufgebautes Bewegungsystem. Dieses ist in der Lage, mit der Simulatorkabine räumliche Bewegungen auszuführen. Die abbildbaren Frequenzen, Beschleunigungen und Lagen sind durch die mechanischen Grenzen des Bewegungssystems und die Masse der zu bewegenden Kabine begrenzt.
Allgemeine Anforderungen an Bewegungssysteme von Fahrsimulatoren sowie deren Anwendungen werden in der Regel für Straßenfahrten von PKW und Nutzfahrzeugen ([Gre05; Fis09; SGP03; Sch05; Slo08; Wil03]) definiert. Die Diskussion der Anwendung bei Geländefahrten ([For03; HB03; TF01]) zeigt die prinzipelle Eignung einer *Stewart-Plattform* zur Darstellung der Fahrzeugbewegungen mobiler Arbeitsmaschinen. Dies gilt insbesondere unter Berücksichtigung der erreichbaren Geschwindigkeiten.
Eine tiefergehende Auseinandersetzung mit der Bewegungssimulation ist nicht Gegenstand dieser Arbeit. Vielmehr wird das vorhandene Bewegungssystem als Black-Box mit definierter Schnittstelle behandelt. Die kontinuierlich an das Bewegungssystem übertragenen Daten werden aus dem Zustandsraum des MKS-Modells berechnet (Abbildung 4.28) und mit Kontroll- und Steuerdaten aus einer Komponente (Abbildung 4.29) für die grafische Benutzeroberfläche (Abschnitt 4.4.2) kombiniert. Zur Abstimmung des Systems können sowohl synthetisch generierte Daten als auch Messdaten eingespielt werden ([Zor11]).

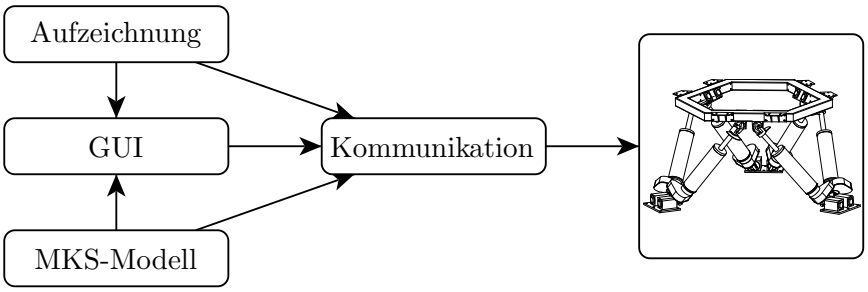


Abbildung 4.28: Ansteuerung des Bewegungssystems

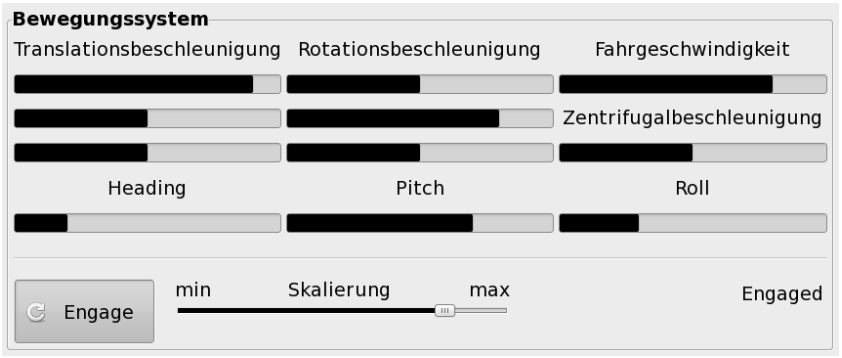


Abbildung 4.29: GUI-Komponente der Bewegungssystemsteuerung

## 4.5 Schnittstellen

### 4.5.1 Kommunikation im lokalen Netzwerk

Aufgrund der Hardwareanforderungen werden VR-Systeme als verteilte Systeme realisiert. Dabei wird die Funktionalität auf mehrere Prozesse verteilt, welche einer beliebigen Anzahl von Prozessoren zugeordnet werden können ([Ben+08; TS08]). Für den Anwender ist die Verteilung des Systems transparent. Die Interaktion erfolgt wie mit einem einzelnen geschlossenen System.

Da die einzelnen Prozesse über keinen gemeinsamen Speicher verfügen, erfolgt die Kommunikation und Synchronisation durch Nachrichten. Der Austausch der Nachrichten wird in SARTURIS durch Netzwerkkomponenten realisiert. Aufgrund der spezifischen Anforderungen wird auf den primären Einsatz von Middleware oder Remote Procedure Calls verzichtet. Diese generischen Ansätze sind im Wesentlichen für eine fehlertolerante Kommunikation in nicht-lokalen Netzen konzipiert. Der dafür

notwendige Zusatzaufwand reduziert die Leistungsfähigkeit für eine Kommunikation unter Echtzeitbedingungen.
Das VR-System ist im Allgemeinen an einen spezifischen Ort gebunden. Dies gilt insbesondere für Fahrsimulatoren. Die Verteilung der Software auf mehrere Prozesse erfolgt in lokalen Netzwerken ([RS06]). Die Anforderungen an die Datensicherheit werden durch die Regelung des Zugangs zum VR-System erfüllt. Alle für die Simulation relevanten Daten werden periodisch neu erzeugt, d. h. zu jedem Synchronisationspunkt liegt eine eigenständige Menge gültiger Daten vor. Der Verlust weniger Nachrichten kann toleriert werden, da die Daten innerhalb kurzer Intervalle aktualisiert und neu verteilt werden.
In der Regel werden Daten von mehreren Prozessen benötigt. Zwischen Erzeuger und Verbraucher existiert eine 1:n-Beziehung. Diese durch 1:1-Beziehungen auf der Kommunikationsebene abzubilden, reduziert die Leistungsfähigkeit des Gesamtsystems, da der Erzeuger mit einer entsprechenden Anzahl von Kommunikationspartnern interagieren muss (Abbildung 4.30).

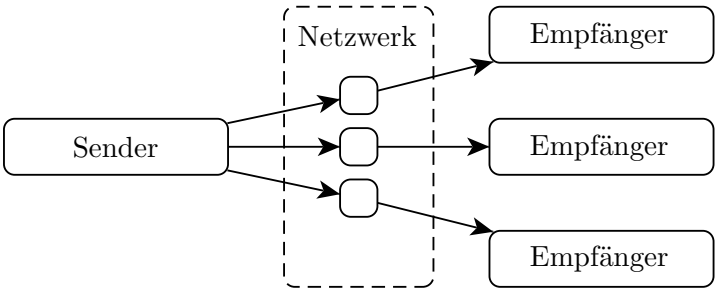


Abbildung 4.30: Serielles Senden einer Nachricht an mehrere Prozesse

Für die lokale Kommunikation im VR-System ist daher eine einfache Netzwerkschicht geeignet. Diese basiert auf dem UDP-Protokoll ([Tan03]). Der Datenaustausch erfolgt durch Pakete, welche als Datagramme bezeichnet werden. Wesentlich für die Leistungsfähigkeit der Kommunikation ist die Möglichkeit, Nachrichten per Broadcast an alle mit dem Netzwerk verbundenen Prozesse zu senden (Abbildung 4.31). Die Verteilung der Nachricht an alle potenziellen Empfänger wird durch die Netzwerkhardware realisiert. Dadurch wird der sendende Prozess entlastet und die Empfänger erhalten die Nachrichten nahezu zeitgleich.

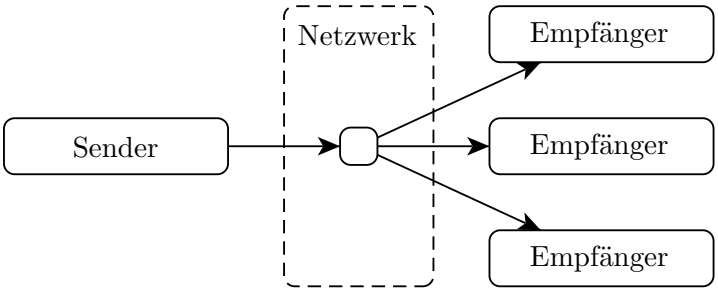


Abbildung 4.31: Senden einer Nachricht per Broadcast

Eine 1:1-Abbildung der Schnittstellenobjekte auf Nachrichten ist aufgrund der Größe der Daten nicht sinnvoll. Für jedes Datagramm entsteht zusätzlicher Aufwand, welcher die Leistungsfähigkeit des Systems reduziert. Aus diesem Grund werden einer Nachricht mehrere Schnittstellenobjekte zugeordnet. Die Trennung der Daten erfolgt durch eine eindeutige Kennung (Abbildung 4.32). Der Aufbau der Nachrichten und die Zuordnung zu den Schnittstellen werden in der Konfiguration definiert.

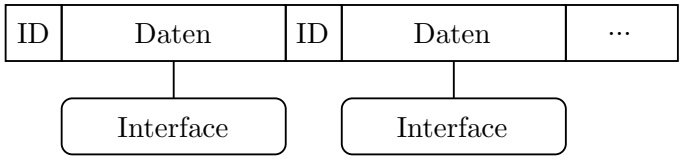


Abbildung 4.32: Aufbau einer lokalen Netzwerknachricht

### 4.5.2 Middleware und Remote Procedure Calls

Generische Ansätze für die Kommunikation in verteilten Anwendungen sind Middleware ([Vin97]) und Remote Procedure Calls ([Ben+08]). Die Implementierung des Nachrichtenaustauschs wird von der Verwendung der Informationen getrennt. Dies führt zu erheblichen Vereinfachungen bei der Realisierung verteilter Systeme. Durch entsprechende Standardisierung der Nachrichtenübertragung können verteilte Systeme plattformneutral aufgebaut werden.
Aufgrund des generischen Ansatzes und dem Anspruch einer Vielzahl von Anwendungsfällen zu genügen, unterscheiden sich die Anforderungen an entsprechende Implementierungen fundamental von denen der lokalen Netzwerke von VR-Umgebungen. Von Interesse sind Middleware und Remote Procedure Calls vor allem für

die Kommunikation über die Grenzen des lokalen Netzwerks einer VR-Umgebung hinaus (Abbildung 4.33).

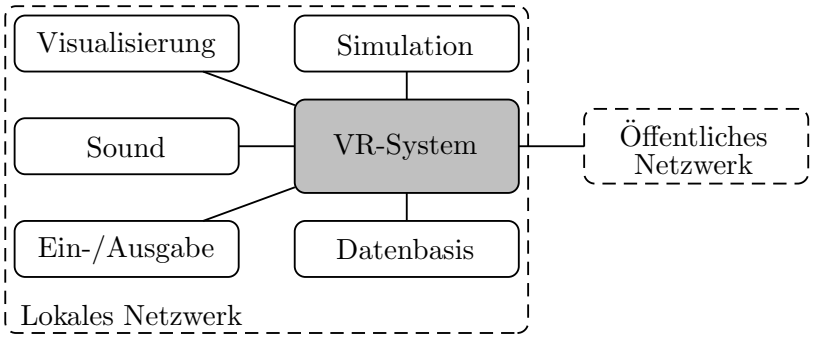


Abbildung 4.33: Kommunikation über die Grenzen des VR-Systems

Mit dem CORBA-Standard ist eine objektorientierte Middleware definiert ([And04; SGM02; Vin97]). Die Schnittstellen werden in CORBA mittels der INTERFACE DEFINITION LANGUAGE spezifiziert. Daraus wird durch einen Compiler Objektcode für den Server und den Client generiert (Abbildung 4.34).

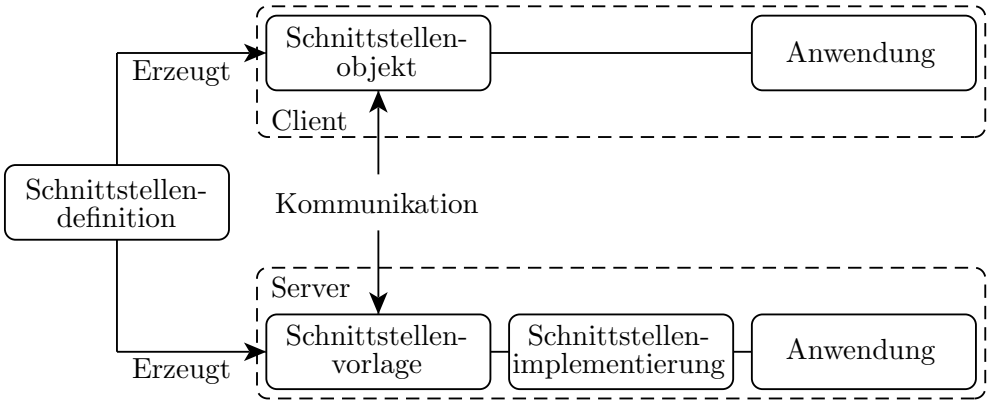


Abbildung 4.34: Schnittstellenobjekte in CORBA

Die generierten Objekte kapseln den Nachrichtenaustausch und können in der Anwendungslogik durch einfache Methodenaufrufe benutzt werden.
Die generischen CORBA-Komponenten in SARTURIS verbinden die Schnittstellenobjekte (siehe Abschnitt 4.1.2) mit einer CORBA-Implementierung. Dafür werden die entsprechenden Definitionen in der INTERFACE DEFINITION LANGUAGE notiert (Abbildung 4.35). Die generierten Objekte werden für die Implementation der

CORBA-Komponenten verwendet. Diese ermöglichen die Kommunikation einer SARTURIS-Anwendung mit einem verteilten System (Abbildung 4.36).

```
1 interface AnalogInput {
2   double getValue();
3 };
4 interface AnalogOutput {
5   void setValue(in double value);
6 };
7 interface Executable {
8   void exec();
9 };
```

Abbildung 4.35: Interfacedefinitionen für die CORBA-Komponente

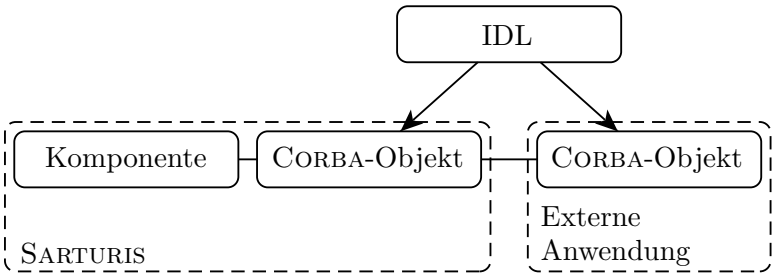


Abbildung 4.36: Kommunikation zu einem verteilten System

Der CORBA-Standard wird kontrovers diskutiert ([Hen06]). Dabei stehen sowohl technische als auch organisatorische Aspekte im Mittelpunkt der Diskussion. Aus der Perspektive dieser Arbeit demonstrieren die CORBA-Komponenten die prinzipielle Vorgehensweise bei der Adaption von Middleware.
Zum Austausch von Nachrichten in verteilten Systemen sind leichtgewichtigere Standards, wie z. B. XML-RPC ebenso geeignet ([Eng+00; Gri02]). XML-RPC benutzt für den Nachrichtentransport das HTTP-Protokoll. Die Nachricht sowie die Metainformationen werden in einem XML-Format übermittelt. Durch die Anwendung dieser beiden Standards vereinfacht sich die Implementation. Eine Spezifikation der Schnittstellen und die Generierung von Code sind nicht notwendig.

### 4.5.3 Virtual Reality Frameworks

Der Aufbau von Schnittstellenkomponenten zu anderen VR-Systemen wird am Beispiel des frei verfügbaren Frameworks VRJuggler beschrieben ([Bie+01; VRJ]). Dieses Framework unterstützt die Entwicklung komplexer VR-Anwendungen unter Abstraktion des eingesetzten VR-Systems. Ein- und Ausgabeschnittstellen werden ebenso durch Konfigurationen beschrieben wie Projektionsflächen und die Audioausgabe. Eine VRJuggler-Anwendung kann dadurch sehr flexibel in verschieden ausgestatteten VR-Systemen ausgeführt werden.
Gegen eine Implementation des Fahrsimulators mit VRJuggler sprechen neben der Tatsache, dass eine konkrete Anwendung in C++ implementiert werden muss, die fehlenden Konzepte für die Modellbildung und Simulation. Die Beobachtung einer interaktiven Simulation in Sarturis in einem mit VRJuggler realisiertem VR-System ergibt einen interessanten Anwendungsfall für die Kombination von Sarturis und VRJuggler (Abbildung 4.37).

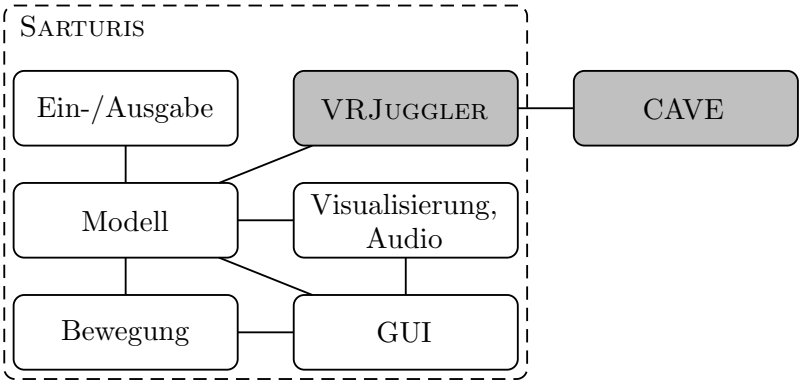


Abbildung 4.37: Beobachtung einer interaktiven Simulation

Die Architektur von VRJuggler definiert einen Kernel, der gemeinsam mit einer Implementation des Interfaces `vrj::Application` die Anwendung bildet ([Bri09]). Dabei implementiert der Kernel die Grundfunktionen und die `vrj::Application` die Anwendungslogik. Sowohl der Kernel als auch die Anwendungslogik können durch Sarturis-Komponenten dargestellt werden (Abbildung 4.38). Die Verknüpfung erfolgt somit nicht durch den in VRJuggler-Anwendungen notwendigen manuellen Programmcode ([Bri09]) sondern durch das Sarturis-Framework.

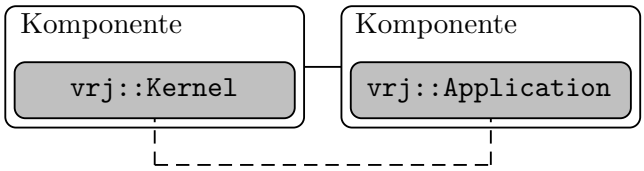


Abbildung 4.38: VRJuggler-Komponenten

Für die verteilte VRJuggler-Anwendung ist es dabei transparent, dass ein Knoten in eine Sarturis-Umgebung eingebettet ist (Abbildung 4.39). Für den Datenaustausch wird die Geräteabstraktion Gadgeteer im gemeinsamen Adressraum dieses Knotens genutzt.

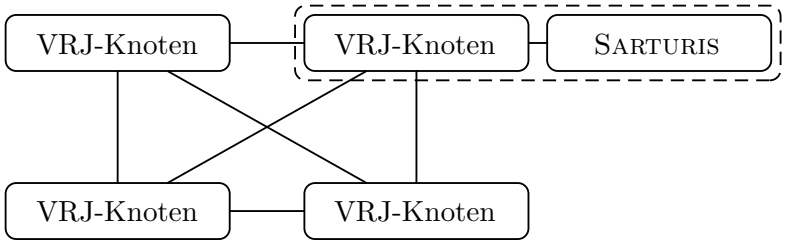


Abbildung 4.39: VRJuggler-Netzwerk mit Kopplung zu Sarturis

# 5 Referenzanwendung - Fahrsimulation eines Radladers

## 5.1 Allgemeines

Die Fahrsimulation eines Radladers dient als Referenzanwendung für das implementierte Softwaresystem. Diese kann entweder mit einer vereinfachten Bedienumgebung an einem Desktop-PC (siehe Abschnitt 4.4.1) oder in einem interaktiven Simulator (Abbildung 1.3) unter Verwendung einer Originalkabine ausgeführt werden.
Die virtuelle Welt besteht aus einer fiktiven Teststrecke, die klassische Elemente zur Bewertung des Fahrverhaltens (Schwellen, Haufwerke, etc. ) mobiler Arbeitsmaschinen enthält. Für die Teststrecke und den Radlader existieren entsprechende 3D-Modelle, mit denen die Visualisierungskomponente (siehe Abschnitt 4.4.3) konfiguriert ist. Die Positionierung der Bauteile des Radladers erfolgt anhand der Berechnung des kinematischen Modells.

Abbildung 5.1: Radlader und Teststrecke

Die kinematischen Beziehungen bilden die Basis für die symbolische Ableitung der Bewegungsgleichungen. Diese beschreiben gemeinsam mit den Gleichungen für das Antriebsmodell sowie einem vereinfachten Reifenmodell das Modell zur Simulation des Radladers. Dieses besteht aus einem System gewöhnlicher Differentialgleichungen (siehe Abschnitt 2.1) und wird durch Codeexport von MAPLE in eine SARTURIS-Komponente überführt (siehe Abschnitt 3.2.3).
Ein Prozessmodell für das Befüllen und Entleeren der Ladeschaufel ist nicht implementiert. Als Anwendungsfall für das Modell ist das interakive Fahren im Gelände definiert. Dabei kann jeder Punkt der virtuellen Welt angesteuert werden. Einschränkungen hinsichtlich spezieller Fahrspuren existieren nicht. Zur Variation der Bedingungen kann die Füllung der Ladeschaufel verändert werden. Die Masse des Erdstoffes wird dabei im Modell als Punktmasse angenommen und durch einen normierten Parameter skaliert. Dieser Parameter ist eine Schnittstelle des Modells und kann entweder durch die grafische Benutzeroberfläche oder durch den Bediener mit Hilfe eines Stellteils variiert werden. Zur visuellen Rückmeldung kann die eingestellte Gutmenge grafisch dargestellt werden.
Die Ansteuerung des Bewegungssystems erfolgt auf der Basis der berechneten Neigungen, Geschwindigkeiten und Beschleunigungen. Das zugehörige Koordinatensystem ist entsprechend der Einbausituation der Radladerkabine im Fahrsimulator definiert. Als akustische Signale werden die mechanischen Anschläge der Hydraulikzylinder sowie das drehzahlabhängige Motorgeräusch abgebildet.
Mögliche Entwicklungsszenarien für eine solche Anwendung sind der Test verschiedener Stellteile, die Analyse von Assistenzsystemen sowie die Variation von Achskonzepten. Für diese Szenarien kann die Bewertung der Simulation ohne den Einfluss des Prozessmodells erfolgen. Die Bewertung von Antriebskonzepten mit dem Fokus auf der Traktion oder die Analyse unterschiedlicher Kinematiken der Arbeitsausrüstung sind mit dem abgebildeten Modell nicht möglich.

## 5.2 Radladermodell

### 5.2.1 Starrkörpermodell

Das Starrkörpermodell des Radladers lässt sich in die Baugruppen Arbeitsausrüstung, Vorderwagen, Knicklenkung und Hinterwagen unterteilen (Abbildung 5.2). Dabei ist die vordere Achse starr mit dem Vorderwagen verbunden, die Hinterachse ist als Pendelachse mit mechanischen Anschlägen ausgeführt. Die Kabine ist federnd auf

dem Hinterwagen gelagert. Die Arbeitsausrüstung ist als Z-Kinematik ausgeführt (siehe Abschnitt 2.2.1, Abbildung 2.5f).

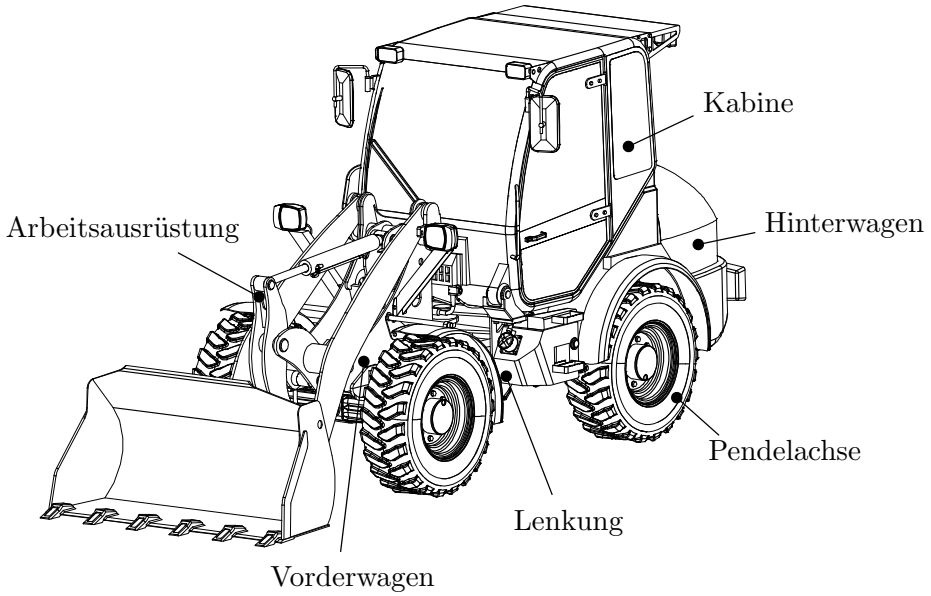


Abbildung 5.2: Starrkörpermodell des Radladers

Das resultierende Starrkörpersystem besteht aus 19 Körpern (Abbildung 5.3). Die Bewegungsgleichungen (Gleichung 5.1) werden in Minimalkoordinaten als System gewöhnlicher Differentialgleichungen formuliert ([HHS97]). Der Vektor der Minimalkoordinaten hat die Dimension 12.

$$\mathbf{H}\ddot{\mathbf{q}} + \mathbf{q}_c + \mathbf{q}_g = \mathbf{q}_f \tag{5.1}$$

Für die Berechnung der Massematrix $\mathbf{H}$ (Gleichung 5.2) sowie der Kraftvektoren $\mathbf{q}_c$, $\mathbf{q}_g$ und $\mathbf{q}_f$ (Gleichung 5.4) werden die kinematischen Beziehungen symbolisch formuliert und in MAPLE in eine entsprechende Berechnungsvorschrift eingesetzt. Die für die Berechnung der translatorischen und rotatorischen JACOBI-Matrizen (Gleichung 5.3) notwendigen Ableitungen werden dabei in MAPLE symbolisch aus den kinematischen Beziehungen berechnet.

$$\mathbf{H} = \sum_{i=1}^{n_k} \left( m_i \mathbf{J}_{Ti}^{\mathrm{T}} \mathbf{J}_{Ti} + \mathbf{J}_{Ri}^{\mathrm{T}} \mathbf{R}_i \mathbf{I}_i \mathbf{R}_i^{\mathrm{T}} \mathbf{J}_{Ri} \right) \tag{5.2}$$

$$
\begin{aligned}
J_{Ti,j} &= \frac{\partial S_i}{\partial q_j} \qquad\qquad i = 1,2,3 \quad j = 1,\ldots,n_f \\
J_{R1,j} &= \frac{\partial R_{31}}{\partial q_j} R_{21} + \frac{\partial R_{32}}{\partial q_j} R_{22} + \frac{\partial R_{33}}{\partial q_j} R_{23} \\
J_{R2,j} &= \frac{\partial R_{11}}{\partial q_j} R_{31} + \frac{\partial R_{12}}{\partial q_j} R_{32} + \frac{\partial R_{13}}{\partial q_j} R_{33} \\
J_{R3,j} &= \frac{\partial R_{21}}{\partial q_j} R_{11} + \frac{\partial R_{22}}{\partial q_j} R_{12} + \frac{\partial R_{23}}{\partial q_j} R_{13}
\end{aligned} \tag{5.3}
$$

$$
\begin{aligned}
\mathbf{q}_c &= \left[q_{c1}, q_{c2}, \ldots, q_{cn_f}\right]^{\mathrm{T}} \text{ mit } \quad q_{ci} = \sum_{j=1}^{n_f} \sum_{k=1}^{n_f} c_{ijk} \dot{q}_j \dot{q}_k \\
&\quad c_{ijk} = \frac{\partial H_{ij}}{\partial q_k} - \frac{1}{2} \frac{\partial H_{jk}}{\partial q_i} \,, \quad k = 1,\ldots,n_f \\
\mathbf{q}_g &= \sum_{i=1}^{n_k} m_i \mathbf{J}_{Ti}^{\mathrm{T}} \mathbf{g}^{\mathrm{T}} \\
\mathbf{q}_f &= \sum_{j=1}^{n_F} \tilde{\mathbf{J}}_{Tj}^{\mathrm{T}} F_j^{\mathrm{T}} + \sum_{k=1}^{n_T} \tilde{\mathbf{J}}_{Rk}^{\mathrm{T}} T_k^{\mathrm{T}}
\end{aligned} \tag{5.4}
$$

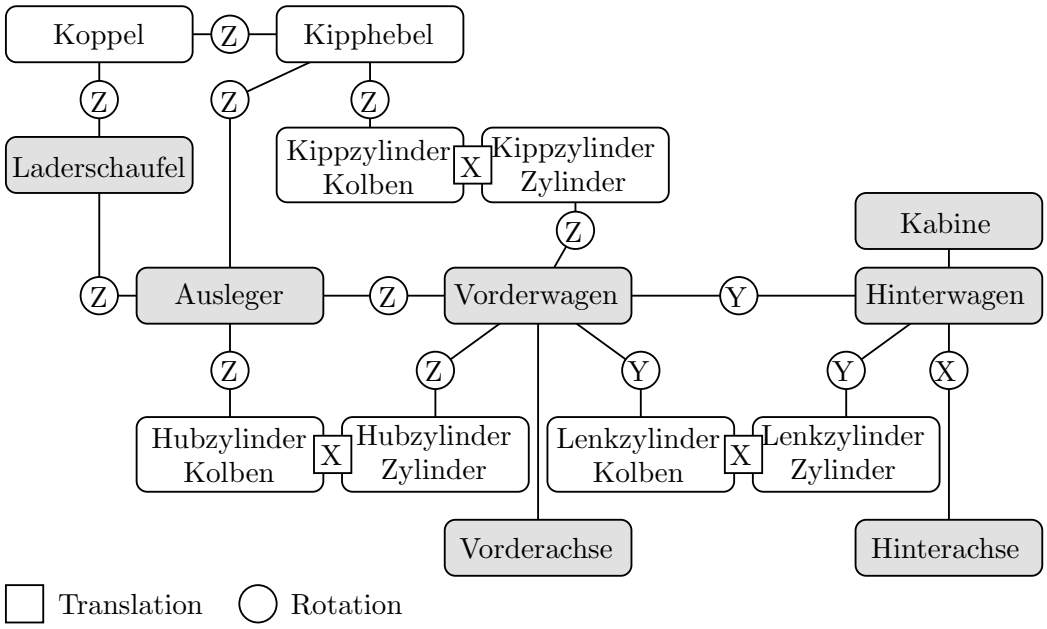


Abbildung 5.3: Topologie des MKS-Modells

Bis auf die Arbeitsausrüstung und die Lenkung bildet das Starrkörpermodell des Radladers eine Baumstruktur mit elementaren Gelenken (Abbildung 5.3). Die Arbeitsausrüstung (Abbildung 5.4) besteht aus einem 9-gliedrigen Mechanismus in Z-Kinematik (Abbildung 2.5f). Die Lenkung wird durch einen 4-gliedrigen Mechanismus (Abbildung 2.5b) realisiert. Zur Auflösung der geschlossenen kinematischen Schleifen werden die Übertragungsfunktionen der gebildeten Mechanismen bestimmt (siehe Abschnitt 2.2.1). Dadurch wird das gesamte Starrkörpermodell in eine Baumstruktur überführt (siehe Abbildung 5.5).

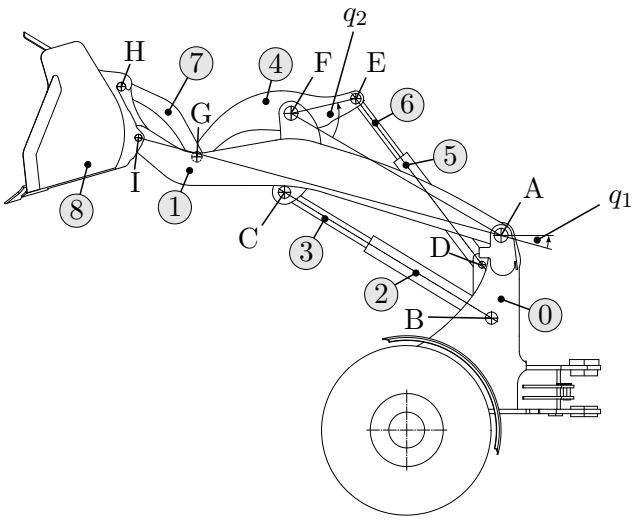


Abbildung 5.4: Kinematik der Arbeitsausrüstung

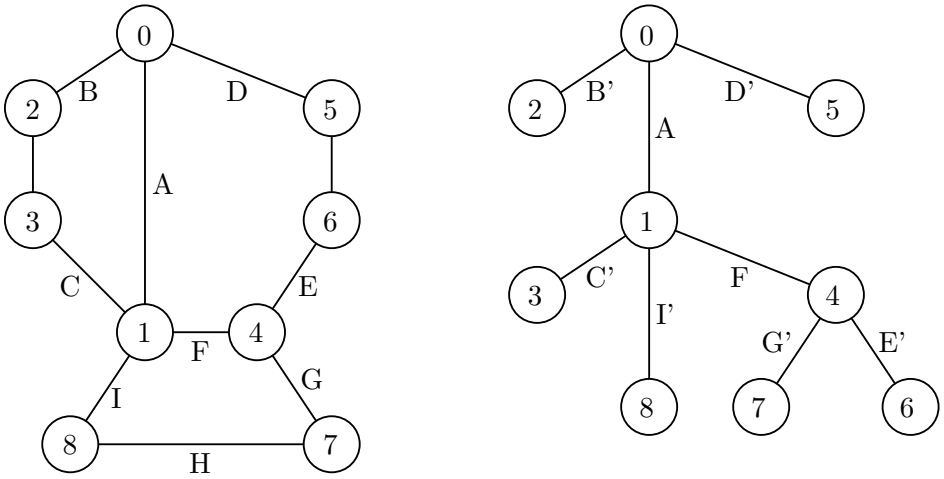


Abbildung 5.5: Überführen der Arbeitsausrüstung in eine Baumstruktur

Am Beispiel der Arbeitsausrüstung werden die kinematischen Schleifen an den Hydraulikzylindern (Körper 2 und 3 sowie Körper 5 und 6) sowie zwischen Koppel und Ladeschaufel (Körper 7 und 8) gelöst (Abbildung 5.5).
Die Positionierung der Bauteile des Radladers in der virtuellen Umgebung erfolgt durch die Auswertung der kinematischen Beziehungen. Aus diesen wird eine Berechnungsvorschrift für die Absolutkinematik (siehe Abschnitt 4.4.3) des Radladers symbolisch abgeleitet. Diese berechnet anhand der generalisierten Koordinaten die Transformationen der Bauteile und übergibt diese an die entsprechenden Schnittstellen (Abbildung 5.6).

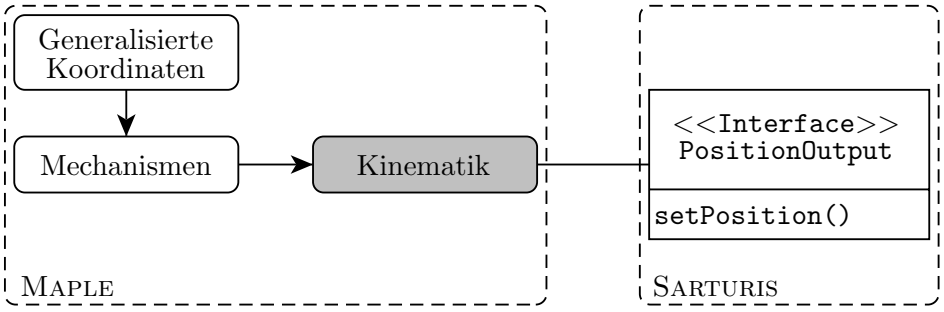


Abbildung 5.6: Kinematikberechnung und Schnittstellen

Die Verbindung des Starrkörpermodells mit der virtuellen Umgebung und den Bedienereingaben erfolgt durch die Berechnung der äußeren Kräfte und Momente ($\mathbf{q}_f$ in Gleichung 5.4). Dabei werden durch das Reifenmodell die Radkräfte in globalen kartesischen Koordinaten berechnet. Das Antriebsmodell berechnet die Aktuatorkräfte im lokalen Koordinatensystem des jeweiligen Aktuators. Die Umrechnung der Kraftwirkung auf die Minimalkoordinaten des Starrkörpermodells erfolgt durch die Berechnungsvorschrift für $\mathbf{q}_f$ (Gleichung 5.4).
Das Antriebsmodell und das Reifenmodell werden ebenfalls in MAPLE formuliert und liegen in Form symbolischer Gleichungen vor. Die Verbindung zum Starrkörpermodell erfolgt in MAPLE. Dadurch verfügt das Starrkörpermodell formal über keine direkte Schnittstelle zum VR-System, da die Abfrage des Terrains durch das Reifenmodell und die Abfrage der Bedienelemente durch das Antriebsmodell gekapselt werden.
Durch die Einschränkung des Arbeitsprozesses auf den Fahrbetrieb können Beladungszustände nicht durch die Interaktion zwischen Arbeitsausrüstung und virtueller Umgebung verändert werden. Dies erfordert ein entsprechendes Prozessmodell (siehe Abschnitt 2.2.4), welches nicht vorliegt und auch nicht Gegenstand dieser Arbeit ist.

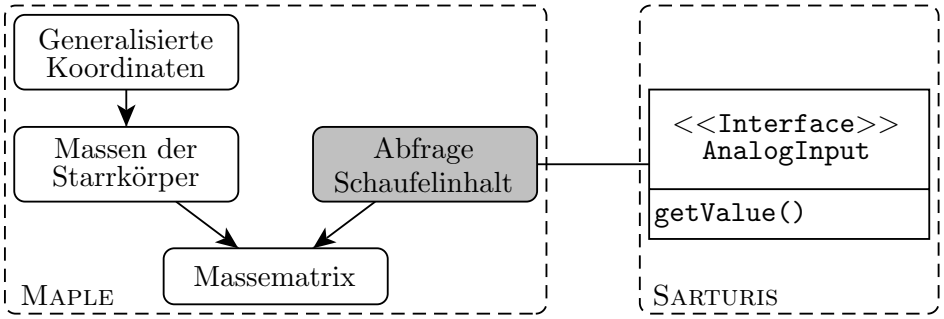


Abbildung 5.7: Abfrage der veränderlichen Gutmasse

Für die Untersuchung des Fahrverhaltens mit unterschiedlichen Beladungszuständen wird daher im Starrkörpermodell eine Punktmasse definiert, welche den Beladungszustand abbildet. Der Betrag der Gutmasse wird an einem geeigneten Referenzpunkt der Ladeschaufel angetragen und kann durch eine entsprechende Schnittstelle verändert werden. Die Berechnung der Massematrix und Kraftvektoren berücksichtigt dabei die veränderliche Masse (Abbildung 5.7).

### 5.2.2 Antriebsmodell

Das Antriebsmodell beschreibt die Erzeugung der mechanischen Leistung im Dieselmotor sowie deren Übertragung auf die einzelnen Verbraucher. Der Dieselmotor wird vereinfacht durch eine Drehzahl-Drehmoment-Kennlinie beschrieben. Die wesentlichen Verbraucher sind die Zylinder der Arbeitsausrüstung, der Fahrantrieb und die hydraulische Lenkung. Die Versorgung und Ansteuerung der Verbraucher erfolgt über entsprechende hydraulische Grundschaltungen (Abbildung 5.8).
Neben der hydraulischen Leistungsübertragung erfolgt die Steuerung und Regelung des Antriebes ebenfalls über ein hydraulisches Netzwerk. Je nach Aufgabenstellung werden diese Elemente durch die Grundgleichungen der Hydraulik oder wie im vorliegenden Radladermodell durch vereinfachte generische Übertragungselemente beschrieben.
Da die Motorleistung in der Regel geringer ist als die Summe der Leistung der einzelnen Verbraucher, ist eine entsprechende Verteilung zu realisieren. In der Realität wird diese Verteilung z. B. durch Sekundärregelungen oder Load-Sensing erreicht. Die Synthese bzw. der Nachweis eines solchen Systems ist nicht Gegenstand der Referenzanwendung. Für das vorliegende Modell kann daher ein ideales System

angenommen werden. Die Gleichungen des Modells bilden die gewünschten Ergebnisse der Leistungsverteilung ab.

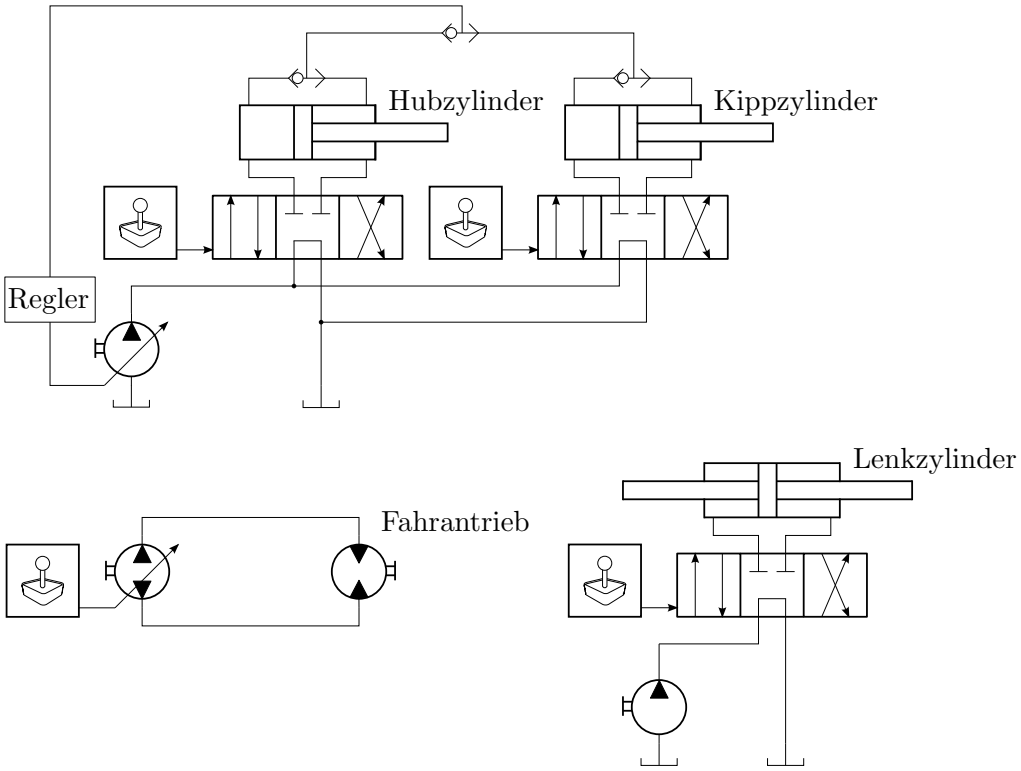


Abbildung 5.8: Hydraulische Grundschaltungen

Die Modellbildung hydraulischer Antriebe führt in der Regel auf steife Differentialgleichungen, welche Berechnungen unter Echtzeitbedingungen wesentlich erschweren ([BP00]). Das vorliegende Modell wurde daher weiter vereinfacht (Abbildung 5.9). Innerhalb des Antriebsmodells werden Sollgrößen für die Position und Geschwindigkeit des Aktuators berechnet. Die Reaktionskraft im MKS-Modell ergibt sich aus dem Abgleich mit den Ist-Größen unter Benutzung eines linearen Feder-Dämpfer-Modells. Diese Modellierung ist unabhängig von der Art des Aktuators (rotatorisch oder translatorisch).

Im Ergebnis wird ein Systemverhalten modelliert, bei dem der Bediener mit dem Eingabegerät die Geschwindigkeit des Aktuators vorgibt und eine lastunabhängige Reaktion zu beobachten ist. Bei Überschreiten der verfügbaren Antriebsleistung erfolgt eine Reduzierung der Vorgabegeschwindigkeiten nach einem vorgegebenen Algorithmus.

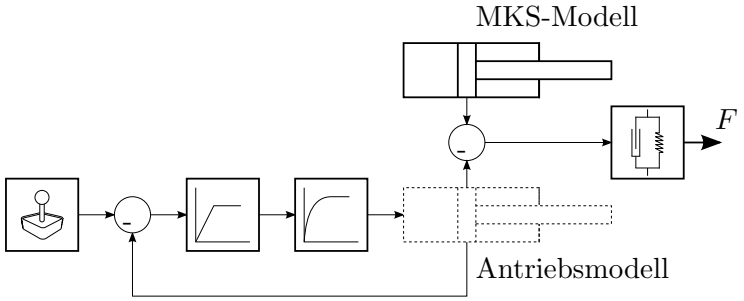


Abbildung 5.9: Vereinfachung des Modells hydraulischer Aktuatoren

Die Zustandsgrößen des Antriebsmodells beschreiben unter anderem die Motordrehzahl sowie die Aktuatorpositionen und -geschwindigkeiten. Dadurch wird die Interaktion des Gesamtmodells mit dem Bediener durch die Schnittstellen des Antriebsmodells realisiert. Diese werden in MAPLE durch generische Funktionen repräsentiert und beim Codeexport in entsprechende Funktionsaufrufe transformiert. Diese werden benutzt, um die Radlader-Komponente mit den SARTURIS-Schnittstellen zu verbinden (Abbildung 5.10).

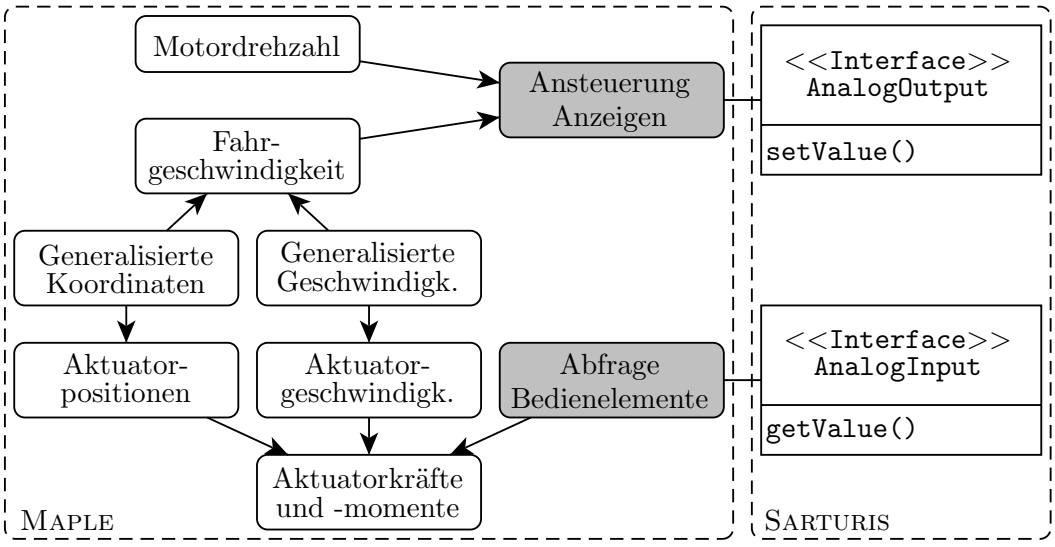


Abbildung 5.10: Ansteuerung der Anzeigen und Abfrage der Bedienelemente

### 5.2.3 Reifenmodell

Für die Referenzanwendung wurde ein vereinfachtes Reifenmodell gewählt. Dabei werden die Reifen durch lineare Feder-Dämpfer-Elemente abgebildet. Die Geometrie des Reifens sowie die Ausbildung des Latsches werden vernachlässigt. Der Bodenkontakt findet immer am idealisierten Aufstandspunkt statt. Der geometrische Fehler bei der Überfahrt einer Schwelle (Abbildung 5.11) wird für die Referenzanwendung vernachlässigt.

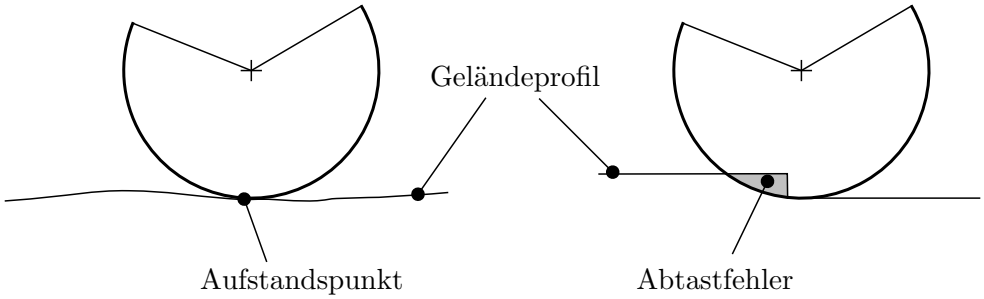


Abbildung 5.11: Aufstandspunkt für Reifenmodell

Die Parameter der Reifen (Steifigkeit, Dämpfung) werden durch grobe Vergleichsmessungen mit einer vergleichbaren realen Maschine bestimmt. Dabei wird im Wesentlichen das Modell durch Abgleich mit den Messdaten kalibriert. Analytische Methoden zur Parameterbestimmung werden nicht angewendet, da diese die Qualität des groben Modells nicht weiter positiv beeinflussen.
Im Ergebnis ist das Reifenmodell in der Lage, die Fahrzeugbewegung im Gelände so abzubilden, dass der Bediener im Fahrsimulator einen realistischen Eindruck der Fahrsituation bekommt. Für höhere Anforderungen sind entsprechend weiterentwickelte Reifenmodelle zu verwenden (siehe Abschnitt 2.2.3).
Unabhängig von der Wahl des Reifenmodells ist bei der Notation der entsprechenden Gleichungen die Abfrage des Terrains zu beachten. Diese erfordert eine Kommunikation mit dem Softwaresystem zur Laufzeit des Modells, da die Terraindaten nicht fest in das Reifenmodell integriert werden. Die Formulierung der Gleichungen des Reifenmodells in Maple benutzt entsprechende Platzhalter. Diese werden bei der Codegenerierung in Funktionsaufrufe transformiert. Die Implementation dieser Funktionen erfolgt in der Sarturis-Komponente des Radladermodells (Abbildung 5.12).

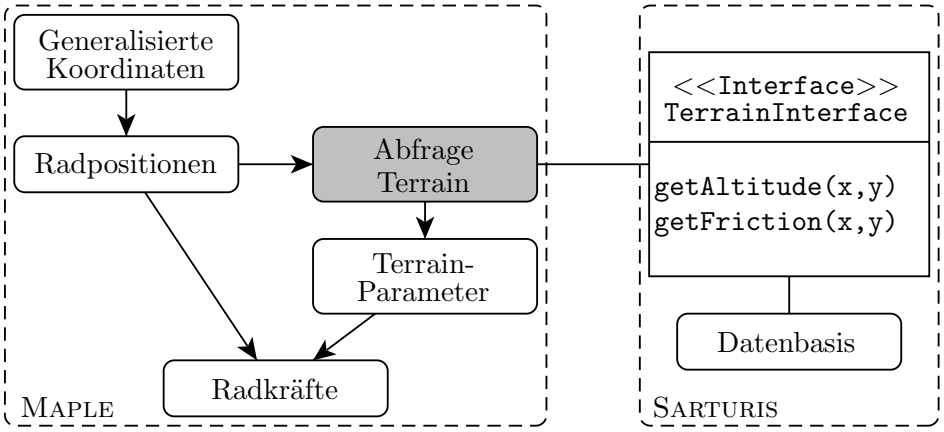


Abbildung 5.12: Ermittlung der Radkräfte

## 5.3 Erstellen der Komponente

Wesentlich für die Referenzanwendung ist die Komponente zur Berechnung der Differentialgleichungen des Radladermodells (siehe Abschnitt 3.2.3). Der Code zur Berechnung des resultierenden Systems gewöhnlicher Differentialgleichungen wird durch MAPLE generiert (Abbildung 5.13).

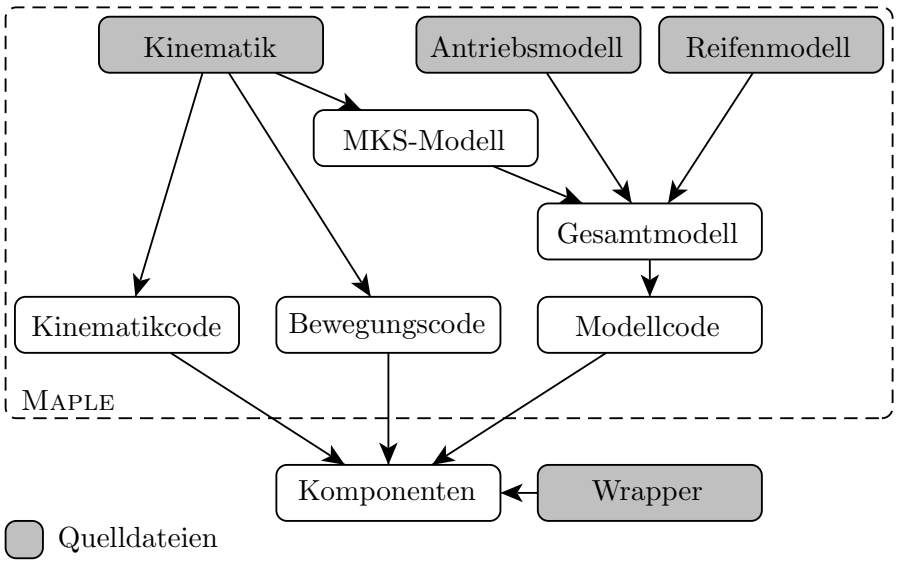


Abbildung 5.13: SARTURIS-Komponenten des Radladers erzeugen

Als Eingabedaten für MAPLE dienen die kinematischen Zusammenhänge, das Antriebsmodell und das Reifenmodell. Das Starrkörpermodell wird durch symbolisches Auswerten (siehe Abschnitt 5.2.1) einer entsprechenden Vorschrift ([HHS97]) durch MAPLE erzeugt und mit dem Antriebs- und dem Reifenmodell verknüpft. Zusätzlich werden die kinematischen Zusammenhänge benutzt, um den Code zur Berechnung der Bauteilpositionen (siehe Abschnitt 4.4.3) und der Berechnung der Daten zur Bewegungssystemsteuerung (siehe Abschnitt 4.4.5) zu erzeugen. Damit wird das *One-Single-Source*-Prinzip befolgt und es entstehen keine Redundanzen bezüglich der Abmessungen und kinematischen Zusammenhänge des Radladers.
Der durch MAPLE generierte Code entspricht nicht dem Komponentenmodell von SARTURIS. Dieses muss durch einen manuell zu implementierenden *Wrapper*-Code abgebildet werden. Neben der Implementation und Nutzung der entsprechenden Schnittstellen muss dieser Code die Speicherverwaltung sowie die Initialisierung des Modells und der Variablen implementieren.
Durch diese Indirektion ist es möglich, den generierten Code für verschiedene Anwendungsfälle in unterschiedlichen Komponenten zu verwenden (Abbildung 5.14). Dafür sind die entsprechenden zusätzlichen *Wrapper* zu implementieren. So kann der Code zum Berechnen der Differentialgleichungen direkt in einer `OdeSystem`-Komponente verwendet werden. Damit wird der Ansatz der Trennung von numerischer Methode und Modell (siehe Abschnitt 2.1) auf der Komponentenebene realisiert. Zur Steigerung der Effizienz der Berechnung im Hinblick auf die Echtzeitfähigkeit des Codes kann diese Verknüpfung auch innerhalb einer Komponente auf der Ebene der Klassen und Objekte realisiert werden (Abbildung 5.14). Die resultierende Komponente implementiert das Interface `Thread` und kann als *Coarse-grained* klassifiziert werden.

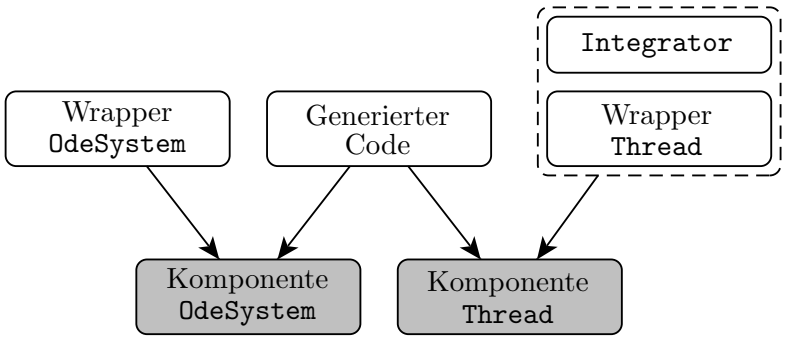


Abbildung 5.14: Mehrfachverwendung des generierten Codes

Die notwendigen *Wrapper*-Codes lassen sich durch die Anwendung von Templates verallgemeinern. Dadurch reduziert sich der Aufwand für die Implementierung der verschiedenen Modelle und Anwendungsfälle. Die Schnittstelle des durch MAPLE generierten Codes ist sehr einfach und daher gut geeignet für eine Generalisierung mit Templates. Daten werden entweder als Skalare in Form einer doppelt genauen Gleitkommazahl oder als Felder in Form eines Pointers ausgetauscht. Komplexe Datenstrukturen oder Objekte sind nicht als Parameter oder Rückgabewerte der generierten Funktionen vorgesehen.
Aufgrund der einfachen Schnittstelle (Gleichung 5.5) müssen die Daten innerhalb der Komponente so organisiert werden, dass der Zugriff auf die Werte entweder über die skalaren Größen oder die Startadresse von Vektoren möglich ist. Dabei treten Redundanzen bezüglich der Dimension der Vektoren und der Reihenfolge der Zustandsgrößen in den Vektoren auf. Der resultierende Prozess zur Erzeugung der Komponenten ist aus diesem Grund sehr fehleranfällig.

$$\begin{aligned} \text{<arg>} &::= \text{double} \mid \text{double*} \\ \text{<arglist>} &::= \text{<arg>} \mid \text{<arglist> <arg>} \\ \text{<function>} &::= \text{double fname(<arglist>)} \end{aligned} \tag{5.5}$$

## 5.4 Anwendung

Für die Referenzanwendung werden die Komponenten für die Berechnung der Differentialgleichungen, der Kinematik und der Bewegungsdaten in einem Modul zusammengefasst. Der Codeexport aus MAPLE lässt sich in den Prozess der Erstellung des Moduls (Abbildung 3.5 und Abbildung 4.10) integrieren. Änderungen an den MAPLE-Dokumenten (Abbildung 5.13) führen zur Neuerstellung aller relevanten Dateien und des Moduls.
Zur Anwendung in einer interaktiven Simulation werden mindestens die Komponenten zur Modellberechnung und die Kinematik benötigt. Diese beiden Komponenten bilden alle notwendigen Ein- und Ausgabeschnittstellen ab. Die Komponente zur Berechnung der Bewegungsdaten ist optional und wird ausschließlich am interaktiven Fahrsimulator verwendet. Die generierten Komponenten stehen über spezialisierte Schnittstellen miteinander in Beziehung. Diese Schnittstellen werden innerhalb des generierten Moduls spezifiziert und strukturieren den Datenaustausch zwischen den Komponenten (Abbildung 5.15).

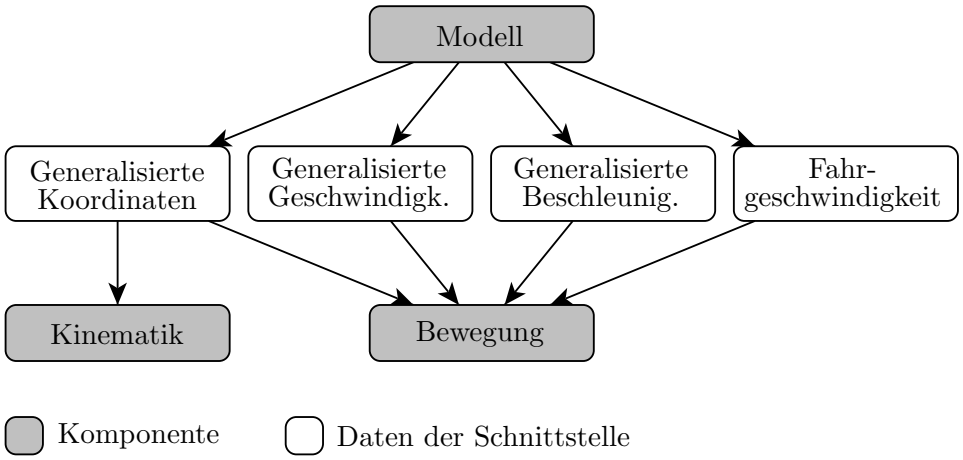


Abbildung 5.15: Datenaustausch zwischen den Radladerkomponenten

Der aus den generierten Komponenten gebildete *virtuelle Radlader* verfügt über entsprechende Ein- und Ausgabeschnittstellen (Abbildung 5.16). Diese werden für eine Desktopsimulation durch Joysticks und Pedale aus dem Bereich der Consumerelektronik bereitgestellt ([PKG07]). Die Anzeige der Werte für Geschwindigkeit und Drehzahl erfolgt in der Visualisierung. Für die Anwendung im interaktiven Simulator wurde das CAN-Protokoll der Originalbedienteile implementiert, um die Originalkabine zu verwenden (siehe Abschnitt 4.4.1).

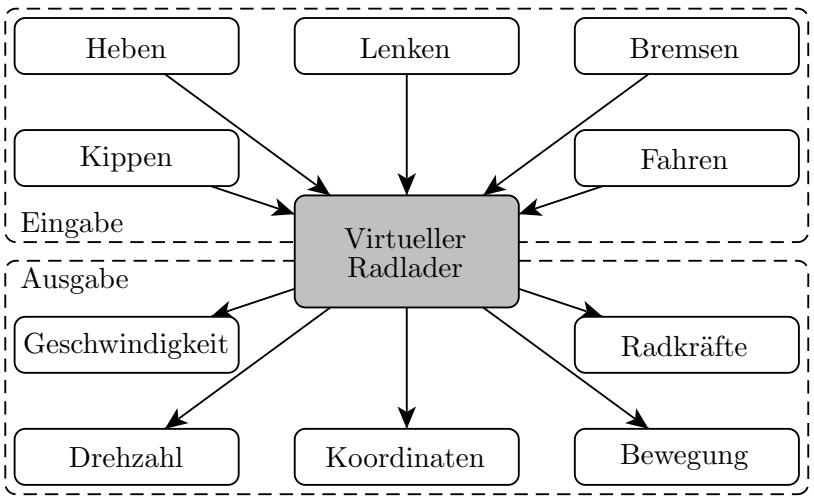


Abbildung 5.16: Schnittstellen des Radladermodells

# 6 Konfiguration mit Skriptsprachen

## 6.1 Motivation

Die gegebene Referenzimplementation des SARTURIS-Frameworks sowie die Anwendung bei der interaktiven Fahrsimulation eines Radladers zeigen die Leistungsfähigkeit des Ansatzes der komponentenbasierten Entwicklung. Durch die Verteilung der Funktionalität auf die Komponenten und die Strukturierung des Softwaresystems durch klar definierte Schnittstellen werden die einzelnen Komponenten überschaubar und sind somit leicht zu implementieren und zu warten.
Durch den Ansatz der Konfiguration der Anwendungsfälle mit XML werden die einzelnen Komponenten in unterschiedlichen Anwendungen und Umgebungen wiederverwendet. Dies erhöht automatisch die Anzahl der Anwendungsfälle und führt in Kombination mit der gegebenen Übersichtlichkeit zu einem robusten Framework.
Die Veränderung der physikalischen Modelle erfolgt auf der Modellbildungsebene in einer geeigneten Notation. Aufgrund der automatisierten Komponentenerstellung durch Codegeneratoren sind für die Integration der Modelle in das Framework keine Kenntnisse in C++ erforderlich. Dadurch erhält der Anwender ein leistungsfähiges Werkzeug für die Abbildung der Maschinenmodelle.
Das Softwaresystem ist nicht an eine spezielle VR-Umgebung gebunden. Die Entwicklung der Modelle und Komponenten sowie die Anpassung der Konfigurationen kann an reduzierten Systemen (z. B. Desktop-PC) erfolgen. Die Übertragbarkeit auf komplexe VR-Systeme (z. B. Fahrsimulator) ist prinzipbedingt gegeben. Durch die Wiederverwendung der bestehenden Komponenten sowie die automatisierte Komponentenerstellung im Bereich der Simulationsmodelle wird der hauptsächliche Aufwand bei der Realisierung einer Simulationsanwendung durch die Konfiguration bestimmt. Dies gilt insbesondere bei Aufgabenstellungen aus dem Bereich der Maschinen- und Modellentwicklung (siehe Abschnitt 3.1.1).
Die Simulationsanwendung wird durch die Komponenten und deren Verknüpfung definiert. Das flexible Framework ermöglicht unterschiedliche Konfigurationen unter Verwendung der identischen Menge an Komponenten. Die dabei entstehenden Konfi-

gurationsdateien beschreiben sowohl anwendungsspezifische als auch allgemeingültige Fragmente.
Die identische Abbildung der Komponentenstruktur in der Konfigurationsebene führt zu umfangreichen und unübersichtlichen Konfigurationsdokumenten. Stattdessen kann das Konzept der Schnittstellen auf die Konfiguration übertragen werden. Dadurch können analog zu den Komponenten einzelne Strukturen der Konfiguration ausgetauscht werden. Dabei wird die resultierende Komponentenstruktur gekapselt (Abbildung 6.1). Die Zuordnung zwischen den Konfigurationsobjekten und den einzelnen Komponenten ist beliebig. Schnittstellen werden durch die Komponentenspezifikation definiert und analog auf die Konfigurationsobjekte übertragen. Dadurch wird die Komponentenstruktur der Simulationsanwendung durch die Konfigurationsobjekte nachgebildet. Die Prüfung auf Korrektheit einer Konfiguration kann ohne die Bildung von Instanzen der Komponenten in der Konfigurationsebene erfolgen.

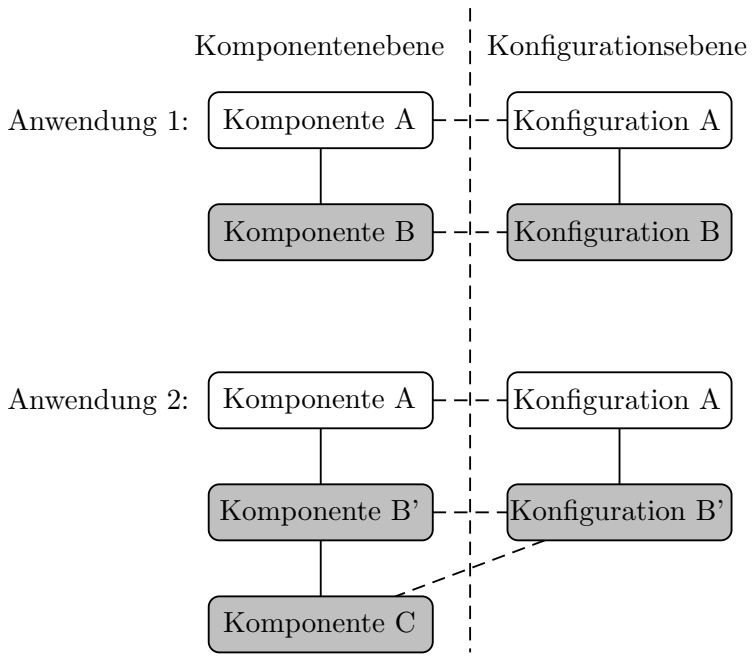


Abbildung 6.1: Beziehungen zwischen Komponenten und Konfigurationen

Eine Analyse vorhandener Konfigurationen des Frameworks im Kontext konkreter Aufgabenstellungen bei der Maschinenentwicklung ([Bür+11; Hos+12; Pen+12; SNK11]) zeigt, dass sich eine Simulationsanwendung in vier wesentliche Teilbereiche gliedern lässt:

- **Maschine:** Die Konfiguration der Maschine beschreibt sowohl die für die Simulation verwendeten Rechenmodelle als auch die grafische und akustische Repräsentation in der VR-Umgebung. So gilt z. B. die Verbindung einer 3D-Geometrie mit einem Koordinatensystem des Maschinenmodells ausschließlich im Kontext der abgebildeten Maschine.

- **Bedienelemente:** Je nach Anwendung erfolgt die Interaktion mit dem Modell mit unterschiedlichen Bedienumgebungen. Dabei werden beliebige Kombinationen realer Geräte (siehe Abschnitt 4.4.1) und von Ersatzelementen der GUI (siehe Abschnitt 4.4.2) eingesetzt. Die Schnittstellen zum Maschinenmodell ändern sich dabei nicht (siehe Abbildung 5.16). Für Aufgaben der Modell- und Methodenentwicklung existieren generische Bedienumgebungen, die mit unterschiedlichen Modellen verbunden werden.

- **VR-System:** Die Hardware für die Ausführung der Simulation variiert von normalen Desktop-PCs bis zu komplexen interaktiven Umgebungen. Die Konfiguration des VR-Systems beschreibt die entsprechende Infrastruktur (Projektion, Audio, Netzwerk, Bewegung).

- **Datenbasis/Szenario:** Die Datenbasis konfiguriert die geometrischen und physikalischen Daten der virtuellen Umgebung. Anwendungsspezifische Zusatzelemente und Sonderereignisse werden durch die grafische Benutzeroberfläche gesteuert.

Diese Teilbereiche bilden standardisierte Konfigurationsmuster mit entsprechenden Schnittstellen ab (Abbildung 6.2). Die Trennung der Bereiche ermöglicht die Wiederverwendung bestehender Fragmente und die einfache Anpassung von Simulationsanwendungen. Aus den vorliegenden Konfigurationsmustern ist für eine konkrete Simulationsanwendung bestehend aus einer Maschine, einer Bedienumgebung, einem VR-System und einer Datenbasis für jeden auszuführenden Prozess eine Konfiguration zu erzeugen.
Die Konfiguration mit XML kann diese Anforderungen nicht erfüllen, da diese direkt die Komponentenstruktur eines konkreten Prozesses beschreibt. Übergeordnete Hierarchien (siehe Abbildung 6.2) werden nicht berücksichtigt. Die Aufteilung der Konfiguration auf mehrere Dokumente ermöglicht prinzipiell die Wiederverwendung von Teilen der Konfiguration. Die fehlende Abbildung von verbindlichen Schnittstellen und parametrisierten Strukturen schränken diese aber deutlich ein.

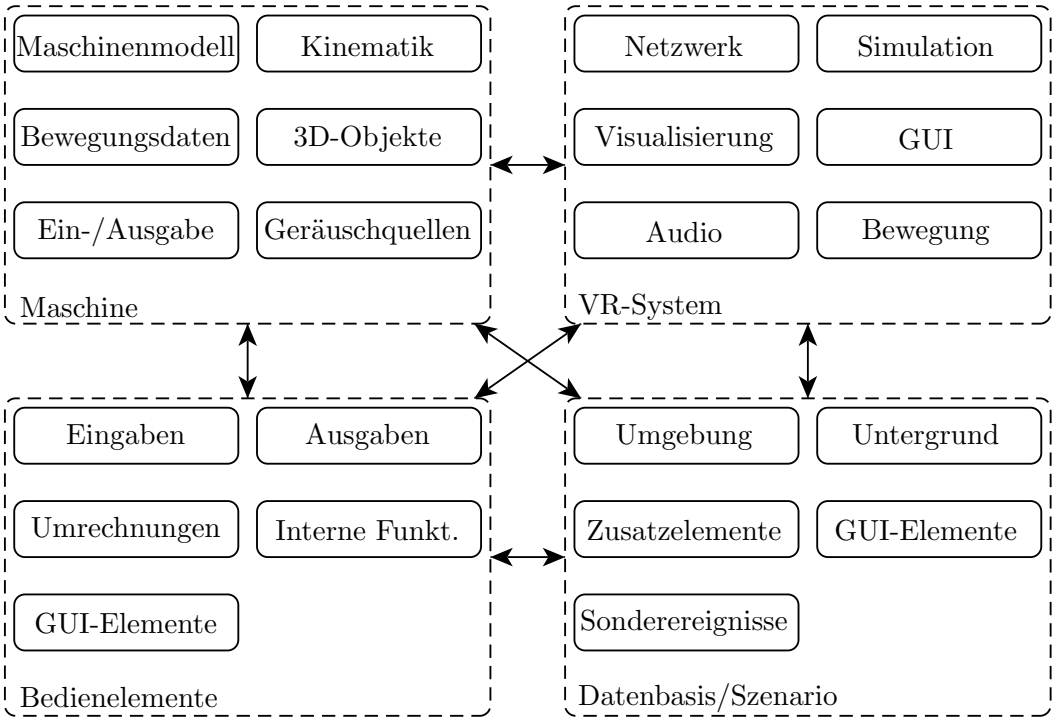


Abbildung 6.2: Konfigurationsaspekte einer Simulationsanwendung

Aus Gründen der Übersichtlichkeit, Wartbarkeit und Korrektheit der Konfigurationen entstehen erweiterte Anforderungen an die Konfigurationssprache:

- **Modularisierung:** Häufig wiederkehrende Konfigurationsmuster sind in gekapselten Objekten zusammenzufassen. Die Anzahl an Komponenten, welche durch ein solches Konfigurationsobjekt beschrieben werden, ist beliebig. Die internen Verknüpfungen werden automatisch durch das Konfigurationsobjekt erzeugt. Externe Verknüpfungen werden durch entsprechende Schnittstellen und die Verbindung zu anderen Konfigurationsobjekten hergestellt.

- **Parametrisierung:** Die definierten Konfigurationsobjekte können durch Parameter verändert werden. Dabei können sowohl die Struktur des resultierenden Komponentensystems als auch die Parameter der Komponenten beeinflusst werden.

- **Kontrollstrukturen:** Innerhalb der Konfigurationsdokumente können Verzweigungen und Schleifen genutzt werden, um die Konfiguration an externe Vorgaben oder Systemvariablen anzupassen.

- **Typsicherheit:** Die Prüfung der Typsicherheit erfolgt in der Konfigurationsebene.

Diese Anforderungen werden durch Auszeichnungssprachen wie XML nicht erfüllt. Die Forderung nach parametrisierbaren Konfigurationsobjekten, Kontrollstrukturen und Typsicherheit bedingt die Anwendung einer objektorientierten Programmiersprache als Konfigurationssprache. Aus Gründen der Einfachheit, Flexibilität und Übersichtlichkeit wird die Anwendung von C++ als Konfigurationssprache nicht betrachtet. Stattdessen wird PYTHON als objektorientierte Skriptsprache eingesetzt.

## 6.2 Konfiguration mit PYTHON

### 6.2.1 Allgemeines

PYTHON ist eine höhere Programmiersprache. Standardmäßig wird diese in einem Interpreter ausgeführt. Dieser übersetzt die PYTHON-Dokumente vor der Ausführung in einen binären Code. Dabei wird die Syntax der Dokumente überprüft. PYTHON-Programme sind plattformneutral. Die Ausführung auf einer speziellen Zielplattform setzt die Verfügbarkeit des Interpreters und der notwendigen Bibliotheken voraus. Für die Zielplattformen des SARTURIS-Frameworks ist diese Voraussetzung erfüllt ([PYT]).
Der PYTHON-Interpreter ist in ein ausführbares Programm und eine dynamische Bibliothek unterteilt. Die dynamische Bibliothek verfügt über ein entsprechendes API. Unter Nutzung dieses API kann ein PYTHON-Code sehr einfach im Adressraum einer C++-Anwendung ausgeführt werden. Dadurch ist PYTHON sehr gut als Skriptsprache geeignet ([Lan09]).
PYTHON unterstützt verschiedene Programmierparadigmen wie Objektorientierung, Aspektorientierung und funktionale Programmierung. Somit können unterschiedliche Problemstellungen sehr flexibel mit dem jeweils am besten geeigneten Konzept gelöst werden. PYTHON ist eine dynamische Programmiersprache, d. h. Klassen- und Funktionsdefinitionen können zur Laufzeit manipuliert werden. Dies ist vor allem vor dem Hintergrund der Anwendung in einem flexiblen Komponentensystem von Vorteil, da die Klassenobjekte berechnet werden können. Im Gegensatz zu C++ können dynamisch entsprechende Instanzen erzeugt werden ([Pil10]). Die Typsicherheit ist dabei explizit durch PYTHON gegeben.
PYTHON verfügt über eine umfangreiche Standardbibliothek, die es z. B. ermöglicht, Informationen über die Systemkonfiguration der im VR-System eingesetzten Rechner

oder die angeschlossenen Bedienerschnittstellen während der Konfiguration zu ermitteln. Dadurch ergeben sich erweiterte Möglichkeiten bei der Abbildung parametrischer Konfigurationsobjekte.

### 6.2.2 Abstraktion der Konfiguration

Die Erweiterung der Konfigurationsmöglichkeiten von SARTURIS soll das vorhandene Konzept der Konfiguration mit XML für den Anwender nicht verändern. Aus diesem Grund wurde die Konfigurationsschicht abstrahiert. Dabei werden analog zu den Komponentenmodulen dynamische Bibliotheken benutzt.
Die Konfigurationsdokumente werden beim Aufruf der SARTURIS-Anwendung als Parameter übergeben. Innerhalb der Konfigurationsschicht existiert eine Instanz der Klasse `ConfigReader`. Diese ist als *Singleton* implementiert und erzeugt den Konfigurationsbaum aus den gegebenen Dokumenten (Abbildung 6.3).

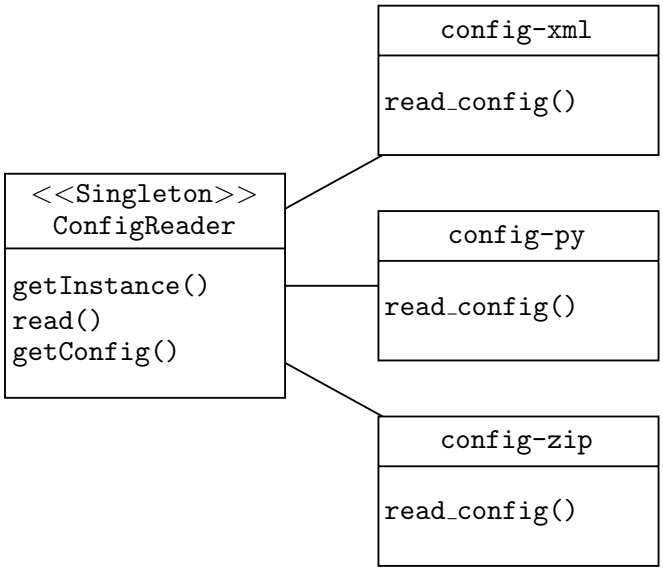


Abbildung 6.3: `ConfigReader` und Abstraktion der Konfiguration

Dabei wird anhand der Dateiendung das zugehörige Konfigurationsmodul geladen. Dieses implementiert die Funktionen zum Lesen und Analysieren des Dokumentes sowie zum Erzeugen der entsprechenden Datenstrukturen für den Konfigurationsbaum. Neben den Modulen zum Laden von XML- und PYTHON-Dokumenten ermöglicht das ZIP-Modul die Zusammenfassung der Konfiguration und zusätzlichen Daten (3D-Modelle, Sounds, etc.) in einer Datei. Es wird vorausgesetzt, dass sich in der Wurzel

des ZIP-Files eine Datei mit der Bezeichnung `top.xml` oder `top.py` befindet. Diese wird durch rekursives Anwenden der `ConfigReader`-Instanz geladen und beschreibt die Konfiguration.

### 6.2.3 Konfigurationsobjekte

Analog zur Konfiguration in XML wird die Simulationsanwendung durch eine Struktur von Konfigurationsobjekten beschrieben. Diese Konfigurationsobjekte repräsentieren die Komponenten mit den zugehörigen Parametern und Verknüpfungen. Bei der Konfiguration in XML werden die Konfigurationsobjekte durch das `Instance`-Tag beschrieben (siehe Abbildung 4.7). Da XML als Auszeichnungssprache über keinen Zuweisungsoperator verfügt, erfolgt die eigentliche Instanzbildung in der Laufzeitumgebung von SARTURIS. Objekte im Sinne von Instanzen existieren in XML nicht. Die Benennung der Objekte erfolgt durch das Attribut `Name`. Der Namensdienst zur Identifikation der Objekte sowie die Prüfung der Typsicherheit erfolgt durch die Laufzeitumgebung von SARTURIS.
Im Gegensatz dazu können in PYTHON Konfigurationsklassen definiert werden. Diese reflektieren die SARTURIS-Komponenten mit den entsprechenden Parametern und Schnittstellen. Das Konzept der Namensräume kann durch Anwendung von PYTHON-Modulen abgebildet werden. Durch die Zuweisung von Instanzen der Konfigurationsklassen auf PYTHON-Variablen wird ein Namensdienst in der Konfigurationsebene realisiert. Die Verknüpfung von Komponenten erfolgt durch Kombination der entsprechenden Konfigurationsobjekte (Abbildung 6.4).

```
1  # Ein- und Ausgaben
2  pedal = Pedal()
3  meter = Meter(Min = 0.0, Max = 35.0)
4
5  # Radlader
6  wheelloader = WheelLoader(Model = 'WL612X')
7
8  # Verknuepfungen
9  wheelloader.drivepedal(pedal)
10 wheelloader.speed(meter, Unit = 'KMH')
```

Abbildung 6.4: PYTHON-Konfiguration einer Instanz der Radladerkomponente

Die Verbindung der Objekte erfolgt durch spezialisierte Methoden, welche passend zum Namen der Schnittstelle durch die Konfigurationsobjekte implementiert werden. Innerhalb dieser Methoden findet die Überprüfung der Typsicherheit statt. Dabei wird die Vererbungshierarchie des übergebenen Objektes analysiert und mit der Spezifikation der Schnittstelle verglichen. Dieser Vergleich basiert auf den Klassenobjekten der zugehörigen Konfigurationsklassen. Die Namen von Parametern können sowohl in den Konstruktoren der Konfigurationsobjekte als auch in den Verknüpfungsmethoden angegeben werden. Dadurch wird die Lesbarkeit der Konfiguration verbessert.
Die Problematik der Redundanz des Schemas der XML-Konfiguration und der Komponentenspezifikationen (Abschnitt 4.2.3) gilt analog für die Klassen der elementaren Konfigurationsobjekte. Die Lösung erfolgt identisch durch das Generieren der Klassen aus den Komponentenspezifikationen. Dabei wird das Konzept der Checksumme zur Überprüfung der SARTURIS-Komponenten (Abbildung 4.10) analog in PYTHON umgesetzt.
Der Vergleich der Konfiguration in PYTHON und XML zeigt, dass die Anwendung von PYTHON keine Nachteile gegenüber XML hat. Die Darstellung der Informationen (Parameter, Verknüpfung, etc.) ist nahezu identisch. Die syntaktisch notwendigen zusätzlichen Informationen beider Sprachen sind vergleichbar. Die wesentlichen Vorteile der Konfiguration in PYTHON im Vergleich zu XML liegen in der Objektorientierung, Parametrisierung, Typsicherheit und der Möglichkeit, die Konfiguration durch Kontrollstrukturen zu beeinflussen. Zur flexiblen Parametrisierung und Strukturierung der Konfiguration sowie zur Interaktion mit dem Anwender können alle durch die PYTHON-Standardbibliothek und die zahlreichen Erweiterungsmodule bereitgestellten Funktionen benutzt werden.
Die generierten elementaren Konfigurationsobjekte bilden die Komponentenstruktur von SARTURIS ab. Die Konfiguration einer Simulationsanwendung mit diesen elementaren Objekten führt zu umfangreichen Dokumenten analog zur Konfiguration in XML. Der einfache Ersatz der Notation der Konfiguration (PYTHON statt XML) bietet keine Verbesserung hinsichtlich der Wiederverwendung bestehender Konfigurationsmuster.
Das wesentliche Potenzial für eine bessere Strukturierung der Konfiguration und Wiederverwendung von Fragmenten (siehe Abbildung 6.2) liegt in der Verknüpfung der elementaren Konfigurationsobjekte zu übergeordneten Objekten mit eigenen Parametern und Schnittstellen. Die resultierende Struktur wird automatisch in eine Struktur aus elementaren Konfigurationsobjekten transformiert. Die so entstehende Konfi-

gurationssystematik ist transparent für das Framework und kann problembezogen implementiert und erweitert werden.

## 6.3 Erweiterte Konzepte und Muster der Konfiguration

### 6.3.1 Konnektorenkonzept

Die Verknüpfung von Konfigurationsobjekten erfolgt durch die Anwendung der Verknüpfungsmethoden. Diese werden durch die automatische Generierung der Konfigurationsklassen erzeugt. Dabei werden die Methodennamen aus den Bezeichnern der Schnittstellen berechnet. Veränderungen an diesen Bezeichnern erfordern die Anpassung der Konfiguration. Dies kann vor allem dann vermieden werden, wenn die Schnittstelle anhand der Komponententypen durch einen Automaten ermittelt werden kann. Dieser Fall ist gegeben, wenn eine Komponente für jeden Typen einer zugeordneten Komponente nur eine Schnittstelle anbietet (Abbildung 6.5).

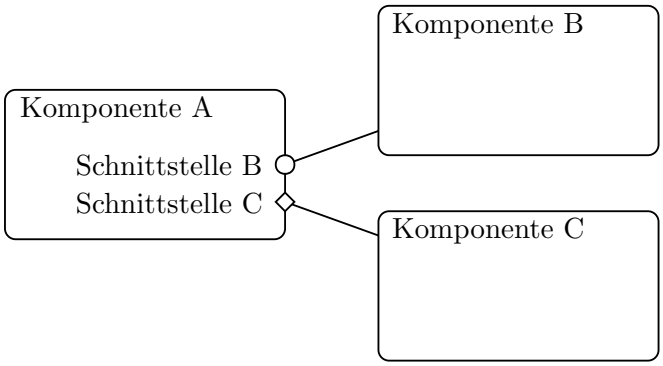


Abbildung 6.5: Komponenten mit eindeutigen Schnittstellentypen

In diesem Fall führt die Anwendung einer generischen `connect`-Methode zu einer Konfiguration, welche bei Veränderungen der Schnittstellenbezeichner nicht angepasst werden muss (Abbildung 6.6).
Für die Anwendung der `connect`-Methode bei mehreren Schnittstellen gleichen Typs wird der Parameter `slot` eingeführt. Dieser muss bei gegebener Mehrdeutigkeit auf den entsprechenden Schnittstellenbezeichner gesetzt werden. Die spezialisierten Verknüpfungsmethoden bleiben erhalten und werden durch die Anwendung des $\lambda$-Operators auf die generische `connect`-Methode abgebildet (Abbildung 6.7).

Die Einführung einer generischen `connect`-Methode kann durch den Generator für die Konfigurationsobjekte erfolgen. Dieser generiert die Methode und die entsprechenden Verzweigungen auf der Basis des Klassenobjektes der zugeordneten Instanz. Dieser Ansatz schränkt die Anwendung der `connect`-Methode auf die durch das Framework vorgegebenen elementaren Konfigurationsklassen ein. Manuell implementierte höherwertige Konfigurationsobjekte können durch die generierten `connect`-Methoden nicht berücksichtigt werden, da deren Klassenobjekte dem Generator nicht bekannt sind.

```
1  # Instanzen
2  a = ComponentA()
3  b = ComponentB()
4  c = ComponentC()
5
6  # Variante 1: Anwendung der Verknuepfungsmethoden
7  a.slotb(b)
8  a.slotc(c)
9
10 # Variante 2: Anwendung der generischen connect-Methode
11 a.connect(b)
12 a.connect(c)
```

Abbildung 6.6: Anwendung der generischen `connect`-Methode

```
1  class ComponentA:
2    def __init__(self):
3      self.slota = lambda r0 : self.connect(r0, slot = 'SlotA')
4      self.slotb = lambda r0 : self.connect(r0, slot = 'SlotB')
```

Abbildung 6.7: Verknüpfungsmethoden mit Anwendung des $\lambda$-Operators

Ein geeigneter Ansatz ist die Einführung von Konnektoren. Dies sind Objekte, welche die Verknüpfung zwischen zwei Konfigurationsobjekten anhand der Klassenobjekte herstellen (Abbildung 6.8). Dabei greift die generische `connect`-Methode auf die Menge der verfügbaren Konnektoren zu und berechnet den geeigneten Konnektor anhand der Konfigurationsobjekte.
Durch die im Framework definierten Komponentenbeziehungen wird eine grundlegende Menge an Konnektoren definiert, welche automatisch anhand der Komponenten-

spezifikation erzeugt werden kann. Mit diesen Konnektoren können die elementaren Konfigurationsobjekte verbunden werden. Die Menge der Konnektoren ist ausreichend, um die gesamte durch die SARTURIS-Komponenten abgebildete Funktionalität zu konfigurieren. Der Vorteil des Konnektorenkonzeptes liegt in der Erweiterbarkeit der Menge an Konnektoren. So können Konnektoren für die höherwertigen Konfigurationsobjekte hinzugefügt werden. Dadurch können beliebige Verbindungen zwischen elementaren und höherwertigen Objekten hergestellt werden. Dabei wird die Vererbungshierarchie der Konfigurationsobjekte analysiert und die Verbindung durch den Konnektor hergestellt, welcher die höchste Spezialisierung benutzt.

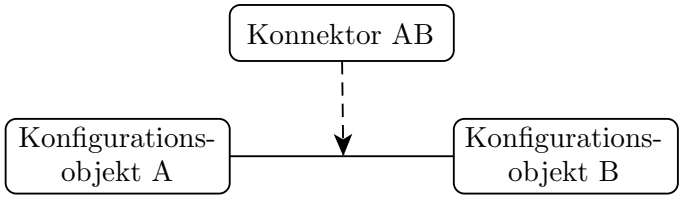


Abbildung 6.8: Konnektorenkonzept

Häufig wiederkehrende Beziehungen zwischen Komponenten können ebenfalls durch das Konnektorenkonzept abgebildet werden. Beispielsweise erfordert die Implementation des Interfaces `Updateable` im Allgemeinen die Zuordnung einer `Update`-Instanz. Diese implementiert das Interface `Executable` und wird zur Ausführung einer entsprechenden Komponente (z. B. Thread oder Timer) zugeordnet (Abbildung 6.9).

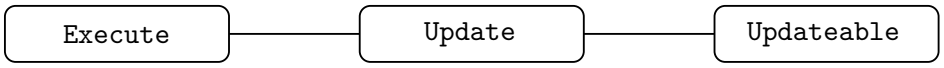


Abbildung 6.9: Ausführung eines `Updateable`

Die konsequente Anwendung des *fine grained*-Ansatzes (siehe Abschnitt 3.2.2) untersagt die direkte Verbindung der ausführenden Komponenten mit einem `Updateable` auf der Implementationsebene der Komponenten. Im Ergebnis müssten alle ausführenden Komponenten mit einer Schnittstelle zu `Updateable`-Instanzen ausgestattet werden. Dies führt zu erheblichen Redundanzen im Code und in den Komponentenspezifikationen.
Durch einen entsprechenden Konnektor kann die ausführende Komponente in der Konfigurationsebene mit einer Instanz vom Typ `Updateable` verbunden werden, ohne

dass die Implementation der Komponenten davon beeinflusst wird. Der Konnektor legt die notwendige `Update`-Instanz an und verknüpft diese sowohl mit der ausführenden Komponente als auch mit der `Updateable`-Instanz. (Abbildung 6.10).

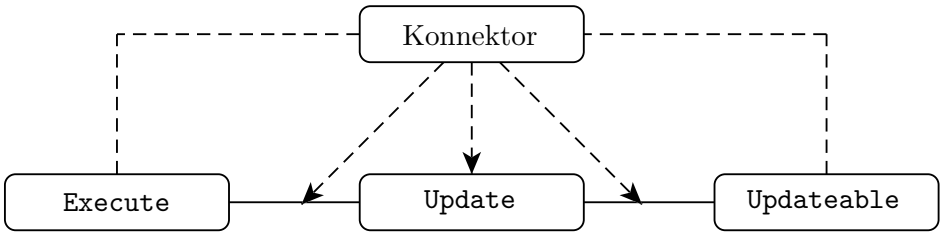


Abbildung 6.10: Konnektor zur Ausführung eines `Updateable`

### 6.3.2 Adapterkonzept

Durch die Einführung von Konnektoren wird die Verknüpfung von Komponenten vereinfacht. Die Abstraktion der Zuordnung von Instanzen zu einer Komponente durch die generische `connect`-Methode ermöglicht die flexible Erweiterung der Konfigurationsobjekte. Nachteilig wirkt sich aus, dass keine Abstraktion der Zielkomponente der Verknüpfung erreicht wird. Im Beispiel des `Updateable` muss für jede ausführende Komponente ein Konnektor implementiert werden.
Dies kann durch die Anwendung eines Adapterkonzeptes vermieden werden. Analog zu den Konnektoren bilden die Adapter eine erweiterbare Menge von Objekten. Diese erstellen auf der Basis der Klassenobjekte entsprechende Konfigurationsmuster, um Verbindungen zwischen Konfigurationsobjekten herzustellen. Im Gegensatz zu den Konnektoren erfordert die Adaption einer Komponente an eine fremde Schnittstelle entsprechendes Wissen. Dieses ist nicht Bestandteil der Komponentenspezifikationen. Es werden daher keine Adapter generiert. Das SARTURIS-Framework bietet auf der Konfigurationsebene die entsprechenden Mechanismen für den Aufbau einer flexiblen Bibliothek von Adaptern an. Die Organisation dieser Bibliothek erfolgt im Dateisystem unter Nutzung des Modulkonzeptes von PYTHON.
Die Anwendung des Adapterkonzeptes auf das diskutierte Beispiel der Ausführung eines `Updateable` (Abschnitt 6.3.1) führt auf einen Adapter, der eine beliebige `Updateable`-Instanz durch Erzeugen und Verknüpfen einer `Update`-Komponente in eine Konfigurationsstruktur umwandelt. Diese kann jeder Komponente mit einer `Executable`-Schnittstelle zugeordnet werden. Diese Zuordnung erfolgt durch den elementaren Konnektor, der aus den Komponentenspezifikationen generiert wurde.

In diesem Zusammenhang stellt das Adapterkonzept eine Erweiterung des Konnektorenkonzeptes dar. Die Implementation der Adapter erfolgt wissensbasiert, d. h. häufig wiederkehrende Konfigurationsmuster werden problembezogen durch Adapter abgebildet. Mögliche Änderungen und Erweiterungen werden in den Adaptern implementiert und betreffen somit direkt alle abgeleiteten Konfigurationen.

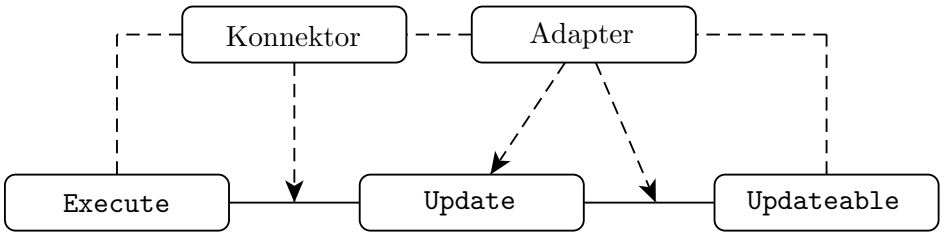


Abbildung 6.11: Adapter zur Ausführung eines `Updateable`

Ein Problem bei der Anwendung des Adapterkonzeptes stellt die automatische Bestimmung geeigneter Adapter dar. Die konsequente Anwendung des Konzeptes erlaubt die Adaption eines Konfigurationsobjektes in mehreren Stufen. Dabei treten Mehrdeutigkeiten auf (Abbildung 6.12). Diese müssen durch einen entsprechenden Algorithmus behandelt werden. Dabei sind die Kanten des resultierenden Graphen zu wichten.

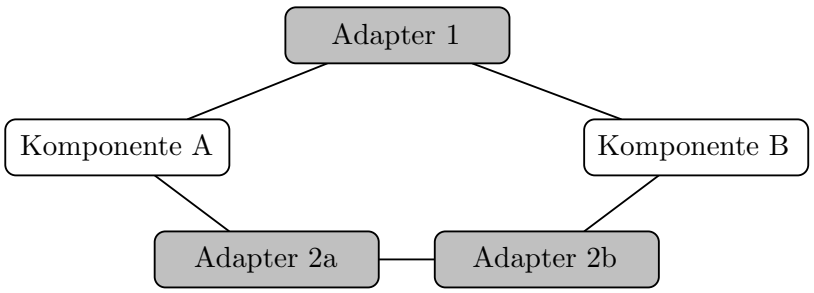


Abbildung 6.12: Mehrdeutigkeit der Adapter

Die vollständige Implementation einer automatischen Adaption der Komponenten ist jedoch nicht Gegenstand dieser Arbeit. Aus diesem Grund und zur Vermeidung zirkularer Referenzen wird die Bestimmung geeigneter Adapter auf eine Ebene begrenzt.

### 6.3.3 Konfigurationsmuster Steckerleiste

Für die Kombination komplexer Konfigurationsstrukturen bietet sich die Übertragung des Konzeptes der Steckerleiste auf die Konnektoren an. Eine Steckerleiste stellt eine standardisierte Schnittstelle für die Verbindung eines Sockels und eines Steckers dar. Dabei erfolgt die Standardisierung auf geometrischer Ebene durch die Bauform der Elemente. Auf elektrischer und logischer Ebene ist eine entsprechende Spezifikation notwendig. Dabei können gezielt einzelne Kanäle der Schnittstelle (Pins) ungenutzt bleiben, ohne die prinzipielle Austauschbarkeit zu verletzen. Die Signale der ungenutzten Pins werden für die spezfische Anwendung nicht benötigt.
Die Übertragung des Konzepts bietet sich vor allem in Anwendungsfällen an, in denen häufig optionale Referenzen konfiguriert werden. Für die Modellentwicklung werden in der Regel nicht alle Ausgabeschnittstellen der numerischen Simulation benutzt. Im Gegensatz dazu werden diese bei der Anwendung im VR-System benötigt. Die Bedienerschnittstellen treten je nach Anwendungsfall in unterschiedlicher Komplexität auf. Um eine flexible Austauschbarkeit (siehe Abbildung 6.2) der entsprechenden Konfigurationen zu ermöglichen, sind entsprechende Schnittstellen zu implementieren.

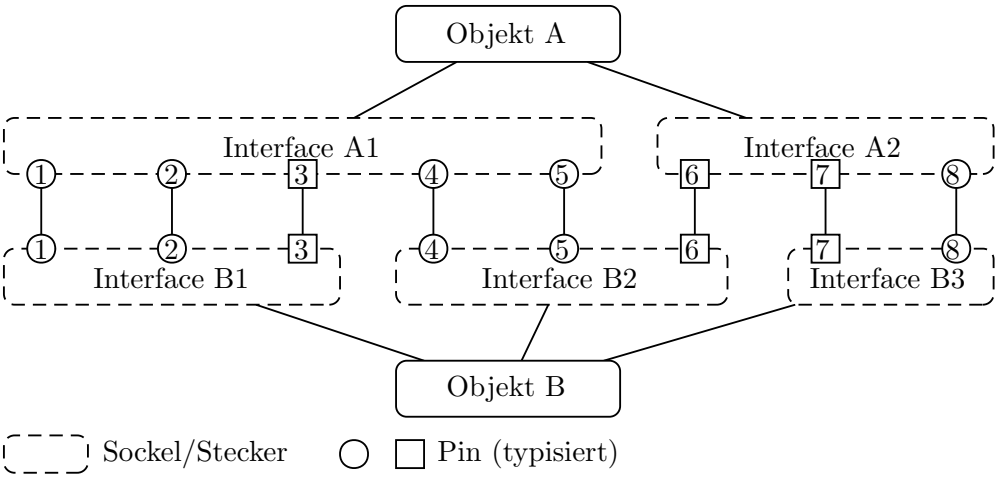


Abbildung 6.13: Konfigurationsmuster Steckerleiste

Die Anwendung des Musters der Steckerleiste führt zur Definition von Objekten für Pins, Sockel und Stecker (Abbildung 6.13). Dabei ist die Anzahl der Pins je Sockel bzw. Stecker beliebig. Bei der Verbindung eines Steckers und eines Sockels werden die zu verbindenden Pins durch ihre Namen identifiziert. Die Typprüfung erfolgt bei der Verbindung von Stecker und Sockel. Die Adaption von Ein- und Ausgabeschnittstellen

(siehe Abbildung 4.3 und Abbildung 4.5) erfolgt automatisch. Nicht verbundene Pins werden toleriert, wenn sie optionale Referenzen repräsentieren.
Die notwendigen Speicher und Komponenten zur Aktualisierung der Daten werden durch die Steckerleiste angelegt und verknüpft. Im Ergebnis werden mehr Daten gespeichert und ausgetauscht als theoretisch für die Ausführung der Simulationsanwendung notwendig sind. Aufgrund der Leistungsfähigkeit der verfügbaren Rechner wird dieser Zusatzaufwand als nicht kritisch bewertet. Die Vorteile bei der flexiblen Konfiguration komplexer Schnittstellen können am Beispiel der Referenzanwendung nachgewiesen werden (Abbildung 6.14 und Abbildung 6.15).

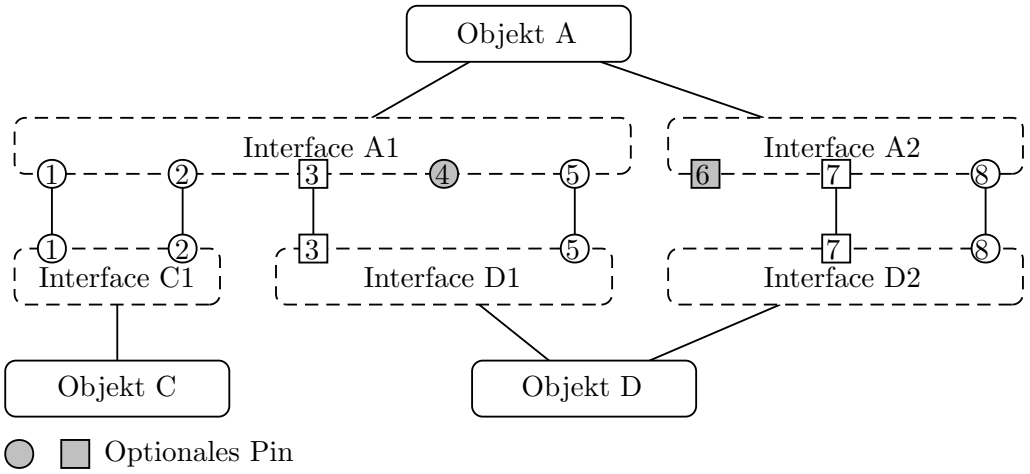


Abbildung 6.14: Austausch der Steckerleiste

```
1  # Variante 1: 2 Joysticks
2  joy1        = Joystick()
3  joy2        = Joystick()
4  wheelloader = WheelLoader()
5  wheelloader.connect(joy1, {'Heben' : 'AxisX', 'Kippen': 'AxisY'})
6  wheelloader.connect(joy2, {'Fahren': 'AxisX', 'Lenken': 'AxisY'})
7
8  # Variante 2: Komplette Kabine
9  cabin       = Cabin1()
10 wheelloader = WheelLoader()
11 wheelloader.connect({'Heben' : 'JoyX' , 'Kippen': 'JoyY',
12                      'Fahren': 'Pedal', 'Lenken': 'SteeringWheel'
13                      'Hupe'  : 'Horn'})
```

Abbildung 6.15: Flexible Konfiguration der Schnittstellen des Radladers

### 6.3.4 Konfigurationsmuster Kinematikanimation

Ein elementares Konfigurationsmuster bei der interaktiven Simulation in einer virtuellen Umgebung wird durch die Animation der simulierten Maschine abgebildet. Dabei werden die Koordinaten der Bauteile der Maschine durch eine Kinematikkomponente berechnet. Die zugehörigen Transformationen werden an den Szenegraphen übergeben. Dieser stellt anschließend die zugeordneten 3D-Geometriemodelle entsprechend dar (siehe Abschnitt 4.4.3).
Die Implementation dieser Funktionalität in einer Komponente widerspricht dem *fine grained*-Ansatz. Die Aktualisierung der Transformation eines 3D-Objektes ist eine Standardfunktion des Szenegraphen und wäre redundant, da nicht nur das Maschinenmodell positioniert und animiert wird. Die Position eines 3D-Objektes in der virtuellen Umgebung wird aus unterschiedlichen Quellen definiert.
Die Aktualisierung der Animation erfolgt nicht automatisch, sondern ist aus Gründen einer deterministischen Synchronisation durch die Konfiguration zu definieren (siehe Abschnitt 3.2.2).

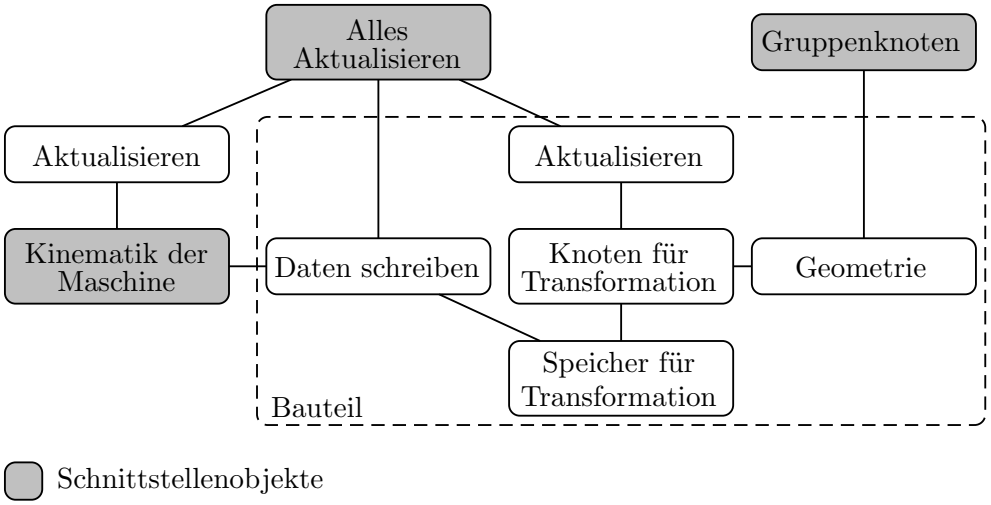


Abbildung 6.16: Konfiguration der animierten Maschine

Im Ergebnis entstehen für die Konfiguration einer animierten Visualisierung umfangreiche Konfigurationen aus den elementaren Konfigurationsobjekten des Komponentensystems (Abbildung 6.16). Für die Animation des Radladers in der Referenzanwendung sind je nach Anwendungsfall 80 bis 100 elementare Konfigurationsobjekte zu erzeugen und zu verknüpfen. Als Schnittstellenobjekte werden neben der Kinematik

nur der Gruppenknoten des Radladers sowie ein Element zum Aktualisieren aller relevanten Daten (unter Beachtung der Reihenfolge) benötigt.
Sowohl die Struktur als auch die Semantik der einzelnen Konfigurationsobjekte sind auf andere Maschinen und Anwendungsfälle übertragbar. Durch die Implementation eines entsprechenden Musters kann die Konfiguration deutlich vereinfacht werden. Im einfachsten Fall reduziert sich der Aufwand auf die Zuordnung der Koordinatensysteme der Maschine zu den entsprechenden 3D-Geometrien (Abbildung 6.17).

```
1  from sarturis.models import WheelLoader
2  from sarturis.patterns import Animation
3
4  # Radladermodell (mit Kinematik)
5  wheelloader = WheelLoader()
6
7  # Animation
8  animation = Animation(wheelloader
9                        {'Vorderwagen' : 'vorderwagen.osg',
10                       {'Hinterwagen' : 'hinterwagen.osg',
11                       {'Kabine'      : 'kabine.osg'     ,
12                       ...
13                       {'Schaufel'    : 'schaufel.osg'   ,
```

Abbildung 6.17: Konfiguration der Radladeranimation

Die Betrachtung der Kinematikanimation zeigt einen neuen Aspekt in der Diskussion über die Granularität der Komponenten (siehe Abschnitt 3.2). Der als *fine grained* bezeichnete Ansatz mit wenig Funktionsumfang in den einzelnen Komponenten führt zu übersichtlichen Komponenten mit wenig redundantem Code. Nachteilig sind die resultierenden umfangreichen Konfigurationen. Ein entsprechender Ansatz mit einer Komponente, welche die Animation eigenständig realisiert (*coarse grained*) führt zu einer einfachen Konfiguration (analog zu Abbildung 6.17), da die zugehörige Anwendungslogik in der Komponente implementiert ist. Dies führt zu Redundanzen im Code des Komponentensystems, da Teile dieser Anwendungslogik allgemeingültig sind und in entsprechend generischen Komponenten wiederholt werden.
Das Konfigurationsmuster der Kinematikanimation verbindet die Vorteile beider Ansätze. Es ist nicht notwendig, Komponenten mit redundantem Code zu implementieren. Die Beschreibung der Anwendungslogik erfolgt in der Konfigurationssprache.

Diese ist einfacher und flexibler als die Sprache für die Implementation der Komponenten.

## 6.4 Integration von Funktionalität

Zur weiteren Flexibilisierung der Anwendung des Frameworks sind einfache Funktionen aus der Komponentenebene in die Konfigurationsebene zu verlagern. So müssen die durch das Simulationsmodell berechneten Zustandsgrößen für die Ansteuerung von Ausgabegeräten bzw. grafischen Anzeigen im Allgemeinen umgerechnet werden. Diese Umrechnung ist anwendungsspezifisch und häufig durch einfache Skalierungen, Offsets und Normierungen gekennzeichnet.
Die Implementation spezieller Umrechnungskomponenten ist daher nicht sinnvoll, da der Aufwand für die Spezifikation der Komponente wesentlich höher ist als der Aufwand für die Implementation der Umrechnung. Eine allgemeine Umrechnungskomponente erfordert entweder die Implementation einer umfangreichen Gleichung, deren Koeffizienten bei der Anwendung größtenteils 0 sind oder einen Funktionsparser, der die Umrechnungsfunktion dynamisch aus einer Zeichenkette erzeugt.
Die Anwendung des Funktionsparsers stellt eine akzeptable Lösung dar. Der Nachteil ist die Zuordnung von Ein- und Ausgaben zu den Variablen der Gleichung (Abbildung 6.18). Bei der Konfiguration in XML erschwert der Ersatz der reservierten Zeichen <, > und / die Lesbarkeit der Ausdrücke.

```
1  <Instance xsi:type="AnalogInputMath" Name="MyFunc" Formula="z1-z2">
2
3    <!-- Zuweisung von Eingaengen auf Variablen -->
4    <Input Variable="z10" Instance="Increment_plus"/>
5    <Input Variable="z20" Instance="Increment_minus"/>
6
7    <!-- Zuweisung von Ausdruecken auf Variablen -->
8    <Expression Variable="z1" Value="if(z10 &gt; 0, 1.0, 0.0)"/>
9    <Expression Variable="z2" Value="if(z20 &lt; 0, 1.0, 0.0)"/>
10
11 </Instance>
```

Abbildung 6.18: Konfiguration einer Berechnung in XML

Die Integration elementarer Funktionen in die Konfigurationsebene bietet einen Ansatz für die übersichtliche Umrechnung beliebiger Größen. Die bidirektionale

Integration der PYTHON-Bibliothek in C++ ermöglicht den gegenseitigen Funktionsaufruf von Objekten der Konfigurationsebene und Komponenten ([Lan09]). In der Konfigurationsebene werden Objekte definiert, welche mit entsprechenden Komponenten (Wrapper) verknüpft werden. Die Verknüpfung erfolgt durch das Konfigurationsobjekt der Komponente (Abbildung 6.19).
Durch die Anwendung des objektorientierten Konzeptes lässt sich jeder Komponente, die eine Hülle für Funktionsobjekte bildet, ein PYTHON-Klassenobjekt zuordnen. Dadurch kann die Typsicherheit der Zuordnung sichergestellt werden.
Die dem Funktionsobjekt zugeordneten Komponenten sind während der Konfiguration nicht vorhanden. Die zugehörigen Konfigurationsobjekte sind mit dem Funktionsobjekt verknüpft (Abbildung 6.20). Anhand der Konfigurationsobjekte werden die zugehörigen Komponenten beim ersten Funktionsaufruf zur Laufzeit der Simulationsanwendung ermittelt. Dazu wird der Namensdienst des Frameworks benutzt.

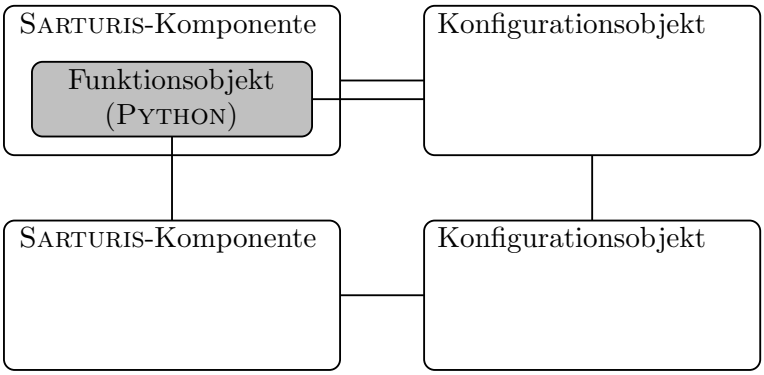


Abbildung 6.19: Interaktion zwischen Funktionsobjekten, Komponenten und Konfigurationen

Der zu dem in XML konfigurierten Beispiel (Abbildung 6.18) äquivalente PYTHON-Code (Abbildung 6.21) ist wesentlich übersichtlicher und entspricht der intuitiven Abbildung von Rechenoperationen durch Programmiersprachen.
Die Möglichkeit, einfache Funktionen wie Umrechnungen in der Konfigurationsebene abzubilden, erhöht die Flexibilität des Frameworks. Häufig sind Umrechnungen der Zustandsgrößen des Simulationsmodells an konkrete Ein- und Ausgabekomponenten gebunden. Die Umrechnung kann der Konfiguration dieser Komponenten zugeord-

net und parametrisiert werden. Dadurch entstehen flexible und wiederverwendbare Konfigurationsmuster.

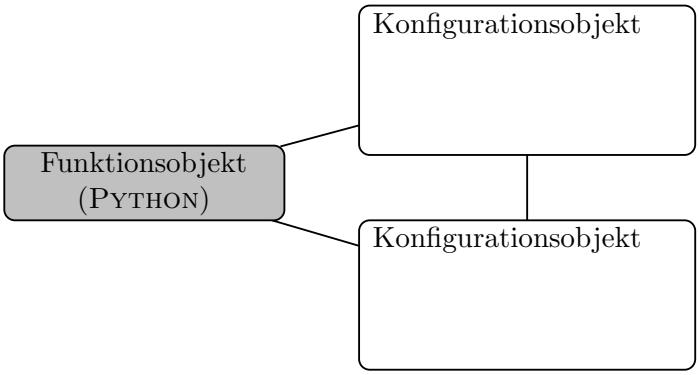


Abbildung 6.20: Funktionsobjekt während der Konfigurationsphase

```
1  class MyFuncObject(Updateable, AnalogInput):
2    # implementiert die Schnittstelle Updateable
3    def Update(self)
4      if (self.increment_plus > 0.0):
5        z1 = 1.0
6      else:
7        z1 = 0.0
8      if (self.increment_minus > 0.0):
9        z2 = 1.0
10     else:
11       z2 = 0.0
12
13     self.value = z1 - z2
14
15   # implementiert die Schnittstelle AnalogInput
16   def GetValue(self):
17     return self.value
```

Abbildung 6.21: Berechnung einer Funktion in Python

# 7 Virtuelle Prototypen in der Produktentwicklung

## 7.1 Verallgemeinerung des Modellbegriffs

Durch das komponentenbasierte SARTURIS-Framework wird ein flexibles und leistungsfähiges Softwaresystem für die interaktive Simulation mobiler Arbeitsmaschinen bereitgestellt. Die konsequente Anwendung dieser Technologie im Rahmen von Maschinenentwicklungen resultiert in einer permanenten Anpassung der Maschinenmodelle und Konfigurationen des Simulationssystems vor allem im Anwendungsfall der Maschinenentwicklung aus technischer Sicht (siehe Abschnitt 3.1.1). Unterschiedliche *virtuelle Prototypen* werden anhand identischer Aufgabenstellungen in der gleichen virtuellen Umgebung untersucht. Im Gegensatz zu den anderen grundlegenden Anwendungsfällen des SARTURIS-Frameworks entsteht der wesentliche Aufwand bei der Modellbildung und Konfiguration der virtuellen Maschine.
Die im Rahmen der komponentenorientierten Softwareentwicklung diskutierten Aspekte der Wiederverwendung sind auf die Modellbildung und Notation der Modelle übertragbar. Durch entsprechende objektorientierte Modellbeschreibungen und Schnittstellen lassen sich Teilmodelle parametrisch formulieren und zu komplexen Systemmodellen zusammensetzen. Dadurch werden Redundanzen vermieden und die Anpassung bzw. der Austausch von Modellstrukturen wesentlich vereinfacht.
Zur komponentenorientierten Beschreibung der physikalischen Zusammenhänge des Maschinenmodells ist die Sprache MODELICA geeignet ([EM97; FE98; Fri04; Fri11]). Diese ermöglicht neben der reinen Beschreibung der physikalischen Zusammenhänge die Definition von Schnittstellen sowie die Strukturierung der Modelle in Komponenten und Kompositionen. Durch die akausale Beschreibung der Gleichungen erhöht sich der Grad der Wiederverwendung, da kein Zuweisungsoperator existiert, der implizit die Richtung des Informationsflusses vorgibt.
Die Anwendung dieses Konzeptes auf die Modellbildung mobiler Arbeitsmaschinen erlaubt die effiziente Formulierung der Gleichungen der Maschinenmodelle in Form

von Komponenten ([Fre+09; Fre10; PF10]). Durch den geringen Grad der Abstraktion der Sprache MODELICA ist die Notation der Modelle in Textform für den Anwender übersichtlich. Grafische Editoren sind ebenfalls verfügbar ([Asg+11]).
Es konnte nachgewiesen werden, dass das beschriebene Konzept der Codegeneratoren für die automatisierte Erzeugung von SARTURIS-Komponenten zur Berechnung der Maschinenmodelle (siehe Abschnitt 3.2.3) direkt auf MODELICA-Modelle anwendbar ist ([Fre09]). Der Anwender des SARTURIS-Frameworks kann somit MODELICA-Modelle ohne C++-Kenntnisse in die Simulationsanwendung integrieren.
Nicht gelöst wird in diesem Zusammenhang der Umgang mit Informationen, welche nicht die mathematischen und physikalisch-technischen Zusammenhänge der virtuellen Arbeitsmaschine abbilden. Es zeigt sich, dass strukturelle Abhängigkeiten zwischen den Modellgleichungen und der Konfiguration der Simulationsanwendung bestehen, die mit den betrachteten Ansätzen der Modellbildung nicht abgebildet werden können. Veränderungen an der kinematischen Struktur einer Arbeitsausrüstung erfordern in der Regel auch entsprechende Anpassungen der 3D-Objekte und deren Konfiguration für die Visualisierung. Diese Anpassungen sind aufwändig und fehleranfällig, da redundante Informationen in verschiedenen Notationen zu pflegen sind. Der im MODELICA-Umfeld praktizierte Weg, diese Informationen mit *Annotations* abzubilden, ist für komplexe Entwicklungsprojekte nicht geeignet, da *Annotations* toolspezifisch und ohne Metamodell implementiert werden.
Der Begriff des Maschinenmodells ist daher zu erweitern. Im Kontext einer interaktiven Simulation in virtuellen Umgebungen sind nicht mehr ausschließlich die physikalischen Gleichungen und deren Parameter als Modell zu verstehen, sondern die Gesamtheit der Daten und Schnittstellen einschließlich der Datenformate und Notationen (Abbildung 7.1).
Die daraus resultierende Verallgemeinerung des Modellbegriffs umfasst neben dem Maschinenmodell aus Sicht der numerischen Simulation auch den Modellbegriff aus Sicht der virtuellen Realität (3D-Geometrie, Sound, etc.) sowie die Beschreibung von Schnittstellen. Die resultierende Integration und Kapselung der Modelldaten in einer einzigen Quelle vermeidet Redundanzen und ermöglicht die effiziente Verwaltung der Modelle in den entsprechenden entwicklungsbegleitenden Informationssystemen. Dadurch wird die Abbildung der interaktiven Simulation in den IT-Prozessen der Produktentwicklung wesentlich vereinfacht.

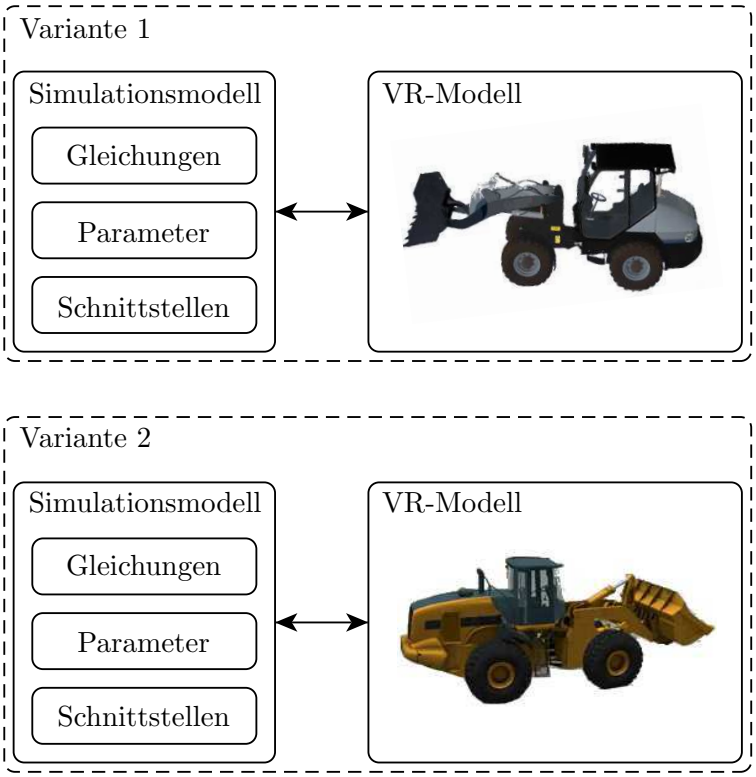


Abbildung 7.1: Verallgemeinerung des Modellbegriffs

## 7.2 Übertragung des Kompositionskonzeptes

Für die Definition eines leistungsfähigen Komponenten-Konzeptes zur Beschreibung der verallgemeinerten Modelle ist der Begriff der Komponente im Kontext der interakiven Simulation mobiler Arbeitsmaschinen zu erweitern. Es zeigt sich, dass mindestens zwei unterschiedliche Sichten (Abbildung 7.2) betrachtet werden müssen:

- **Fachwelt:** Aus der Sicht der Fachwelt ist eine Komponente ein physisches oder logisches Teilsystem der Maschine. Die Komponente ist nicht an eine bestimmte Domäne (Mechanik, Hydraulik, etc.) gebunden. Die resultierenden Strukturen werden im Allgemeinen durch Stücklisten beschrieben.

- **Simulationswelt:** Die Komponente der Simulationswelt beschreibt die Gleichungen, Algorithmen, Konfigurationen und Daten, die zur Abbildung des entsprechenden Teilmodells innerhalb der Simulationsanwendung notwendig sind. Dabei wird der erweiterte Modellbegriff aus der Kombination von numerischer Simulation und virtueller Realität verwendet (siehe Abschnitt 7.1).

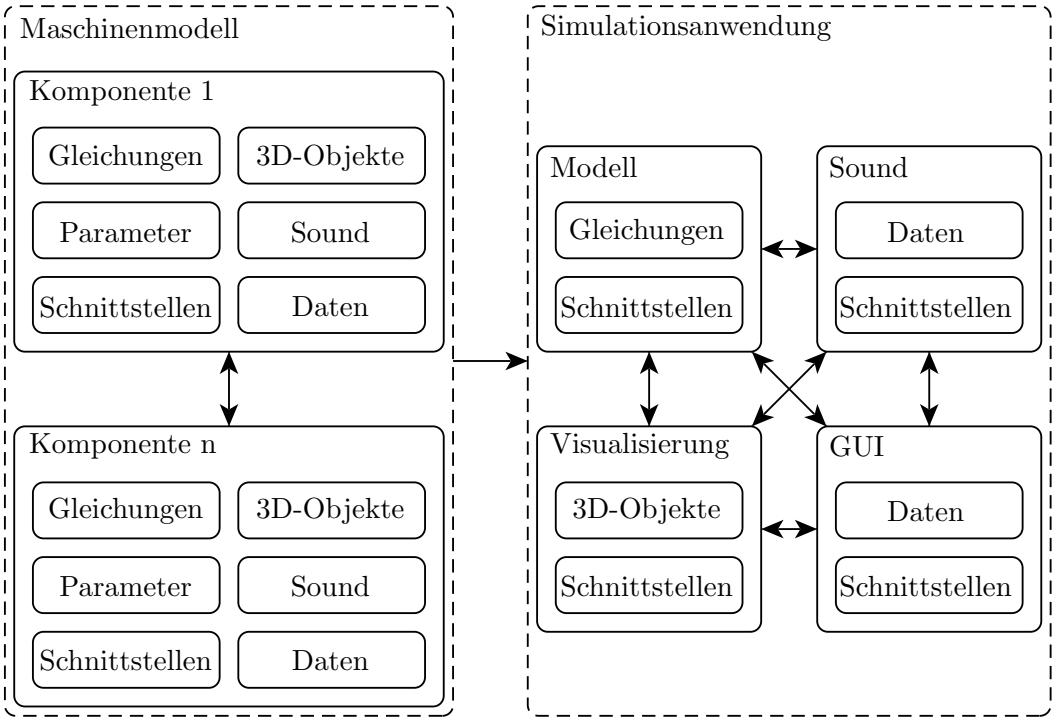


Abbildung 7.2: Komponenten im Maschinenmodell und der Simulationsanwendung

Diese Sichten bilden das Gerüst eines Referenzmodells für die interaktive Simulation ([EGL09; Ess+10; Kun+10; ELS09]). Dieses ist als Ebenenmodell aufgebaut und integriert neben den klassischen Bereichen der Modellbildung, Simulation und Darstellung der virtuellen Arbeitsmaschine auch Meta- und Produktdaten sowie den Prozess des Entwicklungsprojektes.
Die Manipulation des Maschinenmodells erfolgt in der Fachebene ([EGL09; Ess+10; Pen+12]). Durch die Beschreibung der Komponenten aus Sicht des Maschinenentwicklers ist der Grad der Wiederverwendung in dieser Ebene am größten. Die Integration aller notwendigen Daten in den Komponenten sowie die Abbildung der Strukturen in Form von Stücklisten vermeidet redundante Informationen.
Die Transformation der Informationen in die Simulationsebene erfolgt durch Automaten ([Fre+11; Pen+12]). Diese arbeiten wissensbasiert, analysieren das Gesamtmodell und generieren Gleichungen, Algorithmen und Konfigurationen für die Simulationsanwendung (Abbildung 7.3). Dieses Konzept überträgt das bei der Konfiguration mit Skriptsprachen definierte Prinzip der Komposition (siehe Abbildung 6.2) auf die Maschinenmodelle. Im Ergebnis entstehen komplexe Simulationsanwendungen auf der Basis formaler Beschreibungen der Maschine (Stücklisten, Modelle und Daten), des

VR-Systems (Visualisierung, Sound, Bewegung), der Schnittstellen und der virtuellen Welt (Datenbasis).

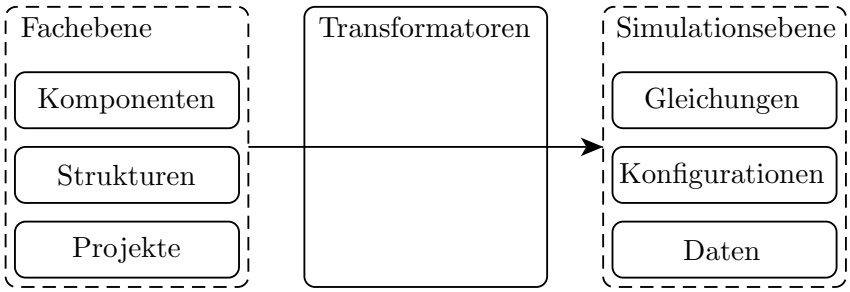


Abbildung 7.3: Transformation aus der Fachebene in die Simulationsebene

# 8 Zusammenfassung und Ausblick

Die vorliegende Arbeit beschreibt ein Framework für die interaktive Simulation mobiler Arbeitsmaschinen in virtuellen Umgebungen (SARTURIS). Die Notwendigkeit der interaktiven Simulation wird anhand der Analyse der Mensch-Maschine-Interaktion sowie der Betrachtung vorhandener Bedienermodelle abgeleitet. VR-Systeme bieten eine ideale Voraussetzung, die notwendige Immersion des Bedieners zu erzeugen. Die Bedienhandlungen in der virtuellen Umgebung sind mit denen der realen Welt vergleichbar. Damit können die Simulationsergebnisse unter Voraussetzung einer entsprechenden Modellgüte für die Bewertung von Konzepten, Komponenten und Systemen genutzt werden.

Sowohl die Komplexität der interaktiven Simulation in VR-Systemen als auch die Vielfältigkeit an Entwicklungsaufgaben erfordern ein flexibles Softwaresystem. Dieses konnte auf der Basis eines komponentenbasierten Entwicklungsansatzes geschaffen werden. Durch diesen Ansatz werden die einzelnen Problemfelder in klar abgegrenzte Bereiche unterteilt. Die Entwicklung, Wartung und Pflege der Komponenten wird dadurch wesentlich vereinfacht. Die Integration zu einem Gesamtsystem erfolgt auf der Basis von Schnittstellen, die intuitiv für die Problemdomäne der interaktiven Simulation in virtuellen Umgebungen definiert wurden.

Am Beispiel einer Referenzanwendung wurde die Funktionalität des SARTURIS-Frameworks nachgewiesen. Die vorhandene Implementation verknüpft freie Bibliotheken und Standards ([GTK; OSG; VRJ; VLC]) sowie etablierte Komponentenmodelle (CORBA) und komplexe VR-Frameworks (VRJUGGLER). Dabei wird der Funktionsumfang dieser Bibliotheken und Frameworks nicht erweitert, sondern durch ein einheitliches Komponentenmodell zu einem Gesamtsystem integriert. Damit folgt die definierte Architektur des SARTURIS-Frameworks den Prinzipien *Separation of Concerns* und *Unification of Concepts* ([NA00]).

Anhand der vorliegenden Implementation zeigt sich, dass bei einer flexiblen, möglichst nicht auf eine Anwendung zugeschnittenen VR-Simulations-Plattform der Aufwand für die Konfiguration eines Anwendungsfalles enorm steigt. Unter den Gesichtspunkten der Wiederverwendung und Kapselung steigt die Menge der Komponenten,

während der Funktionsumfang einer einzelnen Komponente sinkt (*fine grained*-Ansatz). Verschiedene Anwendungsfälle lassen sich aus der Menge der verfügbaren Komponenten konfigurieren. Die Komplexität sowie die Menge der zu einem Anwendungsfall gehörenden Elemente verschieben den Aufwand für die Erstellung einer Simulationsanwendung auf die Konfiguration. Die Aspekte der Wiederverwendung und Parametrisierung von Konfigurationsfragmenten gewinnen dadurch an Bedeutung. Auf Konfigurationsebene resultieren daraus die gleichen Anforderungen hinsichtlich der Kapselung, Vererbung und Schnittstellendefinition wie auf der Implementationsebene.

Die Konfiguration mittels einer strukturierten Auszeichnungssprache wird am Beispiel von XML beschrieben. Die Leistungsfähigkeit dieser Technologie in Bezug auf die Wiederverwendung und Parametrisierung von Konfigurationen ist begrenzt. Am Beispiel von PYTHON wird die Eignung von Skriptsprachen zur Abbildung eines Kompositionssystems gezeigt. Dabei wird in der Konfigurationsebene eine Abbildung der Komponentenstruktur erzeugt, auf deren Basis eine typsichere Konfiguration des Simulationssystems erfolgen kann. Die Überprüfung auf Korrektheit der Konfiguration erfolgt in der Konfigurationsebene ohne dass das Simulationssytem benötigt wird.

Die Funktionalität von PYTHON gestattet den Aufbau komplexer Konfigurationsobjekte, welche standardisierte Komponentenmuster repräsentieren. Die damit abgebildeten Konfigurationen sind überschaubar und benötigen nur wenige Instanzen zur Abbildung einer Simulationsanwendung. Die daraus resultierende einfache Konfiguration wird mit den Vorteilen eines feingranularen Komponentensystems kombiniert.

Die Integration von Funktionalität des Simulationssystems in die Konfigurationsebene ermöglicht einfache Umrechnungen unter Anwendung der Skriptsprache PYTHON. Der Aufwand für die Erstellung von Komponenten mit trivialen Funktionen entfällt. Dadurch erhöht sich die Übersichtlichkeit und Flexibilität des Frameworks. Die gesamte Funktionalität einer Simulationsanwendung kann beliebig zwischen den Sprachen C++ und PYTHON verteilt werden. Damit wird das Konzept *Applications = Components + Scripts* ([NA00; SN99]) umgesetzt.

Der im Bereich der Echtzeitsimulation populäre Ansatz der Codegeneration zur Abbildung der Maschinenmodelle wird durch die Komponentenarchitektur wesentlich unterstützt. Vorhandene Ansätze zur Abbildung ganzheitlicher Simulationen mit Codeexport benötigen in der Regel die manuelle Anpassung des generierten Codes ([Böh01; Dro04]). Im Rahmen von konkreten Produktentwicklungen ist diese Vorgehensweise aufgrund des Aufwandes und der Redundanzen nicht akzeptabel.

Der komponentenbasierte Ansatz vereinheitlicht die Integration der generierten Codes für die Gleichungen des Maschinenmodells. Durch die Adaption des Komponentenmodells können die generierten Codes in entsprechende Module übersetzt und in das Gesamtsystem integriert werden. Am Beispiel der Fahrsimulation eines Radladers wurde die Modellbildung und automatisierte Erstellung der entsprechenden Komponenten durchgeführt. Für diese Referenzanwendung wurden die Modelle in Gleichungsform notiert und durch ein Computer-Algebra-System in C++-Code transformiert. Die Eingabe der Gleichungen erfolgt nicht objektorientiert. Einer geeigneten Strukturierung und Wiederverwendung der Modelle sind somit enge Grenzen gesetzt.

Eine wesentlich elegantere Möglichkeit, Modelle in Form von Gleichungen zu formulieren und in SARTURIS zu verwenden, bietet MODELICA. Mit MODELICA lassen sich gleichungsbasierte Simulationsmodelle objektorientiert formulieren. Durch Abstraktion und Definition sinnvoller Schnittstellen können Modellstrukturen sehr einfach wiederverwendet, erweitert und ausgetauscht werden. Die Organisation der Modelle in Bibliotheken gestattet die übersichtliche Verwaltung. Die leistungsfähige Standardbibliothek bietet Modelle für wesentliche physikalisch-technische Problemstellungen und ist aufgrund ihrer Verbreitung hinreichend getestet.

Mit Hilfe eines MODELICA-Compilers lassen sich sowohl C-Code als auch Dokumente mit Metainformationen erzeugen. Diese sind geeignet, den C-Code automatisch anzureichern, so dass eine SARTURIS-Komponente entsteht ([Fre09; Fre+09; Fre10; PF10]). Dieser Prozess kann vollständig automatisiert werden, so dass ausschließlich das MODELICA-Modell als Informationsquelle dient. Ein generischer Ansatz für die Integration von Simulationskomponenten wird durch das FMI-Interface definiert ([Blo+11; SNK11; SVS11]). Dieses beschreibt einen allgemeinen Ansatz für die Implementation von Komponenten zur Beschreibung gleichungsbasierter Simulationsmodelle. Dabei werden sowohl die verwendete Simulationsplattform als auch die Notation der Modelle abstrahiert.

Die Integration in SARTURIS erfolgt durch eine allgemeine FMI-Komponente, welche das Modell dynamisch lädt und die Schnittstellen verbindet (Abbildung 8.1). Der Einsatz von PYTHON zur Konfiguration erlaubt die Analyse der Struktur des FMI-Moduls in der Konfigurationsebene. Dadurch kann die Konfiguration der FMI-Komponente dynamisch spezialisiert werden.

Das Erzeugen von SARTURIS-Komponenten aus MODELICA-Modellen ist nicht mehr notwendig, da die Modellberechnung durch das entsprechende FMI-Objekt erfolgt.

Durch die Adaption des FMI-Standards werden zukünftige Modellbildungsmethoden und Notationen durch das Sarturis-Framework unterstützt (Abbildung 8.1).

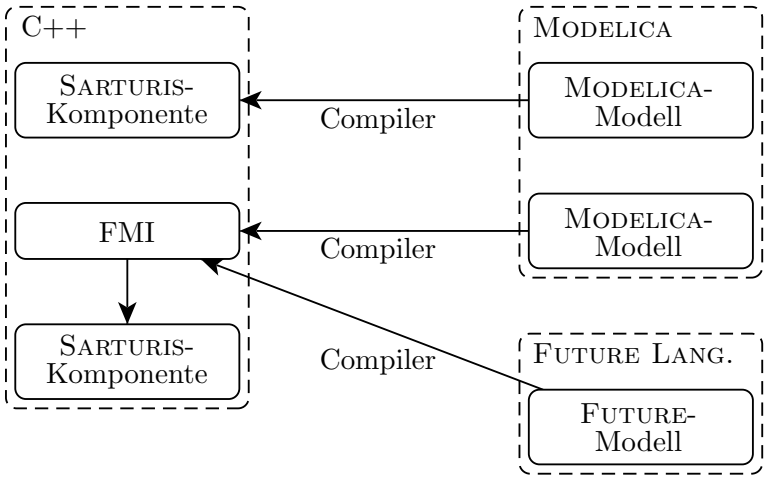


Abbildung 8.1: Integration des FMI-Interface in Sarturis

Das in der vorliegenden Arbeit benutzte Radladermodell bildet einen komplexen Produktentwicklungsprozess nur unzureichend ab. Die Variation von Teilsystemen und der beschreibenden Modelle mit dem Ziel der Bewertung von Konzepten und Varianten durch die interaktive Simulation ist ohne entsprechende Referenzprozesse kaum beherrschbar ([EGL09; Ess+10; ELS10; ELS09]). Eine effiziente Anpassung und Wiederverwendung vorhandener Modelle erfordert die ganzheitliche Sicht auf den Entwicklungsprozess und die damit verbundenen Produktdaten ([Ess+10; Kun+10; Pen+12]). Die im Rahmen dieser Arbeit präsentierten Ansätze sind noch nicht ausreichend und müssen im Rahmen weiterer Forschungsarbeiten ergänzt werden.
Die für das vorliegende Radladermodell getroffenen wesentlichen Vereinfachungen bei der Modellbildung bieten ebenfalls viel Raum für Verbesserungen. Dies betrifft vor allem die Modelltiefe und die abgebildete Genauigkeit. Die verwendeten Reifenmodelle sind stark vereinfacht und ausschließlich für die grobe Abbildung des Fahrverhaltens des Radladers geeignet. Weitergehende Untersuchungen mit dem Ziel der Bewertung von Fahrdynamik und Fahrkomfort erfordern die Abbildung komplexerer Reifenmodelle. Die in dieser Arbeit getroffenen Aussagen zu den Schnittstellen zum Starrkörper- und Antriebsmodell sowie zur Datenbasis sind übertragbar.
Prozessmodelle für konkrete Arbeitsprozesse mobiler Arbeitsmaschinen wurden in dieser Arbeit nicht behandelt. Die im Framework definierten Schnittstellen zu den vor-

liegenden Maschinenmodellen konnten bereits erfolgreich eingesetzt werden ([GKK10; KKG11]). Auf dem Gebiet der Prozessmodelle für mobile Arbeitsmaschinen besteht dennoch entsprechender Forschungsbedarf.

Die Implementation des Simulationssystems wurde vollständig auf der Basis von OpenSource-Software durchgeführt. Diese betrifft sowohl die Komponenten selbst als auch die komplette Infrastruktur für das Übersetzen und Verteilen der Software sowie die Realisierung des interaktiven Fahrsimulators. Der komponentenbasierte Ansatz ist gerade in Verbindung mit freien Bibliotheken, Tools und Standards sehr leistungsfähig, da problemorientiert die jeweils „beste“ Lösung ohne das Risiko von Fehlinvestitionen gesucht werden kann. Strategische Entscheidungen für den Einsatz einer bestimmten Bibliothek erfolgen ohne die Betrachtung monetärer Aspekte und ohne die Zwänge kommerzieller „Partnerschaften“.
Der Funktionsumfang des Sarturis-Frameworks wird hauptsächlich durch die eingesetzten Bibliotheken und Tools bestimmt. Der Anspruch dieser Arbeit ist die Entwicklung eines einheitlichen Komponentenmodells für die interaktive Simulation mobiler Arbeitsmaschinen, die Auswahl geeigneter Bibliotheken und Tools sowie deren Integration.

*Außerdem erlaubt es Software, unterschiedliche Aspekte auf das raffinierteste miteinander zu verknüpfen. Das Zusammenspiel von Berechnung, Kommunikation, Datenspeicherung, Auswertung und Visualisierung sowie angepassten Konzepten der Mensch-Maschinen-Interaktion bis hin zu Einbindung in physikalische Vorgänge schafft Gestaltungsmöglichkeiten, die nur durch unsere Phantasie begrenzt sind.*

Broy, M.: Cyber-Physical Systems, Springer-Verlag 2010.

# Literaturverzeichnis

[Abe+09] Abendroth, B.; Bruder, R.; Fuchs, K.; Schramm, T. „Beurteilung von Systemen im Fahrzeug - welche Messmethoden sind geeignet?“ In: *Zeitschrift für Arbeitswissenschaft* 3 (2009), S. 223–231.

[AC01] Attardi, G.; Cisternino, A. „Reflection Support By Means of Template Metaprogramming“. In: *Proceedings of Third International Conference on Generative and Component-Based Software Engineering, LNCS*. Springer-Verlag, 2001, S. 118–127.

[ACR93] Ascher, U. M.; Chin, H.; Reich, S. „Stabilization of DAEs and invariant manifolds“. In: *Numer. Math* 67 (1993), S. 131–149.

[Aki+03] Akiduki, H.; Nishiike, S.; Watanabe, H.; Matsuoka, K.; Kubo, T.; Takeda, N. „Visual-vestibular conflict induced by virtual reality in humans.“ In: *Neuroscience Letters* 340.3 (2003), S. 197–200.

[Aki+05] Akizuki, H.; Uno, A.; Arai, K.; Morioka, S.; Ohyama, S.; Nishiike, S.; Tamura, K.; Takeda, N. „Effects of immersion in virtual reality on postural control“. In: *Neuroscience Letters* 379.1 (2005), S. 23–26.

[And04] Andresen, A. *Komponentenbasierte Softwareentwicklung*. München, Wien: Carl Hanser Verlag, 2004.

[Ant+09] Antonson, H.; Mårdh, S.; Wiklund, M.; Blomqvist, G. „Effect of surrounding landscape on driving behaviour: A driving simulator study“. In: *Journal of Environmental Psychology* 29.4 (2009), S. 493–502.

[Arn+11] Arnold, M.; Burgermeister, B.; Führer, C.; Hippmann, G.; Rill, G. „Numerical methods in vehicle system dynamics: state of the art and current developments“. In: *Vehicle System Dynamics* 49.7 (2011), S. 1159–1207.

[Arn98] Arnold, M. *Zur Theorie und zur numerischen Lösung von Anfangswertproblemen für differentiell-algebraische Systeme von höherem Index*. Fortschritt-Berichte VDI : Reihe 20, Rechnerunterstützte Verfahren; 264. Düsseldorf: VDI-Verlag, 1998, IX, 186 S.

[AS09] Arnold, M.; Schierz, T. „Effizienz und Robustheit numerischer Kopplungsalgorithmen im MODELISAR Co-Simulation-Interface“. In: *Tagungsband ASIM-Treffen STS/GMMS und DASS*. 2009.

[Asg+11] Asghar, S. A.; Tariq, S.; Torabzadeh-Tari, M.; Fritzson, P.; Pop, A.; Sjölund, M.; Vasaiely, P.; Schamai, W. „An Open Source Modelica Graphic Editor Integrated with Electronic Notebooks and Interactive Simulation“. In: *Proceedings of the 8th International Modelica Conference*. Dresden, Germany, 2011, S. 739–747.

[Aßm03] Aßmann, U. *Invasive software composition : with 9 tables*. Literaturverz. S. [321] - 334. Berlin [u.a.]: Springer, 2003, XII, 334 S.

[Bau72] Baumgarte, J. „Stabilization of constraints and integrals of motion in dynamical systems“. In: *Computer Methods in Applied Mechanics and Engineering* 1.1 (1972), S. 1–16.

[Bay+12] Bayrasy, P.; Burger, M.; Dehning, C.; Kalmykov, I.; Speckert, M. „Applications for MBS-FEM-coupling with MpCCI using automotive simulation as example“. In: *Proceedings of the 2nd Commercial Vehicle Technology Symposium (CVT 2012)*. Aachen: Shaker, 2012, S. 385–394.

[BCP95] Brenan, K. E.; Campbell, S. L.; Petzold, L. R. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. Philadephia, PA: Society for Industrial und Applied Mathematics, 1995.

[BDL09] Bonvini, M.; Donida, F.; Leva, A. „Modelica as a design tool for hardware-in-the-loop simulation“. In: *Proceedings of the 7th International Modelica Conference*. Como, Italy, 2009, S. 378–385.

[Bea99] Beater, P. *Entwurf hydraulischer Maschinen - Modellbildung, Stabilitätsanalyse und Simulation hydrostatischer Antriebe und Steuerungen*. VDI-Buch Series. Springer-Verlag GmbH, 1999.

[Ben+08] Bengel, G.; Baun, C.; Kunze, M.; Stucky, K.-U. *Masterkurs Parallele und Verteilte Systeme*. Wiesbaden: Vieweg + Teubner, 2008.

[Ben94] Bennett, S. *Real-time computer control - An introduction (second edition)*. Prentice Hall International series in systems and control engineering. Prentice Hall, 1994, S. I–XIV, 1–417.

[Bie+01] Bierbaum, A.; Just, C.; Hartling, P.; Meinert, K.; Baker, A.; Cruz-Neira, C. „VR Juggler: A Virtual Platform for Virtual Reality Application Development“. In: *Proceedings of the Virtual Reality 2001 Conference (VR'01)*. VR '01. Washington, DC, USA: IEEE Computer Society, 2001, S. 89–96.

[Blo+11] Blochwitz, T.; Otter, M.; Arnold, M.; Bausch, C.; Clauß, C.; Elmquist, H.; Junghanns, A.; Mauss, J.; Monteiro, M.; Neidhold, T.; Neumerkel, D.; Olsson, H.; Peetz, J. V.; Wolf, S. „The Functional Mockup Interface for Tool independent Exchange of Simulation Models“. In: *Proceedings of the 8th International Modelica Conference*. Dresden, Germany, 2011, S. 105–114.

[BN05] Bouchner, P.; Novotny, S. „System with driving simulation device for HMI measurements“. In: *Proceedings of the 9th WSEAS International Conference on Systems*. ICS'05. Stevens Point, Wisconsin, USA: World Scientific, Engineering Academy und Society (WSEAS), 2005, 80:1–80:6.

[BO03] Beater, P.; Otter, M. „Multi-Domain Simulation: Mechanics and Hydraulics of an Excavator“. In: *Proceedings of the 3rd International Modelica Conference*. Linköping, Sweden, 2003, S. 331–340.

[Böh01] Böhler, H. „Traktormodell zur Simulation der dynamischen Belastungen bei Transportfahrten“. Dissertation. Technische Universität München, 2001.

[Bos94] Bossel, H. *Modellbildung und Simulation : Konzepte, Verfahren und Modelle zum Verhalten dynamischer Systeme ; ein Lehr- und Arbeitsbuch*. 2., veränd. Aufl. mit verb. Simulationssoftware. Literaturverz. S. 391 - 394. Braunschweig [u.a.]: Vieweg, 1994, 402 S.

[BP00] Borchsenius, F.; Pfeiffer, F. „Simulation nichtlinearer hydraulischer Steuersysteme“. In: *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik* 80.S2 (2000), S. 505–506.

[BR96] Baillot, Y.; Rolland, J. *Fundamental principles of tracking technology for virtual environments*. Technical Report TR96-004. Orlando, USA: Center for Research, Education in Optics und Lasers (CREOL), 1996.

[Bri09] Brill, M. *Virtuelle Realität*. Informatik im Fokus. Literaturverz. S. [175] - 179. Berlin [u.a.]: Springer, 2009, IX, 184 S.

[Bub77] Bubb, H. „Ergonomie des Mensch-Maschine-Systems“. Habilitation. Technische Universität München, 1977.

[Bür+11] Bürkle, K.; Schmauder, M.; Hoske, P.; Kunze, G. „Virtuelle Abbildung dynamischer Prozesse zur Sichtbewertung bei Erdbaumaschinen“. In: *Technische Sicherheit* Heft 4 (2011).

[BW98] Baraff, D.; Witkin, A. „Large steps in cloth simulation“. In: *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '98. New York, NY, USA: ACM, 1998, S. 43–54.

[Car84] Cardelli, L. „A semantics of multiple inheritance.“ In: *Proc. of the international symposium on Semantics of data types*. New York, NY, USA: Springer-Verlag New York, Inc., 1984, S. 51–67.

[CCK04] Choi, S. W.; Chang, S. H.; Kim, S. D. „A Systematic Methodology for Developing Component Frameworks“. In: *FASE*. 2004, S. 359–373.

[CH96] Cohen, S. D.; Hindmarsh, A. C. „CVODE, a stiff/nonstiff ODE solver in C“. In: *Comput. Phys.* 10.2 (1996), S. 138–143.

[Chi95] Chiba, S. „A metaobject protocol for C++“. In: *Proceedings of the tenth annual conference on Object-oriented programming systems, languages, and applications*. OOPSLA '95. New York, NY, USA: ACM, 1995, S. 285–299.

[CK06] Cellier, F.; Kofman, E. *Continuous System Simulation*. Springer, 2006.

[CKW98] Chuang, T.-R.; Kuo, Y. S.; Wang, C.-M. „Non-intrusive object introspection in C++: architecture and application“. In: *Proceedings of the 20th international conference on Software engineering*. ICSE '98. Washington, DC, USA: IEEE Computer Society, 1998, S. 312–321.

[Coe09] Coetzee, C. J. „The Modelling of Bulk Materials Handling using the Discrete Element Method“. In: *1st African Conference on Computational Mechanics - An International Conference, AfriComp*. Sun City, South Africa, 2009.

[Coh+95] Cohen, J. D.; Lin, M. C.; Manocha, D.; Ponamgi, M. „I-COLLIDE: An interactive and exact collision detection system for large-scale environments“. In: *Proceedings of the 1995 symposium on Interactive 3D graphics*. I3D '95. New York, NY, USA: ACM, 1995, S. 189–196.

[Col+07] Colditz, J.; Dragon, L.; Faul, R.; Meljnikov, D.; Schill, V.; Unselt, T.; Zeeb, E. „Use of Driving Simulators within Car Development“. In: *Driving Simulator Conference*. Iowa City, USA, 2007.

[Cro+98] Crolla, D.; Chen, D.; Whitehead, J.; Alstead, C. *Vehicle Handling Assessment Using a Combined Subjective-Objective Approach*. SAE Technical Paper 980226. 1998.

[Cru+92] Cruz-Neira, C.; Sandin, D. J.; DeFanti, T. A.; Kenyon, R. V.; Hart, J. C. „The CAVE: audio visual experience automatic virtual environment“. In: *Commun. ACM* 35.6 (1992), S. 64–72.

[CS02] Cleary, P. W.; Sawley, M. L. „DEM modelling of industrial granular flows: 3D case studies and the effect of particle shape on hopper discharge“. In: *Applied Mathematical Modelling* 26.2 (2002), S. 89–111.

[CS79] Cundall, P. A.; Strack, O. D. L. „A discrete numerical model for granular assemblies“. In: *Géotechnique* 29.1 (1979), S. 47–65.

[CSD93] Cruz-Neira, C.; Sandin, D. J.; DeFanti, T. A. „Surround-screen projection-based virtual reality: the design and implementation of the CAVE“. In: *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '93. New York, NY, USA: ACM, 1993, S. 135–142.

[DCL07] Devadithya, T.; Chiu, K.; Lu, W. „C++ reflection for high performance problem solving environments“. In: *Proceedings of the 2007 spring simulation multiconference - Volume 2*. SpringSim '07. San Diego, CA, USA: Society for Computer Simulation International, 2007, S. 435–440.

[DKE10] De Troyer, O.; Kleinermann, F.; Ewais, A. „Enhancing virtual reality learning environments with adaptivity: lessons learned“. In: *Proceedings of the 6th international conference on HCI in work and learning, life and leisure: workgroup human-computer interaction and usability engineering*. USAB'10. Berlin, Heidelberg: Springer-Verlag, 2010, S. 244–265.

[Dro04] Dronka, S. „Die Simulation gekoppelter Mehrkörper- und Hydraulik-Modelle mit Erweiterung für Echtzeitsimulation“. Dissertation. Technische Universität Dresden, 2004.

[EAB09] Engel, S.; Alda, W.; Boryczko, K. „Real-time Computer Simulator of Hydraulic Excavator“. In: *7th Conference on Computer Methods and Systems*. Kraków, Poland, 2009.

[ECE00] Ensor, J. R.; Carraro, G. U.; Edmark, J. T. „Accommodating performance limitations in distributed virtual reality systems“. In: *Computer Communications* 23.3 (2000), S. 199–204.

[Edg02] Edgar, S. F. „Estimation of Worst-Case Execution Time Using Statistical Analysis“. PhD Thesis. University of York, 2002.

[EF98] Eich-Soellner, E.; Führer, C. *Numerical methods in multibody dynamics*. European Consortium for Mathematics in Industry. Literaturverz. S. [277] - 286. Stuttgart: Teubner, 1998, 290 S.

[EGD05] Espié, S.; Gauriat, P.; Duraz, M. „Driving Simulators Validation: The Issue of Transferability of Results Acquired on Simulator“. In: *Driving Simulator Conference*. Orlando, USA, 2005.

[EGL09] Esswein, W.; Greiffenberg, S.; Lehrmann, S. „Framework zur modellgestützten Simulation“. In: *Wissensportal www.baumaschine.de* Jahrgang 8.2 (2009).

[Ell89] Ellis, S. R. „Visions of Visualization Aids: Design Philosophy and Observations“. In: *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*. 1989.

[ELS09] Esswein, W.; Lehrmann, S.; Stark, J. „The Potential of Reference Modeling for Simulating Mobile Construction Machinery“. In: *Business Process Management Workshops*. 2009, S. 683–694.

[ELS10] Esswein, W.; Lehrmann, S.; Stark, J. „Kontinuierliche Referenzmodellverwaltung für die Maschinensimulation“. In: *Diskussionsbeiträge des 2. Workshops Dienstleistungsmodellierung (DLM 2010)*. Alpen-Adria-Universität Klagenfurt, Österreich, 2010.

[EM97] Elmqvist, H.; Mattsson, S. E. „An Introduction to the Physical Modeling Language Modelica“. In: *Proc. 9th European Simulation Symposium ESS97, SCS Int.* 1997, S. 110–114.

[End88] Endsley, M. R. „Design and Evaluation for Situation Awareness Enhancement“. In: *Human Factors and Ergonomics Society Annual Meeting Proceedings* (1988), S. 97–101.

[Eng+00] Engelen, R. van; Gallivan, K.; Gupta, G.; Cybenko, G. „XML-RPC Agents for Distributed Scientific Computing“. In: *In IMACS'2000 Conference*. Lausanne, Switzerland, 2000, S. 168–182.

[ES00] Ericsson, A.; Slättengren, J. „A model for predicting digging forces when working in gravel or other granulated material“. In: *15th European ADAMS Users' Conference*. Rome, Italy, 2000.

[Ess+10] Esswein, W.; Lehrmann, S.; Kunze, G.; Gubsch, I.; Penndorf, T. „Integrative Produktentwicklung mit virtuellen Prototypen“. In: *Werkstattstechnik online* Jahrgang 100.1 (2010).

[FE98] Fritzson, P.; Engelson, V. „Modelica - A Unified Object-Oriented Language for System Modelling and Simulation“. In: *Proceedings of the 12th European Conference on Object-Oriented Programming*. ECCOP '98. London, UK: Springer-Verlag, 1998, S. 67–90.

[Fea08] Featherstone, R. *Rigid Body Dynamics Algorithms*. New York: Springer, 2008.

[Fed04] Fedotov, S. „Untersuchung von Parallelmechanismen hinsichtlich deren Eignung als Geräteschnittstelle von Traktoren“. Dissertation. Technische Universität Dresden, 2004.

[FEP05] Filla, R.; Ericsson, A.; Palmberg, J.-O. „Dynamic Simulation of Construction Machinery: Towards an Operator Model“. In: *IFPE 2005 Technical Conference*. Las Vegas (NV), USA, 2005, S. 429–438.

[Fer09] Ferhadbegovic, B. „Entwicklung und Applikation eines instationären Reifenmodells zur Fahrdynamiksimulation von Ackerschleppern“. Dissertation. Universität Hohenheim, 2009.

[FGH05] Friedrichs, A.; Große-Kappenberg, S.; Happe, J. „Erprobung von Fahrerassistenzsystemen mit dem Interactive Driving Simulator“. In: *Erprobung und Simulation in der Fahrzeugentwicklung - Mess- und Versuchstechnik. Tagung der VDI-Gesellschaft Fahrzeug- und Verkehrstechnik*. Würzburg, 2005.

[Fil09] Filla, R. „A Methodology for Modeling the Influence of Construction Machinery Operators on Productivity and Fuel Consumption“. In: *Proceedings of the 2nd International Conference on Digital Human Modeling: Held as Part of HCI International 2009*. ICDHM '09. Berlin, Heidelberg: Springer-Verlag, 2009, S. 614–623.

[Fis09] Fischer, M. „Motion-Cueing-Algorithmen für eine realitätsnahe Bewegungssimulation“. Dissertation. Technische Universität Braunschweig, 2009.

[FO00] Featherstone, R.; Orin, D. „Robot Dynamics: Equations and Algorithms“. In: *IEEE Int. Conf. Robotics and Automation*. 2000, S. 826–834.

[For03] Fortmüller, T. „Der Einfluss visueller und vestibulärer Informationen auf die Qualität des Fahreindrucks in der Fahrsimulation“. In: Der Fahrer im 21. Jahrhundert : Tagung Braunschweig, 2. und 3. Juni 2003. Düsseldorf: VDI-Verlag, 2003, S. 313–330.

[Fre+09] Frenkel, J.; Schubert, C.; Kunze, G.; Jankov, K. „Using Modelica for Interactive Simulations of Technical Systems in a Virtual Reality Environment“. In: *Proceedings of the 7th International Modelica Conference*. Como, Italy, 2009, S. 531–540.

[Fre+11] Frenkel, J.; Schubert, C.; Voigt, S.; Kunze, G.; Knoll, C. „PyMbs: Ein generisches Software-Werkzeug für die Simulation von Mehrkörpersystemen“. In: *VDI Mechatronik-Tagung*. Dresden, 2011.

[Fre09] Frenkel, J. „Integration von OpenModelica in das Softwaresystem Sarturis“. Diplomarbeit. Technische Universität Dresden, 2009.

[Fre10] Frenkel, J. „Using Modelica for Interactive Simulations of Technical Systems in a Virtual Reality Environment“. In: *4th MODPROD Workshop on Model-Based Product Development*. Linköping, Sweden, 2010.

[Fri04] Fritzson, P. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley & Sons, 2004.

[Fri11] Fritzson, P. *Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica*. Wiley & Sons, 2011.

[FS96] Fisette, P.; Samin, J. C. „Symbolic generation of large multibody system dynamic equations using a new semi-explicit Newton/Euler recursive scheme“. In: *Archive of Applied Mechanics* 66 (3 1996), S. 187–199.

[Füh88] Führer, C. „Differential-algebraische-Gleichungssysteme in mechanischen Mehrkörpersystemen: Theorie, numer. Ansätze u. Anwendungen“. Dissertation. Technische Universität München, 1988.

[GDI04] Gerstmayr, J.; Dibold, M.; Irschik, H. „Dynamik geregelter flexibler Mehrkörpersysteme unter Berücksichtigung hydraulischer Aktorik“. In: *Elektrotechnik und Informationstechnik* 121.9 (2004), S. 307–312.

[GGM10] González, F.; González, M.; Mikkola, A. „Efficient coupling of multibody software with numerical computing environments and block diagram simulators“. In: *Multibody System Dynamics* 24 (3 2010), S. 237–253.

[Gha00] Ghassemi-Tabrizi, A. *Realzeit-Programmierung.* Literaturverz. S. [347] - 349. Berlin [u.a.]: Springer, 2000, IX, 357 S.

[Gip99] Gipser, M. „FTire, a New Fast Tire Model for Ride Comfort Simulations“. In: *International ADAMS User's Conference.* Berlin, 1999.

[GKK10] Grüning, T.; Kunze, G.; Katterfeld, A. „Simulating the working process of construction machines“. In: *Bulk Solids Europe 2010.* Glasgow, Scotland, 2010.

[GKL06] Geimer, M.; Krüger, T.; Linsel, P. „Co-Simulation, gekoppelte Simulation oder Simulationskopplung? Ein Versuch der Begriffsvereinheitlichung“. In: *O+P Zeitschrift für Fluidtechnik - Aktorik, Steuerelektronik und Sensorik* 50.11-12 (2006), S. 572–576.

[Gol10] Goldmann, P. „Gekoppelte Simulationen mit dem Softwaresystem SARTURIS“. Diplomarbeit. Technische Universität Dresden, 2010.

[Göt11] Götz, M. „Vergleich zwischen Hybridisierung und Elektrifizierung eines Traktors“. In: *3. Fachtagung Hybridantriebe für mobile Arbeitsmaschinen.* Karlsruhe, 2011.

[Gre05] Green, P. „How Driving Simulator Data Quality Can Be Improved“. In: *Driving Simulator Conference.* Orlando, USA, 2005.

[Gri02] Grimm, H.-P. „CORBA und XML-RPC-basierte Web Services als Plattformen für Anwendungsentwicklung in verteilten Systemen“. Studienarbeit. Humboldt-Universität Berlin, 2002.

[Gro08] Großer, M. „Entwicklung einer Visualisierungskomponente für das Programmsystem SARTURIS“. Diplomarbeit. Hochschule für Technik und Wirtschaft Dresden, 2008.

[GTK] GTK. *The GTK+ Project*. www.gtk.org.

[GW11] Großer, M.; Wacker, M. „High Performance Interactive Painting On The GPU“. In: *Eurographics*. 2011.

[Han91] Hanke, M. „Index Reduction and Regularization for Euler-Lagrange Equations of Constraint Mechanical Systems“. In: *Proceedings of the 2nd International Symposium on Implicit and Robust Systems*. Warsaw, Poland, 1991.

[Har02] Harnisch, C. „Dynamische Echtzeitsimulation der Geländefahrt mehrachsiger Radfahrzeuge“. Dissertation. Universität der Bundeswehr Hamburg, 2002.

[HB02] Heißing, B.; Brandl, H. *Subjektive Beurteilung des Fahrverhaltens*. Vogel Fachbuch. Vogel, 2002.

[HB03] Harnisch, C.; Breidenbach, C. „Geländefahrt von Radfahrzeugen im Fahrsimulator“. In: *Simulation und Simulatoren - Mobilität virtuell gestalten*. Düsseldorf: VDI-Verlag, 2003.

[HB06] Hoffmann, S.; Buld, S. „Darstellung und Evaluation eines Trainings zum Fahren in der Fahrsimulation“. In: *Integrierte Sicherheit und Fahrerassistenzsysteme*. VDI-Berichte 1960. Düsseldorf, 2006, S. 113–132.

[HBB05] Hummel, J.; Baack, E.; Bernard, J. „Operator in the Loop Simulation for Farm and Construction Vehicles“. In: *Driving Simulator Conference*. Orlando, USA, 2005.

[HDF] HDF5. *The HDF Group*. www.hdfgroup.org.

[Hen06] Henning, M. „The Rise and Fall of CORBA“. In: *Queue* 4.5 (2006), S. 28–34.

[HHS97] Hardtke, H.-J.; Heimann, B.; Sollmann, H. *Lehr- und Übungsbuch Technische Mechanik, Band 2: Kinematik/Kinetik - Systemdynamik - Mechatronik*. München, Wien: Fachbuchverlag Leipzig im Carl Hanser Verlag, 1997.

[Hin+05] Hindmarsh, A. C.; Brown, P. N.; Grant, K. E.; Lee, S. L.; Serban, R.; Shumaker, D. E.; Woodward, C. S. „SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers“. In: *ACM Trans. Math. Softw.* 31.3 (2005), S. 363–396.

[HKN08] Hildebrand, R.; Keskinen, E.; Navarrete, J. A. R. „Vehicle vibrating on a soft compacting soil half-space: Ground vibrations, terrain damage, and vehicle vibrations“. In: *Journal of Terramechanics* 45.4 (2008), S. 121–136.

[Hor04] Horauer, M. „Clock Synchronization in Distributed Systems“. Dissertation. Technische Universität Wien, 2004.

[Hos+12] Hoske, P.; Kunze, G.; Bürkle, K.; Schmauder, M.; Brütting, M.; Böser, C. „Interaktiver Simulator für mobile Arbeitsmaschinen - Virtuelle Prototypen im Einsatzkontext erleben“. In: *Tagungsband zur Konferenz „Entwerfen - Entwickeln - Erleben“, Methoden und Werkzeuge in Produktentwicklung und Design*. Dresden, 2012.

[HRW03] Hirschberg, W.; Rill, G.; Weinfurther, H. „User Appropriate Tyre Modeling for Vehicle Dynamics in Standard and Limit Situations“. In: *Vehicle Dynamics* 38.2 (2003), S. 103–125.

[HRW07] Hirschberg, W.; Rill, G.; Weinfurter, H. „Tyre Model TMeasy“. In: *Vehicle System Dynamics* 45.S1 (2007), S. 101–119.

[HW10] Hairer, E.; Wanner, G. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Bd. 2. Literaturverz. S. [577] - 604. Berlin [u.a.]: Springer, 2010, XV, 614 S.

[Jan10] Janschek, K. *Systementwurf mechatronischer Systeme : Methoden - Modelle - Konzepte*. Literaturangaben. Berlin [u.a.]: Springer, 2010, XVII, 842 S.

[JMS96] Jürgensohn, T.; Müller, W.; Scheffer, T. *Verbesserte Methoden zur Objektivierung von subjektiven Bewertungen des Fahrverhaltens*. Forschungsbericht. Techische Universität Berlin, Zentrum Mensch-Maschine-Systeme, 1996.

[KGJ09] Kunze, G.; Göhring, H.; Jacob, K. *Baumaschinen : Erdbau- und Tagebaumaschinen ; mit 147 Tabellen*. 1. Aufl [Nachdr.] Fördertechnik und Baumaschinen. Wiesbaden: Vieweg, 2009, XI, 407 S.

[Kim05] Kim, G. J. *Designing virtual reality systems : the structured approach.* London: Springer, 2005, IX, 233 S.

[KKG11] Katterfeld, A.; Kunze, G.; Grüning, T. „Coupled discrete element and multibody simulation for analysis of the operation of construction machines“. In: *Continuum and distinct element numerical modeling in geomechanics* (2011).

[KM08] Karimi, D.; Mann, D. D. „Role of Visual Cues from the Environment in Driving an Agricultural Vehicle“. In: *The Ergonomics Open Journal* 1.1 (2008), S. 54–61.

[Kop11] Kopetz, H. *Real-time systems : design principles for distributed embedded applications.* 2. ed. Real-time systems series. Literaturverz. S. 359 - 368. New York, NY [u.a.]: Springer, 2011, XVIII, 376 S.

[Kor09] Korkealaakso, P. „Real-Time Simulation of Mobile and Industrial Machines Using the Multibody Approach“. Dissertation. Lappeenranta University of Technology, 2009.

[KP03] Kemeny, A.; Panerai, F. „Evaluating perception in driving simulation experiments“. In: *Trends in Cognitive Sciences* 7.1 (2003), S. 31–37.

[Krz] Krzyzanowski, P. *Lectures on distributed systems - Clock Synchronization.* Technical Report CS 417: Distributed Systems. Rutgers University.

[KSG06] Kriegel, T.; Spann, O.; Gies, S. „Von der objektiven Größe zur subjektiven Bewertung der Fahrdynamik“. In: *5. Tag des Fahrwerks.* Institut für Kraftfahrwesen, Aachen, 2006.

[Kun+10] Kunze, G.; Schubert, C.; Esswein, W.; Lehrmann, S. „Software Architecture for Interactive Simulation of Mobile Machinery - Efficient Design and Simulation of Virtual Prototypes“. In: *Proceedings of the 1st Commercial Vehicle Technology Symposium (CVT 2010).* Aachen: Shaker, 2010, S. 417–426.

[Kun03] Kunze, G. „Virtuelle Realität - Simulationsmethoden in Forschung und Entwicklung“. In: *Schriftenreihe der Forschungsvereinigung Bau- und Baustoffmaschinen e.V. (FVB).* 23. 2003, S. 63–78.

[Lam78] Lamport, L. „Time, clocks, and the ordering of events in a distributed system“. In: *Commun. ACM* 21.7 (1978), S. 558–565.

[Lan09] Langtangen, H. P. *Python scripting for computational science*. 3. ed., corr. 2. printing. Texts in computational science and engineering ; 3. Literaturverz. S. [739] - 740. Berlin [u.a.]: Springer, 2009, XXVI, 750 S.

[LPP96] Lee, N. S.; Park, J. H.; Park, K. S. „Reality and human performance in a virtual world“. In: *International Journal of Industrial Ergonomics* 18.2-3 (1996), S. 187–191.

[Mac03] Macadam, C. C. „Understanding and Modeling the Human Driver“. In: *Vehicle System Dynamics* 40.1-3 (2003), S. 101–134.

[MB11] Mikelsons, L.; Brandt, T. „Generation of continuously adjustable vehicle models using symbolic reduction methods“. In: *Multibody System Dynamics* 26 (2 2011), S. 153–173.

[Mic85] Michon, J. „A Critical View of Driver Behavior Models - What Do We Know, What Should We Do?“ In: *Human behavior and traffic safety* (1985).

[MJD96] Malenfant, J.; Jacques, M.; Demers, F. N. „A tutorial on behavioral reflection and its implementation“. In: *Proceedings of the Reflection'96 Conference*. San Francisco, 1996, S. 1–20.

[MJR03] Mulder, J. D.; Jansen, J.; Rhijn, A. van. „An affordable optical head tracking system for desktop VR/AR systems“. In: *Proceedings of the workshop on Virtual environments 2003*. EGVE '03. New York, NY, USA: ACM, 2003, S. 215–223.

[Moh11] Mohr, M. „Elektrifizierung und Hybridisierung von Antriebssträngen für Baumaschinen“. In: *3. Fachtagung Hybridantriebe für mobile Arbeitsmaschinen*. Karlsruhe, 2011.

[Mor07] Morency, K. W. „Automatic Generation of Real-Time Simulation Code for Vehicle Dynamics using Linear Graph Theory and Symbolic Computing“. Master Thesis. University of Waterloo, Canada, 2007.

[Mot03] Mothes, M. „Modellbildung von Bodenstoffen in virtueller Umgebung“. In: *Wissensportal www.baumaschine.de* Jahrgang 2.1 (2003).

[NA00] Nierstrasz, O.; Achermann, F. „Separation of Concerns through Unification of Concepts“. In: *In ECOOP 2000 Workshop on Aspects & Dimensions of Concerns*. 2000.

[Neg07] Negele, H. J. „Anwendungsgerechte Konzipierung von Fahrsimulatoren für die Fahrzeugentwicklung“. Dissertation. Technische Universität München, 2007.

[Net] Netlib. *The Netlib*. www.netlib.org.

[NS01] Nehmer, J.; Sturm, P. *Systemsoftware : Grundlagen moderner Betriebssysteme*. 2., aktualisierte Aufl. dpunkt-Lehrbuch. Literaturverz. S. 345 - 351. Heidelberg: dpunkt Verl., 2001, XI, 362 S.

[OC04] Odhams, A.; Cole, D. „Models of driver speed choice in curves“. In: *Proceedings of the 7th International Symposium on Advanced Vehicle Control (AVEC 04)*. Arnheim, the Netherlands, 2004, S. 439–444.

[OCS03] Oliveira, M.; Crowcroft, J.; Slater, M. „An innovative design approach to build virtual environment systems“. In: *Proceedings of the workshop on Virtual environments 2003*. EGVE '03. New York, NY, USA: ACM, 2003, S. 143–151.

[OSG] OSG. *OpenSceneGraph*. www.openscenegraph.org.

[Pac07] Pacejka, H. B. *Tyre and vehicle dynamics*. 2.ed., reprint. Literaturverz. S. [595] - 605. Amsterdam [u.a.]: Elsevier/Butterworth-Heinemann, 2007, XIII, 642 S.

[PB93] Pacejka, H. B.; Bakker, E. „The magic formula tyre model, tyre models for vehicle dynamic analysis“. In: *Proceedings of the First International Colloquium on Tyre models for vehicle dynamic analysis*. Swets & Zeitlinger, 1993, S. 1–18.

[Pen+12] Penndorf, T.; Karch, G.; Gubsch, I.; Kunze, G. „Referenzmodell für die variantenorientierte Produktsimulation“. In: *Proceedings of the 2nd Commercial Vehicle Technology Symposium (CVT 2012)*. Aachen: Shaker, 2012, S. 18–26.

[Pen06] Penndorf, T. „Universelles Framework zur Abbildung von Maschinenmodellen in virtuellen Umgebungen“. In: *Schriftenreihe der Forschungsvereinigung Bau- und Baustoffmaschinen e.V. (FVB)*. 34. 2006, S. 183–192.

[PF10] Penndorf, T.; Frenkel, J. „A Study of OpenModelica in Realtime Simulation for Virtual Reality Environments“. In: *OpenModelica Annual Workshop*. Linköping, Sweden, 2010.

[Pil10] Pilgrim, M. *Python 3 - Intensivkurs : Projekte erfolgreich realisieren.* Xpert.press. Berlin [u.a.]: Springer, 2010, XVIII, 351 S.

[PK06] Penndorf, T.; Kunze, G. „Codegenerator für die Echtzeitsimulation von Mehrkörpersystemen“. In: *19. Symposium Simulationstechnik.* Hannover, 2006.

[PKG07] Penndorf, T.; Kunze, G.; Gubsch, I. „"Durchgespielt" - Interaktive Simulation von Baumaschinen“. In: *IX-MAGAZIN FÜR PROFESSIONELLE INFORMATIONSTECHNIK* 08 (2007).

[PKH01] Preuße, C.; Keller, H.; Hunt, K. J. „Fahrzeugführung durch ein Fahrermodell“. In: *Automatisierungstechnik at* 49.12 (2001).

[PMT03] Paillot, D.; Merienne, F.; Thivent, S. „CAD/CAE visualization in virtual environment for automotive industry“. In: *Proceedings of the workshop on Virtual environments 2003.* EGVE '03. New York, NY, USA: ACM, 2003, S. 315–316.

[PPG11] Patel, B. P.; Prajapati, J. M.; Gadhvi, B. J. „An Excavation Force Calculations and Applications: An Analytical Approach“. In: *International Journal of Engineering Science and Technology (IJEST)* 3.5 (2011).

[PR03] Pankiewicz, E.; Rulka, W. „From Off-Line to Real Time Simulations by Model Reduction and Modular Vehicle Modelling“. In: *ASME Conference Proceedings.* 2003.

[Pre97] Pree, W. *Komponentenbasierte Softwareentwicklung mit Frameworks.* 1. Aufl. dpunkt-Lehrbuch. Literaturverz. S. 91 - 94. Heidelberg: dpunkt-Verl., 1997, XII, 130 S.

[PYT] PYTHON. *Python Programming Language.* www.python.org.

[Ras86] Rasmussen, J. *Information processing and human-machine interaction: an approach to cognitive engineering.* North-Holland series in system science and engineering. North-Holland, 1986.

[Ras95] Rasmussen, J. „The concept of Human Error and the Design of reliable Human-Machine-Systems - Verläßlichkeit von Mensch-Maschine-Systemen“. In: *1. Berliner Werkstatt Mensch-Maschine-Systeme.* Berlin, 1995.

[RC07] Rill, G.; Chucholowski, C. „Real Time Simulation of Large Vehicle Systems“. In: *MULTIBODY DYNAMICS 2007, ECCOMAS Thematic Conference*. Milano, Italy, 2007.

[Ree64] Reece, A. R. „The fundamental equation of earth-moving mechanics“. In: *ARCHIVE Proceedings of the Institution of Mechanical Engineers Conference Proceedings* 179.1964 (1964), S. 16–22.

[RG11] Rüdenauer, A.; Geimer, M. „Physikalische Modellierung 2.0: Unternehmensübergreifende Co-Simulation mit Hilfe der GUSMA-Plattform“. In: *Kongress für Simulation im Produktentstehungsprozess*. Veitshöchheim, 2011.

[Rie03] Riecke, B. E. „How far can we get with just visual information? Path integration and spatial updating studies in Virtual Reality“. Dissertation. Eberhard Karls Universität Tübingen, 2003.

[Ril05] Rill, G. „TMeasy - Ein einfach handhabbares Reifenmodell“. In: *Simulation in der Fahrzeugdynamik. Tagung Nr. H030-02-223-5*. Essen, 2005.

[RP05] Rulka, W.; Pankiewicz, E. „MBS Approach to Generate Equations of Motions for HiL-Simulations in Vehicle Dynamics“. In: *Multibody System Dynamics* 14 (3 2005), S. 367–386.

[RS06] Raffin, B.; Soares, L. „PC Clusters for Virtual Reality“. In: *Proceedings of the IEEE conference on Virtual Reality*. VR '06. Washington, DC, USA: IEEE Computer Society, 2006, S. 215–222.

[RS10] Rill, G.; Schaeffer, T. *Grundlagen und Methodik der Mehrkörpersimulation : mit Anwendungsbeispielen*. 1. Aufl. Studium. Wiesbaden: Vieweg + Teubner, 2010, VIII, 215 S.

[Rud05] Rudik, R. „Untersuchungen zur Entwicklung der Steuerung eines neuartigen Traktorhubwerkes auf der Basis der Hexapod-Mechanismen“. Dissertation. Technische Universität Dresden, 2005.

[Ruf97] Ruff, K. „Fahrzeugbewegung im Gelände mit dem Simulationssystem ORIS“. Dissertation. Universität der Bundeswehr Hamburg, 1997.

[Sam97] Sametinger, J. *Software engineering with reusable components*. Berlin [u.a.]: Springer, 1997, S. I–XVI, 1–272.

[SB05] Stoehr, J.; Bulirsch, R. *Numerische Mathematik 2*. Berlin, Heidelberg, New York: Springer, 2005.

[SBK10] Schubert, C.; Beitelschmidt, M.; Kunze, G. „Handling kinematic loops of mobile machinery in real-time applications“. In: *PAMM* 10.1 (2010), S. 59–60.

[Sch+03] Schwarz, C.; Hench, S.; Gilmore, B.; Romig, B.; Watson, G.; Dolan, J.; Allen, S.; Cable, S. „Development of an Off-Road Agricultural Virtual Proving Ground“. In: *Driving Simulator Conference*. Dearborn, USA, 2003.

[Sch+05] Schmitt, J.; Schorn, M.; Stählin, U.; Isermann, R. „Entwicklungsumgebung mit echtzeitfähigen Gesamtfahrzeugmodellen für sicherheitskritische Fahrerassistenzsysteme (Development Environment with Real-time Models for Safety Assistance Systems)“. In: *Automatisierungstechnik* 53.1 (2005), S. 28–35.

[Sch05] Schwarz, C. „Two Migration Strategies for Motion System Limits“. In: *Driving Simulator Conference*. Orlando, USA, 2005.

[Sch06] Schmalzl, J. „Simulation des dynamischen Verhaltens von Flurförderzeugen in der Lagertechnik“. Dissertation. Technische Universität München, 2006.

[Sch10] Schatten, A. *Best Practice Software-Engineering : Eine praxiserprobte Zusammenstellung von komponentenorientierten Konzepten, Methoden und Werkzeugen*. Literaturverz. S. 429 - 432. Heidelberg: Spektrum Akad. Verl., 2010, VII, 440 S.

[Sch88] Schmoll, K.-P. *Modularer Aufbau von Mehrkörpersystemen unter Verwendung der Relativkinematik*. Fortschritt-Berichte VDI : Reihe 18, Mechanik/Bruchmechanik ; 57. Düsseldorf: VDI-Verl., 1988, 167 S.

[Sch94] Scheffler, M. *Grundlagen der Fördertechnik : Elemente und Triebwerke*. Fördertechnik und Baumaschinen. Braunschweig [u.a.]: Vieweg, 1994, VIII, 340 S.

[Sch99] Schneider, M. „Modellbildung, Simulation und nichtlineare Regelung elastischer, hydraulisch angetriebener Großmanipulatoren“. Dissertation. Gerhard-Mercator-Universität - Gesamthochschule, Duisburg, 1999.

[See04] Seewald, S. „Grafik-APIs zur Softwareentwicklung für die computergestützte Sehschulung“. Diplomarbeit. Technische Universität Dresden, 2004.

[SFR91] Simeon, B.; Führer, C.; Rentrop, P. „Differential-algebraic equations in vehicle system dynamics“. In: *Surveys on Mathematics for Industry* (1991), S. 1–37.

[SGM02] Szyperski, C.; Gruntz, D. W.; Murer, S. *Component Software : Beyond Object-Oriented Programming*. 2. ed. The Addison-Wesley Component Software Series. Literaturverz. S. 519 - 542. New York: ACM Press, 2002, XXXII, 589 S.

[SGP03] Schwarz, C.; Gates, T.; Papelis, Y. „Motion Characteristics of the National Advanced Driving Simulator“. In: *Driving Simulator Conference*. Dearborn, USA, 2003.

[Sha+08] Sharples, S.; Cobb, S.; Moody, A.; Wilson, J. R. „Virtual reality induced symptoms and effects (VRISE): Comparison of head mounted display (HMD), desktop and projection display systems“. In: *Displays* 29.2 (2008), S. 58–69.

[Slo08] Slob, J. *State-of-the-Art Driving Simulators, a Literature Survey*. Technical Report. Eindhoven University of Technology, 2008.

[SMK98] Stanney, K. M.; Mourant, R. R.; Kennedy, R. S. „Human factors issues in virtual environments: A review of the Literature“. In: *Presence Teleoperators Virtual Environments* 7.4 (1998), S. 327–351.

[SN99] Schneider, J.-G.; Nierstrasz, O. „Components, Scripts and Glue“. In: *Software Architectures – Advances and Applications*. Springer, 1999, S. 13–25.

[SNK11] Schubert, C.; Neidhold, T.; Kunze, G. „Experiences with the new FMI Standard Selected Applications at Dresden University“. In: *Proceedings of the 8th International Modelica Conference*. Dresden, Germany, 2011, S. 344–352.

[SPB06] Sprague, D. W.; Po, B. A.; Booth, K. S. „The importance of accurate VR head registration on skilled motor performance“. In: *Proceedings of Graphics Interface 2006*. GI '06. Toronto, Canada: Canadian Information Processing Society, 2006, S. 131–137.

[SS07] Shiiba, T.; Suda, Y. „Evaluation of driver's behavior with multibody-based driving simulator“. In: *Multibody System Dynamics* 17 (2 2007), S. 195–208.

[Sta95] Stanney, K. „Realizing the full potential of virtual reality: human factors issues that could stand in the way“. In: *Proceedings of the Virtual Reality Annual International Symposium (VRAIS'95)*. VRAIS '95. Washington, DC, USA: IEEE Computer Society, 1995, S. 28–34.

[Sto+07] Stoll, S.; Kliffken, M.; Behm, M.; Wang, X. „Regelungskonzepte für hydrostatische Antriebe in mobilen Arbeitsmaschinen“. In: *Automatisierungstechnik* 55 (2007).

[Str05] Sträter, O. *Cognition And Safety: An Integrated Approach To Systems Design And Assessment*. Ashgate, 2005.

[Str09] Sträter, O. „Cognitive Parameter for the Relationship of Situation Awareness and Behaviour“. In: *Beiträge aus der Zeitschrift für Arbeitswissenschaft* (2009).

[Str89] Stroustrup, B. „Multiple Inheritance for C++“. In: *Computing Systems* 2.4 (1989), S. 367–395.

[Sun] Sundials. *SUNDIALS*. www.sundials.org.

[SVS11] Sun, Y.; Vogel, S.; Steuer, H. „Combining Advantages of Specialized Simulation Tools and Modelica Models using Functional Mock-up Interface (FMI)“. In: *Proceedings of the 8th International Modelica Conference*. Dresden, Germany, 2011, S. 491–494.

[SW06] Schnell, G.; Wiedemann, B. *Bussysteme in der Automatisierungs- und Prozesstechnik : Grundlagen, Systeme und Trends der industriellen Kommunikation*. 6., überarb. und aktualisierte Aufl. Vieweg Praxiswissen. Literaturangaben. Wiesbaden: Vieweg, 2006, XII, 414 S.

[SW99] Schwertassek, R.; Wallrapp, O. *Dynamik flexibler Mehrkörpersysteme.: Methoden der Mechanik zum rechnergestützten Entwurf und zur Analyse mechatronischer Systeme*. Grundlagen und Fortschritte der Ingenieurwissenschaften. Vieweg Friedr. + Sohn Verlag, 1999.

[Tan03] Tanenbaum, A. S. *Computernetzwerke*. 4., überarb. Aufl. Informatik. Literaturverz. S. 914 - 931. München: Pearson Studium, 2003, 960 S.

[TF01] Tomaske, W.; Fortmüller, T. „Der Einfluss von Wahrnehmungsschwellwerten auf die Auslegung von Bewegungssystemen in der Fahrsimulation“. In: *Human Factors bei der Entwicklung von Fahrzeugen*. Deutsche Gesellschaft für Luft- und Raumfahrt e. V. 2001.

[TF11] Tomulik, P.; Fraczek, J. „Simulation of multibody systems with the use of coupling techniques: a case study“. In: *Multibody System Dynamics* 25 (2 2011), S. 145–165.

[TG07] Thiebes, P.; Geimer, M. „Hybridantriebe für Mobile Arbeitsmaschinen - Entwicklung unter Berücksichtigung der Betriebsstrategie“. In: *O+P Zeitschrift für Fluidtechnik - Aktorik, Steuerelektronik und Sensorik* 51.11–12 (2007), S. 630–635.

[The07] Theimert, C. „Untersuchung der Fahrer-Fahrerhaus-Kommunikation mit dem Werkzeug Lkw-Fahrsimulator“. Dissertation. Technische Universität München, 2007.

[Tob04] Tobolář, J. „Reduktion von Fahrzeugmodellen zur Echtzeitsimulation“. Dissertation. Tschechische Technische Universität Prag, 2004.

[TS08] Tanenbaum, A. S.; Steen, M. van. *Verteilte Systeme : Prinzipien und Paradigmen*. 2., aktualisierte Aufl. Literaturangaben. München: Pearson Studium, 2008, 761 S.

[Tuu09] Tuuk, T. van der. „Elektrische Hybridantriebe für mobile Arbeitsmaschinen“. In: *2. Fachtagung Hybridantriebe für mobile Arbeitsmaschinen*. Karlsruhe, 2009.

[Uch11] Uchida, T. K. „Real-time Dynamic Simulation of Constrained Multibody Systems using Symbolic Computation“. PhD Thesis. University of Waterloo, Canada, 2011.

[Vah09] Vahlensieck, B. „Elektrische Antriebe für mobile Arbeitsmaschinen - Ein methodischer Ansatz zum Übertragen existierender Lösungen“. In: *2. Fachtagung Hybridantriebe für mobile Arbeitsmaschinen*. Karlsruhe, 2009.

[Vin97] Vinoski, S. „CORBA: Integrating diverse applications within distributed heterogeneous environments“. In: *IEEE Communications Magazine* 35.2 (1997), S. 46–55.

[VLC] VLC. *VideoLAN - Official page for VLC media player, the Open Source video framework!* www.videolan.org.

[Voi09] Voigt, S. „Perspektiven und Methoden für die Integration des Bedienerverhaltens bei der Simulation mobiler Arbeitsmaschinen“. In: *Wissensportal www.baumaschine.de* Jahrgang 8.2 (2009).

[VRJ] VRJ. *vr juggler*. www.vrjuggler.org.

[Wag01] Wagner, F. „Konzepte und Methoden zu allgemeinen, physikalisch basierten Animationssystemen auf der Grundlage der Lagrange-Faktoren-Methode“. Dissertation. Universität Rostock, 2001.

[Wal+07] Wallis, G.; Chatziastros, A.; Tresilian, J.; Tomasevic, N. „The role of visual and nonvisual feedback in a vehicle steering task“. In: *Journal of Experimental Psychology: Human Perception and Performance* 33.5 (2007), S. 1127–1144.

[Wan+07] Wang, Y.; Zhang, W.; Wu, S.; Guo, Y. „Simulators for Driving Safety Study - A Literature Review“. In: *Virtual Reality*. Hrsg. von Shumaker, R. Bd. 4563. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, S. 584–593.

[War03] Warkus, M. *GNOME 2.0 : Das Entwickler-Handbuch*. Galileo Computing. Literaturverz. S. 671 - 674. Bonn: Galileo Press, 2003, 715 S.

[Wat08] Watter, H. *Hydraulik und Pneumatik : Grundlagen und Übungen - Anwendungen und Simulation ; mit 23 Tabellen*. 2., überarb. Aufl. Studium. Literaturverz. S. 234 - 236. Wiesbaden: Vieweg + Teubner, 2008, XIII, 241 S.

[WH82] Wehage, R. A.; Haug, E. J. „Generalized coordinate partitioning for dimension reduction in analysis of constrained dynamic systems“. In: *ASME Journal of Mechanical Design* 104.1 (1982), S. 247–255.

[Wil+08] Wilhelm, R.; Engblom, J.; Ermedahl, A.; Holsti, N.; Thesing, S.; Whalley, D.; Bernat, G.; Ferdinand, C.; Heckmann, R.; Mitra, T.; Mueller, F.; Puaut, I.; Puschner, P.; Staschulat, J.; Stenström, P. „The worst-case execution-time problem - overview of methods and survey of tools“. In: *ACM Trans. Embed. Comput. Syst.* 7.3 (2008), 36:1–36:53.

[Wil03] Wilkens, R. „Bewegungssysteme in der Simulation“. In: *Simulation und Simulatoren - Mobilität virtuell gestalten*. Düsseldorf: VDI-Verlag, 2003.

[Win06] Winger, A. „Die thermohydraulische Freikolbenmaschine als Antriebslösung für mobile Arbeitsmaschinen“. In: *Schriftenreihe der Forschungsvereinigung Bau- und Baustoffmaschinen e.V. (FVB)*. 34. 2006, S. 45–58.

[Win09] Winger, A. „Gegengewichtsstapler mit Freikolbenmaschine“. In: *Schriftenreihe der Forschungsvereinigung Bau- und Baustoffmaschinen e.V. (FVB)*. 37. 2009, S. 9–18.

[WN03] Wasfy, T.; Noor, A. „Computational strategies for flexible multibody systems“. In: *Applied Mechanics Reviews* 56.6 (2003).

[WQ05] Wang, A.; Qian, K. *Component-oriented programming*. Wiley Interscience, 2005.

[WSG11] Will, D.; Ströhl, H.; Gebhardt, N. *Hydraulik : Grundlagen, Komponenten, Schaltungen*. 5., neu bearb. und erw. Aufl. Literaturverz. S. [491] - 504. Berlin [u.a.]: Springer, 2011, XI, 510 S.

[Xia08] Xia, K. „A framework for earthmoving blade/soil model development“. In: *Journal of Terramechanics* 45.5 (2008), S. 147–165.

[YFR06] Yeh, T. Y.; Faloutsos, P.; Reinman, G. „Enabling real-time physics simulation in future interactive entertainment“. In: *Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*. Sandbox '06. New York, NY, USA: ACM, 2006, S. 71–81.

[Zie+02] Zieschang, H.; Müller-Gethmann, H.; Reinert, D.; Schmauder, M.; Schmidt, W. „Anforderungen an Multifunktions-Stellteile“. In: *Die BG* 3 (2002), S. 114–117.

[ZK95] Zhang, J.; Kushwaha, R. L. „A modified model to predict soil cutting resistance“. In: *Soil & Tillage Research* 34 (1995), S. 157–168.

[Zor11] Zorn, S. „Experimentelle Untersuchung des Übertragungsverhaltens eines interaktiven Fahrsimulators“. Studienarbeit. Technische Universität Dresden, 2011.

[ZS07] Zimmermann, W.; Schmidgall, R. *Bussysteme in der Fahrzeugtechnik : Protokolle und Standards ; mit 99 Tabellen*. 2., aktualisierte und erw. Aufl. ATZ/MTZ-Fachbuch. Literaturverz. S. 347. Wiesbaden: Vieweg, 2007, XIII, 356 S.

[Züh02] Zühlke, D. „Bedienung komplexer Maschinen - heute, morgen und übermorgen“. In: *Bedienen und Verstehen: 4. Berliner Werkstatt Mensch-Maschine-Systeme*. Hrsg. von Marzi, R.; Karavezyris, V.; Erbe, H.-H.; Timpe, K.-P. Bd. 22. Fortschritt-Berichte VDI 8. Düsseldorf, 2002, S. 42–54.