Missionsbezogener modellgestützter Entwurf mobiler automatischer Systeme

am Beispiel eines autonomen Unterwasserfahrzeugs

Dissertation
zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt der

Fakultät für Informatik und Automatisierung der
Technische Universität Ilmenau

von

Dipl.-Ing. Thomas Volker Liebezeit
geboren am 12.08.1976 in Weimar

Gutachter:

1. Univ.-Prof. Dr. (USA) Horst Salzwedel, Technische Universität Ilmenau
2. Univ.-Prof. Dr.-Ing. habil. Jürgen Wernstedt, Technische Universität Ilmenau
3. Prof. Dr.-Ing. habil. Georg Bretthauer, Forschungszentrum Karlsruhe

eingereicht am: 01. September 2004
wissenschaftliche Aussprache am: 16. Dezember 2005

urn:ubn:de:gbv:ilm1-2005000240

Danksagung

Maßgebend für den Erfolg meiner Arbeit war eine vielseitige fachliche und menschliche Unterstützung. Dafür habe ich zu danken.

Als erstes gilt mein Dank den Professoren Jürgen Wernstedt und Horst Salzwedel, die mir während der vergangenen drei Jahre Unterstützung und fachliche Kritik gewährten und durch ihre Hinweise maßgeblich zur Struktur der vorliegenden Arbeit beitrugen. Auch Professor Georg Bretthauer danke ich herzlich für die Erstattung des Gutachtens.

Einen besonderen Dank möchte ich meinem Betreuer, Dr.-Ing. Volker Zerbe, aussprechen. Während unserer Zusammenarbeit an der Technischen Universität Ilmenau stand er mir jederzeit als konstruktiver und kritischer Gesprächspartner zur Seite und gab zahlreiche wertvolle Anregungen.

Verbunden bin ich auch dem gesamten *DeepC*-Team an der Technischen Universität Ilmenau, namentlich Dr.-Ing. Thorsten Pfützenreuter, Mike Eichhorn, Dr.-Ing. Divas Kalmanzira und Dr.-Ing. Peter Otto, für die konstruktive Zusammenarbeit und den fachlichen Austausch.

Eine große Unterstützung war darüber hinaus die freundliche und kollegiale Aufnahme, die mir die Mitarbeiter des Institutes für System- und Steuerungstheorie der Technischen Universität Ilmenau zuteil werden ließen.

Nicht nur für die Durchsicht und Korrektur dieses Manuskriptes, sondern auch für ihre Geduld, ihr Interesse und ihre vielen kleinen Hilfeleistungen danke ich meiner Lebensgefährtin und meiner Familie.

Kurzfassung

Gegenstand dieser Arbeit ist die Anwendung des Mission Level Designs auf den Entwurf mobiler automatischer Systeme. Im Mittelpunkt steht dabei die Simulation auf Missionsebene. Sie verfolgt das Ziel, möglichst frühzeitig im Entwurfsprozess Aussagen bezüglich der Leistungsfähigkeit des Systems zu gewinnen und mögliche Probleme zu erkennen. Beim Mission Level Design handelt es sich um eine missionsbezogene modellgestützte Systementwurfsmethode für komplexe Systeme. Sie überführt die Systemspezifikation in ein ausführbares Gesamtsystemmodell auf Systemebene und simuliert dieses. Gegenüber anderen Entwurfsmethoden zeichnet sich das Mission Level Design durch die Einbeziehung der sogenannten Missionen aus. Bei ihnen handelt es sich um typische Einsatzszenarien des zu entwerfenden Systems. Durch die missionsbezogene Simulation wird sichergestellt, daß das System die angestrebten Eigenschaften erreicht.

Die vorliegende Arbeit identifiziert die für die Modellierung mobiler automatischer Systeme zentralen Aspekte. Darüber hinaus stellt sie fest, daß der Entwurf dieser Systeme eine Reihe von Erweiterungen des Modellierungsprozesses notwendig macht: Zum einen muss das Gesamtsystemmodell in die Teilmodelle Systemmodell und Umwelt untergliedert werden. Zum anderen erfordert die Einbeziehung der Missionen eine Erweiterung des Konzepts der Simulationsdurchführung um die neue Phase der Missionshandhabung und die Auswertung muß missionsspezifischen erfolgen.

Auf Basis ihrer theoretischer Überlegungen stellt die Arbeit im zweiten Teil ein Framework für den missionsbezogenen modellgestützten Entwurf mobiler automatischer Systeme vor. Sein Kern ist das kommerzielle Entwurfsprogramm MLDesigner, erweitert um die neuentwickelten Programme MLEditor und MLVisor. Dieses Framework gestattet erstmals eine einfache Durchführung von Simulationen auf Missionsebene.

Als erster konkreter Anwendungsfall des Frameworks dient das *DeepC*-Projekt, welches das Ziel verfolgt, ein neuartiges autonomes Unterwasserfahrzeug zu entwickeln. Für dieses wurden unter Nutzung des Frameworks Untersuchungen auf Missionsebene durchgeführt. Das eigens für diesen Zweck erstellte Gesamtsystemmodell des *DeepC*-Fahrzeugs besteht aus einer virtuellen Umwelt und einem Fahrzeugmodell, welches neben der Dynamik, der Sensorik und Aktorik auch die Softwarestruktur und ihre Abarbeitung auf einer virtuellen Hardware nachbildet. Desweiteren integriert das Modell den energetischen Aspekt, der als begrenzte Ressource für ein autonomes Unterwasserfahrzeug von besonderer Bedeutung ist. Anhand dieses praktischen Beispiels wird die Durchführung von Simulationen auf Missionsebene demonstriert.

Abstract

Matter of this work is the application of Mission Level Design to the design of mobile automatic systems. It focuses on the mission level simulation, which aims at getting early estimations of the available system performance and at the discovery of possible problems. Mission Level Design is a mission oriented, model based design method for complex systems. It transforms the system specification into an executable, so called overall system model and simulates it. Compared to other design methods, Mission Level Design features by the inclusion of the so called missions. They represent typical use cases of the system, that is to be developed, and are used as test conditions. The mission oriented simulation ensures, that the system will achieve its defined features.

This work identifies the central aspects for the modeling of mobile automatic systems. Furthermore it shows, that the design of such systems requires several modifications of the modeling process: On one hand, the overall system model has to be partitioned into the system model and the environment. On the other hand, the inclusion of missions requires the enlargement of the so far usual concept of simulation by a mission management and a mission-specific evaluation phase.

Based on theoretic considerations, this work introduces a newly developed framework for the mission oriented model based design of mobile automatic systems. Its basis is the commercial design tool MLDesigner, completed by the software tools MLEditor and MLVisor. The framework for the first time allows the easy realization of simulations on mission level.

The framework's first practical use case is the *DeepC*-project, which aims at the development of a novel autonomous underwater vehicle. Using the framework, investigations on mission level were carried out. The specially for this purpose developed overall system model of the *DeepC* vehicle consists of a virtual environment and the system model of the vehicle. The latter contains the dynamics, sensors and actors as well as the functional software structure running on a virtual hardware. Furthermore, the model includes the energetic aspect, which constitutes a main limitative resource for autonomous underwater vehicles and is therefore of great importance. By this concrete example, this work demonstrates the realization of simulations on mission level.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Ein System ist eine gegenüber seiner Umwelt abgegrenzte Anordnung von Komponenten, die miteinander in Beziehung stehen. Der Systementwurf, dessen Gegenstand die Entwicklung von Systemen ist, gewinnt angesichts der steigenden Komplexität technischer Systeme zunehmend an Bedeutung. Die Entwicklung dieser Systeme weist einige Besonderheiten auf: Sie bestehen aus einer Reihe ihrerseits komplexer Teilsysteme, von denen jedes zur Realisierung einer speziellen Teilaufgabe dient. Das Gesamtsystem ist dabei auf die reibungslose Zusammenarbeit seiner Teilsysteme angewiesen, denn lokale Fehler in einem Teilsystem verursachen unter Umständen erhebliche Störungen der Gesamtfunktionalität.

Die Komplexität der Teilsysteme stellt dabei eine neue Qualität der Herausforderung dar. Sie führt dazu, daß der Überblick über die Gesamtfunktionalität erschwert wird. Für die Entwicklung der Teilsysteme ist in der Regel der Einsatz entsprechend spezialisierter Entwickler notwendig, die nicht selten organisatorisch und räumlich getrennt voneinander arbeiten. In diesem Fall entstehen die Teilsysteme unabhängig und parallel zueinander, was nicht nur eine besondere koordinatorische Herausforderung für den Systementwurf darstellt, sondern auch eine erhöhte Gefahr des Auftretens von Entwicklungsfehlern mit sich bringt.

Die Grundlage der gemeinsamen Entwicklungsarbeit stellt die Spezifikation dar, die im Rahmen des Systementwurfs erarbeitet wird. Hierzu wird ein Konzept des Systems erstellt, indem die einzelnen Teilaufgaben und -systeme identifiziert werden und ihre komplexe Zusammenarbeit festgelegt wird. Dieses Konzept wird in der Spezifikation beschrieben und dient als Ausgangspunkt für die Entwicklung der Teilsysteme. Die Spezifikation stellt damit eine Systembeschreibung dar, die eine Umsetzung der Anforderungen definiert, die das System erfüllen soll.

Eine häufige Fehlerquelle im Systementwurf liegt in einer nicht korrekten oder nicht vollständigen Beschreibung der Anforderungen. Hiervon können sowohl die allgemeinen Anforderungen an das System, als auch diejenigen an die Teilsysteme betroffen sein. Werden die allgemeinen Anforderungen aufgrund einer inkorrekten Spezifikation nicht erfüllt, erhält der Auftraggeber ein System, das für ihn nicht einsetzbar ist. Wurden einzelne Teilsysteme falsch entwickelt, treten Probleme an den Schnittstellen zu anderen

Teilsystemen auf. Dies kann den Ausfall von Teilen der Funktionalität oder im Extremfall des gesamten Systems zur Folge haben.

Unter diesen Rahmenbedingungen kommt der Erstellung der Spezifikation eine zentrale Bedeutung zu. Sie wird vom Projektmanagement vorgenommen, das gleichzeitig die Gesamtkoordination des Projektes betreibt und gegenüber dem Projektinitiator die Verantwortung für den erfolgreichen Abschluß der Entwicklung trägt. Das übergeordnete Ziel ist dabei die Entwicklung des Systems innerhalb der vorgegebenen Rahmenbedingungen, inklusive der Einhaltung vereinbarter Limitierungen des Realisierungsrahmens sowie der Finanzen.

Für das tatsächliche Auftreten der genannten Probleme im Prozeß der Systementwicklung finden sich in der Literatur zahlreiche Hinweise. Prominente Beispiele sind der Absturz der Ariane 5-Rakete ([Sch00]) und die aktuellen Probleme bei der Entwicklung eines satellitengestützten Mautsystems ([Bor03]).

1.2 Zielstellung

Um die angesprochenen Probleme zu lösen, wurden diverse Entwurfsmethoden entwickelt, unter denen das Mission Level Design eine der jüngsten Entwicklungen darstellt. Bei diesem handelt es sich um eine missionsbezogene modellgestützte Entwurfsmethode zur Unterstützung der Entwicklung komplexer, technischer Systeme durch die Simulation auf abstrahiertem Niveau.

Die vorliegende Arbeit beschäftigt sich mit dem Einsatz des Mission Level Designs für die Entwicklung mobiler automatischer Systeme. Diese Systeme sind in der Regel durch einen technisch komplexen Aufbau gekennzeichnet und werden im Hinblick auf ihre Leistungsfähigkeit und Zuverlässigkeit mit höchsten Ansprüchen konfrontiert. Ein unentdeckter Fehler kann ein unter Umständen sehr aufwendiges Redesign erforderlich machen. Im Extremfall kann er zum Verlust des gesamten Systems führen, was insbesondere deshalb problematisch ist, weil es sich bei diesen Systemen häufig um Einzelanfertigungen handelt.

Im Rahmen dieser Arbeit soll eine fundierte Analyse und Bewertung des aktuellen Standes des missionsbezogenen modellgestützten Entwurfs erarbeitet und seine Anwendbarkeit für mobile automatische Systeme geprüft werden. Dabei sind die folgenden, zu Beginn der Arbeit offenen Fragen zu beantworten:

- **Wie kann das Mission Level Design für mobile automatische Systeme angewendet werden?**
- **Wie lassen sich die Simulationen im Rahmen des missionsbezogenen modellgestützten Entwurfs mobiler automatischer Systeme mittels ML-Designer durchführen?**

- **Wie gestaltet sich der praktische Einsatz des missionsbezogenen modellgestützten Entwurfs mobiler automatischer Systeme?**

Für die Beantwortung der letztgenannten Frage werden die im Rahmen des *DeepC*-Projektes durchgeführten, praktischen Arbeiten herangezogen. Ziel dieses Projekts ist der Entwurf eines neuartigen, autonomen Unterwasserfahrzeugs für große Tauchtiefen und eine lange Einsatzdauer. Bei dieser Art von Fahrzeugen handelt es sich um Unterwasserroboter, die gemäß ihrer vorgegebenen Aufgabe vollkommen selbständig agieren. Sie stellen eine eigene Kategorie der mobilen automatischen Systeme dar.

Das Mission Level Design für ein autonomes Unterwasserfahrzeug beinhaltet dabei die simulative Untersuchung des Fahrzeugs auf Missionsebene, wozu eine ganzheitliche Modellierung des Fahrzeugs und seiner Missionen notwendig ist. Diese Untersuchungen setzen eine Simulationsumgebung voraus, für deren Erstellung im Rahmen dieser Arbeit auf das Simulations- und Entwurfsprogramm MLDesigner zurückgegriffen wurde.

1.3 Gliederung

Die Arbeit ist in ihrer Struktur entsprechend der ihr zugrundeliegenden Fragestellungen dreistufig gegliedert (Kapitel: drei, vier und fünf). Im einzelnen unterteilt sie sich wie folgt:

Kapitel 2 – **Grundlagen**:
Ein einführendes Grundlagenkapitel vermittelt zunächst eine Übersicht über die Teilgebiete, die diese Arbeit umfaßt. Dazu zählen neben dem Systementwurf, also der Entwicklung von Systemen, auch die Simulation, die mobilen automatischen Systeme und eine Übersicht zur Simulation autonomer Unterwasserfahrzeuge.

Kapitel 3 – **Missionsbezogener modellgestützter Entwurf mobiler automatischer Systeme**:
Das dritte Kapitel wendet sich der Kernthematik dieser Arbeit zu, indem es sich mit dem missionsbezogenen modellgestützten Entwurf mobiler automatischer Systeme beschäftigt. Dabei wird eine Analyse des missionsbezogenen Entwurfs für diese Kategorie von Systemen vorgenommen. In diesem Zusammenhang werden die Anforderungen formuliert, die mobile automatische Systeme an diese Entwurfsmethode, inbesondere an die missionsbezogene Simulation, stellen.

Kapitel 4 – **Framework für den Entwurf auf Missionsebene**:
Auf der Grundlage der im vorausgehenden Kapitel erarbeiteten Anforderungen wird im vierten Kapitel ein Framework für die Durchführung der Simulationen im Rahmen des missionsbezogenen modellgestützten Entwurfs vorgestellt. Dieses Framework nutzt die Funktionalitäten des

Simulations- und Entwurfsprogramms MLDesigner, erweitert sie jedoch mit Hilfe zweier neu erstellter Programme (MLEditor und MLVisor) um die spezifischen Anforderungen, die sich aus dem Umgang mit Missionen ergeben.

Kapitel 5 – AUV DeepC:
Das fünfte Kapitel beschreibt die praktischen Untersuchungen, die im Rahmen des *DeepC*-Projekts vorgenommen wurden. Dabei werden Simulationen gemäß des missionsbezogenen modellgestützten Entwurfs mit dem Framework vorgenommen. Basis dieser Untersuchungen sind ein eigens für diesen Zweck entwickeltes Gesamtsystemmodell des *DeepC*-Fahrzeugs und die zugehörigen Missionen.

Kapitel 6 – Zusammenfassung:
Im sechsten und letzten Kapitel werden die Ergebnisse dieser Arbeit gezielt zusammengefaßt. Dabei erfolgt neben der Beantwortung der Fragen, die in dieser Einleitung aufgeworfen wurden, auch ein Ausblick auf potentielle Ansatzpunkte für die Nutzung der Ergebnisse und weiterführende Arbeiten.

2 Grundlagen

Das folgende Grundlagenkapitel gibt einen Überblick über die dieser Arbeit zugrunde liegenden Teilgebiete.

Dabei handelt es sich um den Systementwurf, die Simulation, die mobilen automatischen Systeme und die Simulation autonomer Unterwasserfahrzeuge, die jeweils in eigenen Abschnitten einführend behandelt werden. Desweiteren gibt ein Abschnitt eine kurze Einführung zum im Rahmen der Arbeit verwendeten Entwurfswerkzeug MLDesigner.

2.1 Systementwurf

2.1.1 Systembegriff

Ausgangspunkt aller Überlegungen zum Systementwurf ist der Systembegriff, dessen Gebrauch semantische Unterschiede aufweisen kann. Als Basis der weiteren Ausführungen soll deshalb folgende Begriffsklärung dienen:

> **System:** Ein System ist eine abgegrenzte Anordnung von Komponenten, die miteinander in Beziehung stehen.

Bei einem System handelt es sich um ein Objekt, das sich durch die Abgrenzung von seiner Umwelt bildet. Die Grenze zwischen den Systemkomponenten und der Umwelt ist die Systemgrenze. Über diese hinweg interagiert das System mittels Ein- und Ausgängen mit seiner Umwelt. Abbildung 2.1 illustriert diese Definition eines allgemeinen Systems. Auf

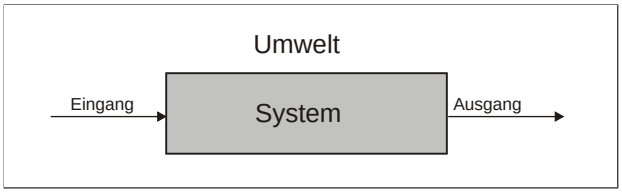

Abbildung 2.1: System

ihrer Basis lassen sich Systeme und ihr Verhalten allgemein beschreiben. Systeme besitzen bestimmte Eigenschaften (Zeitverhalten, Änderungsverhalten, Übertragungsverhalten, Reproduzierbarkeit [Ada01]) und können wiederum aus (Sub-)Systemen aufgebaut sein.

Im Entwurfsprozeß wird diese Definition genutzt, um die zu entwickelnden Systeme im Rahmen der Spezifikation detailliert zu beschreiben. Letztere beinhaltet eine Auflistung aller Komponenten und Subsysteme inklusive ihrer Kennwerte. Desweiteren umfaßt sie Informationen dazu, wie die Systemkomponenten zusammenarbeiten sollen.

Spezifikation: Die Spezifikation ist eine Systembeschreibung für den Entwurf.

2.1.2 Entwurfsansätze

Für den Ablauf der Entwicklung von Systemen existieren verschiedene modellhafte Ansätze, die alle das Ziel verfolgen, typische Probleme während des Entwurfs proaktiv zu vermeiden. Im Folgenden sollen beispielhaft fünf bedeutsame Entwurfsmodelle näher vorgestellt werden.

2.1.2.1 Wasserfallmodell

Das Wasserfallmodell wurde von Royce vorgestellt und ist beispielsweise in [Bue00, S. 17] beschrieben. Es wurde für den Entwurf von Softwaresystemen formuliert und ordnet, wie die folgende Auflistung zeigt, die einzelnen Entwicklungsschritte nacheinander an:

1. Erarbeitung der Systemspezifikation
2. Erstellung der Softwarespezifikation
3. Analyse der Anforderungen
4. Entwurf der Software
5. Implementierung der Software
6. Durchführung von Tests
7. Einsatz der Software

Die Ausführung erfolgt schrittweise nacheinander, wobei erst zum Schluß ein Test des realisierten Systems durchgeführt wird.

Dieses Modell zeichnet sich vor allem durch seine Einfachheit aus. Desweiteren beinhaltet es die Erkenntnis, daß sich die Entwicklung in einzelnen, in sich geschlossenen Schritten durchführen läßt. Der Test und die Integration werden jedoch bis zum Ende verschoben,

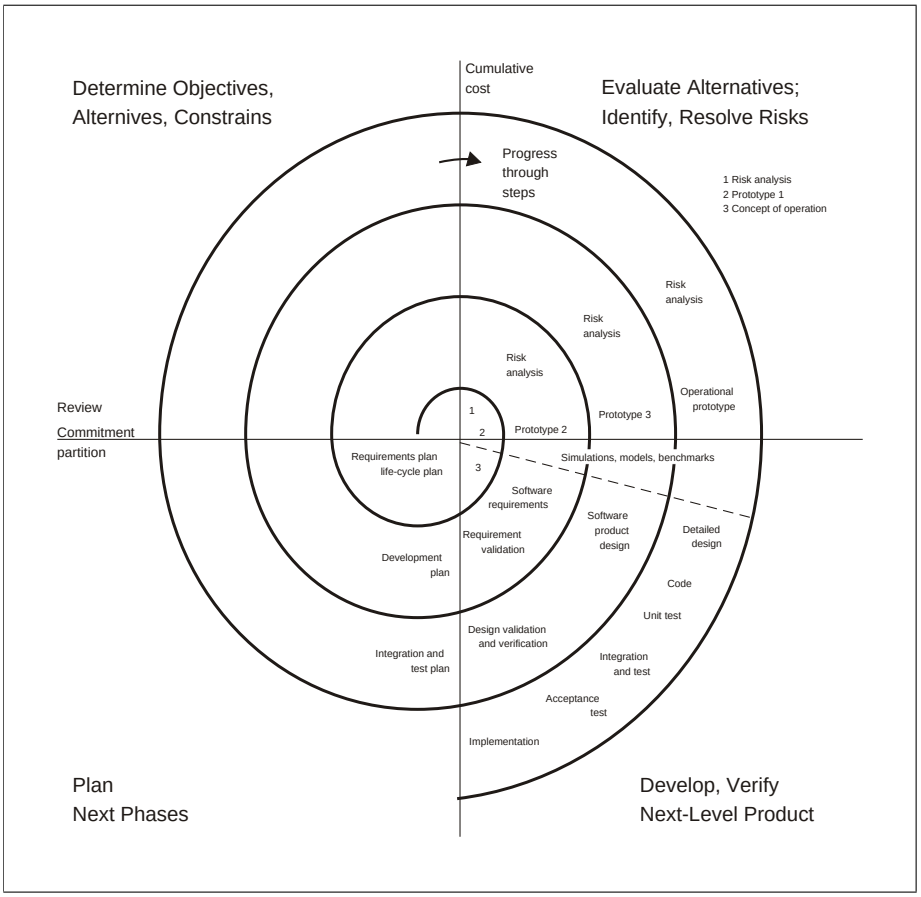

Abbildung 2.2: Spiralmodell (nach [Boe88, S. 64])

wodurch keine stufenweise Erprobung stattfindet. Das heißt, es kommt nicht bereits während der Entwicklung zu einem Test, ob das System die gestellten Anforderungen erfüllt. Damit existiert bis zum Ende kein Raum für Änderungen am Entwurf, was hinsichtlich der angesprochenen Probleme beim Entwurf nicht optimal ist.

2.1.2.2 Spiralmodell

Für die Softwareentwicklung stellt Boehm in [Boe88] ein Modell mit einem spiralförmigen Ablauf vor, das sogenannte Spiralmodell. Die Spirale beginnt im Ursprung eines Koordinatensystems, dessen Quadranten einzelne Phasen darstellen: Erstens die Bestimmung von Zielen, Alternativen und Begrenzungen, zweitens die Einschätzung der Alternativen sowie die Identifizierung der Risiken, drittens der Entwurf und seine Verifizierung und viertens die Planung der nächsten Phase. Diese vier Phasen werden schrittweise durchlaufen, wobei Boehm wiederum vier Durchläufe unterscheidet ([Boe88, S. 66ff.]):

1. Machbarkeitsstudie
2. Arbeitskonzept

3. Systemspezifikation
4. endgültige Realisierung.

Das von ihm vorgeschlagene Entwurfsmodell wird in Abbildung 2.2 graphisch veranschaulicht.

Der grundlegend neue und bedeutsame Aspekt dieses Ansatzes besteht in der Integration des zyklischen Charakters in die Darstellung des Entwicklungsprozesses. Desweiteren ist das Modell von einer direkten Einbeziehung des Nutzers (durch Tests) sowie der ständigen Suche nach möglichen Alternativen (in jedem Durchlauf) gekennzeichnet. Auf diese Weise können Lücken in den Anforderungen durch die Iterationen nach und nach geschlossen werden.

Wideman weist ergänzend zu diesen positiven Eigenschaften des Modells auf einige nachteilige Aspekte hin ([Wid]): Es besteht die Gefahr, durch zusätzliche Iterationen in ein endloses Kreisen zu gelangen, was stetig verbesserte Prototypen hervorbringt, aber einen Abschluß des Projektes verhindert. Das Kreisen erfordert vom Entwurfskoordinator deshalb Durchführungsdisziplin sowie die genaue Überwachung der Projektzeit- und Projektbudgetpläne, da anderenfalls die Gefahr einer Verletzung derselben besteht.

Forsberg, Mooz und Cotterman bemerken, daß die radiale Darstellung der Zeit in bezug auf die Standardmodellierungsformate inkonsistent sei: "Damit stellt das Modell die Notwendigkeit, die entstehende Zeitbasis zu kontrollieren, verwirrend dar." ([FMC96, S. 25]).

2.1.2.3 V-Modell

Ein weiteres Entwurfsmodell zur Beschreibung der Systementwicklung stammt von Forsberg, Mooz und Cotterman ([FMC96]). Es wird in Anlehnung an seine Form als V-Modell bezeichnet.

Das V-Modell beschreibt ausschließlich die technische Ebene der Entwicklung komplexer technischer Systeme, Budget- und Businessprozesse werden in ihm ausgeblendet. Abbildung 2.3 zeigt die graphische Umsetzung des Modells, das sich aus zwei Phasen zusammensetzt: Den linken, abfallenden Teil der V-Form bilden die Dekomposition und die Definition, während der aufsteigende, rechte Teil die Integration und die Verifikation beschreibt. Beide weisen sowohl in vertikaler (System → Komponente), wie in horizontaler Richtung (Zeit) eine Bedeutungsdimension auf.

Die Dekomposition teilt das System auf allen Ebenen in seine Einzelbestandteile ein. Sie entwirft diese Einzelbestandteile, indem sie Schritt für Schritt nach unten geht und dabei die Spezifikation aufbaut (Design). Die hierbei zu vollziehenden Schritte lauten:

- Analyse der Nutzeranforderungen, Erstellung eines Konzepts und eines Validierungsplans

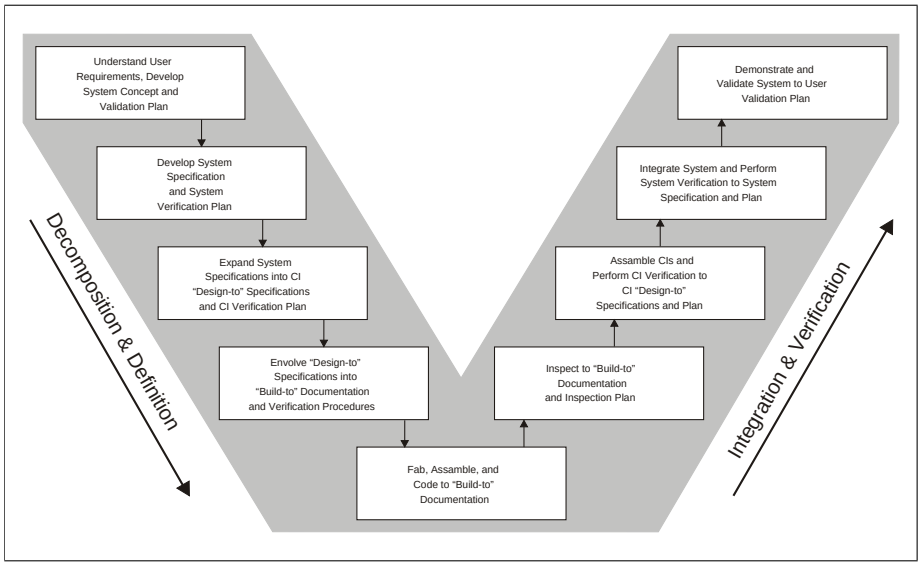

Abbildung 2.3: V-Modell (nach [FMC96, S. 36])

- Entwicklung der Systemspezifikation und eines zugehörigen Verifizierungsplans
- Erweiterung der Systemspezifikation zur Entwurfsspezifikation der Komponenten und Erstellung eines zugehörigen Verifizierungsplans
- Umsetzung der Komponentenentwurfsspezifikation in eine Bauanleitung, inklusive eines Verifizierungsplans

Die Basis der V-Form stellt die Herstellung der Komponenten dar, welche auf Grundlage der Bauanleitung vorgenommen wird.

In der Rekomposition (Integration) wird das Gesamtsystem aus seinen nun hergestellten Komponenten schrittweise zusammengesetzt, wobei auf jeder Ebene die zuvor festgelegten Tests zur Verifikation durchgeführt werden. Die Entwicklung führt also auf der rechten Seite des Modells wieder Schritt für Schritt nach oben. Bei Nichterfüllung eines Tests erfolgt ein Absteigen in ein neues "V" mit einer abgeänderten Spezifikation. Als Resultat eines erfolgreichen Entwurfsprozesses steht das vollständige System für den Einsatz bereit. Ein charakteristisches Merkmal des V-Modells ist, daß die Bestandteile der linken Seite der V-Form jeweils die Tests auf dessen rechter Seite definieren.

Die linke Seite stellt somit den Analyse- und Entwurfsprozeß[1] dar, während die rechte Seite den Integrations- und Verifikationsprozeß[2] beschreibt. Hierin kommt die Unabhängigkeit der Projektevents und des "angewandten Situationsprozesses"[3] zum Ausdruck ([FMC96, S. 37]).

[1] engl.: System Analysis and Design Process (SA&D)
[2] engl.: System Integration and Verifikation Process (SI&V)
[3] engl.: situationally-applied process

Bemerkenswert an diesem Ansatz ist die Einbeziehung von Tests bereits im Entwurfsstadium sowie das strukturierte Vorgehen (top-down) während desselben. Wideman bescheinigt dem Modell eine gute Eignung für Anwendungsfelder, in denen die Anforderungen klar umrissen sind und sich nicht beliebig verändern können ([Wid]). Ein Kritikpunkt ergibt sich jedoch bei den von den Autoren vorgesehenen, umfangreichen Dokumentationsvorgängen: Diese können bei Projekten, die häufigen Veränderungen unterworfen sind, zu Problemen bei der permanenten Aktualisierung der Spezifikationsdokumente führen.

2.1.2.4 System Level Design

Eine der jüngsten Entwicklungen dieser Disziplin stellt der Systementwurf auf Systemebene[4] dar. Das System Level Design basiert in weiten Teilen auf dem V-Modell, ergänzt dieses aber um die Simulation als Testverfahren. Eine Einführung in diese Thematik findet sich bei Buede ([Bue00]).

Platin und Stoy beschreiben den entscheidenden Grund für die Entwicklung des System Level Designs mit dem Schlagwort: Codesign von Funktion und Architektur ([PS99]). Die bisherigen Systementwurfskonzepte betrachteten den Entwicklungsprozeß in einem Voranschreiten von einer Spezifikation der Funktionalität bis zur Realisierung. Komplexe Systeme erbringen ihre Funktion durch eine Mischung aus Software und Hardware. Entscheidungen zur optimalen Aufteilung müssen jedoch frühzeitig getroffen werden und bestimmen die Entwicklungskosten und die Entwicklungszeit maßgeblich mit. Gleichzeitig sind die frühen Entwicklungsphasen von einem Mangel an präzisen Informationen geprägt.

Um die nötigen Entwicklungsentscheidungen frühzeitig treffen und objektiv gestalten zu können, setzt das System Level Design deshalb auf Simulationen, die das "was wäre wenn" ([KL96]) testen. Maßgebliches Kennzeichen der Simulationen ist dabei die Einführung abstrakter Modellierungskomponenten, die dem Fehlen detaillierter Informationen Rechnung tragen. Die Komponentenebene früherer Ansätze wird dabei ergänzt durch ein Umgebungs- und Architekturmodell. Hierdurch entsteht auf der Systemebene ein sogenanntes Gesamtsystemmodell, wie Abbildung 2.4 illustriert.

Für die Durchführung der Tests mittels Simulation werden zusätzlich zum Gesamtsystemmodell ein Treiber- und ein Bewertungsmodell benötigt. Das Treibermodell hält Eingabesequenzen bereit, während das Bewertungsmodell die Systemantwort auswertet. Auf Basis der so durchgeführten Simulationen lassen sich die Auswirkungen von Designentscheidungen untersuchen und gegeneinander abwägen. Der Entwurf berücksichtigt somit sowohl die benötigte Funktionalität, als auch die realisierbare Performance.

> Mangeruca et al. zeigen dies beispielsweise in [MF+03]: Hier beschreiben sie, wie sie mittels des System Level Designs einen Algorithmus zur Klopfgeräuscherkennung auf einer optimierten Hardware umsetzen konnten.

[4] engl.: System Level Design (SLD)

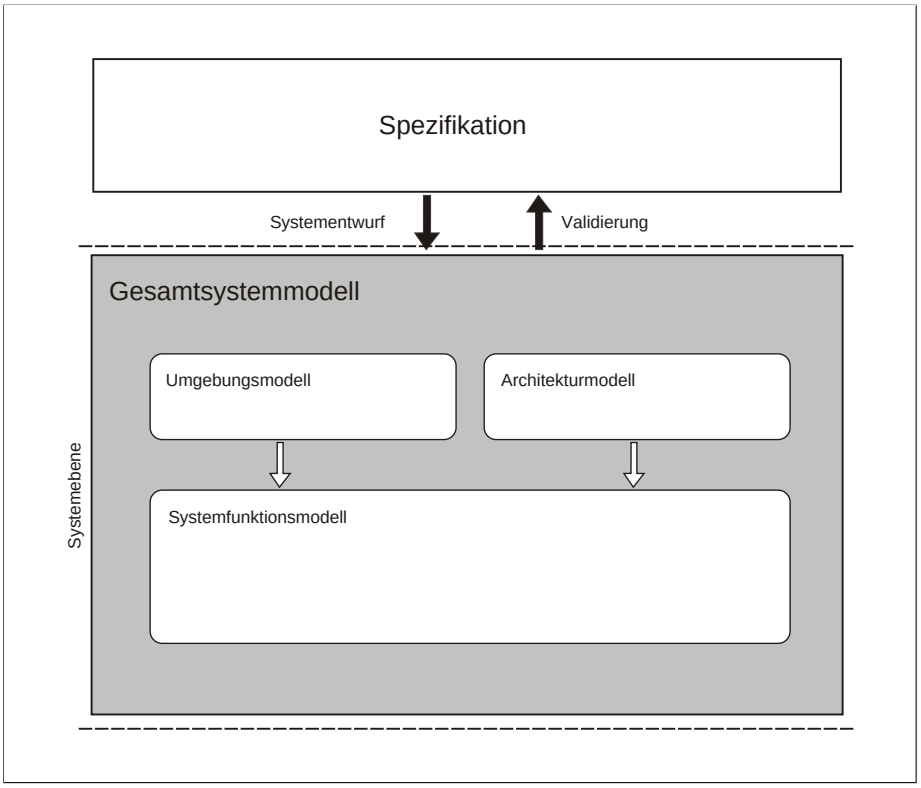

Abbildung 2.4: Systementwurf auf Systemebene (nach [Sch00, S. 21])

Takala stellt fest, daß mit dem Gesamtsystemmodell eine ausführbare Spezifikation entsteht, die sich als virtueller Prototyp interpretieren läßt ([Tak01]).

2.1.2.5 Mission Level Design

Schorcht erweitert den Systementwurf auf Systemebene für Mobilkommunikationssysteme. Er entwickelte eine anforderungsorientierte Systementwurfsmethode mit der Bezeichnung Mission Level Design und nutzt diese für den Entwurf von Mobilkommunikationssystemen ([Sch00]).

Die an ein System gestellten Anforderungen werden hier in Form von definierten, typischen Einsatzszenarien[5], den sogenannten Missionen, beschrieben.

> **Mission:** Eine Mission ist ein typisches Einsatzszenario des Systems.

Die Missionen dienen der Validierung der Spezifikation und damit des gesamten Entwurfs und ersetzen die Treiber- und Bewertungsmodelle des System Level Designs. Ihre Einführung stellt, wie Abbildung 2.5 veranschaulicht, eine Erweiterung des Systementwurfs

[5] engl.: use cases

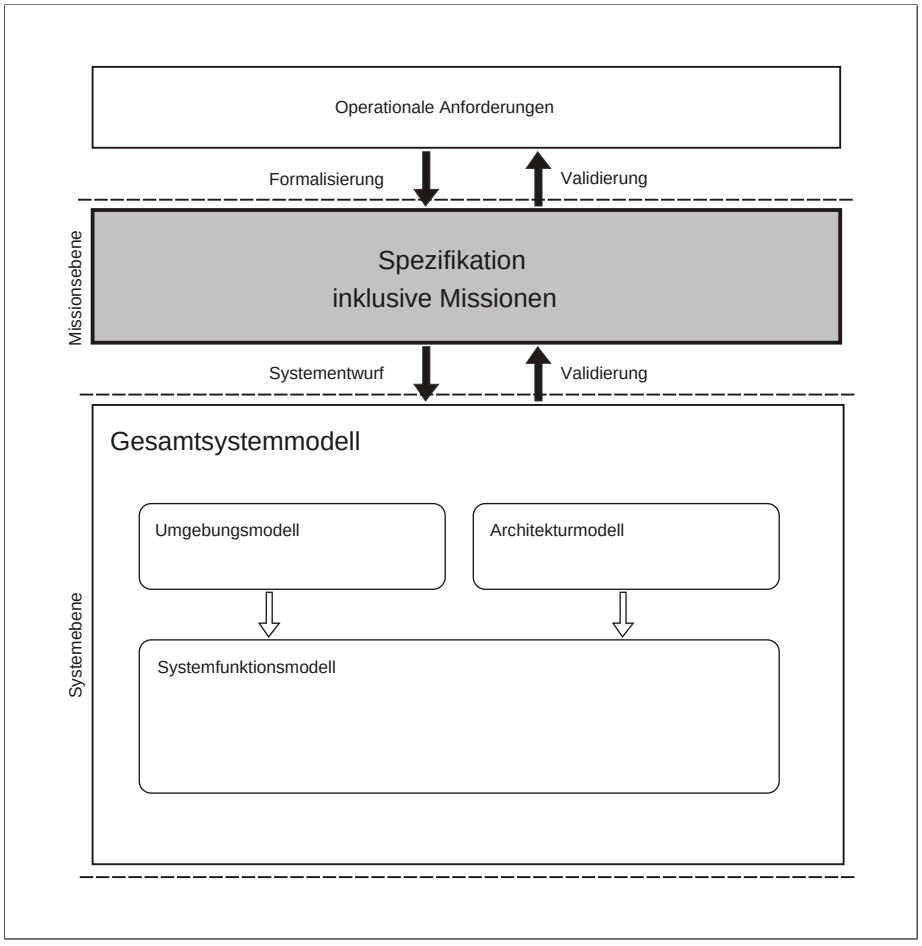

Abbildung 2.5: Systementwurf auf Missionsebene (nach [Sch00, S. 25])

auf Systemebene dar. Als neuer Teil der Spezifikation schreiben die Missionen charakteristische Nutzeranforderungen fest. Das Mission Level Design nimmt somit eine weitere Formalisierung des Entwurfs vor und verschiebt den notwendigen Arbeitsaufwand in die frühen Entwurfsphasen. Hierdurch soll die Systemintegration weniger aufwendig und vor allem fehlerfrei verlaufen. Für die Simulation wird - soweit möglich - auf verifizierte Bibliotheken zurückgegriffen. Der hiermit verbundene Vorteil liegt in der Wiederverwendung bereits getesteter Bestandteile. Angesichts der üblichen Praxis, in der nur selten vollständig neue Systeme entworfen werden, stellt dies einen realistischen Ansatz dar, der gerade für den Entwurf komplexer Systeme sinnvoll ist.

Die Einführung von Missionen ermöglicht eine deutlich verbesserte Interpretation der gewonnenen Ergebnisse. Das Ergebnis ihrer Verwendung zum Test des Systementwurfs sind nicht allgemeingültige Aussagen über das allgemeine Systemverhalten: Durch den Test typischer Fälle werden mittlere, typische Aussagen für Nutzungsszenarien getroffen. Dies ist eine gewünschte Einschränkung: Bedingt durch die den Systemen zugrunde liegende Komplexität sind allgemeingültige Aussagen für alle denkbaren Fälle nicht mehr möglich. Der Missionsauswahl kommt insofern eine erhebliche Bedeutung für die Testergebnisse zu. Ziel muß es sein, eine Auswahl sinnvoller, typischer Varianten des gesamten Einsatzspektrums vorzunehmen. Als limitierende Faktoren für die Anzahl der möglichen Missionen sind die Simulationsdauer sowie die Grenzen der manuellen Auswertung zu beachten. Die erzielten Ergebnisse basieren dabei stets auf der Auswertung der Gesamtheit aller Missionen.

2.2 Simulation

2.2.1 Einführung

Die Simulation stellt eine universelle Methodik dar, für die sich entsprechend vielfältige Einsatzmöglichkeiten bieten. Der folgende Abschnitt dient zur Klärung ihrer allgemeinen Begrifflichkeiten. Zunächst stehen die Begriffe Simulation und Modell im Zentrum des Interesses. Hieran schließen sich Ausführungen zu den drei Phasen der Simulation - Modellierung, Modellnutzung und Auswertung - an.

2.2.1.1 Simulation und Modell

Basis der Auseinandersetzung mit der Simulationsthematik ist ein grundlegendes Verständnis ihres Wesens und ihrer zentralen Begriffe. Zu diesem Zweck definiert die VDI-Richtlinie 3633 den Simulationsbegriff wie folgt ([VDI96, Jak97]):

Simulation: Die Simulation ist ein Verfahren zur Nachbildung eines Systems mit seinen dynamischen Prozessen in einem experimentierbaren Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind.

Der Schwerpunkt dieser Definition liegt auf der Aussage, daß die Simulation eine Nachbildung des Systems inklusive seiner Prozesse vornimmt. Unter einem Prozeß ist ein Vorgang in einem System zu verstehen, der zum Transport, zur Speicherung oder zur Umformung von Energie, Materie oder Informationen dient. Hierbei wird auf den bereits in Abschnitt 2.1.1 vorgestellten Systembegriff zurückgegriffen. Für die Nachbildung der Dynamik wird der Systembegriff um die Zustandsvariablen (oder kurz: Zustände) erweitert, die die zeitliche Veränderung von internen Systemgrößen festhalten.

Die VDI-Definition des Simulationsbegriffs weist desweiteren auf den vorrangigen Zweck einer Simulation - den Erkenntnisgewinn - hin. Die Simulation wird also typischerweise genutzt, um ein besseres Verständnis komplizierter dynamischer Prozesse zu erlangen ([Ada01, S. 57ff.]), beziehungsweise um komplexe technische Systeme zu entwerfen ([Ben95, S. 9],[Sch02, S. 34]).

In jedem Fall benötigt die Simulation für ihre Untersuchungen ein Modell des Systems. Der Begriff des Modells stellt insofern einen zweiten zentralen Begriff des Simulationsbereichs dar. Er ist folgendermaßen definiert ([VDI96]):

Modell: Ein Modell ist eine vereinfachte Nachbildung eines existierenden oder gedachten Systems mit seinen Prozessen in einem anderen begrifflichen oder gegenständlichen System. Es unterscheidet sich hinsichtlich der untersuchungsrelevanten Eigenschaften nur innerhalb eines vom Untersuchungsziel abhängigen Toleranzrahmens vom Vorbild.

Bemerkenswert ist hierbei die Feststellung, daß ein Modell seinem Vorbild (dem realen System) nur hinsichtlich der untersuchungsrelevanten Eigenschaften ähnlich ist. Damit wird klar, daß ein Modell immer im Hinblick auf einen bestimmten Aspekt erstellt wird und daß die Simulationsergebnisse nur für diesen Aspekt Gültigkeit besitzen.

In der Simulation wird zwischen den Begriffen Simulationsstudie, Experiment und Simulationslauf unterschieden. Sie stehen zueinander in folgendem Zusammenhang:

- Eine **Simulationsstudie** ist eine simulationsunterstützte Untersuchung eines Systems.
- Ein **Experiment** ist eine gezielte Untersuchung des Modellverhaltens über einen bestimmten Zeithorizont. Dabei wird der Eingang festgelegt, auf den das System reagieren soll. Durch die Reaktion erzeugt das System einen Ausgang, aus dessen Verhältnis zum Eingang auf das interne Verhalten des Systems geschlossen werden kann.

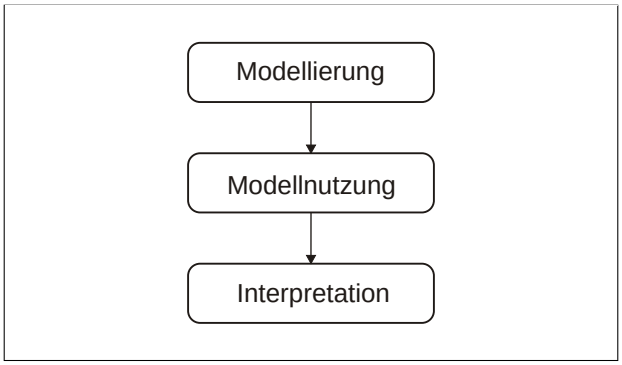

Abbildung 2.6: Phasen der Simulation

- Ein **Simulationslauf** ist die Nachbildung des Verhaltens eines Systems mit einem spezifischen, ablauffähigen Modell über einen bestimmten (Modell-)Zeitraum. Das heißt, er ist eine Durchführung eines Experiments.

Eine Simulationsstudie kann demnach verschiedene Experimente umfassen, wobei für jedes von ihnen jeweils mehrere Simulationsläufe möglich sind.

Die Durchführung der Simulation geschieht grundsätzlich in drei Phasen ([Sch90, S. 12], [Ben95, S. 18], [VDI96, S. 15]), wie Abbildung 2.6 aufzeigt: Den Anfang bildet die Modellierungsphase, an die sich die Modellnutzungs- und die abschließende Interpretationsphase anschließen. Alle drei Phasen sollen im Folgenden näher beschrieben werden.

2.2.1.2 Modellierung

Die Modellierung stellt die Phase dar, in welcher das Modell, auf dem die Simulation basiert, erzeugt wird. Die VDI-Richtlinie 3633 definiert sie wie folgt ([VDI96, S. 9]):

> **Modellierung:** Die Modellierung umfaßt bei der Simulation das Umsetzen eines existierenden oder gedachten Systems in ein experimentierbares Modell.

Die Betonung liegt dabei auf einer experimentierbaren Umsetzung. Dies bedeutet, daß ein ablauffähiges Modell erstellt wird. Dafür existieren eine Vielzahl von Möglichkeiten, die sich wie in [Möl92, S. 93ff.] oder [Tho96, S. 7] beschrieben kategorisieren lassen.

Für die Modellierung werden heute typischerweise komfortable Simulationsumgebungen genutzt, die eine *blockorientierte, parametrisierbare Modellierung* ermöglichen. Jeder der Blöcke verfügt über eine Menge von Eingängen, Ausgängen und Parametern. Die Ein- und Ausgänge transportieren dabei Daten in den beziehungsweise aus dem Block. Die Parameter konfigurieren die Funktionalität des Blocks, die darin besteht, einen oder mehrere Prozesse nachzubilden. Parameter sind definiert als ([VDI96, S. 11]):

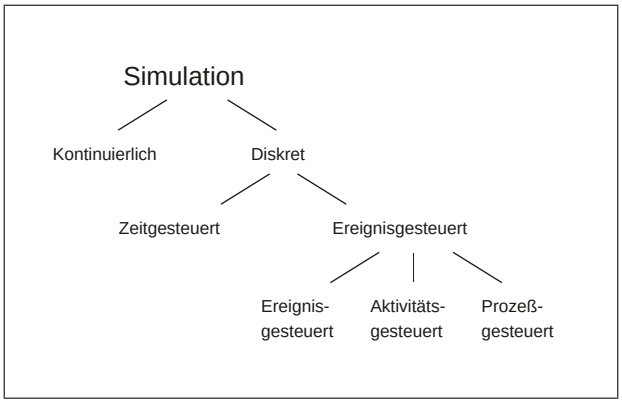

Abbildung 2.7: Ansätze zur Simulation dynamischer Prozesse

Parameter: Ein Parameter ist der Wert eines Attributs, der vor einem Simulationslauf gesetzt wird und sich innerhalb des Laufs nicht ändert.

Für die Modellierung der *Dynamik*, also des zeitlichen Verhaltens, existieren unterschiedliche Ansätze, die sich wie in Abbildung 2.7 dargestellt differenzieren lassen. Zum einen können dynamische Prozesse kontinuierlich sein. Zum anderen lassen sich diskrete dynamische Prozesse durch ein diskretes Zeitraster beschreiben. Trieb unterteilt diese in zeitgesteuert und ereignisgesteuert ([Tri97, S. 30]). Erstere besitzen einen oder mehrere Takte, während letztere von Ereignissen, Aktivitäten oder Prozessen gesteuert werden.

Bei dynamischen Prozessen besitzt ein Block intern zusätzlich sogenannte Zustände, die zeitpunktbezogene Informationen speichern.

Die modernen Simulationsumgebungen stellen mehrere Ansätze für die Modellierung der Dynamik bereit. Sie bezeichnen diese Ansätze dabei oft als *Domänen*. Einige der am häufigsten benutzten sind:

- **Continous Time (CT)**
  modelliert ein kontinuierliches Zeitverhalten von Prozessen. Dies wird durch eine Modellbildung mittels Differentialgleichungen erreicht. Sie beschreiben das Übergangsverhalten der Zustände über die zeitlichen Ableitungen derselben, wobei in jedem gegebenen, endlichen Zeitintervall eine unendliche Anzahl Zustandsänderungen auftreten kann, jedoch nicht zu diskreten Zeitpunkten ([BR00]).
- **Descrete Event (DE)**
  ermöglicht es, das zeitdiskrete Verhalten von Prozessen nachzubilden. In chronologischer Reihenfolge abgearbeitete Ereignisse bestimmen dabei den Ablauf der Simulation. Sie werden durch Taktung oder durch andere Ereignisse generiert. Zwischen zwei diskreten Ereignissen verändert sich der Systemzustand nicht.
- **Finite State Machines (FSM)**
  sind endliche Zustandsautomaten. Sie definieren Zustände, die durch bedingte

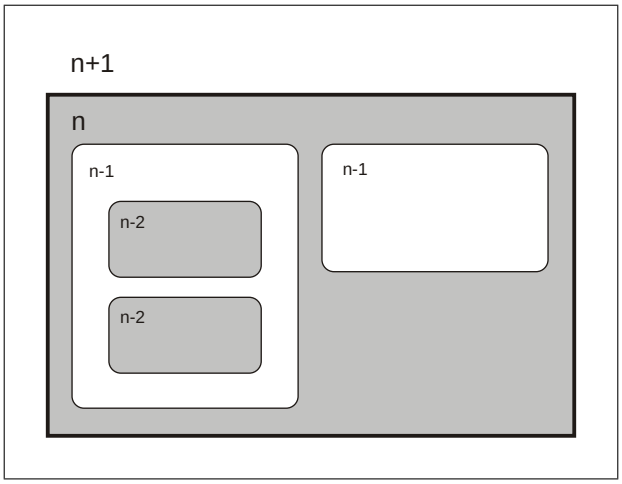

Abbildung 2.8: Hierarchie von Modellen

Übergänge miteinander verbunden sind. Zustandsübergänge erfolgen sequentiell aufgrund von Veränderungen an den Eingängen. Die Ausgabe des Automaten wird vom aktiven Zustand bestimmt. Zustandsautomaten eignen sich besonders für die Beschreibung von Steuerlogiken.

Eine umfassende Beschreibung dieser und weiterer Ansätze findet sich beispielsweise bei Lee ([Lee02]).

Nutzt eine Simulation mehrere Domänen, so wird sie als Multidomän- oder auch hybride Simulation bezeichnet. Die Durchführung übernimmt in der Simulationumgebung ein Scheduler, der die zeitliche Abarbeitung der Blöcke steuert. Eine hybride Simulation stellt an diesen besondere Anforderungen - nicht jede Simulationsumgebung ist deshalb zu deren Durchführung in der Lage.

Unter einer *Hierarchie* versteht man eine Strukturierung des Modells zum Zwecke der größeren Übersichtlichkeit und besseren Wiederverwendbarkeit einzelner Komponenten. Adamski stellt in [Ada01, S. 24ff.] ein Maß für Hierarchieebenen vor. In diesem erhält das zu untersuchende Ausgangssystem die Ordnungszahl n, untergeordnete Systeme werden mit $n-1$, $n-2$, usw., übergeordnete mit $n+1$, $n+2$, usw. gekennzeichnet. Eine Visualisierung dieser Definition zeigt die Abbildung 2.8. Adamski verweist in diesem Zusammenhang darauf, daß die Ordnungszahl 0 ein fiktiver Wert ist, der für ein nicht mehr unterteilbares atomares System steht. Was diesem System entspricht, hängt von der konkreten Fragestellung ab.

Ein weiterer zentraler Gesichtspunkt jeder Simulation ist die Einschätzung, ob das nachgebildete Verhalten dem erwarteten entspricht: die *Validierung*. Deren Definition lautet nach [VDI96, S. 18]:

> **Validierung:** Die Validierung ist die Überprüfung der hinreichenden Übereinstimmung von Modell und System. Es ist sicherzustellen, daß das Modell das Verhalten des realen Systems im Hinblick auf die Untersuchungsziele genau genug und fehlerfrei widerspiegelt. Eine vollständige Übereinstimmung (...) ist aufgrund von Ungenauigkeiten bei der Systemdatenerfassung und durch die Abstraktion beim Modellaufbau nicht möglich.

Eine zentrale Aussage dieser Definition ist der Hinweis, daß lediglich eine im Hinblick auf die konkrete Zielstellung hinreichend genaue Übereinstimmung zwischen Modell und System zu erreichen ist. Das Modell weist demnach eine zielgerichtete Modellierung auf und ist deshalb nur zur Untersuchung von Fragen einsetzbar, die dieser Zielrichtung entsprechen.

Ist aufgrund einer fehlenden Datenbasis ein Vergleich mit Systemdaten nicht möglich, muß die Validierung durch eine Plausibilitätskontrolle ersetzt werden. In diesem Fall erfolgt die Prüfung auf hinreichende Genauigkeit der Nachbildung anhand von Schätzungen beziehungsweise auf Grundlage logischer Überlegungen.

Die *Verifikation* dient - im Gegensatz zur Validierung - der Feststellung der *formalen* Korrektheit des Modells. Dieser Test auf formaler Ebene geht beispielsweise der Frage nach, ob alle Ein- und Ausgänge verbunden sind. Zahlreiche Simulationsprogramme führen diesen Test vor einen Simulationslauf automatisch aus.

2.2.1.3 Modellnutzung

In der zweiten Phase der Simulation werden die einzelnen Simulationsläufe der Experimente durchgeführt. Dazu wird das Modell von der Simulationsumgebung in eine ablauffähige Form umgewandelt und abgearbeitet. In diesem Schritt entstehen die Simulationsergebnisse in Form von Daten, die beim Ausführen des Modells erzeugt werden.

2.2.1.4 Interpretation

In der abschließenden Phase jeder Simulation werden die in den Experimenten generierten Daten aufbereitet und interpretiert. Ziel ist hierbei ein Erkenntnisgewinn im Hinblick auf die zu Anfang definierten Untersuchungsziele.

Die Auswertung der Ergebnisse verlangt zunächst eine *Aufbereitung* der Rohdaten in eine interpretierbare Form. Typischerweise handelt es sich bei den Rohdaten um Zahlenreihen sowie beschreibende oder abgeleitete Daten. Diese müssen selektiert, sortiert, formatiert, verdichtet und dargestellt werden. Die drei erstgenannten Vorgänge bereiten die Daten für die Auswertung vor. Im Anschluß daran reduziert die Verdichtung die Daten auf

eine überschaubare Menge und steigert dadurch ihre Aussagekraft. Für die abschließende *Darstellung* stehen nach [VDI96, S. 4] folgende Möglichkeiten offen:

- **Statistiken:**
  Statistiken sind geordnete Mengen von Informationen in Form von empirischen Daten. Gleichzeitig bezeichnet der Begriff ein Verfahren, nach dem empirische Zahlen gewonnen, dargestellt, verarbeitet, analysiert und für Schlußfolgerungen, Prognosen und Entscheidungen verwendet werden.

- **Monitoring:**
  Hierunter versteht man die Anzeige von Zustandsgrößen mit alphanumerischen oder graphischen Darstellungsformen während des Simulationslaufs am Bildschirm.

- **Präsentationsgraphiken:**
  Sie dienen der Darstellung von Simulationsergebnissen anhand von zeitpunktbezogenen und/oder zeitraumbezogenen Diagrammen.

  Dabei lassen sich verschiedene Typen unterscheiden. Klassische Beispiele, die von den meisten Simulationsprogrammen als konfigurierbare Standarddiagramme bereitgestellt werden, sind Kreis- und Liniendiagramme sowie Kennlinien. Zusätzlich zu diesen existieren jedoch spezielle Lösungen zur Darstellung.[6]

- **Animationen:**
  Animationen sind bewegte Bildschirmgraphiken, durch die Zustandsänderungen oder Bewegungen von Modellelementen angezeigt werden können.

Hinsichtlich ihrer Verfügbarkeit während oder nach der Simulation werden diese Darstellungsmöglichkeiten als *online* oder *offline* bezeichnet. Eine Online-Darstellung erfolgt noch während der Simulation, die Offline-Variante danach.

Typischerweise unterstützen die Simulationsprogramme zusätzlich zu den beschriebenen Darstellungsmöglichkeiten eine Ausgabe der Daten in Dateien, die die Basis für eine externe Weiterverarbeitung der Daten bilden. Eine Alternative zu diesen Dateien bilden *Datenbanken*, die ebenfalls für die Speicherung und Verwaltung der Daten eingesetzt werden können.[7] Nach Lütkepohl existieren für den Einsatz von Datenbanken in der Simulation verschiedene Einsatzgebiete ([LPR93]):

- Speicherung von empirischen Massendaten als Eingabedaten für Simulationen
- Speicherung von Zeitreihen und aggregierten Daten als Ausgabedaten aus Simulationen
- Durchführung von speziellen, nachträglichen Auswertungen der Einzelergebnisse eines Simulationslaufs

[6] Ein Beispiel hierfür ist die Darstellung der Scans eines Computertomographen in Form aufbereiteter dreidimensionaler Strukturen.

[7] Ritzschke und Wiedemann beschreiben dies beispielsweise in [RW98].

- Durchführung von vergleichenden Auswertungen der Daten verschiedener Simulationsläufe.

In Verbindung mit einer Offline-Darstellung ist eine Entkoppelung von der Simulation realisierbar. Dies erlaubt eine wiederholbare Auswertung zeitlich unabhängig von der Simulation. Daneben bietet die Datenbank die Möglichkeit, zugehörige Metadaten (z.B. Datum, Rechner, Benutzer) mit den Simulationsdaten zu verknüpfen. Diese Metadaten helfen bei der Verwaltung einer großen Anzahl von Experimenten und/oder Simulationsläufen, die in der Datenbank an einer zentralen Stelle vorliegen.

2.2.2 Simulationsprogramme

2.2.2.1 Überblick

Für die Durchführung von Simulationen steht eine Vielzahl von Simulationsprogrammen und -sprachen zur Verfügung, wie beispielsweise die von Rizzoli ([Riz]) zusammengestellte Übersicht der aktuell verfügbaren Programme zeigt. Viele dieser Werkzeuge wurden für die Realisation einer speziellen Aufgabe entwickelt. Beispiele hierfür sind die Programme PSpice (Cadense) für den Schaltungsentwurf und RADSIM (SAIC) zur Simulation von Radarbildern.

Daneben existieren universell einsetzbare Programme, von denen viele aber dennoch eine spezielle Ausrichtung aufweisen. Ein Beispiel dieser Kategorie ist MATLAB/Simulink (The MathWorks GmbH [Mat03]). Es handelt sich dabei um eine Simulationsanwendung, die für Aufgaben der Steuerungs- und Regelungstechnik eine weite Verbreitung gefunden hat, aber trotzdem universell einsetzbar ist.

Moderne Vertreter dieser Gattung von Simulationsprogrammen zeichnen sich durch die folgenden Merkmale aus:

- Graphische Programmierumgebung für eine einfache und strukturierte Modellbildung auf Basis vorhandener Blöcke, die nur entsprechend parametrisiert werden müssen. Damit wird eine schnelle und übersichtliche Modellierung ermöglicht.
- Erweiterbarkeit des Programms zur Realisierung eigener Blöcke für spezielle Anforderungen oder als Ersatz für größere Modellstrukturen. Auf diese Weise können anwenderspezifische Bibliotheken geschaffen werden.
- Verschiedene Domänen und eine hybride Simulation stellen sicher, daß jeweils der einfachste Modellierungsansatz verwendet werden kann.

Damit eignen sie sich auch für Simulationen im Rahmen des System Level Designs.

Speziell für den Entwurf auf Missionsebene existiert MLDesigner (MLDesign Technologies, Inc. [MLD03b]). Es handelt es sich dabei um eine integrierte Plattform zur Modellierung von Architektur, Funktion und Umwelt für die Analyse der Leistungsfähigkeit von Systementwürfen.

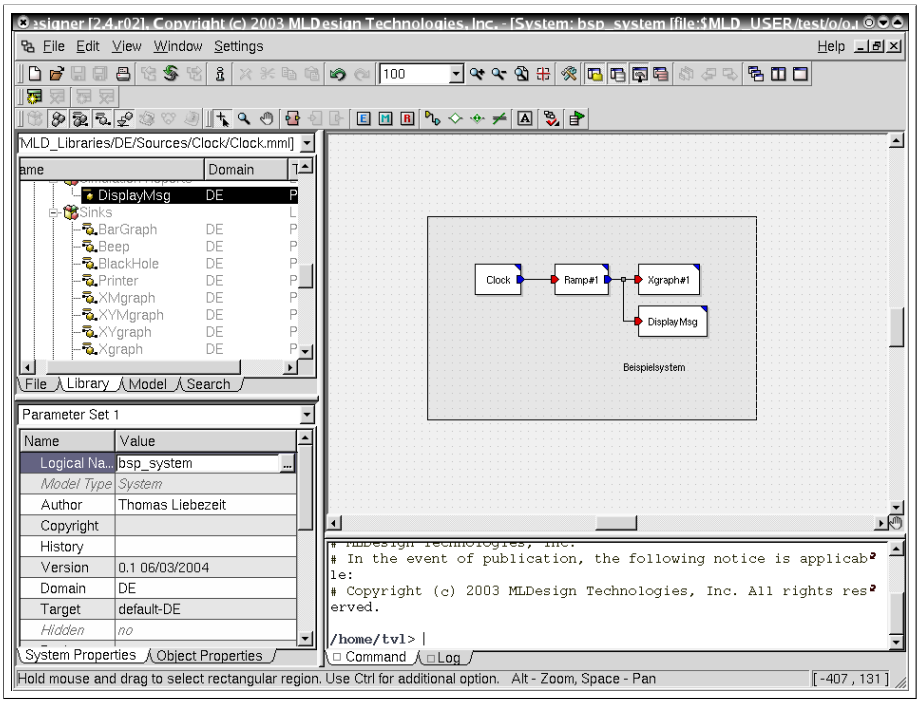

Abbildung 2.9: MLDesigner

2.3 MLDesigner

2.3.1 Überblick

Ein wesentlicher Bestandteil dieser Arbeit beruht auf der Nutzung des Entwurfswerkzeugs MLDesigner.Bei diesem handelt es sich um eine speziell auf die Erfordernisse des Mission Level Designs ausgerichtete Anwendung. Da bei Beginn dieser Arbeit bereits eine Reihe von Projekten existierte, die mit Hilfe von MLDesigner erfolgreich analytische Betrachtungen von Systemen durchführten, bestand eine Rahmenbedingung dieser Arbeit in der Nutzung von MLDesigner.

MLDesigner ist eine Simulationsumgebung der vierten Generation, die für die Arbeit im Rahmen des Systementwurfs entwickelt wird. Entsprechend unterstützt das Programm unterschiedliche Abstraktionsebenen für den Entwurf. Sie reichen von Algorithmen und Funktionen über Architektur und Performance (System Level Design) bis zu operationalen Vorgaben/Missionen (Mission Level Design). Die Fähigkeiten des Programms eignen sich dabei für den top-down und den iterativen Systementwurfsprozeß. Die folgende Auflistung gibt einen kurzen Überblick der Hauptmerkmale des Programms wieder:

- **Integrierte Entwicklungsumgebung (IDE)**
  MLDesigner bietet eine moderne integrierte Entwicklungsumgebung (siehe Abbildung 2.9), die eine graphische, blockorientierte Modellierung ermöglicht. Diese setzt sich aus unterschiedlichen Bestandteilen zusammen. So verfügt MLDesigner über

einen leistungsfähigen Modelleditor[8], in dem das Modell gebaut wird. Ein Filemanager[9] organisiert die dafür zur Verfügung stehenden Blöcke. Die Eigenschaften (Parameter) der verwendeten Blöcke lassen sich im zugehörigen Eigenschaftseditor[10] einstellen. Die Konsole[11] vereint unterschiedliche Funktionalitäten. Sie bietet unter anderem Zugang zu den Logmeldungen, zur Kommandozeile und dem Simulationsfortschritt (im Simulationsmodus). Desweiteren verfügt MLDesigner über einen Editor zum Erstellen von Datenstrukturen.

Das Programm unterstützt die Einbindung von Tcl/Tk in den Simulationsprozeß. So lassen sich beispielsweise dynamische Oberflächen für die interaktive Vorgabe von Simulationswerten erstellen (z.B. Buttons und Schieberegler). Daneben ermöglicht es die komfortable Fehlerbeseitigung, indem es eine schrittweise Ausführung (inklusive der graphischen Hervorhebung des aktuell ausgeführten Blocks und Breakpoints) bereitstellt.

- **Domänen** (multi)
  MLDesigner unterstützt verschiedene primäre und sekundäre Modellierungsdomänen. Unter ihnen sind DE, CTDE, DDF und SDF sowie als Subdomänen FSM und HOF. Die CTDE-Domäne ermöglicht beispielsweise analoge und gemischt analog-diskrete Entwürfe. Die einzelnen Domänen lassen sich dabei gemischt verwenden (hybride Simulation).

- **Kompatibilität und Erweiterbarkeit**
  Um vorhandene Modelle weiterverwenden zu können, wird der Import aus BONeS und COSSAP unterstützt. MLDesigner ist mittels C++ und einer eigenen Beschreibungssprache beliebig erweiterbar.

- **Externe Simulationen**
  Simulationen lassen sich auch außerhalb des Programms durchführen. Dazu wird das Modell übersetzt und mit den Programmbibliotheken verlinkt. Diese externe Abarbeitung beschleunigt die Simulation, wobei die Modellparameter über Parameterdateien weiterhin änderbar bleiben.

Weitere Informationen finden sich auf der Homepage des Herstellers ([MLD03b]).

2.3.2 Modellierung mit MLDesigner

Da die Modellierung des Gesamtsystemmodells im Rahmen dieser Arbeit mit MLDesigner vorgenommen wird[12], gibt der folgende Abschnitt eine überblicksmäßige Einführung in die Modellierung mit diesem Programm. Dazu stellt er zuerst die für die Modellierung

[8] oben rechts in Abbildung 2.9
[9] oben links in Abbildung 2.9
[10] unten links in Abbildung 2.9
[11] unten rechts in Abbildung 2.9
[12] siehe Abschnitt 4.1.2.1, Seite 67

zur Verfügung stehenden Grundelemente vor. Im Anschluß wird die Modellerstellung thematisiert. Weiterführende, detaillierte Informationen finden sich im MLDesigner-Benutzerhandbuch ([MLD03a]).

2.3.2.1 Grundelemente der Modellierung

Für die Modellierung mit MLDesigner stehen die folgenden Grundelemente zur Verfügung:

- **Blöcke:**
  Als blockorientiertes Entwurfswerkzeug unterscheidet MLDesigner drei verschiedene Typen von Blöcken:
  - Die **Primitive** ist der kleinste, nicht mehr unterteilbarer Block. Sie besitzt eine bestimmte Funktionalität, die durch C++-Code realisiert wird.
  - Das **Modul** vereinigt eine oder mehrere Primitiven oder andere Module und sorgt so für eine hierarchische Gliederung des Modells[13].
  - Das **System** stellt die oberste Modellebene (n) dar. Es enthält eine oder mehrere Primitiven und/oder Module.

  Alle drei Arten von Blöcken können *Parameter* besitzen, durch welche sie parametrisiert werden. Module und Primitiven können desweiteren über *Ein- und Ausgänge* verfügen.

  Neben einfachen Ein- und Ausgängen unterstützt MLDesigner Mehrfach-Ein- und Ausgänge (*Multiports*), welche eine beliebige Anzahl von Verbindungen besitzen können. Für Primitiven ist dies vorteilhaft, wenn eine variable Anzahl von Eingängen behandelt und/oder eine variable Anzahl von Ausgängen erzeugt werden muß.

  Ein Beispiel hierfür ist eine Primitive Add, welche die Summe aus einer beliebigen, erweiterbaren Anzahl von Eingangswerten in der DE-Domäne bildet. Eine solche Primitive läßt sich sehr einfach mit einem Multieingang realisieren. Der Code wird so geschrieben, daß die Primitive zur Laufzeit über alle Eingänge iteriert und testet, ob an jedem der Eingänge mit den Summanden ein Wert anliegt. Ist dies der Fall, addiert sie alle Werte und gibt das Ergebnis über den Ausgang aus. Die genaue Anzahl der benötigten Summanden wird in der Modellierung durch die erstellte Anzahl von Verbindungen mit dem Multiport festgelegt.

  Für Primitiven mit Multiports erlaubt MLDesigner, speziell abgeleitete Primitiven mit einer festgelegten Anzahl an Ein- bzw. Ausgängen zu erstellen.

  Für das obige Beispiel bedeutet dies die Möglichkeit, ein Spezial-Add mit drei Eingängen und einen Ausgang zu schaffen, wobei jeder Eingang einen Wert aufnehmen kann.

[13] Die Modellebenen ausgehend von $n-1$ und unterhalb werden immer durch entsprechende Module realisiert.

Diese Spezialprimitiven eignen sich besser für die Modellerstellung als Varianten mit Multiports, da sie eine größere Übersicht über die Belegung der Ein- bzw. Ausgänge bieten.

- **Memories, Events und Ressourcen:**
  Als zusätzliche Modellierungselemente unterstützt MLDesigner Memories (memory shared elements), Events (event shared elements) und Ressourcen (shared items). Memories machen Informationen (Daten) zwischen verschiedenen Modulen/Primitiven nutzbar, die nicht miteinander verbunden sind[14]. Mittels der Events läßt sich ein eigenes Timing realisieren, wobei die Events auch Daten übertragen können. Aus der Gruppe der Ressourcen unterstützt MLDesigner quantitative und Serverressourcen.

- **Datenstrukturen:**
  MLDesigner ermöglicht es dem Anwender ferner, zusammengesetzte Datenstrukturen zu erstellen. Dies ist nützlich, weil sich auf diese Weise eigene Strukturen (z.B. Protokolle) direkt in die Modellierung einbeziehen lassen. Für die Arbeit mit diesen Strukturen stehen entsprechende Primitiven bereit (z.B. zur Erstellung und Dekomposition von Listen).

2.3.2.2 Modellerstellung

Die vorgestellten Grundelemente werden für die Erstellung von Modellen genutzt. Systeme und Module werden in MLDesigner über die graphische Benutzeroberfläche zusammengefügt. Dies geschieht, indem die einzelnen Bestandteile aus den Bibliotheken in das Modell eingefügt werden. Im Anschluß werden die jeweiligen Ein- und Ausgänge miteinander verbunden und die Parameter gesetzt.

Bei Primitiven werden die Ein-, die Ausgänge und die Parameter graphisch erstellt, während die eigentliche Programmierung in einem integrierten Editor vorgenommen wird. Er präsentiert das Grundgerüst mit den Methoden, aus denen jede Primitive aufgebaut ist. Diese werden mit der benötigten Funktionalität ausprogrammiert, wobei auf den vollen Sprachumfang von C++ zurückgegriffen werden kann. Dabei sind beliebige Bibliotheken und eigene Objekte nutzbar[15].

MLDesigner verfügt bereits über eine große Anzahl vorgefertigter Primitiven. Sie sind in einzelnen Bibliotheken nach ihren Domänen und innerhalb derselben nach funktionalen Eigenschaften organisiert. Anhand diverser Beispielsysteme läßt sich ihr Einsatz nachvollziehen. Desweiteren ist zu jeder Primitive eine Beschreibung ihrer Funktionalität

[14] Ein Beispiel für den Einsatz dieses Modellierungselementes bietet die im Rahmen dieser Arbeit geschaffene Datenbankanbindung (siehe Abschnitt 5.3.3.8, Seite 132).

[15] Dazu müssen die entsprechenden Headerdateien eingebunden werden. Zusätzlich sind in diesem Fall die Pfade zu den compilierten Bibliotheken und Objektdateien einzustellen. In diesem Zusammenhang muß die korrekte Reihenfolge der Einträge beachtet werden, da anderenfalls Fehler beim Linken auftreten können.

vorhanden, die auch detaillierte Informationen zu den jeweiligen Eingängen, Ausgängen und Parametern liefert. Die Bibliotheken sind vom Nutzer durch eigene Systeme, Module und Primitiven erweiterbar.

2.4 Mobile automatische Systeme

2.4.1 Mobile Systeme

Gegenstand dieser Arbeit ist die Anwendung der im Abschnitt 2.1.2.5 vorgestellten Entwurfsmethode Mission Level Design auf die Kategorie der mobilen automatischen Systeme. Basis der weiteren Überlegungen muß aus diesem Grund die Vorstellung dieser Kategorie von Systemen sein.

Mit dem Begriff mobile automatische Systeme wird eine Kategorie von Systemen beschrieben, die bestimmte gemeinsame Eigenschaften aufweisen. Diese Eigenschaften sind in den beiden kennzeichnenden Begriffen *mobil* und *automatisch* festgehalten. Aufgrund der zunehmenden Bedeutung dieser Systeme soll an dieser Stelle eine kurze Einführung gegeben werden. Sie klärt zuerst den Begriff mobiles System, um danach die mobilen automatischen und die mobilen autonomen Systeme zu behandeln.

Eine mögliche Definition für mobile Systeme lautet ([Wer03]):

> **Mobiles System:** Ein mobiles System ist ein technisches und/oder biologisches System, daß sich in den Räumen $\Re^1$, $\Re^2$ und $\Re^3$ bewegen und Aufgaben erfüllen kann. Es besitzt Sensoren zur Erfassung der eigenen Zustände und/oder der Umwelt. Informationssysteme verbinden die Sensoren mit Entscheidungssystemen und Aktoren. Die Aktoren des mobilen Systems garantieren die Bewegung im $\Re^n$ und übernehmen weitere Aufgaben. Die Sensor-, Entscheidungs- und Aktoraufgabe kann von technischen und/oder biologischen Systemen übernommen werden.

Mit dieser Definition wird festgestellt, daß sich ein System im Raum bewegen können muß, um als mobil zu gelten. Gleichzeitig wird die allgemeine Struktur mobiler Systeme beschrieben. Sie umfaßt einerseits das mobile System und andererseits die Umwelt. Das mobile System seinerseits besteht aus

- den **Sensoren**, die die Umwelt wahrnehmen,
- einem **Entscheidungssystem**, das die Kontrollentscheidungen trifft und
- einem oder mehreren **Aktoren**, die für die Bewegung sorgen und Aufgaben erfüllen.

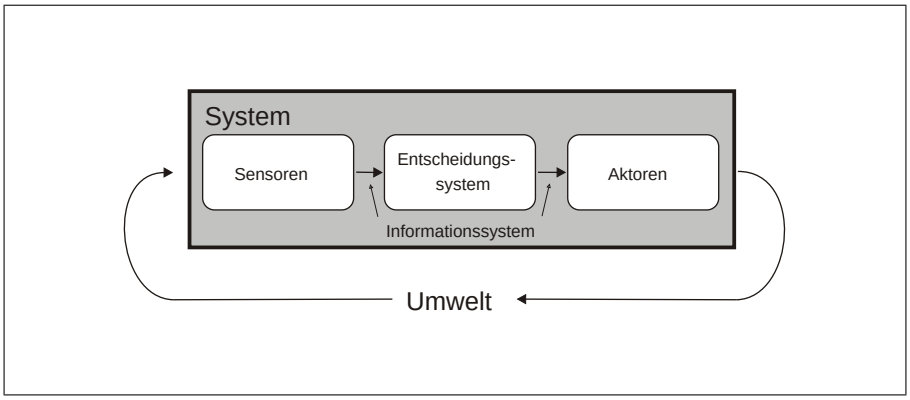

Abbildung 2.10: Aufbau mobiler Systeme

Alle drei Bestandteile sind dabei über ein Informationssystem verbunden, wie Abbildung 2.10 zeigt. Mit dieser sehr allgemeinen Definition werden keine Aussagen über die Beschaffenheit des Systems getroffen. Es kann sich demnach sowohl um ein biologisches, als auch um ein technisches System handeln.

2.4.2 Mobile automatische Systeme

Aufbauend auf der Definition mobiler Systeme lassen sich mobile automatische Systeme wie folgt definieren:

> **Mobiles automatisches System:** Ein mobiles automatisches System ist ein technisches und/oder biologisches System, daß sich in den Räumen $\Re^1$, $\Re^2$ und $\Re^3$ für einen gewissen Zeitraum selbständig bewegen und Aufgaben erfüllen kann. Zusätzlich zu den Eigenschaften des mobilen Systems muß es die Umwelt über Sensoren ausreichend erfassen. Das Entscheidungssystem muß das System auf Basis dieser Daten im Raum selbständig bewegen und es zur Lösung seiner Aufgaben befähigen.

Damit besitzen die mobilen automatischen Systeme die zuvor dargestellten Eigenschaften und die Struktur von mobilen Systemen. Ihre Erweiterung besteht darin, daß sie sich selbständig bewegen können: Mobile automatische Systeme steuern sich demnach auf Basis einer Strategie selbst.

2.4.3 Mobile autonome Systeme

Autonomie

Um eine Definition für mobile autonome Systeme aufstellen zu können, ist zunächst die Klärung des Begriffs der Autonomie nötig. Er leitet sich aus dem griechischen autonomía

ab und bedeutet "selbständig" bzw. "unabhängig". Im technischen Bereich ist damit eine Unabhängigkeit von menschlichen Eingriffen gemeint[16]. Für eine Verwendung in diesem Kontext bietet Kegel in [Keg90, S. 4] die folgende Definition:

> **Autonomie:** Maschinelle Intelligenz erzeugt autonome Handhabung, wenn ein Rechner einen Manipulator in Echtzeit so regelt, daß Kontrollentscheidungen, die auf den Wahrnehmungen des Rechnersystems basieren, sich wiederum aus aufgaben- und umweltspezifischen Eigenschaften zusammensetzen und über die Rückführung verschiedener Sensoren während der Bearbeitung entstehen, vorhanden sind.

Diese sehr kompakte Definition von Autonomie fordert für die maschinelle Intelligenz einen Rechner, der die Aktoren ansteuert und seinerseits über eine Sensorik die Umwelt wahrnimmt. Dies entspricht der vorgestellen Struktur mobiler Systeme[17], nur daß das Entscheidungssystem hier von einem Rechner gebildet wird, der in Echtzeit arbeitet. Der Rechner trifft dabei die Kontrollentscheidungen auf Basis seiner Wahrnehmung, wobei die Entscheidungen sowohl aufgaben-, als auch umweltspezifisch sind. Es liegt also eine zielgerichtete Steuerung vor, die die Realisation auch an die vorhandene Umwelt angepaßt. Die Entscheidungen haben dabei Auswirkungen auf die Umwelt und/oder die Wahrnehmung derselben, was wiederum über die Sensorik erfaßt wird.

Ein System ist insofern dann autonom, wenn es nicht nur einem Plan folgen, sondern in bestimmten Situationen eigenständig, also auf der Grundlage von Zielen, Umwelt- und Statusinformationen, handeln kann.

Eine ähnliche Definition liefert Christaller, indem er sagt: "Unter Autonomie verstehen wir *Handlungsfreiheit* in dem Sinne, daß ein System sich nicht nur automatisch (...) verhält, sondern auch selbststeuernd. (...) Autonome intelligente Systeme zeichnen sich dadurch aus, daß sie aus einer Reihe von gleichwertigen Handlungsalternativen in einer gegebenen Situation eine als angemessen auswählen. Darüber hinaus können sie auf unterschiedliche Weise aus ihren Erfahrungen lernen, um ihr Verhalten zu verbessern." ([Chr98], kursiv im Original). Christaller verdeutlicht damit, daß Autonomie bedeutet, mehr als einen Plan zu verfolgen. Desweiteren erweitert er den Begriff zusätzlich um die Möglichkeit des Lernens.

Der Unterschied zwischen den Begriffen "automatisch" und "autonom" liegt demnach in der Anpassungsfähigkeit der Handlungsweise. Autonome Systeme verfügen über Alternativen, aus denen sie wählen können, während automatische Systeme ihr Verhalten gemäß einer Strategie regeln. In beiden Fällen arbeitet das System formal unabhängig von menschlichem Einfluß, wobei die Unabhängigkeit bei den autonomen Systemen weit stärker ausgeprägt ist.

[16] Vgl. hierzu beispielsweise die Beschreibung, die Wooldridge zur Autonomie von Intelligenten Agenten vornimmt: "Agents are able to act without the intervention of humans or other systems: they have control both over their own internal state, and over their behaviour." ([Woo99]).

[17] Vgl. Abbildung 2.10, Seite 26.

Mobile autonome Systeme

Damit lassen sich mobile autonome Systeme wie folgt allgemein definieren ([Wer03]):

> **Mobiles autonomes System:** Ein mobiles autonomes System ist ein technisches und/oder biologisches System, daß sich in den Räumen $\mathfrak{R}^1$, $\mathfrak{R}^2$ und $\mathfrak{R}^3$ für einen gewissen Zeitraum selbständig bewegen und Aufgaben erfüllen kann. Zusätzlich zu den Eigenschaften des mobilen Systems muß es die Umwelt über Sensoren ausreichend erfassen, Missionen planen und neuplanen können, das Manövermanagement beherrschen und Sondersituationen bewältigen. Die Entscheidungssysteme der verschiedenen Ebenen müssen zur Lösung dieser Teilaufgaben folgende Fähigkeiten besitzen: Adaption, Lernen, Fehlertoleranz, Kooperation sowie prädiktive Diagnose und Steuerung/Regelung.

Die mobilen autonomen Systeme zeichnen sich in diesem Sinne dadurch aus, daß sie ihre Aufgaben (Missionen) planen und vor allem umplanen können, um insbesondere Sondersituationen zu beherrschen. Dazu muß das Entscheidungssystem unterschiedliche Eigenschaften aufweisen. Neben der Steuerung/Regelung sind dies vor allem das Lernen neuer und die Adaption bestehender Strategien. Die Fehlertoleranz erfordert desweiteren eine Robustheit gegenüber den Eingabewerten, während unter der prädiktiven Diagnose eine vorausschauende Wahrnehmung zu verstehen ist.

2.4.4 Autonome Unterwasserfahrzeuge

Autonome Unterwasserfahrzeuge (AUV)[18] gehören zu den mobilen autonomen Systemen und sind für den Einsatz im Unterwasserbereich vorgesehen. Als mobile Unterwasserroboter agieren sie selbständig, um verschiedenste Aufgaben auszuführen. Dabei ist die Freiheit ihrer Handlungsweise so weit fortgeschritten, daß ihr Verhalten als autonom bezeichnet wird.

Derzeit arbeiten weltweit circa 45 Gruppen oder Firmen an der Konstruktion von autonomen Unterwasserfahrzeugen, wobei die USA mit circa 20 Fahrzeugen eine dominierende Rolle einnehmen ([Zim04, Val04, Wer99]). Im Folgenden sollen einige der derzeit existierenden Fahrzeuge mit ihren Kennwerten kurz vorgestellt werden:

- **Ocean Explorer** (Florida Atlantic University [Flo04])
  Ocean Explorer ist der Name einer Fahrzeugfamilie, die modular aufgebaut und für unterschiedliche Nutzlasten ausgelegt ist. Das Basisfahrzeug verzeichnet eine Länge von circa 2,5 m. Es existiert desweiteren eine Dockingstation für das AUV, die mit einer Fuzzy-Steuerung arbeitet.

[18] engl.: Autonomous Underwater Vehicles

- **Maridan 600** (Maridan ApS [Mar03])
  Dieses Fahrzeug wurde für Einsätze mit einer Dauer von bis zu 20 Stunden und einer maximalen Tauchtiefe von 1500 m entwickelt. Es erreicht eine Geschwindigkeit von 4 kn, ist modular aufgebaut und verfügt über eine Langzeitnavigation. Das Maridan 600 läßt sich mit diverser Nutzlast[19] bestücken.

- **Autosub** (University of Southampton [Sou03])
  Das Fahrzeug ist 7 m lang und besitzt einen Durchmesser von 0.9 m. Es verfügt über eine Carbon-Hülle und erreicht eine Einsatzdauer von bis zu 50 Stunden. Die tiefste seiner insgesamt über 270 absolvierten Missionen (Gesamtdauer: 750 h) betrug 1003 m. Die Genauigkeit des inertialen Navigationssystems (INS) wird mit 0.2 Prozent der zurückgelegten Strecke angegeben.

- **Hugin 3000** (Kongsberg Maritime [Kon04])
  Das Hugin 3000 hat eine Länge von 5.35 m und einen Durchmesser von 1.0 m. Es ist für Tiefen bis 3000 m und eine Einsatzdauer von 50 Stunden ausgelegt und verfügt über ein Aided INS. Die erreichbare Geschwindigkeit beträgt 4 kn. Das Fahrzeug besitzt desweiteren einen nichttieftauchenden, militärischen Bruder (HUGIN 1000).

- **DeepC** (ATLAS ELEKTRONIK GmbH [ATL04])
  Das *DeepC*-Fahrzeug weist eine geplante Fahrzeuglänge von 5,80 m auf. Es verfügt über ein inertiales Navigationssystem und setzt auf Brennstoffzellen als Energiespeicher. Die maximale Einsatzdauer soll 40 Stunden betragen. Als maximale Tauchtiefe sind 4000 m geplant.

Die einzelnen Fahrzeuge unterscheiden sich demnach deutlich, obwohl ihnen ein ähnlicher Aufbau zugrundeliegt.[20] Sie bestehen aus den Subsystemen Antrieb, Energie, Nutzlast, Maschinen und Steuerung. Ihre Verschiedenheit resultiert aus ihrem vorgesehenen Einsatzbereich, der wiederum vom Entwicklungsziel abhängt.

Als die treibenden Kräfte der Entwicklung treten nach Wernli ([Wer00])

- die internationale Forschungsgemeinschaft,
- die Industrie,
- sowie das Militär auf.

Dementsprechend variieren die Einsatzzwecke der geplanten Fahrzeuge erheblich: Industrievertreter des Öl- und Gassektors, aber auch der Telekommunikationsbranche nutzen die Fahrzeuge zur Unterwasserdatenerfassung, wie beispielsweise im Falle der Erkundung von Kabeltrassen. Das Militär ist vorrangig an der Auffindung von Mienen und an zukünftigen strategischen Operationen interessiert. Im Gegensatz hierzu entwickelt die Wissenschaft ihre Fahrzeuge für andere Forschungsrichtungen.

[19] engl.: payload
[20] In [SUF00, S. 2], [FOPS95] und [DBH00] finden sich weitere Fahrzeugbeschreibungen.

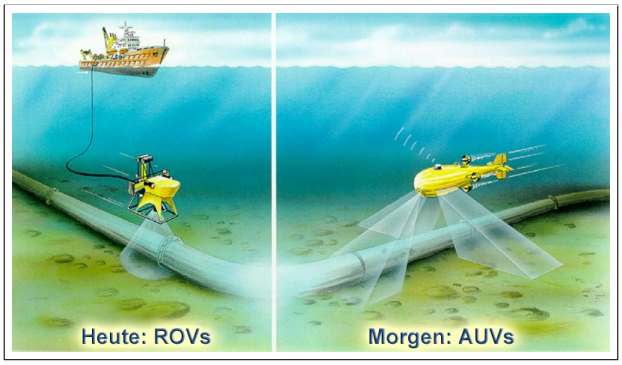

Abbildung 2.11: ROV vs. AUV (nach [ATL04])

So beschreibt Fletcher das Beispiel eines meeresbiologischen Einsatzes, in dem ein AUV zum Sammeln von Daten verwendet wurde ([Fle01]). Die Daten bezogen sich auf die Ausbreitung von Schwaden eines Farbstoffes im Meer und mit ihrer Hilfe konnte ein mathematisches Modell für die Verbreitung chemischer Substanzen im Wasser validiert werden.

Für die Industrie besteht eine wesentliche Motivation zur Entwicklung autonomer Unterwasserfahrzeuge in der durch ihren Einsatz erhofften Kostenreduktion. Die Fahrzeuge sollen in verschiedenen Aufgabenbereichen die bisherigen Lösungen, wie Remotely Operated Vehicles (ROVs)[21], ersetzen. Die Abbildung 2.11 zeigt die damit verbundene Herausforderung für die Entwicklung von Autonomen Unterwasserfahrzeugen.

Auf der linken Seite ist ein ROV abgebildet, wobei die Kabelverbindung zum Mutterschiff deutlich erkennbar ist. Diese Verbindung wird für die Steuerung durch einen Operator genutzt, kann aber gleichzeitig zur Versorgung des Fahrzeugs mit Energie verwendet werden. Ein Nachteil dieses Vorgehens sind jedoch die dazu nötigen, dicken und dadurch schweren Kabel, die die Navigierbarkeit des Fahrzeugs drastisch einschränken. Neuere ROVs führen deshalb ihre Energie mit sich, so daß die Steuerung über leichte Lichtwellenleiterkabel realisiert werden kann.

Ein Beispiel hierfür liefert der "Seebär", dessen Steuersoftware an der Technischen Universität Ilmenau mitentwickelt wurde ([Tec04]). Das kompakte, mit Batterien ausgestattete ROV ist über ein LWL-Kabel mit dem Steuerrechner des Operateurs verbunden.

Die rechte Seite der Abbildung zeigt den Einsatz eines AUV für die gleiche Aufgabe. Das Mutterschiff hat sich entfernt. Es wird lediglich zum Aussetzen und Einholen des Fahrzeugs benötigt, dazwischen ist es frei für andere Aufgaben.

Die Entwicklung kabelloser Fahrzeuge hat zur Folge, daß die Fahrzeugsteuerung nicht mehr durch einen Operator durchgeführt werden kann. Entsprechend ergibt sich die Forderung nach der Autonomie des Fahrzeugs. Diese stellt erhöhte Anforderungen an das System und seine Zuverlässigkeit: Ein Fehler eines AUV kann zum Verlust des Fahrzeugs und zu erheblichen finanziellen Einbußen führen, während ein kabelgebundenes Fahrzeug im Störungsfall vergleichsweise einfach zu bergen ist.

[21] dt.: Ferngesteuertes Fahrzeug

Change und Northcutt weisen in [CN01] darauf hin, daß im Jahr 2001 die meisten Fahrzeuge eher Unethered Underwater Vehicles (UUV)[22] als autonome Unterwasserfahrzeuge waren: Sie benötigten nach wie vor eine externe Überwachung zur Erfüllung ihrer Aufgaben. Ungeachtet dieses Einwandes werden in der Literatur die meisten Fahrzeuge als AUV bezeichnet. Im engeren Sinne sind als AUV unbemannte, autonom agierende Fahrzeuge zu bezeichnen, die unabhängig von einem Trägerschiff verschiedenste Aufgaben übernehmen können. Ihre Autonomie ermöglicht ihnen dabei völlig neue Einsatzgebiete (zum Beispiel unter Eis) sowie Kostenvorteile gegenüber dem Einsatz von ROVs. Ein Beispiel liefern Change und Northcutt in ihrem Artikel "Deep water AUV experiences" ([CN01]):

> Sie berichten von ersten Erfahrungen im Einsatz mit einem Hugin 3000. Das Fahrzeug wurde für die Seevermessung eingesetzt, wobei sich eine Zeitersparnis von circa 60 Prozent gegenüber einem konventionellen towfish, einer kabelgezogenen Sonde, ergab. Diesem Einsparpotential lagen nach Angaben der Autoren zwei Faktoren zugrunde: Erstens war die Einsatzgeschwindigkeit des AUV mit 4 kn höher als die des kabelgezogenen Gerätes, welches über eine maximale Geschwindigkeit von 2,5 kn verfügte. Zweitens konnten 180 ° Wenden mit einem wesentlich kleineren Radius gefahren werden. Dieses Manöver nahm bislang bis zu 50 Prozent der Einsatzzeit in Anspruch, da die Radien beim Betrieb des towfish sehr groß sein müssen, um ein Aufschlagen auf dem Meeresboden zu vermeiden.

In dem beschriebenen Szenario wurde das AUV weiterhin von einem Schiff aus überwacht. Zukünftig werden sich weitere Einsparpotentiale ergeben, wenn die Missionsdauer über einen langen Zeitraum ausgedehnt werden kann, wodurch das Überwachungsschiff zwischenzeitlich für andere Aufgaben bereitsteht.

Wernli identifiziert in [Wer99] die drei Hauptanforderungen an die Leistungsfähigkeit eines AUV:

- die Fähigkeit zur Mitnahme möglichst großer Energiemengen
- eine ausreichende Rechenkapazität
- eine akkurate Langzeitnavigation.

Der erstgenannte Punkt ermöglicht eine lange Einsatzdauer und verbessert insoweit die Rentabilität des Fahrzeugs. Während der bis in die siebziger Jahre des vorigen Jahrhunderts zurückreichenden Entwicklungsperiode autonomer Unterwasserfahrzeuge stellte lange Zeit die mangelhafte Rechenkapazität die begrenzende Ressource dar. Hier hat sich die Ausstattungssituation mittlerweile verbessert, wenngleich - angesichts zunehmend anspruchsvoller Aufgaben - auch die Anforderungen an die Rechenkapazität steigen. Die Notwendigkeit einer akkuraten Langzeitnavigation ergibt sich dabei aus der Forderung nach einer langen Missionsdauer. Unter Wasser ist kein Empfang von GPS-Signalen möglich, weshalb sich die Navigation nicht auf diese Daten stützen kann, sondern mit einer exakten Trägheitsnavigation arbeiten muß.[23]

[22] dt.: Ungebundenes Unterwasserfahrzeug

[23] Für ein Positionsupdate per GPS ist dann ein Auftauchen nötig, was bei Tiefwassermissionen energieintensiv ist.

2.5 Simulation autonomer Unterwasserfahrzeuge

2.5.1 Modellierungsansätze

Die in der Literatur beschriebenen Simulationen und Simulationsumgebungen lassen sich in mehrere Ansätze unterteilen, die sich eng an der mit der Modellierung verbundenen Fragestellung orientieren:

- **Hydrodynamische Simulationen[24]:**
  Diese Simulationen bestehen im wesentlichen aus einem Modell, das das dynamische Verhalten der Bewegung des AUV simuliert. Das Modell wird in der Regel auf Basis der Bewegungsgleichungen realisiert.[25] Teilweise werden diese ergänzt durch weitere Modelle für die Sensorik. Diese modellieren dann die für die Navigation benötigte Wahrnehmung der Fahrzeugposition und -orientierung. Zusätzlich beinhalten die Simulationen Regler für die Steuerung der Bewegung.

  Der Zweck dieser Art von Simulationen besteht im Entwurf der Steuerungen. Entsprechend genügt es ihnen, allein die Bewegung im Raum nachzubilden. Auf diese Weise läßt sich eine Stabilisierung und Steuerung des Fahrzeugverhaltens entwerfen. Diese Basissteuerung sorgt dafür, daß das Fahrzeug einfache Basismanöver (Geraden und Kreise) fahren kann.

  Beispielhaft sei an dieser Stelle auf die Arbeiten von Fryxell et al. ([FOPS95]) und Carreras et al. ([CRGU02]) verwiesen. Erstere beschäftigen sich mit einem Ansatz zur Steuerung der Bewegung. Er basiert auf einem "gain scheduled controller", der für verschiedene Arbeitspunkte zwischen den am linearisierten Modell ausgelegten linearen Reglern umschaltet. Carreras et al. entwerfen eine Regelung der nichtlinearen Dynamik auf Basis Neuronaler Netze.

  In der Literatur finden sich verschiedenste Ansätze für den Entwurf der Steuerungen. Neben den klassischen Reglern und Neuronalen Netzen ([SUF00]) werden Petri Netze ([AKT+97]), Finite State Machines ([SMP99, Lar99]) und das Fuzzy-Konzept ([SS00]) genutzt.

- **Verhaltenssimulationen:**
  Diese Gruppe von Simulationen modelliert nicht nur die Bewegungsdynamik, sondern auch die Umwelt des Fahrzeugs. Dabei beinhaltet die Umwelt Hindernisse, andere Fahrzeuge, den Seeboden und/oder Eigenschaften des Meeres. Chappell und Peng nennen in [CP99, S. 114] unter anderem ein Modell des Salzgehalts und der Strömungen des Meeres als mögliche Umgebungsdaten. Mit der Modellierung der Umwelt geht auch eine Modellierung der Sensorik zur Wahrnehmung derselben einher. Die Simulationen enthalten selbstverständlich auch Modelle der Fahrzeugsteuerung.

[24] Chappell und Peng führen diesen Begriff in [CP99, S. 113] ein. Sie klassifizieren damit eine Gruppe von bestimmten Simulationen.

[25] siehe Dynamikmodelle (Abschnitt 2.5.2, Seite 35)

Die Arbeiten auf der Basis dieses Ansatzes beschäftigen sich nicht mehr lediglich mit der Basisregelung des Fahrzeugverhaltens. Vielmehr versuchen sie, das Verhalten des Fahrzeugs in seiner Umwelt zu kontrollieren und ermöglichen somit eine Steuerung gemäß einer Mission.

Roeckel et al. stellen in [RRGL99] beispielsweise einen Regler vor, der auf Umgebungsdaten (Sonardaten) zurückgreift. Ein anderes Beispiel liefern Kuroda, Aramaki und Ura. In ihrer Simulationsumgebung Multi-Vehicle Simulator (MVS, [KAU96]) lassen sich mehrere Fahrzeuge in einer Simulation gemeinsam untersuchen (Schwarmverhalten, Verhalten bezüglich anderer dynamischer Objekte).

Typisch für diese Arbeiten ist eine hierarchische Gliederung der Regelungsstrategie bezüglich des Zeithorizonts. Ein Missionscontroller generiert Steuerkommandos entsprechend eines Missionsplanes, während ein Autopilot die Basisregelung auf Grundlage dieser Werte übernimmt.

- **Hardware-in-the-loop:**
  Diese spezielle Art der Simulation erzeugt für die reale Hardware eine virtuelle Umwelt. Das bedeutet, daß alle Umgebungsdaten durch die Simulation generiert werden. Da die Umwelt lediglich virtuell existiert, können die realen Sensoren keine Werte liefern. Aus diesem Grund müssen zusätzlich alle Sensorwerte von der Simulation bereitgestellt werden. Die Simulation umfaßt demnach die Umwelt und die Sensorik. Zur korrekten Koppelung an die reale Hardware müssen die Aktorwerte (zum Beispiel vom Antrieb) an die Simulation übertragen werden. Ein Merkmal dieser Art von Simulation ist die Forderung nach der Echtzeitfähigkeit, damit die reale Hardware unbeeinflußt arbeiten kann. Hierdurch ergibt sich jedoch eine der Realzeit entsprechende Simulationszeit, was gerade für eine lange Missionsdauer ungünstig ist.

  Ein Beispiel für den Einsatz von Hardware-in-the-loop beschreiben Brutzman, Kanayama und Zyda in [BKZ92]. Ihr System dient im "pseudo-mission"-Modus (hardware-in-the-loop) zur Visualisierung und Analyse des AUV-Verhaltens. Daneben ermöglicht es pre-mission-Vorbereitungen und post-mission-Auswertungen. Ein weiterer Beispielfall ist die Core Simulation Engine (CSE, [Duf96, CHCC99]). Sie stellt eine Verbindung zwischen Verhaltenssimulation und Hardware-in-the-loop dar, da sie neben dem Einbinden realer Hardware auch die modellgestützte Modellierung unterstützt.

  Kuroda, Aramaki und Ura ordnen diese Art der Simulation dem fortschreitenden Entwicklungsprozeß zu ([KAU96, S. 369]). Sie beschreiben den Testprozeß, der bei der reinen Computersimulation beginnt und beim Einsatz in der wirklichen Welt endet. Einen Zwischenschritt auf diesem Weg stellt der Test mit einer virtuellen Umwelt dar.

Abbildung 2.12 stellt die genannten Simulationsansätze bezüglich der Realisierung von System (AUV) und Umwelt gegenüber. Im Hinblick auf das System werden dabei zwei Kategorien unterschieden (real/virtuell[26]), im Hinblick auf die Umwelt drei (real/virtuell/nicht vorhanden). Die Vorteile des virtuellen Systems liegen in der Beschleunigung

[26]Mit virtuell ist in diesem Fall modelliert gemeint.

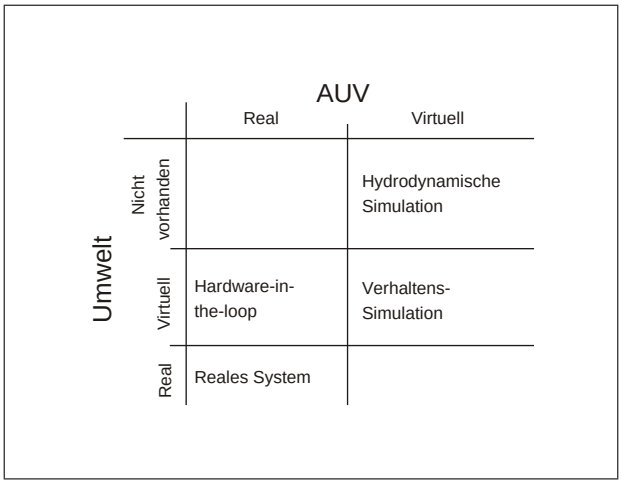

Abbildung 2.12: Simulation: reale und virtuelle Komponenten

und der Möglichkeit zur raschen Vornahme von Änderungen. Für eine virtuelle Umwelt sprechen die Beeinflußbarkeit, die Reproduzierbarkeit und die Beobachtbarkeit.

Vergleichbare Arbeiten, die sich mit der Modellierung von autonomen Unterwasserfahrzeugen beschäftigen, haben sich in der Regel auf die Nachbildung bzw. Untersuchung lediglich einzelner Teilaspekte konzentriert.

> Als Beispiel hierfür kann die Arbeit von Doucy gelten ([DBH00]): Seine MATLAB-Simulationsumgebung für ein AUV ist für die Untersuchung des Station-Keeping und Near Surface Manoeuvering konzipiert. Aspekte wie das zeitliche Verhalten der Software oder die Einbeziehung der Energie finden keinen Eingang in seine Untersuchungen. Trotz dieses Mangels können die Hauptmerkmale der Simulationsumgebung als beispielhaft gelten: Diese ist modular aufgebaut, erfordert keinerlei Programmierarbeit seitens des Benutzers und verfügt über eine 3D-Visualisierung des Verhaltens. Doucy setzt dabei auf das verbreitete Simulationsduo MATLAB/Simulink und PC-Hardware.

Neben den bislang vorgestellten Arbeiten sind zwei weitere erwähnenswert, welche die Optimierung im Zusammenhang mit AUVs beschreiben.

> McAllister et al. beschreiben in [MS+02] eine Testumgebung für die Optimierung von Entwurfsparametern für autonome Unterwasserfahrzeuge. Diese Parameter stammen aus den Bereichen Guidance and Control, Payload, Power, Machinery und Hydrodynamics. Zusätzlich definieren die Autoren Systemvariablen, wie die Länge des Fahrzeugs oder die Operationstiefe. Für die einzelnen Bereiche stellen sie Entwurfsregeln auf und implementieren diese. Auf der Basis dieses Modells beschreiben sie einen automatisierten Entwurf in der Konzeptphase. Er ist am Standardlayout eines AUV ausgerichtet, was bedeutet, daß die AUV-Struktur festgelegt ist. Mit dieser Arbeit läßt sich die optimale Auslegung der Komponenten bestimmen, welche auf den formulierten Entwurfsregeln basiert.

Rendas und Turcci stellen in [RT98] ein statistisches Analyseprogramm vor, das für eine gegebene Mission die Hauptmissionsparameter optimiert. Die Optimierung hat dabei das Ziel, den Energieverbrauch zu minimieren, die räumliche Erfassung zu maximieren und die Gefahr eines Fahrzeugverlusts zu begrenzen. Für die Optimierung selbst werden genetische Algorithmen eingesetzt, die für die vorgegebenen Bedingungen einen optimalen Missionsverlauf ermitteln.

2.5.2 Dynamikmodelle

Die Dynamikmodelle der oben vorgestellten Simulationen lassen sich auf zwei verschiedene Ansätze zurückführen:

- **Bewegungsgleichung[27]:**
  Der verbreiteteste Ansatz wird unter anderem von Fossen ([Fos94, Fos02]) und Brutzman ([Bru94]) detailliert vorgestellt. Er gründet auf einem physikalischen Kraft-Momenten-Ansatz. Dieser führt zu einem Gleichungssystem der Art:

  $$\dot{x} = A(t) \cdot x + B(t) \cdot u$$

  Die Matrizen $A(t)$ und $B(t)$ enthalten dabei fahrzeugspezifische Parameter, eine Beschreibung der Antriebselemente sowie Modelle der Umwelteffekte ([Fos94, S. 100ff.]). Dieses Verfahren ermöglicht die detaillierte Modellierung des hydrodynamischen Fahrzeugverhaltens. Für die Validierung des Modells sind jedoch Messungen mit Hilfe eines physikalischen Modells oder eines realen Fahrzeugs nötig. Aus diesen Messungen lassen sich die fahrzeugspezifischen Gleichungsparameter identifizieren. Einen Einblick in das hierbei verwendete Vorgehen bietet Goheen ([Goh91, S. 301]).

  Für die Simulation in MATLAB/Simulink hat Fossen die MATLAB GNC Toolbox implementiert, welche die Gleichung in Form fertiger, parametrisierbarer Blöcke bereithält ([Fos02, S. 533ff.]).

- **Neuronales Netz:**
  Sayyadi verwendet in [SUF00] einen anderen Ansatz. Seine Motivation ist eine unabhängige Identifikation der hydrodynamischen Parameter. Um sie zu erreichen, trainiert er ein Neuronales Netz mit gemessenen Kurven und benötigt aus diesem Grund ebenfalls eine reales Fahrzeug.

2.5.3 Simulationsauswertung

Ein Bestandteil aller Arbeiten ist die Aufbereitung und Darstellung der Simulationsdaten. Dabei greifen die Autoren vorwiegend auf zweidimensionale Diagramme zurück. Einige nutzen zur interpretationsfördernden Visualisierung auch 3D-Darstellungen. Beispiele hierfür finden sich in [BKZ92, KAU96, PSX98, DBH00, RBC01].

[27] engl.: equations of motion

Hackett und Nahon stellen in [HN98] diverse Instrumente zur anschaulichen Darstellung der Simulationswerte vor. Sie greifen dabei auf bekannte Anzeigen zurück, die in anderen Bereichen entwickelt wurden. So findet sich hier unter anderem eine Kompaßdarstellung und die aus der Luftfahrt entnommene Darstellung eines künstlichen Horizonts für die Lagedarstellung.

2.6 Zusammenfassung

Unter einem **System** wird im vorliegenden Zusammenhang eine gegenüber seiner Umwelt abgegrenzte Anordnung von Komponenten verstanden, die miteinander in Beziehung stehen. Für den technischen Entwurf einer solchen Anordnung ist dessen Spezifikation von entscheidender Bedeutung. Sie umfaßt eine Auflistung aller Subsysteme und Komponenten des Zielsystems, inklusive deren Kennwerte und der Festlegung, wie die Systembestandteile miteinander interagieren sollen.

Für den **Systementwurf** resultiert ein grundlegendes Problem aus der zunehmenden Komplexität der zu entwickelnden Systeme: Je komplexer der Aufbau eines Systems und die in ihm stattfindenden Interaktionsprozesse zwischen den Komponenten, desto zahlreicher die Auswirkungen einzelner Entwurfsentscheidungen auf das Gesamtsystemverhalten. Folge hiervon ist eine mit steigender Systemkomplexität zunehmende Unübersichtlichkeit des Entwurfs, die diesen fehler- und störungsanfällig macht.

Um diesem Problem zu begegnen, existieren verschiedene Entwurfsmethoden, die dem Entwickler ein erprobtes Ablaufschema für den Systementwurf zur Verfügung stellen. Grundlegend für die vorliegende Arbeit ist hierbei das **System Level Design**. Es bildet das System mit seinen dynamischen Prozessen in einem experimentierbaren Modell nach. Mit seiner Hilfe können Entwurfsentscheidungen objektiviert werden, indem ihre Auswirkungen auf das Gesamtsystem simulativ untersucht werden.

Eine der jüngsten Entwicklungen auf dem Gebiet der Entwurfsmethoden stellt das **Mission Level Design** dar, das von Schorcht für den Entwurf von Mobilkommunikationssystemen entwickelt wurde. Ziel ist es, das zu entwerfende System möglichst frühzeitig im Entwurfsprozeß umfassend testen zu können. Hierzu greift dieser Ansatz auf das aus dem System Level Design stammende Gesamtsystemmodell, als eine alle zentralen Systemaspekte umfassende Nachbildung der Spezifikation, zurück und führt zusätzlich die sogenannten Missionen ein. Diese stellen charakteristische Einsatzszenarien dar, mit denen umzugehen das zu entwerfende System im realen Einsatz in der Lage sein muß. Das Zusammenspiel von Gesamtsystemmodell und Missionen ermöglicht den umfassenden Test der Systemspezifikation im Hinblick auf seinen geplanten Einsatzzweck. Auf diese Weise wird nicht nur die Validierung der Spezifikation, sondern auch die Prognose der erwartbaren Leistungsparameter des Systems möglich.

Bei der **Simulation** handelt es sich um ein Verfahren zur Nachbildung eines Systems mit seinen dynamischen Prozessen. Um Erkenntnisse zu gewinnen, die in die Realität

übertragbar sind, nutzt sie ein experimentierbares Modell, das eine vereinfachte Nachbildung der untersuchungsrelevanten Systemeigenschaften darstellt. Die Simulation gliedert sich in die Phasen Modellbildung (Erstellung des Modells), Modellnutzung (Durchführung der Simulationsläufe) und Interpretation (Auswertung der Simulationsergebnisse). Für die Durchführung von Simulationen existieren diverse Simulationsprogramme und -sprachen. **MLDesigner** ist ein Entwurfswerkzeug der vierten Generation auf Basis einer Simulationsumgebung, das für den Systementwurf entwickelt wird. Über eine moderne Integrierte Entwicklungsumgebung ermöglicht es eine hierarchische, graphische und blockorientierte Modellierung sowie die Durchführung hybrider Simulationen.

Ein zentrales Ziel der vorliegenden Arbeit ist es, die Eignung des Mission Level Designs für den Entwurf mobiler automatischer Systeme im allgemeinen und am Beispiel des autonomen Unterwasserfahrzeugs *DeepC* im besonderen zu untersuchen. **Mobile automatische Systeme** sind dabei biologische oder technische Systeme, die sich selbständig bewegen und Aufgaben erfüllen können. Dazu besitzen sie eine bestimmte Struktur, die aus Sensorik, Aktorik, Entscheidungs- und Informationssystem besteht. Demgegenüber sind **mobile autonome Systeme** in ihrer Handlungsfreiheit noch weiter fortgeschritten. Sie verfügen über komplexere Entscheidungssysteme, die das Manövermanagement beherrschen und Sondersituationen bewältigen können.

Wie die Literatur zeigt, waren bisherige Unterwasserfahrzeuge in der Regel auf die Anwesenheit eines Mutterschiffs zur Steuerung oder Überwachung ihrer Arbeit angewiesen. Ziel des *DeepC*-Projektes ist abweichend hiervon die Entwicklung eines Fahrzeugs, das sich durch tatsächliche Autonomie auszeichnet, seine Aufgaben also vollkommen selbsttätig, unabhängig von einem Mutterschiff und der Überwachung durch einen Operator, ausführt.

Aus dem Ziel, die Entwicklung des *DeepC*-AUV simulativ zu unterstützten, ergibt sich die Frage nach hierfür geeigneten **Simulationsansätzen**. Die umfassende Auswertung der Literatur zeigt, daß sich die bislang für autonome Unterwasserfahrzeuge verwendeten Simulationsansätze in aller Regel auf einen Teilaspekt beschränken: So verwendet ein Teil der Arbeiten ein Modell des dynamischen Fahrzeugverhaltens und befaßt sich hierauf aufbauend einzig mit der Regelung desselben. Andere Aspekte des Systemverhaltens bleiben bei diesen sogenannten hydrodynamischen Simulationen unberücksichtigt. Die Verhaltenssimulationen beziehen zusätzlich die Systemumwelt ein und nehmen eine Simulation der für die Umweltwahrnehmung notwendigen Sensorik vor. Allen Arbeiten ist jedoch gemeinsam, daß sie keine ganzheitliche Betrachtung der zur Verfügung stehenden Ressourcen vornehmen. Gerade diese stellen aber einen für das Agieren jedes AUV zentralen und begrenzenden Aspekt dar.

Zusammenfassend kann festgehalten werden, daß die bislang vorliegenden Arbeiten zur Simulation autonomer Unterwasserfahrzeuge einzig dem Zweck der Entwicklung auf Komponentenebene dienen. Unter ihnen findet sich kein Beispiel für eine im Sinne des Mission Level Designs ganzheitliche Simulation. Für das Ziel dieser Arbeit, den Entwurf komplexer automatischer Systeme zu unterstützten, muß diese Herangehensweise als ungenügend bezeichnet werden: Für die theoretisch fundierte, praxisorientierte Simulation

des zu entwickelnden *DeepC*-AUV ist ein ganzheitliches Vorgehen unabdingbar. Dieses muß insbesondere die für den praktischen Einsatz zentrale Prognose der Ressourcenbelastung des Fahrzeugs umfassen.

3 Missionsbezogener modellgestützter Entwurf mobiler automatischer Systeme

Bei mobilen automatischen Systemen handelt es sich um komplexe technische Systeme, die aus den unterschiedlichsten Einzelkomponenten bestehen. Ihre Entwicklung wirft eine große Bandbreite von Teilproblemen aus verschiedenen Ingenieurdisziplinen und Forschungsrichtungen auf. Zu deren Lösung ist der Einsatz hochspezialisierter Entwickler erforderlich, was bei großen Projekten jedoch zu den in Abschnitt 2.1.1 angesprochenen Problemen führen kann. Um diesen Problemen zu begegnen, wurden verschiedene Entwurfsstrategien entwickelt. In dieser Arbeit wird untersucht, inwieweit sich der missionsbezogene modellgestützte Entwurf (Mission Level Design) für die Entwicklung mobiler automatischer Systeme empfiehlt.

Das folgende Kapitel widmet sich dabei den Anforderungen, die mobile automatische Systeme an den missionsbezogenen modellgestützten Entwurf stellen. Desweiteren wird dieses Systementwurfsverfahren in bezug auf seinen Entwurfsablauf, die Missionen, die Durchführung der Simulation und die organisatorische Einordnung untersucht.

3.1 Anforderungen mobiler automatischer Systeme

3.1.1 Modellierungsaspekte

Aufgabe des Gesamtsystemmodells ist es, das zu entwickelnde System mit seinen wichtigsten Funktionalitäten abzubilden. Wie die Betrachtungen zum Systementwurf auf Systemebene gezeigt haben, stellt es damit für den Entwurf eine ausführbare Umsetzung der Systemspezifikation bereit.

Als eine solche Umsetzung der Spezifikation muß das Gesamtsystemmodell alle wesentlichen Aspekte des Systems korrekt nachbilden. Das System Level Design führt dazu drei verschiedene Modelle für

- die **Funktion**,
- die **Architektur**

- und die **Umgebung** ein.

Sie entsprechen den Fragestellungen, die mit der Modellierung und der anschließenden Simulation verbunden sind. Für das Mission Level Design handelt es sich dabei um die Validierung des Systemdesigns sowie um die Formulierung von Prognosen bezüglich der Leistungsdaten des Systems. Das Gesamtsystemmodell wird demnach für den Test der Funktionalität und zur Dimensionierung der Architektur eingesetzt. Die Ausprägung der Modelle für die Funktion, die Architektur und die Umgebung werden dabei vom System vorgegeben.

Anpassungen für mobile automatische Systeme

Für mobile automatische Systeme ergeben sich an dieser Stelle spezifische Anpassungen.

1. Die ursprüngliche Idee eines generellen Mappings verschiedener Architekturen auf ein Funktionsmodell steht im Widerspruch zur Idee einer testweisen Umsetzung der Spezifikation. Eine klar voneinander abgegrenzte Modellierung von Architektur und Funktionalität fällt zudem schwer, wenn die Architektur Funktion übernimmt. Aus diesem Grund vereint der in dieser Arbeit verfolgte Ansatz das Funktions- und das Architekturmodell im *Systemmodell*. Es enthält beide Aspekte und repräsentiert damit diesen Teil der Spezifikation des Systems.

   Systemmodell: Das Systemmodell repräsentiert einen Teil der Spezifikation des Systems. Es enthält sowohl die architektonischen, als auch die funktionalen Systemaspekte.

2. Die Mobilität der Systeme erfordert ein Umgebungsmodell, das die Bewegung des Systems berücksichtigt. Je nach Einsatzzweck ist es dem System zusätzlich möglich, seine Umgebung aktiv zu verändern. Dieser Gesichtspunkt muß auch in die Modellierung übernommen werden. Dies kann verdeutlicht werden, indem die Rückwirkung des Systemmodells auf das Umgebungsmodell in die Darstellung integriert wird. Das bedeutet aber auch, daß das Umgebungsmodell quasi eine virtuelle *Umwelt* bereitstellen muß. Daneben stehen mobile automatische Systeme in einer Interaktion mit ihrer Umwelt. Hier ist das Umgebungsmodell nicht mehr durch z.B. Stimulibeschreibungen[28] modellierbar.[29]

   Um diese Anforderung an das Umgebungsmodell zu unterstreichen, wird es im Folgenden als *Umwelt* bezeichnet.

Abbildung 3.1 faßt die im Rahmen dieser Arbeit vorgenommenen Änderungen am von Schorcht formulierten Ansatz des Mission Level Designs zusammen. Die abgewandelten

[28] Vgl. Schorcht in [Sch00, S. 22].

[29] Vielmehr ist die Position des Systems im Raum entscheidend für seinen Abstand zu etwaigen Hindernissen, wobei diese Position jedoch vom System selbst bestimmt wird. Entsprechend muß die Umwelt die Hindernisse als solche definieren.

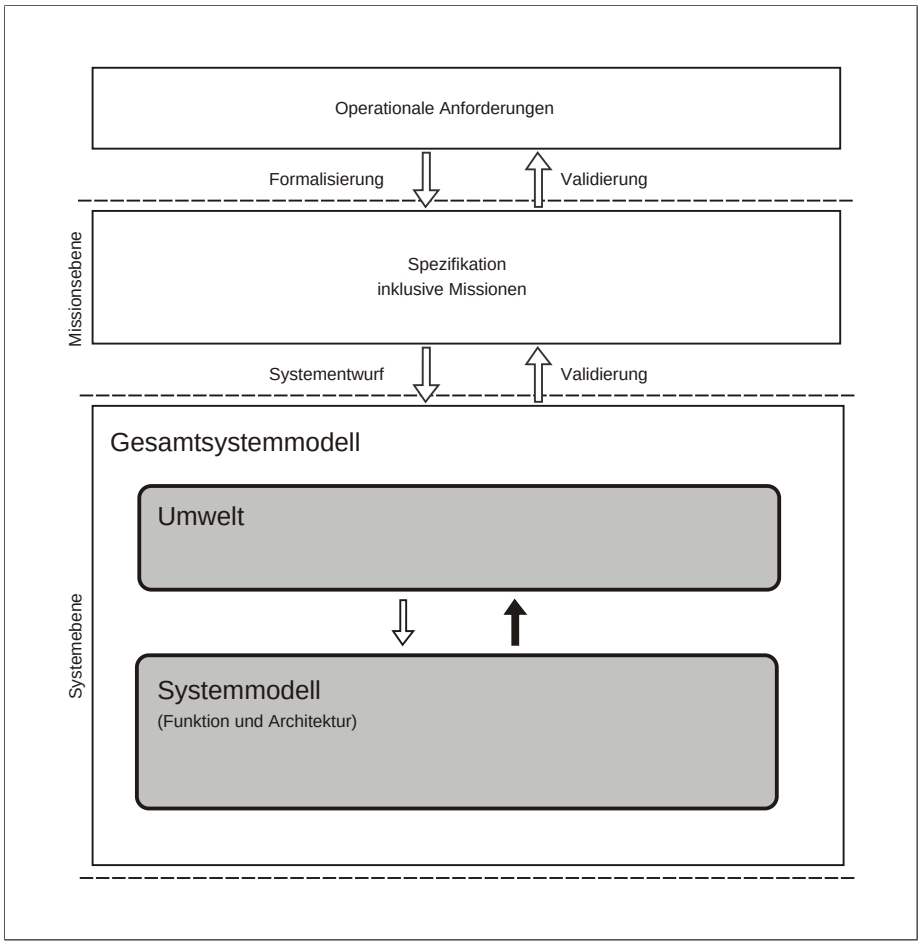

Abbildung 3.1: Missionsbezogener modellgestützter Entwurf für automatische Systeme

Gesichtspunkte werden dabei in die für das Mission Level Design gültige Darstellung integriert.[30] Im oberen Bereich sind die operationalen Anforderungen angeordnet. Sie werden durch die Missionen auf der Missionsebene spezifiziert und durch die Simulation validiert. Die Missionen selbst wirken auf die Systemebene, die vom Gesamtsystemmodell gebildet wird. Das Architektur- und das Funktionsmodell sind nun im Systemmodell vereinigt, welches zusammen mit der Umwelt (früher Umgebungsmodell) das Gesamtsystemmodell konstituiert. Die mögliche Rückwirkung des Systems auf die Umwelt wird durch einen zusätzlichen Pfeil zwischen Systemmodell und Umwelt repräsentiert.

Aspekte mobiler automatischer Systeme

Für das Systemmodell und die Umwelt lassen sich verschiedene Teilaspekte benennen, die für die Simulation mobiler automatischer Systeme in das Gesamtsystemmodell integriert werden müssen. Sie umfassen die Bereiche Funktion, Architektur und Umwelt.

- **Objekte und Prozesse** (Umwelt):
  Für mobile automatische Systeme sind einerseits andere, in der Umwelt existierende Objekte (Systeme) von Bedeutung. Andererseits müssen das System beeinflussende Prozesse in die Modellierung integriert werden. Beide gestalten die Umwelt, in welcher sich das System behaupten muß.

  Dabei ist der Begriff Objekt bewußt allgemein gewählt. In diese Kategorie fallen mithin nicht nur feste Hindernisse (z.B. Wände), sondern auch bewegte Systeme. So können beispielsweise auch andere automatische Systeme Teil der Umwelt sein.

- **Wahrnehmung** (Funktion):
  Die Einbeziehung der Wahrnehmung von Umweltobjekten und -prozessen in die Simulation entspricht der Modellierung der Sensorik. Dabei stellt die Sensorik einen Grundbestandteil mobiler automatischer Systeme dar, wie deren Definition aufgezeigt hat.

- **Dynamische Bewegung/Aktorik** (Funktion):
  Das System muß in der Simulation fähig sein, sich in seiner Umwelt zu bewegen. Damit eine physikalisch korrekte Simulation dieser Funktionalität erfolgen kann, sind entsprechende spezifische Randbedingungen zu realisieren.

  Verfügt das mobile automatische System über weitere relevante Aktorik, ist diese ebenfalls in die Modellierung mit einzubeziehen.

- **Rechenleistung** (Architektur & Funktion):
  Ein weiterer bedeutsamer Teilaspekt der Simulation ist die Rechenleistung. Dies ergibt sich aus der Tatsache, daß automatische Systeme in ihrer Struktur eine Verarbeitung der Sensorinformationen zur Aktoransteuerung vornehmen müssen.[31] Diese Umformung geschieht durch Algorithmen, die durch einen Rechner ausgeführt

[30] Vgl. Abbildung 2.5, Seite 12.
[31] Vgl. die Definition mobiler automatischer Systeme.

werden. Die Rechenleistung umfaßt sowohl architektonische, als auch funktionale Eigenschaften, weswegen sich an ihrem Beispiel gut die Durchdringung von Architektur und Funktionalität verdeutlichen läßt: Zur Architektur zählen beispielsweise die Rechner mit ihren Prozessoren, Bussen und ihrer Vernetzung (untereinander und zur Sensorik/Aktorik). Die funktionale Seite wird von der Software gebildet, die die Algorithmen in ausführbarer Form enthält. Darüber hinaus besitzt aber auch die Software architektonische Eigenschaften, nämlich durch die Strukturierung ihrer Einzelprozesse. Umgekehrt stellt auch der Rechner Funktionalität bereit, indem er die zeitliche Abarbeitung der Prozesse vornimmt.

- **Energie** (Architektur & Funktion):
  Da die Energie in der Regel eine begrenzte Ressource darstellt, bildet sie für viele mobile automatische Systeme einen zentralen Gesichtspunkt, der ebenfalls architektonische und funktionale Aspekte vereinigt. Die Architektur wird von der Hardware, also den Batterien, Leitungen usw. gebildet, während erst die Energie auf funktionaler Ebene die Arbeit der Komponenten bzw. des Systems ermöglicht. Sie übt damit einen Einfluß auf die Funktionalität aus. Der einfachste hierbei denkbare Fall tritt ein, wenn keine Energie mehr zur Verfügung steht und das System deshalb nicht mehr arbeiten kann, was in der Simulation, z.B. durch deren Beendigung, Berücksichtigung finden muß.

Die genannten Aspekte stellen eine Erweiterung des Modells gegenüber herkömmlichen Modellen dar. In ihnen spiegelt sich der Gesamtsystemcharakter der Modellierung wider.

Vergleich regelungstechnisches Modell und Gesamtsystemmodell

Anhand eines Vergleichs mit einem herkömmlichen regelungstechnischen Modell, wie es für auch für Komponentensimulationen Verwendung findet, läßt sich der Unterschied deutlich herausstellen: Das regelungstechnische Modell modelliert das Übertragungsverhalten des Systems, um Aussagen über die Güte und den Stellaufwand zu erhalten. Das Gesamtsystemmodell erweitert diesen Ansatz um weitere Aspekte, wie Energie und Rechenleistung. Desweiteren bildet es als simulierbare Spezifikation auch die Struktur des Systems exakt nach und ist mit dem Ziel verbunden, neben der Systemleistung auch eine Validierung des Systemdesigns zu erhalten.

Die verwendeten Testfälle bei beiden Modellen unterscheiden sich voreinander. Das regelungstechnische Modell verwendet einen Fahrplan in Form von Kurven für die Eingangsverläufe, während das Gesamtsystemmodell durch Missionen konfiguriert wird. Diese sind Nutzungsszenarien, in denen sich das System beweisen muß. Sie bestehen, wie Abschnitt 3.2.3 zeigt, für mobile automatische Systeme aus dem Missionsziel und den Konfigurationen für das Systemmodell und die Umwelt. Eine Mission ist damit eine freiere, erweiterte Form eines Fahrplans: Erweitert um zusätzliche Aspekte, freier durch die indirekte Vorgabe des Handelns.

	Inhalt	Testfälle	Zielaussagen
Regelungs-technisches Modell	Regelkreis (Streckenmodell und Regler)	Fahrplan	Performance (Güte, Stellaufwand)
Gesamt-system-modell	Systemmodell (mit Bewegung, Objekten/Prozessen, Sensorik, Rechen-leistung, Energie) und Umwelt	Missionen	Performance Validierung

Abbildung 3.2: Vergleich von regelungstechnischem Modell und Gesamtsystemmodell

Der Vergleich zeigt die Verwandtschaft von regelungstechnischem Modell und Gesamtsystemmodell. Abbildung 3.2 verdeutlicht dies, indem sie beide Modelle explizit gegenüberstellt. Deutlich treten jedoch auch die Unterschiede, die sich durch eine divergierende Zielstellung ergeben, hervor. Wie bereits die Definition des Modellbegriffs festgestellt hat[32], führt dies zu Unterschieden der Modelle in Art und Umfang (Aspekte).

Systemmodell und Virtueller Prototyp

In Abschnitt 2.1.2.4 zum System Level Design wurde erwähnt, daß Takala das Gesamtsystemmodell mit einem virtuellen Prototyp vergleicht. Für den missionsbezogenen modellgestützten Entwurf mobiler automatischer Systeme trifft diese Aussage so nicht mehr zu. Vielmehr ergibt sich durch die Unterteilung des Gesamtsystemmodells in Systemmodell und Umwelt eine andere Zuordnung. Durch die Aufwertung des Umgebungsmodells zur virtuellen Umwelt beschreibt dieser Modellteil nicht das System, sondern seine Umwelt. Dagegen umfaßt das Systemmodell eine Beschreibung des Systems in Form von dessen Funktion und Architektur. Demzufolge entspricht das Systemmodell einem virtuellen Prototyp, was die folgende Definition verdeutlicht.

> **Virtueller Prototyp:** Das Systemmodell stellt einen virtuellen Prototyp dar, weil es das System gemäß seiner Spezifikation testweise in der Simulation umsetzt und damit Untersuchungen am System im virtuellen Raum der Simulation gestattet.

Der missionsbezogene modellgestützte Entwurf mobiler autonomer Systeme untersucht demnach das Systemdesign anhand eines virtuellen Prototyps, der vom Systemmodell

[32] siehe Abschnitt 2.2.1.1, Seite 13

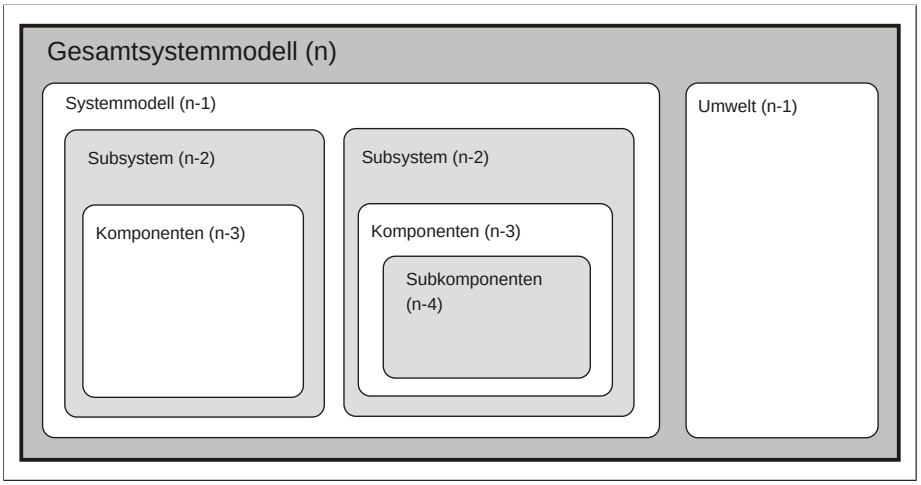

Abbildung 3.3: Modellierungsebenen

gebildet wird. Dazu werden dessen Verhalten und Leistungsdaten in einer virtuellen Umwelt anhand von Missionen analysiert. Diese Untersuchungen werden durch entsprechende Simulationen vorgenommen.

3.1.2 Hierarchische Modellgliederung

Auf der Grundlage der vorgestellten Struktur des Gesamtsystemmodells[33] kann seine hierarchische Gliederung gemäß dem in Abschnitt 2.2.1.2 vorgestellten Ansatz definiert werden.

Auf der obersten Ebene (n) sind das Systemmodell und die Umwelt angeordnet. Das Systemmodell entspricht dabei dem virtuellen Prototyp. Abbildung 3.3 illustriert diese Unterteilung und führt sie für die tieferen Ebenen fort.

Auf der Ebene $n-1$ werden der virtuelle Prototyp (Systemmodell) und die Umwelt in ihrer Ausprägung modelliert. Der virtuelle Prototyp setzt sich aus diversen Subsystemen zusammen (Ebene $n-2$). Hierzu zählen beispielsweise gemäß der Definition mobiler automatischer Systeme die Sensorik, die Verarbeitung und die Aktorik. Charakteristisch ist dabei, daß für das Systemmodell (den virtuellen Prototyp) die Modellstruktur mit der Struktur des zu entwerfenden Systems übereinstimmt. Die Subsysteme bestehen aus Komponenten (Ebene $n-3$) und diese ihrerseits aus Subkomponenten (Ebene $n-4$). Für das Gesamtsystemmodell ergibt sich somit eine Gliederung in mindestens vier Ebenen (ohne Subkomponenten).

[33] Vgl. Abbildung 3.1, Seite 41.

3.2 Missionsbezogener modellgestützter Entwurf

3.2.1 Einführung

Beim Mission Level Design handelt es sich um einen missionsbezogenen modellgestützten Entwurfsansatz für die Entwicklung komplexer Systeme. Aus diesem Grund wird im Folgenden synonym der Terminus missionsbezogener modellgestützter Entwurf verwendet, da er die Charakteristika dieser Entwurfsmethode besser herausstellt.

Die Besonderheit gegenüber anderen Ansätzen besteht gerade in der expliziten Einbeziehung von Nutzeranforderungen (*Missionen*, in der Spezifikation festgeschrieben) in den Systementwurf auf Systemebene. Dazu werden an einem *Gesamtsystemmodell* simulative Untersuchungen in Form von Missionen durchgeführt. Dieses Vorgehen entspricht einer Einschätzung des Entwurfs anhand eines virtuellen Prototyps in einer virtuellen Umwelt.

Der nicht neue Gedanke der Systemsimulation wird beim missionsbezogenen modellgestützten Entwurf auf ein abstraktes Gesamtsystemmodell mit den diversen Aspekten des zu entwerfenden Systems angewendet. Damit unterscheiden sich die Art und der Umfang der Modellierung wesentlich von Simulationen der Einzelsystemkomponenten.

Die bisherigen Ausführungen waren eher abstrakt und haben daher einige offene Fragen für den konkreten praktischen Einsatz aufgeworfen:

- Es wurde festgestellt, daß der missionsbezogene modellgestützte Entwurf gerade in den frühen Entwurfsphasen hilfreich ist. Wie verhält sich dies über den gesamten Entwurfsablauf hinweg und welche Hilfestellungen sind hierbei zu erwarten?
- Die Missionen wurden allgemein als Nutzungsszenarien definiert. Wie lassen sie sich in der Simulation nachbilden?
- Ergeben sich durch die Einführung der Missionen Auswirkungen auf die Simulationsdurchführung?
- Der missionsbezogene modellgestützte Entwurf ist eine Systementwurfsmethode. Wie gestaltet sich seine organisatorische Durchführung?

Diese Fragen bilden den Ausgangspunkt für die weiteren Betrachtungen in den folgenden Unterabschnitten.

3.2.2 Entwurfsablauf

In Abschnitt 2.1.2.5 zum Mission Level Design wurde darauf verwiesen, daß der Simulation innerhalb des Konzepts eine zentrale Rolle zukommt. Ihr übergeordnetes Ziel ist es, Fehler im Systementwurf frühzeitig finden und damit kostenintensive Designkorrekturen in späten Entwicklungsphasen zu vermeiden.

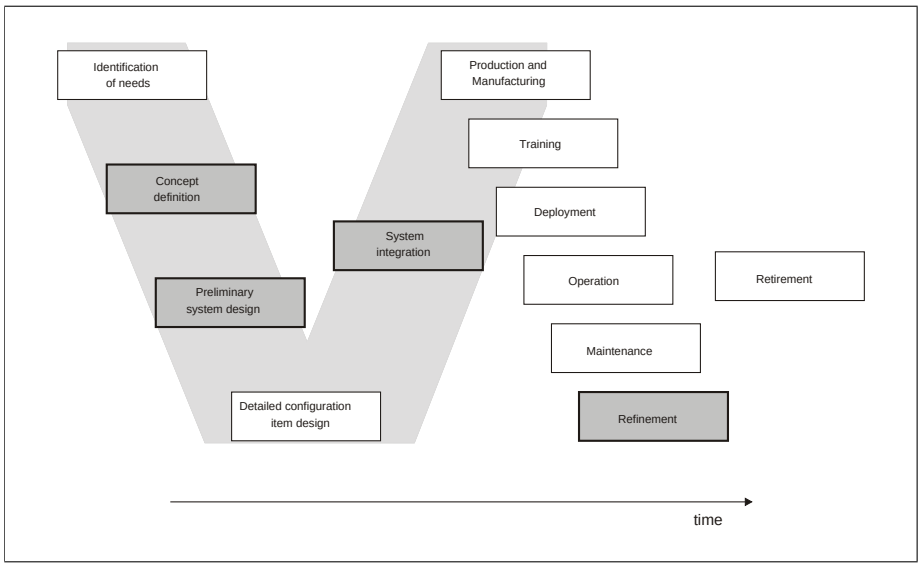

Abbildung 3.4: Lebenspanne eines Systems (nach [Bue00, S. 6])

Um die damit verbundenen Aufgaben näher spezifizieren zu können, ist es hilfreich, sich einen Überblick über den Lebenszyklus eines Systems zu verschaffen. Dieser stellt einen geeigneten Ausgangspunkt für die systematische Beschreibung der Aufgaben der Simulation dar.

Buede stellt in [Bue00, S. 6] den Lebenszyklus eines Systems im Zusammenhang mit dem System Level Design vor. Nach diesem durchlebt jedes System in seiner Entwicklung verschiedene Phasen, wobei die Simulation in verschiedenen Stadien sinnvolle Hilfestellungen leisten kann. Abbildung 3.4 beschreibt diesen Prozeß über die Zeit:

1. Ausgangspunkt eines Systems ist stets die Existenz eines spezifischen Bedürfnisses.
2. Dieser Bedarf führt in der nächsten Phase zur Definition eines Konzepts.
3. Es folgt der Entwurf eines Systemdesigns.
4. An ihn schließt sich die detaillierte Konfiguration der einzelnen Komponenten an.
5. Ist diese abgeschlossen, erfolgt die Systemintegration, also die Zusammenführung der Einzelkomponenten zu einem System.
6. In seiner sechsten Phase geht das entstandene System in die (Massen-)Herstellung.
7. An sie schließen sich die Phasen des Trainings, der Aufstellung, des Einsatzes und der Aufrechterhaltung an.
8. Auf der Basis des praktischen Einsatzes erfolgt gegebenenfalls eine Weiterentwicklung des Systems.
9. Den Abschluß des Lebenszyklus stellt die Außerbetriebnahme des Systems dar.

Die Entwicklung eines Systems im eigentlichen Sinne erstreckt sich von der Idee bis zur Produktion (hervorgehoben durch das V des V-Modells).

Buede beschreibt den Lebenszyklus als einen kontinuierlichen Prozeß von der Idee bis zur Außerbetriebnahme, wobei er in seiner graphischen Abbildung diejenigen Phasen markiert, in denen der Systementwurf auf Systemebene das System prägt (siehe Abbildung 3.4). Hierbei handelt es sich um

- die Projektierung,
- die Komponentenentwicklung,
- die Integration
- und die Weiterentwicklung des Systems.

Da das Mission Level Design mit dem beschriebenen Konzept des System Level Designs eng verwandt ist, kommt es in den oben festgestellten Phasen zum Einsatz. Die folgenden Unterabschnitte erläutern für jede dieser Phasen, welche Bedeutung der missionsbezogenen Simulation[34] im Rahmen des missionsbezogenen modellgestützten Entwurfsprozesses zukommt.

3.2.2.1 Projektierung

Während der Projektierung wird das Design des Systems erarbeitet. Diese Phase erstreckt sich von der ersten Idee über die Konzipierung einer Lösung bis zur exakten Definition auf der Systemebene. In ihrem Verlauf wird die Struktur des Systems entwickelt. Zu diesem Zweck wird es in Subsysteme partitioniert, die jeweils spezifische Aufgaben übernehmen. Für jedes Subsystem werden die benötigten Komponenten, ihre Schnittstellen und Protokolle definiert. Desweiteren findet die Auslegung der Komponenten statt, die von den Zielvorgaben des Entwurfs bestimmt wird. Dieser Schritt legt insofern die zukünftigen Leistungsdaten des Systems fest. Dabei handelt es sich insbesondere für komplexe Systeme um einen anspruchsvollen Vorgang, da eine Unterauslegung zur Nichterreichung der geforderten Leistungsdaten führt, während eine Überauslegung das System verteuert.

Der missionsbezogene modellgestützte Entwurf setzt auf unterstützende Simulationen auf Missionsebene, die bei der Formulierung der Spezifikation helfen. Die Simulation wird nicht mit abstrakten Daten durchgeführt, sondern erfolgt auf der Grundlage zukünftiger Nutzungsszenarien des Systems. Die Nutzungsszenarien sind dabei selbst als Teil der Systemspezifikation festgeschrieben und bleiben über die gesamte Entwicklungsdauer hinweg gültig. Sie dienen insofern als dauerhafte Referenz für die Anforderungen an das System. Die Simulation verwendet für die Untersuchungen ein Gesamtsystemmodell, welches einer testweisen Umsetzung der Spezifikation entspricht. Damit ermöglicht

[34] d.h. der Simulation auf Missionsebene

sie es, die Auswirkungen von Entwurfsentscheidungen auf das Systemverhalten zu untersuchen.

Durch die Verwendung von Bibliotheken fertiger und validierter Komponenten liefert die Simulation schnell greifbare Ergebnisse. Neue Komponenten müssen zunächst abstrakt beschrieben werden, worin eine der Hauptaufgaben dieser Entwicklungsphase besteht.

Zusammenfassend lassen sich die folgenden Punkte festhalten:

- Die Simulation unterstützt ein besseres Verständnis des Designs: Eine ausführbare Spezifikation ist im Vergleich zu einer Textversion wesentlich anschaulicher.
- Durch die testweise Umsetzung kann eine Validierung der Spezifikation vorgenommen werden. Sie ermöglicht die Prüfung der Spezifikation auf Vollständigkeit und Widerspruchsfreiheit.
- Die Leistungsdaten des Systems können unter Einhaltung der Spezifikation prognostiziert werden. Dies wirkt sich vor allem positiv auf den Prozeß der Komponentenauslegung aus.
- Da alle Informationen aufbereitet ausgegeben werden, unterstützt die Simulation auch die Erstellung einer Dokumentation.
- Durch die Simulation entsteht ein erhöhter Arbeitsaufwand.

Im Ergebnis der Projektierung liegt die Spezifikation des Systems vor, auf deren Basis die Umsetzung erfolgen kann.

3.2.2.2 Komponentenentwicklung

Im Zuge der Implementierung erfolgt die Umsetzung des Designs in ein konkretes System, das heißt, die zuvor beschriebenen Komponenten werden an dieser Stelle entwickelt. Aufgrund des hohen Komplexitätsgrades werden zu diesem Zweck verschiedene Entwicklergruppen parallel herangezogen, wobei sich jede von ihnen auf die Spezifikation stützt.

Diese Parallelität im Entwicklungsprozeß ist eine Ursache für mögliche Probleme. Werden zum Beispiel die Schnittstellen falsch oder nicht ausreichend exakt beschrieben, arbeiten die Komponenten später nicht reibungslos zusammen. Der missionsbezogene modellgestützte Entwurf bietet hier den Vorteil, daß die Spezifikation keinen abstrakten Charakter trägt, sondern ausführbar ist. Sie wurde durch die Simulation auf Missionsebene bereits überprüft und veranschaulicht den einzelnen Entwicklern die Funktion ihrer Komponenten im Kontext des Gesamtsystems.

In diesem Stadium kann durch die Anpassung des Gesamtsystemmodells an die Implementierungsdetails fortlaufend überprüft werden, inwieweit die Spezifikation von der Simulation eingehalten wird. Damit übernimmt die Simulation die Aufgabe einer globalen Kontrolle des Projekterfolgs. Es findet eine Gesamtüberprüfung im Hinblick auf die

Erreichung des Projektziels statt, wobei alle wesentlichen Aspekte einbezogen werden. Da im Verlauf des Projektfortschritts das Modell stetig konkretisiert wird, erlangt diese Kontrolle zudem eine zunehmende Realitätsnähe und Güte. Die Simulation erreicht auf diese Weise eine Zusammenführung der Informationen aller Entwicklergruppen. Für die Koordinierung der Entwicklung hat dies den Vorteil, daß die Auswirkungen von Entscheidungen auf Komponentenebene im Gesamtsystemkontext untersucht werden können.

Die Simulation auf Missionsebene fördert an dieser Stelle insofern:

- eine ständige Überprüfung des Projekts bezüglich der Erreichung der Entwurfsziele[35] und
- eine erweiterte Dokumentation, da die Details aller Komponenten einfließen.

Bei Abschluß dieser Phase sind die Komponenten des Systems durch die beteiligten Entwicklergruppen fertig entwickelt.[36]

3.2.2.3 Integration

Nach der Implementierung müssen alle Komponenten zu einem System integriert werden. Hierzu werden schrittweise Tests durchgeführt, die zunächst die einzelnen Komponenten, dann Gruppen von Komponenten und/oder Subsysteme und schließlich das gesamte System untersuchen.

Zu diesem Zeitpunkt zeigt sich die Güte der Prognosen der Simulationen auf Missionsebene. Treffen sie zu, so sind zwei Sachverhalte gewährleistet:

- Erstens kann davon ausgegangen werden, daß auch das reale System die angestrebten Leistungsdaten erreicht.
- Zweitens können Probleme im Zusammenspiel der Komponenten aufgrund von Fehlern in der Spezifikation weitgehend ausgeschlossen werden.

Hierfür gelten lediglich zwei Einschränkungen: Zum einen ist durch die verwendete Abstraktion eine geringfügige Diskrepanz zwischen realen Messungen und den Prognosen möglich. Zum anderen können Modellierungsfehler (Nichteinhaltung der Spezifikation) und Simulationsfehler nicht gänzlich ausgeschlossen werden.

Zu diesem Zeitpunkt zeigt sich, ob im Gesamtsystemmodell tatsächlich alle relevanten Aspekte abgebildet wurden. Aus nicht modellierten Aspekten resultieren unter Umständen Probleme. In diesem Fall ist kritisch zu hinterfragen, warum der betreffende Aspekt nicht im Modell berücksichtigt wurde.

[35] Insbesondere bezüglich der Auswirkungen von Entscheidungen auf Komponentenebene auf das Gesamtsystemverhalten

[36] Für ihre Arbeit nutzen sie dabei unter Umständen eigene Simulationen, in denen jeweils ihre Komponte(n) nachgebildet werden. Diese Komponentensimulationen dienen ausschließlich der Entwicklung der einzelnen Komponente(n). Die ihnen zugrunde liegenden Modelle sind auf andere Fragestellungen ausgerichtet und unterscheiden sich deshalb von den im Gesamtsystemmodell verwendeten.

War der missionsbezogene modellgestützt Entwurf erfolgreich, besitzt das reale System keine Entwurfsfehler und erfüllt die gestellten Benutzeranforderungen. In diesem Fall besteht seine Leistung darin, alle potentiellen Designfehler frühzeitig im Entwurfsprozeß erkannt und beseitigt zu haben.

3.2.2.4 Weiterentwicklung

Gleichzeitig zu den beschriebenen Hilfestellungen stellt der missionsbezogene modellgestützte Entwurf die Grundlage für eine Weiterentwicklung des Systems bereit. Eine solche wird immer dann notwendig, wenn Anpassungen an spezielle Kundenwünsche vorgenommen oder veränderte Anforderungen berücksichtigt werden müssen. Letztere können beispielsweise in einer erweiterten Funktionalität bestehen.

In diesem Fall steht mit dem Gesamtsystemmodell ein optimales Versuchsobjekt bereit. Die Güte dieses Modells wurde durch die Tests mit dem realen System geprüft und kann somit verläßlich eingeschätzt werden. Die veränderten Anforderungen lassen sich durch variierte Missionen definieren. Neue Funktionalitäten können dem Modell hinzugefügt werden, wofür das Gesamtsystemmodell als Ausgangsbasis wiederverwendet werden kann. Hierbei muß nicht zwingend die letzte Version des Modells aus der Integrationsphase verwendet werden, vielmehr ist es möglich, auch auf frühere Versionen zurückzugreifen. In jedem Fall unterstützt das Gesamtsystemmodell die Entwickler dabei, die nötigen Änderungen abzuschätzen.

3.2.3 Missionen

Im Zusammenhang mit dem Mission Level Design ist immer wieder von Missionen die Rede. Bisher wurde gezeigt, daß sie ein elementarer Bestandteil dieses Konzepts sind und die eigentliche Erweiterung gegenüber dem System Level Design bilden.

In Abschnitt 2.1.2.5 wurden Missionen als festgelegte Einsatzfälle definiert, deren Zweck es ist, alle Situationen und Aktionen zu beschreiben, mit denen das System in definierter Art und Weise umzugehen in der Lage sein muß. Damit definieren sie Testbedingungen für die Auslegung des Systemdesigns, wobei diese Tests an einem virtuellen Prototyp simulativ vorgenommen werden.

Offen ist bislang die Frage: Wie sehen die Missionen für die Simulation konkret aus? Schorcht beantwortet sie allgemein durch den Hinweis, daß Missionen das Treibermodell des System Level Designs ersetzen. Stellen Missionen demnach Eingangsdaten für das Gesamtsystemmodell dar?

3.2.3.1 Modellierung von Missionen

Klassischerweise werden Systeme im Hinblick auf ihr Verhalten bezüglich bestimmter Eingangsdaten untersucht.

> Ein typisches Beispiel stellt hier ein Regelkreis dar. Für ihn ist das Übertragungverhalten von entscheidendem Interesse. Dabei wird untersucht, wie Änderungen am Eingang am Ausgang realisiert werden.

Mobile automatische Systeme nehmen ihre Eingangsdaten aus ihrer Umwelt auf. Diese stellen aber in der Simulation keine konstanten Größen dar, da das System permanent mit seiner Umwelt interagiert und sich auf der Basis der dabei gesammelten Daten selbst steuert. Die Eingangsdaten hängen also vom Systemverhalten selbst ab. Insofern kann festgehalten werden, daß Missionen keine Eingangsdaten des Systems darstellen, was sich anhand des folgenden Falls nachvollziehen läßt:

> Als Beispiel soll ein mobiler autonomer Roboter dienen, der in einem Hindernisparcours von einer Startposition aus eine Zielposition erreichen soll. Für die Wahrnehmung der Hindernisse besitzt er eine geeignete Sensorik, für die Bewegung eine entsprechende Aktorik.
> Seine Eingangsdaten bestehen aus den relativ zu seiner eigenen Position wahrgenommenen Positionen der Hindernisse. Seine Position steuert der Roboter dabei so, daß er sein Missionsziel erreicht. Damit hängen die wahrgenommenen Hindernispositionen von der Strategie zur Erreichung der Zielposition ab.

Die Daten, die in der Simulation eine Mission nachbilden, müssen also eine Abstraktionsebene höher angesiedelt sein und sowohl die Situation, als auch das Missionsziel beschreiben. Zudem müssen sie sich im Gesamtsystemmodell wiederfinden. Da das Gesamtsystemmodell auf der obersten Ebene aus dem Systemmodell und der Umwelt besteht, lassen sich hier die Konfiguration des Systems und die Konfiguration der Umwelt unterscheiden. Das dritte Element der Definition einer Mission stellt das Missionsziel dar:

- **Missionsziel:**
  Das Missionsziel definiert die durchzuführende Tätigkeit oder ein zu erreichendes Ziel für das mobile automatische System.
- **Konfiguration des Systems:**
  Die Konfiguration des Systems beschreibt, in welchem Zustand sich das System befindet. Dabei kann es sich sowohl um den Anfangszustand des Systems handeln, als auch um den Zustand über der Zeit.
- **Konfiguration der Umwelt:**
  Gleiches gilt für die Konfiguration der Umwelt. Sie gibt den Rahmen für die Erreichung des Missionszieles vor, indem sie die Umweltbedingungen festsetzt.

Für das vorangegangene Beispiel bedeutet dies, daß die Missionsdaten sich wie folgt zusammensetzen:

Die Startposition stellt den Anfangszustand des Systems dar, während der Zielpunkt das Missionsziel beschreibt. Die globalen Positionen der Hindernisse sind als Teil der Umwelt Bestandteil der Mission. Sie beschreiben die Szenerie, in der das System sein Ziel erreichen will.
Als Beispiel für die Beschreibung eines Zustands über die Zeit kann der Status eines Sensors dienen. Ein exemplarisches Entwurfsziel könnte lauten, das Ziel trotz eines während der Mission auftretenden Sensorenausfalls zu erreichen. In diesem Fall würde der Zustand des Systems über die Zeit - und nicht nur zum Anfangszeitpunkt - beschrieben werden.

Die bislang angestellten Betrachtungen haben gezeigt, welche Daten für die Beschreibung von Missionen in der Simulation im Gesamtsystemmodell vorkommen. Dies beantwortet jedoch noch nicht die Frage, wie und an welchen Stellen des Gesamtsystemmodells die Missionsdaten auftreten.

Wie im Abschnitt 2.2 beschrieben, eignet sich für die Modellierung eine blockorientierte Herangehensweise. In ihr entspricht ein Block einer logischen Struktur, also beispielsweise einem System, einem Subsystem oder einer Komponente. Jeder Block besitzt Eingänge, Ausgänge und Parameter, über die er konfiguriert wird. Die Konfiguration entspricht also einer Beschreibung des Blocks.

Das Gesamtsystemmodell des Roboters enthält somit einen Block, der diesen beschreibt. Dieser Block besitzt einen Parameter, der ein variables Missionsziel in Form eines Punktes aufnimmt. Er sichert die Variabilität, die unabdingbar ist, damit das Modell verschiedene Missionen nachbilden kann.
Auch dynamische Zustandsänderungen (beispielsweise ein Sensorausfall zu Zeitpunkt t_1) lassen sich auf diese Weise modellieren. Dazu kann der Block beispielsweise einen Parameter nach außen führen, der den Zeitpunkt aufnimmt, zu dem ein Sensor ausfällt.

Eine Mission zu beschreiben meint in diesem Zusammenhang also, das Modell zu parametrisieren. Oder andersherum ausgedrückt: Missionen lassen sich durch eine entsprechende Ausprägung aller zu ihrer Beschreibung dienenden Parameter nachbilden. Mit der folgenden Definition wird die soeben entwickelte Erkenntnis noch einmal zusammengefaßt:

Mission: Aus Sicht des Entwurfs mittels Simulation werden Missionen für mobile automatische Systeme durch eine Ausprägung von Modellparametern nachgebildet. Diese Parameter beschreiben das Missionsziel, die Systemzustände und die Konfiguration der Umwelt. Aus ihrer Summe resultiert jeweils ein Nutzungsszenario zum Test des Systementwurfs.

Die Tatsache, daß die gesamte Beschreibung einer Mission über die Parameter geschieht, läßt sich an einem Beispiel verdeutlichen:

Als Beispiel soll wiederum der mobile autonome Roboter dienen. Er soll sich von A nach B bewegen, wobei sich in seinem Weg die Hindernisse H_1 bis H_n befinden. Zum Zeitpunkt t_1 fällt einer seiner Sensoren aus. Damit sieht eine Nachbildung der Mission mittels der Missionsparameter wie folgt aus:

Parameter	Bedeutung	Mission 1
A	Startposition	$(0.0, 0.0)$
B	Ziel	$(10.0, 20.0)$
H_1	Hindernis 1	$(5.1, 6.3, 15.0, 4.2, 0.0)$
...	...	...
H_n	Hindernis n	$(2.4, 3.0, 10.0, 0.2, 45.0)$
t_1	Sensorausfall	25.0

Die Hindernisse bilden die Konfiguration der Umwelt, der Zielpunkt stellt das Missionsziel dar. Die Konfiguration des Systems besteht in diesem Fall aus der Startposition und dem Zeitpunkt des Sensorausfalls.

Das Beispiel verdeutlicht, daß für die Simulation jede Mission über eine Ausprägung von Parametern darstellt werden kann. Sie gibt für bestimmte Parameter des Modells konkrete Werte vor, die so zu wählen sind, daß die Simulation ein definiertes Nutzungsszenario nachbildet.

3.2.3.2 Typen von Modellparametern

Aus der Feststellung, daß Missionen mittels der Parameter nachgebildet werden, ergibt sich im Umkehrschluß die Frage: Über welche Arten von Parametern verfügt das Gesamtsystemmodell?

Eine formale Untersuchung der Parameter zeigt, daß sie sich in vier Gruppen unterteilen lassen. Jede der Gruppen nimmt dabei Informationen aus einem spezifischen Gebiet auf, was an dieser Stelle als Unterscheidungsmerkmal genutzt wird. Die folgende Auflistung stellt sie kurz vor:

- **Konstanten:**
  Eine erste Gruppe wird von den Konstanten gebildet. Bei ihnen handelt es sich um feste Werte, die unabhängig vom Systementwurf sind.

  Dies trifft zum Beispiel auf Naturkonstanten und andere unveränderlich gegebene Größen zu. Diese Gruppe umfaßt damit auch die Parameter, die ausschließlich für die Durchführung der Simulation benötigt werden, wie beispielsweise der Name und/oder der Pfad einer Ausgabedatei.

- **Systemparameter:**
  Desweiteren existieren Parameter, die Kenngrößen des zu entwerfenden Systems darstellen. Sie beschreiben als Systemparameter die Dimensionierung des Systems. Diese charakteristischen Größen bilden zusammen mit der Struktur des Systemmodells eine Beschreibung des Systems (den virtuellen Prototyp).

Entsprechend handelt es sich bei ihnen um Größen wie die Höchstgeschwindigkeit oder die Datenrate einer Verbindung.

- **Missionsparameter:**
  Drittens werden Parameter unterschieden, die zur Beschreibung von Missionen dienen. Sie werden im Rahmen dieser Arbeit als Missionsparameter bezeichnet.

  Da sie ausschließlich der Charakterisierung verschiedener Einsatzfälle dienen, beinhalten diese Parameter vor allem Anfangswerte oder -konfigurationen. Missionsparameter treten dabei sowohl im System, als auch in der Umwelt auf. Sie beschreiben beispielsweise die Positionen von Hindernissen oder die Statusänderung eines Sensors. Desweiteren gehören zu dieser Gruppe diejenigen Parameter, die dem System das Missionsziel (zum Beispiel einen Zielpunkt) vorgeben.

- **Entwurfsparameter:**
  Entwurfsparameter bilden die allgemeinen Zielvorgaben des Systementwurfs ab. Es handelt sich bei ihnen also um diejenigen Größen, die vom System erreicht werden sollen (Entwurfsziele). Sie werden nach Abschluß der Simulation durch die Auswertung der Simulationsergebnisse überprüft.

  Ein Beispiel hierfür stellt die maximal erreichbare Einsatzdauer dar. Entwurfsparameter müssen nicht notwendigerweise im Modell vorhanden sein.

Als Fazit dieser Untersuchung läßt sich feststellen, daß die Missionsparameter nur ein Teil der Gesamtheit der Modellparameter sind.

3.2.4 Konzept für die Durchführung der Simulation

3.2.4.1 Problemstellung

Nachdem der vorausgegangene Unterabschnitt gezeigt hat, daß Missionen sich mittels der Missionsparameter des Gesamtsystemmodells beschreiben lassen, wird nachfolgend untersucht, inwieweit dies den Simulationsablauf beeinflußt.

Im Rahmen der Simulationsgrundlagen wurden im Abschnitt 2.2.1 die Phasen der Simulation vorgestellt. Die einzelnen Schritte, denen die Simulation dabei folgt, sind die Modellierung, die Modellnutzung und die Interpretation/Auswertung.

Für die Simulation im Rahmen des missionsbezogenen modellgestützten Entwurfs ergeben sich in diesem Zusammenhang zwei Probleme:

- Erstens besteht ein wesentlicher Unterschied im Vergleich zu einer herkömmlichen Simulation darin, daß ein variierendes Gesamtsystemmodell mit unterschiedlichen, festgeschriebenen Missionen parametrisiert werden muß. Für eine große Anzahl von Missionen ist es nicht praktikabel, das Modell vor jeder Simulation manuell zu parametrisieren. Dieses Vorgehen wäre zu arbeitsaufwendig und zudem stark fehleranfällig.

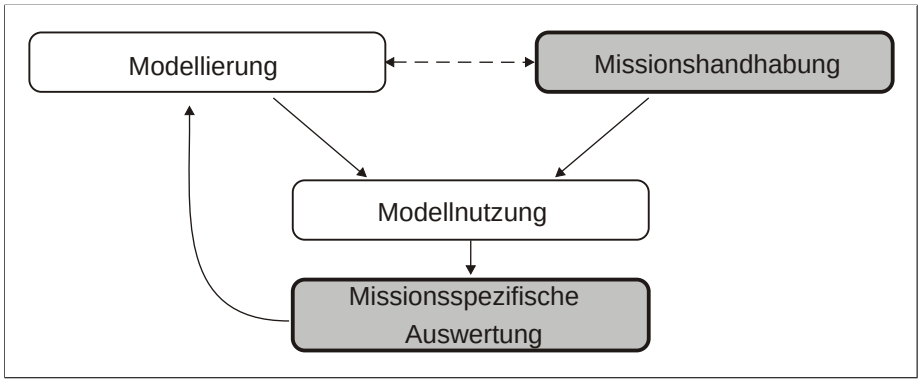

Abbildung 3.5: Phasen des missionsbezogenen modellgestützten Entwurfs

- Zweitens führt die Simulation diverser Missionen mittels ein- und desselben Modells dazu, daß für alle nötigen Ausgabedaten entsprechende Ausgabemöglichkeiten im Modell vorgesehen werden müssen. Da jedoch nicht alle Ausgaben für die Auswertung jeder Mission von Bedeutung sind, ist eine Selektion der Daten notwendig, die in direktem Zusammenhang mit den Missionen steht. Die Konfiguration der Selektion - aber auch die der Visualisierung - ist dabei missionsabhängig (missionsbedingt), aber konstant über die verschiedenen Simulationsläufe einer Mission.

3.2.4.2 Konzept

Zur Lösung der beschriebenen Probleme wird ein erweitertes, an den missionsbezogenen modellgestützten Entwurf angepaßtes Konzept für die Durchführung der Simulation vorgestellt. Dieses besitzt eine neue Phase der Missionshandhabung in der zentral die Missionserstellung und -verwaltung erfolgt. Das Modell wird auf diese Weise vor jeder Simulation automatisch mit einer Mission konfiguriert, wodurch sich die Arbeit mit den Missionen erheblich beschleunigen und die Fehleranfälligkeit minimieren läßt. Zudem entspricht eine zentrale Handhabung dem Mission Level Design-Prinzip, nach welchem die Missionen einen Teil der zentral festzuschreibenden Spezifikation darstellen. Desweiteren fordert das Konzept für die Auswertung eine erweiterte Funktionalität, damit diese den spezifischen Anforderungen des Umgangs mit Missionen gerecht wird.

Damit besteht die Simulation, wie in Abbildung 3.5 ersichtlich, nunmehr aus vier Phasen, nämlich - wie in herkömmlichen Simulationen üblich - aus der Modellierung, der Modellnutzung und der Auswertung, die um die die neue Phase der Missionshandhabung ergänzt werden. Die Auswertung erfährt eine Erweiterung, indem sie *missionsspezifisch* erfolgt.

Eine umfassende Charakterisierung der einzelnen Phasen und ihrer Zusammenhänge liefert die folgende Auflistung:

- **Modellierung:**
  Den ersten Punkt der Simulation bildet weiterhin die Modellierung, die die Erstellung des Gesamtsystemmodells mit allen wichtigen Aspekten des Systems beinhaltet und eine Nachbildung der aktuellen Spezifikation des Systems und der Umwelt, in der es agiert, vornimmt.

- **Missionshandhabung:**
  Die Missionen werden - als Teil der Spezifikation - in der Missionshandhabung einmalig festgelegt, später müssen sie lediglich verwaltet werden. Die Festlegung umfaßt dabei (abstrakt formuliert) die Schaffung verschiedener Ausprägungen für die existierenden Missionsparameter oder die Zuordung der Missionsparameter zu allgemein definierten Missionen.

  Im Hinblick auf den missionsbezogenen modellgestützten Entwurf besteht die Aufgabe in der Entwicklung einer einfachen und handhabbaren Möglichkeit zur Arbeit mit verschiedenen Missionen.

- **Modellnutzung:**
  In der Modellnutzung werden die Missionen auf das Modell angewendet, wodurch konfigurierte Modelle entstehen. Deren Simulation erfolgt rechentechnisch und erzeugt eine Vielzahl von Simulationsergebnissen.

  Dabei wird der virtuelle Prototyp in seiner virtuellen Umgebung mit den einzelnen Nutzungsszenarien (Missionen) konfrontiert, wobei sein Verhalten und seine Zustände aufgezeichnet werden.

- **Missionsspezifische Auswertung:**
  Im Rahmen der missionsspezifischen Auswertung werden die Simulationsergebnisse untersucht. Die große Menge anfallender Informationen muß dazu anschaulich aufbereitet werden, was neben der Selektion eine vielseitige Datenaufbereitung erfordert. Desweiteren ist eine Visualisierung nötig, die unterschiedliche Daten jeweils optimal darstellt. Für die Auswertung werden alle relevanten Ergebnisse missionsweise analysiert und daraus eine Gesamteinschätzung des Systemdesigns bezüglich der Nutzeranforderungen erarbeitet. Auf dieser Analyse aufbauend kann im Anschluß der Entwurf fortgeführt und optimiert werden.

  Während der Projektierungsphase[37] bedeutet dies beispielsweise, daß eine Designänderung oder eine andere Auslegung einer Ressource vorgenommen wird, was eine anschließende erneute simulative Überprüfung erfordert. In der Phase der Komponentenentwicklung[38] gibt das Ergebnis der Auswertung dagegen darüber Auskunft, ob die Nutzeranforderungen erfüllt werden können. Ist dies der Fall, kann die Implementierung unverändert fortgeführt werden, während anderenfalls entsprechende Gegenmaßnahmen durchzuführen sind. In einer solchen Situation unterstützt die Simulation die Entwickler dabei, die nötigen Modifikationen festzulegen, da sich

[37] Vgl. Abschnitt 3.2.2.1, Seite 48.
[38] Vgl. Abschnitt 3.2.2.2, Seite 49.

die Auswirkungen der Änderungen zuvor am Modell aus Gesamtsystemsicht testen lassen.

Das hier vorgestellte, wesentlich erweiterte Konzept trägt den Besonderheiten von Simulationen im Rahmen des missionsbezogenen modellgestützten Entwurfs Rechnung. Es ergänzt die herkömmlichen Phasen einer Simulation um die neue Phase der Missionshandhabung und um erweiterte Auswertungsmöglichkeiten für Missionen.

Im Gegensatz zur bisher geltenden Meinung[39] zeigt sich somit, daß der Aufwand der Treiber- und Bewertungsmodellerstellung durch das Mission Level Design nicht entfällt. Vielmehr verlagert er sich in Richtung der Missionshandhabung und -auswertung. Erst wenn diese beiden Aspekte voll durch entsprechende Werkzeuge unterstützt werden, reduziert sich der Aufwand im Vergleich zum System Level Design.

3.2.5 Organisatorische Durchführung

Der missionsbezogene modellgestützte Entwurf ist ein Entwurfskonzept für komplexe Systeme. Entsprechend obliegt seine Durchführung dem Projektmanagement. Letzteres muß ihn als Entwurfsmethode anwenden.

Dessen ungeachtet setzt dieses Entwurfsverfahren, wie bisher gezeigt wurde, auf die Simulation zur Unterstützung des Entwurfsvorganges. Dafür müssen das Gesamtsystemmodell und die Missionen erstellt, die Simulationen durchgeführt und ausgewertet werden. Diese Aufgaben verlangen - gerade im Bereich der Modellierung - nach Spezialwissen, das nicht beim Projektmanagement vorhanden ist. Aus diesem Grund sollte ein MLD-Team mit entsprechender Kompetenz eingerichtet werden.

Dieses führt die simulativen Untersuchungen auf Missionsebene durch und präsentiert dem Projektmanagement die Ergebnisse. Es pflegt sich ergebende Änderungen in das Gesamtsystemmodell ein und stellt die neuen Resultate vor. Auf diese Weise schreitet der Entwurf in der Projektierung iterativ voran.

In der Komponentenentwicklung im Anschluß an die Projektierung wird das Gesamtsystemmodell an die Realisierungsentscheidungen angepaßt und überprüft. In dieser Phase ist die Informationsanbindung des MLD-Teams besonders wichtig. Sie läßt sich durch eine geeignete organisatorische Zuordnung beeinflussen.

Abbildung 3.6 zeigt drei mögliche Varianten dieser Zuordnung:

a) Eigenständiges Team:
   Das MLD-Team ist eigenständig und quasi ein weiteres Entwicklerteam.

   Für die erste Phase der Projektierung ist dies problemlos möglich. Hier arbeitet das MLD-Team direkt mit dem Projektmanagement zusammen.

[39] Vgl. Schorcht in [Sch00, S. 26].

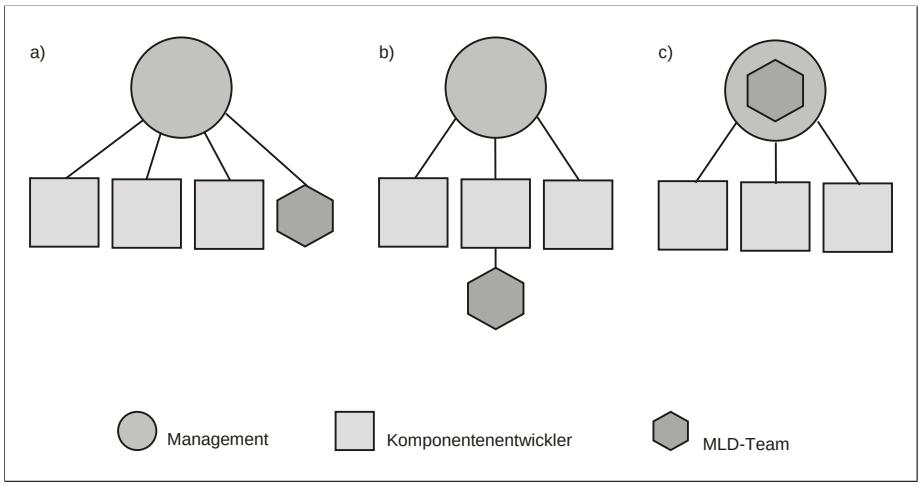

Abbildung 3.6: Möglichkeiten der organisatorischen Zuordnung

In der Phase der Komponentenentwicklung hingegen ist seine parallele Anordnung zu den Entwicklergruppen problematisch. Damit das Gesamtsystemmodell und die Realisierung miteinander verglichen werden können, benötigt das MLD-Team von den Entwicklern Informationen in Form von Realisierungsentscheidungen oder detaillierten Modellbeschreibungen. Um an diese zu gelangen, muß es mit den Entwicklerteams kommunizieren, wofür zwei Wege existieren:

1. Die Kommunikation mit den Entwicklern erfolgt *direkt* und damit am Projektmanagement vorbei. Dies macht es schwieriger, den Überblick über die Korrektheit der im Modell verwendeten Informationen zu wahren. Die benötigten Informationen müssen dabei vom MLD-Team abgefragt werden, wobei ihre Aktualität permanent überprüft werden muß. Hierdurch entsteht die Gefahr, daß das MLD-Team über einen anderen Informationsstand verfügt als das Projektmanagement. In jedem Fall sind zwei Informationswege vorhanden, da die Entwicklerteams ihre Daten sowohl an das Projektmanagement, als auch an das MLD-Team kommunizieren müssen. Dabei kann eine Einsicht der Entwickler in Notwendigkeit der Arbeit des MLD-Teams nicht automatisch vorausgesetzt werden.

2. Die Kommunikation erfolgt *indirekt* über das Projektmanagement. In diesem Fall ist das MLD-Team darauf angewiesen, daß das Projektmanagement unterstützend agiert und die Informationen beschafft und weiterleitet. Dieser indirekte Weg ist nicht realisierbar: Er setzt auf das Projektmanagement, das jedoch hauptsächlich mit anderen Aufgaben beschäftigt ist. Für das Projektmanagement sind die benötigten, detaillierten Realisierungsinformationen zudem nicht von unmittelbarem Interesse.

Durch keine der beiden Varianten ergibt sich eine günstige Informationsanbindung des MLD-Teams.

b) Integriert in ein Entwicklerteam:
Das MLD-Team wird in ein Entwicklerteam integriert.

Für diese Konstellation spricht die Tatsache, daß die Simulationskompetenz bei den Entwicklern oft ohnehin vorhanden ist. Dabei gilt es allerdings zu beachten, daß sich die abstrakte Gesamtsystemmodellierung von eventuellen Simulationen speziell zum Test einer Komponente grundsätzlich unterscheidet.

Für den Fluß der Informationen in den Entwicklungsphasen ergeben sich keine großen Änderungen. Damit bleibt die ungünstige Informationsstruktur erhalten. Diese ist nur für dasjenige Team optimal, in dem die Simulationen durchgeführt werden. An dieser Stelle besteht jedoch zusätzlich die Gefahr, daß die Gesamtsystemsimulation zu einer speziellen Simulation zum Komponententest wird.

c) Integriert in das Projektmanagement:
Die dritte Konstellation besteht in einer Integration in das Projektmanagement.

In der Projektierung ist keine formale Trennung zwischen MLD-Team und Projektmanagement mehr vorhanden. Später agiert es an zentraler Stelle (als Teil des Projektmanagements) und verfügt damit über einen umfassenden Überblick auf Systemebene. Dieser erstreckt sich über alle von den Entwicklern aktuell gemeldeten Daten, so daß das MLD-Team zu jedem Zeitpunkt Zugriff auf den aktuellen Informationsstand hat. In der Kommunikation mit den Entwicklern tritt das MLD-Team als Teil des Projektmanagements auf.

In dieser Variante ist die bestmögliche Informationsstruktur gegeben, denn das Projektmanagement wird um entsprechende Kompetenzen erweitert.

Aus den obigen Ausführungen läßt sich das Fazit ziehen, daß die Anordnung des MLD-Teams beim Projektmanagement am vorteilhaftesten ist. An dieser Position, an der die Gesamtsystemsicht im Vordergrund steht, laufen alle notwendigen Informationen zusammen. Seine Aufgabe besteht in der Durchführung der Simulation auf Missionsebene, wozu die Erstellung des Gesamtsystemmodells (Modellierung), die Missionshandhabung, die Modellnutzung sowie die missionsspezifische Auswertung gehören.

3.3 Zusammenfassung

Der modellgestützte missionsbezogene Entwurf mobiler automatischer Systeme macht es erforderlich, im Rahmen eines Gesamtsystemmodells alle relevanten Komponenten und Funktionalitäten des zu simulierenden Systems nachzubilden. So entsteht ein Modell, anhand dessen das Systemverhalten und seine Leistungsfähigkeit umfassend erprobt werden können.

Dabei wurde durch diese Arbeit der von Schorcht entwickelte Ansatz in zweifacher Hinsicht verändert: Um die Spezifikation besser testweise umzusetzen, werden zum einen die

funktionalen und architektonischen Systemaspekte zu einem **Systemmodell** vereinigt. Dieses Systemmodell entspricht der Idee eines virtuellen Prototyps, also einer testweisen Implementierung der Spezifikation. Zum anderen wird das Umgebungsmodell zu einer virtuellen **Umwelt** aufgewertet, in der der virtuelle Prototyp frei agieren kann.

Für das aus Umwelt und Systemmodell bestehende Gesamtsystemmodell sind diverse **Teilaspekte** zu berücksichtigen: Damit sich das Systemmodell in seiner Umwelt fortbewegen kann, benötigt es die Funktionalität der dynamischen Bewegung. Ebenso muß die Umwelt mit ihren Objekten und Prozessen sowie die Sensorik zu deren Wahrnehmung modelliert werden. Zusätzlich sind die zentralen, begrenzenden Ressourcen Rechenleistung und Energie in die Modellierung einzubeziehen. Beide Aspekte stellen eine Erweiterung herkömmlicher regelungstechnischer Modelle dar und ermöglichen die ganzheitliche Betrachtung des Systems im Sinne des Mission Level Designs.

Innerhalb des missionsbezogenen modellgestützten Entwurfs kommt der **Simulation** eine zentrale Rolle zu, da sie in der Lage ist, alle wesentlichen Phasen des Systementwurfs zu unterstützen: Während der Projektierung dient sie dazu, die in der Entwicklung befindliche Spezifikation anhand der zukünftigen Nutzungsszenarien des Systems zu testen. Dadurch können die Auswirkungen einzelner Entwurfsentscheidungen unmittelbar untersucht werden. Während der Komponentenentwicklung kann durch die schrittweise Einbeziehung von Realisierungsdetails in das Systemmodell fortlaufend geprüft werden, ob das Entwicklungsziel erreicht werden wird. War der missionsbezogene modellgestützte Entwurf erfolgreich, erfüllt das am Ende der Integrationsphase vorliegende reale System die Benutzeranforderungen und weist keine Entwicklungsfehler auf. Soll es anschließend weiterentwickelt werden, stellt das Gesamtsystemmodell ein optimales, weil durch Tests des realen Systems verläßlich erprobtes Versuchsobjekt dar.

Mobile automatische Systeme nehmen ihre Eingangsdaten aus ihrer Umwelt auf und steuern sich auf dieser Basis selbst. Bei der Simulation dieses Verhaltens spielen die Missionen eine herausragende Rolle. Sie stellen typische Nutzungsszenarien des Systems dar, die durch eine Ausprägung von **Modellparametern** nachgebildet werden. Diese beschreiben sowohl das Missionsziel, als auch die Systemzustände und die Konfiguration der Umwelt.

Der Test des Systems erfolgt auf der Basis einer ausgewählten Anzahl von Missionen. Dies bedeutet, daß ein Modell mit mehreren Missionen parametrisiert werden muß, was zu einer erheblichen Menge von Ausgabedaten führt. Aus beiden Gründen muß im Rahmen des Mission Level Designs das herkömmliche Durchführungsverfahren von Simulationen - bestehend aus Modellierung, Simulation und Auswertung - um den Bestandteil der Missionshandhabung ergänzt und die Auswertung um missionsspezifische Funktionalität erweitert werden.

In dem hier vorgestellten, erweiterten **Konzept** umfaßt die Modellierung die Erstellung des Gesamtsystemmodells. Im Rahmen der Missionshandhabung müssen die Missionen einmalig als Teil der Spezifikation festgelegt werden. Sie werden in einer zentralen Missionserstellung und -verwaltung festgehalten, was den Umgang mit ihnen erheblich

beschleunigt und fehlerresistenter macht. Für die Simulation werden die Missionen auf das Gesamtsystemmodell angewendet, wodurch konfigurierte Modelle entstehen, die simuliert werden können. Die auf diese Weise entstehenden Ausgabedaten können im Anschluß missionsweise ausgewertet werden. Dadurch wird es möglich, eine Gesamteinschätzung des Systemdesigns bezüglich der Nutzeranforderungen vorzunehmen und die gewonnenen Erkenntnisse zur Optimierung des weiteren Systementwurfs zu nutzen. Dieses im Rahmen der vorliegenden Arbeit vorgestellte, wesentlich erweiterte Konzept trägt den Besonderheiten von Simulationen im Rahmen des missionsbezogenen modellgestützten Entwurfs Rechnung.

Wie Überlegungen zur organisatorischen Durchführung der Simulation zeigen, sollte ein separates **MLD-Team** gebildet werden. Dieses ist im Kontext des Projektmanagements anzusiedeln, da nur diese Zuordnung den umfassenden Zugang zu den benötigten Informationen und einen Projektüberblick aus der Gesamtsystemperspektive ermöglicht.

4 Framework für den Entwurf auf Missionsebene

Im dritten Kapitel wurde ein Konzept für die Durchführung der Simulation im Rahmen des missionsbezogenen modellgestützten Entwurfs mobiler automatischer Systeme erarbeitet. Nachfolgend wird eine Umsetzung dieses Konzeptes vorgenommen. Dabei handelt es sich um ein Framework aus drei Programmen, in dessen Zentrum das Entwurfsprogramm MLDesigner[40] steht.

Der erste Abschnitt gibt eine Übersicht über das Framework und beschreibt die Gründe für die Auswahl der Programme. Eine detaillierte Beschreibung der Lösung erfolgt direkt im Anschluß.

4.1 Übersicht über das Framework

4.1.1 Analyse der Fähigkeiten von MLDesigner

In Abschnitt 3.2.4 wurde ein aus vier Phasen bestehendes Konzept für die Durchführung der Simulation im Rahmen des missionsbezogenen modellgestützten Entwurfs mobiler automatischer Systeme entwickelt. Bei den Phasen handelt es sich um die Modellierung des Systems, die Missionshandhabung, die Simulation des mit den Missionen konfigurierten Gesamtsystemmodells (Modellnutzung) sowie die missionsspezifische Auswertung. Jede dieser Phasen muß von dem zur Simulationsdurchführung genutzten Programm unterstützt werden. Damit stellt sich bezogen auf die Vorgabe des Einsatzes von MLDesigner die Frage, welche der vier identifizierten Simulationsphasen mit MLDesigner durchgeführt werden können und welche Erweiterungen beziehungsweise Änderungen für den genannten Einsatzzweck notwendig sind.

Um diese Frage sachgerecht beantworten zu können, ist eine Analyse der Möglichkeiten von MLDesigner bezüglich der gestellten Anforderungen unerläßlich.

[40] Eine Einführung zum Programm bietet Abschnitt 2.3 ab Seite 21.

4.1.1.1 Erstellung und Simulation von Modellen

Dabei soll zunächst die Eignung von MLDesigner für die Modellierung und Modellnutzung untersucht werden:

Die Stärke von MLDesigner besteht in seinen sehr guten Fähigkeiten bei der Erstellung und Simulation von Modellen. Das Programm unterstützt die graphische Modellierung mittels einer übersichtlichen Benutzeroberfläche. Die Blöcke (Standardblöcke oder selbst erstelle) werden hierbei einfach in das Modell gezogen. Danach sind ihre Aus- und Eingänge zu verbinden und die Parameter zu setzen. Dabei steht für die Modellerstellung eine Vielzahl vorgefertigter Blöcke zur Verfügung, die zudem aus mehreren Domänen stammen und gemischt eingesetzt werden können. Damit ergibt sich die Möglichkeit einer hybriden Simulation, die es dem Anwender erlaubt, für jedes Teilmodell den jeweils optimalen Modellierungsansatz zu nutzen.

4.1.1.2 Parametrisierung

Im Hinblick auf die Frage, inwieweit sich MLDesigner für die Missionshandhabung eignet, spielen die Möglichkeiten zur Modellparametrisierung eine entscheidende Rolle. Hierbei verfolgt MLDesigner den folgenden Ansatz: Die Parameter der einzelnen Blöcke lassen sich entweder direkt mit einem Wert belegen oder auf obere Modellebenen verlinken. Das Verlinken geschieht, indem auf der nächsthöheren Ebene ein Parameter vom selben Typ erzeugt wird und der Blockparameter an diesen Wert gekoppelt - verlinkt - wird. Mittels dieses Verfahrens lassen sich die Parameter Schritt für Schritt bis zur obersten Ebene verlinkt. Dort stehen sie dann für eine zentrale Belegung zur Verfügung.

Dieser von MLDesigner gewählte Ansatz für die Parametrisierung eignet sich lediglich für flache Modellhierarchien mit wenigen Parametern. In Abschnitt 3.1.2 wurde für die Modellierung mobiler automatischer Systeme jedoch dargelegt, daß diese Modelle stark hierarchisch gegliedert sind. Somit müssen die Parameter über mehrere Ebenen hinweg verlinkt werden. Angesichts der großen Zahl von Verlinkungen werden dadurch Änderungen am Modell erschwert, da in diesem Fall alle entsprechenden Verlinkungen angepaßt werden müssen. Außerdem birgt dies bei einer größeren Parameteranzahl schnell die Gefahr der Unübersichtlichkeit. Dies resultiert daraus, daß sich der Verwendungsort eines Parameters nur feststellen läßt, indem die Parameter der einzelnen Blöcke auf ihre verlinkten Parameter hin untersucht werden.

Die Eingabe von Missionen erlaubt MLDesigner nur durch direkte Vorgabe von Parameterwerten, wobei keine Unterscheidung der Parametertypen vorgesehen ist. Das bedeutet, daß der Nutzer bei der Parametrisierung für jede Mission selbst entscheiden muß, welche Werte wann zu ändern sind. Dabei müssen die Werte für jede Mission immer wieder von neuem eingeben werden, da MLDesigner keine Speicherung von Parametersets vorsieht.

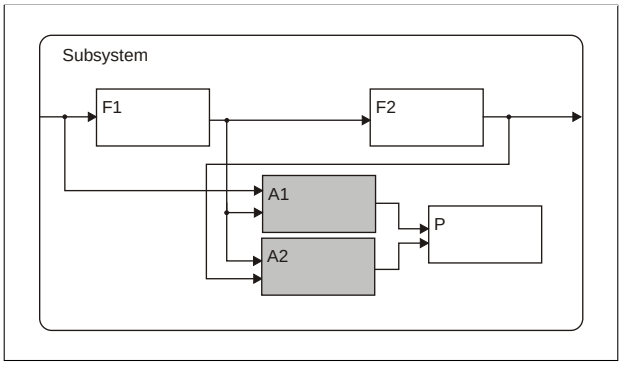

Abbildung 4.1: Beispiel mit Aufbereitung

Es läßt sich zudem keine graphisch oder anderweitig unterstützte Eingabe realisieren, vielmehr ist die Eingabe eine rein textbasierte Wertvorgabe[41].

4.1.1.3 Simulationsauswertung

Für die missionsspezifische Auswertung der Simulationsergebnisse bietet MLDesigner standardmäßig eine Basisauswahl von Blöcken an: Es verfügt über Blöcke zur Kurvendarstellung (2D Plot) und zur Anzeige von Zahlenwerten. Desweiteren stehen über die Tcl/Tk-Anbindung diverse einfache Ausgabemöglichkeiten bereit. Zusätzlich lassen sich die Daten zur Weiterverarbeitung in Dateien schreiben. Damit stellt MLDesigner die grundlegenden Möglichkeiten für eine Auswertung von Simulationsergebnissen bereit.

Für den Einsatz im Rahmen des missionsbezogenen modellgestützten Entwurfs ergeben sich jedoch drei prinzipielle Probleme:

- **Vermischung:**
  Die Aufbereitung der Daten für die Darstellung erfolgt in MLDesigner durch zusätzliche Blöcke, die den Darstellungsblöcken vorgelagert sind.

  Ein Beispiel gibt die Abbildung 4.1: Um die Ausgabe der Funktionsblöcke F1 und F2 mit P darstellen zu können, werden die der Aufbereitung dienenden Blöcke A1 und A1 zwischengeschaltet. Sie finden nur in der Darstellung Verwendung.

  MLDesigner vermischt an dieser Stelle zwei verschiedene Aufgaben. Es integriert die Aufbereitung der Daten für die Simulationsauswertung in die Modellierung, was die Interpretierbarkeit des Modells negativ beeinflußt. Der Betrachter muß zunächst herausfinden, ob die Blöcke zum Modell oder zur Aufbereitung der Daten für die Ausgabe gehören. Das Problem verschärft sich, wenn für die Aufbereitung Daten benötigt werden, die nicht lokal innerhalb dieses Blocks verfügbar sind. In diesem Fall müssen die entsprechenden Daten aus anderen Teilen des Modells

[41] Es existiert zwar eine Tcl/Tk-Anbindung. Diese ist mit ihren interaktiven Eingabeelementen jedoch nicht für die Einstellung von Parametern geeignet.

bezogen werden. Dafür sind zusätzliche Verbindungslinien und entsprechend zusätzliche Ein- oder Ausgänge nötig. Diese Verbindungslinien, Ein- und Ausgänge repräsentieren keine Struktur des Systems, sondern werden dem Modell einzig zum Zweck der Datenaufbereitung hinzugefügt. Im Ergebnis verschlechtert sich die Erfaßbarkeit der Systemstruktur. Diese ist aber gerade ein Teil des Systementwurfes. Der Entwurf wird somit durch simulationstechnische Fragen negativ beeinflußt.

- **Fenster:**
  Ein weiteres Problem besteht in der Tatsache, daß MLDesigner für jeden der Ausgabeblöcke ein Fenster mit dem zugehörigen Plot erzeugt. Dabei läßt sich für jedes der Fenster dessen Position und die Größe einzeln durch Parameter des Ausgabeblocks einstellen. Bei einer großen Anzahl von Ausgabewerten sind jedoch viele Fenster offen, die sich teilweise überlagern, was eine gezielte Interpretation einzelner Verläufe erschwert.

- **Simulationsnähe:**
  Drittens erzwingt MLDesigner mit seiner simulationsnahen, sich direkt an die Simulation anschließenden Darstellung der Plots deren sofortige Auswertung. Zwar können Daten in Dateien geschrieben werden, aber das Programm ist nicht in der Lage, sie für eine spätere Auswertung direkt wieder anzuzeigen. Für eine große Anzahl von Missionen ist dies problematisch.

  Damit unterstützt MLDesigner zwar eine Speicherung der gewonnenen Ergebnisse, bietet aber keine die Möglichkeit mit den Ergebnisse außer direkt nach der Simulation zu arbeiten. Beispielsweise fällt ein direkter Vergleich zweier Simulationsläufe einer Mission schwer. Dies aber stellt eine wichtige Hilfe bei der Bewertung der am Design, und damit auch am Gesamtsystemmodell, vorgenommenen Veränderungen dar.

Die Analyse der Fähigkeiten von MLDesigner bezüglich der Anforderungen, die der missionsbezogene modellgestützte Entwurf mobiler automatischer Systeme stellt, legt somit einige Probleme offen. Diese betreffen die Unterstützung von Missionen in der Parametrisierung und in der Auswertung, nicht jedoch die Modellierung und Simulation.

4.1.2 Konzept für das Framework

Aus dem vorausgegangenen Abschnitt läßt sich das Fazit ziehen, daß MLDesigner nicht optimal für den missionsbezogenen modellgestützten Entwurf mobiler automatischer Systeme geeignet ist. Deshalb wurde ein Konzept für ein Framework entwickelt, das diesem Einsatzzweck entspricht und mit dem folgenden Abschnitt vorgestellt werden soll.

4.1.2.1 Zuordnung der Programme zu den einzelnen Phasen

Das Framework setzt gemäß der Aufgabenstellung weiterhin auf das Entwurfsprogramm MLDesigner. Es führt jedoch zwei wesentliche Prinzipien ein, die aus der Analyse der Fähigkeiten von MLDesigner abgeleitet wurden:

- **Trennung von Modellierung und Missionshandhabung**
  Durch die Trennung von Modellierung und Missionshandhabung läßt sich die Unterstützung von Missionen ebenso wie eine erweiterte Funktionalität bei der Parametereingabe ermöglichen.

  Die Unterstützung von Missionen impliziert, daß die Parametrisierung zwischen den verschiedenen Parametertypen unterscheidet. Auf diese Weise wird die Beschreibung der Missionen (mittels der Missionsparameter) von der Auslegung des Systems (mittels der Systemparameter) getrennt.

  Desweiteren lassen sich die anderen Kritikpunkte, wie die Nichtspeicherbarkeit von Parametersets für die einzelnen Missionen oder die rein textorientierte Wertevorgabe lösen. Die damit verbundene Vereinfachung und Unterstützung der Parametrisierung hat zudem zur Folge, daß auch Nutzer die Simulationen durchführen können, die nicht mit der Modellierung und mit den Modell vertraut sind.

- **Trennung von Modellnutzung und Auswertung**
  Die Trennung von Modellnutzung und Auswertung führt dazu, daß die Simulationsergebnisse zwischengespeichert werden müssen, was gleichzeitig ihre Archivierung erleichtert. Dies vereinfacht wiederum den Vergleich zwischen aktuellen und früheren Resultaten, weil das externe Programm immer auf gespeicherte Ergebnisse zugreift.

  Vorteilhaft ist desweiteren, daß sich sowohl die Aufbereitung, als auch die Darstellung getrennt von der Simulation erweitern lassen. Es ist also möglich, neue Aufbereitungsalgorithmen und Plots auch zu einem späteren Zeitpunkt in die Auswertung einfließen zu lassen, ohne Änderungen am Modell vornehmen oder neu simulieren zu müssen. Indem die Datenaufbereitung nach der Beendigung der Simulation durchgeführt wird, wird sie aus dem Modell entfernt, womit das Modell besser die Struktur des Systems abbildet.

Aus beiden Prinzipien ergibt sich eine Auftrennung der einzelnen Phasen zu verscheidenen Programmen. Dadurch können die benötigten Veränderungen im Bereich der Parametrisierung und der missionsspezifischen Auswertung erfolgen, ohne daß in das Programm MLDesigner eingegriffen wird.

Die Veränderungen werden mit Hilfe zweier neuer Programme realisiert, die im Zusammenspiel mit MLDesigner das Framework für den missionsbezogenen modellgestützten Entwurf mobiler automatischer Systeme bilden. In ihm übernehmen spezialisierte Programme die einzelnen Phasen, wie Tabelle 4.1 mit der gewählten Zuordnung veranschaulicht.

Phase	Programm
Modellierung	MLDesigner
Missionshandhabung	spezialisiertes Programm (MLEditor)
Simulation	MLDesigner
Auswertung	spezialisiertes Programm (MLVisor)

Tabelle 4.1: Phasen der Simulation und zugehörige Programme des Frameworks

Die Zuordung lautet dabei wie folgt:

- **MLDesigner** wird für die Modellierung und die Simulation genutzt.
- Für die Handhabung der Missionen (also die Parametrisierung des Modells) existiert ein spezielles Programm. In Anlehnung an seine Funktion und an den Namen MLDesigner trägt es die Bezeichnung **MLEditor**.
- Gleiches gilt für **MLVisor**, das speziell für die missionsspezifische Auswertung zuständig ist.

Eine graphische Veranschaulichung des entwickelten Konzeptes für das Framework findet sich in Abbildung 4.2.[42]

4.1.2.2 Schnittstellen zwischen den Programmen

Die bloße Zuordnung der Programme des Frameworks zu den Aufgaben ist noch nicht ausreichend. Vielmehr ist die Betrachtung der Schnittstellen zwischen ihnen und damit des Aspekts des Datenaustausches notwendig. Im nachfolgenden Abschnitt wird dies geboten, indem auf die Schnittstellen eingegangen und die Zusammenarbeit der Programme beschrieben wird. Die einzelnen Punkte lassen sich dabei anhand der Abbildung 4.2 und der in ihr vergebenen Nummern nachvollziehen.

Parameterübergabe

Eine Erkenntnis des vorigen Abschnitts besteht in der Notwendigkeit, die Parametrisierung aus MLDesigner herauszulösen und diese Aufgabe an MLEditor zu übergeben, wodurch eine verbesserte Erstellung und Verwaltung der Missionen erreicht wird. Entsprechend müssen beiden Programme miteinander interagieren, da nur so die Missionen außerhalb des Modells verwaltet werden können.

Der gewählte Weg führt hierbei über die MLDesigner-Modelldateien. Dieses Vorgehen hat den Vorteil, daß die Änderungen für MLDesigner vollkommen transparent gestaltet werden können. Es erfordert jedoch einerseits, daß MLEditor die MLDesigner-Modelle lesen und die darin enthaltenen Parameter erfassen kann (1). Andererseits muß MLEditor

[42] Vgl. Abbildung 3.5, Seite 56 für das allgemeine Vorgehen.

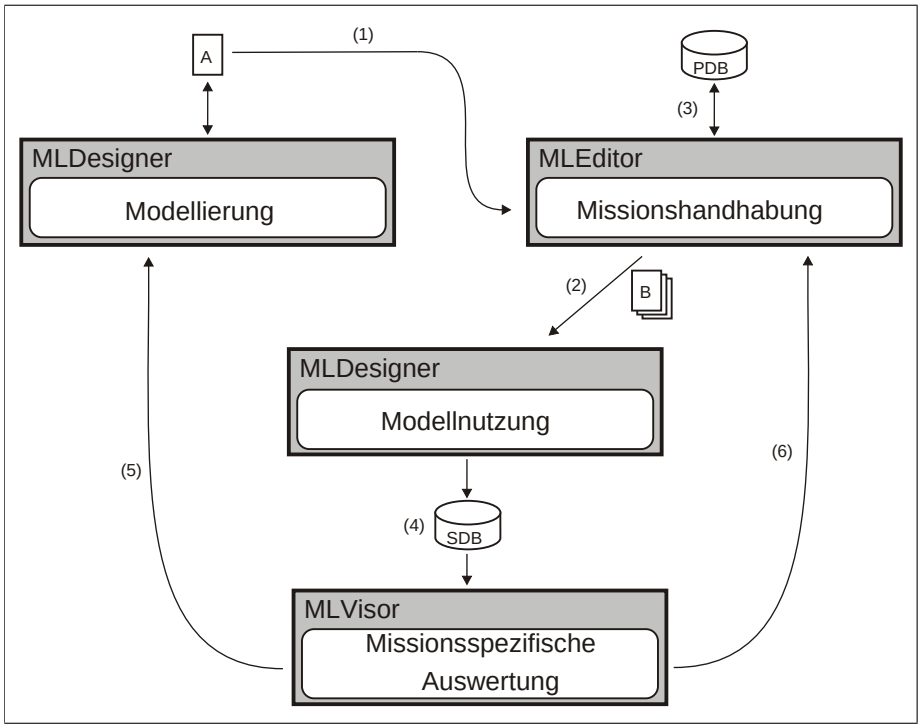

Abbildung 4.2: Konzept des Frameworks

in der Lage sein, die Modelle mit neuen Parametern für die Simulation zu schreiben (2). Auf der Basis der gelesenen Parameter wird somit von MLEditor die Parametrisierung des Modells unterstützt. Durch dieses Vorgehen des Frameworks entstehen zwei Typen von Modellen:

- **Entwurfsmodell** (A):
  Das Entwurfsmodell ist dasjenige MLDesigner-Modell, an dem der Entwurf vorgenommen wird, wobei es jedoch noch nicht parametrisiert ist.
- **Konfigurierte Simulationsmodelle** (B):
  Diese Simulationsmodelle werden von MLEditor automatisch aus dem Entwurfsmodell temporär erzeugt. Sie dienen der Simulation, denn sie stellen eine Version des Entwurfsmodells dar, die für eine Mission parametrisiert wurde. Ihre Anzahl entspricht dabei der jeweiligen Anzahl von Missionen.

Parameterspeicherung

Die Parameter werden nun für verschiedene Modelle zentral in einer Datenbank verwaltet (3). Diese Parameterdatenbank (PDB) bietet folgenden Vorteil: Die Parametrisierung für eine Mission erfordert vom Nutzer nur die Angabe der Mission und nicht mehr das Einstellen der Werte, da die Parameterwerte gespeichert sind. Eine Mission spiegelt sich somit in einer Konfiguration des parametrisierten Modells wider. Folglich muß MLEditor für jede Mission eine parametrisierte Version des Entwurfsmodells erzeugen, welches

dann mit MLDesigner zu simulieren ist. Die Missionssimulation erfolgt mithin durch die Teilschritte Modellierung (MLDesigner), Missionserstellung (MLEditor) und Simulation (MLDesigner).

Die Besonderheit dieses Vorgehens gegenüber dem allgemeinen Konzept besteht darin, daß die Missionen direkt an das Modell gebunden und somit nicht frei von diesem erstellbar sind. Das heißt, Missionen lassen sich ausschließlich mit Parametern des Entwurfsmodells beschreiben und sind insofern von diesem abhängig.

Speicherung der Simulationsergebnisse

Durch die Trennung von Modell und Aufbereitung bzw. Darstellung wird es nötig, die Simulationsergebnisse zu speichern. Dafür wird im hier vorgestelltem Konzept eine Simulationsdatenbank (SDB) genutzt (4).

MLDesigner schreibt die Daten während der Simulation in die SDB, MLVisor nutzt sie für die Darstellung. Ein Vorteil der SDB liegt in der gleichzeitigen Archivierung der Ergebnisse. So lassen sich die Ergebnisse früherer Simulationen abrufen und direkt mit neueren Ergebnissen vergleichen. Außerdem kann die Auswertung der Missionen ohne Unterbrechung durch Simulationen und in beliebiger Reihenfolge durchgeführt werden. Da zusätzlich auf die Informationen aus der Parameterdatenbank zurückgegriffen werden kann, ist es zudem möglich, einen halbautomatischen Test auf die Einhaltung der System- oder der Entwurfsparameter durchzuführen.

Die Methodik der nachgelagerten Datenauswertung mittels Datenaustausch über eine Datenbank läßt jedoch keine interaktive Beeinflussung des Simulationsablaufs mehr zu. Der Nutzer ist also nicht in der Lage, über die (nachgelagerte und damit zeitversetzte) Auswertung die Simulation selbst zu beeinflussen.

> Ein Beispiel verdeutlicht diesen Sachverhalt: Bei einem Flugsimulator steuert der Nutzer die Simulation. Um aktiv einzugreifen, nutzt er eine Aufbereitung der Simulationsdaten, z.B. in Form eines über einen Monitor ausgegebenen Bildes des Flugzeuges in seiner Umwelt. Das hier vorgestellte Konzept sieht diese aktive Rückwirkung auf die Simulation nicht vor. Es ist vergleichbar mit einem Flugschreiber, der lediglich die nachträgliche Untersuchung des Flugverlaufs zuläßt.

Für mobile automatische Systeme kann dies ein Problem sein, wenn das Modell im Zusammenhang mit der Simulation eine Interaktion mit dem Nutzer vorsieht. Um dies zu vermeiden, ist die Nutzerinteraktion durch ein geeignetes Modell zu ersetzen. Dieses ist zudem präziser in bezug auf die Reproduzierbarkeit der Modellinteraktion als eine manuelle Vorgabe und unterstützt die Festschreibung der Missionen.

Im Gegensatz dazu wird die Simulation mobiler autonomer Systeme nicht eingeschränkt, weil hier das mobile autonome System in der Simulation selbständig handelt und von daher keine Nutzerinteraktion mehr vorhanden ist. Aus der Loslösung von direkten Benutzereingaben resultiert die Möglichkeit einer Zeitraffung der Simulation. Das heißt, die

Simulation kann schneller als die Echtzeit ablaufen. Dies stellt bei einer großen Anzahl von Missionen und einer langen Missionsdauer einen Vorteil dar.

Nutzung der Ergebnisse

Die aus der Auswertung gewonnenen Erkenntnisse werden anschließend in das Entwurfsmodell übertragen. Dabei können Änderungen am Design des Systems vorgenommen werden (5). Eine andere Möglichkeit besteht in der Anpassung der Auslegung der Systemkomponenten über eine Änderung ihrer Systemparameter in MLEditor (6).

Das verwendete Konzept sieht als Akteur für beide Vorgehensweisen den Entwickler vor. Es wäre jedoch auch denkbar, daß die Änderungen automatisch vorgenommen werden. In diesem Fall ließe sich das Systemdesign automatisch optimieren. Für komplexe Systeme stellt dies jedoch eine überaus anspruchsvolle Aufgabe dar, weshalb an dieser Stelle auf die menschliche Erfahrung und Intelligenz gesetzt wird.

Das hier präsentierte Framework-Konzept wurde vom Autor in "A Framework for Mission Level Design" im Detail vorgestellt ([LZL03]).

4.1.3 Randbedingungen

Für die Realisierung des erläuterten Konzepts bestehen verschiedene Randbedingungen:

- **Linux:**
  Als Betriebssystem wurde GNU Linux gewählt ([Lin]), da es die native Arbeitsumgebung von MLDesigner ist. Hierdurch konnte ein aufwendiger Betriebssystemwechsel vermieden werden. Zusätzlich bietet diese Plattform alle nötigen Voraussetzungen zur Erstellung eigener Software (C++-Compiler, Standardbibliotheken [RW00]).
- **Qt:**
  Für die Bedienoberflächen[43] wurde Qt herangezogen ([Tro]). Qt unterstützt die Programmierung von Oberflächen mittels entsprechender C++-Klassen, beispielsweise für Fenster und Menüs. Ferner stellt die Bibliothek leistungsfähige Klassen für die Arbeit mit Datenbanken und Datenstrukturen zur Verfügung, wodurch sich der Programmieraufwand verringert. Durch den Einsatz des Qt-Designers ist es möglich, die Oberflächen graphisch zusammenzustellen. Im Ergebnis entsteht ein Gerüst für die Oberfläche, das um die spezifische Funktionalität erweitert werden muß.

  Ein weiterer Grund für den Einsatz von Qt ist die Tatsache, daß es für verschiedene Betriebssysteme existiert, was eine spätere Portierung von Programmen erleichtert.

[43] engl.: Graphical User Interface (GUI)

Bei einem nichtkommerziellen Einsatz fallen zudem keine Lizenzkosten an, da die Bibliothek unter der GNU General Public License (GPL, siehe unten) nutzbar ist. Nicht zuletzt muß die hohe Qualität der Bibliothek als Grund ihrer Verwendung angeführt werden. Diese äußert sich unter anderem in ihrer weiten Verbreitung unter Linux.

- **MySQL:**
  Die Wahl einer Datenbank fiel auf die Open-Source Datenbank MySQL ([MyS]), die über die für den bezweckten Einsatz benötigte, grundlegende Funktionalität verfügt. Für MySQL spricht zudem ihre weite Verbreitung und ihre GPL-Lizenz bei nichtkommerzieller Nutzung. Hinzu kommt, daß Qt bereits eine direkte Unterstützung für den Zugriff auf MySQL-Datenbanken anbietet.
- **GNU General Public License (GPL):**
  Abschließend sind einige ergänzende Bemerkungen zur GNU General Public License ([Fre]) und deren Auswirkungen auf die entwickelten Programme notwendig. Die GPL erlaubt es dem Nutzer, Programme auszuführen, zu kopieren, zu verteilen, zu analysieren und zu verbessern. Um dies zu erreichen, müssen GPL-Programme stets inklusive ihres Quellcodes verteilt werden. Unter Nutzung von GPL-Code geschriebene Programme sind ihrerseits unter GPL zu lizenzieren. Damit lassen sich keine kommerziellen Programme auf Basis von GPL-Code aufbauen. Zum Teil ist es (wie im Fall von Qt und MySQL) jedoch möglich, eine kommerzielle Lizenz zu erwerben, welche die Erstellung von kommerziellen Programmen erlaubt.

4.2 Detaillierte Beschreibung des Frameworks

4.2.1 Modellierung mit MLDesigner

Der erste Schritt im Rahmen des missionsbezogenen modellgestützten Entwurfs mobiler automatischer Systeme besteht in der Modellierung des Systems. Das Gesamtsystemmodell enthält das Systemmodell (den virtuellen Prototyp) mit seinen verschiedenen Aspekten sowie die Umwelt. Es wird durch die System- und die Missionsparameter parametrisiert, die beide von MLEditor verwaltet werden. Die Modellierung selbst wird mit MLDesigner vorgenommen.

MLDesigner unterstützt die blockorientierte Modellierung über eine graphische Benutzeroberfläche. Blöcke besitzen sowohl Ein- und Ausgänge, als auch Parameter, durch welche sie konfiguriert werden. Desweiteren können aus Blöcken Subsysteme zusammengestellt werden. Letztere tragen in MLDesigner die Bezeichnung Module, während die Blöcke selbst als Primitiven bezeichnet werden. Eine detaillierte Einführung findet sich in der Dokumentation des Programms [MLD03a].

Im Lieferumfang von MLDesigner ist eine große Anzahl einsatzbereiter Primitiven für verschiedene Domänen enthalten. Diese werden, geordnet nach Domänen, in Bibliotheken

verwaltet. Um das Auffinden der Primitiven und Module zu vereinfachen, sind die Bibliotheken in sinnvolle Untergruppen untergliedert. So werden für die DE-Domäne 40 Untergruppen mit Primitiven beispielsweise für Datenquellen (Sources) und -senken (Sinks), arithmetische Operationen (Arithmetic) oder Statistiken (Statistics) unterschieden. Sollte die große Menge vorgefertigter Primitiven für eine spezielle Aufgabe nicht ausreichen, ist es möglich, zusätzlich eigene zu schreiben. MLDesigner stellt hierfür einen Editor und die nötige Dokumentation zu Verfügung. Eigene Primitiven werden zusammen mit den eigenen Modulen und Systemen in einer lokalen Bibliothek verwaltet. Sie lassen sich sowohl exportieren, als auch zu einem späteren Zeitpunkt wieder importieren.

Die graphische Modellierung erlaubt das Zusammenfügen selbst komplexer Modelle aus Einzelblöcken. Hierzu muß sie der Nutzer aus den Bibliotheken auswählen, in das Modell einfügen, ihre Ein- und Ausgänge verbinden und abschließend ihre Parameter mit Werten belegen. Die Bedienoberfläche hält hierzu eine einfache Möglichkeit für die Parametereingabe bereit.

Durch das für den missionsbezogenen modellgestützten Entwurf mobiler automatischer Systeme vorgestellte Konzept[44] ergeben sich Änderungen gegenüber der standardmäßigen Modellierung mit MLDesigner:

- **Parametermarker:**
  Die erste Änderung resultiert aus der Verwaltung der Parameter mittels MLEditor außerhalb von MLDesigner. Das vorgestellte Konzept sieht vor, daß die Parameter im MLDesigner-Gesamtsystemmodell durch MLEditor erkannt werden. Die Eingabe der Werte für diese Parameter erfolgt dann innerhalb von MLEditor.

  Für die Modellierung bedeutet dies, daß die Werteeingabe für die Parameter nicht in MLDesigner geschieht. Entsprechend ist keine Verlinkung der Parameter auf höhere Ebenen mehr nötig.

  Vielmehr muß die Erkennung der Parameter durch MLEditor unterstützt werden. Dies geschieht, indem die System- und Missionsparameter durch Marker gekennzeichnet werden. Ein solches Vorgehen hat den Vorteil, daß im Modell klar ersichtlich ist, welche Parameter extern verwaltet werden. Außerdem wird der Aufwand für MLEditor reduziert[45].

  Da die Entwurfsziele durch die Trennung von Modell und Datenaufbereitung nicht mehr Teil des Modells sind, existieren für die Entwurfsparameter keine Marker. Das Gesamtsystemmodell beschreibt lediglich das System und seine Umwelt.

[44] Vgl. Abschnitt 4.1, Seite 63.

[45] Bei einer Lösung ohne die Marker müßten alle Parameter in MLEditor zur Auswahl angeboten werden. Die Selektion der Missions- und Systemparameter, für die eine Eingabe ihrer Werte erfolgen soll, wäre in diesem Fall eine Aufgabe des MLEditor-Nutzers, der nicht notwendigerweise ein umfangreicheres Wissen über das Modell besitzt. Durch die Marker erkennt MLEditor die Parameter direkt, der erforderliche Aufwand - für den Nutzer und für das Programm - wird so reduziert.

Die Verwendung der Marker führt dazu, daß das Modell nicht mehr durch MLDesigner ausführbar ist. Der Grund dafür resultiert aus der Tatsache, daß MLDesigner die Marker nicht kennt und damit nicht unterstützt.

- **Datenbank:**
  Eine zweite Änderung betrifft die Vorbereitung der Simulationsauswertung: Im Gegensatz zum üblichen Modellierungsprozeß muß hier keine Aufbereitung der Daten für die Visualisierung in das Modell integriert werden. Diese Aufgabe wird von MLVisor übernommen.

  Damit MLVisor die Daten zur Verfügung stehen, müssen sie in die Simulationsdatenbank geschrieben werden. Zu diesem Zweck werden gezielt dort spezielle Primitiven plaziert, wo Daten vorhanden sind, die ausgewertet werden sollen. Sie ersetzen also die Primitiven für die Plots, während diejenigen für die Aufbereitung wegfallen. Die neuen Primitiven stellen eine Verbindung zur Simulationsdatenbank her und hinterlegen dort die Daten.

4.2.2 Missionshandhabung mit MLEditor

Als Resultat des ersten Schritts entsteht das Entwurfsmodell mit Markern für die Missions- und Systemparameter, das im internen Dateiformat von MLDesigner vorliegt. Die Missionshandhabung nimmt nun eine Konfiguration des Modells mit den Missionen vor, wozu das Programm MLEditor benutzt wird. Dahinter steht die Idee, die Arbeit mit den Missionen an einer zentralen Stelle zu bündeln und dadurch zu vereinfachen. MLEditor kommt also die Funktion zu, die Parameter aus dem Entwurfsmodell zu extrahieren, die Erstellung von Missionen zu ermöglichen und die mit den Missionen konfigurierten Simulationsmodelle zu schreiben.

Zu diesem Zweck wird das Gesamtsystemmodell zunächst von MLEditor eingelesen. Dabei erkennt das Programm die Marker und erzeugt ein internes Abbild des Parameterbaums[46]. Da dieser Schritt vor jeder Parametrisierung wiederholt wird und zudem der letzte Parameterbaum in der Parameterdatenbank abgespeichert wird, werden Änderungen im Parameterbaum erkannt, wobei sich drei mögliche Veränderungen unterscheiden lassen:

1. Neue Parameter kommen hinzu:
   Für Missionsparameter erweitert sich in diesem Fall jede Mission um einen Parameterwert. Entsprechend müssen in der Datenbank Veränderungen durchgeführt werden. Handelt es sich dagegen um einen Systemparameter, muß für diesen nur ein neuer Eintrag geschaffen werden.

[46] Wie Abschnitt 3.1.2 gezeigt hat, sind die Gesamtsystemmodelle hierarchisch gegliedert. Diese Gliederung läßt sich durch eine Baumstruktur beschreiben. Ergänzt man in dieser Struktur für jeden Block die Parameter, erhält man den Parameterbaum, der somit alle Parameter mit dem Ort ihres Auftretens beschreibt.

2. Parameter werden aus dem Modell entfernt:
   Für Missionsparameter müssen die Werte dieses Parameters aus jeder Mission entfernt werden. Bei Systemparametern wird der gesamte Parameter aus der Datenbank gelöscht.

3. Parameter werden im Modell verschoben:
   In diesem Fall muß lediglich der Parameterbaum angepaßt werden. Es ergeben sich keine Änderungen in der Datenbank.

Durch die Identifikation dieser Veränderungen wird die Datenbank zu jedem Zeitpunkt auf dem aktuellen Stand gehalten.

Die Verwaltung der verschiedenen Parametertypen des Systems erfolgt getrennt voneinander.

- **Missionsparameter:**
  Für die Missionsparameter wird die Erstellung und das Löschen von Missionen unterstützt. Eine Mission wird dabei jeweils durch eine Konfiguration der Werte aller Missionsparameter nachgebildet. Für jede der Missionen kann jeweils jeder Parameterwert eingegeben werden.

- **Systemparameter:**
  Den Systemparametern läßt sich jeweils nur ein Wert zuweisen, der die aktuelle Auslegung einer Komponente festlegt.

- **Entwurfsparameter** (vorgesehen):
  In Abschnitt 4.2.1 wurde festgestellt, daß die Entwurfsparameter nicht mehr Teil des Modells sind. Sie beschreiben die Entwurfsziele und sind deshalb nur für die Auswertung von Interesse. Da MLEditor die zentrale Eingabeinstanz für Parameter darstellt, ist es nützlich, auch die Entwurfsparameter hier zu verwalten. Damit sind alle Parametertypen bis auf die Konstanten, die im Modell verbleiben, an einer zentralen Stelle einstellbar. Um die Entwurfsparameter nutzen zu können, muß die Auswertung direkt auf die Parameterdatenbank zugreifen.

Für die Eingabe der Parameterwerte ist es wichtig, daß diese soweit wie möglich unterstützt wird. Dazu gehört als Mindestanforderung die Möglichkeit zur typsicheren Eingabe der verschiedenen Datentypen (z.B. Gleitkommazahlen). Für eine optimale Missionserstellung ist freilich eine erweiterte Eingabemöglichkeit wünschenswert, die den Nutzer aktiv bei der Parametrisierung, z.B. über eine visuelle Rückkopplung, unterstützt. Folgendes Beispiel illustriert diese Forderung:

> Eine Mission bei der Simulation des Verhaltens eines autonomen Tischroboters besteht im Erreichen eines Zielpunkts in einem Hindernisparcours. Die Positionen der Hindernisse, der Start- und der Zielpunkt sind drei der Missionsparameter. Für ihre Eingabe wäre ein graphischer Plot hilfreich. Dieser sollte es sowohl erlauben, die Hindernisse graphisch darzustellen, als auch, sie per Koordinatenvorgabe zu plazieren. Desweiteren sollten die Hindernisse bei der Vorgabe des Start- und des Zielpunktes sichtbar sein. Der Plot entspricht in diesem Fall der visuellen Rückkopplung. Verglichen mit einer reinen Vorgabe der Werte als Koordinatenpaare verbessert er die Handhabung der Werteeingabe.

Eine andere Form der unterstützten Dateneingabe ist die geführte Eingabe in Form von Datenmasken. Diese ist zum Beispiel dann anwendbar, wenn der einzugebende Parameter ein komplexes Protokoll besitzt. Das Ziel liegt in jedem Fall in der Unterstützung des Nutzers bei der Eingabe der Parameterwerte.

Die Parameterkonfiguration ermöglicht damit die Eingabe und Verwaltung der Missions-, System- und Entwurfsparameter. Die Missions- und Entwurfsparameter werden dabei einmalig erstellt. Werden Änderungen am Entwurfsmodell vorgenommen, müssen die Missionsparameter entsprechend angepaßt werden. Auch die Systemparameter werden von Änderungen im Entwurfsmodell beeinflußt. Zusätzlich werden im Rahmen des Entwurfs auch ihre Werte, die die Auslegung von Systemkomponenten beschreiben, immer wieder manipuliert.

Die Missionshandhabung in MLEditor führt zur Unterscheidung von Entwurfsmodell (Gesamtsystemmodell mit den Parametermarkern) und konfigurierten Simulationmodellen (für Missionen parametrisierte Entwurfsmodelle). Die Missionen zum Test des Systemdesigns sowie die Auslegung des Systems wird über Modellparameter vorgenommen. Zum Test des Systemdesigns durch die Simulation wird das Entwurfsmodell mit den Werten der Systemparameter sowie mit denen der Missionsparameter aus einer der Missionen konfiguriert. Für jede Mission wird also eine eigens konfigurierte Version des Entwurfsmodells in einem gesondert wählbaren Verzeichnis erzeugt. Da die Marker durch entsprechende Parameterwerte ersetzt wurden, sind diese konfigurierten Simulationsmodelle nun wieder durch MLDesigner ausführbar.

4.2.3 Modellnutzung mit MLDesigner

Nach Abschluß der Konfiguration liegen die mit den Missionen konfigurierten Simulationsmodelle vor. Sie bilden den Ausgangspunkt der sich anschließenden Simulation, für die jedes dieser Modelle einzeln in MLDesigner geladen wird.

Die dabei erzeugten Ergebnisse werden unter Zuhilfenahme der während der Modellierung eingefügten Datenbankprimitiven in die Simulationsdatenbank geschrieben. Dazu öffnen die Primitiven während der Simulation eine Verbindung zur Datenbank und legen die Daten geordnet nach Simulation, Mission und Datensatz ab. Zusätzlich werden zur leichteren Handhabung der einzelnen Ergebnisse Metadaten, wie die Nummer der Simulation, das Datum und der Name des Nutzers und des Rechners, hinterlegt.

Eine Datenbank beinhaltet dabei stets alle Simulationen eines Gesamtsystemmodells. Sie archiviert also die Ergebnisse der Simulationen für alle Missionen und Simulationsläufe. Dies dient der nachfolgenden Auswertung mit MLVisor sowie dem späteren Vergleich der Resultate. Dabei bleibt es dem Nutzer überlassen, nicht mehr benötigte Ergebnisse manuell zu löschen.

Die Simulationen nehmen einen Test des Systems in verschiedenen Missionen vor. Am Ende jeder Simulation stehen ihre Ergebnisse für die Auswertung in der Simulationsdatenbank bereit.

4.2.4 Missionsspezifische Auswertung mit MLVisor

Der letzte Schritt besteht in der Auswertung der durch die Missionen erzeugten Daten. Diese sind in der Simulationsdatenbank hinterlegt und müssen von MLVisor, dem neuentwickelten Programm für die missionsspezifische Auswertung, gelesen werden.

Das Programm liest zunächst die Struktur der Datenbank aus und präsentiert eine nach Missionen geordnete Übersicht der vorhandenen Simulationsdaten. Aus dieser wählt der Nutzer eine Mission aus, deren Daten für die nachfolgende Arbeit mit dem Programm herangezogen werden.

MLVisor unterstützt dabei:

- **Datenvorverarbeitung:**
  Die Datenvorverarbeitung bietet die Möglichkeit, die Daten für die Visualisierung aufzubereiten. Es handelt sich hierbei um diejenige Funktionalität, die vom Modell getrennt wurde. Sie findet nach der Simulation statt und kann dadurch nachträglich angepaßt werden.
- **Visualisierung:**
  Daraufhin werden wahlweise die aufbereiteten oder die Rohdaten genutzt, um die visuelle Präsentation durchzuführen, welche die Voraussetzung für die eigentliche Auswertung darstellt.
- **Halbautomatische Auswertung** (vorgesehen):
  Das hier vorgestellte Konzept erlaubt eine halbautomatische Auswertung der Daten. Halbautomatisch bedeutet in diesem Zusammenhang, daß die Visualisierung an die Ergebnisse der Vorverarbeitung gekoppelt wird. Das ermöglicht es, den Nutzer bei der Auswertung zu entlasten, indem die Zahl der zu interpretierenden Daten reduziert wird. Diese Vorgehensweise bietet sich für den einfachen Test von Kurven gegenüber System- und Entwurfsparametern an. Für die Entwurfsparameter kann dadurch festgestellt werden, ob einfache Entwurfsziele erreicht wurden.

  Ein Entwurfziel für die Steuerung des bereits erwähnten Tischroboters könnte in einer maximalen Zeitspanne für das Erreichen des Zielpunkts bestehen. Dafür wird ein Entwurfsparameter t_{max} definiert. In der Auswertung kann die Endzeit der Simulation mit diesem Parameter verglichen werden. Ist die Zeit größer als t_{max}, wird der Nutzer explizit darauf hingewiesen, daß das Entwurfsziel in dieser Mission verfehlt wurde.

  Für Systemparameter kann auf diese Art ermittelt werden, ob die Simulation korrekt verlaufen ist.

  Ein Systemparameter des Tischroboters könnte die maximal Geschwindigkeit v_{max} sein. In diesem Fall stellt der Test sicher, daß keine Geschwindigkeiten größer als v_{max} auftreten.

In Abhängigkeit von den Ergebnissen dieses Vergleichs kann die Visualisierung dahingehend angepaßt werden, daß nur im Falle einer Verletzung der System- oder Entwurfsparameter eine Darstellung stattfindet.

Die Auswertung erfolgt, indem die visualisierten Ergebnisse durch den Nutzer interpretiert werden. In ihrem Resultat steht eine Einschätzung des Systemdesigns im Hinblick auf die definierten Missionen zur Verfügung. Auf ihrer Basis kann der Nutzer Schlüsse bezüglich notwendiger Änderungen des Designs ziehen oder das Design bestätigen.

Änderungen werden in das mit Markern versehene Entwurfsmodell eingepflegt oder aber an den Systemparametern vorgenommen. Anschließend muß das Design erneut mit den Missionen überprüft werden. Dazu werden neue, konfigurierte Simulationsmodelle mit MLEditor erzeugt und diese mit MLDesigner simuliert. Da die ursprünglichen Simulationsdaten weiterhin in der Datenbank vorhanden sind, können so die Auswirkungen der Änderungen untersucht und mit den ursprünglichen Werten verglichen werden.

4.3 Zusammenfassung

Eine Vorgabe für die Umsetzung des in dieser Arbeit vorgestellten Konzepts für die Simulation im Rahmen des missionsbezogenen modellgestützten Entwurfs war der Einsatz des Entwurfsprogramms **MLDesigner**. Das dabei verfolgte Ziel bestand in der Schaffung einer Arbeitsumgebung, die die einfache Durchführung und Auswertung der missionsbezogenen Simulationen ermöglicht.

Bei MLDesigner handelt es sich um ein Simulationsprogramm, das für den Systementwurf und die Systemanalyse auf Missionsebene entwickelt wird. Im Hinblick auf die im theoretischen Konzept definierten vier Phasen des missionsbezogenen modellgestützten Entwurfs mobiler automatischer Systeme zeigt die **Analyse** jedoch, daß die MLDesigner-Funktionalität hierfür nicht ausreicht:

Die Modellierung in MLDesigner erfolgt innerhalb einer graphischen Oberfläche auf Basis von Bibliotheken. MLDesigner stellt hierfür eine einfache und funktionale Oberfläche bereit, die ein schnelles und benutzerfreundliches Modellieren erlaubt. Darüber hinaus verfügt es über eine umfangreiche Bibliothek einsatzbereiter, aus mehreren Domänen stammender Blöcke für verschiedenste, grundlegende Funktionen, mittels derer eine Vielzahl von Modellen aufgebaut werden kann.

Die Parametrisierung der Modelle erfolgt im Rahmen der Modellierung durch eine Vorgabe der Werte in Form alphanumerischer Zeichenfolgen (Textstrings). MLDesigner unterscheidet dabei nicht zwischen den identifizierten Parametertypen[47], sondern verwendet einen eigenen Ansatz für die Parametergruppierung, der für die hier verwendeten Modelle jedoch ungeeignet ist.

[47] Konstanten, Missions-, System- und Entwurfsparameter

Die Modellnutzung umfaßt hybride Simulationen und erfüllt damit die im Rahmen des missionsbezogenen modellgestützten Entwurfs an die Simulation gestellte Anforderung.

Für die Auswertung werden während der Modellierung spezielle Blöcke in das Modell eingefügt, die nach Beendigung der Simulation Diagrammdarstellungen der Simulationsergebnisse erzeugen. Hiermit ist der Nachteil verbunden, daß die notwendige Aufbereitung der Ergebnisdaten ebenfalls im Modell stattfindet, wodurch das Modell nicht nur das System, sondern auch Blöcke für die Aufbereitung enthält. Eine missionsspezifische Auswertung wird nicht unterstützt.

Dies zeigt, daß die Stärken von MLDesigner in der Modellierung und Simulation, also den typischen Einsatzgebieten eines Simulationswerkzeuges, liegen. Auch die Möglichkeiten zur Auswertung der Ergebnisse entsprechen den üblicherweise gestellten Anforderungen. Für den in dieser Arbeit gewünschten Einsatzzweck mangelt es jedoch an der notwendigen Unterstützung des Missionskonzeptes. Dies trifft sowohl für die Eingabe und Verwaltung von Missionen, als auch für die missionsspezifische Auswertung zu. Aus diesem Grund wurde ein Konzept für ein **Framework** entwickelt, das speziell für den Entwurf von mobilen automatischen Systemen auf Missionsebene geeignet ist. Es greift auf die Stärken von MLDesigner zurück und ergänzt diese um die benötigte Unterstützung von Missionen. Dazu wurden zwei neuentwickelte Programme realisiert: MLEditor übernimmt die Erstellung und Verwaltung von Missionen, während MLVisor auf die missionsspezifische Auswertung spezialisiert ist.

Um das Missionskonzept mit MLDesigner in Einklang zu bringen, erfolgte im Rahmen des Frameworks eine Trennung zwischen dem Modell und seiner Parametrisierung. Die Modellierung findet, wie beschrieben, in **MLDesigner** statt, wobei sich zwei Änderungen ergeben: Erstens werden die System- und Missionsparameter markiert, zweitens ersetzen spezielle Ausgabeblöcke die bisher vorhandenen. Sie sorgen während der Simulation für die Ausgabe der Simulationsergebnisse in eine Datenbank.

Im nächsten Schritt erfolgt die Parametrisierung mit **MLEditor**. Das Programm liest das MLDesigner-Entwurfsmodell aus und erkennt die markierten Parameter. Für diese bietet es eine nutzerfreundliche, erweiterbare Werteeingabe und eine Verwaltung der Werte in einer Parameterdatenbank. Für die Missionsparameter läßt sich dabei eine beliebige Anzahl von Parametersätzen erzeugen, die jeweils eine Mission beschreiben.

Für die Durchführung der Simulation erstellt MLEditor mit Missionen parametrisierte MLDesigner-Simulationsmodelle, deren Simulation dann mittels MLDesigner durchgeführt wird. Die dabei erzeugten Simulationsergebnisse werden mit Hilfe der neuen Ausgabeblöcke in der Simulationsdatenbank gespeichert. Zusätzlich werden Metadaten hinterlegt, die die Zuordnung der einzelnen Simulationen erlauben. Die Verwendung einer Datenbank trennt dabei die Simulation von der Auswertung und gestattet es so, das Modell strikt von der Aufbereitung loszulösen.

Mit Hilfe von **MLVisor** wird die Auswertung vorgenommen, die aus den Einzelschritten Ergebnisaufbereitung und -darstellung besteht. Das Programm verfügt über eine er-

weiterbare Anzahl von Diagrammen und Vorverarbeitungsalgorithmen. Darüber hinaus unterstützt es die Erstellung von missionsspezifischen Auswertungsoberflächen. Durch die Herauslösung der Darstellung aus dem Modell und die damit verbundene Speicherung der Simulationsergebnisse in einer Datenbank verschiebt sich auch die Datenaufbereitung vom Modell in die Auswertung. Damit ist der Vorteil verbunden, daß die ausschließlich der Simulation dienenden Bestandteile nicht mehr die Interpretation des Modells beeinflussen, was das Modell zu einer besseren Nachbildung der Spezifikation macht.

Durch die Anwendung dieses neuentwickelten Frameworks lassen sich missionsbezogene Simulationen durchführen, wie sie der missionsbezogene modellgestützte Entwurf mobiler automatischer Systeme benötigt.

5 AUV DeepC

Kapitel fünf dieser Arbeit widmet sich den Simulationen im Rahmen des *DeepC*-Projekts. Dabei handelt es sich um die praktische Umsetzung der theoretischen Ausführungen zum missionsbezogenen modellgestützten Entwurf mobiler automatischer Systeme. Die Untersuchungen verwenden das im vorausgegangenen Kapitel vorgestellte Framework rund um das Programm MLDesigner.

In diesem Kapitel wird zunächst das *DeepC*-Projekt mit seinen Zielen und den beteiligten Partnern und anschließend das System *DeepC* (Abschnitt 5.2) vorgestellt. Nach diesen allgemeinen Ausführungen wird im Abschnitt 5.3 auf die Modellierung des Systems eingegangen, welche den ersten Bestandteil der Simulation auf Missionsebene bildet. Der zweite Bestandteil, die Missionen, wird im darauffolgenden Abschnitt 5.4 präsentiert, der sich mit den Parametern des Gesamtsystemmodells beschäftigt. Abschließend werden im Abschnitt 5.5 die erreichten Ergebnisse diskutiert.

5.1 DeepC-Projekt

5.1.1 Ziel

Ziel des *DeepC*-Projektes ist die Entwicklung eines autonomen Unterwasserfahrzeugs für große Tiefen und eine langen Einsatzdauer. Es handelt sich bei ihm um eine durch das Bundesministerium für Bildung und Forschung (BMBF) geförderte Kooperation von acht Partnern, von denen jeder einen speziellen Teilaspekt des Projektes bearbeitet ([ATL04]).

Die wichtigsten Zielvorgaben des *DeepC*-Projektes stellt Tabelle 5.1 vor. Durch sie hebt sich das AUV *DeepC* von vielen anderen Fahrzeugen ab, was ein Vergleich mit konkurrierenden Fahrzeugen, wie er bereits im Abschnitt 2.5 vorgenommen wurde, zeigt. Deutlich ist in diesem Zusammenhang die Ausrichtung von *DeepC* auf die Erreichung von größeren Tauchtiefen (bis 4000 m) und einer längeren Einsatzdauer (bis 40 h) erkennbar. Daneben ist ein hoher Nutzlastanteil vorgesehen.

Um die oben genannten Zielvorgaben zu erreichen, setzt *DeepC* auf neueste Technologien. Ein Beispiel hierfür ist der Einsatz von Brennstoffzellen, die, verglichen mit herkömmlichen Batterien, eine wesentlich höhere Energiedichte besitzen und damit bei gleicher Masse eine längere Einsatzdauer ermöglichen.

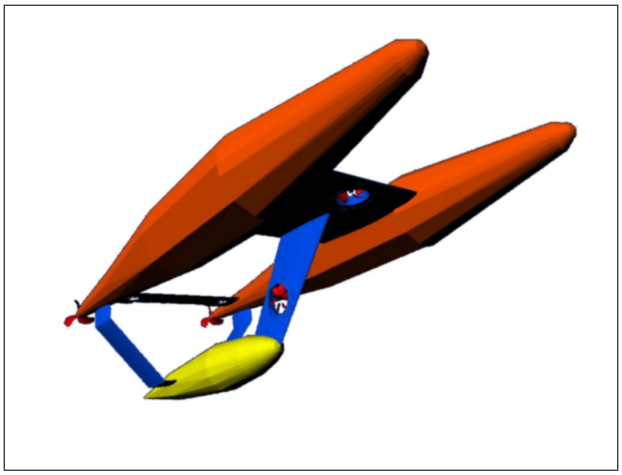

Abbildung 5.1: Autonomes Unterwasserfahrzeug *DeepC*

Tauchtiefe	Mission	4000 m
	Maximum	6000 m
Geschwindigkeit	Mission	4 kn (0,5 kn rückwärts)
	Maximum	6 kn
Einsatzdauer		40 h (bei 4 kn)
Fahrstrecke		ca. 400 km
Nutzlast		300 kg
Gewicht (in Luft)		2,4 t
Druckhüllenmaterial		CFK

Tabelle 5.1: Systemkenndaten des AUV *DeepC*

5.1.2 Partner

Die Zielvorgaben stellen hohe Anforderungen an die Entwicklung des AUV. Um diese realisieren zu können, erfolgt die *DeepC*-Entwicklung durch ein deutsches, interdisziplinäres Konsortium aus Industrie und Wissenschaft mit den folgenden spezialisierten Firmen und Einrichtungen:

- **ATLAS ELEKTRONIK GmbH, Bremen:**
  Projektkoordination und Aktive Autonomie
- **ATI Küste GmbH, Rostock:**
  Energieanlagen
- **AIR Fertigung-Technologie GmbH, Hohen Luckow:**
  Hülle, Manövrieranlagen und Trimmung
- **ENITECH Energietechnik-Elektronik GmbH, Rostock:**
  Energiesystem, Motoren

- **Zentrum für Sonnenenergie u. Wasserstoff-Forschung, Ulm:**
  Brennstoffzelle
- **Technische Universität Ilmenau:**
  Fahrzeugführung und Mission Level Design
- **Universität Karlsruhe:**
  Virtuelle Realität
- **OSAE-Offshore Survey and Engineering GmbH, Bremen:**
  Nutzlast.

Die Auflistung weist zusätzlich zu den beteiligten Firmen beziehungsweise Einrichtungen die verschiedenen Spezialgebiete aus, die diese bei der Entwicklung des Fahrzeugs übernehmen. Dies macht deutlich, daß es sich bei *DeepC* um ein komplexes System handelt, dessen Entwurf eine große Herausforderung darstellt.

An der Technischen Universität Ilmenau werden insgesamt vier Arbeitspakete bearbeitet:

- **Machine Lerning:** [KO03]
  Erzeugung von Strukturen und Lernalgorithmen für das Arbeitspaket EvasionAndIdentification (Ausweichen) auf Basis von Fuzzy- und Neurotechnologie
- **EvasionAndIdentification:** [Eic02, Eic03]
  Softwaremodul, das die Führung des Fahrzeugs im Fall des Ausweichens und während Identifizierungsmanövern übernimmt
- **Mission Replaning:** [Pfü03a, Pfü03b]
  Softwaremodul für die Manöverumplanung im Falle von Umwelteinflüssen (z.B. zu starke Strömung) oder internen Zuständen (z.B. Sensorausfall, nicht ausreichende Energiereserven)
- **Mission Level Design:** [Lie02, LZ04]
  Unterstützung des Systementwurfs durch Untersuchungen im Sinne des missionsbezogenen modellgestützten Entwurfs.

5.1.3 Arbeitspaket Mission Level Design

Beim Arbeitspaket Mission Level Design handelt es sich um eine praktische Umsetzung der Gedanken aus dem dritten Kapitel für das *DeepC*-Projekt. Dabei stellt der missionsbezogene modellgestützte Entwurf eine Anwendung des Mission Level Designs auf mobile automatische Systeme dar[48].

[48] Aus diesem Grund werden die Begriffe missionsbezogener modellgestützten Entwurf und Mission Level Design synonym verwendet. Bezeichnet werden damit immer die Betrachtungen gemäß Kapitel drei.

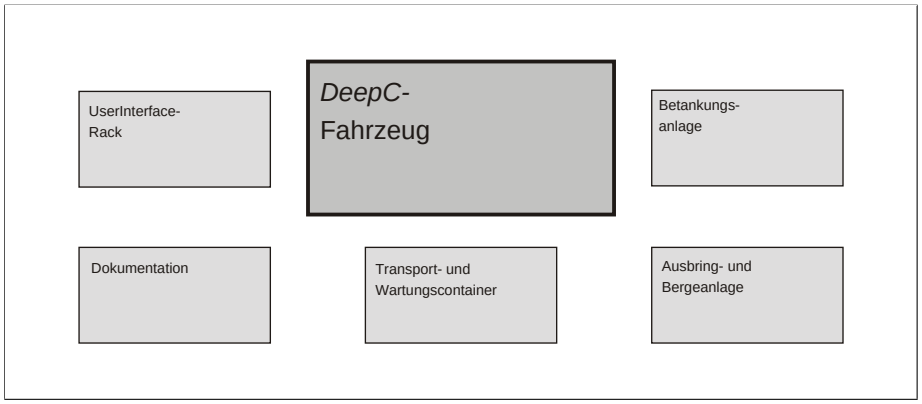

Abbildung 5.2: Gesamtsystemübersicht für das *DeepC*-System

Ziel des Arbeitspakets ist die Unterstützung des Entwurfs durch den Test des Designs auf Missionsebene. Die dafür nötigen missionsbezogenen Simulationen sollen mittels eines Gesamtsystemmodells das zukünftige Systemverhalten nachbilden. Um dies leisten zu können, müssen im Rahmen des Arbeitspakets ein Gesamtsystemmodell des *DeepC*-Systems sowie entsprechende Missionen erstellt werden.

Zum Verständnis der nachfolgenden Ausführungen sind einige kurze Vorbemerkungen hilfreich. Erstens wurden die hier beschriebenen Arbeiten erst begonnen, als die Spezifikation bereits abgeschlossen war. Zudem standen für die Modellierung keine Simulationsumgebung für Simulationen auf Missionsebene und keine geeigneten, vorgefertigten Komponentenmodelle für den Einsatz im Gesamtsystemmodell zur Verfügung. Dementsprechend geben die nachfolgend beschriebenen Arbeiten keinen idealtypischen Verlauf, wie ihn Abschnitt 3.2.2 formuliert hat, wieder, wenngleich sich in ihnen viele charakteristische Gesichtspunkte erkennen lassen.

5.2 Systemübersicht

5.2.1 DeepC-System

Das *DeepC*-System umfaßt nicht nur das eigentliche Fahrzeug, sondern auch eine Reihe weiterer Bestandteile, wie Abbildung 5.2 verdeutlicht. Unter ihnen ist zunächst die für den Transport und die Wartungsarbeiten notwendige Technik zu nennen. Desweiteren sind eine Vorrichtung zum Ausbringen und Bergen des Fahrzeugs sowie eine Betankungsanlage vorhanden. Vervollständigt wird das Gesamtsystem durch ein User Interface Rack zur Inbetriebnahme des Fahrzeugs sowie durch die Dokumentation.

Für die Modellierung im Rahmen des missionsbezogenen modellgestützten Entwurfs liegt das Zentrum des Interesses jedoch auf dem eigentlichen Fahrzeug. Die übrigen Systembestandteile stellen in dieser Sichtweise lediglich Randbedingungen dar, die für den Betrieb

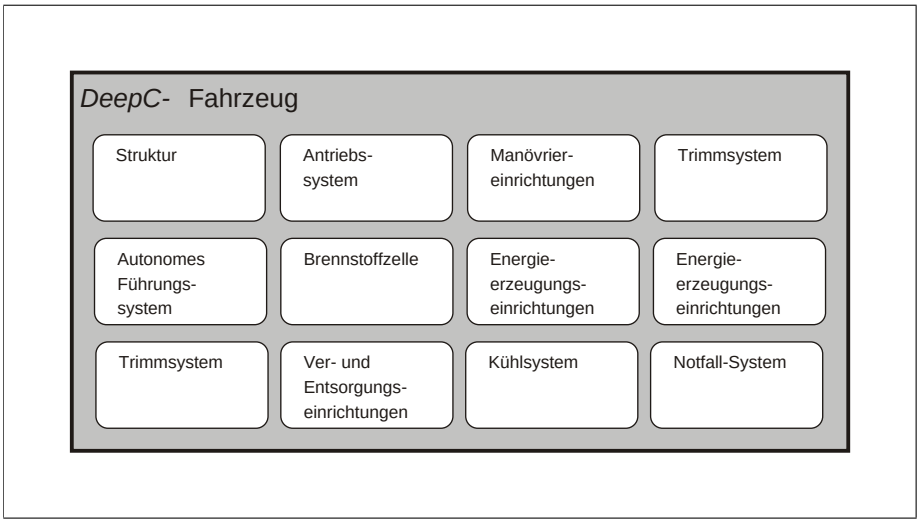

Abbildung 5.3: *DeepC*-Systemteile

des Fahrzeugs benötigt werden. Aus diesem Grund werden sie bei der nachfolgenden Modellierung vernachlässigt.

5.2.2 Aufbau des Fahrzeugs

Das *DeepC*-Fahrzeug besteht aus verschiedenen Subsystemen, die, wie zu Anfang dieses Kapitels erwähnt, von den beteiligten Projektpartnern entwickelt werden. Konkret unterteilt es sich in zwölf separat entwickelte Komponenten (siehe Abbildung 5.3), die im Folgenden überblicksmäßig vorgestellt werden sollen.

- **Struktur:**
  Das als *Struktur* bezeichnete Subsystem bildet die Hülle, also das Grundgerüst des AUV, in das alle anderen Subsysteme eingebaut werden. Hierbei kommt ein neuartiges Konzept für das Hüllendesign zum Einsatz. Es besteht aus zwei modularen Rümpfen, welche die Antriebseinheiten beherbergen, sowie einem separaten Modul für die Nutzlast. Alle drei Hüllensegmente sind durch Streben miteinander verbunden. Abbildung 5.1 bietet eine graphische Darstellung der *DeepC*-Körperstruktur. Deutlich lassen sich die beiden identischen Antriebseinheiten (oben) und die Nutzlast (untere Mitte) erkennen.

  Eines der Hauptprobleme tieftauchender Systeme ist der enorme Druck, dem sie in großen Wassertiefen ausgesetzt sind. Damit die Hülle diesem Druck standhalten kann, sind bei der Verwendung herkömmlicher Werkstoffe sehr große Wanddicken notwendig. Das hieraus resultierende, hohe Eigengewicht des Fahrzeugs vermindert jedoch den möglichen Nutzlastanteil erheblich. Aus diesem Grund werden für das AUV *DeepC* kohlefaserverstärkte Verbundwerkstoffe (CFK) eingesetzt, die einen hohen Nutzlastanteil bei großen Tauchtiefen sicherstellen.

- **Antriebssystem, Manövriereinrichtungen:**
  Das *Antriebssystem* besteht aus je einem Elektromotor pro Antriebseinheit. Angeordnet im hinteren Teil der Antriebseinheiten, treiben beide Motoren jeweils einen Propeller an. Für die Steuerung des Fahrzeugs sorgen mehrere *Manövriereinrichtungen*: In der hinteren Verstrebung zwischen den Antriebseinheiten befindet sich das Höhenruder. Seitliche Drehungen des Fahrzeugs lassen sich durch die Variation der Drehzahlen beider Propeller erzeugen. Für Manöver im Stand beziehungsweise bei kleiner Fahrt besitzt das Fahrzeug insgesamt vier Thruster[49]. Von diesen sind zwei an der hinteren Querverstrebung angebracht, ein weiterer befindet sich in der vorderen oberen Querverstrebung. Mittels dieser Thruster kann sich das Fahrzeug in der horizontalen Ebene bewegen. Der vierte Thruster ist in der Verbindung von Nutzlastmodul und vorderer Querverstrebung installiert.

- **Brennstoffzelle mit Ver- und Entsorgungssystem, Energieerzeugungssystem:**
  Ein wesentlicher Bestandteil jedes autonomen Unterwasserfahrzeugs ist die Energieversorgung. *DeepC* setzt hierbei auf je einen PEM-*Brennstoffzellen*-Stack pro Antriebseinheit. Eine Brennstoffzelle formt zur Energiegewinnung Wasserstoff und Sauerstoff zu Wasser um. Bei einer PEM[50]-Zelle geschieht dies mittels einer Polymerfolie, die als Trennwand zwischen Anode und Kathode fungiert ([IB]). Der Vorteil von Brennstoffzellen gegenüber herkömmlichen Batterien liegt in der größeren mitführbaren Energiemenge. Auf diese Weise wird eine lange Missionsdauer bei gleichzeitig relativ großer Durchschnittsgeschwindigkeit erreicht. Um Brennstoffzellen betreiben zu können, sind jedoch eine Reihe von Zusatzmodulen notwendig. So wird ein *Ver- und Entsorgungssystem* benötigt, das aus Tanks, Zuleitungen und Steuereinrichtungen besteht. Letztere stellen die mengenmäßig stets korrekte Zuführung von Sauerstoff und Wasserstoff sicher. Da es in großen Tiefen energetisch nicht sinnvoll ist, das durch die Brennstoffzellen erzeugte Wasser aus dem Fahrzeug zu pumpen, wird dieses mitgeführt. Zusätzlich zur Ver- und Entsorgung wird ein komplexes *Energieerzeugungssystem* benötigt. Es besteht aus Energiewandlungs- und Energieverteilungsbaugruppen sowie aus Speicherbatterien. Letztere sind zur kurzzeitigen Pufferung nötig, um die Brennstoffzellen vor Lastspitzen zu schützen.

- **Trimm- und Kühlsystem:**
  Von außen nicht erkennbar, besitzt das AUV sowohl ein *Trimm-*, als auch ein *Kühlsystem*. Während das Erstgenannte dem Ausgleich von Gewichtsunterschieden und der Erzeugung eines Nullauftriebs dient, leitet das Kühlsystem die von elektrischen Komponenten und Brennstoffzelle erzeugte Wärme ab.

- **Notfallsystem:**
  Um den Verlust des Fahrzeugs zu verhindern, verfügt das AUV *DeepC* über ein

[49] Thruster (dt.: Stoßstrahlruder): Ihr Aussehen gleicht einem inversen Propeller, d.h. die Rotorblätter befinden sich auf einem Ring und sind in die Mitte ausgerichtet. Die von ihnen erzeugten Strömungen werden zum Manövrieren verwendet.

[50] Polymer Electrolyte Membrane

Notfallsystem. Es besteht aus einem letztinstanzlichen Mechanismus zum stromlosen Notauftauchen. Zusätzlich enthalten auch die anderen Subsysteme Vorkehrungen für Notfälle. So bestehen beispielsweise für die fahrzeuginterne Energieverteilung Möglichkeiten zur Havarieschaltung, desweiteren findet eine permanente Überwachung der Sensoren und Antriebseinheiten, ein sogenanntes Monitoring, statt.

- **Autonomes Führungssystem, Energiemanagementsystem:**
  Das *Autonome Führungssystem* bildet das Herzstück des autonomen Verhaltens. Es enthält unter anderem diverse Sensorik zur Erkennung der Umwelt. Für die Unterwasserlangzeitnavigation werden die Daten eines inertialen Navigationssystems und mitgeführte Seekarten verwendet. Eine exakte Aktualisierung wird durch einen GPS[51]-Empfänger erreicht, der aber nur an der Wasseroberfläche nutzbar ist. Unter Wasser stellt die Hinderniserkennung sicher, daß Objekte in der Umgebung von der Steuerung wahrgenommen werden. Dazu verwendet sie die Daten eines Forward-Looking-Sonars. Die Steuerung des AUV erfolgt durch ein hierarchisches Softwarekonzept, welches auf zwei Rechnerclustern läuft. Das Ausweichen zur kollisionsfreien Abarbeitung von Missionen wird von einem spezialisierten Modul berechnet. Daneben stützt sich die Autonomie des Fahrzeugs unter anderem auf das Mission Monitoring, das für die Überwachung des Missionsverlaufs sorgt, sowie auf das Mission Replanning, welches eine Umplanung der aktuellen Mission vornehmen kann. Derartige Umplanungen können dabei nicht nur von Hindernissen oder dem Ausfall von Komponenten initialisiert werden, sondern auch vom *Energiemanagementsystem*, das über die Energiereserven wacht.
- **Nutzlastmodul:**
  Das *Nutzlastmodul* dient der Aufnahme des Spezialequipments des Nutzers. Da es ein eigenes Modul der Körperstruktur bildet, kann es beliebig ausgetauscht werden. Dieser Aufbau ermöglicht es, in Abhängigkeit vom jeweiligen Einsatzzweck verschiedenste Module mitzuführen.

5.3 Modellierung des Gesamtsystemmodells

5.3.1 Überblick über das Gesamtsystemmodell

Der nachstehende Abschnitt beschreibt das im Rahmen der vorliegenden Arbeit neuentwickelte Gesamtsystemmodell für das AUV *DeepC* in seinem Aufbau. Zunächst wird eine graphische Übersicht über das Modell gegeben, die im Anschluß anhand von Screenshots dem realem Modell gegenübergestellt wird.

[51] Global Positioning System

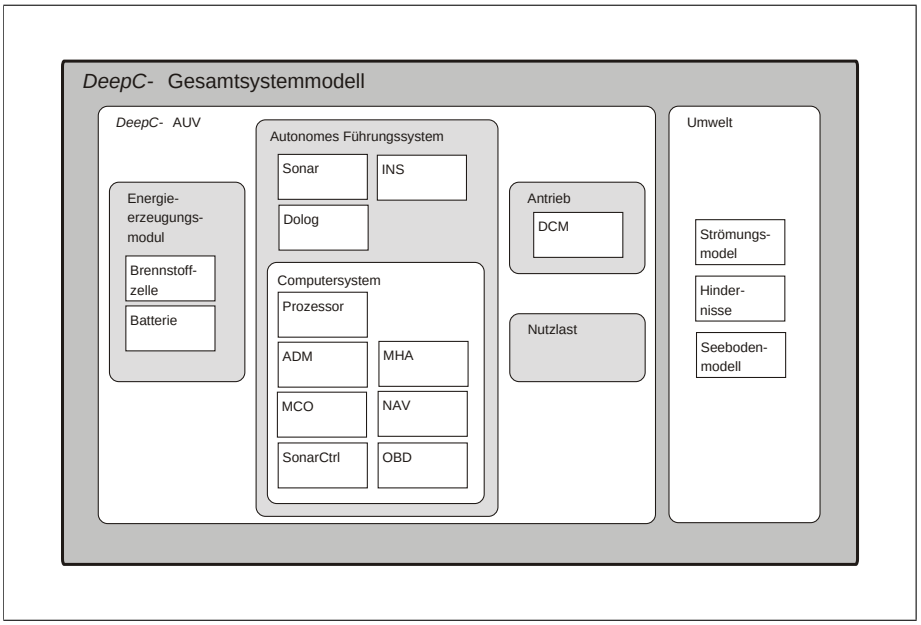

Abbildung 5.4: Gesamtsystemmodell (Übersicht)

Die Modellierung des Gesamtsystemmodells beruht dabei auf den im dritten Kapitel dieser Arbeit erarbeiten theoretischen Grundlagen und wurde mit MLDesigner durchgeführt.

Graphische Darstellung

Abbildung 5.4 zeigt eine graphische Übersicht des Gesamtsystemmodells, indem sie die modellierten Komponenten in ihrer Hierarchie darstellt. Diese entspricht der allgemeinen, im Abschnitt 3.1.2 vorgestellten Struktur mobiler autonomer Systeme. In der graphischen Darstellung werden die Komponenten durch Blöcke symbolisiert. Befindet sich ein Block innerhalb eines anderen, handelt es sich bei ihm um ein Subsystem. Damit gilt:

- **(n)-Ebene:**
  Auf der obersten Ebene besteht das Gesamtsystemmodell aus dem AUV und seiner Umwelt.

- **($n-1$)-Ebene:**
  Auf der ($n-1$)-Ebene untergliedert sich sowohl das Fahrzeug, als auch die Umwelt in Module.

  Für das Erstgenannte lassen sich vier Teilmodule unterscheiden: der Antrieb, das Autonome Führungssystem, das Nutzlast- sowie das Energieerzeugungsmodul. Wie der Vergleich mit Abbildung 5.3 auf Seite 85 zeigt, entsprechen diese dabei den Modellen der entsprechenden Systembestandteile: Das Energieerzeugungsmodul faßt die Systemteile Brennstoffzelle und Energieerzeugungseinrichtungen zusammen, während die Manövriereinrichtungen und das Antriebssystem den Antrieb

bilden. Zunächst nicht modelliert wurden das Kühlsystem, das Notfallsystem und das Trimmsystem. Der Systembestandteil Struktur wird indirekt an verschiedenen Stellen im Modell abgebildet.

Die Umwelt umfaßt drei Modelle, nämlich die Hindernisse, ein Strömungsmodell und ein Modell des Seebodens.

- **$(n-2)$-Ebene:**
  Jedes der Fahrzeugteilmodule auf der Ebene $(n-1)$ besteht seinerseits aus Subkomponenten.

  Das Energieerzeugungsmodul beinhaltet die Brennstoffzelle und die Pufferbatterien.

  Für den Antrieb existiert lediglich eine DCM[52] genannte Komponente, die ein Modell des geregelten Fahrzeugverhaltens beinhaltet. In diese sind die Funktionen der Manövriereinrichtungen integriert, so daß hierfür kein separates Teilmodul modelliert werden mußte.

  Das autonome Führungssystem besteht aus der Sensorik mit Sonar, INS[53] und Dolog sowie dem Rechnersystem, auf welchem die Steuersoftware läuft.

- **$(n-3)$-Ebene:**
  Auf dieser Ebene untergliedert sich das Computersystem in seine Hardware (Prozessor) und die einzelnen Softwaremodule[54], die auf der Hardware laufen (ADM, MHA, MCO, NAV, OBD und SonarCtrl)[55].

Screenshots

Um diese abstrakte Beschreibung anschaulich zu illustrieren, stellen die folgenden Abbildungen 5.5 und 5.6 das in MLDesigner realisierte Gesamtsystemmodell exemplarisch graphisch dar.

- **Oberste Ebene:**
  Abbildung 5.5 präsentiert die oberste Ebene das Systems. Neben den beiden Modulen mit der Umwelt und dem autonomen Unterwasserfahrzeug besitzt es eine Primitive für die Ausgabe der Ergebnisdaten in eine Datenbank und ein Memory zur Speicherung des zugehörigen Zeigers.[56]

[52] Dynamic Controlled Motion
[53] Inertial Navigation System
[54] Der Term Softwaremodul bezeichnet einen modularen Teil der *DeepC*-Softwarestruktur und nicht ein MLDesigner-Modul.
[55] Näheres hierzu findet sich im Abschnitt 5.3.2.3 ab Seite 94.
[56] Näheres hierzu findet sich im Abschnitt 5.3.3.8 ab Seite 132.

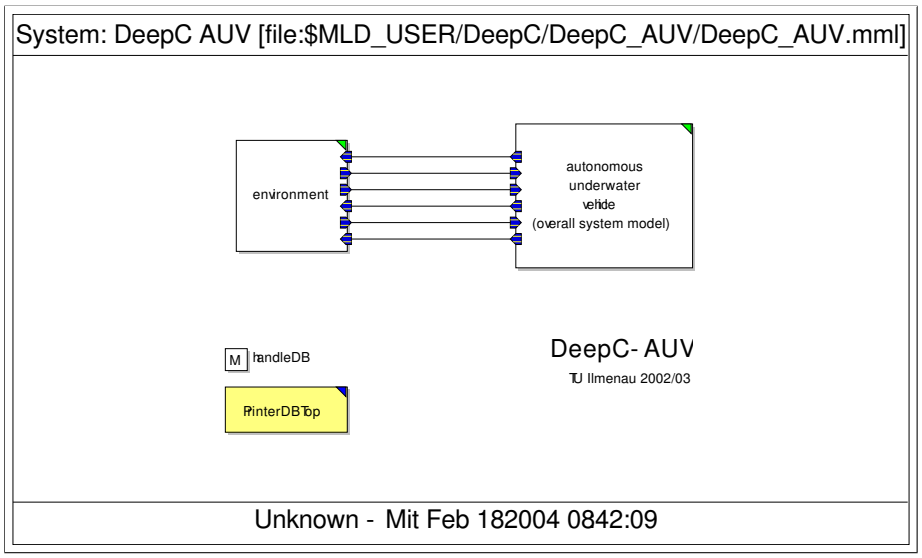

Abbildung 5.5: Oberste Modellebene (in MLDesigner)

- **Computersystem:**
  Das Modell des Computersystems ist in Abbildung 5.6 dargestellt. Es enthält zwei Computer und acht Softwaremodule. Letztere sind sowohl mit einem der Computer, als auch untereinander verbunden. Zusätzlich dienen weitere Blöcke der Datenbankausgabe der Simulationsergebnisse und der Verzweigung von Verbindungen.

Vergleicht man die beiden Beispiele mit ihrer Darstellung in der graphischen Übersicht (Abbildung 5.4), so erkennt man, daß die einzelnen wesentlichen Bestandteile vollständig vorhanden sind.

5.3.2 Sichten

Bevor im folgenden Abschnitt 5.3.3 die einzelnen Komponenten vorgestellt werden, ist es für das Verständnis vorteilhaft, das Gesamtsystemmodell aus bestimmten Sichtweisen heraus zu erläutern. Damit eine strukturierte Erklärung des Gesamtverhaltens erreicht werden kann, heben einzelne Sichten jeweils einen Modellaspekt[57] hervor und beschreiben die zugehörigen Komponenten in ihrer Zusammenarbeit. Dabei ist jedoch zu berücksichtigen, daß sich die Sichten gegenseitig überlagern und insofern nicht unabhängig voneinander sind.

[57] Vgl. Aspekte des missionsbezogenen modellgestützten Entwurfs mobiler automatischer Systeme im Abschnitt 3.1.1 ab Seite 39.

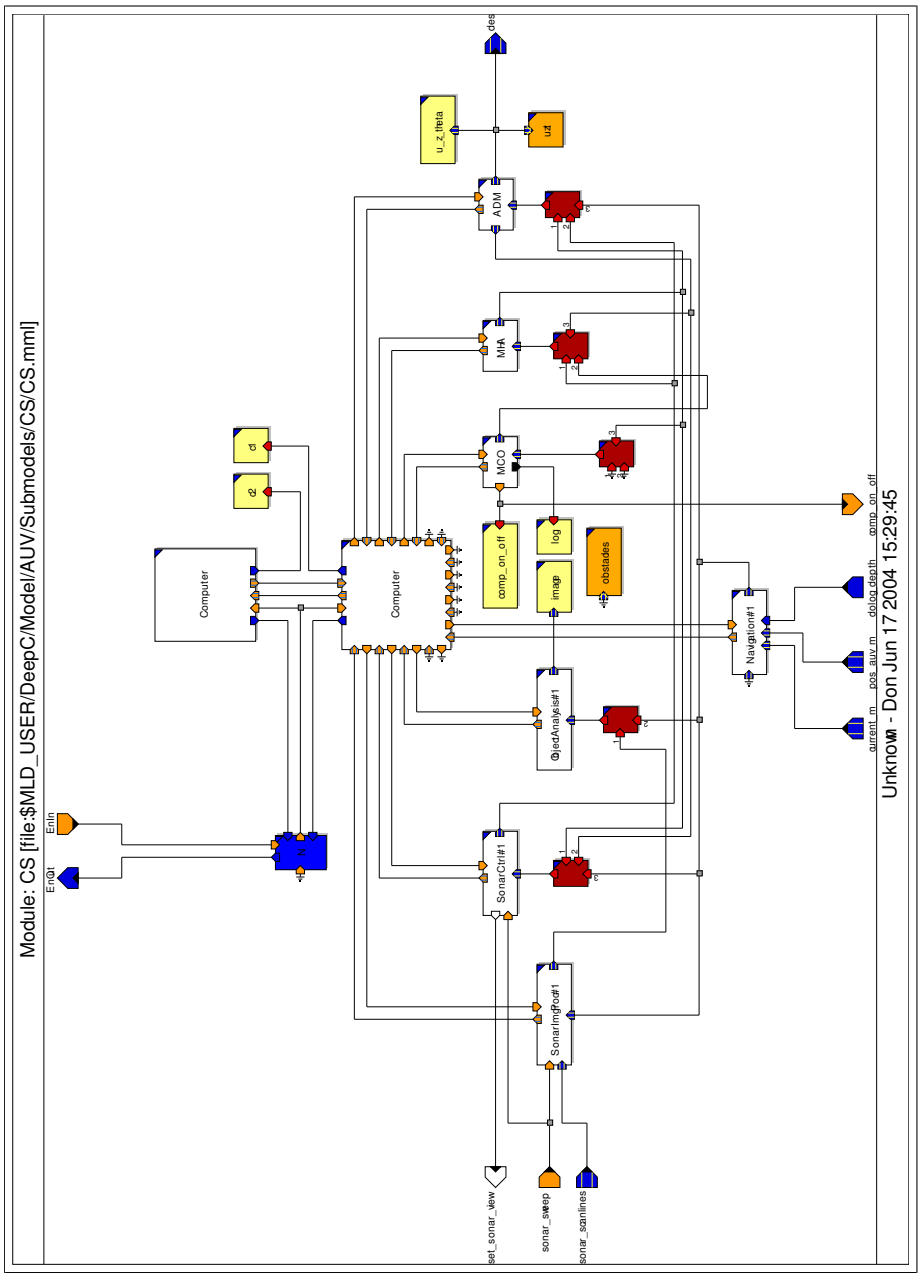

Abbildung 5.6: Computersystem (in MLDesigner)

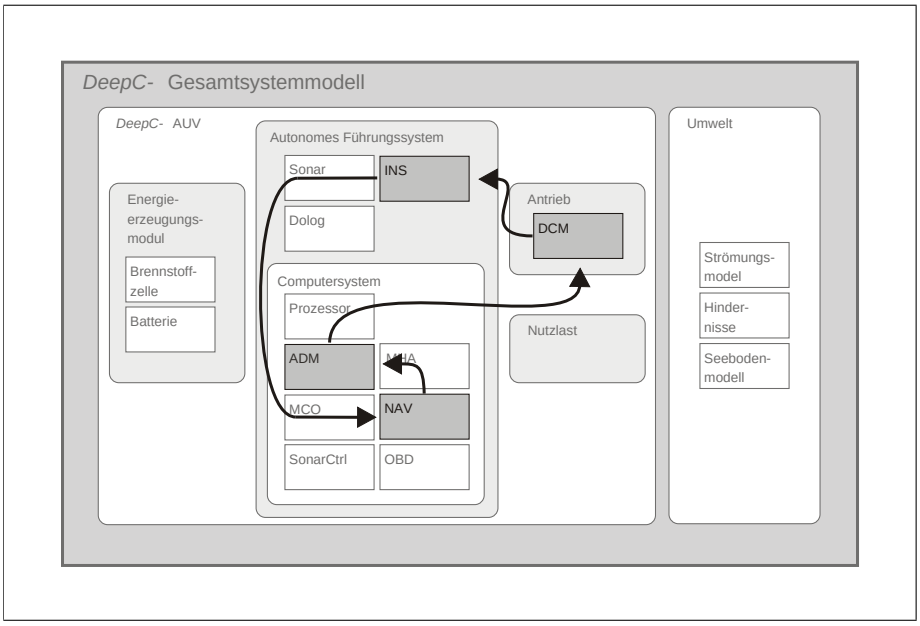

Abbildung 5.7: Teilsicht: Antrieb

5.3.2.1 Bewegung

Ein zentraler Bereich für die Modellierung eines AUV ist die Nachbildung seiner Bewegung im Raum. Das Fahrzeug soll sich im dreidimensionalen Raum mit allen sechs Freiheitsgraden bewegen können und dabei die charakteristische Dynamik aufweisen.

Abbildung 5.7 stellt die hierbei beteiligten Blöcke sowie die Verbindungen zwischen ihnen dar. Sie nutzt dazu als Basis die schon verwendete graphische Darstellung des Gesamtsystemmodells[58]. In dieser werden nun diejenigen Bestandteile hervorgehoben, die in Verbindung mit der Bewegung stehen. Hierbei handelt es sich um die Blöcke DCM (geregeltes Fahrzeugverhalten), INS (inertiales Navigationssystem), NAV (Navigation) und ADM (Basisregler). Die folgende Auflistung zeigt deren schrittweises Zusammenspiel auf:

1. Die Dynamik wird von einem Modell des geregelten Fahrzeugverhaltens erzeugt. Es erhält die Steuersignale Kurs, Geschwindigkeit und Tiefe und erzeugt daraus die aktuelle Fahrzeugposition.
2. Das Fahrzeug nutzt für seine Positionbestimmung ein inertiales Navigationssystem. Im Modell werden die vom Fahrzeug kommenden, exakten Positionsdaten zum INS geleitet. Dieses generiert aus ihnen die gemessene AUV-Position.
3. Diese Position wird vom INS an das Softwaremodul Navigation übermittelt, das die Daten für andere Softwaremodule zugänglich macht.

[58] Vgl. Abbildung 5.4, Seite 88.

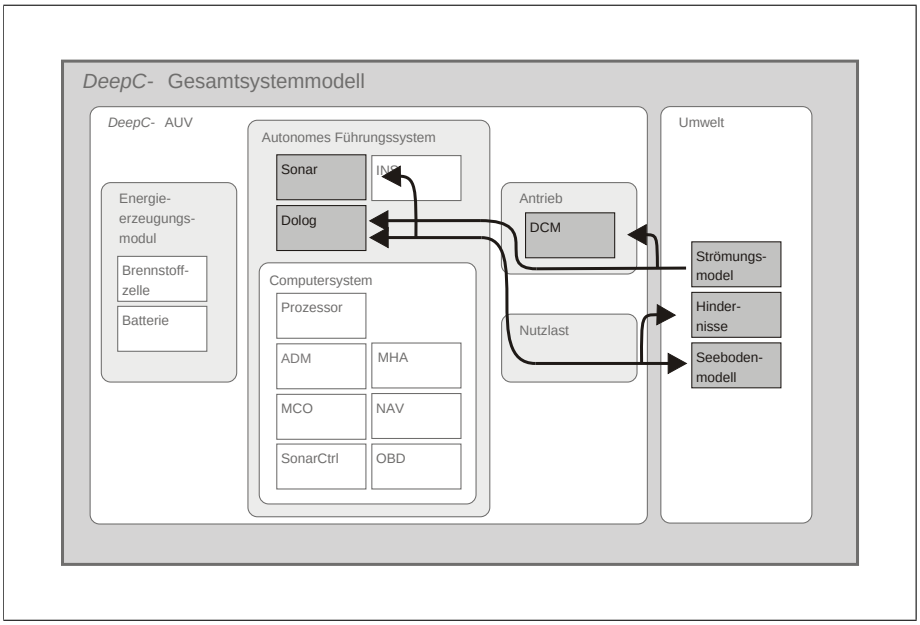

Abbildung 5.8: Teilsicht: Umwelt und Sensorik

4. Auf Basis der aktuell gemessenen Position erzeugt der als Softwaremodul realisierte Basisregler ADM die Steuersignale. Er hält das Fahrzeug auf einem Basisbahnelement, bei dem es sich um eine Linie oder ein Kreissegment handeln kann. ADM gibt die Steuersignale für DCM vor (zurück zu 1.).

5.3.2.2 Umwelt und Sensorik

Daß die Modellierung der Umwelt einen Bestandteil des Gesamtsystemmodells bilden muß, wurde in Abschnitt 3.1.1 herausgearbeitet. Gleichzeitig hierzu muß jedoch auch die Sensorik zum Wahrnehmen der Umwelt nachgebildet werden.

Abbildung 5.8 hebt die an diesem Vorgang beteiligten Blöcke hervor.

- Hierbei sind erstens die Hindernisse, das Bodenmodell, das Sonar und das Dolog zu nennen. Von ihnen sind die Hindernisse und das Bodenmodell zur Umwelt zu zählen, während das Sonar und das Dolog der Sensorik zuzurechnen sind.

  Die Zusammenarbeit zwischen der Sensorik und der Umwelt geschieht über Anfragen (Pings). Diese bilden jeweils einen Suchstrahl nach und werden sowohl vom Sonar, als auch vom Dolog an die Umwelt gesendet. Dort werden sie entgegengenommen und an das Seebodenmodell und das Modell der Hindernisse weitergeleitet. Die Antworten werden ausgewertet und an dasjenige Sensormodul weitergereicht, welches die Anfrage gestellt hat. Durch dieses Vorgehen wird es möglich, auf einfache Art und Weise weitere Umwelteinflüsse hinzuzufügen.

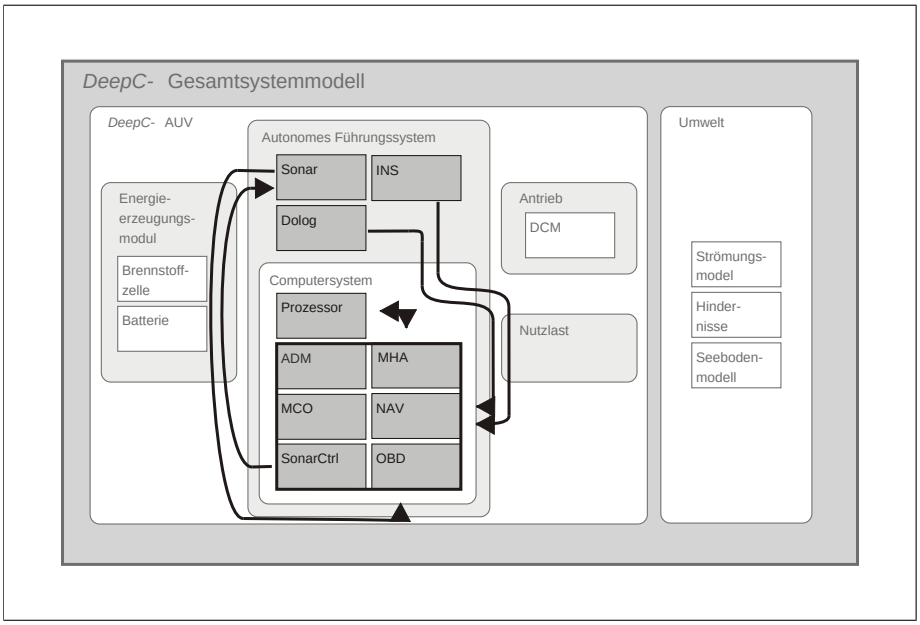

Abbildung 5.9: Teilsicht: Computer

- Ein zweiter Umwelteinfluß wird durch das Strömungsmodell modelliert. Es erzeugt für jeden Punkt im Raum einen Geschwindigkeitsvektor. Dieser vom Dolog gemessene Vektor wird bei der Generierung der Position in DCM berücksichtigt und beeinflußt so die Bewegung des Fahrzeugs im Raum.

5.3.2.3 Computersystem

Das Computersystem ist ein wesentlicher Bestandteil des Systems, da auf ihm die Softwareanwendungen zur Erzeugung der maschinellen Intelligenz laufen. Aus diesem Grund wurde die Softwarestruktur des AUV *DeepC* nachmodelliert. Sie besteht aus verschiedenen Softwaremodulen, die jeweils spezifische Aufgaben erfüllen. Abbildung 5.9 hebt nicht nur die Softwaremodule, sondern auch die dazugehörige Hardware hervor. Letztere modelliert die wichtige Ressource Rechenleistung.

In diesem Kontext lassen sich zwei unterschiedliche Teilsichten unterscheiden:

- Erstens arbeiten Hardware und Software nicht unabhängig voneinander, vielmehr nutzt die Software die vorhandene Hardware. In der Simulation wird dieser Umstand dadurch modelliert, daß die Softwaremodule eine gewisse Rechenarbeit benötigen. Die Ablaufzeit zur Erfüllung dieser Arbeit ist dabei nicht festgeschrieben, sondern von der Hardware und ihrer Auslastung abhängig. Aus diesem Grund besitzt jeder Rechner eine parametrisierbare Menge an Rechenleistung, die prioritätsgesteuert auf die parallelen Anfragen der Softwaremodule verteilt wird. Ist die Arbeit erledigt, werden die Softwaremodule benachrichtigt.

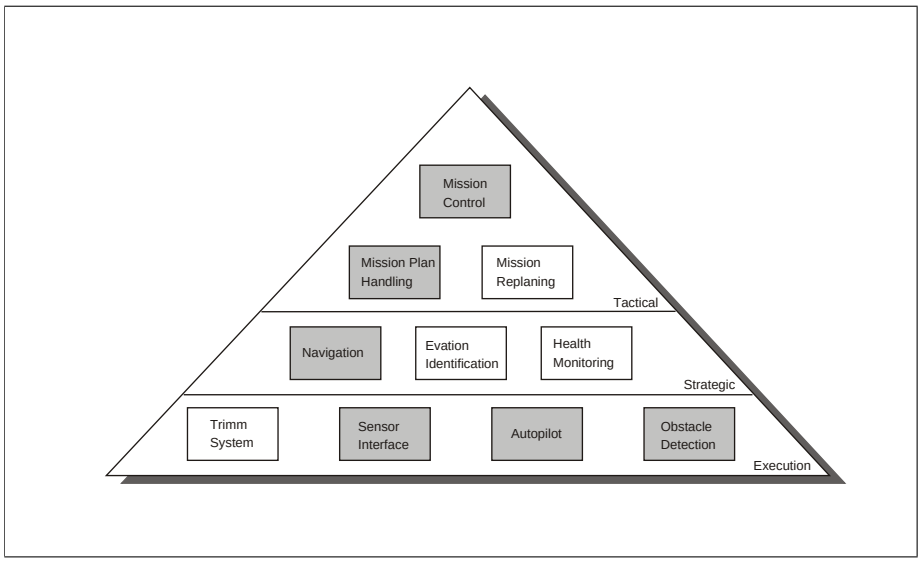

Abbildung 5.10: Vollständige *DeepC* Softwarestruktur (nach [Pfü03a])

- Die zweite Teilsicht betrifft die Zusammenarbeit der Softwaremodule untereinander. Das AUV *DeepC* besitzt eine hierarchische Softwarestruktur, wie sie Abbildung 5.10 zeigt[59].

  An deren Spitze steht die taktische Ebene. Sie wird vom Mission Controller (MCO), Mission Plan Handling (MHA) und dem Mission Replaning gebildet. Der Mission Controller stellt die höchste Entscheidungsinstanz dar und kann insofern als der Kapitän des Fahrzeugs bezeichnet werden. Das Mission Plan Handling verwaltet die einzelnen Elemente des Missionsplanes, welcher den Ablauf jeder Mission steuert, während das Mission Replaning für eine ereignisgesteuerte Umplanung des Missionsplanes verantwortlich ist.

  Die strategische Ebene wird von Softwaremodulen gebildet, die spezielle strategische Aufgaben übernehmen. Navigation (NAV) übernimmt alle im Zusammenhang mit der Navigation nötigen Aufgaben. Das Health Monitoring überwacht das Funktionieren wichtiger Systembestandteile und EvasionAndIdentification steuert das Ausweichen beziehungsweise das Identifizieren von Objekten.

  Auf der untersten, der ausführenden Ebene finden sich Softwaremodule, die direkt mit der Sensorik beziehungsweise Aktorik interagieren. Die Hinderniserkennung (Object Detection, OBD) wertet als Basis der Kollisionsvermeidung die Sonarbilder aus. Der Autopilot (ADM) erzeugt die Steuersignale für den Antrieb, das Sensor Interface übernimmt die Kommunikation mit der Sensorik, während Trimm System für die Trimmung des Fahrzeugs sorgt.

[59] Alle farblich hinterlegten Blöcke sind im Gesamtsystemmodell berücksichtigt.

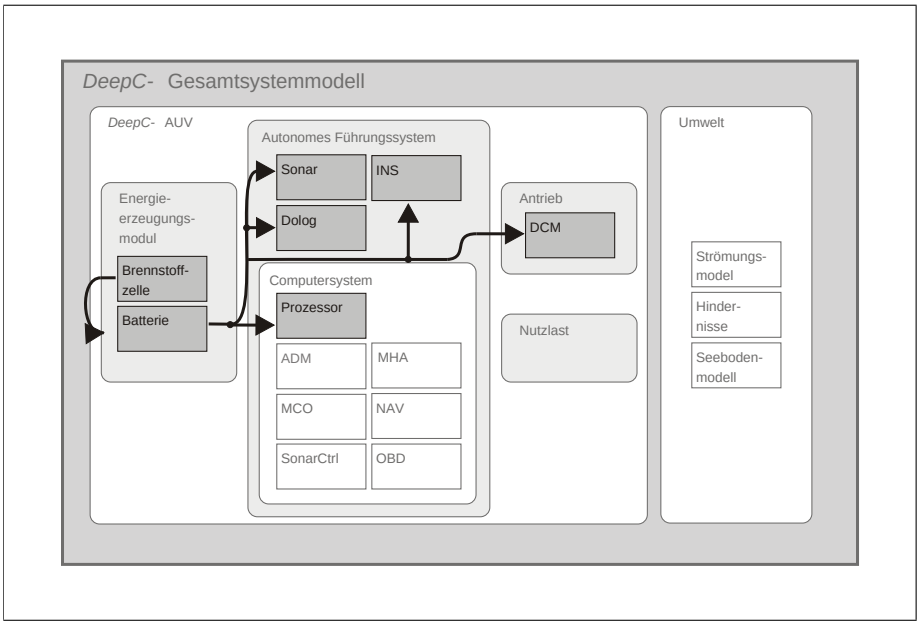

Abbildung 5.11: Teilsicht: Energie

5.3.2.4 Energie

Der Energiekreislauf und die daran beteiligten Komponenten sind in Abbildung 5.11 hervorgehoben. Gespeist wird der Energiekreislauf von den Brennstoffzellen. Diese sind über Pufferbatterien mit den zahlreichen Abnehmern verknüpft, wobei diese Verknüpfung einem Energienetz entspricht.

Als Verbraucher treten unter anderem der Antrieb (DCM), die Prozessoren sowie die gesamte Sensorik (Sonar, Dolog, INS) auf. Einzelne Komponenten sind zuschaltbar oder ihr Energieverbrauch wird lastabhängig modelliert. Dies trifft beispielsweise für die Lampen (zuschaltbar) sowie für den Antrieb zu, welcher in Abhängigkeit von der Geschwindigkeit mehr oder weniger Energie verbraucht (lastabhängig).

5.3.3 Komponenten

Nach der Darstellung der verschiedenen Sichten sollen nachfolgend die einzelnen Komponenten im Detail erläutert werden. Die Vorstellung der Primitiven erfolgt dabei geordnet nach Kategorien.

Die verwendete Syntax bezeichnet die MLDesigner-Primitiven jeweils mit Name. *Methoden()*, *Eingänge*, *Ausgänge* und *Parameter* sind wie in diesem Satz dargestellt hervorgehoben.

Der bei der Modellierung verfolgte Ansatz läßt sich grundsätzlich wie folgt beschreiben: Wann immer möglich, wurde auf Standardprimitiven zurückgegriffen. Bedingt durch die

komplexe Funktionalität des AUV mußten diese jedoch durch zahlreiche eigene Primitiven ergänzt werden. Die auf diese Weise entstandene AUV-Bibliothek stellt ein wesentliches Ergebnis der vorliegenden Arbeit dar. In Fällen, in denen Spezialprimitiven und Module aus Basisprimitiven alternative Varianten darstellen, wurden in der Regel Spezialprimitiven eingesetzt. Dies ist begründet in zwei Vorteilen: Zum einen ist mit der Verwendung von speziellen Primitiven eine geringere Simulationsdauer verbunden, weil ein Teil des Overheads der Simulationsumgebung entfällt[60]. Zum anderen führt es zu einer größeren Übersichtlichkeit des Modells, wenn eine spezielle Primitive eine größere Anzahl von Basisprimitiven ersetzt.

5.3.3.1 Energie

Basisenergiemodell

Hinsichtlich der Modellierung des Energieaspekts waren die folgende Anforderungen zu berücksichtigen:

1. Der Energieaspekt ist nur ein Teilaspekt der Komponenten, über diesen hinaus besitzt jeder Verbraucher eine spezifische Funktionalität.
2. Die Struktur der Modellierung muß der eines realen Energiesystems entsprechen. Dies ist insofern wichtig, als daß nur so die Struktur des Energiesystems im Gesamtzusammenhang richtig zu verstehen ist.
3. Die Lösung soll numerisch nicht zu anspruchsvoll sein, damit die Dauer der Simulationen nicht unnötig verlängert wird.
4. Aufgrund der Tatsache, daß es keine einheitliche Gesamtsystemtaktzeit gibt, muß die Lösung in der Lage sein, mit dem gleichzeitigen Auftreten verschiedener Tastzeiten umzugehen. Für die Modellierung soll die DE-Domäne verwendet werden.
5. Höhere Funktionen eines Energienetzes, wie beispielsweise die redundante Anbindung einzelner Hauptkomponenten, sind zu ermöglichen.

Unter Berücksichtigung dieser Anforderungen wurde die nachfolgend kurz darzustellende Lösung erarbeitet. Ausführlich ist sie in "Modeling of the energy system for the mission level design of the DeepC AUV" beschrieben ([AL03]).

Aus makroskopischer Sicht läßt sich ein einfaches Energiesystem in drei wesentliche Bestandteile untergliedern: die Energiequelle, die Verbindungen und die Verbraucher. Mit diesen Komponenten ist jedoch auch die Erstellung komplexer Verbraucherszenarien realisierbar.

[60] Der Scheduler der Simulationsumgebung muß die Abarbeitung der einzelnen Blöcke steuern. Übernimmt nun eine spezielle Primitive die Aufgabe von 20 allgemeinen, verringert sich die Simulationszeit.

Als Modellansatz für die Energiesimulation wurde in der vorliegenden Arbeit die Energiebilanz gewählt, bei der diskrete Energiemengen über das Energienetz transportiert werden. Der Bezug von Energie aus der Batterie vollzieht sich in diesem Ansatz wie folgt: Fordert ein Verbraucher eine bestimmte Energiemenge aus der Batterie an, so wird eine entsprechende Nachricht über das Energienetz transportiert. Sobald sie die Batterie erreicht, wird die angeforderte Menge vom Batterieinhalt abgezogen. Dieser Vorgang wird nicht von einer Antwortnachricht begleitet, da hierfür eine direkte Identität des Verbrauchers gegeben sein müßte. Zudem würde durch den Versand derartiger Nachrichten ein unnötiger Simulationsaufwand entstehen. Ist die Batterie leer, wird entweder die Simulation beendet oder eine entsprechende Nachricht an alle Verbraucher geschickt. Somit werden in der Richtung Verbraucher - Batterie Energieanforderungen gesendet, während in der Gegenrichtung Statusinformationen transportiert werden.

Alle verwendeten Nachrichten sind, wie Listing 5.1 zeigt, in der Datei `$MLD_USER/DeepC/include/energy_protocol.h` definiert. Sie stellen Statusinformationen über den Zustand des Energienetzes beziehungsweise der Batterie dar.

```
#define ENERGY_BATTERY_EMPTY 0
#define ENERGY_CONNECTION_BROKEN -1
#define ENERGY_REDUNDANT_MODE 10
#define ENERGY_CONNECTION_OK 1
```

Listing 5.1: Verwendete Energienachrichten (energy_protocol.h)

Für die einzelnen Basisbestandteile des Energiesystems existieren fünf Primitiven:

Die Batterie (Battery) stellt einen allgemeinen Energiespeicher dar und weist die Parameter *Capacity*, *Level*, *SampleTime* und *StopSimulation* auf. *Capacity* legt die Größe der Energiemenge fest, *Level* den Füllstand zum Startzeitpunkt, *SampleTime* bewirkt, daß entsprechend der angegebenen Zeit der aktuelle Füllstand am Ausgang *Out* vermerkt wird. Durch den Parameter *StopSimulation* kann sichergestellt werden, daß die Simulation beendet wird, sobald die Batterie leer ist. Die Ankopplung an das Energienetz geschieht über den Eingang *InDS* und den Ausgang *Status*. Für eine redundante Anbindung existieren analoge Gegenstücke (*InDSR* und *StatusR*).

Die Energieverbindung (EnNode) ist ein Teil des Energienetzes. Die Stromleitungen werden in der Simulation durch die Verbindungslinien zwischen den Ein- und Ausgängen der Primitiven repräsentiert. Die Abzweigungen beziehungsweise Aufspaltungen von Stromleitungen werden mittels EnNode-Primitiven modelliert. Hierzu besitzen diese multiple Eingänge *In* für die Entgegennahme der Energieanforderungen, die dann über den Ausgang *Out* in Richtung Batterie weitergeleitet werden. In der Gegenrichtung liegen die Informationen am Eingang *Inf_get* an und werden über den Ausgang *Inf_put* weitergegeben. Jeder EnNode besitzt zusätzlich den Parameter *Broken*, welcher den Startzustand der Verbindung angibt. Während der Simulation kann über den Eingang *broken* eingestellt werden, ob die Verbindung intakt oder gestört ist. Zu diesem Zweck können an diesen die Statusnachrichten ENERGY_CONNECTION_OK oder ENER-

GY_CONNECTION_BROKEN gesendet werden. Der Status wird über *Inf_put* an die angeschlossenen Verbraucher oder Subnetze ausgegeben. Ist die Verbindung gestört, werden alle Nachrichten an *In* und *Inf_get* ignoriert.

Für die Modellierung der Verbraucher wurden die beiden Basisprimitiven EnBaseDevice und EnBaseDeviceRS geschaffen. Letztere ist von der Primitive RepeatStar abgeleitet. Von ihr erbt sie die Fähigkeit, sich selbst in einem durch den Parameter *SampleTime* festlegbaren Abstand aufzurufen. Beide Primitiven besitzen jeweils den Parameter *Wattage*, den Eingang *EnergyIn* sowie den Ausgang *EnergyOut*: Über *EnergyIn* erlangen sie die Statusinformationen des Energienetzes, während sie über *EnergyOut* ihre Energieanforderungen absetzen. Die Höhe dieser Anforderungen wird dabei durch *Wattage* beeinflußt. Da die Energie nur einer der Aspekte einer Komponente ist, wurde eine Lösung geschaffen, die es erlaubt, diesen Aspekt durch Vererbung auf die Komponenten zu übertragen. Die Primitiven EnBaseDevice und EnBaseDeviceRS beinhalten neben Parametern, Ein- und Ausgängen auch Methoden, welche die Arbeit mit dem Energieaspekt unterstützen. So verfügen sie beispielsweise über die Methode *getEnergy()*, mit der Energieanfragen ausgelöst werden.

Die redundante Anbindung von Verbrauchern an das Energienetz ist die Aufgabe der Primitive EnRedNode. Sie baut ein paralleles Netz auf, welches beim Versagen des ersten den redundant angebundenen Verbrauchern weiterhin den Bezug von Energie ermöglicht. Die Primitive EnRedNode besitzt keine Parameter, aber die Eingänge *In_global* (multi), *In_local* (multi), *Inf_get* und *If_broken* sowie die Ausgänge *Inf_put_global*, *Inf_put_local* und *Out*. Die Unterscheidung zwischen global und local bezieht sich darauf, daß an globale In- bzw. Ausgänge nur das redundante Netz angeschlossen werden darf, während die lokalen In- und Ausgänge für die Verbraucher vorgesehen sind. Das Verhalten der Primitive wird über den Eingang *If_broken* gesteuert. Dieser nimmt den Status des zugehörigen normalen Nodes auf. Lautet er ENERGY_CONNECTION_BROKEN, wird über *Inf_put_local* die Nachricht ENERGY_REDUNDANT_MODE versendet. Diese signalisiert den Verbrauchern, daß sie weiterhin Energie anfordern können. In diesem Fall werden eintreffende Anforderungen von *In_local* nach *Out* und von *Inf_get* nach *Inf_put_global* und *Inf_put_local* weitergeleitet. Für das Funktionieren des redundanten Netzes ist es wichtig, daß unabhängig vom lokalen Status Anfragen über *In_global* immer über *Out* weitergegeben werden.

Brennstoffzelle

Die Brennstoffzelle liefert die Energie für das AUV. Sie ist ein elektrochemischer Energiewandler und besteht aus zwei Elektroden (Anode und Kathode), die durch einen Elektrolyten räumlich voneinander getrennt sind ([IB]). Bei dem verwendeten PEM-Zellentyp ist dies eine Polymerfolie (undurchlässig für Gase, durchlässig für Ionen), die beidseitig mit einer Katalysatorschicht überzogen ist. Der Anode wird Wasserstoff, der Kathode Sauerstoff zugeführt. Beide reagieren in einer kontrollierten Reaktion zu Wasser, wobei die chemische Energie in Wärme und elektrische Energie umgewandelt wird

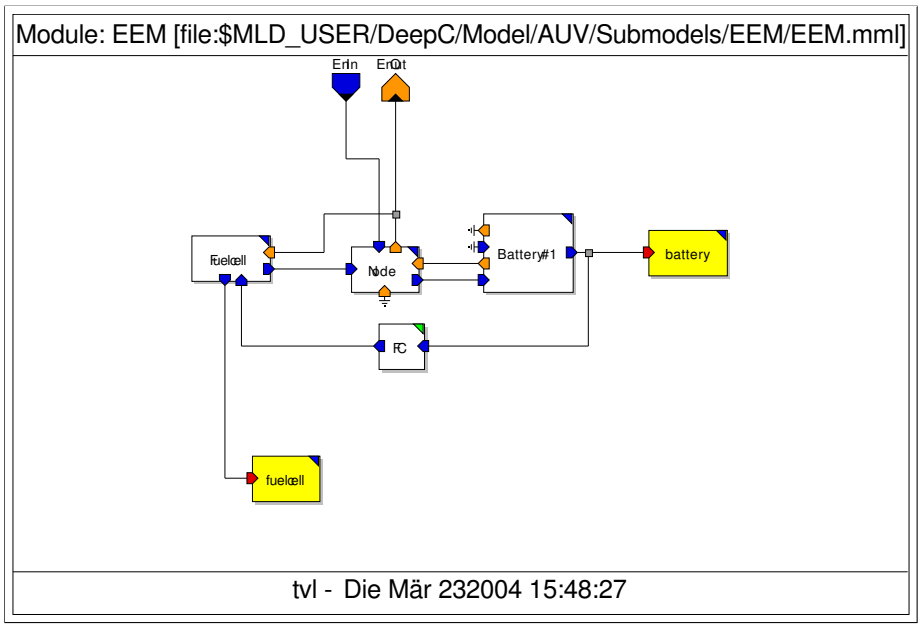

Abbildung 5.12: Energieerzeugungsmodul

([Ker]). Um technisch nutzbare Spannungen zu erhalten, werden je nach erforderlicher Leistung mehrere Zellen zu einem sogenannten Stack verbunden. Die anfallende Wärme wird über ein Kühlsystem abgeleitet.

Die Primitive fuelCell modelliert dieses Verhalten abstrakt. fuelCell ist von EnBaseDeviceRS abgeleitet, erzeugt aber kontinuierlich Energie (*SampleTime*), anstatt sie zu verbrauchen. Die Brennstoffzelle besitzt eine bestimmte Kapazität (Parameter *Capacity*), die der chemischen Energie des mitgeführten Sauerstoffs und Wasserstoffs entspricht. Der vorhandene Vorrat (Startfüllstand: *Level*) wird gesteuert in Energie umgesetzt. Über den Eingang *ctrl* wird die benötigte Energiemenge pro Zeit festgelegt. Änderungen erfolgen dabei mit einer gewissen Dynamik, die durch ein T_1-Verhalten nachgebildet wird (*Factor*)[61]. Die verfügbare Energiemenge pro Zeit ist jedoch begrenzt (*Power*). Über den Ausgang *Load* wird der noch vorhandene Prozentsatz von *Capacity* ausgegeben.

Energieerzeugungsmodul

Das Energieerzeugungsmodul stellt die vom Fahrzeug benötigte Energie zur Verfügung. Die Energie wird dabei von den Brennstoffzellen erzeugt. Zwischen den Verbrauchern und der Brennstoffzelle sind Pufferbatterien geschaltet, die die Brennstoffzellen vor Lastspitzen schützen.

Das Modul EEM bildet das Energieerzeugungsmodul nach. Es besteht aus einer Brennstoffzelle (fuelCell) und einer Pufferbatterie (Battery). Die Steuerung der Brenn-

[61] $Faktor = \exp(-T/T_1)$, nach [Gün97, S. 250]

stoffzelle geschieht in Abhänigkeit vom Füllstand der Pufferbatterie (FC). Daneben beinhaltet das Modul Bestandteile des Energienetzes (EnNode, *EnIn* und *EnOut*) und zwei Ausgabeprimitiven. Abbildung 5.12 zeigt das modellierte Energieerzeugungsmodul, in dem die oben vorgestellten Primitiven deutlich erkennbar sind.

Lampe

Die am AUV angebrachten Scheinwerfer werden durch die Primitive Lamp modelliert. Sie ist als Energieverbraucher von EnBaseDeviceRS abgeleitet. Zusätzlich zu den ererbten Eingängen, Ausgängen und Parametern besitzt sie den Eingang *on_off* und den Parameter *ID*. Letzterer weist der Primitive eine Nummer zu, über die sie ab- bzw. angeschaltet werden kann. Die Schaltvorgänge selbst erfolgen über den Eingang *on_off*. Nur im angeschalteten Zustand verbraucht Lamp Energie, wobei die Menge durch den ererbten Parameter *Wattage* festgelegt wird.

5.3.3.2 Aktorik

ObjMatrix

Das Softwareobjekt[62] ObjMatrix vereinfacht den Umgang mit FloatMatrix-Matrizen, in denen in jeder Zeile eine Position codiert wird. Solche Matrizen werden innerhalb des Modells genutzt, um Informationen zwischen Primitiven auszutauschen.

ObjMatrix bildet eine interne Repräsentanz dieser Positionen in Form einer Liste. Über die Methode *add()* können neue Positionen hinzugefügt werden. Per *size()* läßt sich die aktuelle Anzahl an Positionen ermitteln und über *empty()* testen, ob ein Objekt aktuell keine Positionen enthält. Die Methode *clear()* löscht alle Positionen aus dem Objekt. Durch *isInMatrix()* kann ermittelt werden, ob eine bestimmte Position bereits vorhanden ist. Über die Methoden *getValue()* und *setValue()* lassen sich die einzelne Positionen auslesen beziehungsweise setzen. Die Methode *asFloatMatrix()* schließlich gibt den Inhalt von ObjMatrix wieder codiert in einer FloatMatrix aus.

ObjFrameTrans

Das Objekt ObjFrameTrans ermöglicht den Übergang zwischen verschiedenen Koordinatensystemen, indem es die dafür notwendige Transformation kapselt.

Das Softwareobjekt stellt die Methoden *updateTrans()* und *doTrans()* bereit. Durch *updateTrans()* wird die aktuelle Transformation in das Objekt übertragen, mit *doTrans()* kann im Anschluß eine beliebige Anzahl von Punkten transformiert werden.

[62] Mit Softwareobjekten oder kurz Objekten werden in diesem Kapitel C++-Objekte bezeichnet, die im Rahmen der Modellierung außerhalb von MLDesigner erstellt wurden, um die Primitivenerstellung zu vereinfachen, indem sie spezielle Funktionalitäten kapseln. Sie tragen jeweils die Bezeichnung obj<Name>. Ihre Definition erfolgt grundsätzlich in den zugehörigen Dateien im Verzeichnis `$MLD_USER/DeepC/include/`.

Als Koordinatensysteme werden nach Fossen die Koordinaten $[x,y,z,\phi,\theta,\psi]$ für die Positionen und $[u,v,w,p,q,r]$ für die Geschwindigkeiten verwendet ([Fos94, Fos02]). Die Transformation eines Punktes $A=[x,y,z]^T$ zwischen verschiedenen Koordinatensystemen ist dabei wie folgt definiert:

$$A_T = R \cdot A + T$$

T ist die Translation zwischen beiden Systemen, R ist die Rotationsmatrix.

$$R = \begin{bmatrix} \cos\psi\cos\theta & -\sin\psi\cos\phi+\cos\psi\sin\theta\sin\phi & \sin\psi\sin\phi+\cos\psi\cos\phi\sin\theta \\ \sin\psi\cos\theta & \cos\psi\cos\phi+\sin\psi\sin\theta\sin\phi & -\cos\psi\sin\phi+\sin\psi\sin\phi\cos\theta \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}$$

Die Rücktransformation wird berechnet, wenn die Methode *doTrans()* mit dem Parameter *inv=true* aufgerufen wird.

JTrans

Die Primitive JTrans rechnet die lokalen Geschwindigkeiten (Eingänge: u,v,w) mittels der Rotationen (Eingänge: *phi*,*theta*,*psi*) in globale Geschwindigkeiten (Ausgänge: x,y,z) um. Die Transformation folgt dabei der obigen Formel von ObjFrameTrans.

Engine

Zur Modellierung des Energieverbrauchs des Antriebs wurde die Primitive engine erstellt. Sie ist von EnBaseDevice abgeleitet und bildet den Verbrauch geschwindigkeitsabhängig mittels der Formel

$$E = \begin{cases} (u/3)\cdot W\cdot dt|_{u>0} \\ |u|\cdot W\cdot dt \end{cases}$$

nach. Dabei ist E die benötigte Energie und u die Geschwindigkeit (Eingang *Input1*). Dem Parameter *Wattage* entspricht W und dt der Schrittweite (Parameter *SampleTime*).

Pitch

Die Primitive pitch bildet dynamische Prozesse innerhalb des Antriebs nach. Dabei handelt es sich um einen nichtlinearen, dynamischen Zusammenhang zwischen dem Abtauchwinkel und der Geschwindigkeit.

Als Eingänge verlangt die Primitive den Sollnickwinkel (Eingang *theta_des*) sowie die Sollgeschwindigkeit in Längsrichtung (Eingang *u_des*). Aus beiden Größen werden der erreichbare Nickwinkel und die erreichbare Geschwindigkeit u berechnet und über die Ausgänge *theta* und *u* ausgegeben. Zusätzlich werden die internen Größen Drehzahl n

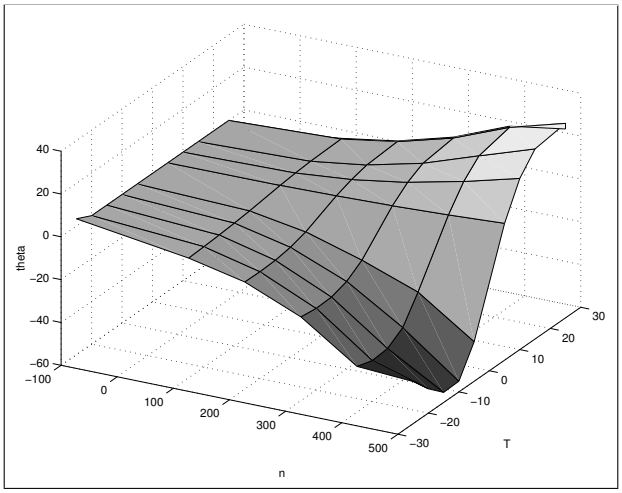

Abbildung 5.13: Funktion $\theta = f(n,T)$

und Ruderwinkel T über *n_intern* und *T_intern* nach außen kommuniziert. Über die Parameter *funktion*, *b00*, *b20*, *b21* und *a* wird das charakteristische Verhalten eingestellt. Hierbei handelt es sich um einen funktionalen Zusammenhang der Form $\theta = f(n,T,u)$, der durch entsprechende interne Untersuchungen der Firma ATLAS ELEKTRONIK GmbH ermittelt wurde. Abbildung 5.13 zeigt eine graphische Darstellung des Zusammenhangs für $\theta = f(n,T)$. An den unteren Achsen sind die Drehzahl n und der Ruderwinkel T abgetragen, die dritte Achse wird vom Nickwinkel θ gebildet. Der Parameter *funktion* der Primitive PITCH nimmt diese Funktion ($\theta = f(n,T)$) in Form einer Wertetabelle gemäß Tabelle 5.2 auf. *size* nimmt die Größe der Tabelle auf (Realteil: Höhe, Imaginärteil: Breite). Der Zusammenhang zwischen n und u läßt sich mittels $u = f(n,T) = (b_{21} \cdot n + b_{20}) \cdot T^2 + b_{00} \cdot n$ approximieren. Die Parameter *b00*, *b20* und *b21* können aus den Daten durch die Methode der kleinsten Quadrate bestimmt werden. Der Parameter *a* stellt einen Filterkoeffizienten dar, mit dem die zeitliche Änderung des Ruderwinkels geglättet wird.

0	n_1	n_2	n_3
T_1	θ_{11}	θ_{12}	θ_{13}
T_2	θ_{21}	θ_{22}	θ_{23}

Tabelle 5.2: Funktion $\theta = f(n,T)$

Der folgende Abschnitt beschreibt die interne Vorgehensweise der Primitive Schritt für Schritt.

1. Mit Hilfe des aktuellen u und des T_{k-1} aus dem letzten Zeitpunkt wird über $n = (u - (b_{20} \cdot T_{k-1}^2))/((b_{21} \cdot T_{k-1}^2) + b_{00})$ das Soll-n bestimmt. Die Formel ergibt sich dabei aus der Umstellung des obigen Zusammenhangs $u = f(n,T)$ nach n.

2. Danach erfolgt die Berechnung des nötigen Ruderwinkels T. Dazu wird eine bilineare Interpolation über die nächstliegenden Werte von n und T aus der Wertetabelle durchgeführt.

   a) Zunächst erfolgt eine Suche nach n im Tabellenkopf von $\theta = f(n,T)$. In deren Ergebnis wird ein Vektor für $n_a < n < n_b$ erzeugt. Dieser ergibt sich aus der linearen Interpolation der Spalten mit n_a und n_b. Sollte n nicht innerhalb der Tabellenwerte liegen, wird die entsprechende Randspalte n_{min} beziehungsweise n_{max} benutzt.

   b) In dem erzeugten Vektor wird nach einem Vorkommen der Art $\theta_c < \theta < \theta_d$ gesucht. Ein solches kann mehrmals oder auch gar nicht auftreten.

   c) Für diese Punkte findet eine lineare Interpolation der zugehörigen T-Werte statt. Wenn mehrere Punkte zur Auswahl stehen, wird die kleinste Lösung durch ein quadratisches Kriterium ausgewählt. Wird kein Punkt gefunden, fällt die Wahl entsprechend auf das größte beziehungsweise kleinste θ. In diesem Fall gilt $\theta_{out} \neq \theta$. Für T wird der Wert verwendet, von welchem an das größte beziehungsweise kleinste θ auftritt.

3. Um Sprünge des Ruderwinkels zu vermeiden, wird das berechnete T gefiltert. Die diskrete Filtergleichung lautet $T_k = a \cdot T_{k-1} + (1-a) \cdot T$. Es läßt sich zeigen, daß dies einem per Sprunginvarianzmethode approximierten, kontinuierlichen PT_1-Glied entspricht ([Gün97, S. 250]). Demnach gilt für $a = \exp(-dt/t_1)$, mit dt als Schrittweite und t_1 als Zeitkonstante.

4. Mit Hilfe des soeben berechneten Ruderwinkels T_k kann $u_{out} = (b_{21} \cdot n + b_{20}) \cdot T_k^2 + b_{00} \cdot n$ bestimmt werden.

5. Im letzten Schritt erfolgt die Ausgabe von u_{out} und θ_{out} sowie der internen Größen T_k und n zum Zwecke der Überprüfbarkeit.

Antrieb

Das Antriebsmodell bildet das Fahrverhalten des geregelten AUV nach. Die in der Literatur verfügbaren Fahrzeugmodelle setzen hier meist bei den Kräften und Momenten an[63]. Aus diesem Vorgehen resultiert jedoch ein Modell, das zahlreiche Parameter besitzt, die zum Zeitpunkt der Modellierung im missionsbezogenen modellgestützten Entwurf noch nicht bekannt sind. Diese lassen sich auch nicht von anderen Fahrzeugen übernehmen, da sie von der genauen Fahrzeugform und -größe abhängen.

Deshalb modelliert das Antriebsmodell das dynamische Verhalten mittels eines Ansatzes, der die gegebenen Anforderungen an das Fahrverhalten in ein mathematisches Modell umsetzt. Es basiert somit auf den Vorgaben für die Fahrzeugkenngrößen, welche beispielsweise die Strecke bis zum Erreichen der Höchstgeschwindigkeit vorschreiben. Die

[63] Vgl. die Ausführungen zu den Dynamikmodellen im Abschnitt 2.5.2 auf Seite 35.

Qualität dieser Vorgaben läßt nur eine derartig abstrakte Modellierung auf Basis eines regelungstechnisch orientierten Modells zu.

Abbildung 5.14 zeigt das Modul DCM, das das Fahrverhalten nachbildet. Es läßt sich vereinfachend wie folgt beschreiben: Auf der linken Seite befindet sich ein Block, der den Vektor mit den Führungsgrößen Kurs, Tiefe und Geschwindigkeit in seine Bestandteile aufspaltet (FM2F). Die Geschwindigkeitsvorgabe wird über ein PIT_1-Verhalten umgesetzt. Tiefenänderungen realisiert z_controller entweder - bei geringen Geschwindigkeiten - über die Thruster (*w_des*) oder über ein Ab- bzw. Auftauchen (*theta_des*). Zwischen dem realisierbaren Abtauchwinkel und der Geschwindigkeit besteht ein Zusammenhang, der von der bereits vorgestellten Primitive pitch (pitch_dyn) modelliert wird. Die Geschwindigkeitsvorgabe für die Thruster (*w_des*) wird über ein T_1-Glied realisiert (*w*).

Die lokalen Geschwindigkeiten (*u*, *w*) werden danach in globale Geschwindigkeiten überführt (JTrans), die Strömungsgeschwindigkeiten hinzugefügt und dann zur Position aufintegriert (Integratoren für x, y und z). Die Winkel und ihre Änderungen werden als T_1-Glieder modelliert. Dabei ist phi begrenzt (Limiter) und psi wählt den kleineren Winkel (0 bis 180 Grad). Für die Ausgabe wird die aktuelle Position (x, y, z, phi, theta, psi) in einem Vektor zusammengefaßt (Mux).

Der Energieverbrauch wird in Abhängigkeit von der Geschwindigkeit mittels engine nachgebildet. Das gesamte Modell ist diskret realisiert, wobei die Schrittweite durch das Memory dt festgelegt wird.

Dieses Modell wurde parallel auch in leicht reduzierter Form in MATLAB umgesetzt und für den Einsatz in der Testumgebung für das maschinelle Lernen genutzt ([Eic02]).

5.3.3.3 Sensorik

Sonar

Die Bezeichnung Sonar stellt ein Akronym für "sound navigation and ranging" dar. Ein Sonar wird primär für die Erkennung und Lokalisierung von Unterwasserobjekten genutzt. Hierzu werden reflektierte akustische Wellen verwendet, die entweder vom Sonar selbst ausgesendet werden (aktives Sonar) oder von externen Quellen stammen (passives Sonar)[64]. Eine entwurfsorientierte Einführung in diese Thematik bietet beispielsweise Waite ([Wai96]).

Für die Modellierung im Rahmen des Mission Level Designs wurde eine abstrakte Nachbildung des Verhaltens eines aktiven Sonars vorgenommen. Dieses zeichnet sich dadurch aus, daß es nacheinander einzelne Pings einer spezifischen Form aussendet. Dabei nimmt es für jeden Ping eine neue Position ein. Auf diese Weise wird ein Sektor Schritt für Schritt vollständig abgesucht.

[64] Vgl. hierzu Weik [Wei96].

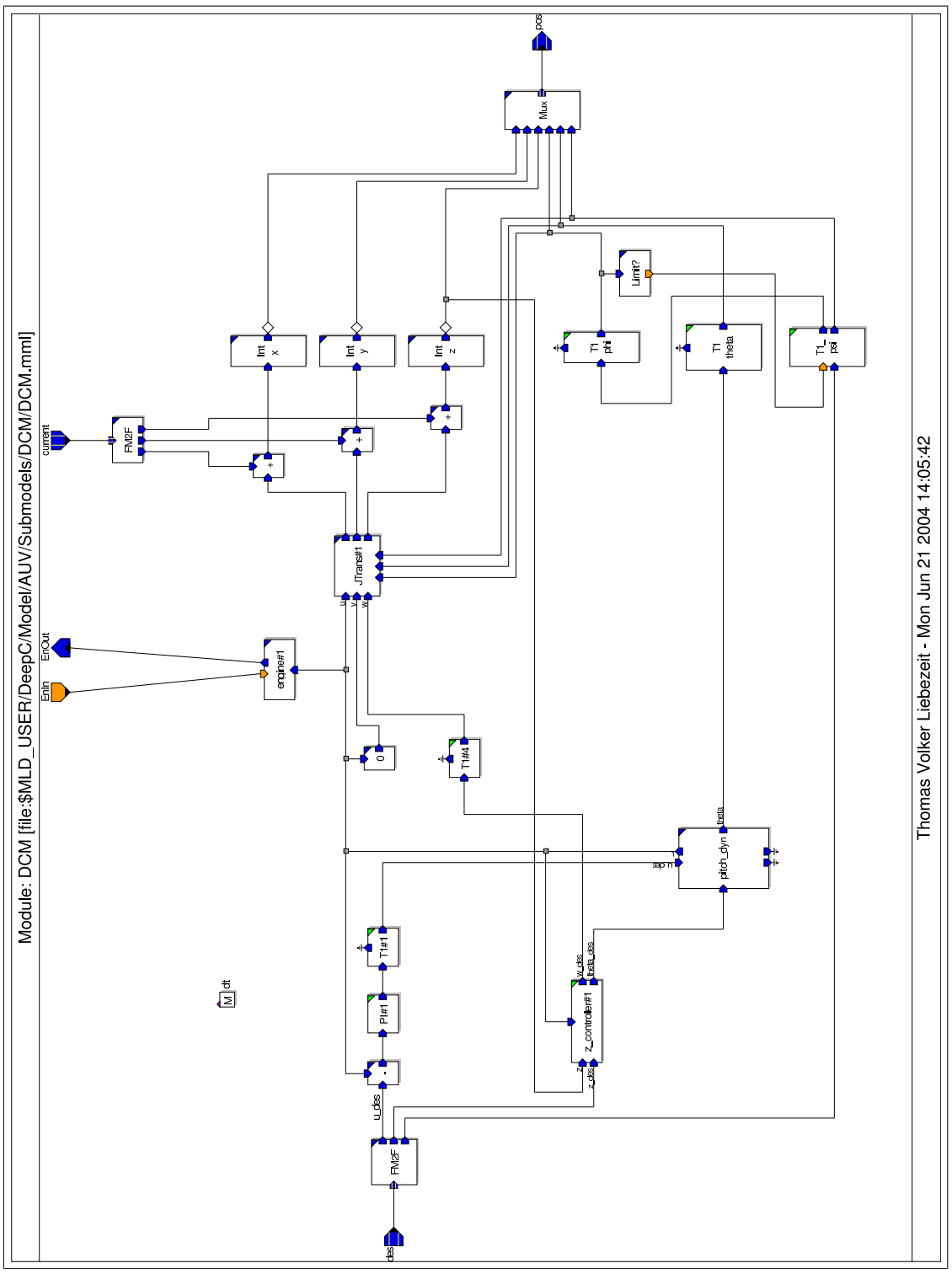

Abbildung 5.14: Struktur des Antriebsmodells

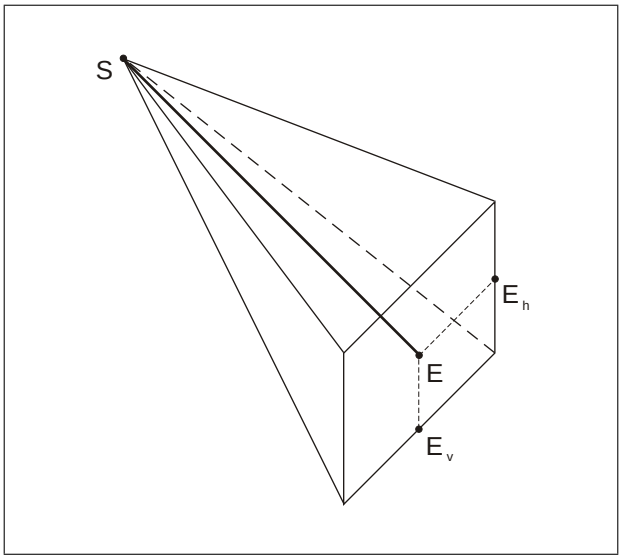

Abbildung 5.15: Suchraum der Pings

Um dieses Verhalten nachzuahmen, existiert die Primitive SONAR. Da es sich bei einem Sonar um einen Energieverbraucher handelt, ist die Primitive von EnBaseDeviceRS abgeleitet. Sie besitzt den Parameter *dAlpha*, welcher den Winkel angibt, um den das Sonar in jedem Schritt um die z-Achse gedreht wird. *StartPos* legt die Lage des Sonars bezüglich des AUV-Mittelpunkts fest. Die Ausprägung des Pings wird durch die komplexwertigen Parameter *PingDist* und *PingAngle* festgelegt. Ersterer enthält im Realteil die minimale und im Imaginärteil die maximale Reichweite des Pings. *PingAngle* beschreibt im Realteil den vertikalen und im Imaginärteil den horizontalen Öffnungswinkel. Die zeitliche Distanz zwischen zwei Pings wird durch die *PingRate* festgelegt. *PingArea* gibt den zu untersuchenden Sektor vor. Dieser ist während der Simulation, genauer jeweils am Ende des Abscannens eines Sektors, durch den Eingang *set_view* manipulierbar. Der Eingang *posAUV* nimmt die aktuelle Position des Fahrzeugs im Weltkoordinatensystem[65] auf. Über den Ausgang *ask_for_objects* werden die Pings versendet. Ihre Reflexion wird durch den Eingang *input_obj* empfangen und nach Rücktransformation in lokale AUV-Koordinaten über den Ausgang *output* weitergereicht. Das Ergebnis eines Pings wird in Anlehnung an die realen Verhältnisse scanline genannt, da ein reales Sonar an dieser Stelle einen Vektor mit Bytewerten liefern würde. Der Ausgang *sweep* zeigt an, wann ein Sektor vollständig abgearbeitet ist.

Die Form eines Ping wird durch eine Pyramide mit rechteckiger Basis nachgebildet (siehe Abbildung 5.15). Hierzu wird das in Tabelle 5.3 aufgelistete Protokoll genutzt. Es beschreibt die Pyramide mittels vier charakteristischer Punkte (Spitze S, Mittelpunkt der Basis E und die Punkte E_v und E_h) in lokalen Sonarkoordinaten.

[65] Das Gesamtsystemmodell unterscheidet drei Koordinatensysteme. Die Weltkoordinaten beziehen sich auf einen globalen Nullpunkt. Daneben existieren die AUV-Koordinaten, die sich auf den Schwerpunkt des Fahrzeugs beziehen und die Sonarkoordinaten, die die Welt aus Sicht des Sonars beschreiben. Bei Bedarf erfolgt eine Umrechnung zwischen den einzelnen Koordinatensystemen.

0	1	2	3	4	5
minDist	0	0	0	0	0
maxDist	0	0	0	0	0
maxDist	maxV	0	0	0	0
maxDist	0	maxH	0	0	0

Tabelle 5.3: Ping (in lokalen Sonarkoordinaten) (Protokoll)

Die interne Verarbeitung eines Pings nach diesem Protokoll erlaubt die leichte Transformation aus den lokalen Sonarkoordinaten in globale Weltkoordinaten. Da die Pings in globalen Weltkoordinaten ausgesendet werden, kommt für die interne Transformation ObjFrameTrans zum Einsatz. Um diese Transformation durchführen zu können, benötigt das Sonar als einen seiner Eingänge die exakte Position des AUV.

Die Antwort auf einen Ping besteht in einer $(n \times 6)$-Matrix, die in jeder Zeile die Position eines gefundenen Objektes enthält. Dabei kann es sich sowohl um ein Hindernis, als auch um den Seeboden handeln. Für die Primitive ist dies nicht unterscheidbar.

Wenngleich das verwendete Sonarmodell durch ein abstrahiertes Vorgehen entstanden ist, bildet es die funktionalen Eigenschaften eines realen Sonars ab. Wichtig ist in diesem Zusammenhang, daß die Pings abstrahiert sind, daß also keine Wellen ausgesendet und deren Reflexionen an Objekten untersucht werden, sondern vielmehr ein dreidimensionaler Suchstrahl verwendet wird. Gleiches trifft auch auf die scanlines zu: Für diese werden Matrizen mit Objektpositionen eingesetzt.

Dolog

Ein Dolog ist ein Dopplersonar, das zum einen die Fahrt durch das Wasser, zum anderen den Abstand zum Seeboden mißt. Dabei ist die meßbare Entfernung jedoch beschränkt.

Die Modellierung erfolgt durch die Primitive dolog, die ähnlich wie das Sonar arbeitet. Sie ist von EnBaseDeviceRS abgeleitet und charakterisiert damit einen Energieverbraucher. Der Parameter *Position* beschreibt die Position relativ zum AUV-Mittelpunkt und damit den Einbauort des Dologs. dolog sendet einen Ping gemäß dem Protokoll aus Tabelle 5.3 aus. Zur Parametrisierung des Pings besitzt die Primitive die Parameter *PingRate* und *PingDist*. Ersterer legt die Anzahl der Pings pro Sekunde fest, während der Zweite die minimale und die maximale Entfernung für Objekte vorgibt, die erkannt werden sollen. Über den Eingang *posAUV* erhält die Primitive die aktuelle globale Position des AUV. Diese wird benötigt, um den Ping intern in globale Weltkoordinaten umzurechnen. Dazu wird auf die Funktionalität des ObjFrameTrans-Objekts zurückgegriffen. Die Pings werden über den Ausgang *ask_for_objects* versendet. Aus der Antwort (Eingang *input_obj*) wird dann die Entfernung zum Boden ermittelt und über *depth* ausgegeben. Ein Wert kleiner null beschreibt dabei den Abstand zum

Grund, wohingegen der Wert eins für die Tatsache steht, daß der Seeboden außerhalb der Reichwerte des Dologs liegt. Für die interne Arbeit mit den Pings werden ObjMatrix-Objekte verwendet.

Parallel dazu erhält die Primitive die aktuellen Strömungswerte über den Eingang *current* und speichert diese intern. Nach jedem Ping wird der letzte eingegangene Strömungswert als "Meßergebnis" über den Ausgang *current_m* weitergereicht.

Inertiales Navigationssystem

Das inertiale Navigationssystem mißt die aktuelle Position des Fahrzeugs und liefert sie an das Navigationmodul. Der Standardaufbau dieses Trägheitsnavigationssystems sieht dabei drei Gyroskope und drei Beschleunigungsmesser vor ([Hin03]). Eren und Fung geben in [JGW99, S. 10-34] eine detaillierte Einführung und nennen dabei die Hauptanforderungen für den Einsatz in autonomen Unterwasserfahrzeugen:

- niedriger Energieverbrauch
- hohe Genauigkeit[66]
- geringes Volumen
- niedriges Gewicht
- geringe Kosten.

In [Dor99] stellt Dorobantu ein Simulinkmodell für eine IMU (Inertial Measurement Unit), den Hauptbestandteil eines INS, vor.

Für die Simulation wurde die Primitive INS geschaffen. Sie ist von EnBaseDeviceRS abgeleitet und besitzt den Eingang *in_pos*, an dem sie die exakte Position empfängt. Über den Ausgang *out_pos* wird die gemessene Position weitergereicht. INS setzt ein deterministisches, ideales Verhalten um. Das bedeutet, daß kein Wegdriften der Werte über die Zeit stattfindet (CEP[67]=0), sondern das INS immer die korrekte Position liefert. Folgerichtig wird der CEP nicht ausgegeben. Daraus folgt desweiteren, daß auf dem CEP aufbauende Manöver (z.B. das GPS-Update) nicht unterstützt werden.

5.3.3.4 Nutzlast

Das Nutzlastmodul nimmt zusätzliches, anwenderspezifisches Equipment auf. Für die in dieser Arbeit durchgeführten Untersuchungen ist eine Modellierung von speziellen Nutzlasten nicht notwendig. Aus diesem Grund ist das entsprechende Modul NLM leer.

[66] Eine Analyse zur Genauigkeit gegenüber anderen Methoden zur Positionsbestimmung (sonar velocity log, doppler velocity log, acoustic transponder navigation) liefern Stambaugh und Thibauld ([ST92]).
[67] Circular Error Probable

5.3.3.5 Prozessor

ObjSWKomm

Das Softwareobjekt ObjSWKomm dient zur Erleichterung der Kommunikation zwischen Softwaremodulen. Es kapselt das für die Simulation verwendete Protokoll zum Austausch von Nachrichten zwischen den Softwaremodulen.

Die Hauptmethoden des Softwareobjekts ObjSWKomm sind *writeSWKommMsg()* und *readSWKommMsg()*. Während erstere das Versenden von Nachrichten ermöglicht, regelt die Zweite deren Empfang. Auch bei diesen Nachrichten handelt es sich um in einer FloatMatrix codierte Informationen. Die Methode *setID()* dient zum Setzen einer eindeutigen Nummer (sender_ID), die in der Kommunikation verwendet wird.

Tabelle 5.4 stellt das verwendete Protokoll vor. Die erste Zeile wird von einem Header gebildet, der aus der sender_ID des Senders, der Nachrichtennummer sowie der Breite (*cols*) und Höhe (*rows*) der Softwarenachricht besteht. In den weiteren Zeilen schließt sich die Softwarenachrichtenmatrix (*info*) an. Die Größe der Nachricht variiert entsprechend der Größe der Informationsmatrix, beträgt aber mindestens (1×4) für eine leere info-Matrix. Die Nachrichtenbreite von vier bleibt auch dann erhalten, wenn die Informationsmatrix selbst eine geringere Breite aufweist. In diesem Fall werden die von der info-Matrix nicht verwendeten Matrixelemente in den Spalten mit Nullen gefüllt.

0	1	2	3	4	...
sender_ID	code	rows	cols	0	0
info					

Tabelle 5.4: Kommunikation von Software zu Software (Protokoll)

Prozessor und Softwaremodule

Für die Modellierung im Sinne des Mission Level Designs besitzen Ressourcen eine zentrale Bedeutung. Eine der wichtigsten stellt hierbei die Rechenzeit der Software dar. Aus diesem Grund ist es notwendig, zusätzlich zur Umsetzung der einzelnen Softwarefunktionalitäten eine abschätzende Simulation des Zeitverhaltens der Software in das Modell zu integrieren.

Takala listet verschiedene Modellierungsebenen für Prozessoren auf ([Tak01]):

- **Design-Based Model:**
  Beim Design-Based Model wird der Prozessor auf Gatterebene, das heißt in einer Hardwarebeschreibungssprache (z.B. VHDL), beschrieben. Diese Modellierung bildet das Timing akkurat nach, erlaubt aber nur eine extrem langsame Ausführung von Programmcode.

- **Cycle Accurate Model:**
  Hierbei handelt es sich um eine Modellierung auf Instruktionsebene (ISS[68]), bei der die Software über die Instruktion, aus denen sie besteht, beschrieben wird. Die Simulation der Ausführung geschieht dabei zyklenakkurat. Der ISS ist in ein Businterface Modell (BIM) eingebettet, das das Timing exakt nachbildet.

- **Function Model:**
  Auch beim funktionalen Modell wird ein BIM genutzt. Die Software wird mittels einer Hochsprache (z.B. C) beschrieben. Es findet kein Timing für ihre Ausführung statt.

- **Physical Model:**
  Ein physikalisches Modell, das heißt ein wirklicher Prozessor, wird verwendet und liefert schnelle, exakte Ergebnisse.

Keines der aufgeführten Modelle ist für eine Modellierung auf Missionsebene geeignet. Denn für diese Simulation wird zwar keine zyklenkorrekte, aber dennoch eine zeitbehaftete Ausführung benötigt, um einerseits die Auslastung des Prozessors (Architektur) und andererseits das Zeitverhalten der Software (Funktion) nachzubilden.

Aus diesem Grund wurde für das *DeepC*-Gesamtsystemmodell ein eigenes Rechnermodell entwickelt. Es setzt auf eine abstrahierte Beschreibung des Prozessors als Cycle-Verteiler (PROCESSOR) und auf Softwaremodule (PROCESSORSWMODUL), die Funktion beschreiben, sich jedoch mit der Hardware abstimmen. Die realisierte Lösung wird im Folgenden ausführlich vorgestellt.

Das Rechnermodul in Form der Primitive PROCESSOR verteilt Rechenarbeit (*cycles*) an die abzuarbeitenden Softwaremodule. Hierzu besitzt es die Parameter *frequency* und *cycles_per_sec*. Bei *frequency* handelt es sich nicht um die wirkliche Frequenz der CPU. Vielmehr gibt der Parameter diejenige Frequenz wieder, mit der die Abarbeitung in der Simulation modelliert wird. Sie legt das betrachtete Zeitintervall fest, in welchem die Softwaremodule parallel abgearbeitet werden. Hierbei genügt in den meisten Fällen eine vergleichsweise große Zeitspanne von einigen Zehntelsekunden, was zu Frequenzen im Bereich um 10 Hz führt. Durch die Auswahl des Parameters *cycles_per_sec* wird die Leistung des Rechners beschrieben, die in Kombination mit der Frequenz die mögliche Arbeit pro Zeitintervall festlegt. Weitere Parameter sind *number*, der die Nummer des Rechners festlegt, und *master*. Nur wenn ein Rechner mittels dieses Parameters als master deklariert ist, ist er in der Lage, Softwaremodule auf anderen Rechnern zu starten. Die Parameter *distribution_error* und *max_depth* beeinflussen die Verteilung der Rechenarbeit. *report_max_depth* gibt an, ob im Fall des Erreichens der Rekursionstiefe *max_depth* eine Warnmeldung ausgegeben werden soll. Um Verbindungen zu anderen Rechnern herzustellen, existieren der Eingang *sendPC* sowie der Ausgang *recvPC*, während Anfragen nach Rechenarbeit über die multiplen Eingänge *get* und Ausgänge *run* abgewickelt werden. Der Ausgang *load* gibt für jeden der Zeitschritte die Prozessorauslastung aus.

[68] Instruction-Set Simulator

Die Softwaremodule benötigen für ihre Abarbeitung ein gewisses Maß an Arbeit (*cycles*). Da ihre Abarbeitung prioritätsgesteuert erfolgt, besitzen sie neben dem Parameter *cycles* den Parameter *priority*. Die höchste Priorität ist durch den Wert 10, die niedrigste durch den Wert 1 gekennzeichnet.

Der Prozessor entscheidet in jedem seiner Zeitschritte, welche der anstehenden Softwaremodule abgearbeitet werden. Das hierbei genutzte Verteilverfahren vollzieht sich nach dem folgenden Algorithmus:

1. Ein Softwaremodul, das abgearbeitet werden muß, übermittelt an den Prozessor die Informationen cycles c und Priorität p.
2. Für jedes der zur Abarbeitung bereitstehenden Module wird das Produkt aus cycles c und Priorität p gebildet ($sm = c \cdot p$). Dieses wird mit einer konstanten Zahl (Z) multipliziert, wodurch sich das interne Maß der zu verrichtenden Arbeit für dieses Softwaremodul $w = sm \cdot Z$ ergibt. Die Multiplikation mit Z sichert die gerechte Verteilung, weil so auch bei kleinen Mengen vorhandener Prozessorarbeit pro Zeitschritt (W) eine korrekte Teilung erfolgt. Der Faktor Z ist so zu wählen, daß das Produkt ($W \cdot Z$) mindestens einen Wert von 1000 ergibt.
3. Die neue Anfrage wird in die interne Liste der abzuarbeitenden Softwaremodule aufgenommen. Dazu werden die Werte w^i und p^i gespeichert.
4. Während jedes Prozessortakts wird die Summe aller Prioritäten p^i gebildet ($P_t = \sum_i p^i|_t$).
5. Die Arbeit W wird proportional zu den Prioritäten verteilt, das heißt, jedes Modul wird um Arbeit der Größe $v_t^i = (W \cdot Z) \cdot (p^i/P_t)$ reduziert ($w_{t+1}^i = w_t^i - v_t^i$). Übersteigt der Wert von v_t^i den von w_t^i ($v_t^i > w_t^i$), so wird nur w_t^i subtrahiert. In diesem Fall bleibt ein Rest an nicht verteilter Arbeit (v^r). Dieser wird nach den obigen Formeln unter den verbleibenden Modulen rekursiv verteilt. Die Rekursion wird durch zwei Kriterien begrenzt: Erstens durch den Parameter *distribution_error*, welcher dasjenige Verhältnis zwischen nicht verteilter und zu verteilender Arbeit festlegt ($v^r/W \cdot Z$), ab dem die Rekursion gestoppt wird. Zweitens kann durch den Parameter *max_depth* die maximale Rekursionstiefe vorgegeben werden. Ist hierbei der Parameter *report_max_depth* gesetzt, wird zusätzlich eine entsprechende Warnmeldung ausgegeben.
6. Ein Softwaremodul gilt dann als abgearbeitet, wenn es keine Arbeit mehr benötigt ($w_{t+1}^i = 0$). In diesem Fall wird am Ende des Zeitintervalls die Beendigung an das Softwaremodul signalisiert.

Durch dieses Verfahren ist es möglich, die zeitliche Abarbeitung von konkurrierenden Prozessen zu simulieren, welche durch eine spezifische benötigte Rechenarbeit und eine Priorität gekennzeichnet sind.

Tabelle 5.5 verdeutlicht die Arbeitsweise des Verteilverfahrens nochmals anhand eines Beispiels.

			t_1		t_1+1			t_1+2
Modul	c	p	w^i	$-v^i\|_1$	w^i	$-v^i\|_1$	$-v^i\|_2$	w^i
S^1	19	1	950	-250	700	-200	-150	350
S^2	7	3	1050	-750	300	-300	-	0
S^3	10	1			500	-200	-150	150
				-1000		-700	-300	

($W = 20$, $Z = 50$)

Tabelle 5.5: Rechenbeispiel zum Prozessorverteilalgorithmus

Der Prozessor weist ein W von 20 auf, Z beträgt deshalb für dieses Beispiel 50 ($W \cdot Z = 1000$). Zum Zeitpunkt t_1 müssen zwei Softwaremodule abgearbeitet werden. S^1 hat eine Priorität von $p^1 = 1$ und ein c von 19. Damit ergibt sich $w^1_{t1} = c \cdot p \cdot Z$ zu 950. Das zweite Softwaremodul S^2 verfügt über eine Priorität von 3 und ein c von 7, was zu einem w^2_{t1} von 1050 führt. Die Summe der Prioritäten $P_{t_1} = 1+3$ ergibt 4, damit entfällt auf S^1 ein $v^1_{t_1}$ von $(20 \cdot 50) \cdot (1/4) = 250$, für S^2 ist $v^2_{t_1} = 750$. In diesem Schritt wird keines der Module vollständig abgearbeitet. Zum nächsten Zeitpunkt $t_1 + 1$ besitzt S^1 ein $w^1_{t_1+1} = 950 - 250$ von 700, für S^2 ist $w^2_{t_1+1} = 300$. Nun kommt ein weiteres, abzuarbeitendes Softwaremodul S^3 hinzu. Es hat eine Priorität von 1 und ein c von 10 ($w^3_{t_1+1} = 500$). Die neue Summe der Prioritäten lautet $P_{t_1+1}|_1 = 1+3+1 = 5$. Daraus ergeben sich $v^1_{t_1+1} = (20 \cdot 50) \cdot (1/5) = 200$ und $v^3_{t_1+1} = 200$. Für $v^2_{t_1+1}$ würde ein v von 600 zur Verfügung stehen, das Softwaremodul

benötigt aber nur 300. Aus diesem Grund bleibt ein Rest von $v^r = 300$, der unter den verbleibenden Modulen in einem weiteren Schritt verteilt wird. Nur die Module S^1 und S^3 benötigen jetzt noch weitere Prozessorarbeit. Damit ist $P_{t_1+1}|_2$ für den zweiten Durchlauf gleich $P_{t_1+1}|_2 = 1+1 = 2$, für $v^1_{t_1+1}|_2$ und $v^3_{t_1+1}|_2$ ergibt sich jeweils $300 \cdot (1/2) = 150$. Nach diesem Durchlauf ist die gesamte Arbeit W verteilt. Am Ende des Zeitschritts $t_1 + 1$ ist S^2 abgearbeitet, die Module S^1 und S^3 haben ein w^1 von 350 bzw. ein w^3 von 150.
Die Ergebnisse des Beispiels lassen sich gut visualisieren. Einerseits erhält man die Auslastung des Prozessors, die in diesem Fall für beide Zeitpunkte bei 100 Prozent bzw. $W = 20$ liegt (siehe Abbildung 5.16). Andererseits läßt sich die Laufzeit der Module (wie in Abbildung 5.17) darstellen. Die Abarbeitung von S^1 und S^2 startet zum Zeitpunkt t_1. Für S^2 endet sie nach dem Schritt t_1+1, während S^1 noch mindestens ein Rechenintervall benötigt. Die Abarbeitung von S^3, erst in t_1+1 gestartet, dauert ebenfalls mindestens bis t_1+2.

Das Prozessormodell bietet also die Möglichkeit, die zeitliche Abarbeitung von Prozessen mit einer bestimmte Menge an Rechenarbeit und mit einstellbarer Priorität zu simulieren. Neben der internen Abarbeitung spielt die Kommunikation zwischen Prozessor und Softwaremodul eine entscheidende Rolle. So muß das Softwaremodul signalisieren, daß es Rechenarbeit benötigt, während im Gegenzug der Rechner die Beendigung der Abarbeitung anzeigen muß. Um diese wechselseitige Kommunikation bei der Programmierung der Softwaremodule einfach nutzbar zu machen, wurde das Standardsoftwaremodul prozessorSWModul realisiert. Diese Primitive besitzt für die Kommunikation mit dem Prozessor den Ausgang *get* und den Eingang *run*. Über *get* werden die Anfragen nach Rechenarbeit abgesetzt, während über *run* die Benachrichtigung über das Ende der Abarbeitung erfolgt. Die für die Formulierung der Anfragen erforderlichen Informationen über die benötigte Rechenarbeit, die Priorität und den zu nutzenden Prozessor werden

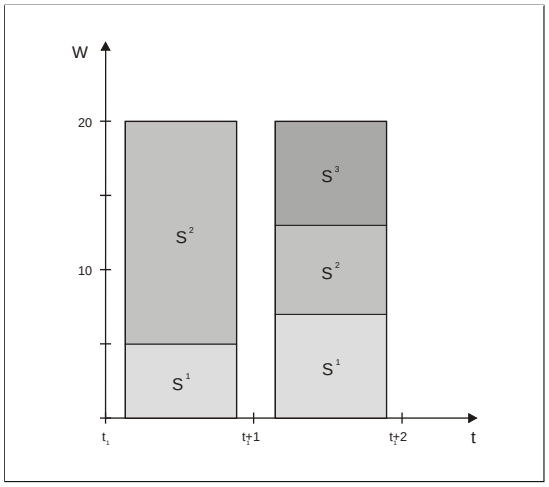

Abbildung 5.16: Auslastung des Prozessors

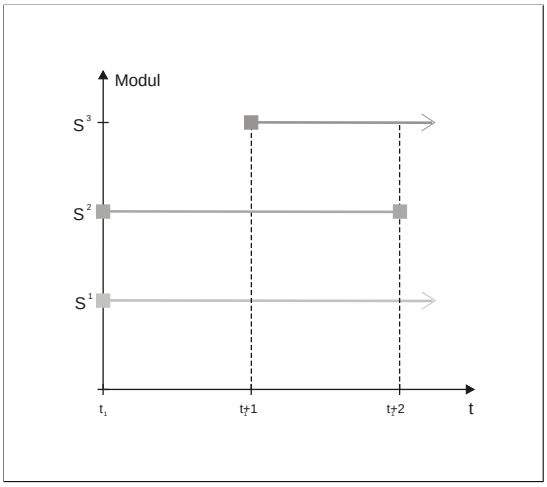

Abbildung 5.17: Laufzeit der Module

über die Parameter *cycles*, *priority* und *CPU* eingestellt. Um Nachrichten mit anderen Softwaremodulen austauschen zu können, existieren der Ausgang *SWout* und der Eingang *SWIn*.

Bei der Primitive processorSWModul handelt es sich um eine Basisprimitive, von der sich funktionale Softwareprimitiven ableiten lassen. Um deren Programmierung zu vereinfachen, stellt processorSWModul die folgenden Methoden bereit: Für die Kommunikation mit dem Prozessor existieren die Methoden *start_SWModul()*, *stop_SWModul()*, *ask_for_processing()* und *computing_finished()*. Eine Vereinfachung der Kommunikation mit den anderen Softwaremodulen wird über die Methoden *readSWKommMsg()* und *writeSWKommMsg()* erreicht, wobei auf die Funktionalität von ObjSWKomm zurückgegriffen wird.

Unter der Voraussetzung, daß sowohl mehrere Rechner, als auch mehrere Softwaremodule existieren, lassen sich drei verschiedene Arten von Kommunikationsprozessen unterscheiden:

1. **Rechner und Software**
   Bei der Kommunikation von Rechner und Software sind verschiedene Richtungen möglich.

   - Software zu Rechner
     Die Softwaremodule schicken Nachrichten an einen *get*-Eingang des Prozessors.

     Das privilegierte Softwaremodul MissionControl kann über Nachrichten andere Softwaremodule starten oder stoppen. Dies ist insofern wichtig, als daß nur gestartete Module Rechenarbeit abrufen können. Gleichzeitig erlaubt es dieser Mechanismus, die Zuordnung von Softwaremodulen zu Rechnern im Laufe

der Simulation zu verändern. Der häufigste Fall sind jedoch die Anfragen aller Softwaremodule nach Rechenarbeit.

0	1	2	3	4	5
Sender	Empfänger	Aktion	Information		
Modul-Nr.	CPU-Nr.	compute (0)	prio	cycl	id
Modul-Nr.	CPU-Nr.	start (1)	0	0	id
Modul-Nr.	CPU-Nr.	stop (2)	0	0	id

Tabelle 5.6: Kommunikation von Software zu Rechner (Protokoll)

Tabelle 5.6 stellt das für die Kommunikation von der Software zum Rechner verwendete Protokoll dar[69]. Zur Kodierung der Informationen verwendet es eine (1×6)-IntMatrix. Das nullte Element wird mit dem Sender (also mit der Modulnummer) belegt. Element eins beinhaltet die Nummer der CPU, auf der das Modul läuft, Element zwei die durchzuführende Aktion. Hierbei sind die Aktionen *compute*, *start* und *stop* definiert. Der Inhalt der drei übrigen Elemente gestaltet sich für die drei Aktionen wie folgt: Für *compute* besteht er aus der Priorität (*prio*), der Rechenarbeit (*cycl*) und der ID desjenigen Sofwaremodules, das Rechenarbeit benötigt. Bei den Aktionen *start* und *stop* wird jeweils nur die ID des betreffenden Softwaremoduls übertragen.

- Rechner zu Software
  Auf Nachrichten von Softwaremodulen antwortet der Prozessor auf dem zum *get*-Eingang korrespondierenden *run*-Ausgang (identische Nummer). Definiert ist hier nur die Nachricht, daß die Abarbeitung der Rechenarbeit abgeschlossen ist. In diesem Fall wird ein definierter Integerwert (2) gesendet.

2. **Rechner und Rechner**
   Ist eine Anfrage nicht für den lokalen Rechner bestimmt, wird sie an die anderen Rechner weitergeleitet. Dies betrifft sowohl das Starten und Stoppen von Modulen, als auch das Ausführen derselben. Sobald die Abarbeitung einer nicht lokalen Anfrage abgeschlossen ist, wird an den Ursprungsrechner eine entsprechende Nachricht versendet.

   Tabelle 5.7 bietet eine kurze Übersicht über das hierbei verwendete Protokoll. Es wird durch die Datei `$MLD_USER/DeepC/include/processor_comm_protocol.h` definiert und besteht aus einem Header mit den Informationen Sender, Empfänger und Aktion sowie aus einem Informationsteil, der vier Elemente breit ist. Das gesamte Protokoll wird in einer (1×7)-IntMatrix kodiert. Die Felder Sender und Empfänger beinhalten die Nummer derjenigen CPU, von welcher die Nachricht

[69] Die Definition für das Protokoll der Kommunikation der Software mit dem Rechner findet in der Datei `$MLD_USER/DeepC/include/processor_comm_protocol.h` statt.

0	1	2	3	4	5	6
Sender	Empfänger	Aktion	Information			
CPU-Nr.	CPU-Nr.	compute (0)	prio	cycl	modul_id	port
CPU-Nr.	CPU-Nr.	finished (1)	0	0	modul_id	port
CPU-Nr.	CPU-Nr.	start (2)	0	0	modul_id	0
CPU-Nr.	CPU-Nr.	stop (3)	0	0	modul_id	0

Tabelle 5.7: Kommunikation von Rechner zu Rechner (Protokoll)

stammt beziehungsweise für welche sie bestimmt ist. Als mögliche Aktionen sind *compute*, *finished*, *start* und *stop* definiert. Für *compute* enthält der Informationsteil die folgenden Daten: Priorität (*prio*), Rechenarbeit (*cycl*), die ID des Rechenarbeit benötigenden Softwaremoduls (*modul_id*) sowie die Nummer desjenigen Ports, an dem das Softwaremodul am sendenden Rechner angeschlossen ist (*port*). Die Antwort (Aktion *finished*) übermittelt nur die ID des Softwaremoduls (*modul_id*) sowie die zuvor empfangene Portnummer (*port*). Die ID des Softwaremoduls (*modul_id*) wird auch bei den Aktionen *start* und *stop* übertragen.

3. **Software und Software**
   Für die Kommunikation von Softwaremodulen untereinander existiert ein eigenes Protokoll, das von ObjSWKomm gekapselt wird. Jedes Softwaremodul sendet seine Nachricht auf einem Bus, wobei alle anderen Module jeweils entsprechend ihrer Funktionalität entscheiden, ob sie diese Nachricht auswerten oder nicht.

   Damit eine eindeutige Zuordnung der Nachrichten möglich ist, erhält jedes Modul eine Nummer zur Absenderidentifizierung (*sender_ID*) und einen codierten Nachrichtentitel. Beide Informationen werden in einer zentralen Datei verwaltet (`$MLD_USER/DeepC/include/SWModule_ID.h`). Dieses Vorgehen ermöglicht eine übersichtliche und an die realen Gegebenheiten angepaßte Modellierung der Kommunikationsprozesse zwischen den Softwaremodulen, da bei der Erstellung der Softwaremodule auf Nachrichtentitel im Klartext zurückgegriffen werden kann. Die Nachrichtentitel werden bei der Beschreibung der realisierten Softwaremodule vorgestellt.

   Das Listing 5.2 gibt beispielhaft einen Auszug aus der Datei `SWModule_ID.h` wieder. ADM_ID ist in diesem Fall eine Absenderidentifizierungsnummer und ADM_-GET_NEXT_MANOUVRE ein Nachrichtentitel.

```
#define SONARCTRL_ID 1
12          #define SONARCTRL_GET_SONARHEADING 1
            #define SONARCTRL_SET_U 2
14
#define ADM_ID 2
16          #define ADM_GET_NEXT_MANOUVRE 1
```

Listing 5.2: Auszug aus SWModule_ID.h

5.3.3.6 Software

Sonarbildgenerierung

Die Sonarbildgenerierung wird vom Softwaremodul SonarImgProc realisiert. Es sammelt zum einen die eingehenden scanlines des Sonars und empfängt zum anderen dessen sweep-Signal, welches anzeigt, daß der Scan eines Sektors vollständig durchgeführt wurde. Daraufhin wird aus den einzelnen scanlines das Sonarbild zusammengesetzt.

Das Softwaremodul wird durch die gleichnamige Primitive SonarImgProc nachgebildet. Im Gegensatz zu einem realen Sonarbild, welches ein Grauwertbitmap ist, arbeitet die Simulation nur mit den Koordinaten der gefundenen Objekte. Dies entspricht einer abstrahierten Modellierung der internen Vorgänge. Die von processorSWModul abgeleitete Primitive besitzt die zusätzlichen Eingänge *input* und *sweep*. Über den ersten Eingang werden die scanlines empfangen (Matrix mit Objektpositionen) und von einem ObjMatrix-Objekt verwaltet. Die Primitive wertet die Softwarenachricht NAV_NEW_DATA aus, um die Transformation der scanlines in globale Weltkoordinaten durchführen zu können. Dazu wird die Funktionalität von ObjFrameTrans benutzt. Der Parameter *delta* steuert die Genauigkeit, mit der zwei Objekte als eines erkannt werden, indem er eine ε-Umgebung vorgibt. *sweep* erhält das sweep-Signal, mit dem das Sonar einen vollständigen Scan anzeigt. Das Signal startet den Vorgang der Bildgenerierung, der in Abhängigkeit von der Anzahl der scanlines eine variable Menge an Rechenarbeit benötigt[70]. Die Ausgabe des berechneten Bildes erfolgt über die Softwarenachricht SONARIMGPROC_NEW_IMAGE. Diese enthält die Positionen der erkannten Objekte in Form einer $(n \times 6)$-Matrix, wie Tabelle 5.8 verdeutlicht.

0	1	2	3	4	5
Matrix $(n \times 6)$					

Tabelle 5.8: SONARIMGPROC_NEW_IMAGE (Softwarenachricht)

[70] Der ererbte Parameter *cycles* definiert dazu an dieser Stelle die benötigten cycles pro scanline.

Dabei entsteht zwischen der vom Sonar genutzten, genauen Position und der vom inertialen Navigationssystem gelieferten Position der Hindernisse eine Abweichung. Dies ergibt sich durch den zeitlichen Versatz (Funktion des Softwaremoduls Navigation), mit dem die Positionswerte geliefert werden.

Sonarobjekterkennung

Das Softwaremodul ObjectAnalysis übernimmt die Sonarbildauswertung und verwaltet gleichzeitig die bereits wahrgenommenen Objekte. Zu diesem Zweck wird intern eine Liste derjenigen Objekte angelegt, die bisher erkannt wurden. Die bei der Auswertung eines Bildes gefundenen Objekte werden mit den bisher gesehenen verglichen. Ist ein Objekt bereits zuvor gesichtet worden, wird ein interner Zähler, der die Anzahl der Sichtungen für jedes Objekt registriert, um den Wert eins erhöht. Handelt es sich um ein bislang unbekanntes Objekt, wird es der Liste hinzugefügt. Auf Anfrage gibt ObjectAnalysis alle Objekte aus, die bisher in einem bestimmten Radius um die aktuelle Position erkannt wurden. Ein Objekt gilt dabei dann als erkannt, wenn es eine festlegbare Anzahl von Sichtungen aufweist.

Die Primitive ObjectAnalysis modelliert die Funktionalität von ObjectAnalysis. Sie ist von processorSWModul abgeleitet und besitzt keine zusätzlichen Ein- oder Ausgänge. Der Parameter *delta* definiert die Genauigkeit für die Sichtung von Objekten. Haben zwei Objekte einen Abstand zueinander, der kleiner als *delta* ist, werden sie als ein Objekt betrachtet. *min_detect* gibt die Anzahl der Sichtungen an, ab der ein Objekt als erkannt gilt. Ein neues Sonarbild erreicht die Primitive über die Softwarenachricht SONARIMGPROC__NEW_IMAGE. Zu diesem Zeitpunkt wird die Berechnung der Sichtung von Objekten ausgelöst. Erst wenn diese beendet ist, werden die Objektpositionen mit denen in der Liste verglichen. Dazu wird intern auf die Funktionalität von ObjMatrix zurückgegriffen. Wurde ein Objekt bereits zuvor wahrgenommen, wird der Zähler mit der Anzahl der Sichtungen hochgezählt. Andernfalls wird das Objekt zur Liste hinzugefügt. Im Anschluß werden über die Softwarenachricht OBJECTANAYSIS__ACTUAL_IMAGE die Objektpositionen, die sich in der Entfernung *epsilon* um die aktuelle Position befinden, ausgegeben[71]. Um die aktuelle Position des Fahrzeugs zu kennen,

0	1	2	3	4	5
Matrix $(n \times 6)$					

Tabelle 5.9: OBJECTANAYSIS__ACTUAL_IMAGE (Softwarenachricht)

wird NAV__NEW_DATA ausgewertet. Die Softwarenachricht OBJECTANAYSIS__ACTUAL_IMAGE selbst besteht aus einer $(n \times 6)$-Matrix (siehe Tabelle 5.9).

[71] Eine Ausgabe aufgrund einer Anfrage eines anderen Softwaremoduls ist über die Softwarenachricht UNDEF__ASK_FOR_IMAGE ebenfalls möglich.

Missionscontroller

Der Missionscontroller (MissionControl, MCO) stellt die oberste Entscheidungsinstanz innerhalb der *DeepC*-Softwarestruktur dar. Das Softwaremodul übernimmt damit die Rolle des Kapitäns. Ein Aspekt dieser Rolle ist die Fähigkeit, alle anderen Softwaremodule starten zu können.

Die komplexe Funktionalität wurde für die Simulation lediglich in ihren grundlegenden Bestandteilen implementiert. Die Primitive MissionControl übernimmt diese Aufgabe. Sie ist, wie alle Softwaremodule, von processorSWModul abgeleitet und verfügt zusätzlich über die Ausgänge *log* und *activate*. Mittels *log* werden die internen Logmeldungen nach außen geführt. Als solche betrachtet MissionControl die Softwarenachrichten MHA_NEXT_MANOUVRE, MHA_NEXT_BASIC_MANOUVRE und MHA_FINISHED, die von ManoeuvreHandling stammen. MHA_NEXT_MANOUVRE liefert in der transportierten Nachricht (Missionplan Header) binär kodiert den Zustand der Komponenten mit. Dieser Zustand wird über den Ausgang *activate* ausgegeben. Er dient der Aktivierung von Komponenten, wie zum Beispiel der Beleuchtung. Zu Beginn der Simulation versendet MissionControl den Missionsplan (MCO_NEW_MISSIONPLAN). Dazu besitzt die Primitive den Parameter *MissionPlan*, der den

0	1	2	...	m
Matrix $(n \times m)$				

Tabelle 5.10: MCO_NEW_MISSIONPLAN (Softwarenachricht)

DeepC-Missionsplan, also die dem Fahrzeug zu Beginn einer Mission vorgegebene Einsatzbeschreibung, aufnimmt. Der Missionsplan ist dabei gemäß einem internen Protokoll in einer $(n \times m)$-Matrix codiert (Tablelle 5.10). Daneben startet MissionControl mit Hilfe der Liste, die *StartupModules* definiert, die anderen Softwaremodule. Dabei muß zusätzlich zur Nummer des Moduls (gemäß `$MLD_USER/DeepC/include/SWModule_ID.h`) auch die Nummer der CPU angegeben werden, auf der das Softwaremodul ausgeführt werden soll. Mit beiden Werten kann mittels der von processorSWModul geerbten Methode *start_SWModul()* das gewünschte Softwaremodul gestartet werden.

Manöverhandhabung

Das Softwaremodul zur Manöverhandhabung (ManoeuvreHandling, MHA) übernimmt die Überwachung der Missionsplanabarbeitung. Dazu zerlegt es den Missionsplan in Basisbestandteile (Header, Linien und Kreise) und verwaltet diese. Zusätzlich beantwortet es die Anfragen von SonarCtrl nach zukünftigen Positionen.

Das Softwaremodul wird von der Primitive ManoeuvreHandling modelliert. Sie ist von processorSWModul abgeleitet und verfügt weder über zusätzliche Ein- und Ausgänge, noch über Parameter. ManoeuvreHandling empfängt über MCO_NEW_MISSIONPLAN den Missionsplan und spaltet ihn in seine Basisbestandteile auf. Diese

werden in einer Liste verwaltet, aus der DynamicManoeuvringSystem (ADM) per ADM_GET_NEXT_MANOUVRE das nächste Basismanöver abfragt. Diese Anfragen werden per MHA_NEXT_BASIC_MANOUVRE beantwortet, solange der Plan noch nicht vollständig abgearbeitet wurde. Andernfalls wird die Softwarenachricht MHA_FINISHED versendet (siehe Tabelle 5.12).

0	1	2	3	...	10	11	12	13
Matrix gemäß Tabelle 5.15 oder 5.16								

Tabelle 5.11: MHA_NEXT_BASIC_MANOUVRE (Softwarenachricht)

leer

Tabelle 5.12: MHA_FINISHED (Softwarenachricht)

Handelt es sich bei dem nächsten Basismanöver in der Liste um einen Header, wird zusätzlich zum darauffolgenden Basismanöver die Softwarenachricht MHA_NEXT_MANOUVRE versendet.

0	1	2	...	11	12	13
Matrix gemäß Tabelle 5.14						

Tabelle 5.13: MHA_NEXT_MANOUVRE (Softwarenachricht)

Der Missionplan besteht aus einer Liste von Standardmanövern, die nacheinander abgearbeitet werden. Sie sind aus den Basismanövern Linie und Kreis(-segment) aufgebaut. Um trotz der Zerlegung in die Basismanöver den Anfang eines neuen Standardmanövers identifizieren zu können, wird ein Header eingeführt, dessen Definition Tabelle 5.14 zu entnehmen ist. Es handelt sich bei ihm um eine (1×13)-Matrix. Die Kennzeichnung als Header erfolgt im Feld mit der Nummer Null durch den Eintrag Null. Feld eins (*activeComp*) enthält den Status der aktiven Komponenten in binär codierter Form. Die Felder maxCEP und EconomyMode beinhalten den maximalen CEP[72] sowie die Information über den Sollenergiemodus[73].

Eine Linie als Teilstück des Missionsplanes wird definiert durch den Startpunkt S und den Endpunkt E (siehe Tabelle 5.15). In beiden Punkten muß eine Geschwindigkeit vorgegeben werden (v_S, v_E). Desweiteren wird die Länge der Linie (*length*) sowie der Übergang zum nächsten Element vermerkt.

[72] Circular Error Probable, siehe inertiales Navigationssystem
[73] Beide werden momentan von der Simulation noch nicht verwendet.

0	1	2	3	4-13
0	activeComp	maxCEP	EconomyMode	0

Tabelle 5.14: Missionsplan – Header (Protokoll)

0	1	2	3	4	5	6	7	8	9	10	11	12	13
71	v_S	v_E	ueber	length	0	0	0	S_x	S_y	S_z	E_x	E_y	E_z

Tabelle 5.15: Missionsplan – Linie (Protokoll)

Das Basiselement Kreis ist gemäß Tabelle 5.16 definiert. Der Mittelpunkt A beschreibt die Lage im Raum, r gibt den Radius an. s ist der Winkel, bei welchem das Fahrzeug in den Kreis einfährt, während e den Winkel bezeichnet, bei dem es den Kreis wieder verläßt. direction gibt die Drehrichtung an. Zusätzlich werden die Geschwindigkeiten am Ein- und Austrittspunkt (v_S, v_E), die Länge des Kreissegments (*length*) und wiederum der anschließende Übergang vermerkt.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
75	v_S	v_E	ueber	length	0	0	0	A_x	A_y	A_z	s	e	r	direction

Tabelle 5.16: Missionsplan – Kreis (Protokoll)

Die Softwarenachricht MHA_NEXT_MANOUVRE enthält den Header eines Manövers, gemäß Tabelle 5.14 in einer Matrix kodiert. Per MHA_NEXT_BASIC_MANOUVRE wird jeweils das nächste Basismanöver (Linie oder Kreis) versendet.

Für die Ausrichtung des Sonars wird die zukünftige Position berechnet. Die diesbezüglichen Anfragen von SonarCrtl erfolgen über die Softwarenachricht SONARCTRL_GET_SONARHEADING. Die Anfrage beinhaltet dabei die aktuelle Position P sowie eine Entfernungsangabe. Diese legt den Punkt B auf dem Kurs fest, auf den das Sonar ausgerichtet wird[74]. B wird auf Basis der Manöverliste bestimmt. Die Orientierung der Strecke PB (*heading*) und die Geschwindigkeit im Punkt B (u) werden in eine Antwortmatrix geschrieben (siehe Tabelle 5.17) und als MHA_ANSW_GET_SONARHEADING gesendet.

Basisregler

Der Autopilot (ADM) übernimmt die Basisregelung der Fahrzeugbewegung. Das Modul sorgt dafür, daß das AUV jeweils ein Basismanöver abfährt. Dazu fordert es diese bei Bedarf von ManoeuvreHandling an. Für die Steuerung verwendet das Reglermodul

[74] Vgl. Abbildung 5.18, Seite 124.

0	1
heading	u

Tabelle 5.17: MHA_ANSW_GET_SONARHEADING (Softwarenachricht)

die aktuelle Position und, falls vorhanden, Daten über die Strömung. Aus diesen Daten generiert es die Steuerkommandos für den Antrieb.

Die Primitive DynamicManoeuvringSystem bildet das Softwaremodul nach. Sie ist von processorSWModul abgeleitet und besitzt lediglich einen zusätzlichen Ausgang mit der Bezeichnung *output*. Über diesen werden die Steuerkommandos an das Modell des geregelten Fahrzeugverhaltens (DCM) ausgegeben. Ein Steuerkommando setzt sich aus Vorgaben für die Geschwindigkeit (u), den Kurs (psi) und die Tiefe (z) zusammen. Durch die Softwarenachricht ADM_GET_NEXT_MANOUVRE wird das nächste Basismanöver von ManoeuvreHandling angefordert. Die Nachricht besteht aus einer lee-

leer

Tabelle 5.18: ADM_GET_NEXT_MANOUVRE (Softwarenachricht)

ren Matrix (Tabelle 5.18), die von ManoeuvreHandling mit MHA_NEXT_BASIC_MANOUVRE (das nächste Basismanöver) oder mit MHA_FINISHED (Missionsplan vollständig abgefahren) beantwortet wird. Von Navigation kommen die Informationen zur Position (NAV_NEW_DATA) und Strömung (NAV_NEW_CURRENT_DATA), die für die Steuerung genutzt werden.

Die folgende Auflistung stellt kurz die interne Methode *compute_new_pos_data()* vor, die zur Erstellung des Steuerkommandos (u, psi, z) von der Primitive verwendet wird.

- Zuerst wird geprüft, ob ein Basismanöver beendet ist (*element_finished()*). In diesem Fall wird ein neues Element per ADM_GET_NEXT_MANOUVRE angefordert. *element_finished()* testet, ob das aktuelle Basismanövers fertig abgearbeitet wurde. Für eine Linie geschieht dies über die Entfernung von der aktuellen Position bis zum Endpunkt, für einen Kreis wird der verbleibende Winkel bis zum Verlassen des Kreises genutzt.
- Die Geschwindigkeit wird gemäß der Angaben des Basismanövers gesetzt. Eine Ausnahme stellen hierbei zwei Fälle dar: Erstens wird eine geringere Geschwindigkeit für den Rest des Basismanövers vorgegeben, wenn dies über SONARCTRL_SET_U gefordert wurde, zweitens wird die Fahrt gestoppt, wenn der Missionsplan abgefahren wurde.
- Danach wird per *compute_depth()* die Solltiefe ermittelt. Sie ist eine lineare Interpolation zwischen der Tiefe beim Start und am Ende des Basismanövers. Wenn der Abstand zum Boden bekannt ist, wird ein Sicherheitsabstand eingehalten.

- Im dritten Schritt wird der Kurs bestimmt. Dies übernimmt *compute_heading()*, wozu es je nach Basismanövertyp *heading_for_line()* beziehungsweise *heading_for_circle()* nutzt. Beide liefern jeweils den von der Position abhängigen Sollkurs und den Abstand zur vorgegebenen Trajektorie zurück. Auf Basis des Abstands und der Strömung wird eine Kurskorrektur vorgenommen.

Navigation

Für die Navigation existiert ein eigenes, gleichnamiges Softwaremodul (Navigation). Es empfängt die Sensordaten vom inertialen Navigationssystem und vom Dolog. Nach ihrer Aufbereitung werden sie den anderen Softwaremodulen zur Verfügung gestellt.

0	1	2	3	4	5	6
x	y	z	phi	theta	psi	depth

Tabelle 5.19: NAV__NEW_DATA (Softwarenachricht)

0	1	2
u	v	w

Tabelle 5.20: NAV__NEW_CURRENT_DATA (Softwarenachricht)

Die zugehörige Primitive, die dieses Verhalten modelliert, heißt Navigation. Sie ist von processorSWModul abgeleitet und besitzt neben den dadurch ererbten Ein- und Ausgängen die Eingänge *pos* und *dolog*. Am ersten Eingang wird die Position vom inertialen Navigationssystem entgegengenommen. Vom Dolog erhält die Primitive die Tiefen- und Strömungswerte. Nach dem Eingang von Daten an *pos* oder *dolog* werden diese in Form der Softwarenachrichten NAV__NEW_DATA bzw. NAV__NEW_CURRENT_DATA an die anderen Softwaremodule weitergesendet. Die Nachricht NAV__NEW_DATA besteht aus einer (1×7)-Matrix und enthält neben der aktuellen Position auch den Abstand zum Boden (Tabelle 5.19). NAV__NEW_CURRENT_DATA liefert die Strömungsgeschwindigkeiten als (1×3)-Matrix (Tabelle 5.20).

Sonarsteuerung

Das Sonarsteuerungsmodul richtet nach jedem vollständigen Scan eines Sektors das Sonar neu aus. Dies ist vor allem für das vorausschauende Einsehen von Kurven nötig, die anderenfalls nicht in jedem Fall erfaßt werden können.

Diese Funktionalität wurde in der Primitive SonarCtrl umgesetzt. Der Parameter *forecast* gibt die Entfernung an. Sie beschreibt einen Punkt B auf dem zukünftigen Kurs, auf den das Sonar ausgerichtet wird (siehe Abbildung 5.18). Über den Eingang

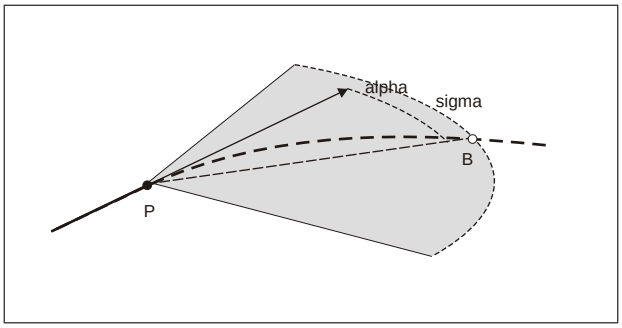

Abbildung 5.18: Sonarsektorwahl

sweep wird das sweep-Signal empfangen, das die Ausrichtung des Sonars steuert. Da das Modul keine Informationen zum eigentlichen Kurs besitzt, fragt die Primitive die Orientierung der Strecke PB mittels der Softwarenachricht SONARCTRL__GET_SONARHEADING von ManoeuvreHandling ab. SONARCTRL__GET_SONARHEADING

0	1	2	3	4	5	6
x	y	z	phi	theta	psi	forcast

Tabelle 5.21: SONARCTRL__GET_SONARHEADING (Softwarenachricht)

besteht, wie Tabelle 5.21 zeigt, aus der aktuellen Position (P) und der Entfernung (*forecast*). Um die Position nutzen zu können, wertet die Primitive die NAV__NEW_DATA Nachricht aus. ManoeuvreHandling antwortet mit MHA__ANSW_GET_SONARHEADING und liefert die gewünschte Orientierung sowie die Geschwindigkeit (siehe Tabelle 5.17). Auf Basis dieser Informationen berechnet SonarCtrl den neuen Sonarscanbereich. Dazu wird aus der Orientierung von PB und der des Fahrzeugs der benötigte Winkel α berechnet. Zu α wird jeweils die halbe Sektorbreite σ hinzugefügt ($\alpha \pm \sigma(u)$). Die Größe des Sektors ist dabei geschwindigkeitsabhängig. Dieses Vorgehen wählt den Sektor so, daß seine Mitte auf einen Punkt B zeigt, wie Abbildung 5.18 veranschaulicht. Bei der Sektorberechnung wird zusätzlich darauf geachtet, daß der aktuelle Kurs innerhalb des Sektors liegt. Überdeckt der Sektor nicht mehr die Trajektorie direkt vor dem Fahrzeug, wird die Geschwindigkeit herabgesetzt, was den Sektor gemäß einem vorgegebenen Zusammenhang von Sektorgröße und Geschwindigkeit vergrößert. Für das Herabsetzen der Geschwindigkeit wird die Softwarenachricht SONARCTRL__SET_U verwendet (siehe Tabelle 5.22). Der ermittelte Sektor wird über den Ausgang

0
u

Tabelle 5.22: SONARCTRL__SET_U (Softwarenachricht)

set_sonar dem Sonar vorgegeben.

5.3.3.7 Umwelt

ObjNetCDF

Das Objekt ObjNetCDF kapselt den Zugriff auf NetCDF[75]-Dateien. Zu diesem Zweck greift es auf die Bibliothek netCDF zurück ([Uni]). Das NetCDF-Datenformat wird in vielfältigen Anwendungen zum Speichern wissenschaftlicher Daten verwendet. Es unterstützt den Einsatz multidimensionaler Daten, die im Interesse der besseren Weiterverwendung selbstbeschreibende Namen erhalten. Im NetCDF-Datenformat liegen die Höhendaten für das Seebodenmodell vor.

Ein ObjNetCDF-Objekt verwaltet jeweils den Zugriff auf die Daten einer Datei. Über die Methode *LoadFile()* wird der Inhalt einer Datei in das Objekt transferiert. Die darin enthaltenen Informationen über das beschriebene Gebiet lassen sich über diverse Methoden auslesen. *getArea()* gibt den Bereich (lat, lon) aus, über den das Objekt Daten enthält. Bei den Daten handelt es sich um eine Matrix mit den jeweiligen Höhenwerten. *getRowNumber()* liefert die Anzahl der Zeilen, *getColumnNumber()* die Anzahl der Spalten zurück. Die zu jedem Höhenwert gehörende Position kann über *getLat()* und *getLon()* für eine Zeile beziehungsweise Spalte abgefragt werden. Mittels *getByRC()* ist der Zugriff auf den zugehörigen Höhenwert möglich.

Damit auch auf die Werte, die nicht auf dem Raster liegen, zugegriffen werden kann, bietet das Objekt die Methode *get()*. Sie nimmt eine bilineare Interpolation zwischen den benachbarten vier Werten vor.

Für die direkte Zusammenarbeit mit den Primitiven Sonar und Dolog unterstützt ObjNetCDF die Suche nach einem Schnittpunkt zwischen einer Gerade und dem Bodenmodell. Diese Suche wird durch die Methode *getPing()* realisiert. Sie erhält den Start- (S) und den Endpunkt (E) der Gerade, welche sich jeweils aus den Komponenten (lon, lat, z) zusammensetzen. Zurückgegeben wird eine Aussage über den Erfolg der Suche und gegebenenfalls die Entfernung zwischen dem Startpunkt und dem Schnittpunkt mit dem Boden. Das hierbei verwendete Verfahren basiert auf geometrischen Überlegungen. Zuerst erfolgt die Projektion der Geraden SE auf die (lat, lon)-Ebene, womit die zugehörige Felder ermittelt werden. Über einen Test der Höhenwerte entlang der Geraden wird festgestellt, ob der Boden durchdrungen wird. Über die Methode *setPingSearchstepWidth()* kann die dabei verwendete Schrittweite beeinflußt werden, welche sich auch über *getPingSearchstepWidth()* abfragen läßt.

Seebodenmodell

Das Seebodenmodell stellt ein digitales Höhenmodell bereit, mittels dessen der Seeboden in die Simulation integriert wird.

[75] network Common Data Form

Die Primitive BOTTOM übernimmt diese Aufgabe. Sie verwaltet die Bodendaten, die im NetCDF-Datenformat vorliegen. Die Topologiedaten stammen vom amerikanischen National Geophysical Data Center ([Nat04]) und haben eine Auflösung von 2 Bogenminuten. Sie sind in separaten Dateien als Kacheln für einen bestimmten (lat, lon)-Bereich beziehbar. Abbildung 5.19 zeigt beispielhaft den Bereich von -1 bis 0 (lon) und 2 bis 3 (lat). Die zugehörigen Tiefen sind farbcodiert dargestellt.

Für den einfachen Umgang mit den Daten wurde der Zugriff auf ein spezialisiertes Softwareobjekt (ObjNetCDF) ausgelagert. Die geladenen Bodendaten werden in einer internen Liste gehalten. Der Parameter *filedir* legt das Verzeichnis fest, in dem sich die Dateien der Bodendaten (zu erkennen an der Endung *.cdf) befinden. *start_lat_lon* beschreibt die Startposition des Fahrzeugs in (lat, lon).

Die Primitive BOTTOM lädt bei Bedarf automatisch benachbarte Gebiete nach. Zu diesem Zweck müssen die Dateien einer vorgegebenen Namenskonvention genügen, welche den zur Verfügung gestellten Bereich im Namen der Datei kodiert.

> Die Datei für die Position (1.100, -4.103) trägt die Bezeichnung 001N-005W.cdf. Um sie zu bilden, wird jeweils der kleinste ganze Wert gesucht und auf drei Stellen gebracht. Für positive lat-Werte wird ein 'N', für alle anderen Werte ein 'S' angefügt. Analog verhält es sich mit den lon-Werten, welche ein 'E' für positive und ein 'W' für negative Werte erhalten. Die Endung lautet immer ".cdf".

Die Aufgabe von BOTTOM liegt in der koordinierten Bearbeitung der Suchanfragen (Pings). Sie erhält die Primitive am Eingang *ask_ping* und beantwortet sie über den Ausgang *answ_ping*.

Der Algorithmus zum Finden eines Schnittpunkts teilt zuerst die Suchanfrage an den Grenzen der beteiligten Bereiche auf. Dabei wird nur der mittlere Suchstrahl der Anfrage verwendet. Abbildung 5.20 verdeutlicht diesen Vorgang. Dabei stellen die gestrichelten Linien die Grenzen eines Bereiches dar. Die Suchanfrage, hier als Gerade SE dargestellt, überschreitet am Punkt C die Bereichsgrenze. Dementsprechend findet eine Unterteilung in zwei Einzelanfragen (SC, CE) an jeden der Bereiche statt. Diese werden an das zuständige ObjNetCDF-Objekt weitergeleitet, die Ergebnisse ausgewertet (kein Schnitt, $|CB|$) und als Antwort zurückgegeben ($|SB|$).

Hindernisse/Objekte

Neben dem Seeboden beinhaltet die Simulation Hindernisse[76] in Form punktförmiger Objekte mit einer fixen Position.

Die Primitive OBJECTS dient dazu, die Hindernisse in die Simulation zu integrieren. Über den Parameter *obstacles* werden die Positionen der Hindernisse vorgeben. Zu diesem

[76] engl.: obstacles

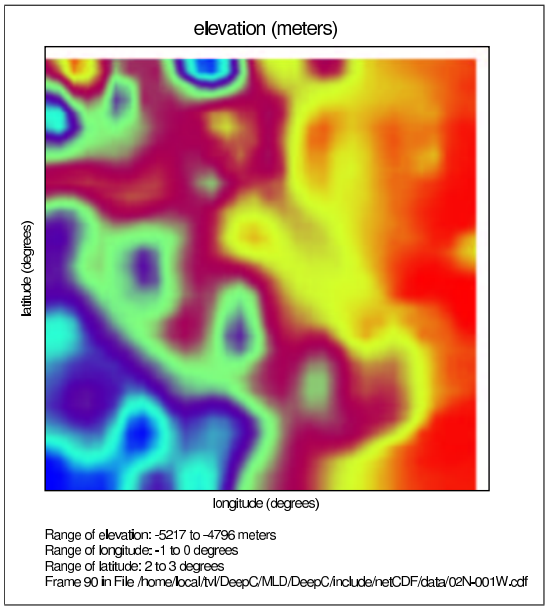

Abbildung 5.19: Seeboden (Falschfarbendarstellung)

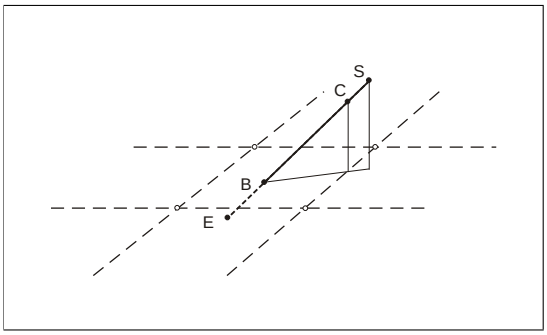

Abbildung 5.20: Suche im Bodenmodell

Zweck nimmt er eine $(n \times 6)$-Matrix auf, in welcher jede Zeile ein Hindernis beschreibt[77]. Dies hat den Vorteil, daß jede Mission eine beliebige Anzahl n von Hindernissen aufweisen kann. Der Eingang *ask_ping* nimmt die Anfragen an, während sie über den Ausgang *answ_ping* beantwortet werden.

Intern wird für jede Anfrage getestet, ob sich ein Objekt innerhalb des Suchraums befindet. Dies geschieht über ein einfaches geometrisches Verfahren. Es nutzt aus der Suchanfrage die Punkte S (Startpunkt des Suchstrahls), E (Endpunkt des Suchstrahls), E_h und E_v (Breite und Höhe der Pyramidengrundfläche). Die folgende Auflistung beschreibt die einzelnen Schritte, die Abbildung 5.21 geometrisch verdeutlicht.

- Die Strecke SE wird als Normale einer Ebene E_1 aufgefaßt, in der der Punkt P liegt. P ist dabei der zu untersuchende Punkt. Die Ebenengleichung lautet:

$$E_1 : \; SE \cdot x + d = 0.$$

  d läßt sich durch P bestimmen ($d = -SE \cdot P$).

[77] Für die Beschreibung eines punktförmigen Objektes wären drei Koordinaten ausreichend. Aus Gründen der Konsistenz - alle Positionsangaben in der Simulation verwenden eine (1×6)-Matrix - und der Erweiterbarkeit wurde jedoch die größere Matrix gewählt. Dabei werden aber nur die ersten drei Komponenten als x, y, und z interpretiert und verwendet. OBJECTS ist leicht auf beliebige Objekte erweiterbar. Dazu muß lediglich der interne, nachfolgend beschriebene Test auf Berührung des Objekts mit der Suchpyramide angepaßt werden.

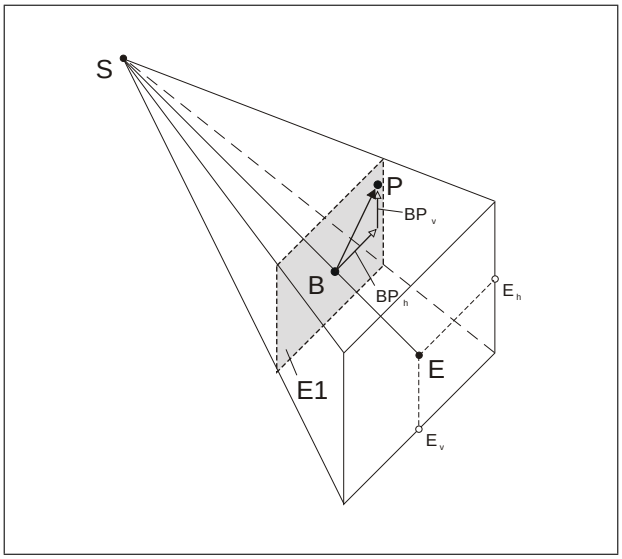

Abbildung 5.21: Geometrische Anordnung beim Test des Suchraums

- Durch S und E läßt sich eine Geradengleichung legen:

$$G_1: \; x = S + \alpha \cdot SE.$$

- Der Durchstoßpunkt B der Geraden G_1 durch die Ebene E_1 ergibt sich bei einem α gemäß[78]:

$$\alpha = \frac{(P-S) \cdot SE}{SE \cdot SE}.$$

  Die Größe von α gibt dabei an, ob B zwischen S ($\alpha = 0$) und E_1 ($\alpha = 1$) liegt.

- Der Vektor BP, der in E_1 liegt, wird gebildet. Aus der Suchanfrage werden die Vektoren e_v und e_h extrahiert. Sie ergeben sich aus E und E_v respektive E_h ($e_v = E - E_v$). BP wird mit ihrer Hilfe in die Richtungsbestandteile BP_v und BP_h zerlegt.

$$BP_v = \frac{BP \cdot e_v}{|e_v|}$$

- Für jeden der beiden Richtungbestandteile wird geprüft, ob P innerhalb des Rechtecks liegt, das von e_v und e_h aufgespannt wird. Dessen Größe ist abhängig von der Entfernung des Punktes von S und damit proportional mit dem Faktor α. Überprüft werden:

$$\begin{aligned} |BP_v| &<= \alpha \cdot |e_v|, \\ |BP_h| &<= \alpha \cdot |e_h|. \end{aligned}$$

[78] Dazu werden die beiden Gleichungen ineinander eingesetzt und nach α umgestellt.

Sind beide Bedingungen erfüllt, liegt der Punkt P im Suchraum.

Diese Schritte werden für jeden der Punkte durchgeführt. Anschließend wird aus allen Punkten innerhalb des Suchraums derjenige mit dem kleinsten α ausgewählt. Bei ihm handelt es sich um den Punkt, der am weitesten im Vordergrund liegt.

Zur Verwaltung der Hindernisse wird intern auf die Funktionalität von ObjMatrix zurückgegriffen.

Umweltkontrollstruktur

Die Primitive envCoordinator übernimmt die Koordination der einzelnen Anfragen (Pings) von Sonar oder Dolog. Sie leitet diese an die angeschlossenen Subkomponenten (objects und bottom) weiter. Desweiteren setzt sie die Einzelantworten in ein gemeinsames Ergebnis um und leitet es an das anfragende Objekt zurück.

Für die Verbindung zu den anfragenden Objekten existieren der Eingang *Input* und der Ausgang *Output*. Zur Anbindung der Umweltprimitiven werden die Eingänge *SubInput* (Multiport) und die Ausgänge *SubOutput* (Multiport) verwendet. Zu den anfragenden Objekten ist für das Sonar und das Dolog jeweils eine Verbindung nötig. Eine Erweiterung auf eine größere Anzahl anfragender Objekte oder Subkomponenten ist problemlos möglich.

Jede Anfrage muß dem im Zusammenhang mit der Primitive Sonar vorgestellten Protokoll[79] genügen und wird an die Subkomponenten weitergeleitet. Die zurückgegebenen Einzelergebnisse werden ausgewertet, sobald alle Antworten eingegangen sind. Als Resultat dieses Prozesses liegt derjenige Punkt vor, an dem innerhalb des Suchraums zuerst ein Hindernis oder der Boden berührt wird. Dieser Punkt wird an das anfragende Modul zurückgesendet, wozu intern der verwendete Eingang vermerkt wird.

Obj3DVectorMatrix

Das Objekt Obj3DVectorMatrix stellt eine dreidimensionale Matrix zur Verfügung und ermöglicht eine trilineare Interpolation zwischen benachbarten Werten. Bei den einzelnen Elementen der Matrix handelt es sich um Vektoren im dreidimensionalen Raum. Abbildung 5.22 bietet eine beispielhafte Visualisierung einer solchen Matrix.

Die Methode *size()* legt die Größe der Matrix fest, während *clear()* alle enthaltenen Werte löscht. Über die Methode *add()* lassen sich Elemente hinzufügen, über *get()* auslesen. Mittels *triLinearInterpolation()* können Zwischenpositionen ermittelt werden. Die

[79] Vgl. Tabelle 5.3 auf Seite 108.

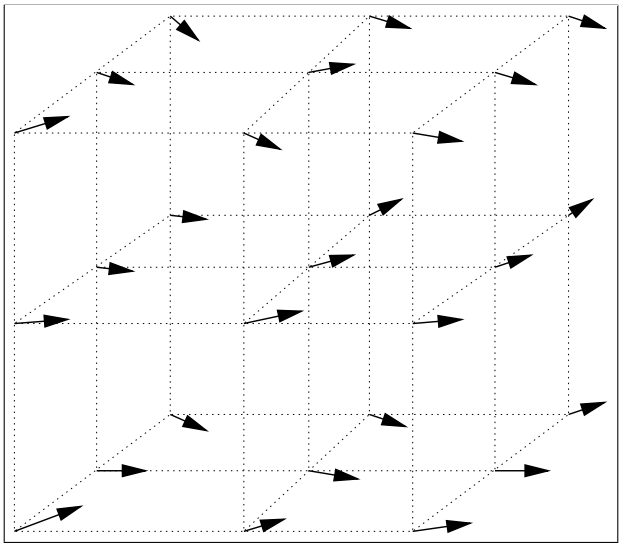

Abbildung 5.22: Dreidimensionales Vektorfeld

trilineare Interpolation wird dabei nach folgender Formel durchgeführt ([RXY+]):

$$V_{xyz} = \begin{bmatrix} (1-x)\cdot(1-y)\cdot(1-z) \\ x\cdot(1-y)\cdot(1-z) \\ (1-x)\cdot y\cdot(1-z) \\ (1-x)\cdot(1-y)\cdot z \\ x\cdot(1-y)\cdot z \\ (1-x)\cdot y\cdot z \\ x\cdot y\cdot(1-z) \\ x\cdot y\cdot z \end{bmatrix}^T \cdot \begin{bmatrix} V_{000} \\ V_{100} \\ V_{010} \\ V_{001} \\ V_{101} \\ V_{011} \\ V_{110} \\ V_{111} \end{bmatrix}.$$

Die einzelnen V sind dabei die acht Vektoren der Eckpunkte, V_{xyz} ist der Ergebnisvektor für einen Punkt innerhalb des würfelförmig aufgespannten Raumes.

Strömungsmodell

Das Strömungsmodell bildet die Strömungsverhältnisse des Meeres mit Hilfe eines Vektorfeldes nach.

Diese Funktion übernimmt die Primitive CURRENT. Über den Parameter *path* wird der Pfad festgelegt, unter dem die Dateien mit den Strömungsdaten abgelegt sind. Die Benennung entspricht dabei dem bei ObjNetCDF vorgestellten Schema, während die Dateiendung in diesem Fall "*.cur" lautet. Das folgende Listing gibt das verwendete Dateiformat wieder.

```
3 4 5 6 -1000 0
10 10 20
1 1 1 <vx vy vz>
1 1 2 <vx vy vz>
...
10 10 20 <vx vy vz>
```

Listing 5.3: Dateiformat für die Strömungsdaten

Die erste Zeile definiert die zugehörigen Bereiche für die Position (lat, lon) und die Tiefe. Danach beschreibt die zweite Zeile die Anzahl der Unterteilungen für jede der Achsen. Im Anschluß folgen die Werte für die Matrix, wobei zuerst der Index, danach der zugehörige Vektor (<vx vy vz>) aufgeführt wird.

Für die interne Speicherung der Daten wird ein 3DVectorMatrix-Objekt genutzt. current erhält die aktuelle Position über den Eingang *pos*. Die Primitive rechnet diesen Wert in die (lat, lon)-Position um und prüft, ob das Strömungsfeld für diese Position schon geladen ist. Ist dies nicht der Fall, wird das zugehörige Feld automatisch nachgeladen. Daraufhin ermittelt die Primitive für die gegebene Position den Strömungvektor und gibt diesen über den Ausgang *current* aus.

5.3.3.8 Weitere Objekte

ObjMySqlHandle

Das Objekt ObjMySqlHandle dient dem Umgang mit einer MySQL-Datenbank. Es besitzt die Methoden *openDB()* und *closeDB()* zum Öffnen respektive Schließen der MySQL-Datenbankverbindung. *isConnected()* gibt den Status der Verbindung zurück, während *pingDatabase()* diesen prüft. Über *existsTable()* kann die Existenz einer bestimmten Tabelle geprüft werden. Mittels *createTable()* läßt sich eine neue Tabelle hinzufügen. Daneben stellt das Objekt Methoden zum Auslesen (*queryDatabase()*), Anhängen (*appendDataSet()*) und Aktualisieren (*updateDataSetByID()*) der Daten einer Tabelle bereit. Die Methode *mysqlError()* liefert den letzten aufgetretenen Fehlercode zurück.

ObjDatabaseHandle

ObjDatabaseHandle kapselt die gesamte Kommunikation mit der Datenbank, wozu es die Funktionalität von ObjMySqlHandle nutzt. Für die Arbeit mit dem Objekt stehen diverse Methoden zur Verfügung. So dienen *openConnection()* und *closeConnection()* dem Auf- und Abbau einer Verbindung. Über *initPlot()* registrieren sich die Ausgabeplugins beim Plugin. Dabei erhalten sie von diesem einen Schlüssel, den sie bei der weiteren Kommunikation verwenden. Dadurch wird sichergestellt, daß die Daten in der richtigen Tabelle gespeichert werden. Diese Daten werden über die Methode *sendData()* an das Objekt übergeben. Durch die *setEndTime()*-Methode wird die zeitliche Länge der

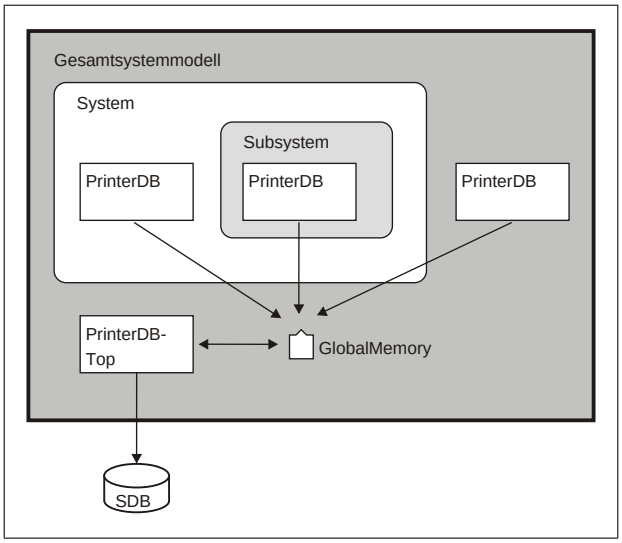

Abbildung 5.23: Datenbankzugriff aus MLDesigner heraus

Simulation in der Datenbank vermerkt, während *error()* jeweils den letzten Fehlercode der Datenbankverbindung liefert. Das verwendete Datenbankformat zur Speicherung der Daten wird ausführlich in der Beschreibung zum Framework vorgestellt ([Lie04]).

Datenbankanbindung der Simulation

Bei der Vorstellung des Konzeptes für das Framework wurde mehrfach darauf verwiesen, daß die Simulation an die Simulationsdatenbank angekoppelt ist. Hierfür sind drei speziell für diesen Zweck entwickelte Primitiven mit den Bezeichnungen PrinterDBTop, PrinterDB und PrinterDB_M zuständig.

Abbildung 5.23 bietet zunächst eine graphische Darstellung des Vorgehens bei der Ausgabe der Simulationsdaten und verdeutlicht deren prinzipielle Funktionsweise. Das ML-Designer-Modell beinhaltet eine PrinterDBTop-Primitive, mehrere PrinterDB-Primitiven, die auf verschiedene Modellebenen verteilt sind, sowie einen globalen Speicher (Memory). PrinterDBTop stellt mittels eines ObjDatabaseHandle-Objekts die zentrale Datenbankverbindung beim Start der Simulation her und hinterlegt einen Zeiger auf das ObjDatabaseHandle-Objekt im globalen Speicher. Die PrinterDB-Primitiven lesen diesen Zeiger aus und können, da sie das Objekt kennen, auf dessen Funktionalität - das Senden von Daten an die Datenbank - zugreifen. Damit eine geordnete Kommunikation erfolgt, registriert sich jede PrinterDB-Primitive beim Start der Simulation bei ObjDatabaseHandle und erhält dabei ein Handle für die folgende Kommunikation. Am Ende der Simulation wird die Verbindung durch PrinterDBTop beendet.

Dieses Vorgehen weist einige Vorteile auf:

1. Die Anzahl der Primitiven, die Daten in die Datenbank schreiben können, ist nicht begrenzt.
2. Die Ausgabeprimitiven können auf verschiedenen Ebenen des Modells liegen, da der globale Speicher aus dem gesamten Modell heraus lesbar ist.
3. Die Programmierung der Ausgabeprimitiven vereinfacht sich durch die Verwendung von ObjDatabaseHandle.
4. Änderungen am Datenbankformat werden vor den Primitiven versteckt, da diese lediglich auf die Methoden des Objektes ObjDatabaseHandle zugreifen.
5. Eine zentrale Konfiguration der Datenbankverbindung erfolgt über die Parameter von PrinterDBTop.
6. Die Übermittlung von Daten an die Datenbank kann für das gesamte Modell zentral abgeschaltet werden, da die gesamte Kommunikation über eine einzige Verbindung erfolgt.

Nachstehend werden die beteiligten Primitiven detailliert vorgestellt.

Die Primitive PrinterDBTop öffnet zu Beginn der Simulation eine Datenbankverbindung und ist daran beteiligt, diese für die Ausgabeprimitiven zur Verfügung zu stellen. Da es sich dabei um eine Ableitung von der MLDesigner-Standardprimitive GlobalMemory handelt, kann sie auf globale Memory-Elemente zugreifen. Von GlobalMemory ererbt sie auch den Parameter *MemoryName*. Durch diesen wird dasjenige globale Memory-Element angegeben, welches den Zeiger auf das interne ObjDatabaseHandle-Objekt aufnimmt. Das Memory-Element besitzt dazu den Typ Root.Pointer. Der Parameter *connect* legt fest, ob das Modell überhaupt Daten in die Datenbank schreiben soll. Auf diese Weise kann die Speicherung der Daten an zentraler Stelle abgeschaltet werden. *mission* nimmt die aktuelle Missionsnummer auf, die in die Datenbank geschrieben wird. Die übrigen Parameter dienen zur Definition der Datenbankverbindung. *database* enthält den Namen der Datenbank, *host* den Namen des Rechners, auf dem diese läuft. *port* gibt den Port an, auf dem die Datenbank auf Anfragen wartet, während *user* den Benutzername für die Datenbank benennt. Das Passwort wird aus Sicherheitsgründen nicht als Parameter gespeichert, sondern jeweils am Anfang einer Simulation abgefragt. *db_model_description* nimmt eine Beschreibung des Modells auf, die ebenfalls in die Datenbank geschrieben wird. Am Ende der Simulation wird die Simulationsendzeit gesetzt und anschließend die Datenbankverbindung geschlossen.

PrinterDB ist ebenfalls von GlobalMemory abgeleitet. Insofern verfügt auch sie über den Parameter *MemoryName*. Er wird genutzt, um die Zeiger auf ObjDatabaseHandle zu lesen. Der Parameter *InfoType* beschreibt den SQL-Datentyp für die Daten. *Name* ist der Name, unter dem diese Daten in der Datenbank gespeichert werden. Er muß über das gesamte Modell hinweg eindeutig sein. Durch *DB_variable_description* läßt sich eine Beschreibung der Daten in der Datenbank hinterlegen. Ob die Primitive überhaupt Daten in die Datenbank senden soll, legt der Parameter *Print* fest. Zu Beginn der Simulation registriert sich der Plot bei der Datenbankverbindung. Über den Eingang

Input1 werden während der Simulation Daten empfangen. Wenn sie übermittelt werden sollen (Parameter *Print*), werden sie inklusive eines Zeitstempels in die Datenbank geschrieben.

Die Primitive PrinterDB_M entspricht in ihrer Funktionsweise und den Parametern im wesentlichen PrinterDB. Die Besonderheiten dieser Primitive hängen jedoch damit zusammen, daß sie als Eingang für den speziellen Datentyp float_env_matrix vorgesehen ist. Dieser definiert eine Matrix, bestehend aus Elementen des Typs Float. Da insofern ihr Typ festgelegt ist, entfällt der Parameter *InfoType*. Andererseits muß *Name* zusätzlich zum Namen auch die Bezeichnungen der Spalten enthalten. Dies resultiert daraus, daß diese Matrizen im *DeepC*-Modell genutzt werden, um Vektordaten auszutauschen. Ein Positionsvektor wird durch eine (1×6)-Matrix nachgebildet und würde bei *Name* den Eintrag "POSAUV x y z phi theta psi" enthalten. Die anderen Parameter sind identisch mit denjenigen von PrinterDB.

5.4 Parameter des Gesamtsystemmodells

5.4.1 Systemparameter

Das Gesamtsystemmodell des *DeepC*-AUV, wie es der vorangegangene Abschnitt detailliert vorgestellt hat, verfügt über eine Reihe von Systemparametern[80], die die Auslegung der einzelnen Komponenten beschreiben. Tabelle 5.23 bietet eine Übersicht über diese Parameter, nachdem die Beschreibung ihrer Funktionen im Zusammenhang mit der Vorstellung der zugehörigen Primitiven bereits im vorangegangenen Abschnitt 5.3.3 erfolgte.

Die Anordnung der aufgelisteten Parameter erfolgt entsprechend den vorgestellten Sichten. Durch ihre wiederkehrenden Parameter (*Wattage* bzw. *CPU*, *cycles* und *priority*) sind die von EnBaseDeviceRS und procSWModule abgeleiteten Primitiven deutlich erkennbar.

Wie Abbildung 5.24 veranschaulicht, wird die Zuweisung von Werten zu den Systemparametern mit MLEditor vorgenommen. Das Programm zeigt die Parameter in Form einer Baumstruktur an (links), neben der eine tabellarische Übersicht über die Parameterwerte angeordnet ist (rechts). Die Eingabe der Werte erfolgt mit Hilfe von Plugins, die die Wertezuweisung in Abhängigkeit vom Datentyp übernehmen[81].

[80] Vgl. Abschnitt 3.2.3 auf Seite 51 zu den unterschiedlichen Parametertypen.
[81] Eine direkte Vorgabe des zu verwendenden Plugins ist ebenfalls möglich (siehe [Lie04]).

Objekt	Parameter
PITCH	*a*, *b00*, *b20*, *b21*, *funktion*
INTEGRATOR<#>	*x_min*, *x_max*, *xdot_min*, *xdot_max*
T1<#>	*K*, *x_min*, *x_max*, *xdot_min*, *xdot_max*
PI<#>	*Kr*, *Tn*
LIMITEDLINEAR	*a*, *b*, *ymin*, *ymax*
DETECTLIMIT	*upper_limit*, *lower_limit*
SONAR	*PingAngle*, *PingArea*, *PingDist*, *PingRate*, *dAlpha*, *StartPos*, *Wattage*
DOLOG	*Position*, *PingRate*, *PingDist*, *Wattage*
PROCESSOR	*cycles_per_sec*, *SampleTime*, *distribution_error*, *max_depth*, *master*, *number*, *Wattage*
SONARIMGPROC	*delta*, *CPU*, *cycles*, *priority*
OBJECTANALYSIS	*epsilon*, *min_detect*, *CPU*, *cycles*, *priority*
SONARCTRL	*forecast*, *CPU*, *cycles*, *priority*
DYNAMICMANOEVREINGSYSTEM	*CPU*, *cycles*, *priority*
MANOEUVREHANDLING	*CPU*, *cycles*, *priority*
MISSIONCONTROL	*CPU*, *cycles*, *priority*
NAVIGATION	*CPU*, *cycles*, *priority*
BATTERY	*Capacity*
FUELCELL	*Capacity*, *Power*, *Factor*
ENGINE	*Wattage*
LAMP	*ID*, *Wattage*

Tabelle 5.23: DeepC-Systemparameter

5.4.2 Missionsparameter

Der nachfolgende Abschnitt geht auf die konkrete Behandlung von Missionen im Rahmen der *DeepC*-Simulation ein. Hierzu stellt er zunächst die Missionsparameter des Gesamtsystemmodells vor, um anschließend zu erläutern, wie die Missionen mit Hilfe der Missionsparameter beschrieben werden.

5.4.2.1 Missionsparameter

Eine Aufzählung der Missionsparameter des *DeepC*-Gesamtsystemmodells ist Tabelle 5.24 zu entnehmen.

Sie untergliedern sich in drei Bereiche:

- **Missionsziel**
  Das Missionsziel, also die Aufgabe des Fahrzeugs in einer Mission, wird durch den Parameter *MissionPlan* des Softwaremoduls MISSIONCONTROL beschrieben.

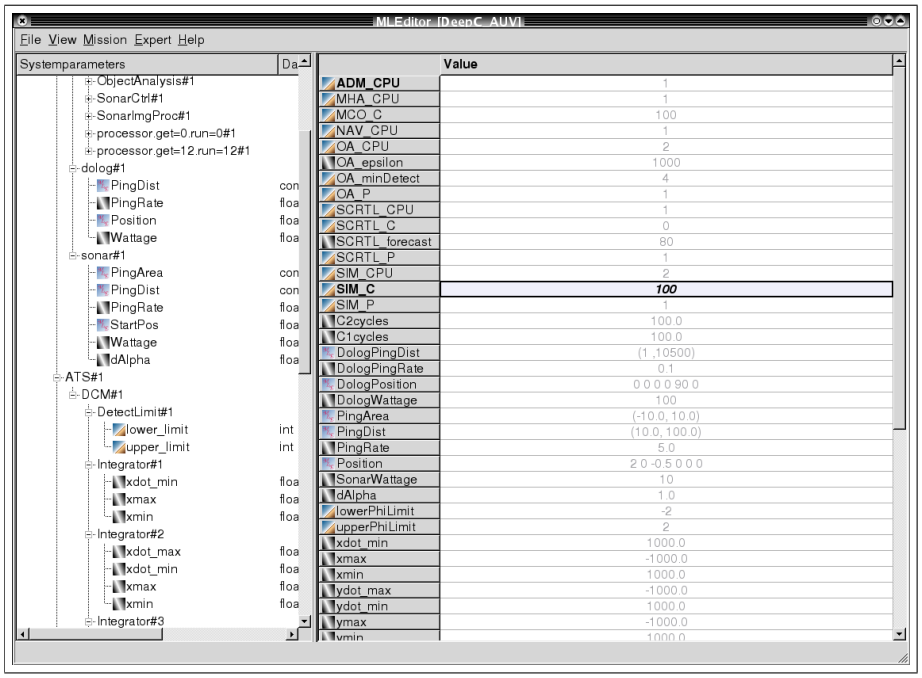

Abbildung 5.24: MLEditor (mit Systemparametern)

Die Missionsnummer wird über den Parameter *mission* (PrinterDBTop) für die Auswertung in die Datenbank übernommen.

- **Konfiguration des Systems**
  Der Anfangszustand des Fahrzeugs wird durch mehrere Parameter beschrieben: Die Startposition und -geschwindigkeit werden über Parameter im Modul DCM eingestellt. MissionControl.*StartupModules* gibt die Softwaremodule an, die am Missionsbeginn vom Prozessor gestartet werden, während Battery.*Level* den Füllstand der Batterie und fuelcell.*Level* den der Brennstoffzelle festlegen. Die *Broken*-Parameter der Energieverzweigungen (EnNodes) definieren die Anfangs-

Objekt	Parameter
MissionControl	*MissionPlan*, *StartupModules*
PrinterDBTop	*mission*
DCM	*x0*, *y0*, *z0*, *phi0*, *theta0*, *psi0*, *u0*
Battery	*Level*
fuelcell	*Level*
EnNode	*Broken*
objects	*obstacles*
bottom	*filedir*, *start_lat_lon*
current	*path*

Tabelle 5.24: DeepC-Missionsparameter

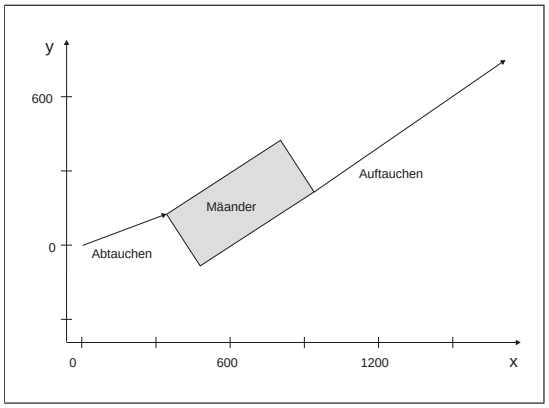

Abbildung 5.25: Manöver der Beispielmission

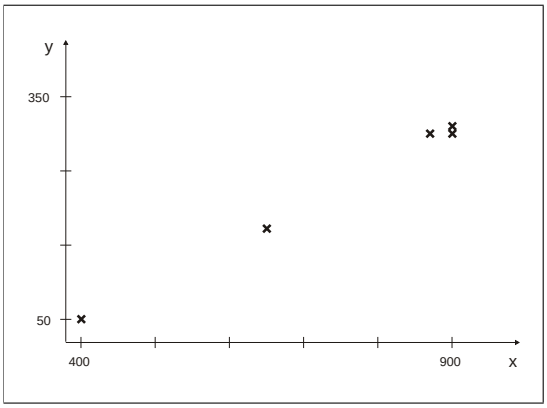

Abbildung 5.26: Hindernisse der Beispielmission

zustände der Energieverbindungen.

- **Konfiguration der Umwelt**
  Die aktuelle Beschaffenheit der Umwelt wird durch vier Parameter bestimmt: OBJECTS.*obstacles* definiert die Hindernisse, BOTTOM.*filedir* und CURRENT.*path* verweisen auf die Verzeichnisse mit den Seeboden- bzw. Strömungsdaten. Der vierte Parameter BOTTOM.*start_lat_lon* enthält die Startsposition des Fahrzeugs.

Die Werteeingabe der Missionsparameter gleicht prinzipiell derjenigen der Systemparameter. Abweichend von dieser zeigt MLEditor jedoch eine Tabelle mit den Werten an. Sie besteht aus einer der Anzahl der Missionen entsprechenden Anzahl von Spalten. Die einzelnen Werte sind jeweils mit den Standardwerten vorbelegt und lassen sich anpassen. Die Eingabe selbst erfolgt wiederum über Plugins, die entsprechend des Datentyps eingesetzt werden.

5.4.2.2 Beispielmission

Im Folgenden soll die Beschreibung von Missionen mittels der Missionsparameter anhand eines exemplarischen Nutzungsszenarios erläutert werden. Diese Beispielmission besteht aus einer dreistufige Fahrt: Zunächst taucht das Fahrzeug in eine Tiefe von 100 m ab, absolviert dort einen Mäander[82] und taucht anschließend wieder auf (siehe Abbildung 5.25). Ein Mäander ist dabei ein für Such- oder Meßvorgänge typisches Manöver, wie die Literatur zeigt ([BKZ92], [RT98], [Fle01], [RBC01]). In der Nähe der geplanten Fahrstrecke befinden sich in der Tiefe des Mäanders fünf Hindernisse. Abbildung 5.26 zeigt ihre Lage aus der Draufsicht. Das Ziel der Mission besteht darin, alle diese Hindernisse zu erkennen. Der Ausgangszustand des Fahrzeugs ist dadurch charakterisiert, daß es seine Fahrt vollaufgetankt und funktionstüchtig am Startpunkt der Mission aufnimmt.

[82] Vgl. Abbildung 5.29 auf Seite 142 für das Aussehen eines Mäanders.

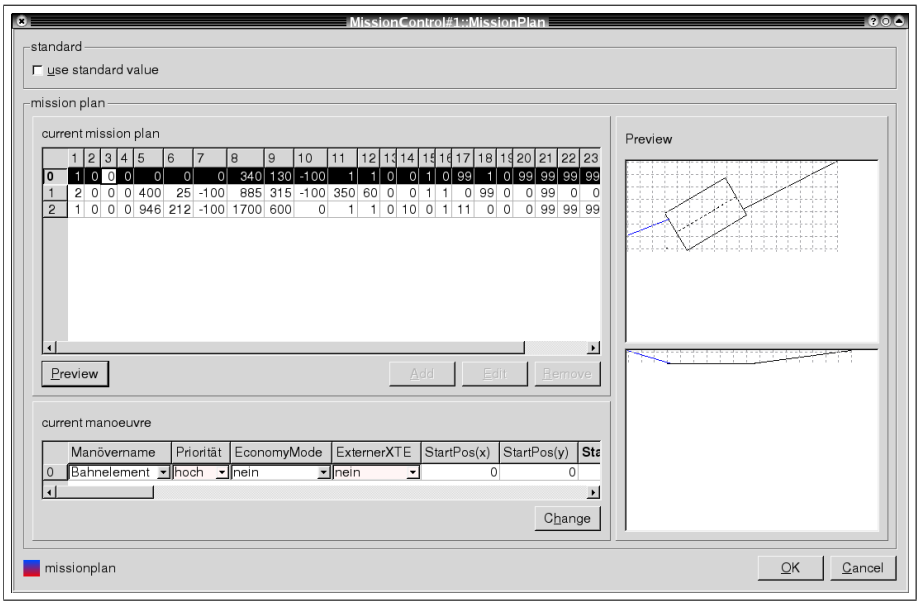

Abbildung 5.27: MLEditor Missionsplan Plugin

Diese Mission soll durch die Missionsparameter nachgebildet werden. Den ersten Teil - die Fahrtroute - beschreibt der Missionsplan. Bei ihm handelt es sich um ein Protokoll, das die zu absolvierenden Manöver festlegt[83]. Das Protokoll wird auch von der Simulation unterstützt, wobei der Missionsplan vom Parameter MissionControl.*MissionPlan* aufgenommen wird.

> Um die Eingabe von Missionsplänen zu vereinfachen, existiert für diese ein spezielles MLEditor-Plugin (siehe Abbildung 5.27). Es gliedert sich in drei Bereiche: Links oben wird der Missionsplan in seiner Simulationsstruktur angezeigt (Matrix mit codierten Informationen). Unterhalb der Matrix befindet sich ein Bereich, in dem je ein Manöver bearbeitet werden kann. An dieser Stelle wird das offizielle Protokoll anzeigt, das heißt, die Informationen werden in verbaler Form beschrieben. Die rechte Seite bietet eine Vorschau über den Fahrtverlauf des Missionsplans (Projektionen auf die x-y und die x-z-Ebene). Durch das Plugin vereinfacht sich die Handhabung wesentlich. Die einzelnen Manöver können bequem eingegeben werden, wobei sämtliche Werte entsprechend des Protokolls übertitelt werden. Über die Vorschaufunktion ist jederzeit der aktuelle Bearbeitungsstand ersichtlich.

objects.*obstacles* nimmt die Objektpositionen der Hindernisse auf. Die Testmission erhält die Nummer Null (PrinterDBTop.*mission*), alle EnNode.*Broken*-Parameter werden auf false gesetzt (Standard). Da das Fahrzeug voll aufgeladen ist, verzeichnen Battery.*Level* und fuelcell.*Level* den Wert 100. Die Startposition wird in den lokalen Nullpunkt gesetzt (DCM.{*x0*, *y0*, *z0*, *phi0*, *theta0*, *psi0*}), an dem das Fahrzeug aus dem Stillstand startet (DCM.*u0*). Global befindet sich das Fahrzeug bei (bottom.*start_lat_lon*). Die Standardverzeichnisse mit den Boden- und Strömungsdaten bleiben unverändert (bottom.*filedir*, current.*path*).

[83] siehe die Beschreibung von ManoeuvreHandling, Seite 119 oder auch [Pfü03a, S. 1235]

Tabelle 5.25 bietet eine vollständige Übersicht über die Werte der Missionsparameter für die Beispielmission.

Parameter	Wert
MissionControl.*MissionPlan*	<Matrix>
MissionControl.*StartupModules*	$(10,1),(1,1),(2,1),(3,1),(4,2),(11,2)$
PrinterDBTop.*mission*	0
EnNode.*Broken* (mehrere)	false
Battery.*Level*	100
fuelcell.*Level*	100
DCM.{*x0*, *y0*, *z0*, *phi0*, *theta0*, *psi0*}	0
DCM.*u0*	0
objects.*obstacles*	$(400,50,-100),(650,172,-100),$ $(870,300,-100),(900,300,-100),$ $(900,310,-100)$
bottom.*filedir*	`$MLD_USER/DeepC/include/netCDF/data/`
bottom.*start_lat_lon*	$(2.1,3.1)$
current.*path*	`$MLD_USER/DeepC/include/current/`

Tabelle 5.25: Beispielmission für *DeepC*

5.4.3 Entwurfsparameter

Die Entwurfsparameter stellen eine Repräsentation der Entwurfsziele dar. Mit ihrer Hilfe wird in der Auswertung überprüft, ob der Entwurf die gestellten Anforderungen erfüllt. Dies geschieht durch den Vergleich von erreichten Werten mit den Vorgaben der Entwurfsparameter.

Durch die Trennung von Auswertung und Simulation im Rahmen des Frameworks sind die Entwurfsparameter nicht mehr im Gesamtsystemmodell vorhanden.

5.4.4 Konstanten

Einen weiteren, bereits in Abschnitt 3.2.3.2 identifizierten Typ von Parametern stellen die Konstanten dar. Sie unterteilen sich in zwei Untergruppen:

- **Naturkonstanten**
  Den ersten Teilbereich stellen Werte dar, die für die Simulation feste Naturkonstanten u.ä. festlegen. Derartige Werte treten im *DeepC*-Gesamtsystemmodell nicht als Parameter auf.

- **Werte für die Simulation**
  Die zweite Gruppe beinhaltet Konstanten, die den Ablauf der Simulation beeinflussen, ohne jedoch zur Gruppe der Missions- oder Systemparameter zu gehören.

  Zu dieser Gruppe zählen beispielsweise die *MemoryName*-Parameter der PrinterDB-Primitiven. Sie verweisen auf das globale Memory-Element und ermöglichen so die Ausgabe der Daten in die Simulationsdatenbank. Dabei dient ihre Existenz allein der Durchführung der Simulation. Die Zuordnung von Parametern ist nicht immer eindeutig möglich[84].

Eine Übersicht zu den wichtigsten Parameter dieser Kategorie liefert Tabelle 5.26.

Objekt	Parameter
PrintDBTop	*MemoryName*, *database*, *host*, *user*, *port*, *connect*, *db_model_description*
PrintDB	*MemoryName*, *Print*, *Name*, *DB_variable_description*, *InfoType*
PrintDB_M	*MemoryName*, *Print*, *Name*, *DB_variable_description*
processor	*report_max_depth*
battery	*SampleTime*, *StopSimulation*
fuelCell	*SampleTime*
DCM	*dt*
EnBaseDevice(RS) (alle abgeleiteten)	*SampleTime*
pitch	*size*

Tabelle 5.26: Konstante Parameter

Die Eingabe der Konstanten erfolgt während der Modellierung in MLDesigner, die erweiterten Fähigkeiten von MLEditor sind hierbei insofern nicht nutzbar.

5.5 Auswertung der Simulationen

5.5.1 Missionsspezifische Auswertung

Nachdem die Parametrisierung des MLDesigner-*DeepC*-Gesamtsystemmodells im vorangegangenen Abschnitt behandelt wurde, präsentiert dieser Abschnitt anhand eines Beispiels, wie die missionsspezifische Auswertung erfolgt und welche Anzeigen sich aus

[84] Verschiedene Parameter, wie zum Beispiel PrintDP.*Print*, lassen sich durchaus auch als Missionsparameter definieren. Im genannten Fall ist durch diese Zuordnung eine Selektion von Simulationsergebnissen bereits in MLEditor festschreibbar, weil je nach Mission nur die ausgewählten Ergebnisdaten von der Simulation in die Datenbank geschrieben werden.

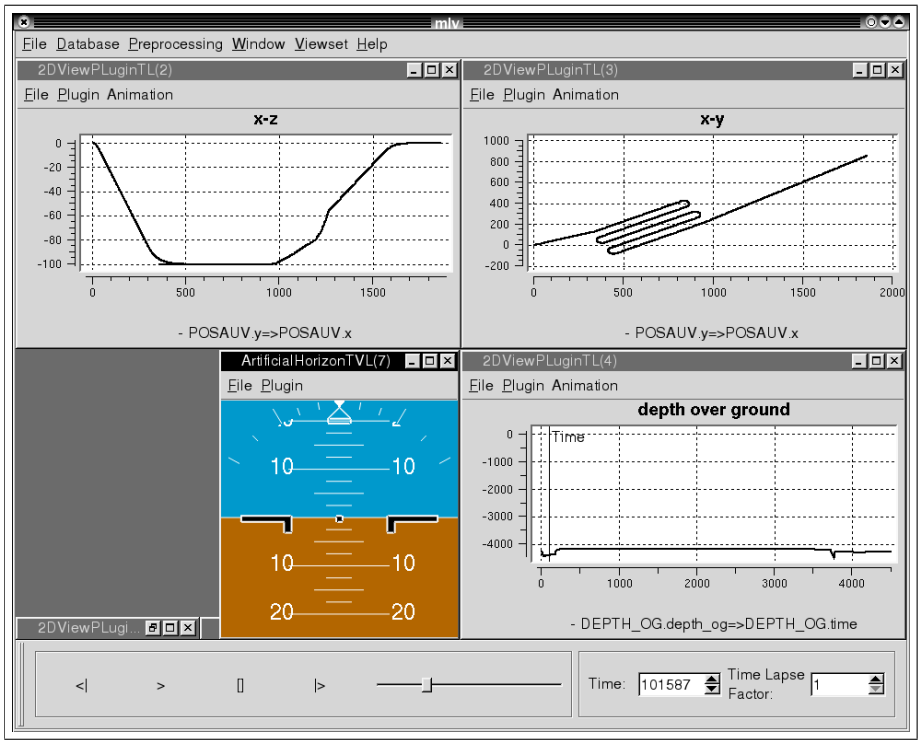

Abbildung 5.28: MLVisor-Hauptfenster

den Ergebnisdaten generieren lassen. Um die Ergebnisdaten zu erhalten, wurden die konfigurierten Modelle zuvor mit MLDesigner simuliert, wobei die entstehenden Ergebnisse in die Simulationsdatenbank geschrieben wurden.

Die Auswertung wird mit Hilfe des für diesen Zweck im Rahmen des Frameworks neu erstellten Programms MLVisor durchgeführt. Abbildung 5.28 zeigt das Programm mit vier Anzeigen (Views), die auf Simulationsergebnissen basieren. Nähere Hinweise für die Arbeit mit MLVisor finden sich in der vom Autor an anderer Stelle vorgestellten Beschreibung des Frameworks ([Lie04]).

Die Simulationsergebnisse der Beispielmission[85] werden nachfolgend anhand der Sichten des Modells präsentiert. Die einzelnen Darstellungen sind dabei exemplarisch für die Auswertung aller Simulationsergebnisse.

- **Bewegung:**
  Die Bewegung der Fahrzeugs vollzieht sich, indem der vorgegebene Missionsplan abgefahren wird. Abbildung 5.29 zeigt die sich dabei ergebende Trajektorie des AUV aus der Draufsicht (x-y-Ebene).[86] Deutlich ist das Abfahren des Missionsplanes - besonders das des Mäanders - nachvollziehbar. Aus dieser Sicht lassen sich die Ab- und Auftauchphase jedoch schwierig erkennen. Deshalb stellt Abbildung 5.30 die Trajektorie aus einer seitlichen Perspektive heraus dar (x-z-Ebene).

[85] Vgl. Abschnitt 5.4.2, Seite 135.
[86] Die Abbildung nutzt das TL2DViewPlugin von MLVisor.

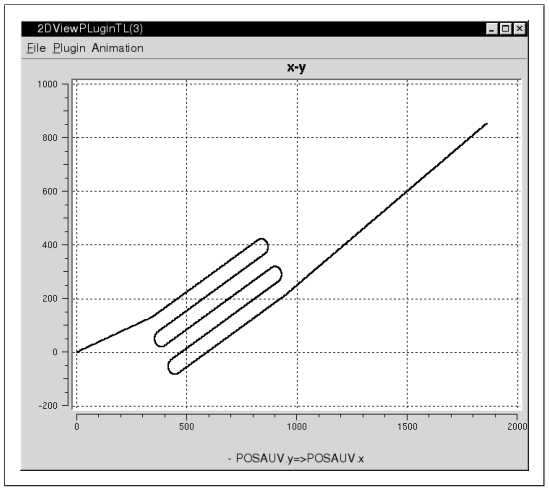

Abbildung 5.29: Missionstrajektorie (x-y-Ebene)

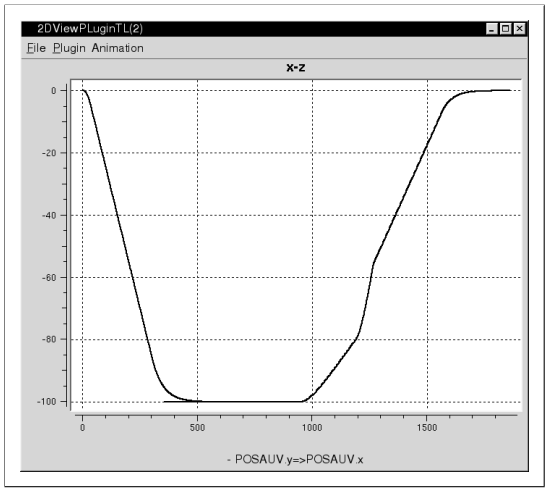

Abbildung 5.30: Missionstrajektorie (x-z-Ebene)

Aus ihr ist das Ab- und Auftauchen besser zu erkennen, wobei der dreistufige Prozeß des Auftauchens besonders gut nachvollziehbar ist.

Durch ihre Projektion auf die Achsen geben die bisherigen Darstellungen die tatsächliche Bewegung im Raum jedoch nur eingeschränkt wieder. Aus diesem Grund steht für die Auswertung zusätzlich eine dreidimensionale Darstellung zur Verfügung (Abbildung 5.31).[87] Sie ermöglicht die Betrachtung der Trajektorie durch Drehung im Raum sowie im animierten zeitlichen Ablauf. Gleichzeitig lassen sich zusätzliche Objekte, wie der Seeboden oder die Wasseroberfläche, in den dargestellten Raum integrieren.

Eine Möglichkeit, eine Vorstellung von der Bewegung des Fahrzeugs aus *dessen* Sicht zu erlangen, bietet die Darstellung seiner Nick- und Roll-Lage mittels eines künstlichen Horizonts. Abbildung 5.32 zeigt dieses aus der Luftfahrt bekannte Instrument.[88]

Ein anderer Aspekt, der sich anhand der bisher vorgestellten Ansichten schwer untersuchen läßt, sind die Zeitpunkte, an denen die einzelnen Manöver des Missionsplanes eingeleitet werden. Das Softwaremodul MissionControl logt die entsprechenden internen Meldungen mit und stellt diese für die Auswertung bereit. Diese Meldungen lassen sich mit der Zeit ihres Auftretens darstellen, was Abbildung 5.33 zeigt.[89] Die Dauer der Mission ist so beispielsweise anhand der letzten Meldung (MHA_FINISHED) exakt ablesbar. Für die Beispielmission sind dies 4053 Sekunden (1 Stunde, 7 Minuten und 33 Sekunden).

[87] Das TL3DViewPlugin, das hier benutzt wird, stellt eine dreidimensionale, auf OpenGL basierende, animierte Darstellung von Bewegungen zur Verfügung.
[88] Die Darstellung erfolgt mit Hilfe des MLVisor-Plugins TVLArtHorizon.
[89] Die Visualisierung wird mit Hilfe des Plugins TVLStringPlugin vorgenommen.

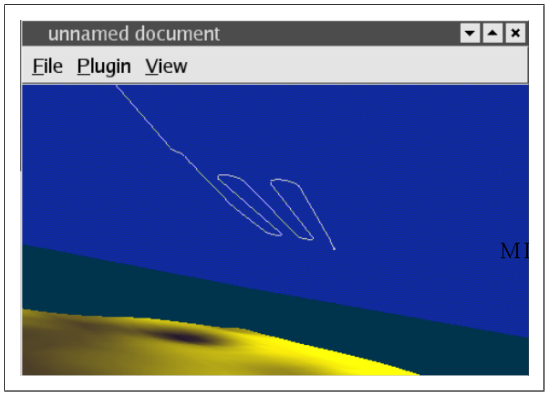

Abbildung 5.31: Trajektorie (3D)

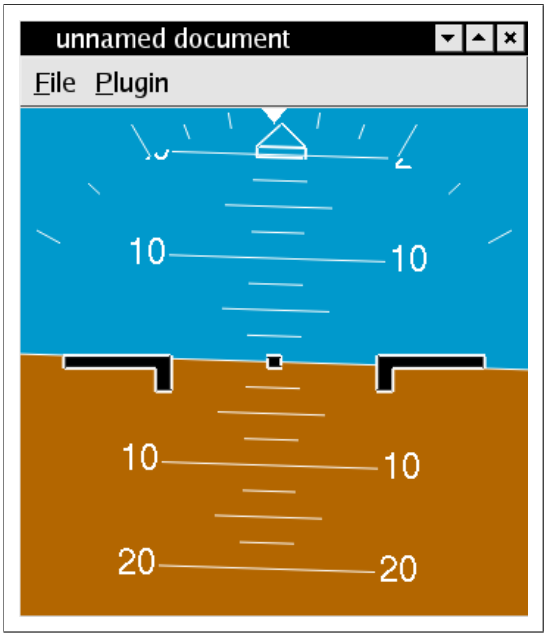

Abbildung 5.32: Künstlicher Horizont

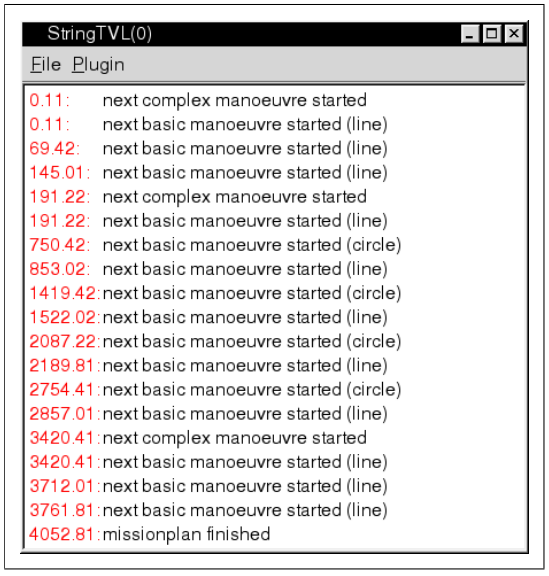

Abbildung 5.33: Protokollierte Missionsschritte

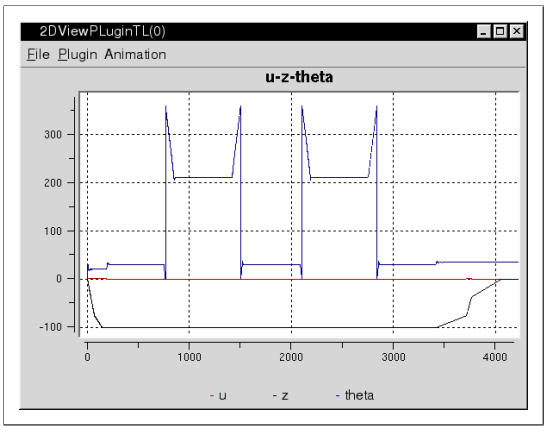

Abbildung 5.34: Steuerkommandos

Abbildung 5.35: Komponetenaktivität

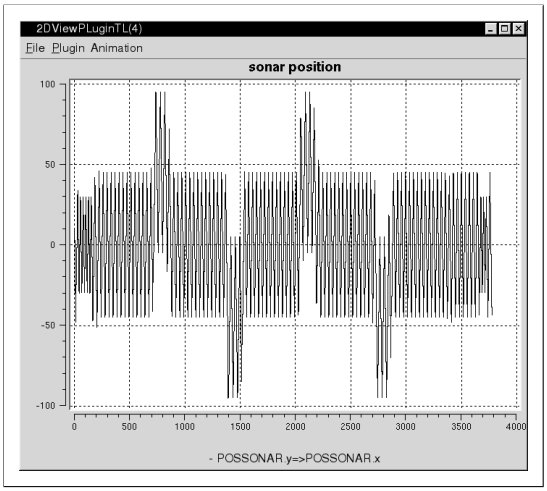

Abbildung 5.36: Sonaransteuerung

Abbildung 5.34 zeigt die Steuerkommandos des DynamicManoeuringSystems, die die gefahrene Trajektorie erzeugen. Durch sie lassen sich die Steuervorgaben des Reglers nachvollziehen. Deutlich sind die Kurs- und die Tiefenvorgabe zu erkennen. Das Kurskommando schwankt zwischen den Werten 0 und 360 °. Die waagerechten Linien repräsentieren einen geraden Kursverlauf, während die Schrägen den Kurven des Mäanders mit ihrer konstanten Kursänderung entsprechen. Die senkrechten Sprünge werden dabei vom charakteristischen 360 °-Übergang hervorgerufen. In der Tiefenvorgabe sind nochmals das Ab- und das Auftauchen erkennbar. Das Geschwindigkeitskommando ist aufgrund seines vergleichsweise geringen Wertebereiches in dieser Ansicht schwer zu unterscheiden.

- **Umwelt und Sensorik:**
  Die aktuellen Zustände lassen sich ebenfalls binär animiert visualisieren (siehe Abbildung 5.35).[90] Auf diese Weise kann das dynamische Ab- oder Zuschalten von Komponenten sowie das Auftreten des sweep-Signals anschaulich überprüft werden.

  Die Ansteuerung des Sonars durch SonarCtrl läßt sich aus Abbildung 5.36 entnehmen. Sie zeigt in Sonarkoordinaten die Ausrichtung des Sonars über die Zeit. In der Ansicht sind deutlich die einzelnen Linien erkennbar, die sich durch das Schwenken des Sonars ergeben. Die vertikale Verschiebung der Linien über die Zeit verdeutlicht, wie das Softwaremodul den Sichtbereich im Missionsverlauf (genauer: vor und während der Kurven) anpaßt und so dafür sorgt, daß das AUV die Kurven vor der Fahrt scannt.

  Abbildung 5.37 visualisiert den Abstand des AUV zum Boden. In der Beispielmission liegt dieser bei über 4000 Metern. Da dieser Abstand während der gesamten

[90] Die Darstellung erfolgt über das Plugin TVLLED, welches eine beliebige Anzahl von Daten in Form binärer Leuchtdioden visualisiert. Dabei werden die Zustände farblich codiert (Rot steht dabei für Null, Grün für Eins).

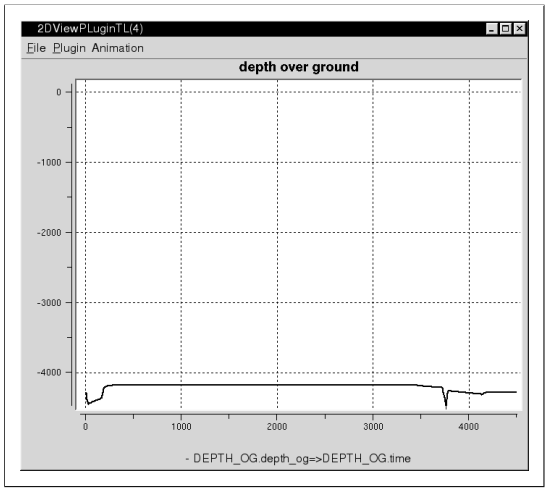

Abbildung 5.37: Bodenabstand

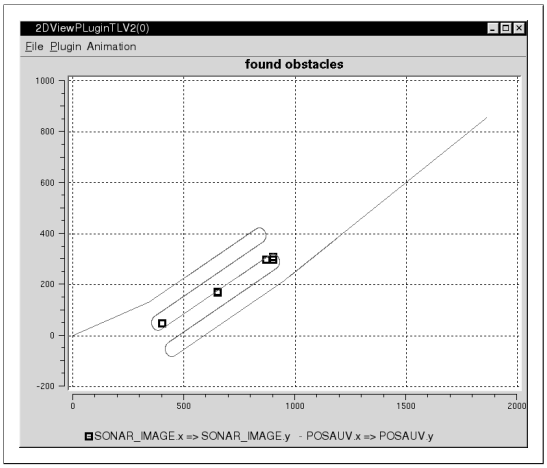

Abbildung 5.38: Erkannte Hindernisse

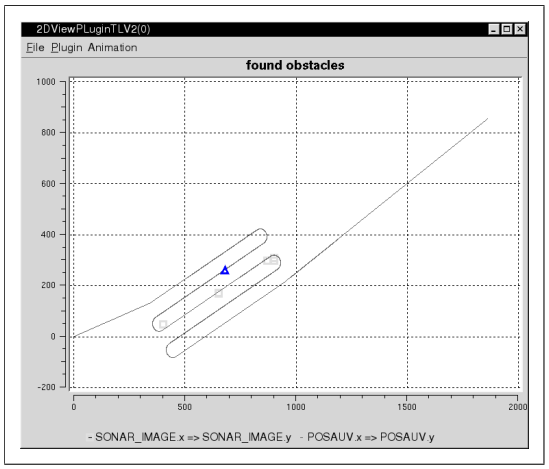

Abbildung 5.39: Erkannte Hindernisse ($t = 1060\,s$)

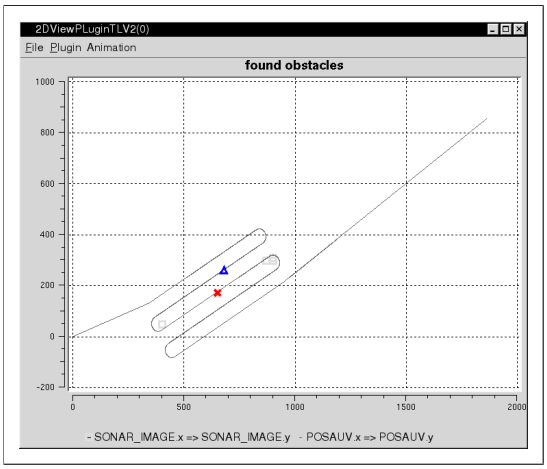

Abbildung 5.40: Erkannte Hindernisse ($t = 1061\,s$)

Mission den maximal meßbaren überschreitet, ist das Dolog nicht in der Lage, einen Abstand zu messen.

Abbildung 5.38 zeigt zusammen mit der Trajektorie die von der Software erkannten Hindernisse (□). Der Vergleich mit Abbildung 5.26 auf Seite 137 zeigt, daß alle vorhandenen Hindernisse erkannt wurden. Durch die Einbeziehung dieser Daten in die animierte Darstellung der Trajektorie[91] läßt sich zusätzlich der Zeitpunkt bestimmen, an dem das Fahrzeug die Objekte registriert hat. Die Abbildungen 5.39 und 5.40 verdeutlichen dies, indem sie jeweils einen Zeitpunkt kurz vor und kurz nach der Hinderniserkennung wiedergeben. Die Position des AUV wird dabei durch das Symbol △ dargestellt, während die Hindernispositionen mit □ (unerkannt) und × (erkannt) markiert werden.

[91] Vgl. Abbildung 5.29, Seite 142.

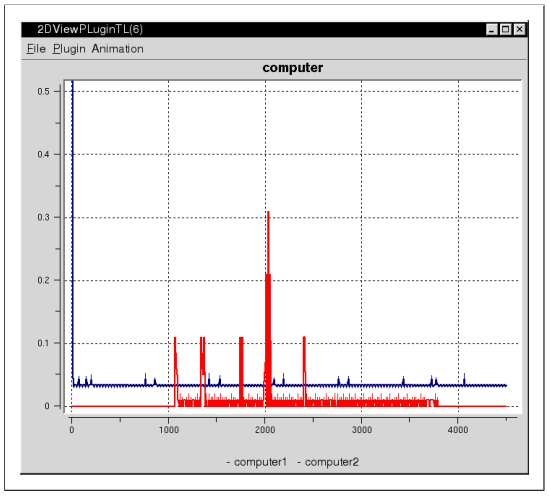

Abbildung 5.41: Prozessorauslastung

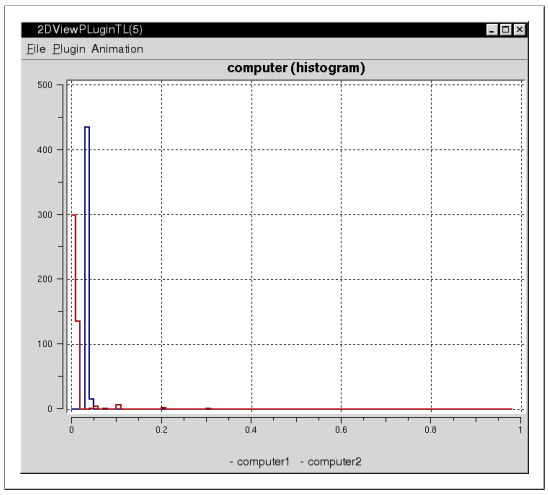

Abbildung 5.42: Histogramm der Prozessorauslastung

- **Computer:**
  In der Auslastung der Prozessoren zeigt sich, wie gut die Ressource Prozessorarbeit dimensioniert ist. Abbildung 5.41 stellt die Ergebnisse der Simulation für beide Prozessoren dar.[92] Computer 1 verzeichnet eine relativ konstante Grundlast, während Computer 2 eine sporadische Nutzung aufweist.

  Neben einem Werteverlauf über die Zeit bietet sich in diesem Fall auch die Betrachtung eines Histogramms der Auslastung an.[93] Abbildung 5.42 zeigt das hierbei von beiden Computern erzielte Ergebnis. Die Verteilungen sind dabei ein Ergebnis der gewählten Aufteilung der Softwaremodule auf die Computer. Die Grafik verdeutlicht, daß im vorliegenden Beispiel beide Rechner nur wenig ausgelastet sind.

- **Energie:**
  Neben der für eine Mission benötigten Energiemenge ist vor allem der Verlauf des Energieverbrauches von Interesse. Abbildung 5.43 bietet eine entsprechende Darstellung für die Brennstoffzelle. Deren Füllstand nimmt im Missionsverlauf kontinuierlich von 100 Prozent auf einen Wert von 99,4 Prozent ab. Die entsprechende Darstellung der Pufferbatterien (Abbildung 5.44) zeigt, daß diese durch die Regelung auf einem konstanten Niveau von 97 Prozent gehalten werden.

  Alternativ kann die Darstellung der Füllstände mittels einer an einem Thermometer angelehnten Anzeige verdeutlicht werden (Abbildung 5.45).[94] Dabei werden die Werte animiert als Säulen dargestellt.

[92] Eine Vorverarbeitung der Werte mittels eines Tiefpasses wäre über das Vorverarbeitungsplugin Filter möglich.
[93] Die Vorverarbeitung erfolgt über das Vorverarbeitungsplugin Histogramm.
[94] Diese Darstellung verwendet das Plugin Thermo.

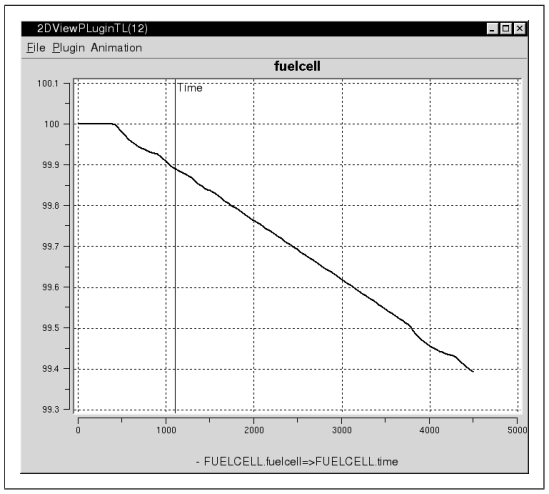

Abbildung 5.43: Verlauf des Füllstandes der Brennstoffzelle

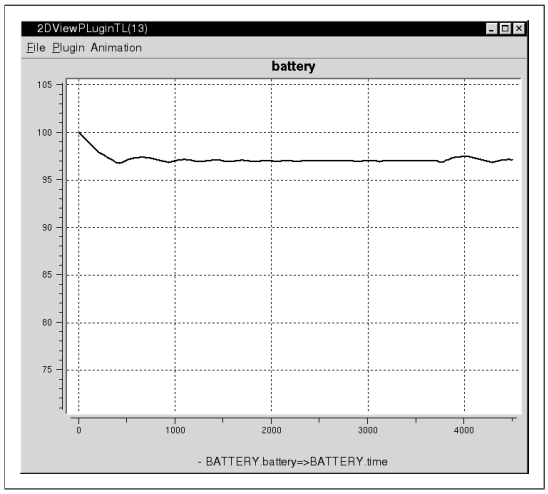

Abbildung 5.44: Verlauf des Füllstandes die Pufferbatterie

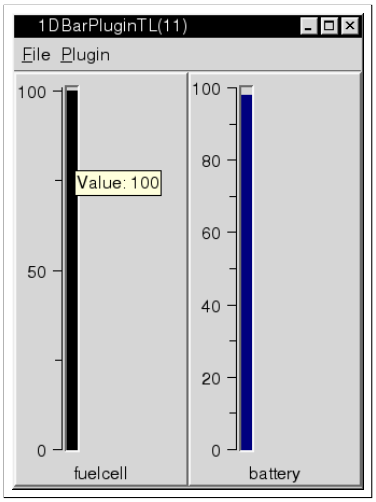

Abbildung 5.45: Füllstände Brennstoffzelle und Pufferbatterie

Die aus der Beispielmission abgeleiteten Ergebnisse lassen sich wie folgt zusammenfassen:

1. Der vorgegebene Missionsplan wurde korrekt abgefahren (Abtauchen, Mäander, Auftauchen).
2. Alle plazierten Objekte wurden vom Sonar und der zugehörigen Auswertung erkannt.
3. Die Computer sind wenig ausgelastet und verfügen in dieser Mission über große Reserven.
4. Die Mission verbraucht angesichts ihrer kurzen Dauer nur einen geringen Teil der verfügbaren Energie.

Wie diese Ergebnisse zeigen, konnte das Fahrzeug die durch die Beispielmission gestellten Anforderungen voll erfüllen.

Die bisherigen Ausführungen stellten die einzelnen zur Verfügung stehenden Darstellungen der Simulationsergebnisse vor. MLVisor kombiniert diese in seinen Auswertungsansichten. Dabei wird jeder Darstellung eine Größe und Position in einer oder mehrerer der fünf möglichen Ansichten zugewiesen. Da sich diese Ansichten speichern und wieder aufrufen lassen, kann für jede Mission eine speziell angepaßte Auswertungsoberfläche mit den benötigten Darstellungen zusammengestellt werden, die wiederverwendbar ist. Hierdurch wird der Umgang mit den unveränderlichen Missionen bei einem veränderlichen Gesamtsystemmodell erheblich erleichtert.

Die bereits vorgestellte Abbildung 5.28 gibt für die Beispielmission eine dieser Ansichten wider.[95] Sie dient der Kontrolle der gefahrenen Trajektorie und faßt hierzu verschiedene Einzeldarstellungen dieser Daten zusammen. Alle Darstellungen, die nicht zur aktuellen Ansicht gehören, werden von MLVisor minimiert angezeigt.

5.5.2 Ergebnisse für DeepC

Im folgenden Abschnitt wird ein Fazit der im Zusammenhang mit der Arbeit für das *DeepC*-Projekt erreichten Ergebnisse und Erkenntnisse gezogen.

- ***DeepC*-Gesamtsystemmodell**
  Als ein Ergebnis der Arbeiten, die der vorliegende Text dokumentiert, entstand das Gesamtsystemmodell des *DeepC*-Fahrzeugs, das die diversen Aspekte des Systems *DeepC* in sich vereint. Den theoretisch erarbeiteten Erkenntnissen entsprechend besteht es aus einem Systemmodell (Nachbildung des Fahrzeugs, Virtueller Prototyp) und der Umwelt. Im Zusammenhang mit der Modellierung beider Modelle lassen sich die folgenden Erfahrungen festhalten:

[95] siehe Seite 141

- Ein zentrales Problem der Modellierung besteht in der Abwägung des umzusetzenden Modellierungsniveaus. Insbesondere ist hierbei die Entscheidung zu treffen, wie abstrakt die Modellierung vorgenommen werden kann bzw. welche Funktionalität das Modell abbilden muß. Die Funktionalität muß so abstrakt wie möglich modelliert und über allgemeine Kennwerte (Systemparameter) parametrisiert werden.
- Für das *DeepC*-System ist insbesondere die Modellierung des Aspekts Energie zentral, da auf ihm entscheidende, für die Autonomie des Fahrzeugs relevante Teile der Funktionalität aufbauen.
- Die durchgeführte Modellierung verdeutlicht die entscheidende Bedeutung, die der Nutzung fertiger Bibliotheken für die Modellierung der einzelnen Bestandteile zukommt: Da diese im vorliegenden Anwendungsfall fehlten, mußten alle benötigten Komponenten neu modelliert werden. Dieser Umstand übte unmittelbaren Einfluß auf die für die Modellerstellung benötigte Zeit aus. Dadurch konnte im hier vorgestellten Modell die Spezifikation nicht vollständig umgesetzt werden.
- Das erstellte Gesamtsystemmodell kann als Ausgangspunkt für die Weiterentwicklung des *DeepC*-Fahrzeugs (Produktfamilie oder spezielle Anpassungen) dienen. Durch die Nutzung der erstellten Komponenten kann es darüber hinaus auch für die Modellierung andere Fahrzeuge herangezogen werden, was den in Abschnitt 3.2.2.2 erarbeiteten Vorstellungen entspricht.

- **Missionen**
  Die Missionen bilden im Gegensatz zu den Treibermodellen des System Level Designs ein implizites bzw. indirektes Belastungsmodell. Sie erzeugen eine Situation, auf die das System reagieren muß. Dabei ist für den Systementwurf sowohl das Ergebnis dieser Reaktion, als auch die zugehörige Ressourcenbelastung von Interesse. Diese Sichtweise läßt sich sehr gut mit dem im Rahmen dieser Arbeit wiederholt angeführten Bild eines virtuellen Prototyps verdeutlichen, der in einer virtuellen Umwelt getestet wird.

  Ein Ergebnis dieser Arbeit besteht darin, daß die konkrete Vorgehensweise zur Erstellung von Missionen vorgestellt werden konnte. Dabei hat sich gezeigt, daß diese Erstellung bis zu einem bestimmten Grad unabhängig vom Modell möglich ist. Die Eingabe der Missions- und auch der Systemparameter läßt sich damit so gestalten, daß auch Personen, die nicht im Detail mit der Modellierung vertraut sind, mit dem Modell arbeiten können[96].

- **Auswertung**
  Mit der vorgestellten Arbeit entstand eine Gesamtsystemsimulation im Sinne des Mission Level Designs bzw. des missionsbezogenen modellgestützten Entwurfs mobiler automatischer Systeme. Sie zeigt, daß die Zusammenführung einzelner Sichten

[96] siehe die Eingabe des Missionsplanes, Seite 137

sinnvoll ist und eine kombinierte Interpretation der Ergebnisse aus Gesamtsystemsicht ermöglicht.

Die vorgestellten Arbeiten im Rahmen des *DeepC*-Projekts stellen aus Sicht des Systementwurfs keinen prototypischen Anwendungsfall dar[97]: Sie waren zeitlich nicht - wie theoretisch vorgesehen - in die vordere Projektphase eingeordnet, sondern vollzogen sich parallel zur Komponentenentwicklung. Dabei mußten sowohl die benötigten Werkzeuge (d.h. das Framework), als auch die Bibliotheken erst entwickelt werden. Aus diesem Grund konnten sie trotz der erreichten Ergebnisse keinen direkten Einfluß auf das Systemdesign nehmen.

5.6 Zusammenfassung

Ein wesentlicher Bestandteil dieser Arbeit ist die Unterstützung des Entwurfs für das ***DeepC*-Projekt**, dessen Ziel die Entwicklung eines autonomen Unterwasserfahrzeugs für große Tiefen und eine lange Einsatzdauer ist. Hierzu fand sich ein interdisziplinäres Konsortium aus acht Firmen und Einrichtungen zusammen, um ihre spezialisierten Fähigkeiten und Erfahrungen im Rahmen dieses Projektes zusammenzuführen.

Eine der beteiligten Einrichtungen ist die Technische Universität Ilmenau, die die Fahrzeugführung (Ausweichen, Maschinelles Lernen und Missionsumplanung) und das Mission Level Design bearbeitet. Ziel des Teilprojekts Mission Level Design ist dabei die Unterstützung des Entwurfs durch Simulation auf Missionsebene. Dazu wurde, aufbauend auf den theoretischen Grundlagen und dem erstellten Framework, einerseits ein Gesamtsystemmodell des Fahrzeugs geschaffen, andererseits wurden Missionen definiert. Auf der Basis beider Bestandteile wurden simulative Untersuchungen zum Test des Systemdesigns durchgeführt.

Das **Gesamtsystemmodell** wurde mit MLDesigner erstellt und bildet einen virtuellen Prototyp des *DeepC*-Systems. Es repräsentiert das Systemdesign und modelliert deshalb alle wichtigen Komponenten des Fahrzeugs. Dabei umfaßt es die Aspekte Bewegung, Umwelt/Sensorik, Rechenleistung und Energie.

Für die Bewegung (Mobilität) existiert ein Modell für das geregelte Fahrzeugverhalten, das auf den Dynamikvorgaben des Entwurfs basiert. Daneben wurde die Umwelt und die zugehörige Sensorik nachgebildet, um innerhalb der Simulation eine autonome Handlungsweise zu ermöglichen. Die Umwelt besteht aus einem Boden- und einem Strömungsmodell sowie den Hindernissen. Für die Wahrnehmung der Umwelteinflüsse besitzt das Fahrzeugmodell ein Sonar und ein Dolog. Desweiteren ist für die Modellierung im Rahmen des missionsbezogenen modellgestützten Entwurfs die Abbildung der Ressourcen wichtig. Aus diesem Grund bezieht das Gesamtsystemmodell die Rechenleistung und die Energie in die Simulation ein. Beides sind Aspekte, die den Entwurf eines

[97] siehe Abschnitt 5.1.3, Seite 83

autonomen Unterwasserfahrzeugs wesentlich bestimmen: Das Letztgenannte erstellt ein Energienetz, in dem Verbraucher aus einer Quelle gespeist werden, was über Energiepakete geschieht, die im Netz geroutet werden. Das Modell für die Rechenleistung stellt ein Prozessormodell bereit, das eine lastabhängige Simulation der Abarbeitungszeit von Softwaremodulen vornimmt. Auf diese Weise wird nicht nur die Funktionalität der Software durch die Softwaremodule nachgebildet, sondern auch die Architektur, auf der sie läuft, sowie die Einflüsse, die dies auf die Abarbeitung hat.

Das vorliegende Modell geht mit seiner Modellierung über die in der Literatur beschriebenen Fahrzeugmodelle hinaus. Diese konzentrieren sich auf die Nachbildung der Bewegung (Hydrodynamische Simulationen) oder Bewegung und Umwelt/Sensorik (Verhaltensimulationen). Demgegenüber ist das hier entwickelte Modell um die Aspekte Rechenleistung und Energie erweitert und bezieht dadurch wichtige Ressourcen jedes autonomen Unterwasserfahrzeugs in die Betrachtung ein.

Das zukünftige Fahrzeugverhalten wird aus übergeordneter Sicht beschrieben. Dieser Ansatz der Gesamtsystemsimulation zeigt sich deutlich in der abstrakten Modellierung der erstellten Modelle. Beispielhaft hierfür läßt sich die Modellierung des Energiesystems anführen, aber auch die Rechenleistung, die Dynamik und die Hinderniserkennung wurden auf abstraktem Niveau realisiert. Dieses Vorgehen ist auch dem in frühen Projektphasen vorherrschendem Fehlen exakter Daten geschuldet.

Als zweiter Bestandteil der Gesamtsystemsimulation wurden mit Hilfe von MLEditor zusätzlich **Missionen** definiert und Systemparameter zugewiesen. Die Missionen wurden anschließend simuliert und ihre **Ergebnisse** mit Hilfe von MLVisor missionsspezifisch ausgewertet. Die dabei erzielten Simulationsexperimente erlauben Prognosen bezüglich der Leistungsfähigkeit des AUV-Systemdesigns.

6 Zusammenfassung

In der vorliegenden Arbeit wurde untersucht, wie sich die Idee des Mission Level Designs, die ursprünglich nur für Mobilkommunikationssysteme vorgestellt wurde, auf den Entwurf von mobilen automatischen Systemen anwenden läßt. Im Mittelpunkt der durchgeführten Betrachtungen stand die Simulation auf Missionsebene, die das Mission Level Design einführt.

Beim Mission Level Design handelt es sich um eine missionsbezogene modellgestützte Systementwurfsmethode für komplexe Systeme. Dabei wird deren Spezifikation in ein ausführbares Modell auf Systemebene überführt und simuliert. Dieses Modell wird als Gesamtsystemmodell bezeichnet, weil es sowohl funktionale, als auch architektonische und Umweltaspekte umfaßt. Das Mission Level Design nutzt in Erweiterung anderer Entwurfsmethoden zukünftige Einsatzszenarien, sogenannte Missionen, als Testbedingungen. Durch ihre Einbeziehung wird sichergestellt, daß der Entwurf die angestrebten Systemeigenschaften erreicht. Mit den Simulationen ist das Ziel verbunden, frühzeitig Aussagen zur erreichbaren Leistungsfähigkeit gewinnen und mögliche Probleme erkennen zu können.

In bezug auf mobile automatische Systeme ergeben sich, wie die vorliegende Arbeit zeigen konnte, einige Änderungen im Bereich der Modellierung. Das Gesamtsystemmodell vereint die Teilmodelle Systemmodell und Umwelt. Für die mobilen automatischen Systeme wurden die für die Modellierung zentralen Aspekte identifiziert. Da der Einsatz von Missionen zum Test des Designs einen Einfluß auf den Ablauf der Simulationen ausübt, wurde zudem die Anpassung des allgemeinen Konzepts notwendig.

Auf Basis dieser theoretischen Überlegungen wurde ein Framework für den missionsbezogenen modellgestützten Entwurf geschaffen, um die praktische Durchführung von Simulationen auf Missionsebene zu unterstützen. Den Kern des Frameworks bildet das kommerzielle Simulations- und Entwurfsprogramm MLDesigner, das durch die neuen Programme MLEditor und MLVisor um missionsspezifische Funktionalität erweitert wird. Mit dem neuentwickelten Framework eröffnet sich erstmals die Möglichkeit, auf einfache Art und Weise Simulationen auf Missionsebene durchzuführen.

Als erster konkreter Anwendungsfall des Frameworks diente das *DeepC*-Projekt, welches das Ziel verfolgt, ein neuartiges autonomes Unterwasserfahrzeug zu entwickeln. Hierfür wurden unter Nutzung des Frameworks Untersuchungen auf Missionsebene durchgeführt.

6.1 Zentrale Ergebnisse

Als Fazit der bisherigen Ausführungen sollen im Folgenden die in der Einleitung aufgeworfenen Fragen beantwortet und damit die zentralen Erkenntnisse dieser Arbeit kompakt zusammengefaßt werden.

- **Wie kann das Mission Level Design für mobile automatische Systeme angewendet werden?**
  Als Systementwurfsmethode muß das Mission Level Design - in dieser Arbeit auch als missionsbezogener modellgestützter Entwurf bezeichnet - vom Projektmanagement durchgeführt werden. Die Methode stützt sich wesentlich auf die Simulation der Spezifikation, damit sowohl die Performance, als auch gegebenenfalls auftretende Entwurfsprobleme möglichst frühzeitig analysiert werden können. Für die Simulation wird ein Gesamtsystemmodell verwendet, dessen Verhalten anhand von Missionen untersucht wird. Durch eine Anpassung des Modells an Implementierungsdetails können während der Phase der Komponentenentwicklung die Auswirkungen von Realisierungsentscheidungen im Gesamtsystemkontext betrachtet werden. Die Korrektheit der Modellprognosen zeigt sich in der Integrationsphase, während der das System zusammengefügt wird. Für etwaige Erweiterungen oder Weiterentwicklungen steht das Modell als Ausgangspunkt zur Verfügung.

  Im Rahmen der Simulationen sind die Modellierung und die Missionen von besonderer Bedeutung.

  Die Modellierung - also die Erstellung des benötigten Gesamtsystemmodells - wird durch die Art des zu modellierenden Systems beeinflußt. Prinzipiell umfaßt das Modell (nach Schorcht) für jeden der Aspekte Funktion, Architektur und Umwelt ein eigenes Modell. Für die mobilen automatischen Systeme zeigt sich, daß diese Aufteilung ungeeignet ist, weshalb eine neue Struktur vorgeschlagen wird. Diese untergliedert das Gesamtsystemmodell in ein Systemmodell und die Umwelt. Das Systemmodell nimmt die funktionalen und architektonischen Aspekte auf und entwickelt durch Nachbildung der Spezifikation ein Abbild des zu entwerfenden Systems, also einen virtuellen Prototyp. Die Umwelt stellt dem Systemmodell eine zielgerichtete, interaktive Nachbildung der realen Umwelt zur Seite. Für die mobilen automatischen Systeme läßt sich feststellen, daß die Aspekte Bewegung (Mobilität), Objekte und beeinflussende Prozesse, ihre Wahrnehmung (Sensorik) sowie die Ressourcen (Energie und Rechenleistung) von besonderer Bedeutung sind und deshalb modelliert werden müssen. Die Bewegung stellt aufgrund des mobilen Charakters der Systeme einen zentralen Aspekt dar. Die Notwendigkeit der Umweltwahrnehmung (Sensorik) ergibt sich aus der Struktur automatischer Systeme. Ressourcen stellen für den Entwurf eine wesentliche Beschränkungen dar, deren Auslastung deshalb durch Prognosen ermittelt werden sollte.

  Der zweite (und gegenüber anderen Methoden neue) Bestandteil des Mission Level Designs sind die Missionen. Wie diese Arbeit zeigt, können diese exemplarischen

Nutzungsbedingungen für die Simulation mit Hilfe der Parameter des Gesamtsystemmodells nachgebildet werden. Die hierfür verwendeten Missionsparameter sind nur eine Teilmenge der Parameter. Neben ihnen existieren System- und Entwurfsparameter sowie Konstanten. Die Missionsparameter setzen sich aus drei Bestandteilen, nämlich dem Missionsziel und den Konfigurationen von System und Umwelt, zusammen. Eine Mission als Nutzungsszenario beschreibt demnach eine Situation und die zugehörige Aufgabe des Systems.

Die Arbeit mit Missionen verlangt ein neues Konzept für die Durchführung der Simulation. Zusätzlich zu ihren herkömmlichen Phasen (Modellierung, Modellnutzung, Auswertung) wird eine Missionshandhabung benötigt, die den Umgang mit Missionen ermöglicht. Desweiteren muß auch die Auswertung missionsspezifisch erfolgen.

- **Wie lassen sich die Simulationen im Rahmen des missionsbezogenen modellgestützten Entwurfs mobiler automatischer Systeme mittels MLDesigner durchführen?**
  Eine Simulationsumgebung für den missionsbezogenen modellgestützten Entwurf muß das theoretisch erarbeitete, erweiterte Konzept unterstützen, das für die Arbeit mit Missionen entwickelt wurde. Um dies zu gewährleisten, wurde - nach einer Analyse der Fähigkeiten von MLDesigner - ein Framework realisiert, in dessen Zentrum MLDesigner steht. Dieses Framework ist charakterisiert durch die Trennung von Modell und Parametern auf der einen Seite und von Modellnutzung und Auswertung auf der anderen Seite. Damit werden die Missionshandhabung (Parametrisierung des Modells) und die missionsspezifische Auswertung aus MLDesigner mit dem Ziel einer besseren Unterstützung des Konzeptes der Missionen herausgelöst. Für beide Phasen wurde jeweils ein spezialisiertes Programm geschaffen (MLEditor für die Missionshandhabung und MLVisor für die missionsspezifische Auswertung).

  Zur Trennung von Modell und Parametrisierung werden die Parameter in MLDesigner lediglich entsprechend ihres Typs markiert. Die Zuweisung ihrer Werte und deren Verwaltung in einer Parameterdatenbank geschieht in MLEditor. Das Programm unterstützt die Erstellung und Verwaltung von Sets für die Missionsparameter (Missionen) ebenso, wie die Zuweisung von Werten für die Systemparameter. Damit sind in dieser Umsetzung des Konzepts die wesentlichen Parameter (alle außen den Konstanten) aus dem Modell herausgelöst.

  Die Trennung zwischen Modellnutzung und missionsspezifischer Auswertung wurde durch den Austausch der Simulationsergebnisdaten über eine Datenbank gestaltet. Durch sie kann einerseits die Datenaufbereitung vom Modell in die Auswertung verlagert werden, was dem Modell zugute kommt. Andererseits sind eine Archivierung der Ergebnisse und vergleichende Untersuchungen möglich.

  Mit dem im Rahmen dieser Arbeit entstandenen Framework lassen sich die Simulationen für den missionsbezogenen modellgestützten Entwurf einfach und benut-

zerfreundlich durchführen.

- **Wie gestaltet sich der praktische Einsatz des missionsbezogenen modellgestützten Entwurfs mobiler automatischer Systeme?**
  Diese Frage wurde anhand der Arbeit für das *DeepC*-Projekt untersucht. Ziel des Projekts ist die Entwicklung eines aktivautonomen und tieftauchenden Unterwasserfahrzeugs. Im Rahmen des Projektes bestand die Aufgabe darin, entsprechende Simulationen auf Missionsebene durchzuführen.

  Dazu wurde ein Gesamtsystemmodell des *DeepC*-Fahrzeugs erstellt, das die benötigten Aspekte aufweist. Diese umfassen die Bewegung, Sensorik und Umwelt sowie die Ressourcen Energie und Rechenzeit. Der virtuelle Prototyp kann sich in einer virtuellen Umwelt bewegen, die aus einem Seebodenmodell, einem Strömungsmodell und festen Objekten (Hindernissen) besteht. Das Fahrzeugmodell verfügt dabei über ein inertiales Navigationssystem, das die aktuelle Position liefert. Ein Sonar nimmt die Objekte und den Seeboden wahr, während ein Dolog den Abstand zum Boden und die Strömung mißt. Die Dynamik der Bewegung wird über ein Modell des geregelten Fahrzeugverhaltens modelliert. Daneben wird die Energie über entsprechende Komponenten nachgebildet. Die Verarbeitung der Sensorsignale und die Erzeugung der aktiven Autonomie des Fahrzeugverhaltens übernehmen die Nachbildungen der zugehörigen Softwaremodule. Deren Abarbeitung wird durch ein abstraktes Modell eines Rechners gesteuert.

  Bei der Modellierung der *DeepC*-Spezifikation konnte jedoch nicht auf fertige Bibliotheken zurückgegriffen werden. Vielmehr mußten neue Modelle entwickelt werden. Dieser Umstand und die Tatsache, daß das Framework ebenfalls erst geschaffen werden mußte, beeinflußten die Arbeit nachhaltig. Für das Modell wurden nur einige Beispielmissionen testweise generiert, deren Ergebnisse jedoch nicht in den realen Systementwurf einfließen konnten, weil dieser bereits beendet war. Daraus läßt sich das Fazit ziehen, daß für einen erfolgreichen Einsatz des Mission Level Designs im Zusammenhang mit der Simulation zwei wesentliche Punkte erfüllt sein müssen: Erstens müssen geeignete, spezialisierte Werkzeuge verfügbar sein. Zweitens verlangt diese Entwurfsmethode Bibliotheken als Voraussetzung für eine schnelle Modellierung, die nötig ist, um rechtzeitig relevante Aussagen hervorbringen zu können.

6.2 Ergebnisse der Arbeit

Die drei wichtigsten Beiträge, die diese Arbeit erbracht hat, lassen sich wie folgt zusammenfassen:

1. **Missionsbezogener modellgestützter Entwurf mobiler automatischer Systeme**
   Für den missionsbezogenen modellgestützten Entwurf wurde ein Konzept für die

Durchführung von Simulationen auf Missionsebene geschaffen. In diesem Zusammenhang konnte gezeigt werden, was Missionen sind, wie sie sich in den Ablauf der Simulationen integrieren lassen und welche Anforderungen sich durch sie ergeben.

Für die mobilen automatischen Systeme wurden in diesem Zusammenhang die Aspekte identifiziert, die für die Modellierung im Rahmen des missionsbezogenen modellgestützten Entwurfs berücksichtigt werden müssen. Dadurch konnte gezeigt werden, daß diese Simulationen sich von herkömmlichen Simulationen auf Komponentenebene darin unterscheiden, daß sie die zusätzlichen Aspekte und ihre veränderte Ausrichtung in der Modellierung, die durch die Gesamtsystemsicht bedingt ist, aufweisen.

2. **Framework**
   Im Rahmen der Arbeit entstand ein Framework für die Durchführung von Simulationen auf Missionsebene. Das Framework unterstützt das Konzept der Missionen im gesamten Ablauf der Simulationen und basiert auf den theoretischen Überlegungen zum missionsbezogenen modellgestützten Entwurf mobiler automatischer Systeme.

   Für das Framework wurden mit MLEditor und MLVisor zwei neue Programme geschaffen. Desweiteren wurden Mechanismen entwickelt, mittels derer sich ML-Designer in das Framework integriert.

3. **DeepC**
   In der Arbeit für das *DeepC*-Projekt wurde einerseits die praktische Durchführung von Simulationen auf Missionsebene beispielhaft vorgeführt. Andererseits beinhaltet das geschaffene Gesamtsystemmodell des *DeepC*-Fahrzeugs neuartige Aspekte im Bereich der Modellierung von autonomen Unterwasserfahrzeugen. So wurden erstmals die Ressourcen Energie und Rechenarbeit in ein Modell eines autonomen Unterwasserfahrzeugs aufgenommen. Mit dem Modell entstand zudem eine Bibliothek von Komponenten für die Simulation mobiler automatischer Systeme im allgemeinen und autonomer Unterwasserfahrzeuge im speziellen.

6.3 Nutzungsmöglichkeiten der Ergebnisse

Nachdem die durch diese Arbeit erzielten Ergebnisse zusammengefaßt wurden, soll nachfolgend der Betrachtungswinkel erweitert und diskutiert werden, wie die Ergebnisse über den hier vorgestellten Rahmen hinaus nutzbringend angewendet werden können. Hierbei sollen auch lohnenswerte Ansatzpunkte für weiterführende Forschungsarbeiten aufgezeigt werden.

- **Missionsbezogener modellgestützter Entwurf mobiler automatischer Systeme**

Wie die Ausführungen dieser Arbeit zeigten, stellt der missionsbezogene modellgestützte Entwurf einen geeigneten Ansatz für den Entwurf mobiler automatischer Systeme dar, weswegen seine Anwendung in anderen Projekten prinzipiell zu befürworten ist. Voraussetzung hierfür ist allerdings, daß die für die Modellierung des Gesamtsystemmodells notwendigen abstrakten Komponentenmodelle bereits vorliegen oder aber der für ihre Erstellung benötigte zeitliche Vorlauf zur Verfügung steht.

Ein sowohl aus wissenschaftlich-technischer, wie aus ökonomischer Sicht interessanter Ansatzpunkt für weitere Forschungsarbeiten sind Untersuchungen zur automatischen Optimierung des Entwurfs: Sie würden einen Entwurf ermöglichen, bei dem auf Basis einer automatischen Auswertung der Missionssimulationen eine zielgerichtete, automatische Anpassung von Systemparametern und/oder der Systemstruktur erfolgt. Dies stellt eine komplexe, herausfordernde Aufgabe dar, die eine weitere vorteilhafte Unterstützung des Entwurfsprozesses mit sich brächte.

- **Framework**
  Angesichts der mit dem hier vorgestellten Framework verbundenen Vorteile für die Missionshandhabung und die missionsspezifische Auswertung ist sein Einsatz in Projekten, die mobile automatische Systeme mit der Systementwurfsmethode des Mission Level Designs entwerfen, unbedingt empfehlenswert. Aber auch für anders gelagerte Simulationen könnte sich die von der Modellierung getrennte Modellparametrisierung als vorteilhaft erweisen. Für missionsbezogene Simulationen stellt diese Handhabung der Missionen und die dadurch erreichte Trennung der Missionen vom Modell eine wesentliche Verbesserung dar.

  Im Hinblick auf die weitere Verwendung des Frameworks ist der Ausbau der Fähigkeiten der beiden neuen Bestandteile MLEditor und MLVisor empfehlenswert. Hierbei ließen sich durch die Erstellung weiterer MLVisor-Plugins die Nutzungsmöglichkeiten und Einsatzfelder des Programms erheblich erweitern. Einen interessanten Erweiterungspunkt für MLEditor stellt die Möglichkeit einer kombinierten Eingabe für mehrere Parameter, zum Beispiel in Form einer virtuellen Welt, dar.

  Eine neue Nutzungsmöglichkeit des Frameworks ergibt sich durch den Einsatz als Systemkonfigurator für stark modulare Systeme im direkten Einsatz beim Kunden: Hierbei müßten in einem ersten Schritt mit dem Kunden Missionen erstellt werden, die die typischen Einsatzszenarien des vom ihm gewünschten Systems abbilden. In weiteren Schritten könnte dann interaktiv und ausgehend von einem Grundsystem mit Hilfe vorgefertigter Module ein System modelliert werden, das exakt an die spezifischen Kundenanforderungen angepaßt ist. Der mit diesem Vorgehen verbundene Vorteil liegt in der direkten Interpretierbarkeit der Ergebnisse, die die in das vorgestellte Konzept integrierte Simulation mit sich bringt. Die Aufbereitung der Simulationsergebnisse der Missionen ermöglicht dem Kunden eine direkte, anschauliche und für ihn nachvollziehbare Aussage über die Leistungsfähigkeit des Systems.

- **DeepC**
  Da das im Rahmen dieser Arbeit entwickelte Gesamtsystemmodell nach der Spezifikation des *DeepC*-Fahrzeugs erstellt wurde, ist es eine ideale Basis für die Entwicklung von möglichen Nachfolgefahrzeugen.

  Darüber hinaus kann das Gesamtsystemmodell im Ganzen oder in Teilen als Ausgangspunkt für die Modellierung und Simulation anderer autonomer Unterwasserfahrzeuge herangezogen werden. Die neuen Aspekte Energie und Rechenleistung stellen dabei eine Erweiterung dar, die - soweit Funktionalität auf ihnen aufbaut - auch in Komponentensimulationen Berücksichtigung finden muß.

Literaturverzeichnis

[Ada01] Adamski, Dirk: *Komponentenbasierte Simulation mechatronischer Systeme*. Fortschritts-Berichte VDI Reihe 10 Nr. 682. Düsseldorf : VDI Verlag, 2001 6, 14, 17

[AKT+97] Anisimov, N.A. ; Kovalenko, A.A. ; Tarasov, G.V. ; Inzartsev, A.V. ; Sherbatyuk, A. P.: A Graphical Environment for AUV Mission Programming and Verification. In: *Proceedings of the 10th International Symposium on Unmanned Unethered Submersible Technology, New Hampshire, USA*, 1997. – online: <http://www.fiztech-usa.net/anisimov/papers/Auv_fin.pdf>, S. 394–405 32

[AL03] Andrejević, Miona ; Liebezeit, Thomas: Modeling of the energy system for the mission level design of the DeepC AUV. In: *Proceedings of the twelfth international scientific and applied science conference ELECTRONICS, ET'2003, Sozopol, Bulgaria*, 2003 97

[ATL04] ATLAS ELEKTRONIK GmbH. *DeepC Projektseite*. online: <http://www.deepc-auv.de>. 2004 29, 30, 81

[Ben95] Bennett, B. S.: *Simulation fundamentals*. Ed. 1. Hertfordshire, UK : Prentice Hall International, 1995 14, 15

[BKZ92] Brutzman, Donald P. ; Kanayama, Yutaka ; Zyda, Michael J.: Integrated Simulation for Rapid Development of Autonomous Underwater Vehicles. In: *Proceedings of the IEEE Oceanic Engineering Society Conference AUV 92, Washington DC*, 1992. – online: <http://citeseer.nj.nec.com/brutzman92integrated.html>, S. 3–10 33, 35, 137

[Boe88] Boehm, B.: A spiral model of software development and enhancement. In: *IEEE Computer* 21 (1988), Nr. 5, S. 61–72 7

[Bor03] Borchers, Detlef: Verursacherbedingt verspätet – Das "fortschrittlichste Maut-System der Welt" und die Realität. In: *magazin für computer technik, c't* (2003), Nr. 22, S. 92–97 2

[BR00] van Beek, D. A. ; Rooda, J. E.: Languages And Applications In Hybrid Modelling And Simulation: Positioning Of Chi. In: *Control Engineering Practice* 8 (2000), Nr. 1, S. 81–91. – online: <citeseer.nj.nec.com/524442.html> 16

[Bru94] Brutzman, Donald P.: *A virtual world for an autonomous underwater vehicle*, Naval Postgraduate School, Monterey, California, Diss., December 1994 35

[Bue00] Buede, Dennis M.: *The Engineering Design of Systems*. Ed. 1. New York : John Wiley and Sons, Inc., 2000 6, 10, 47

[CHCC99] Carson, J. ; Hunter, J. ; Colley, M. J. ; Callaghan, Vic: Component based simulation of underwater vehicles using the high level architecture. In: *Proceedings of the 1999 Summer Computer Simulation Conference, Chicago*, 1999, S. 307–311 33

[Chr98] Christaller, Thomas: Autonome intelligente Systeme. In: *GMD-Spiegel, Thema: Perspektiven der Informationstechnik* (1998), Nr. 3/4, S. 11–15. – online: <http://www.gmd.de/de/GMD-Spiegel/GMD-Spiegel-3_4_98-html/Christaller.html> 27

[CN01] Chance, Thomas ; Northcutt, Jay: Deep Water AUV experiences. In: *Underwater Intervention 2001 Conference Proceedings*, 2001. – online: <http://www.thsoa.org/pdf/h01/12_3.pdf> 31

[CP99] Chappell, Steven G. ; Peng, Liang: Cooperative AUV Development Concept (CADCON): An Environment for High-Level Multiple AUV Simulation. In: *Proceedings of Eleventh International Symposium on Unmanned Unethered Submersible Technology, Durham NH.*, 1999, S. 112–120 32

[CRGU02] Carreras, Marc ; Ridao, Pere ; Garcia, Rafael ; Ursulovici, Zoran: Learning Reactive Robot Behaviors with a Neural-Q Learning Approach. In: *Proceedings of 2002 IEEE-TTTC International Conference on Automation, Quality and Testing, Robotics*, 2002. – online: <http://citeseer.nj.nec.com/carreras02learning.html> 32

[DBH00] Doucy, Olivier ; Brutzman, Donald ; Healey, Anthony: Near Surface Manoeuvring and Station - Keeping for an Autonomous underwater vehicle. In: *Proceedings of NATO Symposium, Applied Vehicle Technology Panel*, 2000. – online: <http://www.ec-nantes.fr/Fr/Liens/Sirehna/gallery/Auv/paper.pdf> 29, 34, 35

[Dor99] Dorobantu, Raul: Simulation des Verhaltens einer low-cost Strapdown IMU unter Laborbedingungen / Technische Universität München, Instituts für Astronomische und Physikalische Geodäsie, Forschungseinrichtung Satellitengeodäsie. 1999. – Forschungsbericht. online: <http://step.iapg.verm.tu-muenchen.de/reports/iapg_fesg_rpt_06.pdf> 109

[Duf96] Duffy, N. D.: The core simulator engine: integrating AUV subsystems with simulations. In: *IEE Colloquium on Autonomous Underwater Vehicles and Their Systems - Recent Developments* 102 (1996), Nr. 3, S. 1–4 33

[Eic02] Eichhorn, Mike: Cotrol Tasks in the Development of Underwater Vehicle. In: *Proceedings of the NRC IMD Seminars, Institute for Marine Dynamics, National Research Council Canada, St. John's, Newfoundland, Canada*, 2002 83, 105

[Eic03] Eichhorn, Maik: Methoden zur Führung von unbemannten Unterwasserfahrzeugen. In: *37. Regelungstechnischen Kolloquium, Boppard, Germany*, 2003 83

[Fle01] Fletcher, Barbara: Chemical Plume Mapping with the REMUS Autonomous Underwater Vehicle. In: *Proceedings of the AUSI Twelfth International Symposium on Unmanned Unethered Submersible Technology, Durham, NH*, 2001. – online: <http://www.spawar.navy.mil/robots/pubs/uust2001.pdf> 30, 137

[Flo04] Florida Atlantic University. *AUV Homepage Ocean Voyager II und Ocean Explorer*. online: <http://www.oe.fau.edu/AMS/auv.html>. 2004 28

[FMC96] Forsberg, Kelvin ; Mooz, Hal ; Cotterman, Howard: *Visualizing project management*. Ed. 1. New York, USA : John Wiley and Sons, Inc., 1996 8, 9

[FOPS95] Fryxell, Daniel ; Oliveira, Paulo ; Pascoal, Antonio ; Silvaestre, Carlos: An Intelligent Approach to the design and Analysis of navigation, guidance and control systems for autonomous underwater vehicles. In: *Proceedings of the Colloquium on Control and Guidance of Remotely Operated Vehicles* Bd. 124. London : IEE Colloquium Digest, 1995 29, 32

[Fos94] Fossen, Thor I.: *Guidance and Control of Ocean Vehicles*. Ed. 1. Chichester : Wiley, John and Sons, Inc., 1994 35, 102

[Fos02] Fossen, Thor I.: *Marine Control Systems*. Ed. 1. Trondheim, Norway : Tapir Trykkeri, 2002 35, 102

[Fre] Free Software Foundation. *Website*. online: <http://www.gnu.org/copyleft/gpl.html> 72

[Goh91] Goheen, K. R.: Modelling Methods for Underwater Robotic Vehicle Dynamics. In: *Journal of Robotic Systems* 8 (1991), Nr. 3, S. 295–317 35

[Gün97] Günther, Manfred: *Kontinuierliche und zeitdiskrete Regelungen*. Ed. 1. Stuttgart : B. G. Teubner, 1997 100, 104

[Hin03] Hinüber, Edgar. *New Approaches in High-Performance Navigation Solutions for AUVs*. online: <http://www.imar-navigation.de/download/underwater_imar.pdf>. 2003 109

[HN98] Hackett, G. ; Nahon, M.: A Portable Simulation Facility for the Design of Autonomous Underwater Vehicles. In: *IEEE Oceanic Engineering Society Newsletter* Bd. 33, 1998, S. 17–21 36

[IB] Initiative-Brennstoffzelle.de. *Glosar zur Thematik Brennstoffzelle*. online: <http://www.initiative-brennstoffzelle.de/ibz/live/ibzglossar/index.php3?suche_text=a> 86, 99

[Jak97] Jakobi, H.: Die VDI-Richtlinie 3633: Simulation im Überblick. In: *Fortschritte in der Simulationstechnik*. Braunschweig/Wiesbaden : Friedr. Vieweg und Sohn Verlagsgesellschaft mbH, 1997, S. 97–102 13

[JGW99] John G. Webster, editor-in-chief: *The Measurement, Instrumentation and Sensors Handbook*. Ed. 1. Boca Raton, FL and Heidelberg : CRC Press LLC and Springer-Verlag GmbH, 1999 109

[KAU96] Kuroda, Y. ; Aramaki, K. ; Ura, T.: AUV Test using Real/Virtual Synthetic World. In: on Autonomous Underwater Vehicle Technology, Symposium (Hrsg.): *Proceedings of the 1996 Symposium on Autonomous Underwater Vehicle Technology*. Piscataway, NJ : IEEE Service Center, 1996, S. 365–372 33, 35

[Keg90] Kegel, Gunther: *Erhöhung der Autonomie von Robotersystemen durch multisensorielle Informationen und Nutzung einer Wissensbasis*, Technische Hochschule Darmstadt, Diss., 1990 27

[Ker] Kern, Jürgen. *Die PEM - Brennstoffzelle, Homepage*. online: <http://www.hycar.de/pemfc.htm> 100

[KL96] Kalavade, Asawaree ; Lee, Edward A.: Complexity Management in System Level Design. In: *Journal of VLSI Signal Processing* 14 (1996), Nr. 2, S. 157–169. – online: <http://ptolemy.eecs.berkeley.edu/publications/papers/95/managing_complexity/managingComplexity.pdf> 10

[KO03] Karimanzira, Divas ; Otto, Peter: Control and Analysis of Nonlinear Systems using Neural Networks. In: *Proceedings of the IASTED International Conference on Neural Networks and Computational Intelligence, NCI 2003, Cancun, Mexico*, IASTED/ACTA Press, 2003, S. 78–83 83

[Kon04] Kongsberg Maritime. *AUV Homepage*. online: <http://www.km.kongsberg.com>. 2004 29

[Lar99] Larson, Thomas D.: Eurodocker, Simulation and Visualisation of Docking / Technical University of Denmark, Lyngby. 1999 (MAS3-CT97-0084). – Forschungsbericht. online: <http://www.iau.dtu.dk/AG/tdl/task52.pdf> 32

[Lee02] Lee, Edward A.: Embedded software. In: *Advances in Computers* (2002), Nr. 56. – online: <http://citeseer.nj.nec.com/lee02embedded.html> 17

[Lie02] Liebezeit, Thomas: Mission Level Design of Autonomous Underwater Vehicles. In: Nahavandi, Saeid (Hrsg.): *Proceedings of the First International ICSC Congress on Autonomous Intelligent Systems, ICAIS 2002, Geelong, Australia* NAISO, ICSC-NAISO Academic Press, 02 2002. – online: <http://www.deepc-auv.de/deepc/bibliothek/pdf/icais.pdf> 83

[Lie04] Liebezeit, Thomas: *Beschreibung des Frameworks für den Entwurf auf Missionsebene*. Ed. 1. : Technische Universität Ilmenau, 2004. – unveröffentlicht 132, 134, 141

[Lin] Linux. *Website*. online: <http://www.linux.org> 71

[LPR93] Lütkepohl, S. ; Page, B. ; Rudloff, A.: Datenbankkonzepte für die Simulation: Eine prototypische Datenbankkomponente für das Simulationspacket DESMO in Modula-2. In: *Fortschritte in der Simulationstechnik*. Braunschweig/Wiesbaden : Friedr. Vieweg und Sohn Verlagsgesellschaft mbH, 1993, S. 199–202 19

[LZ04] Liebezeit, Thomas ; Zerbe, Volker: A Validation Environment for Mission Level Design of Autonomous Underwater Vehicles. In: *Proceedings of the 49. Internationales Wissenschaftliches Kolloguium, Ilmenau, Germany*, 2004 83

[LZL03] Liebezeit, Thomas ; Zerbe, Volker ; Löffler, Tino: A Framework for Mission Level Design. In: *Proceedings of the Twelfth IASTED International Conference on Applied Simulation and Modelling, ASM 2003, Marbella, Spain*, 2003. – online: <http://sahara.theoinf.tu-ilmenau.de/research/publications/documents/2003/asm2003.pdf> 71

[Mar03] Maridan ApS. *Homepage*. online: <http://www.maridan.dk>. 2003 29

[Mat03] MathWorks, Inc. *Produktübersicht, Webpage*. online: <http://www.mathworks.com/products/product_descriptions.shtml>. 2003 20

[MF+03] Mangeruca, Leonardo ; Ferrari, Alberto [u. a.]: System Level Design of Embedded Controllers: Knock Detection, a Case Study in Automotive Domain. In: *Proceedings of the Design Automation and Test in Europe Conference, Designers Forum*, 2003. – online: <http://www-cad.eecs.berkeley.edu/Respep/Research/asves/paper2003/Mangeruca_date03.pdf>, S. 232–237 10

[MLD03a] MLDesigner Technologies, Inc.: *Dokumentation*. V2.4. Palo Alto, CA: , 2003. – local: <file:///usr/local/mld2/doc/manual.pdf> 23, 72

[MLD03b] MLDesigner Technologies, Inc. *MLDesigner Product Homepage*. online: <http://www.mldesigner.com>. 2003 20, 22

[MS+02] McAllister, C. ; Simpson, T. [u. a.]: Multidisciplinary design optimization testbed based on autonomous underwater vehicle design. In: *Proceedings of the 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Atlanta, Georgia*, 2002 34

[MyS] MySQL AB. *Website*. online: <http://www.mysql.com/> 72

[Möl92] Möller, Dietmar: *Modellbildung, Simulation und Identifikation dynamischer Systeme*. Ed. 1. Berlin, Heidelberg : Springer, 1992 15

[Nah96] Nahon, M.: A Simplified Dynamics Model for Autonomous Underwater Vehicles. In: on Autonomous Underwater Vehicle Technology, Symposium (Hrsg.): *Proceedings of the 1996 Symposium on Autonomous Underwater Vehicle Technology*. Piscataway, NJ : IEEE Service Center, 1996, S. 373–379

[Nat04] National Geophysical Data Center. *Global Relief Data*. online: <http://www.ngdc.noaa.gov/mgg/bathymetry/global/global.html>. 2004 126

[Pfü03a] Pfützenreuter, Thorsten: Advanced Mission Management for Long-range Autonomous Underwater Vehicles. In: *Proceedings of Oceans 2003 Marine Technology and Ocean Science Conference, OCEANS'03, San Diego, CA, USA*, 2003 83, 95, 138

[Pfü03b] Pfützenreuter, Thorsten: An Onboard Mission Replaning System for Autonomous Underwater Vehicles. In: *Proceedings of 11th IEEE Mediterranean Conference on Control and Automation, MED'03, Rhodos, Greece*, 2003 83

[PS99] Plantin, J. ; Stoy, E.: Aspects on System Level Design. In: *Proceedings of the 7th International Workshop on Hardware/Software Codesign, CODES'99, Rome, Italy*, 1999. – online: <http://www.sigda.org/Archives/ProceedingArchives/Codes/Codes99/papers/1999/codes99/pdffiles/g_1.pdf>, S. 209ff. 10

[PSX98] Pang, Yongjie ; Shang, You ; Xu, Yuru: Software design techniques for the man-machine interface to autonomous underwater vehicles. In: *Proceedings of the 1998 International Symposium on Underwater Technology, Piscataway, NJ, USA*, 1998 35

[RBC01] Ridao, Pere ; Batlle, Joan ; Carreras, Marc: An Underwater Autonomous Agent. From simulation to experimentation. In: *Proceedings of the 9th Mediterranean Conference on Control and Automation MED'01, Croatia*, 2001. – online: <http://eia.udg.es/~pere/papers/med01.pdf> 35, 137

[Riz] Rizzoli, Andrea. *A Collection of Modelling and Simulation Resources on the Internet, Website*. online: <http://www.idsia.ch/~andrea/simtools.html> 20

[RRGL99] Roeckel, Michael W. ; Rivoir, Robert H. ; Gibson, Ronald E. ; Linder, Stephen P.: Simulation Environment for the design and test of an intelligent controller for autonomous underwater vehicles. In: *Proceedings of the 31st conference on Winter simulation: Simulation - a bridge to the future, Phoenix, Arizona, United States* Bd. 2, 1999, S. 1088–1093 33

[RT98] Rendas, M. J. ; Turcci, Emmanuel: AUV Mission Preparation as a Multicriteria Optimization Problem Using Statistical Performance Prediction. In: *Proceedings of OCEANS'98, Nice, France* Bd. 3, 1998. – online: <http://citeseer.nj.nec.com/article/rendas98auv.html>, S. 1749–1753 35, 137

[RW98] Ritzschke, M. ; Wiedemann, T.: Gewinnung und Aufbereitung von Simulationsdaten zu Vergleichszwecken. In: Engeli, M. (Hrsg.) ; Hrdliczka, V. (Hrsg.): *Fortschritte in der Simulationstechnik* Arbeitsgemeinschaft Simulation (ASIM), Hochschulverlag AG an der ETH Zürich, 1998, S. 283–90 19

[RW00] Rathmann, Markus ; Wieskotten, Christa: *Anwendungsentwicklung für Linux: Programmieren mit Open-Source-Software*. Ed. 1. München : Markt&Technik Verlag, 2000 71

[RXY+] Rung, T. ; Xue, L. ; Yan, J. ; Schatz, M. ; Thiele, F. *Nummerische Methoden der Thermo- und Fluiddynamik: Transfinite Interpolation bei 3D-Gittern, Website*. online: <http://hodgson.pi.tu-berlin.de/Vorlesungen/tfd_skript/node94.html> 130

[Sch90] Schwarze, Gunter: *Digitale Simulation*. Ed. 1. Berlin : Akademie Verlag, 1990 15

[Sch00] Schorcht, Gunar: *Entwurf integrierter Mobilkommunikationssysteme auf Missionsebene*. Ed. 1. Berlin : Logos Verlag, 2000 2, 11, 12, 40, 58

[Sch02] Scholz, Roland: *Parallele und Verteilte Simulation bei der Steuerung komplexer Produktionssysteme*, Technische Universität Ilmenau, Dissertation, 2002 14

[SMP99] Silva, Jorge E. ; Martins, Alfredo ; Pereira, Fernando L.: A Reconfigurable Mission Control System for Underwater Vehicles. In: *Proceedings of the MTS/IEEE Oceans'99 Conference, Seattle, USA*, 1999. – online: <http://dceg.fe.up.pt/~lsts/Arca-Prize-LSTS/files/oceans99recmission.pdf> 32

[Sou03] Southampton Oceanography Centre. *Autosub Homepage*. online: <http://www.soc.soton.ac.uk/OED/index.php?page=as>. 2003 29

[SS00] Song, Feijun ; Smith, Samuel M.: Design of Sliding Mode Fuzzy Controllers for an Autonomous Underwater Vehicle without System Model. In: *Proceedings of the MTS/IEEE OCEANS'2000, Providence, Rhode Island*, 2000. – online: <http://www.oe.fau.edu/~fsong/ocean001.pdf>, S. 835–840 32

[ST92] Stambaugh, J. S. ; Thibauld, R. B.: Navigation requirements for autonomous underwater vehicles. In: *Navigation: journal of the Institute of Navigation* 39 (1992), Nr. 1, S. 79–92 109

[SUF00] Sayyaadi, Hassan ; Ura, Tamaki ; Furjii, Teruo: Collision Avoidance Controller for AUV Systems Using Stochastic Real Value Reinforcement Learning. In: *CDROM of The 39th SICE Annual Conference, Iizuka, Japan*, 2000 29, 32, 35

[Tak01] Takala, Jarmo. *Co-Simulation and Design.* online: <http://www.cs.tut.fi/kurssit/83950/cosimulation.pdf>. 2001 11, 110

[Tec04] Technische Universität Ilmenau, Fachgebiet Systemanalyse. *Modellierung, Regelung und Navigation von ferngesteuerten Unterwasserfahrzeugen, Website.* online: <http://www.systemtechnik.tu-ilmenau.de/~fg_sa/projektrov.html>. 2004 30

[Tho96] Thomas, Carsten: *Ein objektorientiertes Konzept zur Modellierung und Simulation komplexer Systeme.* Fortschritts-Berichte VDI Reihe 20 Nr. 208. Düsseldorf : VDI Verlag, 1996 15

[Tri97] Trieb, Rainer: *Autonome mobile Systeme; Simulation als Werkzeug zum Entwurf und zur Optimierung.* Ed. 1. Aachen : Shaker Verlag, 1997 16

[Tro] Trolltech. *Qt Product Homepage.* online: <http://www.trolltech.com> 71

[Uni] Unidata. *NetCDF Dokumentation.* online: <http://www.unidata.ucar.edu/factsheets/netcdf.html> 125

[Val04] Valavanis, Kimon. *Autonomous Underwater Vehicles Resources, Website.* online: <http://www.cacs.louisiana.edu/~kimon/AUV/>. 2004 28

[VDI96] Verein Deutscher Ingenieure: *VDI Richtlinie 3633: Simulation von Logistik-, Materialfluß-, und Produktionssystemen, Begriffsdefinitionen.* Nov. 1996 13, 14, 15, 17, 19

[Wai96] Waite, A. D.: *Sonar for practising engineers.* Ed. 1. Stockport : Ferranti Thomson Sonar Systems, 1996 105

[Wei96] Weik, Martin H.: *Communications standard dictionary.* Ed. 3. New York, NY : Chapman & Hall, 1996 105

[Wer99] Wernli, Robert L.: AUV's–The Maturity of the Technology. In: *Proceedings of MTS/IEEE OCEANS'99, Seattle, WA*, 1999. – online: <http://www.spawar.navy.mil/robots/pubs/oceans99.pdf> 28, 31

[Wer00] Wernli, Robert L.: AUV Commercialization - Who's Leading the Pack. In: *Proceedings of MTS/IEEE OCEANS 2000, Providence, RI*, 2000. – online: <http://www.spawar.navy.mil/robots/pubs/oceans2000.pdf> 29

[Wer03] Wernstedt, Jürgen. *F/E Schwerpunkt Mobile Systeme am Institut für Automatisierungs- und Systemtechnik im Fachgebiet Systemanalyse.* unveröffentlicht. 2003 25, 28

[Wid] WIDEMAN, Max. *Software development and linearity.* online: <http://www.maxwideman.com/papers/linearity/linearity1.pdf, http://www.maxwideman.com/papers/linearity/linearity2.pdf> 8, 10

[Woo99] WOOLDRIDGE, M.: Intelligent Agents. In: WEISS, G. (Hrsg.): *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press, 1999 27

[Zim04] ZIMMER, Uwe R. *Autonomous Underwater Vehicles: A collection website.* online: <http://www.transit-port.net/Lists/AUVs.html>. 2004 28

Alle Links wurden am 06.08.2004 auf ihre Korrektheit überprüft.

Stichwortverzeichnis

V

W

Erklärung

Ich versichere, daß ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Weitere Personen waren an der inhaltlich-materiellen Erstellung der vorliegenden Arbeit nicht beteiligt. Insbesondere habe ich hierfür nicht die entgeltliche Hilfe von Vermittlungs- oder Beratungsdiensten (Promotionsberater oder anderer Personen) in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Die Arbeit wurde bisher werder im In- noch im Ausland in gleicher oder ähnlicher Form einer Prüfungsbehörde vorgelegt.

Ich bin darauf hingewiesen worden, daß die Unrichtigkeit der vorstehenden Erklärung als Täuschungsversuch angesehen wird und den erfolglosen Abbruch des Promotionsverfahrens zur Folge hat.

(Ort, Datum) (Thomas Liebezeit)

Thesen

1. Mobile automatische Systeme sind biologische oder technische Systeme, die sich selbständig bewegen und Aufgaben erfüllen können.
2. Das Mission Level Design (MLD) stellt ein missionsbezogenes modellgestütztes Entwurfsverfahren für komplexe Systeme dar. Es nutzt Simulationen eines Gesamtsystemmodells, das mit Missionen konfiguriert ist, um die Spezifikation zu validieren und Leistungsdaten zu prognostizieren.
3. Der missionsbezogene modellgestützte Entwurf mobiler automatischer Systeme weist einige Besonderheiten auf:
   a) Die Struktur des Gesamtsystemmodells muß im Vergleich zum Mission Level Design angepaßt werden. Sie besteht nun aus dem Systemmodell und der Umwelt.
   b) Das Systemmodell stellt einen virtuellen Prototypen dar.
   c) Die Modellierung des Gesamtsystemmodells umfaßt die folgenden Aspekte: Bewegung, Objekte/Prozesse, ihre Wahrnehmung und die wichtigen Ressourcen Rechenleistung und Energie.
   d) Missionen setzen sich aus dem Missionsziel und den Konfigurationen von System und Umwelt zusammen.
   e) Das Gesamtsystemmodell besitzt Konstanten, Missions-, System- und Entwurfsparameter. Missionen werden durch die Missionsparameter des Gesamtsystemmodells beschrieben.
   f) Für die Durchführung der Simulation wird ein erweitertes Konzept benötigt. Dieses beinhaltet neben der Modellierung, der Modellnutzung und der nun missionsspezifischen Auswertung jetzt auch eine Missionshandhabung.
4. Der missionsbezogene modellgestützte Entwurf ist in verschiedenen Phasen des Entwicklungsprozesses hilfreich. Während der Projektierung hilft er direkt bei der Erstellung der Spezifikation. Danach in der Komponentenentwicklung unterstützt er die Kontrolle der Zielerreichung. Wenn das Gesamtsystemmodell in der Integrationsphase gezeigt hat, daß alles fehlerfrei war, kann das Modell in der Weiterentwicklung als Ausgangspunkt wiederverwendet werden.

5. Die organisatorische Durchführung der Simulation auf Missionsebene und die Modellierung des Gesamtsystemmodells soll ein MLD-Team übernehmen, das bei Projektmanagement angeordnet ist.

6. Für die Durchführung der Simulationen des missionsbezogenen modellgestützten Entwurf mobiler automatischer Systeme, entsprechend These (3f), unter Nutzung des Simulations- und Entwurfsprogramms MLDesigner sind an diesem Erweiterungen nötig. Diese wurden in einem Framework umgesetzt, das aus den drei Programmen MLDesigner (Modellierung, Modellnutzung), MLEditor (Missionshandhabung) und MLVisor (missionsspezifische Auswertung) besteht.

7. Die praktische Durchführung von Simulationen auf Missionsebene und ihre Auswertung konnte anhand der Arbeit für das *DeepC*-Projekt vorgeführt werden. In diesem Zusammenhang wurde auch das Gesamtsystemmodell für das AUV *DeepC* detailiert vorgestellt.

(Ort, Datum)