Vernetzung im ATEO Projekt aus inhaltlicher und technischer Sicht

Diplomarbeit

Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftlichen Fakultät II
Institut für Informatik

eingereicht von Nicolas Niestroj

Gutachter: Prof. Dr. sc. nat. Klaus Bothe
Prof. Dr. sc. nat. Hartmut Wandke

Betreuer: Dipl.-Psych. Saskia Kain
Dipl.-Psych. Jens Nachtwei

Berlin, den 09. Juli 2009

Nicolas Niestroj
nicolas.niestroj@gmail.com

ZUSAMMENFASSUNG

Die vorliegende Diplomarbeit ist im ATEO Projekt (Arbeitsteilung Entwickler Operateur) des Graduiertenkollegs prometei (Prospektive Gestaltung von Mensch-Technik-Interaktion) an der Technischen Universität Berlin angesiedelt. In diesem Projekt soll ein komplexer, dynamischer, technischer Prozess, mittels der sogenannten Socially Augmented Microworld (SAM) also einer belebten Mikrowelt, durch einen Operateur oder durch von Entwicklern konzipierte Automatiken geregelt werden. SAM ist eine Trackingaufgabe, die von zwei Versuchspersonen kooperativ absolviert werden soll. Diese menschliche Komponente (Mikroweltbewohner) in dieser Mikrowelt bringt die nötige Komplexität in Form von unvorhersagbarem aber trotzdem nachvollziehbarem Verhalten. Dem Operateur wird für seine Aufgabe ein Operateursarbeitsplatz zur Verfügung gestellt, welcher ihm Informationen wie aktuelles Lenkverhalten, Video der Mikroweltbewohner und eine Streckenvorschau sowie regelnde Eingriffsmöglichkeiten in Form von Hinweisen oder Eingriffen in die Steuerung bietet. Die Automatiken erhalten die Daten direkt aus dem System und reagieren auf Basis von Berechnungen aus diesen Daten in Form von Hinweisen und Eingriffen in die Steuerung.

Im Rahmen dieser Diplomarbeit wurde SAM mehrfach angepasst an die Anforderungen des Projektes, erweitert und der Quellcode mit Methoden des Refactorings verbessert. In der ersten Stufe wurde die Komplexität des Trackings von SAM durch Hinzufügen von Hindernissen, Gabelungen und der Manipulation des Steuerungsinputs der Mikroweltbewohner erhöht. Zusätzlich wurde eine Geschwindigkeitsanzeige in das Fahrobjekt integriert und die Instruktionsfenster sowie das Konfigurationsmenü für den Versuch neu implementiert. In einer zweiten Stufe wurde die Simulation auf Basis von empirischen Daten aus ingenieurpsychologischen Experimenten angepasst und konfiguriert.

Neben der Hauptversion von SAM wurde noch eine Version für das Solotracking konfiguriert, bei welchem von einem einzelnen Mikroweltbewohner die Trackingaufgabe gelöst werden soll, in der die meisten Erweiterungen von SAM zusammenspielen. Alle Anpassungen werden inhaltlich konzeptionell und softwaretechnisch beschrieben.

Im zweiten Teil der Arbeit wurden SAM und der Operateursarbeitsplatz vernetzt. Hierfür wurden SOAP als XML-basierte Kommunikation sowie TCP und UDP Sockets eingesetzt. Die Entscheidungsfindung für die Kommunikationsmethoden bei der Vernetzung mit dem Operateursarbeitsplatz wird genauso diskutiert wie die Entscheidung die in einer weiteren Diplomarbeit entwickelten Automatiken nicht mit SAM zu vernetzen. Die Grundlagen der Netzwerktechnik werden für ein besseres Verständnis dieser Diskussionen gesondert vermittelt. SAM wurde in einem letzten Schritt erweitert, um die Funktionen des Operateursarbeitsplatzes in SAM entsprechend ausführen zu können.

ABSTRACT

This thesis is located within the ATEO project (Arbeitsteilung Entwickler Operateur - Division of Labor between Developers and Operators) of the research training group prometei (Prospektive Gestaltung von Mensch-Technik-Interaktion)located at the Technical University in Berlin. In this context a complex, dynamic and technical process namely the Socially Augmented Microworld (SAM) or lively microworld should be regulated by an operator or automatics designed by developers. This human component (microworld inhabitant) in this microworld brings the needed complexity in terms of unpredictable yet explainable behaviour. The Operator is aided by an operators workstation which provides him with information such as current steering behaviour, surveilance footage of the microworld inhabitants or a preview of the track as well as the possibility to give hints or influence the steerage. The automatics get their data directly from the system and react based on calculations by showing Hints and intervening into the steering..

During the development SAM was edited and extended several times and the source code is improved with methods of refactoring. In the first stage, the complexity of the tracking of SAM was increased by adding obstacles, forks and manipulation of the control inputs of the microworld inhabitants. In addition, a display of speed was integrated into the driving object and the instruction windows as well as the setup menu were re-implemented. A second stage of the simulation is an adjustment and configuration based on empirical data from experiments in human factors.

Besides the main version of SAM a solo version of the tracking was configured in which the tracking task is solved by a single microworld inhabitant with most of the extensions of SAM integrated. All adjustments are described from the conceptional and technical point of view.

In a second step SAM and the operators workstation are connected over a network. XML based SOAP and sockets with TCP and UDP are used for the communication. The decision for the communication methods for networking the operators workstation are discussed, so is the decision for not networking SAM including the automatics. The basics of computer networking are explained seperately for a better understanding of these discussions. SAM was extended to work with the functional range of the operators workstation.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

DEFINITIONEN

FORMELZEICHEN UND ABKÜRZUNGEN

ATEO	Das Forschungsprojekt „Arbeitsteilung Entwickler Operateur“
AOF	Automated Operator Functions / Automatiken
HTA	Hierarchical Task Analysis / Hierarchische Aufgabenanalyse
HTTP	Hypertext Transfer Protocol
OA	Operateursarbeitsplatz
prometei	das Graduiertenkolleg Prospektive Gestaltung von Mensch-Technik-Interaktion
RFC	Request For Comment
RMI	Remote Method Invocation / entfernter Methodenaufruf
RPC	Remote Procedure Call / entfernter Funktionsruf
rST	remote Smalltalk
SAM	Socially Augmented Microworld
SOA	Service Oriented Architecture / Service Orientierte Architektur
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

1 EINLEITUNG

1.1 MOTIVATION

Im ATEO Projekt (Arbeitsteilung Entwickler Operateur) des Graduiertenkollegs prometei (Prospektive Gestaltung von Mensch-Technik-Interaktion) des Zentrums Mensch-Maschine-Systeme der Technischen Universität Berlin soll die Funktionsteilung von Mensch und Maschine erforscht werden. Der Schwerpunkt liegt dabei auf der asynchronen Arbeitsteilung zwischen einem Operateur, der ein System nutzen soll, und dem Entwickler, der ein solches System plant, indem er Prozessverläufe und Störungen antizipiert. Es sollen durch verschiedene empirische Experimente Erkenntnisse gewonnen werden, in wie weit eine Funktionsteilung zwischen Mensch und Maschine unter dem Aspekt der Arbeitsteilung von Entwickler und Operateur verbessert werden kann (Wandke und Nachtwei, 2008).

In der ersten Phase des ATEO Projektes wurde die Socially Augmented Microworld (SAM) theoretisch und praktisch entwickelt. Mit Hilfe einer Mikrowelt können komplexe dynamische Prozesse als Computersimulation im Labor untersucht werden. Um eine nichtdeterministische Komplexität in den Prozess einzuführen, werden zwei Versuchspersonen als Mikroweltbewohner genutzt. Damit kann man erreichen, dass der Prozess dieser Mikrowelt nicht vollständig berechenbar oder antizipierbar wird. Im deterministischen Fall könnten die Entwickler perfekte Automatiken entwickeln oder der Operateur erst nach einiger Zeit den Prozess vollständig erfassen und ebenfalls perfekt regeln. Trotzdem hätte er durch die zeitliche Verzögerung einen Nachteil den Automatiken gegenüber. Einen echten stochastischen Prozess zu gestalten, stellt das andere Extremum dar, denn nun ist es unmöglich diesen Prozess, welcher sich nicht vorhersagen oder nachträglich erklären lässt, zu regeln. Automatiken könnten nur für einzelne Ereignisse entwickelt werden, die aber gar nicht auftreten müssen, da sie mit einer bestimmten Wahrscheinlichkeit eintreten. Ein Operateur kann zwar besser auf das Verhalten reagieren, aber nie präventiv Maßnahmen ergreifen und ein mögliches Ereignis antizipieren. Der menschliche Faktor in dieser Mikrowelt bringt die gewünschte Lösung. Das Verhalten des Menschen ist nichtdeterministisch aber trotzdem erklärbar. Es ist also möglich vorherzusehen, wie sich der Mikroweltbewohner verhalten wird, er muss es aber nicht immer tun. Seine Entscheidung sich anders zu verhalten ist aber im Nachhinein nachvollziehbar. Die gewünschte Komplexität ist damit erreicht. Gross und Nachtwei (2006) bieten zum theoretischen Hintergrund von SAM weitere Informationen.

Um Konfliktpotential zwischen den Mikroweltbewohnern hervorzurufen, werden sie gegensätzlich instruiert, während sie selbst von einer gemeinsamen Instruktion ausgehen. So werden sie dazu angewiesen das Tracking möglichst genau oder schnell zu absolvieren. Dadurch kann eine erhöhte Komplexität von SAM erreicht werden. Ebenso sollen verschiedene Erweiterungen Entscheidungssituationen schaffen.

Im Rahmen der zweiten Phase von ATEO sollen ein Operateursarbeitsplatz für die Regelung von SAM entwickelt und optimiert werden. Die Funktionalität wird dabei auf Seiten von SAM implementiert und

die grafische Darstellung und die Möglichkeit diese auszulösen wird im Operateursarbeitsplatz auf einem zweiten Rechner dargestellt. Auch wird ein Framework für Automatiken entwickelt, die in SAM regelnd eingreifen sollen. Das Framework soll dabei sicherstellen, dass beliebige Automatiekn für die Reglung von SAM integriert werden

Der Arbeitsplatz für den Operateur bietet ihm die Möglichkeit regelnd in den Prozess einzugreifen. Hierbei stehen ihm harte und weiche Eingriffe zur Verfügung. Harte Eingriffe bieten ihm die Möglichkeit in den Prozess einzugreifen, während weiche Eingriffe auditive und visuelle Hinweise sind. Auch werden ihm Systemparameter und verschiedene Blickwinkel in den Prozess geboten vor allem über ein Video der Mikroweltbewohner und eine Streckenansicht mit Vorausschau.

Die Automatiken sollen vorerst Funktionen des Operateurs umsetzen, später auf Basis von durch Entwickler konzipierte Automatiken ergänzt oder ersetzt werden. Diese von Entwicklern entworfenen Automatiken werden im Vergleich zu den Leistungen des Operateurs verglichen und bewertet. Erste Arbeiten zur technischen Umsetzung der Automatiken sind von Kesselring (2009) dokumentiert.

Die einzelnen Softwarekomponenten von ATEO werden mit Smalltalk und der Programmierumgebung squeak (http://www.squeak.org) entwickelt. Dabei wurde bei der Entwicklung von SAM anfangs der Entwurf einer Softwarearchitektur vernachlässigt, so dass diese inzwischen unsystematisch ist. Es gab zusätzlich sehr viel toten Quellcode, was darauf schließen lässt, dass die Funktionalität von SAM vielfach den Anforderungen des Projektes angepasst wurde. Alte nicht mehr verwendete Funktionalität wurde nicht entfernt, um die Möglichkeit zu gewähren wieder zu dieser alten Funktionalität zurückzukehren. Auch macht eine übermäßige Nutzung von globalen Variablen das Verstehen der Beziehungen zwischen den Klassen schwerer. Aus diesem und anderen Gründen wurde die Softwarearchitektur gründlich analysiert, dokumentiert und eine neue verbesserte vorgeschlagen. Die Ergebnisse können bei Hildebrandt (2009) nachgelesen werden.

Im Rahmen einer generischen Toolbox soll dort die Funktionalität und die neue Softwarearchitektur von SAM in Java implementiert werden. Dabei wird sich nicht mehr an den konkreten Fragestellungen des ATEO Projektes orientiert. So wird es anderen Forschern ermöglicht eigene Experimente mit SAM durchzuführen und auch Aspekte zu berücksichtigen, wie z.B. neue Eigenschaften der Strecke oder des Steuerobjektes, welche bis jetzt im Rahmen des ATEO Projektes vernachlässigt wurden. Auch können fehlende physikalische Eigenschaften des Fahrobjektes hinzugefügt werden.

Weitere Arbeiten im ATEO Projekt umfassen die Programmierung eines Auswerteprogramms für die Logdateien von SAM. Programmiert in Java soll die Auswertung der empirischen Daten nach einem Versuch automatisiert und damit erleichtert werden (Zeitersparnis und Genauigkeitsgewinn).
Entwickler an den einzelnen Produkten sind oder waren:

SAM

- Knut Polkehn
- Nicolas Niestroj
- Simon Dullat
- Sascha Hanse

Operateursarbeitsplatz:

- Hermann Schwarz
- Christian Leonhard

Automatiken

- Kai Kesselring
- Michael Hasselmann

Generische Toolbox

- Michael Hildebrandt

Logfileauswertung

- Tobias Hampel

Die Koordination dieses Teams sowie die Kommuniktion und der Informationsaustauch bei der Anfertigung dieser Diplomarbeit stellte eine besondere Herausforderung dar. Zur Zeit entsteht ein Wiki, welches Informationen sammelt und als Nachschlagewerk dienen soll. Ebenso wurde ein Forum für Diskussionen eingerichtet, welches aber nur sporadisch genutzt wird. Die örtliche Nähe zu den Projektleitern Saskia Kain und Jens Nachtwei sowie die intensive Arbeit an SAM und Squeak führten zu einer umfassenden Wissensbasis.

1.2 ZIELSTELLUNG DER ARBEIT

Die Mikrowelt SAM soll erweitert und angepasst werden, so dass diese Version im Anschluss mit dem Operateursarbeitsplatz oder den Automatiken vernetzt werden kann. Diese Vernetzung soll geplant und in Soft- und Hardware umgesetzt werden.

1.3 VORARBEITEN IN DER STUDIENARBEIT

Im Rahmen meiner Studienarbeit (Niestroj, 2008) wurden die Erweiterungen Tachometer, neue Gabelungen, Manipulation des Steuerungsinputs der Mikroweltbewohner sowie statische und dynamische Hindernisse definiert. Diese Komponenten des Systems wurden eingeführt, um die Komplexität der Strecke für die Mikroweltbewohner zu erhöhen und vor allem mit verbesserten Gabelungen Entscheidungssituationen entstehen zu lassen. Die Genese dieser Erweiterungen wurde ausführlich dargestellt.

In einem zweiten Schritt wurden diese Erweiterungen in einer ersten Version implementiert. Die grundlegenden Ideen sind auch noch in der Endversion von SAM 1.5 erhalten geblieben, die Inputmanipulation dagegen wurde neu konzipiert und umgesetzt, um vor allem die Verständlichkeit zu erhöhen und die Softwarearchitektur zu verbessern. Im ersten Entwurf der statischen Hindernisse wurden noch einfache Hindernisse angedacht, inzwischen ist man aber zu einem Slalom mit zwei Hindernissen übergegangen.

Desweiteren wurde SAM von der Squeak Version 3.7 auf die Version 3.9 aktualisiert. Dies hatte zur Folge, dass das Konfigurationsmenü ohne die Hilfe von eToys (Website: Etoys oder Squeakland, letzter Zugriff:

02.07.2009) neu implementiert werden musste, da es nicht weiter nutzbar war. Die mit dem alten Menü verbundenen Aktionsskripte sowie die Anbindung an den zugrunde liegenden Smalltalk-Quellcode war in der Squeak Version 3.9 nicht mehr funktionsfähig. Deshalb wurde eine vorläufige Version dieses Konfigurationsmenüs entworfen und umgesetzt.

Das Problem der Performanz wurde erkannt und in einem Kapitel der Studienarbeit diskutiert. Dabei wurden als Engpässe vor allem verschachtelte Morphe, wie sie im Fahrobjekt mit seinen neun Sensormorphen vorlagen, identifiziert. Die Performanz im ATEO Projekt wird mit der durchschnittlichen Zeit gemessen, die es benötigt, einen Morphstep durchzuführen, d.h. einen Programmschritt zu berechnen. Abgelesen werden kann dieser Kennwert in der Logdatei, und zwar in der Spalte namens *TimeDelta*. Mögliche Lösungswege wurden in Zusammenarbeit mit Simon Dullat erörtert und im Anschluss an die Studienarbeit umgesetzt.

Die Arbeiten an den Erweiterungen von SAM wurden in der anschließenden Diplomarbeit weitergeführt und zu einem stabilen Endzustand geführt.

1.4 ERGEBNISSE UND AUFBAU DER ARBEIT

Im Rahmen der vorliegenden Arbeit ist eine stabile Version von SAM, welche den von den Projektleitern gewünschten Funktionsumfang enthält, entwickelt worden. Toter Quellcode wurde entfernt und die Anzahl der globalen Variablen verringert. Die Softwarearchitektur wurde geringfügig verbessert. Diese Version wird inhaltlich keine weiteren Anpassungen erfahren und kann somit als Basisversion für die Integrierung von Automatiken oder des Operateursarbeitsplatzes fungieren. Auf dieser SAM Version aufbauend wurde eine Vernetzung mit dem Operateursarbeitsplatz umgesetzt. Dabei sind grundlegende Funktionen des Operateurs über das Netzwerk nutzbar. Einige Elemente wurden nicht vernetzt, da die Funktionalität des Operateursarbeitsplatzes noch nicht vollständig den Anforderungen des Projektes entsprechend umgesetzt wurde. Dies betrifft die Anzeige der Joystickauslenkung, welche zu grob eingeteilt ist. Die Streckenvorschau im Operateursarbeitsplatz ist zwar schon vernetzt, aber noch nicht voll funktionstüchtig. So weit es möglich war, wurde SAM bereits auf kommende Erweiterungen am Operateursarbeitsplatz angepasst und vorbereitet. Squeak hat prinzipiell Schwierigkeiten Morphe in kurzen Zeitabständen zu bewegen. Da in Squeak Bilder als Morphe gespeichert werden, die jedes einzelne Farbbit in einer Variable speichern, entstehen schnell sehr große Bildmorphe. Dadurch wird die Performanz beeinflusst. Durch die Vernetzung konnte auch die Performanz von SAM und des Operateursarbeitsplatzes gleich gehalten werden, auch wenn die Streckenvorschau auf dem Verschieben von Bildern basiert.

In Bezug auf die Automatiken wurde zunächst von einer Vernetzung abgesehen, da zur Zeit der Entwicklung keine Performanceprobleme vorlagen, welche eine Vernetzung nötig gemacht hätten.

In Kapitel 2 sollen eine kurze Einführung in verteilte Systeme gegeben und die verwendeten Technologien näher beschrieben werden. Dies soll dem Leser die Entscheidungen in späteren Kapiteln nachvollziehbarer machen.

Kapitel 3 beschreibt dann die Entwicklungen an der Softwarekomponente SAM. Dabei wurde eine Version vom 31.07.2007, zuletzt entwi-

ckelt von Maik Burandt (Burandt, 2007) stetig erweitert und angepasst. Eine Zwischenstufe wurde in Niestroj (2008) und Hildebrandt (2009) bereits dokumentiert. Hier sollen in Kapitel 3 die Entwicklungen bis zur finalen Version beschrieben werden, wie sie als Grundlage für eine Vernetzung mit dem Operateursarbeitsplatz diente.

Diese Vernetzung wird dann in Kapitel 4 beschrieben, wobei auf die aufgetretenden Probleme sowie deren Lösung eingegangen wird. Auch wird der Nutzen einer Vernetzung im ATEO Projekt kritisch betrachtet.

In Kapitel 5 wird die Arbeit kritisch diskutiert und in Kapitel 6 zusammengefasst und mit einem kurzen Ausblick geschlossen.

Der Anhang bekommt in dieser Arbeit eine besondere Bedeutung. Dieser soll als Einführung und Nachschlagewerk für nachfolgende Mitarbeiter im ATEO Projekt dienen und den Einstieg in die Bedienung und das Verständnis der Softwarekomponente SAM, der vorgenommenen Vernetzung mit dem Operateursarbeitsplatz und der Arbeit mit Squeak und Smalltalk erleichtern.

Der Anhang A beschreibt alle Änderungen, die seit der in Hildebrandt (2009) dokumentierten Version im Rahmen von ATEO 2.0 vorgenommen wurden. Hierbei wird zwischen den unterschiedlichen Entwicklungsstufen von SAM unterschieden und jeweils auf die Vorgängerversion Bezug genommen.

Im Anhang B wird die im Rahmen einer Anforderungsanalyse entstandene Tabelle der Hierarchischen Aufgabenanalyse des Operateursarbeitsplatzes dargestellt.

Anhang C beschreibt die Konfiguration von SAM, den Versuchsstart bei aktiver Vernetzung sowie eine Schritt für Schritt Anleitung für eine Portierung von SAM oder des Operateursarbeitsplatzes in ein neues Squeak Image.

Anhang D bietet kurze Smalltalkbeispiele für eine Vernetzung mit Sockets.

2 THEORIE

2.1 EINORDNUNG IN DEN SOFTWARE DEVELOPMENT LIFECYCLE

Um die sich ständig auf Grundlage neuer empirischer und theoretischer Erkenntnisse ändernden Anforderungen zeitnah umsetzen zu können, wurde eine iterative, inkrementelle Vorgehensweise gewählt, welche von Basili und Turner (1975) als Alternative zum Wasserfallmodell für den Softwareentwicklungsprozess vorgeschlagen wurde. Über den geschichtlichen Entstehungsprozess sowie den Einsatz in Projekten kann in Larman und Basili (2003) nachgelesen werden. Das inkrementelle oder iterative Vorgehensmodell ist mit dem Spiralmodell (Boehm, 1988) verwandt und beschreibt den kontinuierlichen Verbesserungsprozess, bei dem die Anwendung evolutionär wächst. Im Zuge der iterativen Entwicklung entstehen so in kürzeren Abständen Prototypen, welche evaluiert werden können. Im Zuge des Prozesses können sich die Anforderungen ändern aber ‚nicht wie beim Wasserfallmodell, ohne Probleme umgesetzt werden. Dieses Entwicklungsmodell ist auch die Basis der agilen Softwareentwicklung (Cohen u. a., 2004). Im ATEO Projekt konnten durch dieses flexible Vorgehen funktionierende Prototypen diskutiert und verfeinert werden. Durch empirische Untersuchungen wurden die entwickelten Komponenten verifiziert, auf ihre Brauchbarkeit getestet und im Anschluss weiter verfeinert. Da ein Schwerpunkt der Arbeit die Vernetzung war, wurde eine enge Zusammenarbeit mit den übrigen Projektmitarbeitern gepflegt. Der Einsatz in allen Teilen des Projektes war unvermeidbar und die Informationen aus technischer und inhaltlicher Sicht wurden so gebündelt. Am Ende jeder Entwicklungsstufe wurde das Programm ausführlich getestet und Fehler oder ungewolltes Verhalten dokumentiert und im Anschluss entfernt. Auch das Testen wurde für sich genommen iterativ durchgeführt.

2.2 HIERARCHISCHE AUFGABENANALYSE

Im Rahmen der Anforderungsanalyse für den Operateursarbeitsplatz und der Automatiken wurde eine Hierarchische Aufgabenanalyse (Stanton, 2006) zusammen mit den Informatikern Hermann Schwarz, Kai Kesselring und Michael Hildebrandt sowie den Psychologen Saskia Kain und Jens Nachtwei durchgeführt. Mit Hilfe dieser konnten die Aufgaben des Operateurs und der Automatiken systematisch erfasst werden. Ebenfalls konnte eine gemeinsame Sprache gefunden werden, die von beiden Seiten (Psychologen und Informatiker) verstanden werden konnte. Dabei musste der Informatiker sich auf eine neue Herangehensweise einlassen. Die hierarchische Aufgabenanalyse beginnt mit der Definition eines übergeordneten Zieles, um davon ausgehend mit der Zerlegung dieses Zieles in Unterziele fortzuschreiten. Am Ende der Zerlegung stehen konkrete Unterziele, die im Idealfall nicht weiter zerlegt werden können bzw. deren Zerlegung keinen inhaltlichen Mehrwert mehr birgt.

In sieben Iterationsschritten wurde diese Aufgabenanalyse weiter den Anforderungen des Operateursarbeitsplatzes und der Automatiken entsprechend angepasst und verfeinert. Als Darstellungsform für eine

Hierarchische Aufgabenanlyse bietet sich bei größeren Analysen die tabellarische Auflistung an. Die zur Zeit aktuelle Tabelle befindet sich im Anhang B.

Die Aufgabenanalyse hat dazu beigetragen sich der Anforderungen klar zu werden und sie in einer für Informatiker und Psychologen gemeinsamen Sprache zu notieren und zu verstehen. So konnten schon früh im Projekt die grundlegenden Eigenschaften des Operateursarbeitsplatzes sowie der Automatiken identifiziert und niedergeschrieben werden. Auf Basis dieser Tabelle wurde auch der Operateursarbeitsplatz entwickelt und SAM auf dessen Funktionalität hin erweitert. Ein Beispiel ist hier die Möglichkeit die Maximalgeschwindigkeit zu ändern. Diese Anforderung wurde früh definiert und ist sogar in SAM Solo Tracking eingeflossen.

Die Hierarchische Aufgabenanalyse unterscheidet sich von der Strukturierten Analyse oder Use Cases. Bei Use Cases stehen Systemziele im Mittelpunkt und nicht wie bei der Hierarchische Aufgabenanalyse Ziele eines Nutzers des Systems. Die Strukturierte Analyse hat im Vergleich einen anderen Fokus, nämlich das Modellieren von Funktionen und Datenflüssen, aber es wird auch eine hierarchische Sicht auf die Daten und Prozesse verfolgt. Durch diese Ähnlichkeit zur hierarchischen Aufgabenanalyse ist es einfach als Informatiker diese Methode anzuwenden.

Erfahrungen aus diesem Prozess sind in Hildebrandt u. a. (2009) eingeflossen.

2.3 SQUEAK UND SMALLTALK

Entwickelt von Alan Kay, Ted Kaehler, John Maloney, Scott Wallace und Dan Ingalls, welche unter anderem schon bei der Entwicklung von Smalltalk-80 entscheidend mitgewirkt haben, wurde Squeak in seiner ersten Version im September 1996 veröffentlicht. In Ingalls u. a. (1997) beschreiben die Entwickler von Squeak, dass die Motivation darin bestand eine Entwicklungsumgebung für Lernsoftware zu entwicklen, welche auch von weniger technikaffinen Erwachsenen oder Kindern benutzt werden kann. Dabei waren die Anforderungen Effizienz in Hinblick auf die Speichergröße und Rechenleistung, Portabilität und ein einfaches einheitliches Design, welches ohne Verzögerung angepasst werden konnte. Da Smalltalk zwar den Anforderungen entsprach, es aber keine brauchbare Implementierung zu der Zeit gab, beschlossen sie ein eigenes Smalltalksystem zu entwicklen. Dabei mussten sie nicht von vorne beginnen, sondern konnten auf ein vorhandenes Apple Smalltalk-80 System aufbauen. Squeak ist nun knapp 13 Jahre später in der Version 3.10 mit einer Version 4.0 im Betastadium erhältlich. Die Eigenschaften von Smalltalk und Squeak haben es zu der meist benutzten nichtkommerziellen Smalltalkimplementierung gemacht. Im Folgenden soll kurz auf die Eigenschaften eingegangen werden, die Squeak und Smalltalk von anderen objektorientierten Sprachen wie Java oder C++ unterscheidet.

Smalltalk setzt die Eigenschaften der Objektorientierung streng um, was sich darin äußert, dass alles ein Objekt ist und dies wie in keiner anderen Sprache auch konsequent umgesetzt ist. Weiterhin sind alle Methoden öffentlich und alle Attribute privat. Dies gilt ohne Ausnahme. Smalltalk ist dynamisch getypt, unterstützt einfache Vererbung sowie Polymorphismus. Auch ist Smalltalk reflexsiv in sich selbst programmiert. Dadurch hat man jederzeit Einblick in die Klassenbibliotheken

von Smalltalk und kann Anpassungen vornehmen oder anhand dieser Konzepte für eigene Klassen ableiten. Ein weiterer Vorteil ist der Umstand, dass alle Änderungen sofort im Smalltalkimage aktiv sind, d.h. es gibt keine Kompilierungs- und Linkzyklen. Die aus Java bekannte Virtual Machine sowie die Garbage Collection werden auch in Smalltalk verwendet.

2.4 VERTEILTE SYSTEME

Im Folgenden soll eine kurze Einführung in verteilte Systeme gegeben werden, sowie die Möglichkeiten der Kommunikation in einem solchen Netzwerk aufgezeigt werden. Ebenso sollen Bibliotheken für squeak vorgestellt werden, die im Kontext der Vernetzung im ATEO Projekt von Interesse sind.

Es gibt eine Reihe von Definitionen für verteilte Systeme. Tanenbaum und van Steen (2008) definieren es folgenderweise:

Definition 2.4.1 (Verteiltes System) *Ein verteiltes System ist eine Ansammlung unabhängiger Computer, die den Benutzern wie ein einzelnes kohärentes System erscheinen.*

Desweiteren listen Tanenbaum und van Steen (2008) vier Ziele auf, die mit einem verteilten System erreicht werden sollen. Dies sind im einzelnen der Zugriff auf Ressourcen (beispielsweise Computer im Netzwerk oder eine Datenbank), die Verteilungstransparenz, Offenheit und Skalierbarkeit. Die Transparenz ist in der Netzwerktechnik folgenderweise zu verstehen: ein bestimmter Aspekt eines verteilten Systems ist zwar vorhanden, wird aber vom Nutzer dieses Systems nicht wahrgenommen. Es gibt acht mögliche Arten der Transparenz, so sind Zugriffstransparenz, Ortstransparenz, Migrationstransparenz, Relokationstransparenz, Replikationstransparenz, Nebenläufigkeitstransparenz, Persistenztransparenz und Fehlertransparenz in ISO/IEC 10746-1 (1998) definiert. Exemplarisch soll die Ortstransparenz erklärt werden, für andere sei auf den ISO/IEC Standard verwiesen. Bei der Ortstransparenz bleibt dem Nutzer unbekannt, wo sich der gewünschte Dienst oder die gewünschte Ressource physikalisch befindet. Der Zugriff erfolgt z.B. über das Internet über die URL, diese enthält aber keine Informationen über den Standort des Servers. Selbst die Top-Level-Domain (.de, .com oder .biz) ist nicht bezeichnend, der Server kann in einem anderen Land stehen. Beim Design verteilter Systeme sollte die Transparenz ein wichtiges Kriterium sein. Es sollten aber auch andere Aspekte wie die Leistung oder die Verständlichkeit im Designprozess mitberücksichtigt werden. So kann es notwendig sein, beispielsweise den Ausfall eines Servers (Fehlertransparenz) nicht zu verbergen, um die vom Client aus gestarteten erfolglosen Anfragen zu minimieren und damit die Zeit bis zur Fehlerrückmeldung zu reduzieren.

Offenheit bezeichnet die Eigenschaft verteilter Systeme Dienste nach bestimmten Standardregeln anzubieten. Diese Regeln werden durch Netzwerkprotokolle definiert, während die Dienste über Schnittstellenbeschreibungen definiert werden. Dies soll ermöglichen Dienste einfach nutzen und mit ihnen kommunizieren zu können, Algorithmen von Diensten bei gleichbleibender Schnittstelle auszutauschen und Dienste durch Komposition mehrerer zusammenstellen zu können.

Ein verteiltes System ist skalierbar, wenn man ohne Probleme weitere Ressourcen oder Benutzer hinzufügen kann, das System sich geogra-

fisch an entfernten Orten oder sogar über unterschiedliche Organisationen oder Unternehmen erstrecken kann.

Allgemein kann man drei Arten von verteilten Systemen unterscheiden: verteilte Computersysteme, verteilte Informationssysteme und verteilte pervasive Systeme. Verteilte Computersysteme sind in einem Netzwerk zusammengefasste Computer, welche für Hochleistungsberechnungen genutzt werden. Man unterscheidet dabei zwischen Cluster-Computern (Baker, 2000) und Grid-Computern (Jacob u. a., 2005). Cluster-Computersysteme bestehen aus einer beliebigen Anzahl von gleichartigen Computern, die das gleiche Betriebssystem haben und in einem Netzwerk organisiert sind. Im Gegensatz dazu stehen die Grid-Computersysteme, die beliebige Computer, die unter anderem Cluster-Computer sein können, auch über Unternehmensgrenzen hinaus vernetzen. Diese Heterogenität birgt viele Herausforderungen in Bezug auf die Kommunikation und die Zuteilung von Ressourcen. Mit Hilfe von Web Services kann eine Architektur definiert werden, die ein solches Netzwerk aufbauen und verwalten kann (OGSA - Open Grind Services Architekture (Foster u. a., 2006)).

Unter verteilten Informationssystemen versteht man die Netzanwendungen wie Enterprise Application Integration (EAI) oder Client-Server Anwendungen mit Datenbankanbindung.

Mit verteilten pervasiven Systemen ist in der Regel Ubiquitous Computing gemeint, also das allgegenwärtige Vorhandensein von Computern wie Handys, PDAs etc. und das ad hoc Zusammenschließen dieser in einem Netzwerk (Weiser, 1991). Ein weiteres Beispiel sind Sensornetze (Akyildiz u. a., 2002). Das ATEO Projekt lässt sich hier klar in die verteilten Informationsysteme einordnen, da lediglich zwei maximal drei Rechner vernetzt werden sollen. Es wird weder ein Cluster oder Grid augebaut noch werden die Rechner ad hoc verbunden mit Handys oder PDAs.

Die Definition verteilter System von Coulouris u. a. (2002) betont stärker einen anwendungsorientierten Ansatz.

Definition 2.4.2 *Bei einem verteilten System arbeiten Komponenten zusammen, die sich auf vernetzten Computern befinden und die ihre Aktionen durch den Austausch von Nachrichten koordinieren.*

Beide Aspekte, Hardware und Software, aus den Definitionen sind wichtig und haben Einfluss auf die konkrete Architektur des Systems. Im folgenden sollen verschiedene Architekturstile und die jeweilige Realisierung vorgestellt werden. Der Fokus wird sich dabei in Richtung der Anwendungen verlagern, da es vornehmlich um die Verteilung dieser geht.

2.4.1 *Die Systemarchitektur*

Geschichtete Architekturen

Eine Schichtenarchitektur strukturiert und modularisiert ein System. Ein entscheidener Vorteil ist die Austauschbarkeit der konkreten Realisierung einer Komponente, solange sie der unter ihr liegenden Schicht dieselben Dienste anbietet. Dies ermöglicht eine einfache Anpassung, Erweiterung oder sogar Neuimplementierung von Systemkomponenten ohne an anderen Bestandteilen des Systems Anpassungen vornehmen zu müssen. Diese Architektur ist auch Grundlage für die Strukturierung der Kommunikation im Internet und wurde im ISO/OSI Referenzmo-

dell festgehalten. Im Abschnitt über die Kommunikation wird darauf näher eingegangen.

Objektbasierte Architekturen

In objektbasierten Architekturen werden die einzelnen Komponenten eines Systems lose angeordnet und kommunizieren untereinander durch entfernte Prozeduraufrufe. Dieser Stil wird gut von der Client-Server-Architektur umgesetzt. Diese zentralisierte Systemarchitektur besteht aus einem Server als Dienstanbieter und einem Client als Nutzer dieses Dienstes. Eine Verteilung der Anwendung auf Client und Server kann unterschiedlich vorgenommen werden. Das Programm kann vollständig auf Seiten des Servers liegen, so dass der Client nicht mehr als ein Terminal zum Programm ist oder auf den Client ausgelagert werden. Diese Auslagerung kann unterschiedlich stark vorgenommen werden. Die sogenannten Thin Clients stellen die Benutzungsschnittstelle zur Anwendung bereit und evtl. wird noch die Anwendung auf Server und Client verteilt implementiert. Fat Clients stellen bereits die vollständige Anwendung inklusive Benutzungsschnittstelle und kommunizieren nur noch mit der Datenbank eines Servers. Generell gilt, dass eine starke Auslagerung in Richtung Client unkontrollierbar und fehleranfällig wird und das Programm zu stark abhängig ist von den Resdourcen oder dem Betriebssystem des Clients. Aus diesem Grund geht die Tendenz in Richtung Thin Clients. Für die Kommunikation zwischen Server und Client können entfernte Prozedur- oder Methodenrufe (Remote Procedure Calls/Remote Method Invocation) genutzt werden.

Datenzentrierte Architekturen

In datenzentrierten Architekturen wird ein Dateisystem gemeinsam genutzt und nur über Dateien kommuniziert. Dabei können, müssen aber nicht zwangsläufig, diese auch durch eine Client-Server-Architektur realisiert werden. Webanwendungen sind ein Beispiel für datenzentrierte Architekturen. Hier werden über einen Browser, dem Client, Daten auf einem entfernten Server abgefragt und gesendet. Durch den Einsatz von Web Services muss die Kommunikation nicht mehr unbedingt durch einen Browser abgewickelt werden. Web Services bieten die Möglichkeit Anwendungen zu schreiben, die Dienste und Daten über das Internet zur Verfügung stellen. Dies kann ohne unmittelbar mit dem Endnutzer kommunizieren zu müssen geschehen. Die Basis der Web Services bilden die Standards für WSDL (Web Service Description Language), SOAP (Transportprotokol für die Kommunikation) und UDDI (Universal Description, Discovery and Integration) für das Bekanntgeben, Auffinden und Nutzen von Diensten. Mit Hilfe dieser Architektur können lose gekoppelte und durch Komposition erweiterbare Anwendungen über das Internet zur Verfügung gestellt werden. Diese und die folgende Architektur werden auch durch serviceorientierte Architekturen (SOA) umgesetzt. Erl (2005) beschreibt diese Architektur ausführlich in seinem Buch.

Ereignisbasierte Architekturen

In ereignisbasierten Architekturen kommunizieren Prozesse über die Weitergabe von Ereignissen. Dafür registrieren sich interessierte Prozesse für bestimmte Ereignisse und werden bei Eintreten dieser davon in Kenntnis gesetzt (Publish/Subscribe). Die betroffenen Prozesse können

nun auf dieses Ereignis reagieren, dabei unter anderem auf übermittelte Daten zurückgreifen und wiederum selbst ein Ereignis auslösen. Technisch realisiert werden kann dieses Verhalten wiederum mit Web Services.

Architekturen im ATEO Projekt

Im ATEO Projekt sind geschichtete Architekturen von Relevanz, da sie die Basis der im Anschluss dargestellten Netzwerkkommunikation sind und für eine Neuimplementierung von SAM im Rahmen der generischen Toolbox in Hildebrandt (2009) vorgeschlagen wurden. Auch fiel die Wahl der Systemarchitektur von ATEO auf objektbasierte Architekturen, welche von Client/Server Anwendungen repräsentiert werden. Durch den Einsatz von SOAP bei der Vernetzung von SAM mit dem Operateursarbeitsplatz werden am Rande auch datenzentrierte oder ereignisbasierte Architekturen interessant, auch wenn alleine mit SOAP keine solchen aufgebaut werden können.

2.4.2 *Die Kommunikation*

Um die Kommuniktion über ein Netzwerk zu realisieren, werden Netzwerkprotokolle definiert. Diese Netzwerkprotokolle sind in einer Schichtenarchitektur organisiert, die durch das ISO/OSI-Referenzmodell (ISO/IEC 7468-1, 1994) definiert sein können.
Das ISO/OSI-Referenzmodell definiert sieben Schichten, welche von oben nach unten angeordnet sind:

- Anwendungsschicht
- Darstellungsschicht
- Sitzungsschicht
- Transportschicht
- Vermittlungsschicht
- Sicherungsschicht
- Bitübertragungsschicht

Die Abstraktion nimmt von unten nach oben zu. Die unterste Schicht definiert dabei die physikalische Übertragung und die oberste Schicht bietet dem Anwendungsentwickler eine komfortable Schnittstelle für die Programmierung von verteilten Systemen. Für die Verbindung von Netzwerken eignete sich das ISO/OSI-Referenzmodell allerdings weniger, da es beim Entwurf nicht vorgesehen und berücksichtigt wurde. Aus diesem Grund wurde das TCP/IP-Referenzmodell definiert, um den Protokollen des Internets eine Referenz zu geben. Hier gibt es im Vergleich zum ISO/OSI-Referenzmodell nur vier Schichten, nämlich die Anwendungsschicht, welche die Anwendungsschicht, Darstellungsschicht und die Sitzungsschicht zusammenfasst, die Transportschicht, die Internetschicht, welche der Vermittlungsschicht entspricht, sowie die Netzwerkschnittstelle, welche die Sicherungsschicht und die Bitübertragungsschicht vereint. In Braden (1989a,b) werden diese vier Schichten sowie wichtige Protokolle definiert und näher beschrieben.

Für jede Schicht im ISO/OSI-Referenzmodell sowie im TCP/IP-Referenzmodell gibt es zahlreiche Protokolle und eine über das Netzwerk versendete Nachricht durchläuft den gesamten Protokollstapel.

Im Zuge dessen erweitert jedes Protokoll die Nachricht um einen spezifischen Header, der beim Zielhost wieder ausgelesen, interpretiert und dann entfernt wird.

Im Folgenden soll sich auf die Anwendungsschicht und die Transportschicht mit seinen Protokollen Hypertext Transfer Protocol (HTTP), Transmission Control Protocol (TCP) und User Datagram Protocol (UDP) beschränkt werden. Die beiden anderen Schichten waren für die Vernetzung im ATEO Projekt nicht relevant und sind es in der Regel für die Programmierung verteilter Anwendungen auch nicht.

Anwendungsschicht

Die Anwendungsschicht ist die oberste Schicht und stellt Netzanwendungen bereit, die in der Regel in einer Client-Server-Architektur realisiert werden. Bekannte Protokolle dieser Schicht sind in Bezug auf das Internet beispielweise HTTP, File Transfer Protocoll (FTP) oder Simple Mail Transfer Protocol (SMTP). Im Rahmen des Projektes war vor allem HTTP in Bezug auf SOAP Nachrichten interessant. HTTP realisiert das Versenden von Request und Response Nachrichten zwischen einem Client und einem Server. Desweiteren kann man festlegen, ob die Verbindung zwischen Server und Client persistent oder nichtpersistent sein soll. Wenn, wie bei der Vernetzung im ATEO Projekt, für eine längere Zeit Daten verschickt werden sollen, ist eine persistente Verbindung durch den eingesparten Mehraufwand eines Verbindungsaufbaus einer nichtpersistenten Verbindung vorzuziehen.

Eine Kommunikation zwischen Programmen in verteilten Systemen findet zwischen Prozessen statt. Die beiden kommunizierenden Prozesse werden von Kurose und Ross (2008, Seite 115) so definiert:

Definition 2.4.3 (Client- und Server-Prozesse) *Im Kontext einer Kommunikationssitzung zwischen einem Paar von Prozessen wird der Prozess, der die Kommunikation eröffnet (also erstmals den anderen Prozess zu Beginn der Sitzung kontaktiert), als* ***Client*** *bezeichnet. Der Prozess, der darauf wartet, zu Beginn einer Sitzung angesprochen zu werden, ist der* ***Server****.*

Eine Netzanwendung muss also auf beiden Seiten der Verbindung implementiert werden und versendet seine Nachrichten über Sockets. Sockets bilden die Schnittstelle zwischen dem Prozess in der Anwendungsschicht und dem Protokoll in der Transportschicht und bieten dem Programmierer die Funktionalität für eine Interprozesskommunikation. Diese folgt dem einfachen Konzept:

1. Verbindungsaufbau
2. Datenaustausch
3. Verbindungsabbau

Transportschicht

Die Transportschicht bietet aus Sicht der Anwendungsschicht eine logische Kommunikation zwischen Prozessen. Dabei bleibt dem Prozess der Anwendung verborgen, dass zwischen ihm und seinem Zielprozess beliebig viele Zwischenstationen in Form von Routern stehen und die Daten auf den unterschiedlichsten physikalischen Medien transportiert werden können. Die Abgrenzung der Transportschicht von der Netzwerkschicht leistet genau dies. Die zu sendenden Daten werden von den Prozessen gesammelt und an das Netzwerk weitergegeben, welches

dann für die Zustellung zuständig ist. Die Netzwerkschicht realisiert also eine Host-to-Host Kommunikation und die Transportschicht stellt sicher, dass die empfangenen Nachrichten an die richtigen Prozesse weitergeleitet werden. Dieser Vorgang ist als Multiplexing bzw. Demultiplexing (Kurose und Ross, 2008) oder Downward-Multiplexing bzw. Upward-Multiplexing (Tanenbaum, 2003) bekannt. Die beiden am weitesten verbreiteten Protokolle der Transportschicht sind UDP und TCP und sollen im Folgenden beschrieben werden.

USER DATAGRAM PROTOCOL (UDP) UDP ist im Request For Comment (RFC) 768 (Postel, 1980) definiert als ein Protokol, welches das Minimum an Protokollmechanismen zur Verfügung stellt und keine Garantie für die Zuverlässigkeit der Übertragung gibt. UDP stellt keine Verbindung mit dem entsprechenden Prozess her, ist also verbindungslos, und überprüft auch nicht, ob die Nachricht verloren gegangen ist oder beim Übertragen verändert wurde. Die offensichtlichen Nachteile gleicht UDP damit aus, dass es bei Überlastung des Netzwerkes nicht gedrosselt wird. Dies ist bei Echtzeitanwendungen, die eine kurze Übertragungszeit fordern, ideal. Durch die geringe Bearbeitung der Daten in der Transportschicht wird die Verzögerung gering gehalten und ein Verlust von wenigen Nachrichten fällt beim hochfrequenten Senden nicht ins Gewicht. So ist es auch im ATEO-Projekt, wo ca. alle 40 Millisekunden Daten geschickt werden. Sollte einmal eine Nachricht verloren gehen, wird dies nicht weiter auffallen. UDP besitzt eine Fehlererkennung über Prüfsummen, um eine Veränderung der Daten zu erkennen. Sollte eine Nachricht verändert worden sein, wird der Fehler aber nicht behoben, die Nachricht wird verworfen. So können fehlerhafte Daten beispielsweise keinen Einfluss nehmen auf die Streckenvorschau des Operateursarbeitsplatzes.

TRANSMISSION CONTROL PROTOCOL (TCP) TCP, das Transportprotokoll des Internets, wurde von Robert E. Kahn und Vinton G. Cerf entwickelt (Cerf und Kahn, 1974). Im Vergleich zu UDP ist TCP verbindungsorientiert und zuverlässig. TCP ist verbindungsorientiert, weil zwei Prozesse vor der Kommunikation einen Drei-Wege-Handshake durchführen. Initiiert wird der Handshake vom Client, welcher ein spezielles TCP Segment zum Serverprozess schickt. Der Serverprozess bestätigt die Anfrage mit einem eigenen speziellen TCP-Segment und reserviert die nötigen Ressourcen. Als Reaktion auf dieses TCP-Segment, reserviert der Client Ressourcen und schickt ein letztes TCP-Segment zum Server. Nun ist eine Verbindung aufgebaut und die Kommunikation kann beginnen. Dafür setzt der Client Daten durch einen Socket ab und diese Daten werden im Sendepuffer zwischengespeichert bis das IP Protokoll der Netzwerkschicht die Daten in Richtung Server schickt. Dort angekommen, wird das Segment mit den Daten in einem TCP Empfangspuffer zwischengelagert bis der Socket des Serverprozesses die Daten verarbeiten kann. Ein TCP Segment ist wesentlich komplizierter aufgebaut als ein UDP-Segment. Unter anderem werden Sequenznummern, Acknowledgement-Nummern, Empfangsfenster-Felder und verschiedene Flags zusätzlich zu den auch im UDP-Segment genutzten Quell- und Zielport, Header-Länge und Prüfsumme genutzt. TCP bietet einige mit diesen Feldern verbundene Zusatzleistungen an. Dies sind zuverlässiger Datenverkehr, realisiert durch die Sequenznummer und Acknowledgement-Nummern sowie Datenflusskontrolle, realisiert durch das Empfangsfensterfeld. Die Sequenznummer stellt sicher, dass

die Daten in der richtigen Reihenfolge am Zielport ankommen. Die Acknowledgement-Nummer hilft dabei festzustellen, ob Daten verlorengegangen sind, da vom Serverprozess der Erhalt bestätigt wird. Mit Hilfe des Empfangsfensterfeldes kann realisiert werden, dass bei einem überlasteten Netzwerk weniger Daten geschickt werden. Das Empfangsfenster gibt dabei an, wieviel freier Pufferplatz beim Serverprozess noch vorhanden ist. Wird der Platz zu klein, werden die Segmentgrößen verkleinert und schrittweise wieder erhöht. Durch diesen Mechanismus wird entgegengewirkt, dass Daten verworfen werden, wenn der Empfangspuffer voll ist. Das wiederholte Senden von Daten wird dadurch verringert.

UDP besitzt keine Überlastkontrolle, so dass es vorkommen kann, dass UDP das Netzwerk sehr stark belastet und damit TCP-Verbindungen benachteiligt werden. Aus diesem Grund ist allgemein angedacht auch das UDP Protokoll mit Überlastkontrollmechanismen auszustatten (RFC 4340: Datagram Congestion Control Protocol, Kohler u. a. (2006)). Detailliertere Informationen zum TCP-Protokoll können in Kurose und Ross (2008), Tanenbaum (2003) oder im RFC 793 (Postel, 1981) nachgelesen werden.

2.4.3 *Bibliotheken in Squeak*

Socket

Die Bibliothek Socket ist in jedem Squeak der Versionen 3.7 - 3.10 enthalten und bedarf demnach keiner gesonderten Installation. Die Klasse Socket stellt die Funktionalität für die niedrigste Ebene vernetzter Objekte in Squeak zur Verfügung und wird in der Regel nicht benutzt, da die Programmierung fehleranfällig ist. Es wird empfohlen die weiter unten beschriebene SocketStream Klasse zu benutzen. Socket unterstützt die beiden Protokolle TCP und UDP. Als Standard wird bei der Erstellung eines Objektes dieser Klasse immer eine Verbindung mit dem TCP Protokoll erstellt. Möchte man ein Socket mit dem UDP Socket erstellen, muss man explizit ein Objekt mit UDP erstellen. SocketStream benutzt nur den Standardkonstruktor, so dass auch nur das TCP Protokoll von SocketStream angeboten wird. Die Klasse stellt verschiedene Methoden zum Senden, Empfangen sowie zum Warten auf Daten und für den Verbindungsaufbau zur Verfügung.

Für SAM wurde mit Hilfe des UDP Protokolls Daten für die Streckenvorschau versendet. Das TCP Protokoll dagegen wurde genutzt für die Eingriffe des Operateurs.

SocketStream

Die Klasse SocketStream ist auch in den Squeak Images der Versionen 3.7 - 3.10 enthalten. SocketStream ist ein Wrapper von Socket und bietet die Möglichkeit die Netzwerkverbindung wie einen Stream zu benutzen. Das bedeutet, dass auf Seiten des Servers Daten geschrieben und auf Seiten des Clients ausgelesen werden können. Es handelt sich dabei um eine Zwei-Wege-Kommunikation mit zwei logisch getrennten Kanälen, so dass Daten in beide Richtungen fließen können. Dadurch können Emfangsbestätigungen auf einem seperaten Kanal geschickt werden. Theoretisch sollte es auch möglich sein, Daten auf beide Kanäle von unterschiedlichen Seiten aus zu legen, praktisch wurde dies aber nicht versucht.

Die gesendeten Daten können im ascii oder binären Format vorliegen, wobei standardmäßig Strings, also ascii-Daten, verschickt werden. SocketStream benutzt ausschließlich das TCP Protokoll für die Kommunikation. Für die Vernetzung des Operateursarbeitsplatzes wird eine modifizierte Version namens FastSocketStream genutzt. Diese Klasse ist sehr ähnlich zu SocketStream, zumindest das Interface ist gleich. Intern ist der Algorithmus für das Senden und Empfangen von Daten von Göran Krampe (Website letzter Zugriff 02.07.2009), einem Entwickler von squeak-Paketen, neu geschrieben worden, wodurch die Verbindung schneller und flexibler sein soll. Unterschiede sind im Test nicht merkbar, aber unter Berücksichtigung möglicher Performanzprobleme in der Zukunft, wurde die schnellere Implementierung gewählt. Die Klasse ist komfortabler nutzbar als Socket, was sich auch in der Anzahl verwendeter Methoden zeigt.

Für die Kommunikation vom Operateursarbeitsplatz zu SAM wurden diese Streams dazu genutzt die Eingriffe des Operateurs zu schicken. Für Daten von SAM zum Operateursarbeitsplatz wurde aber ein UDP Socket genutzt, so dass die beiden Kanäle des Streams nicht ausgenutzt wurden.

rST

Die Klassenbibliothek kann über SqueakSource (entwicklet von Diego Gomez Deck, letzter Zugriff: 02.07.2009) heruntergeladen und in das Squeak image geladen werden. Diese Bibliothek macht es möglich, Objekte im Netzwerk freizugeben, damit auf diese entfernt zugegriffen werden kann. Dabei werden über einen rSTBroker Objekte angemeldet. Der rSTBroker macht dann den Zugriff über das Netzwerk möglich. Dabei müssen die Objekte selbst nicht weiter angepasst werden. So wird die Nutzung der Klasse vereinfacht. Wie stark die Performanz von SAM durch die entfernten Zugriffe beinflusst wird, ist nicht bekannt.

SOAP-Core

SOAP-Core ist eine Bibliothek für das Versenden von SOAP Nachrichten. Auch diese Bibliothek ist nicht im Squeak image enthalten und muss nachgeladen werden. Dies kann entweder über den Squeakmap Loader geschehen oder über die Entwicklerwebsite von Masashi Umezawa (Letzter Zugriff 02.07.2009). SOAP-Core implementiert den SOAP 1.1 Standard, kann aber auch auf andere Art und Weise die XML Nachrichten kodieren. SOAP-Core wurde explizit für eine Kommunikation zwischen Squeak-Images entwickelt, es gibt aber schon die Möglichkeit mit weiteren Smalltalk-Systemen neben Squeak zu kommunizieren. Diese feste Bindung an Squeak ist in diesem Projekt jedoch kein Problem, ist unter Umständen bei der Planung anderer Projekte zu beachten. Die Bibliothek an sich kann einfach benutzt werden. Die mitgelieferten Beispiele sind zwar die einzige Dokumentation neben dem Quellcode, helfen aber einfache Nachrichten zu versenden.

3 SAM: IMPLEMENTIERUNG UND ANPASSUNG

SAM wurde in mehreren Stufen entwickelt und angepasst und die Genese ist graphisch in Abbildung 1 dargestellt.

SAM 1.0 bestand aus den grundlegenden Funktionen zum Bewegen des Fahrobjektes, der Versuchskonfiguration und des Versuchsablaufes sowie des Loggens wichtiger Systemparameter. Eine letzte Erweiterung für SAM 1.0 war die Implementierung einiger einfacher Funktionen für den Operateur. So war es ihm in dieser ersten Version möglich den beiden steuernden Mikroweltbewohnern Hinweise visueller und auditiver Form zu geben. Diese Erweiterungen und Anpassungen sind in Burandt (2007) dokumentiert.

Die Arbeiten zur Erhöhung der Komplexität von SAM wurden im Anschluss begonnen und in einer Zwischenstufe in Niestroj (2008) dokumentiert. In Hildebrandt (2009) wurde währenddessen eine weitere Zwischenstufe von SAM analysiert und dokumentiert. Im November 2008 wurde SAM 1.5 fertig gestellt und im Dezember 2008 im Labor mit Versuchspersonen geprüft.

Anhand empirischer Daten konnte entschieden werden, welche der neu entworfenden Features für SAM 2.0 nutzbar sind und damit die Basis aller folgenden Untersuchungen definiert werden.

Auf Grundlage von SAM 1.5 wurde eine modifizierte Version erstellt, und zwar SAM Solo Tracking. SAM Solo Tracking soll Nebenfragestellungen im ATEO 2.0 Projekt beantworten und gleichzeitig als Untersuchung für einen Notfallplan dienen, wenn Versuchspersonen ausfallen. So wurde untersucht, in wie weit die mentale Beanspruchung steigt, wenn man beispielsweise die Gesamtgeschwindigkeit des Systems erhöht. Hätten die Erweiterungen an SAM nicht den gewünschten Effekt auf die Komplexität gehabt, wäre diese Anpassung eine letzte Möglichkeit gewesen, diese zu erhöhen.

Es soll im Folgenden näher erläutert werden, in welchem Maße sich die drei Versionen von SAM unterscheiden und welche Konzepte und Überlegungen hinter der Implementierung stecken.

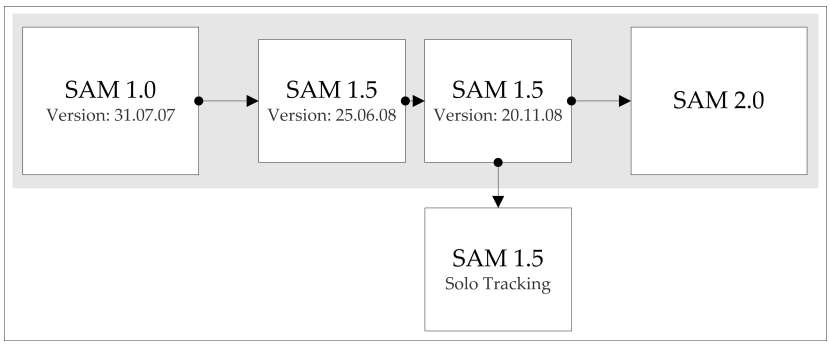


Abbildung 1: SAM Genese

3.1 VERSION 0: AUF DEM WEG ZU SAM 1.5

Als Version 0 werden alle Zwischenstufen zur entgültigen Version von SAM 1.5 bezeichnet, besonders die in meiner Studienarbeit Niestroj (2008) und in Hildebrandt (2009) beschriebenen. So wurden in der Studienarbeit, wie schon in Kapitel 1, erste Konzepte für statische und dynamische Hindernisse probeweise implementiert. Hier haben sich die Anforderungen im Laufe des Projektes geändert, so dass diese später überarbeitet wurden. Die Geschwindigkeitsanzeige wurde in mehreren Versionen implementiert und zu seiner entgültigen Form, wie sie in SAM 1.5 genutzt wurde, entwickelt. Die neuen Gabelungen wurden integriert in die Versuchskonfiguration integriert. Für eine Inputmanipulation, mit der die Joystickinputverteilung der Mikroweltbewohner von einer 50-50 Verteilung automatisch für jeweils einen der beiden Mikroweltbewohner an bestimmten Stellen der Strecke automatisch reduziert und wieder normalisiert werden konnte, wurde die Machbarkeit überprüft. Diese wurde im späteren Verlauf von Grund auf neu gestaltet. Eine Versionsportierung von Squeak 3.7 auf Squeak 3.9 führte zur Neuimplementierung des Konfigurationsmenüs, wobei auch dieses später noch einmal überarbeitet wurde, so dass der Quellcode weniger Redundanz enthielt.

Die Konzepte und Implementierungen sollen im Folgenden anhand der entgültigen Produkte der einzelnen Projektphasen näher erläutert werden.

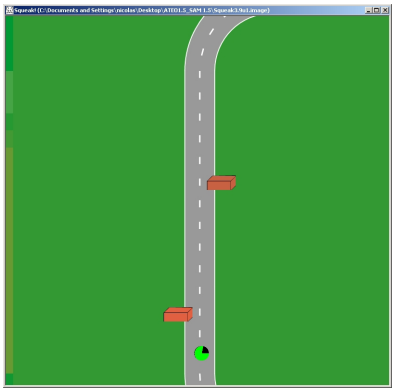
3.2 VERSION 1: SAM 1.5

SAM 1.5 stellt die erste Version dar, die für Untersuchungen im Labor genutzt wurde. Als Ziel der Untersuchungen wurde angestrebt, die neuen Erweiterungen zu validieren und Schwachstellen im Konzept zu entdecken und zu entfernen. Sinn und Zweck dieser Erweiterungen war es, die Trackingaufgabe für die Mikroweltbewohner komplizierter zu gestalten. So sollen für spätere Automatiken oder einem Operateur Situationen geschaffen werden, wo es potentiell zu Konflikten zwischen den Mikroweltbewohnern kommen kann und ein Unterstützungsbedarf durch diese gegeben ist. Im Zuge der Implementierung dieser Erweiterungen sind an vielen Stellen Schwachstellen von SAM aufgefallen. Diese wurden, so weit es möglich war, verbessert. Diese Erweiterungen und alle weiteren Anpassungen an der Software sollen hier beschrieben werden.

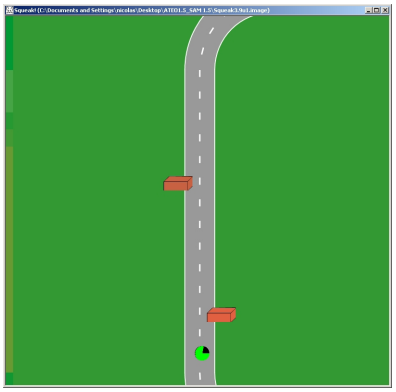
3.2.1 *Konzept*

HINDERNISSE Es wird zwischen statischen und dynamischen Hindernissen unterschieden. Statische Hindernisse sind als Slalom aus zwei Elementen angeordnet und in mehreren Varianten ausgewählt worden. Die Fahrbahnabdeckung variiert zwischen 25% und 50% genauso wie die Position der Hindernisses zwischen links und rechts der Fahrbahn variiert. Die vier Varianten statischer Hindernisse sind in Abbildung 2 veranschaulicht.

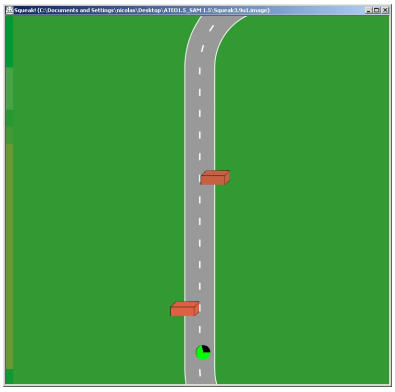
Auf der Strecke kommt zusätzlich ein dynamisches Hindernis vor, welches von links nach rechts die Fahrbahn kreuzt. Ein Kreuzen von rechts nach links ist verworfen worden, da es den Mikroweltbewohnern zu wenig Zeit gibt zu reagieren. Denn die Strecke, die das dynamische Hindernis bis zum potentiellen Zusammenstoß zurücklegen würde, wä-



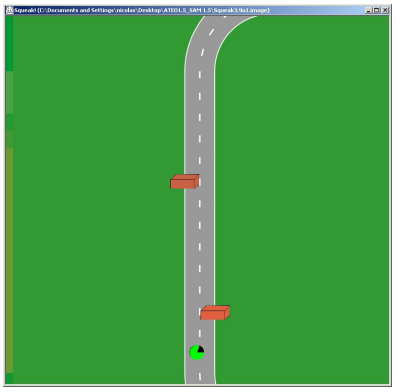
(a) erstes Hindernis links 25% Abdeckung



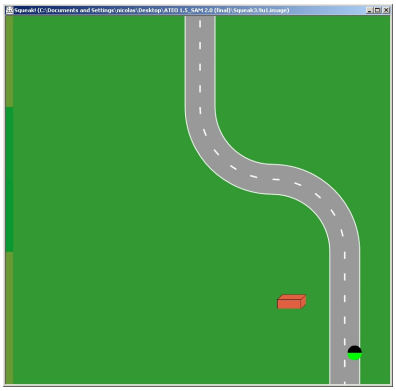
(b) erstes Hindernis rechts 25% Abdeckung



(c) erstes Hindernis links 50% Abdeckung



(d) erstes Hindernis rechts 50% Abdeckung



(e) dynamisches Hindernis

Abbildung 2: Hindernisse

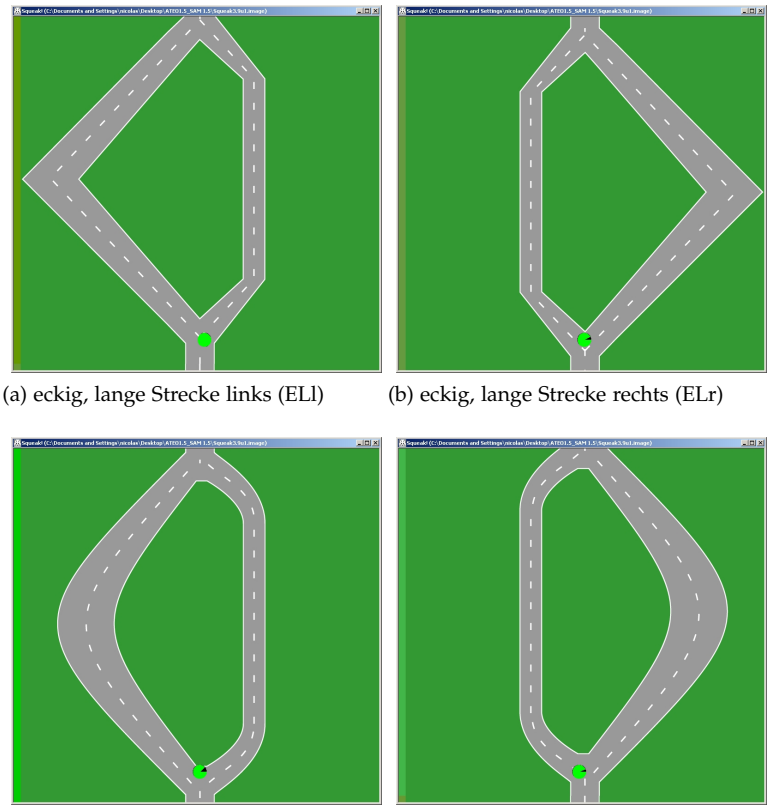
re auf dem dafür benutzten Streckenbild marginal. Die Geschwindigkeit des Hindernisses ist auf die beiden Mikroweltbewohner abgestimmt und wird für jedes Team einmalig auf der ersten kooperativ zu fahrenden Strecke gemessen. Dabei wird auf dem gesamten Streckenbild, auf dem später auch das dynamische Hindernis erscheinen wird, die Geschwindigkeit des Fahrobjektes gemessen. Am Ende wird gemittelt, weil das Streckenbild dreimal vorkommt in der aktuellen Streckenkonfiguration. Es wird also eine adaptive Geschwindigkeitsanpassung vorgenommen, aber für den gesamten Versuch nur während eines einzigen Versuchssteps.

INPUTMANIPULATION Inwieweit man einen Konflikt zwischen den Mikroweltbewohnern hervorrufen kann, wurde unter anderem mit Hilfe der Inputmanipulation untersucht. Dabei wird die Verteilung des horizontalen und vertikalen Joystickinputs manipuliert, wobei von einer 50-50 Verteilung auf beide Mikroweltbewohner ausgegangen wird. Wichtig ist, dass die Manipulation nicht auf den Versuchsleiter oder das System zurückzuführen ist, damit der eine Mikroweltbewohner davon überzeugt ist, dass sein Partner nicht seinem Ziel entsprechend gut genug steuert. Aus diesem Grund wurden vier Stufen der Manipulation getestet: 4%, 8%, 12% und 16% Reduzierung des Inputs jeweils einer der beiden Mikroweltbewohner. Der unbetroffene Mikroweltbewohner behält 50% seines Inputs auf die Joystickauslenkung. Es wird also keine Anpassung des Joystickinputs des unbetroffenen Mikroweltbewohners vorgenommen, so dass die Gesamtgeschwindigkeit ebenfalls reduziert wird.

GABELUNGEN In SAM 1.0 gab es schon Gabelungen, die aber nicht den gewünschten Effekt hatten. Eine Unterscheidung in eine länger und fehlerfreier in Hinsicht auf die Genauigkeit zu fahrende Abzweigung und einer kurzen aber schwereren Abzweigung wurde angestrebt. Die alten Gabelungen waren zu lang, so dass man den Verlauf nicht vollständig einsehen und die Entscheidung nicht auf dieser Basis getroffen werden konnte. Man hat sich also zufällig oder nach einem bestimmten Schema entschieden, z. B. immer links fahren. Dies wurde mit den neuen Gabelungen berücksichtigt und in vier Varianten getestet. Siehe dazu Abbildung 3.
Zusätzlich zu den oben genannten Anpassungen wurden noch diese vorgenommen:

GESCHWINDIGKEITSANZEIGE Eine weitere Erkenntnis aus der ersten Phase von ATEO war, dass die Geschwindigkeit im Vergleich zur Genauigkeit weniger präsent war. Das liegt unter anderem daran, dass es keine direkte Rückmeldung gibt. Fehler in der Genauigkeit können sofort bemerkt und als Fehler interpretiert werden, wenn man die Fahrbahn verlässt. Die Variation in der Zeit dagegen nicht, was dazu führt, dass die Genauigkeit stärker betont wird. Aus diesem Grund wurde in das Fahrobjekt eine Geschwindigkeitsanzeige integriert, welche genauso schnell Rückmeldung geben kann, wie das Verlassen der Strecke wahrgenommen wird.

KONFIGURATIONSMENU Das Konfigurationsmenu wurde in SAM 1.0 mit Hilfe von eToys erstellt, wodurch einige Nachteile entstehen. Zum einen wird eine Portierung von Squeak 3.7 auf Squeak 3.9, sowie generell alle Versuche den Quellcode in ein neues Squeak-Image



(a) eckig, lange Strecke links (ELl) (b) eckig, lange Strecke rechts (ELr)

(c) rund, lange Strecke links (ELl) (d) rund, lange Strecke rechts (ELr)

Abbildung 3: Gabelungen

einzulesen, unmöglich. Dies liegt daran, dass den einzelnen eToys Elementen eine im Squeak-Image eindeutige Identität zugewiesen wird. Beim Einlesen in ein neues Squeak-Image sind die Referenzen auf eToys Elemente nicht mehr korrekt und das Ausführen von Skripten unmöglich. Im späteren Verlauf des Projektes wurde das Einlesen von SAM Klassen in ein neues Image im Zuge von Performance-Untersuchungen wichtig. Ein Konfigurationsmenu wird jetzt durch Smalltalkklassen und -methoden erstellt, was im hohen Maße die Wartbarkeit und Erweiterbarkeit des Programmes verbessert.

INSTRUKTIONEN Auch die Instruktionen wurden in SAM 1.0 einmalig mit eToys erstellt und bei jedem nötigen Anzeigen als Morph von außen in Squeak hineingeladen. Dieses Vorgehen führte zu keinen Problemen beim Versionswechsel, aber das Anzeigen von Umlauten und dem 'ß' funktionierte anschließend nicht mehr. Das Anpassen der Instruktionstexte erschwert sich durch die externen Morphe, da die Texte in Squeak angepasst werden müssen. Eine Neuimplementierung ohne eToys konnte das Problem mit den Umlauten beheben und eine leichte Anpassung der Instruktionstexte außerhalb von Squeak auch ohne Smalltalkkenntnisse ermöglichen. Gleichzeitig konnte die Inkonsistenz der Größe und Position der Instruktionsfenster behoben und dadurch eine einheitliche Darstellung erreicht werden. Die Wartbarkeit und Erweiterbarkeit der Instruktionen konnte durch die Kapselung in eine eigene Klasse verbessert werden.

TASTATURSTEUERUNG Auf Grund der zahlreichen Erweiterungen von SAM musste auch die Tastatursteuerung angepasst werden. Weil diese nur zu Testzwecken implementiert wurde, ist die Fahrt auch nur für eine Person angelegt. Dies hat zur Folge, dass es keine Inputverteilung gibt und diese somit auch nicht manipuliert und getestet werden kannn. So wurde die Steuerung mit der Tastatur nur für die Hindernisse und die Geschwindigkeitsanzeige aktualisiert. Zusätzlich konnte die Möglichkeit rückwärts zu fahren oder über die durch die Joysticksteuerung definierte Maximalgeschwindigkeit hinaus zu beschleunigen entfernt werden.

FEHLERBESEITIGUNG Es wurden viele kleine Fehler entfernt und Verbesserungen, die die Komplexität erniedrigt und die Lesbarkeit verbessert haben, am Quellcode durchgeführt. Näheres hierzu ist im Folgenden nachzulesen oder dem Anhang A zu entnehmen.

3.2.2 *Implementierung*

In diesem Teil soll detailliert beschrieben werden, wie die zuvor genannten Erweiterungen implementiert wurden. Es werden aber lediglich die Änderungen dokumentiert, die nicht schon in Hildebrandt (2009) erfasst wurden. Auf Designentscheidungen wird ebenfalls eingegangen.

INSTRUKTIONEN Die Instruktionen sind in der Klasse ATEOInstruction implementiert. Der Konstruktor *initialize* erzeugt zwei Arrays, in die die 16 Anfangs- und Endinstruktionen aus Textdateien eingelesen und abgespeichert werden. Weiterhin wird das Instruktionsfenster erzeugt, welches aus folgenden Morphen zusammengestellt wird, wobei die Reihenfolge auch die Tiefe der Submorphe widerspiegelt, d.h. *button* und *instructionText* liegen in *window*, *window* wird von *panel* umschlossen:

- AlignmentMorph *panel*
- BorderedMorph *winodw*
- PluggableTextMorph *instructionText*
- SimpleButtonMorph *button*

Die Morphe *panel* und *window* sind nicht unbedingt nötig und wurden in SAM 2.0 entfernt. *button* besitzt zwei unterschiedliche Ereignisse *continueStart* und *continueEnd*. Das Ereignis *continueStart*, welches beim Drücken von *button* ausgelöst wird, versteckt das Instruktionsfenster und startet den Countdown, der zum Start des Versuchsschrittes führt. *continueEnd* wird auch beim Drücken von *button* ausgelöst und versteckt das Instruktionsfenster, leitet aber zusätzlich das Ende des Versuchsschrittes ein. Die beiden Methoden *showOnStart* und *showOnEnd* bilden das Interface zum Anzeigen der Instruktionsfenster mit den vom Versuchsschritt abhängigen Instruktionen. Hier wird auch das Ereignis des *buttons* festgelegt. Beide Methoden werden von der Versuchssteuerung ATEOVuSt aufgerufen.

Die Methode *panel* wird in der Versuchssteuerung ATEOVuSt gerufen, um Zugriff auf das Instruktionsfenster zu bekommen und dieses löschen zu können, wenn der Versuch beendet ist.

KONFIGURATIONSMENU In einer frühen Version des Konfigurationsmenüs wurde jedes einzelne Element des Menüs in ATEOConfig»intialize erstellt und in Instanzvariablen gespeichert. Dies führte

zu sehr vielen Quellcodezeilen mit nur sehr geringfügig verschiedenem Code. Eine Kapselung wurde nun vorgenommen und eine eigene Klasse für ein Menüelement definiert. Durch den Klassenkonstruktor *elementPosition: aPoint contentName: aString startValue: anInteger* kann ein durch Parameter personalisiertes Menüelement erstellt werden. Die Parameter werden an die Methode *position: aPoint contentName: aString startValue: anInteger* weitergereicht, die die eigentliche Erstellung des Morphes als privater Konstruktor vornimmt. Zwei Zugriffsmethoden machen es möglich auf den Wert, der durch das Menüelement eingestellt wird, zuzugreifen und das Objekt des Menüelementes selbst zu bekommen. Dies wird in dem Konstruktor von ATEOConfig, welcher die einzelnen Menüelemente erstellt und vorkonfiguriert, genutzt, um die einzelnen Morphe in einem Morph zusammenzufassen. Außerdem belegt der Konstruktor einige der globalen Variablen mit nil, damit sie dem Squeak-Image bekannt gemacht werden und somit ohne Schwierigkeiten auch in einer frischen Squeak-Version nach dem Laden der ATEO Klassen sofort benutzt werden können. Ohne diese Maßnahme würde es immer zu Fehlern kommen und diese globalen Variablen müssten manuell im Squeak-Image bekannt gemacht werden. Die restlichen globalen Variablen werden in der Klasse ATEOVuSt vorbelegt. Eine Ausnahme bildet die globale Variable Configure. Sie ist die einzige globale Variable in SAM 1.5, die mit einem Objekt und nicht mit nil initialisiert wird. Dies geschieht mit Hilfe eines Buttons, der mit der Klassenmethode *begin* erzeugt wird. Im Anschluss kann jederzeit durch Drücken dieses Buttons ein neues Konfigurationsmenü erzeugt werden.

HINDERNISSE Die statischen Hindernisse des Slaloms werden in den beiden Methoden *slalom25: variation* und *slalom50: variation* erzeugt. Der Parameter *variation* gibt an, ob das erste Hindernis links oder rechts der Straße positioniert wird. Das Erstellen der Bildmorphe geschieht im Konstruktor der Ablaufsteuerung. Dabei werden vier unterschiedlich gefärbte Blöcke geladen, die sich farblich aber so ähnlich sind, dass der Unterschied von den Mikroweltbewohnern nicht bemerkt wird. Die verschiedenen Farben sind notwendig, um unterscheiden zu können, mit welchem Hindernis kollidiert wurde. Die Farben im RBG Code sind in Tabelle 1 gelistet. Bei einer Kollision soll auch nur das entsprechende Hindernis verschwinden, es besteht also die Möglichkeit, mit beiden Hindernissen eines Slaloms unmittelbar hintereinander zu kollidieren. Dies wird in der für die Kollision zuständigen Methode durch den Parameter *obstacleNo* unterschieden. Die Bestrafung der Mikroweltbewohner in Zeit- und Genauigkeit wurde neu programmiert und die Zeitstrafe ist jetzt millisekundengenau einstellbar. Eine erste Implementierung sah die Realisierung einer Verzögerung mit einer leeren Programmschleife vor, was zu keiner eindeutig einstellbaren Zeit geführt hat. Jetzt wird mit Hilfe der Delay Klasse eine Zeit von einer Sekunde definiert und im Anschluss gewartet. Zusätzlich wird das Fahrobjekt mit Hilfe der Sensoren neben die Fahrbahn auf das Grün verschoben.
Die Kollision wird bei Kontakt mit dem Hindernis von den Sensoren ausgelöst und in der Methode *feedback* der Klasse ATEOCarNoFB aufgerufen. In der *feedback* Methode wird anhand der Farbe der Hindernisse genau unterschieden, mit welchem Hindernis kollidiert wurde und zusätzlich noch der Sensor, der die Kollision gemeldet hat, in *collisionGrade* festgehalten. Diese Information wird auch am Ende des Morphsteps in die Logdatei geschrieben.

Hindernis	Farbe (RBG)
block1-1 (erstes Hindernis bei 25/50rl)	R: 225 G: 96 B: 65
block1-2 (erstes Hindernis bei 25/50lr)	R: 225 G: 96 B: 65
block1-3 (zweites Hindernis bei 25/50rl)	R: 200 G: 97 B: 66
block1-4 (zweites Hindernis bei 25/50lr)	R: 200 G: 97 B: 66

Tabelle 1: RGB Codes der statischen Hindernisse

Die Konfiguration des Erscheinens von Hindernissen auf bestimmten Streckenbildern geschieht immer noch in den Textdateien der Strecke, wo an den gewünschten Stellen Zahlen eingefügt werden können, die in der Methode complexity der Klasse ATEOPar Booleanvariablen in ATEOAblaufsteuerung setzt. Diese Variablen werden dann in jedem Morphstep in ATEOAblaufsteuerung»makeItComplex ausgewertet und entsprechende Methoden für die Erstellung der Hindernisse aufgerufen. Bothe u. a. (2008, Seite 32) listet in der Verhaltensspezifikation zur SAM Version 1.5 alle möglichen Zahlen und die damit verbundene Aktivierung der Erweiterungen auf.

ANPASSUNG DES LOGFILES Da die Position der vier möglichen statischen Hindernisse im Logfile festgehalten werden soll, wurden einige Zugriffsmethoden in ATEOAblaufsteuerung hinzugefügt. In ATEOLog»prepareLogEntry wird dann die Position der einzelnen Hindernisse ermittelt und gleichzeitig die erste Sichtung der Hindernisse für die Mikroweltbewohner festgehalten. Die Zugriffsmethoden werden auch in ATEOPar»bewegen benutzt, um die Variable *trackedit* zu setzen. *trackedit* soll in dem Moment, in dem eine Erweiterung der Strecke sichtbar wird, in das Logfile schreiben, welche Erweiterung vorliegt.

Dazu muss für die Hindernisse festgestellt werden, welche Variation aktuell ist, dies geschieht über die Booleanvariablen *static25rl*, *static25lr*, *static50rl*, *static50lr* und *dynamicBuild*. Zusätzlich muss mit Hilfe der Hindernisse selbst festgestellt werden, ab wann sie sichtbar sind. Weil in SAM immer zwei Bilder gleichzeitig geladen werden und in Squeak im Bildschirm oben links die Koordinate 0/0 definiert ist, kommt es dazu, dass auf Streckenbildern nach oben hin außerhalb des Bildschirmes negative Positionskoordinaten vorliegen. Sind die Positionskoordinaten bei der Abfrage positiv sind die Hindernisse sichtbar und die Methode *trackedit:* aus ATEOLog bekommt eine Fließkommazahl übergeben. Diese Fließkommazahl bestimmt, welches bezeichnende Kürzel in die Logdatei geschrieben werden kann.

Die Gabelungen werden anhand der Farbkodierung am Streckenbildrand identifiziert und hier beginnt das Schreiben der Logdatei mit dem Befahren des Streckenbildes.

Bei der Inputmanipulation wird mit der Erstellung eines Inputmanipulationsobjektes die aktuelle Reduzierung durch *trackedit:* übermittelt. Tabelle 2 listet alle Fließkommazahlen und die dazugehörigen Kürzel auf.

ANPASSUNG DER TASTATURSTEUERUNG Die Anpassung der Tastatursteuerung wurde in ATEOAblaufsteuerung»incSchrittweite vorgenommen. Dies hatte zwei Konsequenzen. Zum einen wurde dadurch sichergestellt, dass die durch die Joysticksteuerung definierte Maxi-

TRACKEDIT (PROGRAMM)	TRACKEDIT (LOGFILE)	BEDEUTUNG
1.1	1_RLl	runde Gabelung links
1.2	1_RLr	runde Gabelung rechts
1.3	1_ELl	eckige Gabelung links
1.4	1_ELr	eckige Gabelung rechts
0.04	2_4	Inputmanipulation 4%
0.08	2_8	Inputmanipulation 8%
0.12	2_12	Inputmanipulation 12%
0.16	2_16	Inputmanipulation 16%
3.0	3	dynamisches Hindernis
4.1	4_25lr	statisches Hindernis 25% rechts links
4.2	4_25rl	statisches Hindernis 25% links rechts
4.3	4_50lr	statisches Hindernis 50% rechts links
4.4	4_50rl	statisches Hindernis 50% links rechts

Tabelle 2: Trackedit

malgeschwindigkeit von 20,48 Pixel pro Morphstep nicht überschritten werden kann. Zum anderen wurde auch sichergestellt, dass die Geschwindigkeit nicht negativ werden kann, was in diesem Fall bedeutete, dass das Fahrobjekt rückwärts fährt. Die Folge war, dass keine Streckenbilder mehr befahren werden, weil neue Bilder nur beim Vorwärtsfahren geladen werden und es zu einem Fehler kommt. Als letzte Anpassung wurde die Geschwindigkeitsanzeige integriert. Die Anzeige ist aber ungenau, da die Beschleunigung und das Abbremsen pro Tastendruck eine Veränderung von +1 bzw. -1 bedeutet. Damit kann die Maximalgeschwindigkeit von 20.48 nicht erreicht werden. Da die Tastatursteuerung aber nur zu Testzwecken benutzt wird, wenn keine Joysticks zur Verfügung stehen, kann diese Ungenauigkeit vernachlässigt werden.

FEHLERBESEITIGUNG Während der Arbeiten an SAM und durch das wiederholte Testen der Software konnten einige Fehler und nicht mehr verwendete Codezeilen entfernt werden. Es soll nur kurz dokumentiert werden, welche diese waren und wie sich das Verhalten von SAM geändert hat. Näheres dazu liefert der Anhang A.

1. Es wurde behoben, dass bei einer Steuerung per Joystick sowie per Tastatur die Möglichkeit bestand, am Ziel vorbei zu fahren und dadurch der Versuchsschritt nicht beendet werden konnte. Nur ein genaues Einfahren ins Ziel hatte das Ende des Steps zur Folge. Weil nicht angenommen werden kann, dass bei den psychologischen Untersuchungen im Labor die beiden Mikroweltbewohner immer genau das Ziel treffen, wurde das Ende des Versuchsteps anhand eines neuen Farbcodes am linken Rand des Streckenbildes neu definiert und unabhängig von der Farberkennung des Ziels gemacht.

2. Der Algorithmus zur Berechnung der Zeitstrafe wurde geändert und ist nun genauer einstellbar. Eine erste Umsetzungsvariante beinhaltete eine Verzögerung des Programmablaufes durch eine Schleife ohne Funktion. Dieser Ansatz ist unelegant, ungenau und von der benutzten Hardware abhängig. Dies zu verbessern ist in SAM 1.5 gelungen und weiter oben beschrieben.

3. Die drei globalen Variablen EventSound, EventPopup und Balance wurden als Instanzvariablen von ATEOTastaturabfragen neu definiert. Da auf diese Variablen von anderen Klassen aus zugegriffen werden muss, wurden Zugriffsmethoden hinzugefügt. Die ehemals versteckten Beziehungen zwischen Klassen sind dadurch deutlicher erkennbar.

4. Im alten Konfigurationsmenü gab es einen Button, mit dem man das Morphstepping einstellen konnte. Dieser Button war später ohne Funktion und wurde deshalb nicht in das neue Menü integriert. In ATEOVuSt ist aber noch eine Abfrage des einstellbaren Wertes übrig geblieben, die ebenfalls unnötig war. Diese wurde entfernt, um den Quellcode übersichtlicher zu gestalten.

5. Aus Verständnisgründen wurde die Formel zur Berechnung der Geschwindigkeit aus den Joystickeingaben umgeformt und durch temporäre Variabeln, welche Teilergebnisse zwischenspeichern, übersichtlicher dargestellt. Die alte Formel kann in Bothe und Hildebrandt (2008) nachgelesen werden, die neue in Bothe u. a. (2008).

3.3 VERSION 2: SAM 2.0

3.3.1 *Konzept*

In den Voruntersuchungen im Labor zum Einfluss der Erweiterungen auf das Konfliktpotenzial wurde SAM 1.5 aufgrund der Erkenntnisse angepasst. Die Inputmanipulation hat sich als ineffektiv herausgestellt und wurde aus dem Quellcode entfernt. In den Textdateien der Streckenzusammenstellung wurden weiterhin die als effektiv, im Sinne einer erhöhten Variabilität im Verhalten der Mikroweltbewohner, erkannten Gabelungen (ELr und RLl) und Hindernisse (25lr und 50lr) in die kooperativen Versuchsschritte eingefügt. Nun auch, wie bei der Version SAM Solo Tracking, in kombinierter Weise. Bis auf die ersten vier Versuchssteps wurden alle weiteren Versuchssteps auf kooperatives Steuern umgestellt und ein 17. Versuchsstep mit dreifacher Länge definiert, um für spätere Untersuchungen Hinweise darauf zu erhalten, welchen Einfluss längere Strecken auf die Leistung der Mikroweltbewohner haben. Damit wird die bekannte Strecke nun dreimal hintereinander gefahren. Alle Änderungen spiegeln sich auch in aktualisierten Instruktionen wieder.

In den Voruntersuchungen wurde von den beiden Versuchspersonen ein Umschlag von fünf, in dem die vermeintliche Teaminstruktion steckte, ausgewählt. Tatsächlich haben sich die Umschläge aber nicht voneinander unterschieden und darüber hinaus gab es je Umschlagnummer zwei für den Mikroweltbewohner nicht unterscheidbare Varienten: eine enthielt eine Instruktion eher genau und eine Instruktion eher schnell zu fahren - diese Instruktionen wurden entsprechend auf die MWB verteilt. Dieses Vorgehen wurde derart gestaltet, um von den

gegensätzlichen Instruktionen schnell oder genau zu fahren abzulenken. Dieser Medienbruch zwischen Papier- und Bildschirminstruktion hat wahrscheinlich dazu geführt, dass die Versuchspersonen misstrauischer geworden sind und die gegensätzliche Instruierung erkannt haben. Eine Integration der Instruktionswahl in die Bildschirminstruktionen am Anfang des Versuches wurde vorgenommen.

Eine kleine Änderung wurde auch bei der Geschwindigkeitsmessung des dynamischen Hindernisses vorgenommen. Die Messung erfolgt nun einmalig auf dem Streckenbild, auf dem das dynamischen Hindernis auch erscheint, aber in jedem kooperativen Versuchsstep. Das bedeutet also während die Mikroweltbewohner sich auf ein dynamisches Hindernis einstellen und das Streckenbild befahren, wird gleichzeitig eine Geschwindigkeitsmessung für das Hindernis des nächsten Steps vorgenommen. Diese Änderung ist notwendig, um die stetige Verbesserung der Mikroweltbewohner im Umgang mit der Steuerung in späteren Versuchssteps auch auf die Geschwindigkeit des dynamischen Hindernisses übertragen zu können. Da auf zwei der drei in SAM 1.5 genutzten Streckenbilder nun auch statische Hindernisse zu finden sind, die das Fahrverhalten zu sehr beeinflussen, wurde die Messung auf das eine Streckenbild mit dynamischem Hindernis reduziert. Im ersten kooperativen Versuchsstep mit Hindernissen wird eine mit 50% der Maximalgeschwindigkeit definierte Geschwindigkeit für das erste dynamische Hindernis festgelegt, da zuvor keine repräsentative Messung vorgenommen werden kann.

Weitere Änderungen beziehen sich nur noch auf die Strukturierung des Quellcodes und des Entfernens toten Quellcodes und werden im nächsten Abschnitt und im Anhang A dokumentiert.

3.3.2 *Implementierung*

INSTRUKTIONEN Die erste Instruktion, die den Mikroweltbewohnern angezeigt wird, sind fünf Buttons für die Auswahl der Teaminstruktion. Was vorher mit Umschlägen realisiert wurde, ist nun in das Programm integriert. Die fünf Buttons haben keine Auswirkung auf das System, es wird nur zur nächsten Instruktion weitergeschaltet. Implementiert werden diese Buttons im Konstruktor der Klasse ATEOInstruction als IconicButtons, damit ein Bild mit der Beschriftung integriert werden kann. Dies hat den Vorteil, dass wieder außerhalb von Squeak der Text individuell und mit beliebiger Formatierung angepasst werden kann, so z.B. auch farbig oder mit beliebigen Schriftarten und -größen. Jeder Button hat *start* als Ereignis und diese Methode löscht die fünf Buttons, erstellt den grünen 'Weiter' Button mit *continueStart* als Ereignis und zeigt die erste Textinstruktion an. Am Ende wird eine neue Variable aus ATEOVuSt, die als Zeiger in das Instruktionsarray genutzt wird, inkrementiert.

Diese Variable ist in ATEOVuSt integriert, da es bereits eine globale Variable für den Zugriff gibt und die Anpassung dadurch verringert wird. Ein Zugriff ist nicht zu vermeiden, da in ATEOVuSt *showOnStart* für das Anzeigen der Startinstruktionen aufgerufen wird. *showOnStart* wurde auch angepasst, da als dritte und sechste Instruktion die gegensätzliche Instruktion für die Mikroweltbewohner angezeigt werden soll. Dazu wird zur richtigen Zeit, also wenn instructionIndex 3 oder 6 ist, das Ereignis des Buttons von *continueStart* auf *showTeamInstruction* geändert. *showTeamInstruction* lädt die Instruktion als Bild und zeigt sie mittig im Instruktionsfenster an. Die Ereignismethode für den Button

STEP NR.	MWB	INDEX	ACTIONSELECTOR	INSTRUKTION
1	1	1	start	Teaminstruktion Buttons 1-5
1	1	2	continueStart	Instruktion 1
2	1	3	showTeamInstruction	Instruktion 2
2	1	4	continueStart	Schnelligkeits-instruktion
3	2	5	continueStart	Instruktion 3
4	2	6	showTeamInstruction	Instruktion 4
4	2	7	continueStart	Genauigkeits-instruktion
5	1+2	8	continueStart	Instruktion 5
6	1+2	9	continueStart	Instruktion 6
7	1+2	10	continueStart	Instruktion 7
8	1+2	11	continueStart	Instruktion 8
9	1+2	12	continueStart	Instruktion 9
10	1+2	13	continueStart	Instruktion 10
11	1+2	14	continueStart	Instruktion 11
12	1+2	15	continueStart	Instruktion 12
13	1+2	16	continueStart	Instruktion 13
14	1+2	17	continueStart	Instruktion 14
15	1+2	18	continueStart	Instruktion 15
16	1+2	19	continueStart	Instruktion 16
17	1+2	20	continueStart	Instruktion 17

Tabelle 3: Abfolge der Startinstruktionen

wird im Anschluss wieder auf *continueStart* gesetzt. Der Instruktionszeiger wurde notwendig, da die Anzahl der Instruktionen nicht mehr der Anzahl von Versuchssteps entspricht, sondern drei Instruktionen mehr existieren. Dies wurde auch bei dem Startinstruktionsarray angepasst. Bei den Endinstruktionen hat sich nur der Instruktionstext geändert, nicht aber die Anzahl der Fenster, weswegen hier keine Anpassung vorgenommen werden musste. Dieser Ablauf ist in Tabelle 3 noch einmal tabellarisch für einen Versuchsstep dargestellt, in dem über das Konfigurationsmenü vor dem Versuch eingestellt wurde, dass beispielsweise Mikroweltbewohner 1 startet. Eine kleine Änderung hat sich noch im Aufbau der Instruktionsfenster ergeben; diese sind jetzt nur noch ein Morph und nicht wie zuvor vier. Dadurch ergibt sich ein übersichtlicherer Quellcode.

GESCHWINDIGKEITSMESSUNG Um die Aktivierung der Geschwindigkeitsmessung auf nur ein Streckenbild zu reduzieren, wurde die Bilddatei namens SbQx260806.gif ohne die Farbcodes für die Aktivierung der Messung als neue Bilddatei SbQ2x260806.gif gespeichert und in der Textdatei BilderExp3BaJe.txt am Ende hinzugefügt. Dieses neue

Streckenbild muss nun in der Streckenkonfiguration an den Stellen mit statischen Hindernissen eingefügt werden. Zusätzlich wurde in endStep die Mittlung der drei Messungen entfernt.

Die Geschwindigkeit des dynamischen Hindernisses wird auch in der Logdatei festgehalten, und zwar in der Spalte mit dem Namen SpeedDynamicObstacle.

In ATEOPar»initialize wird die Geschwindigkeit des ersten dynamischen Hindernisses in cumulativeSpeed mit 10.24 Pixel pro Morphstep festgelegt. In ATEOPar»bewegen wurde beim Stopp der Messung noch die kumulative Aufaddierung der alten gemessenen Geschwindigkeiten mit der neuen Geschwindigkeit entfernt.

VERBESSERUNGEN Zahlreiche Verbesserungen wurden durchgeführt, wobei zwei im Projektseminar Mensch-Technik-Interaktion in Echtzeit des Lehrstuhles Softwaretechnik an der Humboldt Universität zu Berlin (Website: Projektseminar letzter Zugriff: 02.07.09) von Studenten vorgeschlagen bzw. als Ungenauigkeit aufgezeigt wurden. Im Folgenden sollen die Verbesserungen kurz erläutert werden.

1. Die drei Methoden *ende, quit* und *streckenReset* aus ATEOVuSt wurden vereint in *finish: mode*. Der Parameter *mode* gibt hierbei an, ob der Versuch beendet (früher *quit*), der Versuchsschritt beendet (früher *ende*) oder der Versuchsschritt neu gestartet werden soll (früher *streckenReset*). Die Methode *fertig* aus ATEOVuSt wurde entfernt und der Quellcode in ATEOAblaufsteuerung»endStep übertragen. *endStep* zeigt nach dem Stoppen des Morphsteppings die Endinstruktion an, während das Drücken des Buttons im Instruktionsfenster wieder *finish: mode* aufruft. Durch diese Neustrukturierung werden verwandte Methoden mit sehr ähnlichen Methodennamen in einer Methode vereint und das Verständnis des Quellcodes weiter verbessert.

2. Die Definition der globalen Variablen wurde zuvor an zwei unterschiedlichen Stellen im Quellcode durchgeführt, nämlich in ATEOVuSt»start und in ATEOConfig»initialize. Diese Definitionen wurden alle nach ATEOConfig»go verschoben, weil einige der globalen Variablen mit Werten aus dem Konfigurationsmenu belegt werden und Definition und Zuweisung der Werte aus dem Konfigurationsmenü gleichzeitig vorgenommen werden kann. Vierzehn ungenutzte Initialisierungen globaler Variablen wurden entfernt, im Einzelnen sind das Vu2Sticks, VuStickLR, VuStickUD, VuShowSticks, LogObj, Logfile, LogfileWriter, RootObj, JoyID2, Instrmorph, Balance, EventSound, EventPopup und Credit. Die Belegung der globalen Variablen mit Objekten oder konstanten Werten erfolgt ebenfalls an dieser Stelle.

3. Da die Methode ATEOVuSt»start durch das Entfernen der globalen Variablen selbst sehr kurz geworden ist, wurde die Methode conf entfernt und die Funktionalität dort eingefügt.

4. Die Umrechnung der Geschwindigkeit für die Geschwindigkeitsanzeige war fehlerhaft und ungenau und wurde nach einem Hinweis im Seminar verbessert. Die aktuelle Geschwindigkeit wird nun nicht mehr grob mit fünf multipliziert, was bei 5 * 20,48 einen Wert von über 100 ergibt, sondern genau in Prozent umgerechnet. Angepasst wurde dies für die Steuerung mit Joystick und Tastatur.

5. In den Klassen ATEOConfig, ATEOInstruction, ATEOLog, ATEOPar, ATEOTastaturabfragen, ATEOVuSt, ATEOrealJoysticksStepping und ATEOAblaufsteuerung wurden einige nicht benutzte Instanzvariablen sowie Methoden entfernt. *posY* aus ATEOrealJoysticksStepping wurde als temporäre Variable in der Methode step umdefiniert. Genaue Auflistungen können im Anhang A nachgelesen werden.

3.4 VERSION 3: SAM SOLO TRACKING

Der Unterschied zwischen SAM 1.5 und SAM Solo Tracking liegt in der erhöhten Maximalgeschwindigkeit in fünf kooperativen Versuchssteps. Hier wurde ein Faktor in der Berechnung der Geschwindigkeit aus den Joystickeingaben modifiziert. Der Normalwert, so wie er in SAM 1.5 eingestellt ist, ist 50. Ein höherer Wert bedeutet, dass die maximal mögliche Geschwindigkeit niedriger wird, also die Bilder langsamer bewegt werden. Ein niedriger Wert bedeutet genau das Gegenteil. Für SAM Solo Tracking wurden vier Stufen definiert, und zwar die Faktoren 18, 26, 34 und 42. In dieser Reihenfolge als absteigende Geschwindigkeit sind diese Faktoren in ATEOrealJoysticksStepping»initialize in einem Array definiert.

In SAM Solo Tracking gibt es insgesamt neun Versuchssteps, so dass in den Versuchssteps, wo keine Anpassung der Maximalgeschwindigkeit vorgenommen werden soll, der Faktor 50 in das Array eingetragen wurde. Im vierten Versuchstep wird beim Faktor 18 die Geschwindigkeitsmessung für das dynamische Hindernis vorgenommen. In den darauf folgenden Versuchsteps wurden Gabelungen und Hindernisse bei veränderter Maximalgeschwindigkeit in den Ablauf hinzugenommen, um eine hohe Komplexität der Strecke zu erreichen. Der letzte Versuchstep ist dann wieder ohne Manipulation. Ein zweites Array mit den um ein Zehntel kleineren Faktoren wird für die Geschwindigkeitsanzeige benötigt. Auf beide Arrays wird in ATEOrealJoysticksStepping»step zugegriffen und die entsprechende Anpassung vorgenommen. Der Zeiger in das Array wird von der Versuchssteuerung in ATEOVuSt»nextStep erhöht. Damit wird sichergestellt, dass vor jedem Versuchstep dieser Zeiger erhöht und der richtige Faktor ausgewählt wird. Dafür wurde eine Zugriffsmethode in ATEOVuSt geschrieben.

Die Konfiguration der einzelnen Versuchsteps in Bezug auf Hindernisse und Gabelungen wurde in den Textdateien für die Strecken angepasst. Das Experiment wird nur von einer Versuchsperson aber mit beiden Joysticks gleichzeitig absolviert. Es wurde und kann also bei Ausfall einer Versuchsperson für das Experiment SAM 1.5 oder SAM 2.0 auf SAM Solo Tracking zurückgegriffen werden. Gleichzeitig kann eine mögliche Komplexitätserhöhung durch einen erhöhten Schwierigkeitsgrad für eine eventuelle Integration getestet werden.

3.5 DIE SOFTWAREARCHITEKTUR VON SAM 1.5 UND SAM 2.0

Zwei wichtige Kriterien für gute objektorientierte Softwarearchitekturen sind die Koppelung und die Kohäsion. Unter Kopplung versteht man die Schnittstellenbeziehung zwischen den Komponenten. Hier wird eine möglichst lose Kopplung angestrebt, d.h. die betrachtete Klasse besitzt nur wenige oder gar keine Abhängigkeiten zu anderen Klassen und es kommt vor allem nicht zu zyklischen Verbindungen zwischen diesen. Vorteile einer losen Koppelung sind die leichte Wiederverwend-

barkeit in anderen Softwarearchitekturen, die leichte Modifizierbarkeit durch beispielsweise Austauschen der Komponente sowie einfache Wartbarkeit, da die Klasse leichter zu verstehen ist, wenn sie nicht mit vielen Klassen der Architektur verbunden ist.

Mit Kohäsion beschreibt man die logische Einteilung der Aufgaben oder Programmteile in Komponenten. Das Ziel hier ist, die Klasse für genau eine Problemlösung zu entwerfen und keine Klasse mehr als eine unterschiedliche Aufgabe im System übernehmen zu lassen. Die Aufgabenzerteilung kann man ähnlich wie bei der hierarchischen Aufgabenanalyse bis zu elementarsten Aufgaben fortsetzen. Dies führt zu leicht verständlichen Klassen, die gut wartbar und modifizierbar sind. Es wird demnach eine starke Kohäsion angestrebt.

Problematisch wird es eine lose Koppelung und eine starke Kohäsion gleichzeitig zu verwirklichen, da beide Forderungen gegeneinander arbeiten. Definiert man viele Komponenten ist die Kohäsion stark aber es kommt zu einer hohen Koppelung, da die vielen einzelnen Komponenten viele Verbindungen zu anderen Komponenten pflegen müssen. Entwirft man eine Architektur mit nur wenigen Klassen, kann eine lose Koppelung erreicht werden, da die Interaktionen zwischen Klassen nun intern stattfinden, was effizienter im Programmablauf aber wenig förderlich für eine starke Kohäsion ist. Es kann davon ausgegangen werden, dass wenige Klassen jeweils mehr als eine Aufgabe übernehmen, wenn es sich nicht um ein sehr kleines System handelt.

In Abbildung 4 und Abbildung 5 sind die Klassendiagramme von SAM 1.5 bzw. SAM 2.0 dargestellt. Man sieht viele Verbindungen zwischen den einzelnen Klassen, so dass man schließen kann, dass die Koppelung eng ist, nur wenige Klassen können leicht entfernt werden, weil sie nur wenige Verbindungen zu vereinzelten Klassen pflegen. Die Kohäsion ist besser, da die meisten Klassen jeweils eine Aufgabe übernehmen. Trotzdem sind die Klassen zum Teil komplex und unübersichtlich, so dass man über eine bessere Verteilung der Funktionalität die Kohäsion nicht verschlechtert und im selben Zuge durch eine losere Koppelung die Architektur verbessern könnte.

Durch die Erweiterungen und Modifikationen hat sich die Architektur von SAM 2.0 verschlechtert. Dies kommt in der angestiegenden Zahl von Zyklen zum Ausdruck. Die Begründung hierfür ist in der schlechten Architektur von SAM 1.0 und SAM 1.5 zu finden. Alle Erweiterungen und Anpassungen wurden in eine schlechte Architektur integriert. Die Beziehungen zwischen den Klassen sind dadurch schlecht geblieben und neue Zyklen wurden geschlossen. Die Folge ist, dass man sich gut im Programm auskennen muss, um Anpassungen an den richtigen Stellen vornehmen zu können. Die Modifizierbarkeit sowie die Änderbarkeit von SAM sind schlecht. Es wurden einige Verbesserungen im Quellcode vorgenommen, um die allgemeine Verständlichkeit zu verbessern und die Einarbeitungszeit generell zu verkürzen. Die Qualität der Softwarearchitektur konnte damit aber nicht erhöht werden. Das Entfernen von totem Code steigert auch nur die Verständlichkeit, hatte aber keinen Einfluss auf die Architektur, da diese unbenutzten Methoden die Beziehungen zwischen den Klassen nicht mehr beeinflusst hatten.

Die Analysierbarkeit ist folglich nicht einfach, der Systembrowser von Squeak und die Möglichkeit Methodenrufe nachzuvollziehen machen es aber leichter.

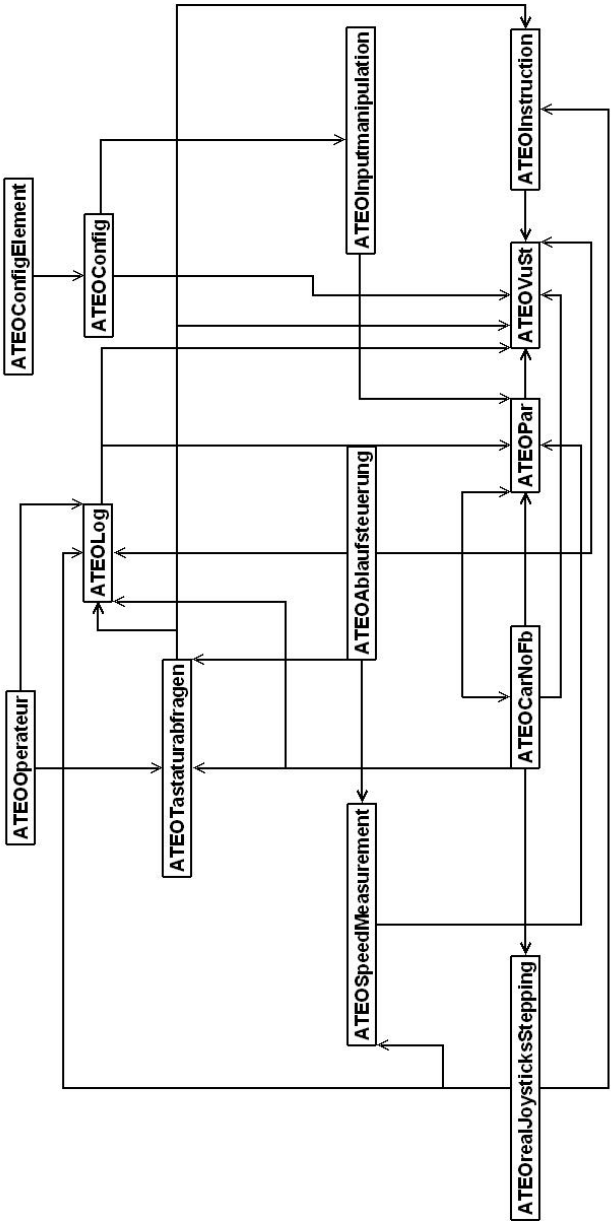


Abbildung 4: SAM 1.5 Klassendiagramm

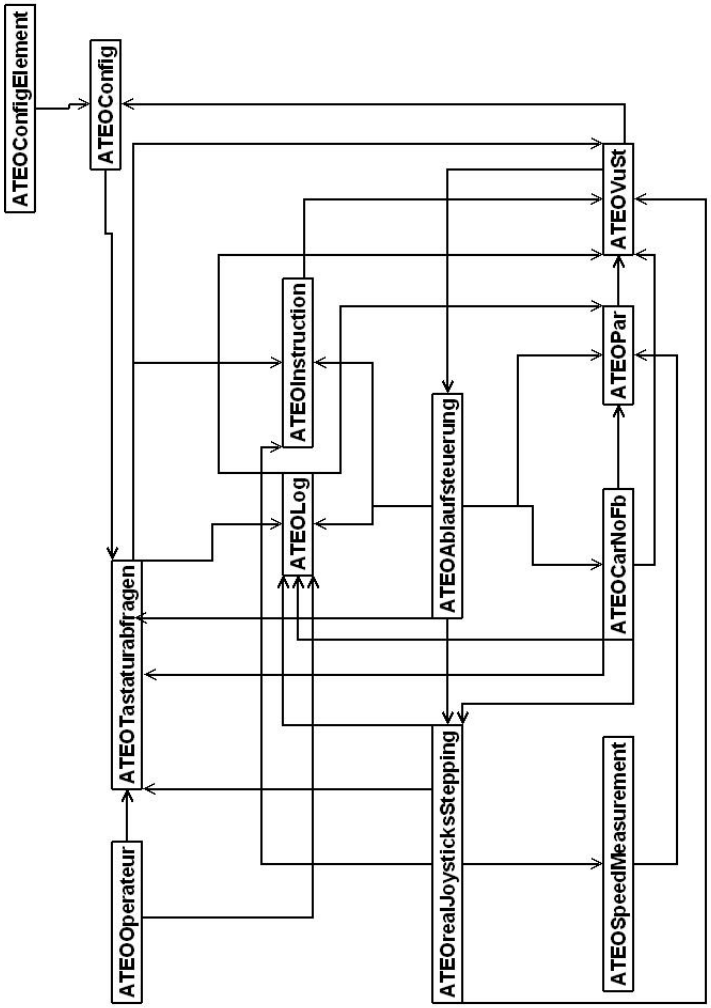


Abbildung 5: SAM 2.0 Klassendiagramm

4 VERNETZUNG: KONZEPTION UND IMPLEMENTIERUNG

4.1 BEGRÜNDUNG DER VERNETZUNG

Ein Kriterium für eine Vernetzung der einzelnen Komponenten ist die Performanz. Wenn von Performanz gesprochen wird, ist im SAM Kontext immer gemeint, wie viel Zeit zwischen zwei Morphsteps liegt. In der Logdatei kann man dies in der Spalte TimeDelta ablesen. Optimal und im Quellocde auch so vorgesehen ist eine Zeit von 30ms. Diese Stepping-Frequenz von 30ms wird allerdings nicht auf den Rechnern des ATEO Projektes erreicht, eine Zeit von 30-49ms ist hier noch akzeptabel. Begründet ist das in den Verzögerungen in der Berechnung der einzelnen Schritte. Hier wird die Joystickumsetzung aus den Eingaben berechnet und dafür genutzt die Streckenbilder nach unten zu verschieben. Dieser Vorgang des Verschiebens, so wie er im Moment implementiert ist, ist sehr rechenintensiv in Squeak und wird nur von sehr rechenstarken Computern in aktzeptabler Zeit bewältigt. Akzeptabel ist hierbei eine Zeit im Bereich von 30-49ms. Können diese Zeiten eingehalten werden, sind keine Verzögerungen beim Verschieben der Bilder bemerkbar. Mit höheren Werten kommt es nicht nur zu einem subjektiv ruckeligen Bewegen der Bilder, auch das Fahrverhalten ändert sich, da die Joystickeingaben schwerfälliger werden. Dies ist eine direkte Folge aus den längeren Zeiten für einen Morphstep. Die Steuerung reagiert weniger fein auf die Eingaben der Mikroweltbewohner, da diese Eingaben weniger genau in die Berechnung einfließen. Diese Schwäche des Systems liegt einzig in Squeak begründet. Versuche mit anderen Programmiersprachen wie Java haben gezeigt, dass die Hardware mehr als ausreichend ist. Es gibt bereits Überlegungen das Verschieben der Bilder effizienter zu gestalten. Ein Beispiel ist die Berechnung der Strecke mit Hilfe von Bezierkurven während der Fahrt. Berechnet wird dann auch nur noch die Straße, welche vor einem grünen Hintergrund bewegt wird. Dabei muss eine neue Möglichkeit gefunden werden, streckenteilspezifische Ereignisse auszulösen, da die Farbcodes am linken Rand der Bilder wegfallen würden. Eine sich über mehrere Textdateien erstreckende Konfiguration der Strecke kann im Zuge dessen ebenfalls optimiert werden.

Die Performanz ohne Programmieraufwand zu verbessern, ist in manchen Fällen ein neues Squeak-Image zu erzeugen. Wie groß der Effekt auf die Performanz dabei ausfällt, hängt davon ab, wieviel Änderungen im Squeak-image abgespeichert wurden. Eine Vermutung der Performanzverschlechterung hängt nämlich mit der changes-Datei des Squeak-Systems zusammen. In dieser changes-Datei werden alle Änderungen im System aufgezeichnet, damit sie später wieder rückgängig gemacht werden können. Je größer diese Datei wird, desto schlechter scheint die Performanz zu werden. Durch das Neuerstellen eines frischen Squeak-Systems wird diese changes-Datei auch wieder zurückgesetzt. Dieser Einfluss sowie mögliche andere Effekte können durch dieses Vorgehen ausgeschlossen werden

Wie ein neues Squeak-System erstellt werden kann und auf was dabei geachtet werden sollte, wird im Anhang D beschrieben.

4.1.1 *SAM - Operateursarbeitsplatz*

Der Arbeitsplatz für den Operateur bietet ihm die Möglichkeit, den laufenden Prozess (SAM) zu überwachen und bei Bedarf einzugreifen. Die Eingriffe sind hierbei in harte und weiche Eingriffe zu differenzieren. Unter harten Eingriffen versteht man die Eingriffe, die direkt in das Verhalten der steuernden Mikroweltbewohner eingreifen. So z.B. ist es dem Operateur möglich, die maximale vertikale Joystickumsetzung auf einer Skala von 0-120% zu beinflussen und damit die Gesamtgeschwindigkeit des Prozesses. Ein weiterer harter Eingriff ist die Möglichkeit, die horizontale Joystickumsetzung zu blockieren und damit das Fahren in eine bestimmte Richtung zu erzwingen, so z.B. an Gabelungen. Als weiche Eingriffe sind dem Operateur auditive und visuelle Hinweise zur Hand gegeben, damit den steuernden Mikroweltbewohnern Hilfestellungen in Hinblick auf ihr gemeinsames Ziel, die schnelle und fehlerfreie Bewältigung der Strecke, gegeben werden können. Außerdem werden dem Operateur verschiedene Möglichkeiten der Überwachung geboten. Zum einen durch ein Videobild von den beiden Mikroweltbewohnern und ihrer Joystickauslenkung. Zum anderen durch eine Streckenansicht, welche den für die Mikroweltbewohner sichtbaren Bereich darstellt aber auch die vor ihnen liegende Strecke, so dass durch diese Vorausschau Probleme vom Operateur antizipiert werden können. Auch zeigt die Streckenansicht an, wie sich die Mikroweltbewohner in Bezug auf Genauigkeit und Geschwindigkeit in der Vergangenheit verhalten haben. Dies wird duch einen Schweif angezeigt, der seine Farbe gemäß der gefahrenden Geschwindigkeit anpasst. Detaillierte Informationen dazu können unter Schwarz (2009) nachgelesen werden.

Die Vernetzung von SAM und Operateursarbeitsplatz ist inhaltlich aus zweierlei Hinsicht sinnvoll. Im Rahmen des ATEO Projektes soll ein leistungsbezogener Vergleich gezogen werden zwischen dem Operateur und von Entwicklern entworfenen Automatiken. Wenn den beiden Mikroweltbewohnern vor dem Versuch erklärt wird, dass sie in ihrer Aufgabe von einem menschlichen Operateur unterstützt werden, verhalten sie sich anders als wenn sie von einer Automatik unterstüzt werden. Um diese sozialen Aspekte auszuklammern und gleichzeitig den Vergleich zu den Automatiken zu erhalten, ist eine räumliche Teilung vorgesehen. Diese räumliche Teilung war in den frühen Phasen des ATEO-Projektes bereits realisiert, es wurde aber nur ein Computer mit zwei Monitoren benutzt. Da der Operateursarbeitsplatz wesentlich komplexer geworden ist, ist mit einem Blick auf die Performanz der Einsatz von zwei Rechnern berechtigt. Die Performanz des Operateursarbeitsplatzes hängt primär von der Streckenvorschau ab. Zur Zeit ist diese mit dem gleichen Algorithmus wie in SAM implementiert, weshalb die Performanz bei Einsatz eines images auf einem Rechner ungenügend ist. Im weiteren Verlauf sind zusätzliche Elemente zur Unterstützung des Operateurs geplant, so z.B. eine Aufbereitung von Messdaten zur Herzratenvariabilität der Mikroweltbewohner oder anderen Kenngrößen aus SAM. Besonders die Implementierung der Herzratenvariabilität ist mit hohem technischen Aufwand verbunden und kann zu einem schwer handhabbaren Menge an Daten führen. Diese Erweiterungen haben einen schwer abschätzbaren Einfluss auf die Performanz, eine Verbesserung kann aber ausgeschlossen werden. Eine Umsetzung in der aktuellen Projektphase ATEO 2.0 ist im Hinblick auf die Erfahrungen zum Aufwand der bisherigen Implementierungen unrealistisch.

Mit den leistungsstarken Rechnern und der physikalischen Trennung der beiden Softwarekomponenten kann auch die angestrebte Stepping-Frequenz von 30-49ms eingehalten werden. Das Steuern des Fahrobjektes ist dadurch sehr flüssig. Zusätzlich ist es möglich, in kurzen Abständen den Systemzustand von SAM in einer Logdatei festzuhalten, da ein Eintrag in die Logdatei in jeden Morphstep geschieht. Die Mikroweltbewohner bleiben vom Schreiben der Logdatei unberührt, wodurch es keinen direkten Einfluss auf das Verhalten hat. Die Folge ist, dass der dynamische Prozess mit hinreichend vielen Messwerten dargestellt und damit die statistische Auswertung genauer wird.

4.1.2 *SAM - Automatiken*

Im Rahmen der Diplomarbeit von Kesselring (2009) wurde eine Softwarearchitektur entworfen, die die Automatiken sauber von SAM trennt und die Vernetzung wurde schon früh mit in den Entwurf einbezogen. Nur wenige Anpassungen mussten tatsächlich im SAM-Quellcode vorgenommen werden. Im Zeitraum der Entwicklung waren die Automatiken im selben Squeak-Image wie SAM gespeichert. Trotzdem hat sich die Performanz beim Einsatz der Testautomatik nicht stark verschlechtert und eine Zeit von durchschnittlich 48ms ergeben. Die Messungen wurden auf den Rechnern des ATEO Projektes ausgeführt, nachdem ein neues Squeak-Image erzeugt wurde. Weitere Messungen von Kai Kesselring haben gezeigt, dass die Berechnungen für das Funktionieren einer Automatik ca. eine Millisekunde dauern. Diese Zeit wird sich vermutlich auch bei komplexeren Automatiken nicht signifikant erhöhen.

Eine mögliche Performanzsteigerung würde im Moment durch den Mehraufwand einer Vernetzung wieder aufgehoben werden. Der Mehraufwand beinhaltet die physikalische Übermittlung, den Verbindungsaufbau und die Datenverarbeitung. Sollte sich die Performanz durch spätere Erweiterungen verschlechtern, kann eine Vernetzung oder eine Neuimplementierung der Streckendarstellung für Entlastung sorgen.

4.2 VERNETZUNG DER HARDWARE

Für die Untersuchungen des Operateursarbeitsplatzes im Labor werden zwei Rechner namens ATEO1 und ATEO2 miteinander vernetzt. Die Hardwarekonfiguration kann Tabelle 4 entnommen werden. Auf welchem Rechner SAM oder der Operateursarbeitsplatz ausgeführt wird, ist der vergleichbaren Konfiguration wegen vernachlässigbar. Für das Netzwerk kommt ein 5-Port 10/100M Fast Ethernet Switch zum Einsatz, dabei ist das Netzwerk im Versuchsaufbau frei von anderem Netzwerkverkehr, da eine Internetanbindung vermieden wird und auch sonst keine weiteren Programme über das Netzwerk kommunizieren. Desweiteren kommen ein 30"Bildschirm für den Operateursarbeitsplatz und ein 19"Bildschirm für die SAM-Simulation zum Einsatz. Das Verhalten der Mikroweltbewohner wird mit der Logitech Pro 9000 QuickCam, eine 2-Megapixel-Webkamara mit Autofokus, an den Operateur übermittelt. Die zwei Mikroweltbewohner benutzen je einen Logitech Attack 3 Joystick. Über jeweils ein Notebook können die Mikroweltbewohner und der Operateur Fragebögen beantworten.

	ATEO1
CPU	Intel Core 2 Quad @ 2.40 Ghz
Arbeitsspeicher	4 GB DDR2 800 Mhz
Grafikkarte	Nvidia Geforce 9800 GTX 512 MB
Netzwerkkarte	Nvidia on board Gigabit-Ethernet
Betriebssystem	Windows XP Professional SP2 64bit
	ATEO2
CPU	Intel Core 2 Quad @ 2.40 Ghz
Arbeitsspeicher	3 GB DDR3 1066 Mhz
Grafikkarte	ATI Radeon HD 4850 512 MB
Netzwerkkarte	Intel on board Gigabit-Ethernet
Betriebssystem	Windows Vista Home Premium SP1 32bit

Tabelle 4: Hardwarekonfiguration von ATEO1 und ATEO2

4.2.1 *Versuchsaufbau SAM Voruntersuchung*

In Abbildung 6 ist der Versuchsaufbau für die Voruntersuchung von SAM dargestellt. Hier wurde SAM 1.5 auf einem Rechner mit 19″ Monitor getestet. Den Mikroweltbewohnern standen zwie Joysticks sowie eine Maus zur Verfügung. Auf zwei Notebooks waren die Fragebögen angezeigt (nicht in der Grafik zu sehen).

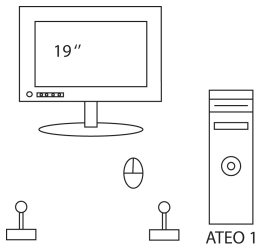


Abbildung 6: Versuchsaufbau SAM Voruntersuchung (SAM 1.5)

4.2.2 *Versuchsaufbau SAM Hauptuntersuchung*

Abbildung 7 zeigt den Versuchsaufbau für die SAM Hauptuntersuchung. SAM 2.0 wurde hier mit zwei Mikroweltbewohnern getestet und der Ablauf mit einer Webcam aufgezeichnet. Damit das Aufzeichnen keinen Einfluss haben kann auf die Leistung von ATEO1 wurde ein zweiter Rechner genutzt (ATEO3). Wieder wurden auf zwei Notebooks die Fragebögen zur Verfügung gestellt.

4.2.3 *Versuchsaufbau OA Untersuchung*

Abbildung 8 skizziert einen wahrscheinlichen Aufbau der Versuche mit Operateursarbeitsplatz. Der Operateursarbeitsplatz in einem seperatem Raum besteht aus einem Rechner mit der Software, die auf

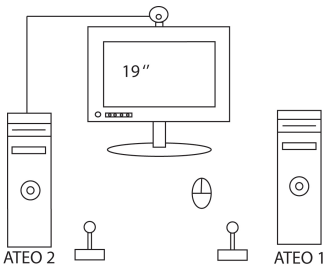


Abbildung 7: Versuchsaufbau SAM Hauptuntersuchung (SAM 2.0)

dem 30'' angezeigt wird. Hinzu kommt die Aufzeichnung des Operateurs via Webcam und der Eisnatz eines zweiten Rechners für diese. Über eine Webcam im Raum der Mikroweltbewohner, welche mit dem Rechner des Operateurs verbunden ist, wird ihm dieses Bild in seinem Arbeitsplatz zur Verfügung gestellt. Die Rechner des Operateurs und der Mikroweltbewohner sind über einen Switch zu einem privaten Netzwerk zusammengeschlossen.

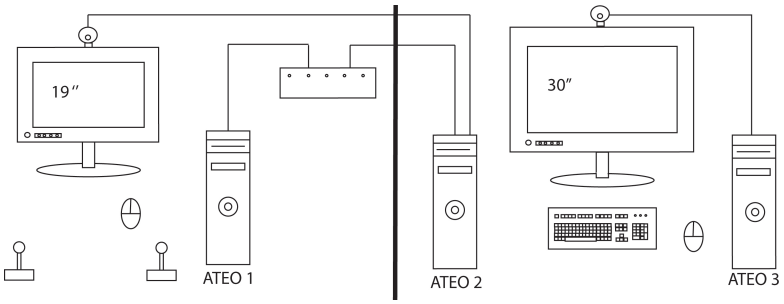


Abbildung 8: Versuchsaufbau OA Untersuchung (SAM 2.0 inkl. Vernetzung)

4.3 VERNETZUNG VON SAM UND OPERATEURSARBEITSPLATZ

Die anfängliche Idee war es SAM und den Operateursarbeitsplatz mit auf XML basierenden SOAP-Nachrichten zu vernetzen. Eine Squeak-Bibliothek ist vorhanden und theoretisch auch die Möglichkeit SAM mit anderen Systemen zu verbinden. Die durch XML erwartete Interoperatibilität ist aber nicht in Bezug auf die verwendete SOAP Bibliothek SOAP-Core gegeben. Es gibt zwar erfolgreiche Tests mit Dolphin Smalltalk und Ruby, aber mit den beliebten Sprachen Java oder C++ nicht. Auch gibt es keine Bibliothek für WSDL, um zumindest eine rudimentäre serviceorientierte Architektur aufzubauen. Trotzdem wurde an dem Entschluss festgehalten und SOAP Nachrichten für die Kommunikation ausprobiert. Sehr schnell hat sich ein weiteres Problem herausgestellt, welches zwar schon von Anfang an vermutet wurde, sich aber später in der Praxis bewahrheitete. Für dynamische Systeme, welche im Millisekundentakt XML Nachrichten versenden und empfangen möchten, ist die Squeak Bibliothek nicht geeignet. Grund dafür ist eine starke Verzögerung beim Empfangen und Verarbeiten der Nachrichten, so dass im Falle des Operateursarbeitsplatzes die Streckenvorschau nicht mehr flüssig und vor allem nicht synchron zu SAM gelaufen ist. Für diesen Fall wurde dann auf Sockets und die Protokolle TCP und UDP

zurückgegriffen. SOAP Nachrichten sind aber gut geeignet für Remote Method Invocation (RMI) und wurden deshalb auch für das Laden und Starten der Streckenvorschau im Operateursarbeitsplatz verwendet.

Für Daten, die im Millisekundentakt verschickt werden sollen, wurden UDP Sockets genutzt. Die Begründung hierbei ist, dass UDP Sockets zwar unsicher sind und verlorengegangene Pakete nicht bemerkt und neu gesendet werden, dies aber bei der sehr hohen Sendefrequenz am Ende nicht bemerkt werden wird. Damit kann man den Geschwindigkeitsvorteil von UDP ausnutzen. Für sensitive, da selten anfallende Daten, wie die Eingriffe des Operateurs, wird ein TCP Socket genutzt. Die Möglichkeit einen dieser Eingriffe oder Hinweise über das Netzwerk zu verlieren und dann Sekunden später den Operateur den Eingriff erneut tätigen zu lassen, ist nicht akzeptabel. Erstens kann es schon zu spät sein für ein Eingreifen und zweitens ist nicht einmal gewährleistet, dass z.B. ein nicht gezeigter visueller Hinweis vom Operateur bemerkbar ist. Er bekommt zur Zeit kein Feedback dazu. Diese Daten dürfen somit nicht verloren gehen und werden deshalb über das zuverlässige aber im Vergleich zu UDP langsamere TCP Protokoll versendet.

4.3.1 *Die Performanz von SOAP*

Warum sich die verwendete SOAP Bibliothek nicht eignet für das Versenden von Nachrichten im Millisekundentakt hat mehrere Gründe. Eine Auflistung möglicher Performanz-Engpässe von SOAP Bibliotheken im Allgemeinen bieten Machado und Ferraz (2005). Mit Hilfe des HTTP Headers konnten einige überprüft werden, andere sind genereller Natur und haben immer Einfluss auf die Leistung einer SOAP Bibliothek. Diese allgemeinen Nachteile in Bezug auf die Bibliothek Soap-Core sind:

- Die Nachrichtengröße ist vier- bis zehnmal größer als die gleiche im binären Format.
- Die Nachrichtengröße wird von SOAP-Core für jede Nachricht berechnet.
- Die XML Nachricht wird während der Programmausführung geparst.
- Die XML Nachricht muss codiert und decodiert werden.

Eine Eigenschaft von SOAP-Core bzw. vom verwendeten HTTP Protokoll ist der Umstand, dass für jede Nachricht eine Verbindung über das Netzwerk aufgebaut und anschließend wieder abgebaut wird, anstelle diese Verbindung nur einmal aufzubauen und für alle folgenden Nachrichten zu verwenden. Wenn nun in jedem Morphstep, also ca. alle 40ms, eine neue Verbindung aufgebaut werden muss und es sich wie beim HTTP und TCP/IP Protokoll um einen drei-Wege-Handshake handelt, dann kommt es zwangsläufig zu Verzögerungen. Diese Verzögerung kombiniert mit den anderen Performanz-Bremsen macht schnell klar, dass zu viele XML Nachrichten in zu kurzen Abständen geschickt werden. In Abbildung 9 sieht man, dass die Verbindung immer geschlossen wird (Connection: close) und die Größe der Nachricht auch jedes Mal berechnet wird (Content-length: 541).
Um den Header anzeigen zu können, muss diese Option gesetzt und ein Transcript geöffnet werden:

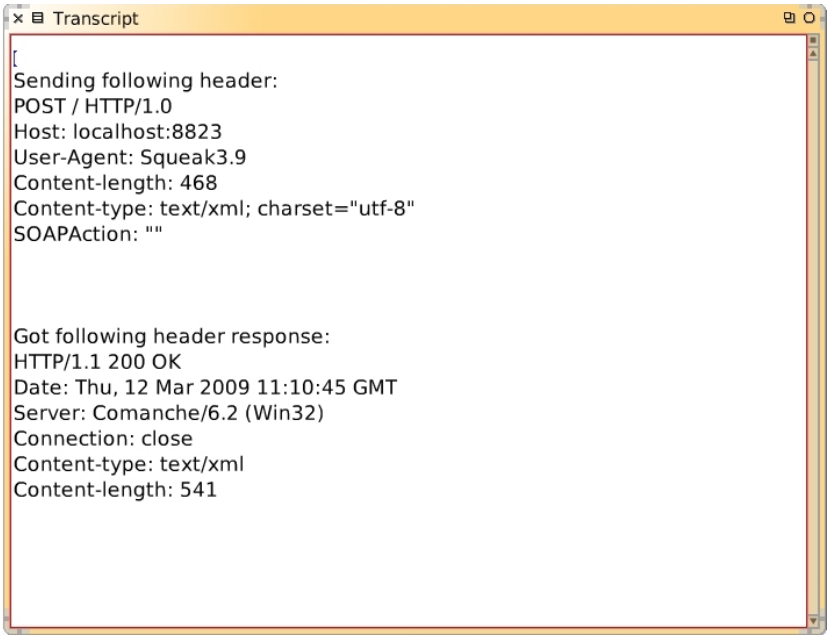


Abbildung 9: HTTP Header einer SOAP Nachricht mit SOAP-Core

SoapSetting showClientLogHttpLevel: true
Alle im Anschluss gesendeten SOAP Nachrichten werden mit ihrem HTTP Header im Transcript angezeigt.

4.4 ERWEITERUNGEN AN SAM

Um die Funktionalität des Operateursarbeitsplatzes in Bezug auf die Eingriffsmöglichkeiten zur Verfügung stellen zu können, musste SAM angepasst und erweitert werden. Die Joystickauslenkung nach rechts oder links blockieren und die Möglichkeit die maximale Geschwindigkeit von SAM manipulieren zu können, fehlte in SAM bisher. Im Folgenden sollen diese zwei Erweiterungen beschrieben werden. Eine detaillierte Auflistung der erweiterten Klassen ist in Anhang A nachzulesen.

JOYSTICKBLOCKIERUNG Im Operateursarbeitsplatz gibt es drei Buttons, die horizontal ausgerichtet sind und drei Zustände repräsentieren. Ist der mittlere Button aktiviert, soll die Joystickauslenkung nicht beeinflusst sein. Wird der rechte Button gedrückt, sollen die Mikroweltbewohner nach rechts fahren, d.h. die Joystickbewegung nach links ist blockiert. Den linken Button zu drücken, bewirkt das Blockieren nach rechts, so dass nur eine Steuerung nach links möglich ist. Diese Information wird über das Netzwerk an SAM gesendet und in ATEOSocket»receiveData werden die beiden Variablen denyRight und denyLeft aus ATEOrealJoysticksStepping auf wahr oder falsch gesetzt. In der step-Methode von ATEOrealJoysticksStepping werden dann die beiden Variablen ausgewertet und die Auslenkung auf Null gesetzt, wenn eine Auslenkung in diese Richtung versucht wird. Die beiden Variablen können ausschließlich über das Netzwerk vom Operateur gesetzt werden, es unterliegt also auch seiner Verantwortung, die Blockierung wieder aufzuheben.

ANPASSUNG DER MAXIMALGESCHWINDIGKEIT Als eine weitere Eingriffsmöglichkeit ist es dem Operateur erlaubt, die maximale Geschwindigkeit zu ändern. Die bisher maximale Geschwindigkeit kann jetzt vom Operateur um bis zu 20% angehoben werden. Ebenso ist es ihm möglich die Geschwindigkeit in 1% Schritten bis auf 0% zu reduzieren. Die über das Netzwerk übermittelte Prozentzahl wird von ATEOAblaufsteuerung»step:punkt: an ATEOrealJoysticksStepping in jedem Morphstep übergeben. Zwei Fälle sind zu unterscheiden, entweder ist die gewünschte Maximalgeschwindigkeit gleich Null oder größer Null. Werte größer als 120 liegen außerhalb des definierten Bereiches des Schiebereglers und können somit nicht auftreten. Wenn die Maximalgeschwindigkeit Null sein soll, wird die Berechnung der Geschwindigkeit aus den Joystickeingaben der Mikroweltbewohner nicht durchgeführt sondern direkt auf Null gesetzt. In allen anderen Fällen wird in der Berechnungsformel die Geschwindigkeit am Ende noch prozentual angepasst. Es ergibt sich folgende modifizierte Formel für die Berechnung:

$$schritt = \frac{(1024 - updown1 * AnteilMB1) + (1024 - updown2 * AnteilMB2)}{vuStickUD * 50 * (100/vmax)}$$

updown1 und updown2 sind die vertikalen Auslenkungen der Mikroweltbewohner, AnteilMB1 und AnteilMB2 die Inputverteilung. vuStickUD ist konstant 2.0 und ist als Dämpfung der Joystickumsetzung implementiert. vmax ist die durch den Operateur eingestellte Anpassung der Maximalgeschwindigkeit in Prozent. Der mögliche Wertebereich von vmax ist 0 bis 120.

4.5 ERWEITERTE SOFTWAREARCHITEKTUR

Die zwei auf Seiten von SAM neu angelegten Klassen OAService und ATEOSocket sind in einer neuen Kategorie namens ATEO-Service organisiert und damit getrennt von der ATEO-Kategorie. Im Folgenden soll eine kurze Beschreibung der Klassen gegeben werden, Details sind im Anhang A festgehalten. Außerdem findet man im Anhang die Klassen und Methoden, die im Operateursarbeitsplatz angepasst wurden, um die Vernetzung möglich zu machen. Dies betrifft vor allem die Ereignisbehandlung der Buttons.

4.5.1 *OAService*

Auf beiden Seiten der Verbindung zwischen Operateursarbeitsplatz und SAM wird die Klasse zuständig für die Kommunikation mit SOAP als OASocket definiert. Dieser Umstand kann zu Verwirrung führen, da nicht immer klar ist, von welcher Klasse die Rede ist. Aus diesem Grund muss immer genauer spezifiziert werden, welche Klasse gemeint ist. Der Grund für diese Benennung liegt in SOAPCore, der Klassenbibliothek, begründet. Wenn eine Verbindung aufgebaut wird und auf Seiten von SAM die entsprechende Klasse nicht OAService heißt, kommt es zu einem Fehler bei SAM. Die Verbindung kann nicht aufgebaut werden und die Kommunikation wird unterbrochen, bevor sie richtig starten konnte. Der Debugger gibt als Fehler an, dass der Schlüssel, mit dem im Smalltalk Dictionary gesucht wird, nicht OAService lautet. Eine naheliegende Lösung des Problems ist somit die Umbenennung des Klassennamens in OAService. Tests haben bestätigt, dass die beiden

Klassen für die fehlerfreie Kommunikation gleich genannt werden müssen.

OAService (SAM) realisiert den entfernten Methodenruf für das Laden und Starten der Streckenvorschau im Operateursarbeitsplatz mit SOAP. Zusätzlich kann der erste aktiv fahrende Mikroweltbewohner auf Seiten des Operateurs von SAM aus eingestellt werden. Dies ist nötig, da auch die Streckenvorschau wie SAM selbst die Konfiguration des Versuches aus der Textdatei steps.txt entnimmt. Im ersten Versuchsstep wird der aktive Mikroweltbewohner laut steps.txt durch das Menü konfiguriert. Da die Streckenvorschau dieses Menü nicht besitzt, kann im ersten Versuchstep der aktive Mikroweltbewohner nicht ermittelt werden. Dieser Mikroweltbewohner wird im Konfigurationsmenü von SAM eingestellt und kann auch für die Streckenvorschau übernommen werden. Aus diesem Grund wird mit Hilfe von SOAP der aktive Mikroweltbewohner von SAM aus im Operateursarbeitsplatz korrekt gesetzt. Eine weitere Möglichkeit wäre, diesen Konfigurationsschritt aus dem Quellcode der Streckenvorschau zu entfernen. Im Moment ist die Streckenvorschau eine geringfügig modifizierte Version von SAM, d.h. die grundlegenden Programmabläufe sind übernommen worden. Irrelevante Programmteile wie der Logging-Mechanismus, das Konfigurationsmenü, die Instruktionen, die Eingriffe des Operateurs und damit jegliche Behandlung von Tastatureingaben wurden entfernt. Die Konfiguration des Versuches ist für eine Streckenvorschau nicht vollständig nötig und könnte minimiert werden, so auch die Ermittlung des aktiven Mikroweltbewohners oder die Anteilbestimmung.

Aus Sicht der Performanz sind beim jetzigen Stand keine Änderungen an der Streckenvorschau oder der Kommunikation nötig. Dies kann sich aber nach Beendigung der Implementierung geändert haben, denn die Streckenvorschau ist zur Zeit nicht vollständig funktionstüchtig. Die Anpassungen werden in Folgearbeiten von Christian Leonhard übernommen. Das Gegenstück dieser SOAP Kommunikation ist auf Seiten des Operateursarbeitsplatzes OAService. Hier sind die von SAM aus gerufenden Methoden zu finden. Im einzelnen sind das *loadConfigsAndGraphics, setAktiverMB:* und *startTracking*. Zusätzlich sind hier die register-Methoden zu finden, mit Hilfe derer diese Methoden im System bekannt gemacht und von SAM aus genutzt werden können.

In SAM werden die SOAP Nachrichten in ATEOInstruction»continue-Start für das Starten des Trackings, in ATEOConfig»go für das Laden der Streckenbilder des Operateursarbeitsplatzes und in ATEO-VuSt»nextStep für das Setzen des aktiven Mikroweltbewohners gesendet.

4.5.2 *ATEOSocket & OASocket*

ATEOSocket im SAM Image wartet auf einen Verbindungsaufbau über das TCP Protokoll und ist für die Verarbeitung der auditiven und visuellen Hinweise, der Manipulation des Joystickinputs durch den Operateur und der Blockierung der Joystickausrichtung nach links oder rechts zuständig. Die zentrale Methode ist *receiveData*, die die über das Netzwerk verschickten Zahlen den richtigen Eingriffen zuordnet und die Methodenrufe absetzt bzw. die Inputverteilung direkt ändert. Auf der Seite des Operateurs wird in der Klasse OASocket die Verbindung mit SAM aufgebaut. Für diese Verbindung wird der Port 58082 verwendet. Gesendet werden die Daten über die globale Variable OASocketStream, welche ein Objekt der OASocket Klasse bereithält.

Tabelle 5 listet die über das Netzwerk gesendeten Daten, mit ihrer Bedeutung und von wo im Operateursarbeitsplatz aus die Daten gesendet werden, auf.

Außerdem baut OASocket vom Operateursarbeitsplatz aus eine Verbindung über das TCP Protokoll mit SAM auf, um die Anpassung der maximal möglichen Geschwindigkeit übertragen zu können. Hier wird der Port 58090 genutzt.

4.5.3 *ATEOAblaufsteuerung*

In ATEOAblaufsteuerung werden die vier Kommunikationsverbindungen *speedStream (Socket UDP)*, *coordiantesStream (Socket UDP)*, *operatorInterferenceStream (ATEOSocket TCP)* und *vmaxStream (Socket TCP)* initialisiert und genutzt. Dabei ist die steps-Methode zentral dafür zuständig, regelmäßig die Daten für die Geschwindigkeit und die Koordinaten zu senden und für die Operateurseingriffe zu empfangen. Das Empfangen der Daten wird für jeden Datenstrom in einem separaten Prozess ausgeführt. So kann erreicht werden, dass der step nicht anhält und SAM beginnt zu stocken.

4.5.4 *Konfiguration der Vernetzung*

Um die Kommunikation zwischen den beiden Versuchsrechnern zu gewährleisten, muss die IP-Adresse an folgenden Stellen im Quellcode angepasst werden. In SAM wird die IP-Adresse in ATEOAblaufsteuerung»initialize und in OAService»initialize eingestellt.
Beim Operateursarbeitsplatz wird die Konfiguration der IP-Adresse in OASocket»initialize durchgeführt.

4.6 VERNETZUNG VON SAM UND DEN AUTOMATIKEN

4.6.1 *Konzeptvorschlag*

Um die Berechnungen für die Automatiken vornehmen zu können, werden aus SAM die vier Objekte ParID (Strecke), CarID (Fahrobjekt), JoyID (Joystickeingaben) und Ctrl (Kontrollobjekt) benötigt. Eine Möglichkeit wäre es die Attribute dieser vier Objekte über Socketstreams zu senden. Dieses Vorgehen würde einige Sockets benötigen und evtl. auch eine einfache Kodierung der Daten. Eine Alternative bietet die Klassenbibliothek rST. Mit Hilfe dieser können diese vier Objekte im Netzwerk zur Verfügung gestellt werden. Umgesetzt ist der entfernte Zugriff auf Objekte durch einen Broker. Dem Broker werden zu Beginn Objekte, auf die über ein Netzwerk zugegriffen werden soll, bekannt gegeben. Anfragen an diese Objekte werden an den Broker gestellt und dieser leitet sie an die entsprechenden Objekte weiter bzw. erfragt die gewünschten Daten. Im Anschluss sendet der Broker die gewünschten Daten an das anfragende Objekt. Dies hat den Vorteil, dass die betroffenden Objekte für diesen Zugriff über ein Netzwerk nicht angepasst werden müssen. Es kann aber zur Beeinflussung der Performanz kommen.

CODE	BEDEUTUNG	GESENDET VON
1	Joystickblockade rechts	OAInputMorph»setDirection:
2	keine Joystickblockade	OAInputMorph»setDirection:
3	Joystickblockade links	OAInputMorph»setDirection:
4	Balance rechts	OAIconicSwitchButtonRightUser »doButtonAction
5	Balance links	OAIconicSwitchButtonLeftUser »doButtonAction
6	Balance links/rechts	OAIconicSwitchButtonBothUsers »doButtonAction
7	Sound 1	OAButtonsMorph»sound1Action
8	Sound 2	OAButtonsMorph»sound2Action
9	Sound 3	OAButtonsMorph»sound3Action
10	Sound 4	OAButtonsMorph»sound4Action
20	Input MWB1 = 0%, MWB2 = 100%	
21	Input MWB1 = 25%, MWB2 = 75%	
22	Input MWB1 = 50%, MWB2 = 50%	OAInputMorph»distribution-MWI1/2ButtonAction
23	Input MWB1 = 75%, MWB2 = 25%	
24	Input MWB1 = 100%, MWB2 = 0%	
41	Popup „oben"	OAButtonsMorph»forward-ButtonAction
42	Popup „unten"	OAButtonsMorph»back-ButtonAction
43	Popup „links"	OAButtonsMorph»left-ButtonAction
44	Popup „rechts"	OAButtonsMorph»right-ButtonAction

Tabelle 5: Netzwerkdaten und ihre Bedeutung

4.7 SCHNITTSTELLEN FÜR ZUKÜNFTIGE ERWEITERUNGEN

Für den Operateursarbeitsplatz ist unter anderem vorgesehen, die Abstufung der Joystickinputverteilung feiner abzustufen. Die Abstufung soll von 25% auf 5% verfeinert werden. Diese Änderung kann die Codes von 20-40 umspannen und muss nur noch auf Seiten des Operateursarbeitsplatzes angepasst werden. Auf Seiten von SAM ist in ATEOSocket»receiveData als Kommentar bereits die Umsetzung implementiert.
Zusätzliche auditive Hinweise können im Zahlencodebereich 7-19 eingefügt werden. In SAM sind bereits 8 Sounddateien vorhanden und in ATEOSocket»receiveData als Kommentar auch schon vorgesehen. Eine größere Anzahl auditiver Hinweise ist ohne weiteres umsetzbar. Die Erhöhung der Anzahl der Popups ist ebenfalls ohne Probleme möglich. Auf Seiten von SAM muss bei allen Erweiterungen dieser Art in ATEOSocket»receiveData die Anpassung vorgenommen werden und auf Seiten des Operateursarbeitsplatzes muss das Ereignis „Button drücken" den korrekten Zahlencode senden.
Sollten neue Sockets nötig werden, können diese an den gleichen Stellen auf Seiten von SAM implentiert werden, wie die bereits vorhandenen. Eine Zusammenfassung aller UDP Sockets in einer Klasse und der Zugriff auf das spezielle Socket mit Hilfe von Zugriffsmethoden, kann bei hoher Anzahl nötig für die Lesbarkeit und Kohäsion werden.
Auf Seiten des Operateursarbeitsplatzes bietet die Klasse OASocket die Möglichkeit weitere UDP Sockets hinzuzufügen.
Neue SOAP Nachrichten können in OAService in SAM und im Operateursarbeitsplatz hinzugefügt werden. Hier kann man sich an den Aufbau der bereits vorhandenen orientieren.

4.8 TEST DER VERNETZUNG

Für das Testen einer Vernetzung mit Hilfe von Sockets und SOAP bieten die Bibliotheken keine Möglichkeit der Automatisierung. Beide Bibliotheken wurden in sich mit SUnit getestet, so dass von der korrekten Funktionsweise dieser ausgegangen werden kann.

Im Allgemeinen ist das Testen von verteilten Systemen schwierig, da eine Reihe von spezifischen Eigenschaften eine einfache Herangehensweise mit beispielweise SUnit stark verkompliziert. Zum einem kann man durch Verzögerungen des Netzwerkes und im Falle von SAM durch die Dynamik des Systems den Systemstatus zu keiner bestimmten Zeit für einen Vergleich als wohldefiniert feststellen. Auch ist es schwer Fehler zu reproduzieren und Ausnahmeszenarios umzusetzen, da das Verhalten von verteilten Anwendungen asynchron und nichtdeterministisch ist.

Für das Testen von verteilten Systemen kann und wird vor allem auf Testumgebungen zurückgegriffen, die das System mit Testdaten füttert und die Ausgabe oder das Verhalten beobachtet und auswertet. Beim Einsatz solcher Methoden muss darauf geachtet werden, dass das System davon nicht so weit verändert wird, dass die Resultate davon betroffen sind. Im Falle von SAM kann der Einsatz einer solchen Testumgebung die Performanz beeinflussen und die Zeit zwischen zwei Morphsteps verlängern.

Die Vernetzung von SAM und Operateursarbeitsplatz wurde auf einfache Art und Weise getestet. So wurde während der Nutzung beider Komponenten jeweils vor und nach der Übertragung Daten auf einem

Transcript ausgeschrieben und der Vergleich händisch durchgeführt. Diese Herangehensweise wurde für die Übertragung von Geschwindigkeit und Position des Fahrobjektes und die Nutzung dieser in der Streckenvorschau verfolgt. Für die Eingriffe des Operateurs wurden diese nacheinander angestoßen und der Einfluss auf SAM und die Mikroweltbewohner überprüft. Die Verbindung der beiden Computer durch Sockets wurde mit Hilfe des Objektinspektors überprüft. Dieses Vorgehen ist nicht automatisierbar, wenn man aber in regelmäßigen Abständen die Funktionalität überprüft, kann eine korrekte Funktionsweise sichergestellt werden. Im Falle von Fehlern konnte mit Hilfe des Debuggers schnell die Ursache gefunden werden.

5 DISKUSSION

Es gibt einige Dinge, die mit jetzigem Wissen und Verständnis anders umgesetzt werden würden als es zur Zeit der Fall ist. Durch die Erfahrung im Umgang mit Squeak, Smalltalk und den verschiedenen Softwarekomponenten SAM, Operateursarbeitsplatz sowie den Automatiken können nun gezielter Schwachstellen erkannt und Verbesserungen implementiert werden. Eine Neuimplementierung von SAM am Anfang der zweiten Projektphase von ATEO wäre der richtige Schritt gewesen. Im Laufe der Anpassungen wurde SAM in fast jeder Klasse verändert. Oft wurde die interne Struktur der Klassen verbessert und Methoden zum Teil neu implementiert und dadurch lesbarer gemacht. Die Softwarearchitektur hat sich dadurch aber nicht verbessert. An einigen Stellen ist die Architektur durch die Neuerungen schlechter geworden, da die vorhandenen globalen Variablen weiter genutzt wurden und dadurch weitere Abhängigkeiten und Zyklen entstehen konnten. Die Funktionalität der Hindernisse wurde zum Teil in vorhandenen Klassen integriert. Hier wäre der bessere Weg eine neue Klasse gewesen. Die Entscheidung keine neue Klasse zu entwerfen, war im anfangs geringen Umfang der Anpassungen begründet. Inzwischen ist aber die Komplexität der Hindernisse angestiegen und über mehrere Klassen verteilt.

Eine Neuimplementierung hätte die Softwarearchitektur bereinigen und verbessern und durch eine verbesserte Kopplung und Modularisierung auf Veränderungen besser eingestellt werden können. Zum jetztigen Zeitpunkt ist diese Neuimplementierung aber nicht mehr von praktischer Relevanz für das ATEO Projekt, da SAM in seiner Funktionalität vollständig ist. Die Programmierung der generischen Toolbox setzt SAM neu um, so dass in Zukunft auf Basis einer Trackingsimulation mit belebter Mikrowelt ingenieurpsychologische Fragestellungen untersucht werden können.

Die Vernetzung könnte man besser kapseln und eine eigene Klasse, die die Sockets zusammenfässt, erstellen. Im gleichen Zuge könnte man auf die Verwendung von SOAP Nachrichten verzichten und sich auf eine Technologie zurückziehen. Dadurch wird der Quellcode verständlicher, weil die gesamte Vernetzung auf Sockets beruhen und gleichzeitig an einem Ort gebündelt vorliegen würde. Zusätzlich könnte man ein Menü für die Netzwerkkonfiguration entwerfen. So könnten die kommunizierenden Computer flexibel konfiguriert werden. Zur Zeit ist die IP Adresse fest im Quellcode integriert. Sobald sich diese ändert, muss sie auch im Quellcode angepasst werden. Desweiteren könnte man über das Menü das Deaktivieren der Vernetzung ermöglichen, so dass es nur noch eine Version gibt und nicht wie zur Zeit zwei, mit und ohne Vernetzung. Im gleichen Zuge könnte das SAM Solo Tracking integriert werden.

Das vorhandene Konfigurationsmenü kann dafür neu implementiert werden, so dass die Bedienung erleichtert wird. Zur Zeit ist es nicht ersichtlich, was die einzelnen Elemente konfigurieren und wie korrekte Eingaben aussehen. Als Beispiel sei das Einstellen des Geschlechtes angeführt. Hier kann man negative sowie positive Zahlen einstellen. Sinnvoll sind aber nur die Werte 1 und 2, welche das Geschlecht ko-

dieren. Ob 1 weiblich und 2 männlich bedeutet oder umgekehrt ist genauso wenig ersichtlich. Hier wären beispielsweise Checkboxes geeignet. Ähnlich verhält es sich mit dem startMB, welcher entweder 1 oder 2 sein kann und den zuerst fahrenden Mikroweltbewohner bezeichnet. Fest steht, dass die Usability verbessert werden muss, damit auch Projektfremde ohne Schwierigkeiten einen Versuch konfigurieren können. Hier könnte Maui [1] eingesetzt werden. Maui bietet die Möglichkeit User Interfaces effizient zu erstellen. Eine ausführliche Dokumentation liegt zu diesem Framework vor. Auch bietet das Morphic Konzept die Möglichkeit solche grafischen Elemente zu nutzen, ein Werkzeug für eine einfache GUI-Programmierung gibt es aber hierfür nicht.

Während der Programmierung sind die Schwächen und die damit verbundenen Einschränkungen von Squeak sowie der übernommenen Architektur deutlich geworden. Für eine Vernetzung sind nur wenige Bibliotheken vorhanden und abstraktere Konzepte wie SOAP oder rST nutzen auch nur die einzige von Squeak angebotene Klassenbibliothek, nämlich Sockets. Die Nutzung von SOAP vereinfacht zwar die Implementierung deutlich und reduziert auch die Fehleranfälligkeit bei der Programmierung, ist aber stark eingeschränkt und bietet in einem dynamischen System wie SAM zu viele Performanzbremsen. Die Nutzung der low-level Bibliothek für Sockets bietet deswegen die angestrebte Performanz, ist aber fehleranfälliger in der Programmierung und weniger komfortabel nutzbar, da die Kommunikation auf einer weniger abstrakten Ebene stattfindet und im Vergleich zu SOAP oder rST hardwarenäher funktioniert.

Die Einschränkungen der vorgegebenen Architektur sind vor allem in der Erweiterbarkeit und Wartbarkeit zu finden, so dass man oft gezwungen ist, die vorhandenen Klassen zu erweitern als sie auszutauschen. Eine weitere Bremse vor allem für die Performanz des dynamischen Systems sind die ineffizient implementierten Bibliotheken von Squeak. Vor allem das Morph Konzept ist hiervon betroffen; so ist es nur mit leistungsstarken Computern möglich Bilder in hoher Frequenz zu verschieben, da Morphe, welche Bilder beinhalten, jeden Pixel als Eigenschaft vorhalten und damit sehr groß werden. Diesen Umstand muss man während des Entwurfes immer mitbedenken, was einem in seiner Kreativität einschränkt. Zusammenfassend kann man feststellen, dass Squeak für dynamische Systeme nur bedingt zu empfehlen ist.

1 http://wiki.squeak.org/squeak/3836

6 ZUSAMMENFASSUNG UND AUSBLICK

6.1 ZUSAMMENFASSUNG

Auf den Arbeiten der Studienarbeit konnte fast durchgängig aufgebaut werden, so dass SAM im Rahmen dieser Diplomarbeit mit der gewünschten Funktionalität ausgestattet werden konnte. Die Arbeiten in Bezug auf die Hindernisse konnten weitergeführt werden, sowie die Inputmanipulation vollendet werden. Diese wurde später wieder entfernt. Am Ende konnte ein stabiler Zustand erreicht und eine von totem Code bereinigte Version erstellt werden. Auf dieser Grundlage wurde eine an die Anforderungen einer Vernetzung angepasste Version entwickelt und auf ein Zusammenspiel mit dem Operateursarbeitsplatz optimiert. Dazu musste die Funktionalität erweitert werden, wobei auf die Aufgabenanalyse durch eine HTA zurückgegriffen werden konnte bzw. eine enge Zusammenarbeit mit Hermann Schwarz ein schnelles Vorankommen gewährleistete. Vernetzt wurden dabei die harten und weichen Eingriffe des Operateurs sowie die Streckenvorschau mit einigen Einschränkungen in der Funktionsweise dieser. In Bezug auf die Vernetzung ist die Streckenvorschau implementiert. Einzig die Anzeige der Joystickbewegungen wurde nicht vernetzt, da sich in diesem Element des Operateursarbeitsplatzes die Funktionalität noch verändern wird und somit auch die Vernetzung.

6.2 AUSBLICK

SAM für das ATEO Projekt und deren Fragestellungen ist weitestgehend abgeschlossen und benötigt keine Erweiterungen mehr. Denkbar sind Erweiterungen, welche im Rahmen anderer Experimente der Sozialpsychologie, Biologischen Psychologie, Allgemeine, Kognitive Psychologie oder Persönlichkeitsdiagnostik nötig werden würden.

Mögliche Verbesserungen oder Anpassungen von SAM auf technischer Seite sind nur grundlegender Natur. So könnte der Algorithmus für die Streckenbewegung effizienter gestaltet werden, wenn man vom Bildverschieben abkommt. Möglich wäre eine Berechnung der Fahrbahn durch Bezierkurven oder Polygone. Diese Anpassungen haben aber direkten Einfluss auf fast jede Klasse, so dass es einer Neuimplementierung von SAM gleichkommen würde. Dies ist der verworrenenden Softwarearchitektur geschuldet, welche den Austausch von Softwarekomponenten fast unmöglich macht.

Desweiteren fehlen SUnit Tests. Im Rahmen des bereits unter Kapitel 3.2 erwähnten Seminars an der Humboldt Universität zu Berlin wurden für SAM 1.5 SUnit Tests erstellt. Für SAM 2.0 fehlen diese aber noch.

SAM muss im Rahmen der Funktionserweiterung des Operateursarbeitsplatzes sicher weiter angepasst werden. Zur Zeit wird an einer Umsetzung der Joystickauslenkungsanzeige im Operateursarbeitsplatz gearbeitet sowie die Inputmanipulation verfeinert. Die Arbeiten im ATEO-Projekt gehen weiter und hoffentlich kann diese Diplomarbeit diese unterstützen und einen schnellen Einstieg fördern.

A KLASSEN-/METHODENBESCHREIBUNG

A.1 SAM 1.5

In diesem Kapitel sollen Veränderungen an SAM im Vergleich zur Dokumentation von Hildebrandt (2009) aufgezeigt werden. Die Dokumentation unveränderter oder umbenannter Methoden kann demnach in Hildebrandt (2009) nachgelesen werden.

A.1.1 *ATEOAblaufsteuerung*

INSTANZVARIABLEN

moveObstacle *geändert*

Typ: Boolean

Beschreibung: Umbenennung von *bewegeHindernis*

collisionCoordinates *geändert*

Typ: Point

Beschreibung: Umbennenung von *collisionsKoordinate*

dynamicObstacle *geändert*

Typ: Boolean

Beschreibung: Umbenennung von *dynHindernis*

staticObstaclePlacing *geändert*

Typ: Integer

Beschreibung: Umbenennung von *statBarrierPlacing*, definierte Werte sind 50 und 25

staticObstacle *geändert*

Typ: Boolean

Beschreibung: Umbenennung von *statHindernis*

schrittVar *entfernt*

Typ: Float

Beschreibung: Diese Variable war unbenutzt und wurde entfernt.

slalomVariation *neu*

Typ: Integer

Beschreibung: Diese Variable speichert die Variation der statischen Variablen. Definierte Werte sind 0 und 1, wobei 0 bedeutet, dass das erste Hindernis rechts vom Pfad und das zweite links erscheint und für 1 umgekehrt.

slalomObstacleLightGreen *neu*

Typ: SketchMorph

Beschreibung: repräsentiert das linke Hindernis bei einer links-rechts Platzierung und hält das Bild block1-1.jpg

slalomObstacleLightGray *neu*

Typ: SketchMorph

Beschreibung: repräsentiert das rechte Hindernis bei einer rechts-links Platzierung und hält das Bild block1-2.jpg

slalomObstacleDarkGreen *neu*

Typ: SketchMorph

Beschreibung: repräsentiert das linke Hindernis bei einer rechts-links Platzierung und hält das Bild block1-3.jpg

slalomObstacleDarkGray *neu*

Typ: SketchMorph

Beschreibung: repräsentiert das rechte Hindernis bei einer links-rechts Platzierung und hält das Bild block1-4.jpg

dynamicBuild *neu*

Typ: Boolean

Beschreibung: Gibt an, ob ein dynamisches Hindernis erstellt wurde (true) oder nicht (false).

static25rl *neu*

Typ: Boolean

Beschreibung: Gibt an, ob ein Slalom mit 25% Pfadabdeckung und erstem Hindernis rechts erstellt wurde (true) oder nicht (false). Siehe die Methode static25rl

static25lr *neu*

Typ: Boolean

Beschreibung: Gibt an, ob ein Slalom mit 25% Pfadabdeckung und erstem Hindernis links erstellt wurde (true) oder nicht (false). Siehe die Methode static25lr

static50rl *neu*

Typ: Boolean

Beschreibung: Gibt an, ob ein Slalom mit 50% Pfadabdeckung und erstem Hindernis rechts erstellt wurde (true) oder nicht (false). Siehe die Methode static50rl

static50lr *neu*

Typ: Boolean

Beschreibung: Gibt an, ob ein Slalom mit 50% Pfadabdeckung und erstem Hindernis links erstellt wurde (true) oder nicht (false). Siehe die Methode static50lr

firstStaticObstacleXCoord *neu*

Typ: Float

Beschreibung: X-Koordinate des ersten statischen Hindernisses eines Slaloms

firstStaticObstacleYCoord *neu*

Typ: Float

Beschreibung: Y-Koordinate des ersten statischen Hindernisses eines Slaloms

secondStaticObstacleXCoord *neu*

Typ: Float

Beschreibung: X-Koordinate des zweiten statischen Hindernisses eines Slaloms

secondStaticObstacleYCoord *neu*

Typ: Float

Beschreibung: Y-Koordinate des zweiten statischen Hindernisses eines Slaloms

Unveränderte Instanz- und Klassenvariablen:
dynamic, lprocess, static, xPos, yPos, Schrittweite

INSTANZMETHODEN

moveDynamicObstacle *geändert*

Beschreibung: Umbenennung von *bewegeDynamischesHindernis*

collision: *geändert*

Parameter:

Name: obstacleNo

Bedeutung: Gibt an, ob das erste Hindernis eines Slaloms (0) oder das zweite (1) getroffen wurde, damit auch nur eines der beiden Hindernisse bei einer Kollision verschwindet.

Beschreibung: Erweiterung von *collision* um den Parameter *obstacleNo* und dessen Verwendung

createDynamicObstacle *geändert*

Beschreibung: Umbenennung von *createDynamicBarrier*

schrittweite *geändert*

Beschreibung: Umbenennung von *getSchrittweite*. Die Richtlinie für die Benennung von Zugriffsmethoden in Smalltalk legt nahe, diese Methoden gleich der Variablen zu nennen ohne vorgestelltem get/set. Diese Namenskonvention wurde in SAM umgesetzt. Siehe Klimas u. a. (1996) Guideline 20 und 22.

schrittweite: *geändert*

Parameter:

Name: schrittweite

Bedeutung: neue Schrittweite

Beschreibung: Umbenennung von *setSchrittweite:*

dynamicBuild: *geändert*

Parameter:

Name: bool

Bedeutung: neuer boolscher Wert für dynamicBuild

Beschreibung: Umbenennung von *setDynHindernis:*

staticObstaclePlacing: *geändert*

Parameter:

Name: num

Bedeutung: neuer Grad der Pfadabdeckung, mögliche Werte sind 50 und 25

Beschreibung: Umbenennung von *setStatBarrierPlacing:*

staticObstacle: *geändert*

Parameter:

Name: bool

Bedeutung: neuer boolscher Wert für staticObstacle

Beschreibung: Umbenennung von *setStatHindernis*

incSchrittweite: *geändert*

Parameter:

Name: inc

Bedeutung: Höhe der Inkrementation

Beschreibung: Die Möglichkeit schneller als 20.48 Pixel pro Morphstep zu fahren sowie des Rückwärtsfahrens wurden entfernt. Die Aktualisierung der Geschwindigkeitsanzeige wurde implementiert. Hierbei ist anzumerken, dass bei Erhöhung der Geschwindigkeit um 1 die beim Fahren mit Joysticks maximal mögliche Geschwindigkeit von 20.48 Pixel pro Morphstep nicht erreicht werden kann. Somit ist auch die Geschwindigkeitanzeige, so wie sie in dieser Version implementiert ist, ungenau.

initialize *geändert*

Beschreibung: Das Erstellen der statischen und dynamischen Hindernisse wird nun im Konstruktor vorgenommen und nicht wie zuvor bei jedem Erscheinen. Dies spart Festplattenzugriffe während eines Versuchssteps und verbessert die Performance. Einige neue Variablen werden mit Werten initialisert.

makeItComplex *geändert*

Beschreibung: Anpassung der Methode an den Slalom, damit dieser erstellt werden kann.

step *geändert*

Beschreibung: Vorbeifahren am Ziel ist nicht mehr möglich. Dadurch wurde zuvor ein Ende des Versuchssteps nicht ausgelöst, was letztendlich in einen Smalltalkfehler endete, wenn keine weiteren Bilder mehr geladen werden konnten. Das Ende des Versuchssteps ist nun unabhängig von der Zieleinfahrt und wird durch einen speziellen Farbcode am linken Streckenbildrand ausgelöst. Zusätzlich wurden einige Erweiterungen der Strecke beim Fahren mit der Tastatur berücksichtigt.

step:punkt: *geändert*

Parameter:

Name: schritt

Bedeutung: Geschwindigkeit in Pixel pro Morphstep, die die Strecke nach unten bewegt werden soll.

Name: koordinate

Bedeutung: neu aus den Joystickeingaben berechnete Koordinaten des Fahrobjektes

Beschreibung: Vorbeifahren am Ziel ist nicht mehr möglich. Siehe dazu auch die Beschreibung der Methode step.

dynamic *neu*

Beschreibung: Gibt das Objekt des dynamischen Hindernisses zurück.

dynamicBuild *neu*

Beschreibung: Gibt die Variable dynamicBuild zurück.

dynamicObstacle: *neu*

Parameter:

Name: bool

Bedeutung: true: ein dynamisches Hindernis soll erstellt werden, false: es soll kein dynamisches Hindernis erstellt werden

Beschreibung: Setzt die Variable dynamicObstacle auf true oder false.

slalom25: *neu*

Parameter:

Name: variation

Bedeutung: Slalomvariation, das erste Hindernis wird rechts (0) oder links (1) der Straße gesetzt.

Beschreibung: ersetzt place25: Anstelle eines Hindernisses werden zwei in Form eines Slaloms gesetzt, wobei zwei Variationen durch den Parameter unterschieden werden können. Die Platzierung erfolgt mit einer Pfadabdeckung von 25%.

slalom50: *neu*

Parameter:

Name: variation

Bedeutung: Slalomvariation, das erste Hindernis wird rechts (0) oder links (1) der Straße gesetzt.

Beschreibung: ersetzt place75: wie slalom25, die Platzierung der Hindernisse erfolgt aber mit einer Pfadabdeckung von 50%.

slalomObstacleDarkGray *neu*

Beschreibung: Gibt das Objekt des rechten Hindernisses bei einer links-rechts Platzierung zurück. Die Benennung der Variable soll es vereinfachen, das entsprechende Bild des Hindernisses zuordnen zu können. Die Platzierung wurde nicht berücksichtigt.

slalomObstacleDarkGreen *neu*

Beschreibung: Gibt das Objekt des linken Hindernisses bei einer rechts-links Platzierung zurück.

slalomObstacleLightGray *neu*

Beschreibung: Gibt das Objekt des linken Hindernisses bei einer links-rechts Platzierung zurück.

slalomObstacleLightGreen *neu*

Beschreibung: Gibt das Objekt des rechten Hindernisses bei einer rechts-links Platzierung zurück.

slalomVariation: *neu*

Parameter:

Name: int

Bedeutung: 0: rechts links Platzierung der Hindernisse eines Slaloms, 1: links rechts Platzierung der Hindernisse eines Slaloms

Beschreibung: Setzt die Variable slalomVariation auf 0 oder 1.

static25lr *neu*

Beschreibung: Gibt die Variable static25lr zurück.

static25lr: *neu*

Parameter:

Name: bool

Bedeutung: true: es existiert ein Slalom mit 25% Pfadabdeckung und dem ersten Hindernis auf der linken Seite, false: es existiert kein solcher Slalom

Beschreibung: Setzt die Variable static25lr auf true oder false.

static25rl *neu*

Beschreibung: Gibt die Variable static25rl zurück.

static25rl: *neu*

Parameter:

Name: bool

Bedeutung: true: es existiert ein Slalom mit 25% Pfadabdeckung und dem ersten Hindernis auf der rechten Seite, false: es existiert kein solcher Slalom

Beschreibung: Setzt die Variable static25rl auf true oder false.

static50lr *neu*

Beschreibung: Gibt die Variable static50lr zurück.

static50lr: *neu*

Parameter:

Name: bool

Bedeutung: true: es existiert ein Slalom mit 50% Pfadabdeckung und dem ersten Hindernis auf der linken Seite, false: es existiert kein solcher Slalom

Beschreibung: Setzt die Variable static50lr auf true oder false.

static50rl *neu*

Beschreibung: Gibt die Variable static50rl zurück.

static50rl: *neu*

Parameter:

Name: bool

Bedeutung: true: es existiert ein Slalom mit 50% Pfadabdeckung und dem ersten Hindernis auf der rechten Seite, false: es existiert kein solcher Slalom

Beschreibung: Setzt die Variable static50rl auf true oder false.

Unveränderte Instanzmethoden:
place25, *place75*, *start*, *stop*, *writeLog*, *xPos*, *yPos*

A.1.2 *ATEOCarNoFb*

INSTANZVARIABLEN

tachometer *geändert*

Typ: PCPieChartMorph

Beschreibung: Umbenennung von *pie*

unveränderte Instanzvariablen: *carName*, *collisionGrade*, *szaehler*

entfernte Instanzvariablen: *carColl*, *cr*, *fertig*, *tab*, *vStop*, *vTyp*

INSTANZMETHODEN

tachometer *geändert*

Beschreibung: Umbenennung von *pie*

feedback *geändert*

Beschreibung: Unterscheidung der zwei Slalomhindernisse bei der Kollision durch Aufruf der neuen Methode collision:

startpos *geändert*

Parameter:

Name: apoint

Bedeutung: Gibt die neue Position des Fahrobjektes an.

Beschreibung: pie wurde in tachometer umbenannt.

trackerbauen *geändert*

Beschreibung: pie wurde in tachometer umbenannt.

collisionGrade: *neu*

Parameter:

Name: value

Bedeutung: Gibt an, welcher Sensor die Kollision gemeldet hat.

Beschreibung: Setzt die Variable collisionGrade.

unveränderte Instanzmethoden: *aktpos, aktposkorr, carName, collisionGrade, fbsensoren, initialize, sensor1,..., sensor9*
entfernte Instanzmethoden: *fbBed, fertig*

A.1.3 *ATEOConfig*

INSTANZVARIABLEN unveränderte Instanzvariablen: *config, versuchsgruppe, geschlecht, operateur, order, team, teamnummer, vpMB1, vpMB2, startMB, steuerButton*
entfernte Instanzvariablen:
minus,..., minus5,minus7,...,minus10, plus,..., plus5,plus7,...,plus10, minusButton,...,minusButton5,minusButton7,..., minusButton10, plusButton,..., plusButton5,plusButton7,..., plusButton10, name,...,name5,name7,..., name10, nameString,..., nameString5,nameString7,..., nameString10, updateString,..., updateString5,updateString7,..., updateString10, value,...,value5,value7,..., value10, val,...,val5,val7,..., val10, measurementObj, inputManipulationObj, anteilreduzierung, button, checkbox1, checkbox2, checkboxText, checkboxText1, checkboxText2, text

INSTANZMETHODEN

begin *geändert*

Beschreibung: Diese Methode ist nun eine Klassenmethode und kann ohne explizite Erstellung eines Objektes aufgerufen werden.

start *geändert*

Beschreibung: Diese Methode ist nun eine Klassenmethode und wird bei Drücken des begin Buttons aufgerufen.

initialize *geändert*

Beschreibung: Die einzelnen Elemente des Konfigurationsmenüs werden nun als ATEOConfigElement erstellt. 14 globale Variablen werden mit nil initialsiert.

contentOfTeam *neu*

Beschreibung: Gibt den Wert des Inputmanipulationsvariante aus der Konfiguration zurück.

contentOfOrder *neu*

Beschreibung: Gibt den Wert der Inputmanipulationsreihenfolge aus der Konfiguration zurück.

unveränderte Instanzmethoden: *config, go, steuerung*
entfernte Instanzmethoden: *value7, value8, value9*

A.1.4 *ATEOConfigElement*

INSTANZVARIABLEN

value *neu*

Typ: Integer

Beschreibung: Die Variable value beinhaltet den im Konfigurationsmenüelement eingestellten Wert.

configElement *neu*

Typ: AlignmentMorph

Beschreibung: configElement ist ein vollständiges Konfigurationsmenüelement, welches aus mehreren Submorphen besteht.

INSTANZMETHODEN

configElement *neu*

Beschreibung: Gibt die Instanzvariable configElement zurück.

valueOfMorph *neu*

Beschreibung: Gibt den im Konfigurationsmenüelement eingestellten Wert zurück.

position: contentName: startValue: *neu*

Parameter:

Name: aPoint

Bedeutung: Position des Konfigurationsmenüelementes

Name: aString

Bedeutung: Name des Konfigurationsmenüelementes, welcher als Label angezeigt wird.

Name: anInteger

Bedeutung: Startwert des Konfigurationsmenüelementes

Beschreibung: Diese Methode erstellt ein Konfigurationsmenüelement und initialisiert es mit den Werten der Parameter.

KLASSENMETHODEN

elementPosition: contentName: startValue: *neu*

Parameter:

Name: aPoint

Bedeutung: Position des Konfigurationsmenüelementes

Name: aString

Bedeutung: Name des Konfigurationsmenüelementes, welcher als Label angezeigt wird.

Name: anInteger

Bedeutung: Startwert des Konfigurationsmenüelementes

Beschreibung: Parametrisierter Konstruktor, welcher die Parameter an die Methode position: contentName: startValue: weiterleitet.

A.1.5 *ATEOInputmanipulation*

INSTANZVARIABLEN unveränderte Instanzvariablen: *reduce*, *increase*, *amountOfReduction*, *reduceMBIArray*, *amountOfReductionArray*, *reduceMWI*, *lprocess*
entfernte Instanzvariablen: *amountOfReductionIndex*, *reduceMWIIndex*

INSTANZMETHODEN

intializeReductionArrays *geändert*

Beschreibung: Ehemals initializeReduction bezieht diese Methode Daten aus dem Konfigurationsmenü mit Hilfe von neuen Zugriffsmethoden.

initialize *geändert*

Beschreibung: Die Initialisierung von amountOfReductionIndex und reduceMWIIndex wurde entfernt.

startReduction *geändert*

Beschreibung: Die Erhöhung von amountOfReductionIndex und reduceMWIIndex wurde entfernt.

amountOfReduction *neu*

Beschreibung: Gibt die Höhe der Inputmanipulation zurück, damit sie im Folgenden formatiert in die Logdatei geschrieben werden kann.

unveränderte Instanzmethoden: *increaseMWI1*, *increaseMWI2*, *reduceMWI1*, *reduceMWI2*, *start*, *startIncrease*, *startReduction*, *step*, *stop*
entfernte Instanzmethoden: *stepTime*

A.1.6 *ATEOInstruction*

INSTANZVARIABLEN

endInstructionArray *neu*

Typ: Array

Beschreibung: Array der Größe 16 mit den Texten der 16 Endinstruktionen

startInstructionArray *neu*

Typ: Array

Beschreibung: Array der Größe 16 mit den Texten der 16 Startinstruktionen

panel *neu*

Typ: AlignmentMorph

Beschreibung: Oberster Instruktionsmorph, welcher die Submorphe window, button und instructionText beinhaltet.

font *neu*

Typ: TextFontReference

Beschreibung: Beinhaltet den Font für die Instruktionen: Font: Accujen Größe: 18 Hervorhebung: Fett.

button *neu*

Typ: SimpleButtonMorph

Beschreibung: Beinhaltet den grünen Button mit der Beschriftung 'Weiter' zum Schliessen des Instruktionsfensters und Starten oder Beenden des Versuchssteps.

instructionText *neu*

Typ: PluggableTextMorph

Beschreibung: Beinhaltet den Text der Instruktion.

INSTANZMETHODEN

continueEnd *neu*

Beschreibung: Ereignisbehandlung für das Drücken des Buttons in Endinstruktionsfenstern

continueStart *neu*

Beschreibung: Ereignisbehandlung für das Drücken des Buttons in Startinstruktionsfenstern

initialize *neu*

Beschreibung: Konstruktor zum Erstellen des Instruktionsfensters und des weiter-Buttons

panel *neu*

Beschreibung: Zugriffsmethode für den Hauptmorph des Instruktionsfensters

showOnEnd: *neu*

Parameter:

Name: stepNumber

Bedeutung: Gibt an, welcher Versuchsstep aktuell ist und welcher Instruktionstext angezeigt werden soll.

Beschreibung: Diese Methode zeigt das Instruktionsfenster mit dem richtigen Text für das Beenden eines Versuchssteps an und definiert das Ereignis des Buttons.

showOnStart: *neu*

Parameter:

Name: stepNumber

Bedeutung: Gibt an, welcher Versuchsstep aktuell ist und welcher Instruktionstext angezeigt werden soll.

Beschreibung: Diese Methode zeigt das Instruktionsfenster mit dem entsprechenden Text vor Beginn eines Versuchssteps an und definiert das Ereignis des Buttons.

A.1.7 *ATEOLog*

INSTANZVARIABLEN

trackedit *neu*

Typ: String

Beschreibung: trackedit gibt an, welche Veränderung auf einem Streckenteil vorliegt und in die Logdatei geschrieben werden soll.

unveränderte Instanzvariablen: *startzeit, letzteZeit*
entfernte Instanzvariablen: *cr*
aus globalen Variablen neu entstandene Instanzvariablen: *logObj, logfile, logfileWriter, rootObj*

INSTANZMETHODEN

initialize *geändert*

Beschreibung: Die Variable trackedit wird mit null initialisiert.

prepareLogEntry: summschritt: schritt: posX: leftright1: leftright2: updown1: updown2: zeitms: *geändert*

Parameter:

Name: frand

Bedeutung: Farbcode an der linken Seite der Strecke

Name: summschritt

Bedeutung: Summe der Schritte in Pixel pro Morphstep, die die Strecke bereits nach unten verschoben wurde

Name: schritt

Bedeutung: aktuelle Geschwindigkeit in Pixel pro Morphstep

Name: posX

Bedeutung: X-Koordinate des Fahrobjektes

Name: leftright1

Bedeutung: horizontale Auslenkung des Joysticks von MWB1

Name: leftright2

Bedeutung: horizontale Auslenkung des Joysticks von MWB2

Name: updown1

Bedeutung: vertikale Auslenkung des Joysticks von MWB1

Name: updown2

Bedeutung: vertikale Auslenkung des Joysticks von MWB2

Name: zeitms

Bedeutung: aktuelle Systemzeit von Squeak

Beschreibung: Bestimmen und Loggen der Hinderniskoordinaten in den neuen Logdateieinträgen FirstStaticObstacleXCoordinate, FirstStaticObstacleYCoordinate, SecondStaticObstacleXCoordinate, SecondStaticObstacleYCoordinate, DynamicObstacleXCoordinate und DynamicObstacleYCoordinate. Außerdem wurde der Logdateieintrag TrackTrait in TrackEdit umbenannt.

resetEventAndBalance *geändert*

Beschreibung: eventSound, evenPopUp, balance sind nicht mehr global und müssen über Zugriffsmethoden zurückgesetzt werden, zusätzlich werden auch collisionGrade und trackedit auf null zurückgesetzt

trackedit: *neu*

Parameter:

Name: value

Bedeutung: value ist eine Fließkommazahl, welche die unterschiedlichen Erweiterungen kodiert.

Beschreibung: Anhand des Parameters value wird die aktive Streckenerweiterung ermittelt und eine entsprechende Zeichenkette in trackedit gespeichert.

unveränderte Instanzmethoden: *addLogElement:, start, rot:*
Es spiegelt sich in folgenden Methoden wider, dass logObj, logfile, logfileWriter, rootObj als Instanzvariablen neu definiert wurden: *closeLog, createLog:, initialize, writeLog*

A.1.8 *ATEOOperateur*

KLASSENINSTANZVARIABLEN Die alleinige Klassenvariable ActivePopup blieb unverändert.

balance *neu*

Typ: String

Beschreibung: Diese Instanzvariable speichert den Stereokanal, welcher vom Operateur ausgewählt wurde. Zuvor wurde dies in der globalen Variable Balance gespeichert.

KLASSENMETHODEN

soundAufBeide *geändert*

Beschreibung: Der in balance zu speichernde Wert wurde umdefiniert auf '3'.

soundAufLinks *geändert*

Beschreibung: Der in balance zu speichernde Wert wurde umdefiniert auf '1'.

soundAufRechts *geändert*

Beschreibung: Der in balance zu speichernde Wert wurde umdefiniert auf '2'.

balance *neu*

Beschreibung: Gibt den Wert von balance zurück.

balance: *neu*

Parameter:

Name: string

Bedeutung: Mögliche Werte sind zur Zeit: '1' (linker Stereokanal), '2' (rechter Stereokanal), '3' (beide Stereokanäle)

Beschreibung: Setzt balance auf string.

unveränderte Instanz- und Klassenmethoden: *initialize* und *activePopup, autoPopupAndSounds, balanceVolumelinks:rechts:, playmp3, popup:, stopmp3:*

A.1.9 *ATEOPar*

INSTANZVARIABLEN

amountOfReductionIndex *neu*

Typ: Integer

Beschreibung: Index in den Array für die Reduktionswerte der Inputmanipulation

reduceMWIIndex *neu*

Typ: Integer

Beschreibung: Index in den Array für den zu reduzierenden MWB

measurementStep *neu*

Typ: Boolean

Beschreibung:

- true: es handelt sich um einen Messschritt und am Ende des Versuchssteps wird der gemessene Wert durch drei dividiert
- false: kein Messschritt. Diese Variable ist nötig, um zu verhindern, dass nach jedem Versuchsschritt die Geschwindigkeit des nächsten dynamischen Hindernisses gedrittelt wird.

unveränderte Instanzvariablen: *imageArray, bild1, bild2, bFile1, bFile2, aktidx, wechsel, seqColl, seqCollNr, parStatus, bildColl, cr, tab, sem, cumulativeSpeed, trackEdit, inputManipulation, measurement, started, stopped*
entfernte Instanzvariablen: *moveDynBarrier, moveStatBarrier*

INSTANZMETHODEN

cumulativeSpeed*geändert*

Beschreibung: Umbenennung von getCumulativeSpeed

bewegen: *geändert*

Parameter:

Name: anumber

Bedeutung: Anzahl an Pixel, die beide aktiven Streckenbilder nach unten verschoben werden sollen.

Beschreibung: Der Wert für trackedit wird berechnet und gespeichert. Außerdem wird nun hier mit Hilfe der Zeiger reduceMWIIndex und amountOfReductionIndex auf die Arrays der Inputmanipulation zugegriffen und die entsprechenden Werte gesetzt.

bilderStatus *geändert*

Beschreibung: In den Statusstring wurde trackedit am Ende eingefügt.

complexity *geändert*

Beschreibung: Die Aktivierung durch das Setzen der boolschen Variablen in ATEOAblaufsteuerung wird nun für den Slalom vorgenommen.

endStep *geändert*

Beschreibung: measurementStep wird benutzt, um die Mittelung der Messung nur vorzunehmen, wenn eine solche erfolgt ist. Damit wird verhindert, dass nach jedem Versuchschritt die Geschwindigkeit des nächsten dynamischen Hindernisses gedrittelt wird.

initialize *geändert*

Beschreibung: amountOfReductionIndex, reduceMWIIndex und measurementStep werden initialisiert

bFile1 *neu*

Beschreibung: Zugriffsmethode für bFile1, sie gibt den in bFile1 gespeicherten Bilddateinamen zurück.

bFile2 *neu*

Beschreibung: Zugriffsmethode für bFile2, sie gibt den in bFile2 gespeicherten Bilddateinamen zurück.

amountOfReductionIndex *neu*

Beschreibung: Gibt den Wert von amountOfReductionIndex zurück.

reduceMWIIndex *neu*

Beschreibung: Gibt den Wert von reduceMWIIndex zurück.

unveränderte Instanzmethoden: *bild1, bild2, bilderLaden:, cumulativeSpeed, started:, status, status:, stopped:, streckeBauen:, streckenLoeschen, streckenVerstecken, trackEdit*

A.1.10 *ATEOSpeedMeasurement*

INSTANZVARIABLEN unveränderte Instanzvariablen: *cumulativeSpeed, stepCount, measuredSpeed, lprocess*

INSTANZMETHODEN

step *geändert*

Beschreibung: Messung für die Tastatursteuerung wurde hinzugefügt.

unveränderte Instanzmethoden: *initialize, start, stop, stopMeasurement*
entfernte Instanzmethoden: *stepTime*

A.1.11 *ATEOTastaturabfragen*

INSTANZVARIABLEN Die beiden ehemals globalen Variablen *eventPopup* und *eventSound* wurden dieser Klasse als Instanzvariablen hinzugefügt.
unveränderte Instanzvariablen: *aktuellerSound*

INSTANZMETHODEN Durch die neuen Instanzvariablen *eventPopup* und *eventSound* wurde in den Methoden popups: und soundSpielen: das Vorkommen der gleichnamigen globalen Variablen durch diese ersetzt. In initialize werden sie mit 0.0 vorbelegt. Alle anderen Methoden blieben unverändert.

eventPopup *neu*

Beschreibung: Gibt den Wert von eventPopup zurück.

eventPopup: *neu*

Parameter:

Name:string

Bedeutung: Enthält die Aktivierung eines visuellen Hinweises kodiert in einen für den jeweiligen Hinweis einzigartigen String.

Beschreibung: Setzt die Variable eventPopup auf string.

eventSound *neu*

Beschreibung: Gibt den Wert von eventSound zurück.

eventSound: *neu*

Parameter:

Name:string

Bedeutung: Enthält die Aktivierung eines auditiven Hinweises kodiert in einen für den jeweiligen Hinweis einzigartigen String.

Beschreibung: Setzt die Variable eventSound auf string.

A.1.12 *ATEOVuSt*

INSTANZVARIABLEN

instructionDisplay *neu*

Typ: ATEOInstruction

Beschreibung: Diese Variable enthält das Instuktionsfenster.

unveränderte Instanzvariablen: *feedback1*, *endmorph*, *feedback2*, *rz*, *sem*, *vZuord*, *vbedStep1*, *vbedStep2*, *vbedStep3*, *vbedStep4*, *vseqstep2*, *stepcoll*, *mbcoll*, *streckecoll*, *anteilcoll*, *instrcoll*, *typcoll*, *tab*, *aktiverMB*
entfernte Instanzvariablen: *bEnde*, *bReset*, *vZiel*

INSTANZMETHODEN

fertig *geändert*

Beschreibung: Das Anzeigen des Endinstruktionsfenster wurde eingefügt.

nextStep *geändert*

Beschreibung: Das Anzeigen des Startinstruktionsfenster wurde eingefügt.

quit *geändert*

Beschreibung: Das Instruktionsfenster wird zerstört. Die Methode pie wurde in tachometer umbenannt.

start *geändert*

Beschreibung: Das Instruktionsfenster wird erstellt und der Quellcode für VuMorphStepping und des Methodenrufes initBilderUndSounds wurden entfernt.

streckenReset *geändert*

Beschreibung: Die Methode pie wurde in tachometer umbenannt.

unveränderte Instanzmethoden: *aktivenMBbestimmen*, *anteilbestimmen*, *aufgabensequenz*, *conf*, *delEndmorph:*, *ende*, *endMorphAusgeben*, *feedback1close*, *feedback2close*, *initBilderUndSounds*, *morphEinlesen*, *sequenz*, *sperreButton*, *vpnZuordnen*

A.1.13 *ATEOrealJoysticksStepping*

INSTANZVARIABLEN Alle Variablen dieser Klasse sind unverändert geblieben.

INSTANZMETHODEN

initialize

Beschreibung: Initialisierung von logCount wurde entfernt

step

Beschreibung: Die Formel zur Berechnung der Geschwindigkeit aus Joystickeingaben wurde vereinfacht und umstrukturiert, so dass die Lesbarkeit und das Verständnis erleichtert werden konnte.

unveränderte Instanzmethoden: *blau:*, *gruen:*, *rot:*, *schritt*, *start*, *stepTime*, *stop*, *writeLog*

A.2 SAM 2.0

Dieses Kapitel dokumentiert die im Vergleich zu SAM 1.5 geänderten, entfernten und neuen Methoden und Variablen.

A.2.1 *ATEOAblaufsteuerung*

INSTANZVARIABLEN unveränderte Instanzvariablen: *collisionCoordinates, dynamic, dynamicBuild, dynamicObstacle, firstStaticObstacleXCoord, firstStaticObstacleYCoord, lprocess, moveObstacle, Schrittweite, secondStaticObstacleXCoord, secondStaticObstacleYCoord, slalomObstacleDarkGray, slalomObstacleDarkGreen, slalomObstacleLightGray, slalomObstacleLightGreen, slalomVariation, static25lr, static25rl, static50lr, static50rl, staticObstacle, staticObstaclePlacing*
entfernte Instanzvariablen: *firstStaticObstacleXCoord, firstStaticObstacleYCoord, secondStaticObstacleXCoord, secondStaticObstacleYCoord, static, xPos, yPos*

INSTANZMETHODEN unveränderte Instanzmethoden: *collision, createDynamicObstacle, dynamic, dynamicBuild, dynamicBuild:, dynamicObstacle:, initialize, makeItComplex, schrittweite, schrittweite:, slalom25:, slalom50:, slalomObstacleDarkGray, slalomObstacleDarkGreen, slalomObstacleLightGray, slalomObstacleLightGreen, slalomVariation:, start, static25lr, static25lr:, static25rl, static25rl:, static50lr, static50lr:, static50rl, static50rl:, staticObstacle:, staticObstaclePlacing:, stop, writeLog*
entfernte Instanzmethoden: *place25, place50, xPos, yPos*

incSchrittweite: *geändert*

Parameter:

Name: inc

Bedeutung: Erhöhung der Geschwindigkeit um inc Pixel pro Morphstep

Beschreibung: Die Umrechnung der Geschwindigkeit für die Geschwindigkeitsanzeige wurde neu implementiert. Sie ist nun genauer als zuvor. Es wird nicht mehr grob die Geschwindigkeit (mögliches Intervall 0-20,24) mit fünf multipliziert und auf 100 abgebildet, sondern prozentual umgerechnet.

moveDynamicObstacle *geändert*

Beschreibung: Die erste Bedingung (dynamic top > 1200) wurde allgemeiner gefasst und tritt nun häufiger ein. Dadurch wird das dynamische Hindernis zuverlässiger entfernt.

slalom25: *geändert*

Parameter:

Name: variation

Bedeutung: Gibt die Variation der Slalomposition an.

Beschreibung: Die Positionskoordinaten werden nicht in Instanzvariablen vorgehalten sondern direkt bei der Positionierung verwendet. xPos und yPos entfallen dadurch.

step und step: *geändert*

Beschreibung: Bei beiden Methoden wurde die neue Instanzmethode endStep aus ATEOAblaufsteuerung anstelle der nun entfernten Methode endStep aus ATEOPar benutzt. Dadurch entfällt der Aufruf der Methode fertig aus ATEOVuSt.

endStep *neu*

Beschreibung: Diese Methode beendet den Versuchsstep, indem die Joystick- oder Tastatursteuerung angehalten und die Endinstruktion angezeigt wird.

A.2.2 *ATEOCarNoFb*

INSTANZVARIABLEN Unverändert zu SAM 1.5 sind folgende Variablen:
carName, collisionGrade, szaehler, tachometer

INSTANZMETHODEN Unverändert zu SAM 1.5 sind folgende Methoden:
aktpos, aktposkorr, carName, collisionGrade, fbsensoren, sensor1,..., sensor9, tachometer, feedback, startpos, trackerbauen, collisionGrade:

initialize *geändert*

Beschreibung: szaehler wird mit 0 initialisiert, was dazu führt, dass in der ersten Zeile der Logdatei kein nil Eintrag in der Spalte NrSensors erscheint. Die Auswertung wird dadurch erheblich erleichtert.

A.2.3 *ATEOConfig*

INSTANZVARIABLEN Alle Instanzvariablen (*config versuchsgruppe geschlecht operateur order team teamnummer vpMB1 vpMB2 startMB steuerButton*) blieben unverändert.

INSTANZMETHODEN

go *geändert*

Beschreibung: Initialisierung aller globalen Variablen des Systems außer Configure werden nun an dieser Stelle vorgenommen und in das Systemdictionary Smalltalk eingetragen.

initialize *geändert*

Beschreibung: Die Initialisierung globaler Variablen und der Konfigurationsmenüelemente order und team wurden entfernt.

unveränderte Instanz- und Klassenmethoden: *config, steuerung, initialize, begin, start*
entfernte Instanzmethoden: *contentOfTeam, contentOfOrder*

A.2.4 *ATEOConfigElement*

Die Klasse ATEOConfigElement wurde im Vergleich zu SAM 1.5 nicht verändert.

A.2.5 *ATEOInputmanipulation*

Die Klasse ATEOInputmanipulation wurde aus SAM 2.0 entfernt.

A.2.6 *ATEOInstruction*

INSTANZVARIABLEN

instructionButton1-5 *neu*

Typ: IconicButton

Beschreibung: Fünf Buttons, die die Auswahl der vermeindlich unterschiedlichen Teaminstruktionen implementiert.

background *neu*

Typ: AlignmentMorph

Beschreibung: background besitzt die zwei gegensätzlichen Instruktionsbilder *Genauigkeitsinstruktion.png* und *Schnelligkeitsinstruktion.png* als Submorphe.

unveränderte Instanzvariablen: *endMorph startMorph font button instructionText*
entfernte Instanzvariablen: *panel*

INSTANZMETHODEN

continueEnd *geändert*

Beschreibung: Die neu implementierte finish: Methode aus ATEOVuSt wird verwendet. instructionText als neuer Instruktionsmorph wird versteckt.

continueStart *geändert*

Beschreibung: Es wird der AlignmentMorph background gelöscht, damit die Bildinstruktion auf späteren Instruktionstafeln nicht mehr auftaucht.

initialize *geändert*

Beschreibung: Die fünf instructionButtons und instructionText als alleiniger Morph für das Instruktionsfenster werden erstellt. button wird nur erstellt, nicht aber angezeigt. startInstructionArray wurde auf eine Länge von 20 vergrößert.

showOnStart: *geändert*

Parameter:

Name: stepNumber

Bedeutung: Übergibt den Index, welcher für das Ermitteln des anzuzeigenden Instruktionstextes benötigt wird.

Beschreibung: Bei instructionIndex 3 und 6 soll die gegensetzliche Teaminstruktion angezeigt werden. Dafür wird das Ereignis des Buttons auf showTeamInstruction umgeschrieben. Die Variable instructionIndex aus ATEOVuSt wird in jedem Fall erhöht.

instructionText *neu*

Beschreibung: Gibt den Morph instructionText, der das Instruktionsfenster ist, zurück.

showTeamInstruction *neu*

Beschreibung: Bei instructionIndex 4 wird die Schnelligkeitsinstruktion erstellt bei instructionIndex 7 die Genauigkeitsinstruktion, in background integriert und im Instruktionsfenster angezeigt. Das Buttonereignis wird auf continue-Start gesetzt und instructionIndex erhöht.

start *neu*

Beschreibung: Buttonereignis für die Teaminstruktionsbuttons. Die fünf Instruktionsbuttons werden gelöscht, der grüne Weiter-Button in das Instruktionsfenster integriert sowie instructionIndex erhöht.

unverändert Instanzmethoden: *showOnEnd*

A.2.7 *ATEOLog*

INSTANZVARIABLEN unveränderte Instanzvariablen: *startzeit ,letzteZeit, trackedit, logObj, logfile, logfileWriter* entfernte Instanzvariablen: *rootObj*

INSTANZMETHODEN

prepareLogEntry: summschritt: schritt: posX: leftright1: leftright2: updown1: updown2: zeitms: *geändert*

Parameter:

Name: frand

Bedeutung: Farbcode an der linken Seite der Strecke

Name: summschritt

Bedeutung: Summe der Schritte in Pixel pro Morphstep, die die Strecke bereits nach unten verschoben wurde

Name: schritt

Bedeutung: aktuelle Geschwindigkeit in Pixel pro Morphstep

Name: posX

Bedeutung: X-Koordinate des Fahrobjektes

Name: leftright1

Bedeutung: horizontale Auslenkung des Joysticks von MWB1

Name: leftright2

Bedeutung: horizontale Auslenkung des Joysticks von MWB2

Name: updown1

Bedeutung: vertikale Auslenkung des Joysticks von MWB1

Name: updown2

Bedeutung: vertikale Auslenkung des Joysticks von MWB2

Name: zeitms

Bedeutung: aktuelle Systemzeit von Squeak

Beschreibung: SpeedDynamicObstacle als neuer Logdateieintrag wurde hinzugefügt.

unveränderte Instanzmethoden: *addLogElement: , trackedit:, closeLog, createLog:, initialize, resetEventAndBalance, writeLog, rot:*
entfernte Instanzmethoden: *start*

A.2.8 *ATEOOperateur*

Die Klasse ATEOOperateur wurde im Vergleich zu SAM 1.5 nicht verändert.

A.2.9 *ATEOPar*

INSTANZVARIABLEN

measuredSpeed *neu*

Typ: Float

Beschreibung: Umbenennung von cumulativeSpeed, da nur noch einmal je Versuchsstep gemessen wird und eine Kumulation der Messwerte nicht mehr vorgenommen wird.

unveränderte Instanzvariablen: *imageArray, bild1, bild2, bFile1, bFile2, aktidx, wechsel, seqColl, bildColl, , measurement, started, stopped*
entfernte Instanzvariablen: *trackEdit, cr, tab, sem, seqCollNr, parStatus, measurementStep, inputManipulation, amountOfReductionIndex, reduceMWIIndex, measurementStep*

INSTANZMETHODEN

bewegen: *geändert*

Parameter:

Name: anumber

Bedeutung: Anzahl an Pixel, die beide aktiven Streckenbilder nach unten verschoben werden sollen.

Beschreibung: Die Geschwindigkeitsmessung wurde angepasst, so dass es nur noch einmal pro Strecke aktiviert wird. Der Quellcode für die Aktivierung der Inputmanipulation wurde entfernt.

complexity *geändert*

Beschreibung: Ein neues ATEOSpeedMeasurement-Objekt wird immer erstellt, wenn auch ein dynamisches Hindernis erstellt werden soll. Der Dummy 200 wird nicht mehr benutzt und wurde entfernt.

initialize *geändert*

Beschreibung: Initialisierung von measurementStep wurde entfernt.

streckeBauen *geändert*

Beschreibung: Initialisierung von seqCollNr wurde entfernt.

measuredSpeed *neu*

Beschreibung: Umbenennung der Methode cumulativeSpeed in Analogie zur Instanzvariable selbigen Namens.

unveränderte Instanzmethoden: *bild1*, *bilderLaden:*, *cumulativeSpeed*, *started:*, *stopped:*, *streckenLoeschen*, *streckenVerstecken*, *cumulativeSpeed*, *bilderStatus*, *bFile1*, *bFile2*
entfernte Instanzmethoden: *bild2*, *trackEdit*, *endStep*, *status*, *status:*, *amountOfReductionIndex*, *reduceMWIIndex*

A.2.10 *ATEOSpeedMeasurement*

Die Klasse ATEOSpeedMeasurement wurde im Vergleich zu SAM 1.5 nicht verändert.

A.2.11 *ATEOTastaturabfragen*

INSTANZVARIABLEN

currentSound *neu*

Typ: aFile

Beschreibung: Umbenennung von aktuellerSound, aFile ist immer eine mp3-Datei.

credit *neu*

Typ: Integer

Beschreibung: Ehemals eine globale Variable, soll sie verhindern, dass mehr als ein Hinweis gleichzeitg gegeben werden kann.

unveränderte Instanzvariablen: *eventPopup*, *eventSound*

INSTANZMETHODEN

admin: *geändert*

Parameter:

Name: event

Bedeutung: modifizierter String mit dem Tastaturereignis, mögliche Werte: rCMD, eCMD, qCMD, lCMD, aCMD

Beschreibung: Aufruf der neuen Methode finish: aus ATEOVuSt bzw. Aufruf der neuen Methode endStep aus ATEOAblaufsteuerung für die unterschiedlichen Möglichkeiten einen Versuchsstep zu beenden.

aKeyDown: *geändert*

Parameter:

Name: event

Bedeutung: Tastaturereignis

Beschreibung: Jedes Vorkommen der globalen Variable Credit wurde durch die Instanzvariable credit ersetzt.

playSound *neu*

Parameter:

Name: event

Bedeutung: modifizierter String mit dem Tastaturereignis, mögliche Werte: a, b, c, 1,...,8

Beschreibung: Umbenennung der Methode soundSpielen:. Außerdem wurde jedes Vorkommen der globalen Variable Credit durch die Instanzvariable credit ersetzt.

initialize

Beschreibung: credit wird mit 0 initialisiert.

currentSound: *neu*

Parameter:

Name: file

Bedeutung: Referenz auf die abgespielte Audiodatei.

Beschreibung: Umbenennung der Methode aktuellenSoundSetzen: Setzt currendSound auf die Referenz einer abgespielten Audiodatei.

causePopUp : *neu*

Parameter:

Name: event

Bedeutung: modifizierter String mit dem Tastaturereignis, mögliche Werte: w, x, y, z

Beschreibung: Umbenennung der Methode popups:

control: *neu*

Parameter:

Name: event

Bedeutung: modifizierter String mit dem Tastaturereignis, mögliche Werte: ä, ö, ü, #

Beschreibung: Umbenennung der Methode steuerung:

unveränderte Instanzmethoden: *balance, countdown, , eventPopup, eventPopup:, eventSound:, eventSound*
entfernte Instanzmethoden: *aktuellerSound*

A.2.12 *ATEOVuSt*

INSTANZVARIABLEN

instructionIndex *neu*

Typ: Integer

Beschreibung: Wird als Index in das startInstructionsArray aus ATEOInstruction genutzt.

unveränderte Instanzvariablen: *rz, stepcoll, mbcoll, streckecoll, anteilcoll, instrcoll, typcoll, aktiverMB, instructionDisplay*
entfernte Instanzvariablen: *feedback1, endmorph, feedback2, sem, vZuord, vbedStep1, vbedStep2, vbedStep3, vbedStep4, vseqstep2, tab*

INSTANZMETHODEN

aktivenMBbestimmen *geändert*

Beschreibung: Die neue Methode finish: wurde integriert.

anteilbestimmen *geändert*

Beschreibung: Die neue Methode finish: wurde integriert.

finish: *neu*

Parameter:

Name: mode

Bedeutung: Art des Beendens

Beschreibung: Eine Methode für folgende Arten der Beendigung: Beenden des Experiments, Beenden eines Versuchsteps, Neustarten eines Versuchsteps

instructionDisplay *neu*

Beschreibung: Gibt den Wert der Variable instructionDisplay zurück.

instructionIndex *neu*

Beschreibung: Gibt den Wert der Variable instructionIndex zurück.

instructionIndexIncrease *neu*

Beschreibung: Erhöhen der Variable instructionIndex um eins.

nextStep *geändert*

Beschreibung: Initialisierung von ParID mit einem neuen ATEOCarNoFb Objekt ist nicht mehr nötig und wird stattdessen bei der Initialisierung der globalen Variable vorgenommen.

start *geändert*

Beschreibung: Der Quellcode der Methode conf wurde hier integriert und conf gelöscht. Das Initialisieren von globalen Variablen wird nun ausschließlich in ATEOConfig»go vorgenommen. instructionIndex und rz wird mit 1 initialisiert. instructionDisplay wird mit einem neuen ATEOInstruction-Objekt initialisiert. Die globale Variable JoyID wird bei angewählter Joysticksteuerung mit einem neuen ATEOrealJoysticksStepping-Objekt initialisiert. Die Streckenbilder werden geladen und zuletzt die Methode nextStep gerufen.

entfernte Instanzmethoden: *aufgabensequenz*, *conf*, *delEndmorph:*, *ende*, *endMorphAusgeben*, *feedback1close*, *feedback2close*, *initBilderUndSounds*, *morphEinlesen*, *sequenz*, *sperreButton*, *vpnZuordnen*, *streckenReset*, *quit*, *fertig*

A.2.13 *ATEOrealJoysticksStepping*

INSTANZVARIABLEN

vuStepLR *neu*

Typ: Float

Beschreibung: Ehemals global speichert diese Variable die Abschwächung der horizontalen Auslenkung des Joysticks. Der Wert ist konstant 1.5.

vuStepUD *neu*

Typ: Float

Beschreibung: Ehemals global speichert diese Variable die Abschwächung der vertikalen Auslenkung des Joysticks. Der Wert ist konstant 2.0.

unveränderte Instanzvariablen: *zeitms, summschritt, updown1, leftright1, updown2, leftright2, frand, posX, schritt*
entfernte Instanzvariablen: *cr tab zeit maxup msensor csensor sem posY logCount*

INSTANZMETHODEN

initialize *geändert*

Beschreibung: Initialisierung von cr, tab, sem, zeit, maxup, msensor, csensor wurde entfernt. Die Initialisierung der beiden neuen Instanzvariablen vuStickLR und vuStickUD wurde hinzugefügt.

step *geändert*

Beschreibung: Eine neue Berechnung der Geschwindigkeitsanzeige, welche genauer ist, wurde implementiert (siehe ATEOAblaufsteuerung»incSchrittweite). posY zuvor Instanzvariable wurde als temporäre Variable definiert, da der Wert dieser Variable außerhalb der Methode nicht abgefragt wird.

unveränderte Instanzmethoden: *rot:, schritt, start, stepTime, stop, writeLog*
entfernte Instanz- und Klassenmethoden: *blau:, gruen:* sowie *additionsToViewerCategories, authoringPrototype, descriptionForPartsBin, includeInNewMorphMenu, initialize, registerInFlapsRegistry, unload*

A.3 SAM SOLO TRACKING

SAM Solo Tracking wurde auf Basis von SAM 1.5 angepasst und in zwei Klassen modifiziert. Alle anderen Klassen blieben im Vergleich zu SAM 1.5 unverändert. Die Verbesserungen am Quellcode, die zu SAM 2.0 geführt haben, sind in dieser Version von SAM nicht umgesetzt.

A.3.1 *ATEOVuSt*

INSTANZVARIABLEN

speedFactorIndex *neu*

Typ: Integer

Beschreibung: Ein Zeiger in die Arrays speedFactor und speedFactorFrac aus ATEOrealJoysticksStepping.

INSTANZMETHODEN

nextStep *geändert*

Beschreibung: Die Variable speedFactorIndex wird um 1 erhöht.

speedFactorIndex *neu*

Beschreibung: Gibt den Wert der Variable speedFactorIndex zurück.

start *geändert*

Beschreibung: Die Variable speedFactorIndex wird mit 0 initialisiert.

A.3.2 *ATEOrealJoysticksStepping*

INSTANZVARIABLEN

speedFactor *neu*

Typ: Array

Beschreibung: Array mit der Manipulation der Maximalgeschwindigkeit je Versuchsstep.

speedFactorFrac *neu*

Typ: Array

Beschreibung: Array mit der um ein Zehntel kleineren Manipulation der Maximalgeschwindigkeit je Versuchsstep.

INSTANZMETHODEN

initialize *geändert*

Beschreibung: Die beiden Instanzvariablen speedFactor und speedFactorFrac werden initialisiert mit #(50 50 50 18 18 26 34 42 50) bzw. #(5 5 5 1.8 1.8 2.6 3.4 4.2 5).

step *geändert*

Beschreibung: In der Berechnungsformel für die Geschwindigkeit wird anstelle durch 50 zu teilen, durch den speedFactor geteilt. speedFactorIndex wird als Zeiger in das Array genutzt. speedFactorFrac wird bei der Umrechnung der Geschwindigkeit für die Geschwindigkeitsanzeige genutzt.

A.4 SAM FÜR DIE VERNETZUNG MIT OA

Diese Version von SAM basiert auf SAM 2.0. Es werden nur die Änderungen im Vergleich aufgeführt.

A.4.1 *ATEOAblaufsteuerung*

INSTANZVARIABLEN

coordinatesStream *neu*

Typ: UDP Socket

Beschreibung: Dieser Socket sendet die x-Koordinate an den Operateursarbeitsplatz. Genutzt wird der Port 58081.

operatorInterferenceStream *neu*

Typ: ATEOSocket

Beschreibung: Dieser TCP Socket empfängt die Eingriffe des Operateurs als Zahlencode. Genutzt wird der Port: 58082

speedStream *neu*

Typ: UDP Socket

Beschreibung: Dieser Socket sendet die Geschwindigkeit an den Operateursarbeitsplatz. Genutzt wird der Port 58080.

vmaxStream *neu*

Typ: TCP Socket

Beschreibung: Dieser TDP Socket empfängt die vom Operateur eingestellte Veränderung der Maximalgeschwindigkeit in Prozent. Genutzt wird der Port 58090.

vmax *neu*

Typ: Integer

Beschreibung: Speichert den vom Operateur gesendeten Prozentwert für die Veränderung der Maximalgeschwindigkeit.

INSTANZMETHODEN

coordinatesStream *neu*

Beschreibung: Gibt den Socket coordinatesStream zurück.

initialize *geändert*

Beschreibung: Die folgenden Streams und Sockets werden initialisiert: coordinatesStream (Socket UDP), operatorInterferenceStream (ATEOSocket), speedStream (Socket UDP) und vmaxStream (Socket TCP). An dieser Stelle wird auch die IP-Adresse des Zieles definiert und vmax initialisiert.

operatorInterferenceStream *neu*

Beschreibung: Gibt den Socket operatorInterferenceStream zurück.

speedStream *neu*

Beschreibung: Gibt den Socket speedStream zurück.

step:punkt: *geändert*

Beschreibung: In den beiden Prozessen process und vmax-Process, beide als temporäre Variablen deklariert, werden Daten vom Operateursarbeitsplatz empfangen. Auch werden die Geschwindigkeit und die x-Koordinate an den Operateursarbeitsplatz gesendet, wobei auch hier ein separater Prozess verwendet wird.

vmaxStream

Beschreibung: Gibt den Socket vmaxStream zurück.

vmax

Beschreibung: Gibt den Wert der Variable vmax zurück.

A.4.2 *ATEOConfig*

INSTANZVARIABLEN Es wurden keine Instanzvariablen neu definiert, geändert oder entfernt.

INSTANZMETHODEN

go *geändert*

Beschreibung: Die SOAP Nachricht loadConfigsAndGraphics wird hier an den Operateursarbeitsplatz versendet. Die Initialisierung der globalen Variable Ctrl wurde in die eigene Klassenmethode connectToOA verschoben.

KLASSENMETHODEN

connectToOA *neu*

Beschreibung: Der Aufruf dieser Methode erzeugt einen Button mit der Ereignismethode *connect*.

connect *neu*

Beschreibung: Ereignismethode des Buttons connectToOA. Es wird für die Initialisierung der Sockets gesorgt, indem die globale Variable Ctrl mit einem ATEOAblaufsteuerung-Objekt initialisiert wird.

A.4.3 *ATEOInstruction*

INSTANZVARIABLEN Es wurden keine Instanzvariablen neu definiert, geändert oder entfernt.

INSTANZMETHODEN

continueStart *geändert*

Beschreibung: Die SOAP Nachricht startTracking wird hier an den Operateursarbeitsplatz versendet. Außerdem wird beim Socket vmaxStream der Puffer geleert.

A.4.4 *ATEOVuSt*

INSTANZVARIABLEN Es wurden keine Instanzvariablen neu definiert, geändert oder entfernt.

INSTANZMETHODEN

finish: *geändert*

Parameter:

Name: mode

Bedeutung: Art des Beendens

Beschreibung: Die Sockets coordinatesStream, operatorInterferenceStream, speedStream und vmaxStream werden bei Beendigung eines Versuchsschrittes oder des gesamten Versuches geschlossen.

nextStep *geändert*

Beschreibung: Die SOAP Nachricht aktiverMB wird hier an den Operateursarbeitsplatz versendet.

A.4.5 *ATEOrealJoysticksStepping*

INSTANZVARIABLEN

denyLeft *neu*

Typ: Boolean

Beschreibung: Diese Variable speichert, ob die Joystickbewegung nach links blockiert werden soll (true) oder nicht (false).

denyRight *neu*

Typ: Boolean

Beschreibung: Diese Variable speichert, ob die Joystickbewegung nach rechts blockiert werden soll (true) oder nicht (false).

INSTANZMETHODEN

denyLeft: *neu*

Parameter:

Name: Boolean

Bedeutung:

- true: Joystickbewegung nach links blockieren
- false: Joystickbewegung nach links nicht blockieren

Beschreibung: Setzt den Wert der Variable denyLeft abhängig vom Paramter.

denyRight: *neu*

Parameter:

Name: Boolean

Bedeutung:

- true: Joystickbewegung nach rechts blockieren
- false: Joystickbewegung nach rechts nicht blockieren

Beschreibung: Setzt den Wert der Variable denyRight abhängig vom Paramter.

initialize *geändert*

Beschreibung: denyLeft und denyRight werden mit false initialisiert.

step *geändert*

Beschreibung: Die durch den Operateur eingestellte Veränderung der Maximalgeschwindigkeit in Prozent wird in die Berechnung der Geschwindigkeit mit einbezogen. Abhängig von den beiden Variablen denyLeft und denyRight werden die Joystickbewegungen nach links oder rechts blockiert.

A.4.6 *ATEOSocket*

INSTANZVARIABLEN

listener *neu*

Typ: TCP Socket

Beschreibung: Diese Variable speichert den TCP Socket, der auf Daten für die Eingriffe des Operateurs wartet. Es wird der Port 58082 genutzt.

INSTANZMETHODEN

close *neu*

Beschreibung: Schließt den Socket.

initialize *neu*

Beschreibung: Öffnet den Socket auf Port 58082.

isDestroyed *neu*

Beschreibung: Prüft, ob der Socket schon geschlossen ist und gibt als Ergebnis true oder false zurück.

receiveData *neu*

Beschreibung: Empfängt und verarbeitet die vom Operateursarbeitsplatz gesendeten Zahlencodes.

A.4.7 *OAService*

KLASSENINSTANZVARIABLEN

hostAddress *neu*

Typ: String

Beschreibung: Diese Variable speichert die IP Adresse des Zieles als String.

port *neu*

Typ: Integer

Beschreibung: Diese Variable speichert den Zielport als Integer. Der Standardport ist hier 8823.

targetObjectURI *neu*

Typ: String

Beschreibung: Speichert die URI des Zieles als String.

transport *neu*

Typ: Symbol

Beschreibung: Speichert das für die Kommunikation benutzte Protokoll.

INSTANZMETHODEN

callAktiverMB: *neu*

Parameter:

Name: aktiverMB

Bedeutung: Speichert den aktiven MWB, mögliche Werte 1 oder 2.

Beschreibung: SOAP Call für das Setzen des aktiven Mikroweltbewohners. Es wird beim Operateursarbeitsplatz die Methode *setAktiverMB* mit dem Parameter aktiverMB gerufen.

callLoadConfigAndGraphics *neu*

Beschreibung: SOAP Call für das Laden der Konfiguration und Streckenbilder im Operateursarbeitsplatz. Es wird die Methode *loadConfigsAndGraphics* gerufen.

callStartTracking *neu*

Beschreibung: SOAP Call für das Starten des Trackings der Streckenvorschau. Es wird beim Operateursarbeitsplatz die Methode *startTracking* gerufen.

hostAddress *neu*

Beschreibung: Setzen der Klasseninstanzvariable hostAddress. Es wird an die Klassenmethode weitergeleitet.

initialize *neu*

Beschreibung: Die vier Klasseninstanzvariablen hostAddress (IP Adresse vom entfernten Rechner), port (Standardport), targetObjectURI(leerer String) und transport (http Protokoll) werden initialisiert. Es werden die Klassenmethoden dafür genutzt.

port *neu*

Beschreibung: Setzen der Klasseninstanzvariable port. Es wird an die Klassenmethode weitergeleitet.

targetObjectURI *neu*

Beschreibung: Setzen der Klasseninstanzvariable targetObjectURI. Es wird an die Klassenmethode weitergeleitet.

transport *neu*

Beschreibung: Setzen der Klasseninstanzvariable transport. Es wird an die Klassenmethode weitergeleitet.

KLASSENMETHODEN

hostAddress *neu*

Beschreibung: Gibt den Wert der Klasseninstanzvariable hostAddress zurück.

hostAddress: *neu*

Parameter:

Name: address

Bedeutung: IP als String

Beschreibung: Setzt den Wert der Klasseninstanzvariable hostaddress.

port *neu*

Beschreibung: Gibt den Wert der Klasseninstanzvariable port zurück.

port: *neu*

Parameter:

Name: portNo

Bedeutung: Port als Integer

Beschreibung: Setzt die Klasseninstanzvariable port.

targetObjectURI *neu*

Beschreibung: Gibt die Klasseninstanzvariable targetObjectURI zurück.

targetObjectURI: *neu*

Parameter:

Name: uri

Bedeutung: URI als String

Beschreibung: Setzt die Klasseninstanzvariable targetObjectURI.

transport *neu*

Beschreibung: Gibt die Klasseninstanzvariable transport zurück.

transport: *neu*

Parameter:

Name: protocol

Bedeutung: Transportprotocol als Symbol

Beschreibung: Setzt die Klasseninstanzvariable transport.

A.5 OPERATEURSARBEITSPLATZ

Im Folgenden sollen die Anpassungen dokumentiert werden, die im Rahmen der Vernetzung vorgenommen wurden. Es werden nur jene Klassen genannt, die sich verändert haben im Vergleich zum Operateursarbeitsplatz so wie er in Schwarz (2009) dokumentiert ist.

A.5.1 *OAButtonsMorph*

INSTANZMETHODEN

backButtonAction *geändert*

Beschreibung: Der Zahlencode 42 wird über das Netzwerk an SAM geschickt.

forwardButtonAction *geändert*

Beschreibung: Der Zahlencode 41 wird über das Netzwerk an SAM geschickt.

leftButtonAction *geändert*

Beschreibung: Der Zahlencode 43 wird über das Netzwerk an SAM geschickt.

rightButtonAction *geändert*

Beschreibung: Der Zahlencode 44 wird über das Netzwerk an SAM geschickt.

soundAction1 *geändert*

Beschreibung: Der Zahlencode 7 wird über das Netzwerk an SAM geschickt.

soundAction2 *geändert*

Beschreibung: Der Zahlencode 8 wird über das Netzwerk an SAM geschickt.

soundAction3 *geändert*

Beschreibung: Der Zahlencode 9 wird über das Netzwerk an SAM geschickt.

soundAction4 *geändert*

Beschreibung: Der Zahlencode 10 wird über das Netzwerk an SAM geschickt.

A.5.2 *OAControlMorph*

KLASSENINSTANZMETHODEN

begin *neu*

Beschreibung: Diese Methode erstellt einen Button, mit dem zwei Elemente des Operateursarbeitsplatzes gestartet werden können.

start *neu*

Beschreibung: Instanzen von OASAMControlMorph und OAJoysticksOutputMorph werden erstellt und angezeigt.

A.5.3 *OAIconicSwitchButtonBothUsers*

INSTANZMETHODEN

doButtonAction *geändert*

Beschreibung: Der Zahlencode 6 wird über das Netzwerk an SAM geschickt.

A.5.4 *OAIconicSwitchButtonLeftUser*

INSTANZMETHODEN

doButtonAction *geändert*

Beschreibung: Der Zahlencode 5 wird über das Netzwerk an SAM geschickt.

A.5.5 *OAIconicSwitchButtonRightUser*

INSTANZMETHODEN

doButtonAction *geändert*

Beschreibung: Der Zahlencode 4 wird über das Netzwerk an SAM geschickt.

A.5.6 *OAInputMorph*

INSTANZMETHODEN

distributionMWI1ButtonAction *geändert*

Beschreibung: Abhängig vom Zustand des Buttons wird der Zahlencode 21, 22, 23, 24 über das Netzwerk an SAM geschickt.

distributionMWI2ButtonAction *geändert*

Beschreibung: Abhängig vom Zustand des Buttons wird der Zahlencode 20, 21, 22, 23 über das Netzwerk an SAM geschickt.

KLASSENINSTANZMETHODEN

setDirection *geändert*

Parameter:

Name: newDirection

Bedeutung: Dieser Parameter stellt den im Schieberegler eingestellten Wert zur Verfügung.

Beschreibung: Der beim Schieberegler eingestellte Wert wird auf einen Wertebereich von 100 und 0 umgerechnet. Abhängig vom umgerechneten Wert des Schiebereglers wird der Zahlencode 1, 2, 3 über das Netzwerk an SAM geschickt.

setSpeed *geändert*

Parameter:

Name: newSpeed

Bedeutung: Dieser Parameter stellt den im Schieberegler eingestellten Wert zur Verfügung.

Beschreibung: Der beim Schieberegler eingestellte Wert wird auf einen Wertebereich von 120 und 0 umgerechnet und im Anschluss über das Netzwerk an SAM gesendet.

A.5.7 *OASAMControlMorph*

KLASSENINSTANZVARIABLEN

coordinatesStream *neu*

Typ: UDP Socket

Beschreibung: Socket für die Übertragung der Fahrobjektkoordinaten

speedStream *neu*

Typ: UDP Socket

Beschreibung: Socket für die Übertragung der Geschwindigkeit des Fahrobjektes

INSTANZMETHODEN

initialize *geändert*

Beschreibung: Der globalen Variable OASocketStream wird ein Objekt der Klasse OASocket zugewiesen. Die Klasseninstanzmethode *initializeSocket* wird gerufen.

quit *geändert*

Beschreibung: Der Socket OASocketStream wird geschlossen und die eigene Klasseninstanzmethode *quit* wird gerufen.

KLASSENINSTANZMETHODEN

initializeSocket *neu*

Beschreibung: Zwei neue UDP Sockets namens coordinatesStream und speedStream werden erstellt und warten am Port 58080 und 58081 auf Daten.

quit *geändert*

Beschreibung: Die beiden UDP Sockets coordinatesStream und speedStream werden geschlossen.

receiveCorrdinates *neu*

Beschreibung: Daten am UDP Socket coordinatesStream werden abgerufen.

receiveSpeed *neu*

Beschreibung: Daten am UDP Socket speedStream werden abgerufen.

A.5.8 *OASAMData*

KLASSENINSTANZVARIABLEN

carCoordinates *neu*

Typ: Point

Beschreibung: Koordinaten des Fahrobjektes für die Streckenvorausschau.

speed *neu*

Typ: Integer

Beschreibung: Geschwindigkeit, mit der das Fahrobjekt in der Streckenvorausschau bewegt wird bzw. die Anzahl Pixel, die das Streckenbild nach unten verschoben wird.

aktiverMB *neu*

Typ: Integer

Beschreibung: Der im aktuellen Versuchsschritt steuernde Mikroweltbewohner.

KLASSENINSTANZMETHODEN

getSpeed *geändert*

Beschreibung: Die Geschwindigkeit wird von OASAMControlMorph abgerufen und in einen Integer umgewandelt, in speed gespeichert und zurückgegeben.

getCarCoordinates *geändert*

Beschreibung: Die x-Koordinate wird von OASAMControlMorph abgerufen und in einen Integer gewandelt. Ein Offset von 1500 @ 1100 wird addiert und damit der Integer in einen Punkt konvertiert. Dieser Wert wird in carCoordinates gespeichert und zurückgegeben.

getSpeedColor: *neu*

Parameter:

Name: currentSpeed

Bedeutung: Dieser Parameter stellt die von SAM gesendete Geschwindigkeit zur Verfügung.

Beschreibung: Die als Parameter übergebene Geschwindigkeit wird verwendet, um die Farbe des Schweifes zu berechnen. Der Parameter wurde nötig, da das Abrufen der Geschwindigkeit über das Netzwerk aus der Methode entfernt wurde.

A.5.9 *OASocket*

INSTANZVARIABLEN

stream *neu*

Typ: TCP Socket

Beschreibung: Socket für die Übertragung der Eingriffe des Operateurs.

streamVmax *neu*

Typ: TCP Socket

Beschreibung: Socket für die Übertragung der vom Operateur eingestellten Maximalgeschwindigkeit von SAM.

INSTANZMETHODEN

close *neu*

Beschreibung: Die beiden TCP Sockets stream und streamVmax werden geschlossen. streamVmax wird auf nil gesetzt.

initialize *neu*

Beschreibung: Die beiden Sockets stream und streamVmax werden als TCP Sockets erzeugt. Die Ports 58082 und 58090 werden benutzt sowie die IP Adresse des entfernten Rechners eingestellt.

stream *neu*

Beschreibung: Gibt eine Referenz auf den Socket stream zurück.

streamVmax *neu*

Beschreibung: Gibt eine Referenz auf den Socket streamVmax zurück.

B

HTA-TABELLE

Tabelle 6: HTA

SUPER-ORDINATE	TASK COMPONENT-OPERATION OR PLAN
0	*Operate SAM (as human operator)* Plan: S1: Do 1, 2
	1. Check GUI adjustments 2. Perform operation of SAM
1	*Check GUI adjustments* Plan: S1: Do 1.1, 1.2, 1.3, 1.4
	1.1 N3: Check whether every input/output/information window appears 1.2 N1: Locate a menu item 1.3 A2: Make adjustment (font, color, size, position, windows) 1.4 M2: Check changes of GUI
2	*Perform operation of SAM* Plan: S6: Repeat S1: Do 2.1, 2.2 until tracking ends
	2.1 Monitor SAM 2.2 Control SAM
2.1	*Monitor SAM* Plan: S1: Do 2.1.1, 2.1.2, 2.1.3
	2.1.1 N1: Locate a displayed system value 2.1.2 E2: Read the values of running system 2.1.3 M1: Detect deviance of system data
2.1.2	*E2: Read the values of runnig system* Plan: S5: Choose one of 2.1.2.1, 2.1.2.2, 2.1.2.3, 2.1.2.4, 2.1.2.5, 2.1.2.6, 2.1.2.7, 2.1.2.8, 2.1.2.9, 2.1.2.10, 2.1.2.11, 2.1.2.12, 2.1.2.13

SUPER-ORDINATE	TASK COMPONENT-OPERATION OR PLAN
	2.1.2.1 E2: Check MWI 1 Behavior (verbal, nonverbal)
	2.1.2.2 E2: Check MWI 2 Behavior (verbal, nonverbal)
	2.1.2.3 E2: Check Joystick moves (vertical, horizontal) of MWI 1
	2.1.2.4 E2: Check Joystick moves (vertical, horizontal) of MWI 2
	2.1.2.5 E2: Check mental workload of MWI 1
	2.1.2.6 E2: Check mental workload of MWI 2
	2.1.2.7 E2: Check MWI's driving: deviance from Track
	2.1.2.8 E2: Check MWI's driving: speed
	2.1.2.9 E2: Check track projection
	2.1.2.10 E2: Check MWI-input distribution
	2.1.2.11 E2: Check direction at track's crotch
	2.1.2.12 E2: Check obstacle characteristics (location, speed)
2.1.3	*M1: Detect deviance of system data*
	2.1.2.13 M3: Check the maximum speed of the tracking
	Plan: S5: Choose one of 2.1.3.1, 2.1.3.2, 2.1.3.3, 2.1.3.4, 2.1.3.5, 2.1.3.6, 2.1.3.7, 2.1.3.8, 2.1.3.9, 2.1.3.10, 2.1.2.11
	2.1.3.1 M1: Detect deviance of joystick moves of MWI 1/2
	2.1.3.2 M1: Detect deviance of driving object from ideal track
	2.1.3.3 M1: Detect deviance from optimal track's crotch
	2.1.3.4 M2: Anticipate possible collision with obstacle
	2.1.3.5 M1: Detect unavoidable collision with obstacle
	2.1.3.6 M2: Anticipate possible wrong decision at track crotch
	2.1.3.7 M1: Detect unavoidable wrong decision at track crotch
	2.1.3.8 M1: Detect deviance of mental workload of MWI 1/2
	2.1.3.9 M1: Detect deviance of Behavior (verbal, nonverbal) of MWI 1/2

SUPER-ORDINATE	TASK COMPONENT-OPERATION OR PLAN
	2.1.3.10 M1: Detect extreme speed (too slow / too fast) of MWI 1/2 2.1.3.11 M1: Detect deviance of driving object from optimal speed
2.2	*Control SAM* Plan: S5: Choose one of 2.2.1, 2.2.2
	2.2.1 A1: Mediated control 2.2.2 A1: Direct control
2.2.1	*A1: Mediated control* Plan: S1: Do 2.2.1.1 S5: Choose one of 2.2.1.2, 2.2.1.3
	2.2.1.1 N1: Locate the key for a signal (audio / popup) 2.2.1.2 A1: Send an audio signal to MWIs 2.2.1.3 A1: Send a visual signal to MWIs
2.2.1.2	*A1: Send an audio signal to MWIs* Plan: S1: Do 2.2.1.2.1, S5: Choose one of 2.2.1.2.2, 2.2.1.2.3, 2.2.1.2.4, 2.2.1.2.5, 2.2.1.2.6, 2.2.1.2.7, 2.2.1.2.8, 2.2.1.2.9
	2.2.1.2.1 A1: push button MWI1, MWI2 or MWI 1+2 2.2.1.2.2 A1: push button sound1 „Ihr Partner wird jetzt die Führung übernehmen" 2.2.1.2.3 A1: push button sound2 „Ihr Partner lenkt zu heftig" 2.2.1.2.4 A1: push button sound3 „Ihr Partner fährt etwas zu schnell" 2.2.1.2.5 A1: push button sound4 „Bitte fahren Sie etwas genauer" 2.2.1.2.6 A1: push button sound5 „Bitte übernehmen Sie die Führung" 2.2.1.2.7 A1: push button sound6 „Ihr Partner lenkt zu schwach" 2.2.1.2.8 A1: push button sound7 „Ihr Partner fährt etwas zu langsam" 2.2.1.2.9 A1: push button sound8 „Bitte fahren Sie etwas schneller"

SUPER-ORDINATE	TASK COMPONENT-OPERATION OR PLAN
2.2.1.3	*A1: Send a visual signal to MWIs* Plan: S5: Choose one of 2.2.1.3.1, 2.2.1.3.2, 2.2.1.3.3, 2.2.1.3.4
	2.2.1.3.1 A1: push button for speedup-popup 2.2.1.3.2 A1: push button for slowdown-popup 2.2.1.3.3 A1: push button for turn left popup 2.2.1.3.4 A1: push button for turn right popup
2.2.2	*A1: Direct control* Plan: S1: Do 2.2.2.1 S5: Choose one of 2.2.2.2, 2.2.2.3, 2.2.2.4, 2.2.2.5
	2.2.2.1 N1: Locate a button for a special action 2.2.2.2 A2: Change direction at track's crotch 2.2.2.3 A2: Change direction at an obstacle 2.2.2.4 A2: Change maximum speed 2.2.2.5 A2: Change MWI-input distribution
2.2.2.2	*A2: Change direction at track's crotch* Plan: S5: Choose one of 2.2.2.2.1, 2.2.2.2.2
	2.2.2.2.1 A2: move slider left (force driving object to left) 2.2.2.2.2 A2: move slider right (force driving object to right)
2.2.2.3	*A2: Change direction at an obstacle* Plan: S5: Choose one of 2.2.2.3.1, 2.2.2.3.2
	2.2.2.3.1 A2: move slider left (force driving object to left) 2.2.2.3.2 A2: move slider right (force driving object to right)
2.2.2.4	*A2: Change maximum speed* Plan: S5: Choose one of 2.2.2.4.1, 2.2.2.4.2
	2.2.2.4.1 A2: move slider left (slow down) 2.2.2.4.2 A2: move slider right (speed up)
2.2.2.5	*Change MWI-input distribution* Plan: S5: Choose one of 2.2.2.5.1, 2.2.2.5.2
	2.2.2.5.1 A2: move the input slider to MWI1 (increase MWI1-Input/decrease MWI2-Input) 2.2.2.5.2 A2: move the input slider to MWI2 (increase MWI2-Input/decrease MWI1-Input)

C

SMALLTALK/SQUEAK REZEPTE

C.1 PORTIERUNG IN EIN NEUES SQUEAK-IMAGE

C.1.1 *Zutaten*

- Squeak VM: Squeak.exe & SqueakFFIPrims.dll
- Squeak Sources z.B. SqueakV39.sources
- Squeak Image und Changes
  z.b. Squeak3.9u1.image & Squeak3.9u1.changes
- bei Universe Images: universetmp (Ordner)

für SAM 2.0:

- Countdown0-3.gif und CountdownStart.gif
- ELl.gif, ELr.gif, RLl.gif, RLr.gif, straight.gif
- SbAx260806.gif, SbBx260806.gif, SbCx260806.gif, SbD2x260806.gif, SbDx260806.gif, SbE2x260806.gif, SbEx260806.gif, SbF2x260806.gif, SbFx260806.gif, SbG2x260806.gif, SbGx260806.gif, SbH2x260806.gif, SbHx260806.gif, SbI2x260806.gif, SbIx260806.gif, SbJ2x260806.gif, SbJx260806.gif, SbK2x260806.gif, SbKx260806.gif, SbL2x260806.gif, SbLx260806.gif, SbMx260806.gif, SbNx260806.gif, SbOx260806.gif, SbPx260806.gif, SbQ2x260806.gif, SbQx260806.gif, SbRx260806.gif, SbSx260806.gif
- Bildl.jpg, Bildo.jpg, Bildr.jpg, Bildu.jpg
- block1-1.jpg, block1-2.jpg, block1-3.jpg, block1-4.jpg, block1.jpg
- Sound1-8.mp3
- Genauigkeitsinstruktion.png, Schnelligkeitsinstruktion.png
- Teaminstruktion1-5.png
- BilderExp3BaJe.txt, steps.txt
- Endinstruction1-17.txt, Instruction1-20.txt
- hauptabschnitt1-6.txt, hauptabschnitt_lang.txt
- lernstrecke.txt, testabschnitt.txt
- Changelog SAM 2.0.txt

für den Operateursarbeitsplatz noch zusätzlich:

- blueleft.png, blueright.png, greenup.png, reddown.png
- bothusers.png, leftuser.png, rightuser.png
- LeftUserIcon.png, RightUserIcon.png

C.1.2 *Zubereitung*

1. Im alten Squeak image die ATEO, OA oder AOF Kategorien mit Hilfe des Klassenbrowsers und dem Befehl file-out speichern. Die so entstandenen .st Dateien in den Ordner mit dem neuen Squeak image und den ATEO Dateien kopieren.

2. Squeak mit dem frischen image starten.

3. Mit der linken Maustaste das World-Menü öffnen und unter open... eine file list öffnen.

4. Der file list Browser ist standardmäßig im Wurzelverzeichnis des eigenen images. In diesem Verzeichnis die .st Dateien anklicken und in der sich dynamisch anpassenden Befehlleiste file-in auswählen. Diesen Vorgang solange wiederholen, bis alle gewünschten .st Dateien in das image geladen wurden.

5. Einige Klassenpakete sollen nun installiert werden. Für SAM werden *PieChartMorph* und mit Vernetzung noch *SoapCore* benötigt. Diese Pakete können entweder über den Universe Browser (linke Maustaste->World Menu->open->Universe Browser), den SqueakMap Package Loader (linke Maustaste->World Menu->open->SqueakMap Package Loader) oder von der Entwicklerwebsite geladen werden. Der Universe Browser ist nur bei speziellen universe-images vorhanden, der SqueakMap Package Loader ist in jedem Squeak image vorhanden. Im SqueakMap Package Loader kann das obere Eingabefeld als Suche benutzt werden. Mit der rechten Maustaste kann ein Menü aufgerufen werden, wo man unter anderem die Pakete aktualisieren kann und auch den Installationsbefehl findet.

6. Mit linke Maustaste->World Menu->open->workspace sollen nun die Buttons erstellt werden. Folgender Smalltalk-Code muss dazu in den Workspace kopiert und mit der Tastenkombination Alt/Strg/Apfeltaste + d ausgeführt werden: *ATEOConfig begin. ATEOConfig connectToOA* in SAM oder *OAControl begin* im Operateursarbeitsplatz. Der Button erstellt das Konfigurationsmenü bei SAM und startet den Operateursarbeitsplatz für den Betrieb in einem image für den Operateursarbeitsplatz.

7. Wenn man ein neues image für den Operateursarbeitsplatz erstellt, muss auch noch folgender Smalltalk-Code im workspace mit Alt/Strg/Apfeltaste + d ausgeführt werden: *OAService registerAll*. Damit werden die SOAP-Services dem image bekannt gemacht. Die Kommunikation mit SOAP funktioniert nicht ohne diese Registrierung.

8. Nachdem diese Schritte vollzogen wurden, muss das image gespeichert. Dazu linke Maustaste->World Menu->save ausführen oder eine der aufgeführten Alternativen.

C.2 VERSUCHSSTART VERNETZTER OPERATEURSARBEITSPLATZ

1. SAM und den Operateursarbeitsplatz auf den beiden Rechnern starten.

2. Die IP Adressen müssen angepasst sein. Hierzu muss an folgenden Stellen die aktuelle IP Adresse eingetragen werden: *ATEOAblaufsteuerung»initialize* und *OAService»initialize* auf Seiten von SAM sowie *OASocket»initialize* auf Seiten des Operateursarbeitsplatz. Dazu öffnet man einen Klassenbrowser mit linke Maustaste->World Menu->open->Browser und navigiert zu den entsprechenden Methoden.
3. Auf Seiten von SAM den connectToOA-Button drücken, um eine Verbindung mit dem Operateursarbeitsplatz zu initiieren.
4. Mit Hilfe des begin Buttons im Operateursarbeitsplatz die Eingriffsfenster starten und die Verbindung mit SAM vervollständigen.
5. In SAM beim Menü den Button Los drücken und der Versuch startet.

Sollte der entfernte Computer trotz korrekter IP-Adresse nicht gefunden werden, liegt das Problem wahrscheinlich an einer Firewall. Ausschalten oder Konfigurieren dieser ist hier die Lösung.

C.3 KONFIGURATION EINES VERSUCHS

C.3.1 *Die Strecke*

Alle in SAM verwendbaren Streckenbilder müssen in der Datei BilderExp3BaJe.txt aufgelistet sein. Neue Streckenbilder sollte man immer am Ende der Datei hinzufügen, da die Bilder in dieser Datei durchnummeriert werden und in weiteren Konfigurationsdateien für die Bilder nur noch diese Indices genutzt werden. Tabelle 7 listet alle zum Zeitpunkt des Schreibens benutzten Streckenbilder und ihre Indices auf.

In einem zweiten Schritt werden dann die einzelnen Strecken aus diesen Bildern zusammengestellt, indem die Indices dieser in weiteren Textdateien gespeichert werden. Die Bilder sind dabei genau aufeinander angepasst, so dass ein nahtloser Bildübergang gewährleistet ist. Der Dateiname ist frei wählbar, muss aber identisch in der Textdatei steps.txt gelistet werden. Das Ziel bzw. das Ende der Strecke ist dabei immer fest als *22;23;24;* zu konfigurieren.

C.3.2 *Die Versuchsteps*

Die Versuchssteps, also die einzelnen Strecken, werden in der Textdatei steps.txt konfiguriert. Hier kann angegeben werden, welche Strecke zu welchem Zeitpunkt im Ablauf erscheinen soll und ob ein oder beide Mikroweltbewohner steuern. Im folgenden soll eine Zeile mit ihren einzelnen Elementen beschrieben werden. Jede Zeile der steps.txt besteht aus 6 Teilen, die mit einem Semikolon voneinander getrennt werden.

- step number
  - Es wird jeder step durchnummeriert.
- Der Typ des trackings
  - Hier wird immer ATEO gespeichert, andere Werte sind möglich haben aber keinen Einfluss, da der Wert nicht von SAM ausgewertet wird.

INDEX	NAME DES STRECKENBILDES
1	SbAx260806.gif;
2	SbBx260806.gif;
3	SbCx260806.gif;
4	SbDx260806.gif;
5	SbD2x260806.gif;
6	SbEx260806.gif;
7	SbE2x260806.gif;
8	SbFx260806.gif;
9	SbF2x260806.gif;
10	SbGx260806.gif;
11	SbG2x260806.gif;
12	SbHx260806.gif;
13	SbH2x260806.gif;
14	SbIx260806.gif;
15	SbI2x260806.gif;
16	SbJx260806.gif;
17	SbJ2x260806.gif;
18	SbKx260806.gif;
19	SbK2x260806.gif;
20	SbLx260806.gif;
21	SbL2x260806.gif;
22	SbMx260806.gif;
23	SbNx260806.gif;
24	SbOx260806.gif;
25	SbPx260806.gif;
26	SbQx260806.gif;
27	SbRx260806.gif;
28	SbSx260806.gif;
29	gabelA.gif;
30	gabelA2.gif;
31	gabelB.gif;
32	RLr.gif
33	ELl.gif
34	ELr.gif
35	RLl.gif
36	straight.gif
37	SbQ2x260806.gif

Tabelle 7: Indexzahlen für die Streckenbilder

- Es wird die Steuergewalt gespeichert, die einzelnen möglichen Einträge bedeuten:
  - MB? : Es wird der im Konfigurationsmenü eingestellte Mikroweltbewohner für den ersten Versuchsstep ausgewählt.
  - MB= : Die Steuergewalt wird nicht abgegeben, sondern der zuletzt aktiv steuernde Mikroweltbewohner steuert erneut.
  - MBW : Diese Einstellung bewirkt einen Wechsel der Steuerungsgewalt, der aktive MWB wird getauscht.
  - MBKOOP: Beide Mikroweltbewohner haben die Steuergewalt.
- Streckendatei, in SAM 2.0 sind möglich (andere Dateinamen möglich soweit erstellt):
  - lernstrecke.txt
  - testabschnitt.txt
  - hauptabschnitt1.txt, ..., hauptabschnitt6.txt
  - hauptabschnitt_lang.txt
- Steuerinputverteilung, möglich sind 0-100% Joystickinput, in SAM 2.0 genutzte Einträge:
  - MB?100 : Der vor dem Start im Konfigurationsmenü festgelegte Mikroweltbewohner bekommt 100% des Joystickinputs.
  - MB= : Es fährt der Mikroweltbewohner vom Versuchsstep zuvor mit unverändertem Joystickinput.
  - MB1P100 : Mikroweltbewohner 1 bekommt 100% Joystickinput.
  - MB2P100 : Mikroweltbewohner 2 bekommt 100% Joystickinput.
  - MB12P50 : beide Mikroweltbewohner bekommen 50% Joystickinput.
- Instruktion
  - Dieser Wert ist immer IE, andere Werte sind möglich haben aber keinen Einfluss, da der Wert nicht von SAM ausgewertet wird.

C.3.3 *Die Instruktionen*

Die Instruktionen vor und nach jedem Versuch werden in den Textdateien Instruction1.txt, ..., Instruction20.txt bzw. Endinstruction1.txt, ..., Endinstruction17.txt gespeichert. Man muss sich allerdings an einige Einschränkungen halten.

- Die Textdateien müssen in UTF-8 Format abgespeichert werden. Dies leistet beispielsweise der Editor notepad von Microsoft nicht.
- Außerdem darf der Zeilenumbruch nicht nach dem Windows Standard abgespeichert werden, sondern nach dem MacOS/Unix Standard. Erreicht werden kann dies durch einen Editor, bei dem dies eingestellt werden kann. Die Dateien unter MacOS abzuspeichern sollte keine Probleme bereiten.

- Instruction1.txt sollte in der Regel leer bleiben, da fünf Buttons für die Auswahl der Teaminstruktion anstelle von Text angezeigt werden. Diese Buttons überlagern vorhandenen Text.
- Auf den Instruktionstafeln für Instruction4.txt und Instruction7.txt werden für jeden der beiden Mikroweltbewohner individuell die gegensätzlichen Anweisungen angezeigt. Dieses Bild wird zentriert, so dass über diesem Platz für Text vorhanden ist, aber nur begrenzt.
- Die Anzahl der Textdateien ist fest und kann nicht ohne Anpassungen im Quellcode erweitert werden, weil auch die Anzahl an angezeigten Instruktionen im Programm festgelegt ist.

D

QUELLCODEBEISPIELE

D.1 SOCKETPROGRAMMIERUNG: TCP

Server:

```
| listener data |

" Initialisierung des Sockets: Warten auf einen eingehenden
    Verbindungsversuch"
listener _ Socket newTCP.
listener listenOn: 60000.

"Empfangen von Daten"
data _ listener receiveData.

"Schließen des Sockets"
listener close.
```

Client:

```
| stream addr |

" Initialisierung des Sockets: Verbindungsaufbau mit wartendem
    Socket"
addr := (NetNameResolver addressForName: '111.11.11.1').
stream := SocketStream openConnectionToHost: addr port: 60000.

"Senden von Daten"
stream nextPutAllFlush: 'Hello World'

"Schließen des Sockets"
stream close.
```

D.2 SOCKETPROGRAMMIERUNG: UDP

Server:

```
| serverStream |

" Initialisierung  des Sockets"
  serverStream := Socket newUDP.
  serverStream setPort: 60000.

"Empfangen von Daten"
  serverStream isDestroyed
  ifFalse : [
    data := serverStream receiveData.
  ]
"Schließen des Sockets"
  serverStream closeAndDestroy
```

Client:

```
| addr clientStream data |

" Initialisierung  des Sockets"
  addr := (NetNameResolver addressForName: '111.11.11.1').
  clientStream := Socket newUDP.
  clientStream setPeer: addr port: 60000.

"Senden von Daten"
  clientStream sendData: 'Hello World'.

"Schließen des Sockets"
  clientStream closeAndDestroy
```

[Etoys] *Etoys*. – URL http://wiki.laptop.org/go/Etoys. – Letzter Zugriff: 02.07.2009

[Projektseminar] *Projekt: Mensch-Technik-Interaktion in Echtzeit (Projektseminar WS 2008/09)*. – URL https://www2.informatik.hu-berlin.de/swt/lehre/PR_MTI_0809/. – Letzter Zugriff: 02.07.2009

[Squeakland] *Squeakland - Home of Squeak Etoys*. – URL http://www.squeakland.org/. – Letzter Zugriff: 02.07.2009

[Akyildiz u. a. 2002] Akyildiz, I.F. ; Su, Weilian ; Sankarasubramaniam, Y. ; Cayirci, E.: A Survey on Sensor Networks. In: *IEEE Communications Magazine* 40 (2002), August, Nr. 8, S. 102– 114

[Baker 2000] Baker, Mark: Cluster Computing White Paper. In: *CoRR* cs.DC/0004014 (2000)

[Basili und Turner 1975] Basili, Victor ; Turner, Albert: Iterative Enhancement: A Practical Technique for Software Development. In: *IEEE Transactions on Software Engineering* 1 (1975), Dezember, Nr. 4, S. 390–396

[Boehm 1988] Boehm, Barry W.: A Spiral Model of Software Development and Enhancement. In: *IEEE Computer* 21 (1988), Mai, Nr. 5, S. 61–72

[Bothe und Hildebrandt 2008] Bothe, Klaus ; Hildebrandt, Michael: *ATEO Verhaltensspezifikation mit Erweiterungswünschen, ATEO-Version 1.0*. 2008

[Bothe u. a. 2008] Bothe, Klaus ; Hildebrandt, Michael ; Niestroj, Nicolas: *ATEO Verhaltensspezifikation mit Erweiterungswünschen, ATEO-Version 1.5*. 2008

[Braden 1989a] Braden, Robert: *RFC 1122: Requirements for Internet Hosts – Communication Layers*. Oktober 1989

[Braden 1989b] Braden, Robert: *RFC 1123: Requirements for Internet Hosts – Application and Support*. Oktober 1989

[Burandt 2007] Burandt, Maik: *ATEO - begleitende Arbeit zur Projektdurchführung*. 2007. – unveröffentlichte Studienarbeit an der Humboldt-Universität zu Berlin

[Cerf und Kahn 1974] Cerf, Vinton G. ; Kahn, Robert E.: A Protocol for Packet Network Intercommunication. In: *IEEE Transactions on Communications* 22 (1974), May, Nr. 5, S. 637–648

[Cohen u. a. 2004] Cohen, David ; Lindvall, Mikael ; Costa, Patricia: An introduction to agile methods. In: *Advances in Computers* 62 (2004), S. 2–67

[Coulouris u. a. 2002] Coulouris, George ; Dollimore, Jean ; Kindberg, Tim: *Verteilte Systeme: Konzepte und Design*. 3. Pearson Studium, 2002

[Deck] Deck, Diego G.: *rST - Remote Smalltalk*. – URL http://www.squeaksource.com/rST.html. – letzter Zugriff: 02.07.2009

[Erl 2005] Erl, Thomas: *Service-Oriented Architecture: Concepts, Technology, and Design*. 1. Prentice Hall, 2005

[Foster u. a. 2006] Foster, Ian ; Kishimoto, Hiro ; Savva, Andreas: *The Open Grid Services Architecture, Version 1.5*. Juli 2006. – GGF Informational Document GFD-I.080

[Gross und Nachtwei 2006] Gross, Barbara ; Nachtwei, Jens: Assistenzsysteme effizient entwickeln und nutzen - Die Mikrowelt als Methode zur Wissensakquisition für Entwickler und Operateure. In: Grandt, M. (Hrsg.) ; Bauch, A. (Hrsg.): *Cognitive Systems Engineering in der Fahrzeug- und Prozessführung*. 1. Deutsche Gesellschaft f. Luft- u. Raumfahrt, 2006 (DGLR-Bericht 2006/2), S. 75–88

[Hildebrandt 2009] Hildebrandt, Michael: *Analyse und Evaluierung der Architektur des ATEO-Systems*. 2009. – unveröffentlichte Studienarbeit an der Humboldt-Universität zu Berlin

[Hildebrandt u. a. 2009] Hildebrandt, Michael ; Kain, Saskia ; Kesselring, Kai ; Nachtwei, Jens ; Niestroj, Nicolas ; Schwarz, Hermann: *Die Hierarchische Aufgabenanalyse im Babel interdisziplinärer Softwareentwicklungsprojekte*. 2009. – Zeitschrift für Arbeitswissenschaft - eingereicht

[Ingalls u. a. 1997] Ingalls, Dan ; Kaehler, Ted ; Maloney, John ; Wallace, Scott ; Kay, Alan: Back to the future: the story of Squeak, a practical Smalltalk written in itself. In: *OOPSLA '97: Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. New York, NY, USA : ACM, 1997, S. 318–326. – ISBN 0-89791-908-4

[ISO/IEC 10746-1 1998] ISO/IEC 10746-1: *Information technology – Open Distributed Processing – Reference Model: Overview*. 1998

[ISO/IEC 7468-1 1994] ISO/IEC 7468-1: *Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model*. 11 1994

[Jacob u. a. 2005] Jacob, Bart ; Brown, Michael ; Fukui, Kentaro ; Trivedi, Nihar: *Introduction to Grid Computing*. 1. IBM Redbooks, 2005

[Kesselring 2009] Kesselring, Kai: *Entwicklung einer Softwarekomponente zur Systemprozessüberwachung und -kontrolle in einer psychologischen Versuchsumgebung*. 2009. – Diplomarbeit

[Klimas u. a. 1996] Klimas, Edward ; Skublics, Suzanne ; Thomas, David A.: *Smalltalk With Style (Volume One)*. 1. Prentice Hall, 1996

[Kohler u. a. 2006] Kohler, Eddie ; Handley, Mark ; Floyd, Sally: *RFC 4340: Datagram Congestion Control Protocol (DCCP)*. March 2006

[Krampe] Krampe, Göran: *FastSocketStream*. – URL http://map.squeak.org/package/4517e5f0-d023-429c-b7b9-fe25c2f4d74e. – letzter Zugriff: 02.07.2009

[Kurose und Ross 2008] Kurose, James F. ; Ross, Keith W.: *Computer Networking - A Top-Down-Approach*. 4th. Addison-Wesley, 2008

[Larman und Basili 2003] LARMAN, Craig ; BASILI, Victor: Iterative and Incremental Development: A Brief History. In: *IEEE Computer* 36 (2003), Juni, Nr. 6, S. 47–56

[Machado und Ferraz 2005] MACHADO, Ana C. C. ; FERRAZ, Carlos A. G.: Guidelines for performance evaluation of web services. In: *WebMedia '05: Proceedings of the 11th Brazilian Symposium on Multimedia and the web*. New York, NY, USA : ACM, 2005, S. 1–10

[Niestroj 2008] NIESTROJ, Nicolas: *Erweiterung des ATEO-Systems zur Komplexitätserhöhung von SAM*. 2008. – unveröffentlichte Studienarbeit an der Humboldt-Universität zu Berlin

[Postel 1980] POSTEL, Jonathan: *RFC 768: User Datagram Protocol*. August 1980

[Postel 1981] POSTEL, Jonathan: *RFC 793: Transmission Control Protocol*. September 1981

[Schwarz 2009] SCHWARZ, Hermann: *Fenster zum Prozess: ein Operateursarbeitsplatz zur Überwachung und Kontrolle von kooperativem Tracking*. 2009. – Diplomarbeit

[Stanton 2006] STANTON, N. A.: Hierarchical task analysis: Developments, applications, and extensions. In: *Applied Ergonomics* 37 (2006), Januar, Nr. 1, S. 55–79

[Tanenbaum 2003] TANENBAUM, Andrew S.: *Computernetzwerke*. 4th. Pearson Studium, 2003

[Tanenbaum und van Steen 2008] TANENBAUM, Andrew S. ; STEEN, Maarten van: *Verteilte Systeme: Prinzipien und Paradigmen*. 2. Pearson Studium, 2008

[Umezawa] UMEZAWA, Masashi: *SoapCore - a Squeak SOAP RPC implementatio*. – URL http://www.mars.dti.ne.jp/~umejava/smalltalk/soapOpera/soapCore.html. – letzter Zugriff: 02.07.2009

[Wandke und Nachtwei 2008] WANDKE, Hartmut ; NACHTWEI, Jens: The different human factor in automation: the developer behind versus the operator in action. In: WAARD, Dick de (Hrsg.) ; FLEMISCH, Frank (Hrsg.) ; LORENZ, Bernd (Hrsg.) ; OBERHEID, Hendrik (Hrsg.) ; BROOKHUIS, Karel (Hrsg.): *Human Factors for assistance and automation*. Maastricht, the Netherlands : Shaker Publishing, 2008, S. 1–10

[Weiser 1991] WEISER, Mark: The computer for the 21st century. In: *Scientific American* 265 (1991), S. 94–104

ERKLÄRUNG

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Berlin, den 09. Juli 2009

Nicolas Niestroj

Einverständniserklärung

Ich erkläre hiermit mein Einverständnis, dass die vorliegende Arbeit in der Bibliothek des Institutes für Informatik der Humboldt-Universität zu Berlin ausgestellt werden darf.

Berlin, den 09. Juli 2009

Nicolas Niestroj