FENSTER ZUM PROZESS: WEITERENTWICKLUNG EINES OPERATEURSARBEITSPLATZES IM PROJEKT ARBEITSTEILUNG ENTWICKLER OPERATEUR (ATEO)

Studienarbeit

im Fachgebiet Informatik

Lehrstuhl: Softwaretechnik

von:	Christian Leonhard
	Institut für Informatik
	Humboldt-Universität zu Berlin
Matrikelnummer:	157370
Erstgutachter:	Prof. Dr. sc. nat. Klaus Bothe
Zweitgutachter:	Prof. Dr. sc. nat. Hartmut Wandke
Betreuer:	Dipl.-Psych. Jens Nachtwei

Juli 2010

Christian Leonhard: *FENSTER ZUM PROZESS: WEITERENTWICKLUNG EINES OPERATEURSARBEITSPLATZES IM PROJEKT ARBEITSTEILUNG ENTWICKLER OPERATEUR (ATEO)*, Studienarbeit, © Juli 2010

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LISTINGS

AKRONYME

ATEO Arbeitsteilung Entwickler - Operateur

SAM Socially Augmented Microworld

OA Operateursarbeitsplatz

MA Mentale Antstrengung

MWB Mikroweltbewohner

GUI Graphical User Interface

TCP/IP Transmission Control Protocol / Internet Protocol

UDP User Datagram Protocol

SOAP Simple Object Access Protocol

1 EINFÜHRUNG

Operateure dynamischer und komplexer Systeme, wie industrieller Fertigungsanlagen oder Transportsysteme, benötigen zur Prozessüberwachung und -führung Schnittstellen zum technischen System .

Die Schnittstellen sollten derart gestaltet sein, dass sie es dem Operateur erlauben Störfälle zu erkennen und zu korrigieren. Aufgrund technischer Möglichkeiten der Prozessüberwachung, können die Systeme eine große Menge an Informationen liefern. Würde man diese Menge ungefiltert dem Operateur zur Verfügung stellen, wäre er damit überfordert. Eine Überflutung mit Informationen ist ebenso problematisch wie ein Mangel dieser [1]. Hinzu kommt, dass Störfälle unter Umständen eine andere Informationsdichte erfordern als der Normalzustand. Aus diesen Gründen muss eine Balance bei der Informationsanzeige gefunden werden.

Im Rahmen des Projektes „Arbeitsteilung Entwickler - Operateur (ATEO)“ [6] werden zwei Gruppen von Personen betrachtet, Operateure und Entwickler. An beide Gruppen wird die Anforderung gestellt, ein System bestmöglich zu überwachen und zu führen. Trotzdem beide unterschiedliche Fähigkeiten zur Bewältigung der Anforderungen einsetzen, existieren Überschneidungen. Beispielsweise müssen beide Situationen antizipieren und diagnostizieren. Der Operateur hat zur Antizipation jedoch deutlich weniger Zeit zur Verfügung als der Entwickler. Besonders wichtig für den Operateur ist die Störungsdiagnose, dass ist das Erkennen von der Norm abweichender Zustände des zu überwachenden Systems. Für den Entwickler hingegen ist die Störungsantizipation von zentraler Bedeutung. Mit dem ihm zur Verfügung stehenden Wissen muss er vorausschauend das Systemverhalten einschätzen.

Die zentrale Frage ist, welche Ressourcen Operateuren und Entwicklern zur Verfügung stehen müssen, um die Anforderung bestmöglich zu erfüllen. Die Ressourcen sind für jede Gruppe unterschiedlich. Für die Operateure ist dies die Schnittstelle zum System, welche im Kontext von ATEO durch den Operateursarbeitsplatz repräsentiert wird und im Fokus dieser Studienarbeit steht. In der fertigen Version des Operateursarbeitsplatz (OA), wird es möglich sein, die Ressource über die Quantität der dargebotenen Informationen zu variieren. Die Leistung der von den Entwicklern entworfenen Automatiken ist Grundlage für den Vergleich zwischen den Gruppen. Die Ressource für die Entwick-

ler sind Informationen über das System die sie zum Entwurf der Automatiken nutzen können. Variation wird über die Menge der bereitgestellten Informationen erreicht.

Das Wissen des Entwicklers umfasst eventuell existierende Vorsysteme und das zu implementierende System. Das Wissen über Einsatzsituationen und deren mögliche Folgen ist im Gegensatz zum Operateur geringer und abstrakter. Mit der „Socially Augmented Microworld (SAM)“ stellt ATEO eine Umgebung zur Verfügung, um die Leistungen von Operateur und Entwickler zu vergleichen.

SAM ist ein System, welches von einem Operateur überwacht und geführt wird. Diese Versuchsumgebung beruht auf einem Trackingparadigma. Die Trackingaufgabe wird von zwei Versuchspersonen, MWB genannt, durchgeführt. Es ist die Aufgabe der MWB, ein Objekt entlang eines Pfades vom Start ins Ziel zu steuern. Die MWB erhalten für den Versuch unterschiedliche Ziele. Während einer Geschwindigkeit priorisieren soll, soll der andere versuchen möglichst genau zu fahren. Die Steuerung des Fahrobjektes ist zwischen beiden MWB zu gleichen Teilen geteilt.

Der Operateur überwacht die Trackingaufgabe und kann bei Bedarf eingreifen. Beispielsweise kann der Operateur die Verteilung der Steuerung während der Überwachung ändern. Im Rahmen seiner Diplomarbeit hat Hermann Schwarz [4] ein Graphical User Interface (GUI) für den OA entworfen. Der OA soll weiche und harte Eingriffe ermöglichen. Weiche Eingriffe sind auditive oder visuelle Hinweise mit Vorschlagscharakter, denen die MWB's folgen können - aber nicht müssen. Harte Eingriffe sind beispielsweise Erhöhung oder Verringerung der maximal möglichen Geschwindigkeit, Verteilung der Steuergewalt zwischen den MWB's oder die Vorgabe einer Richtung, z.B. an einer Gabelung.

Als Informationen stehen dem Operateur beispielsweise das Niveau der Anstrengung[1] zur Verfügung, die individuelle Joystickauslenkungen oder eine Streckenvorschau zur Verfügung.

In dieser Studienarbeit soll der Aufwand und die Machbarkeit für Änderungen am Operateursarbeitsplatz untersucht werden. Eine Umsetzung kann ebenfalls erfolgen, wenn sich der Aufwand hierfür in einem überschaubaren Rahmen hält. Auch sollen andere Aufgaben davon nicht beeinträchtigt sein. In einem solchen Fall erfolgt die Umsetzung als ein Teil der folgenden Diplomarbeit. Änderungen umfassen Weiterentwicklungen aufgrund geänderter Anforderungen oder neuer Erkenntnisse, sowie die Anpassung aufgrund fortschreitender technischer Entwicklungen der zugrundeliegenden Programmiersprache Smalltalk und der Programmierumgebung Squeak.

1 Diese Werte werden individuell für jeden MWB zwischen den Fahrten mit Hilfe einer dritten Softwarekomponente Mentale Antstrengung (MA) erfasst.

Als Ausgangspunkt für diese Studienarbeit dient die Machbarkeitsstudie des Operateursarbeitsplatzes von Herman Schwarz. In ihr sind bereits wichtige Elemente des Operateursarbeitsplatzes enthalten, wie die Zuweisung der Steuergewalt oder die Begrenzung der Geschwindigkeit. Durch Erkenntnisse aus dieser Vorarbeit sowie fortschreitende fachliche Betrachtungen, wurden weitere Aufgaben identifiziert. Im Folgenden wird auf die Probleme, ihre denkbaren Lösungswege und Implementierungsmöglichkeiten eingegangen.

2 PROBLEMSTELLUNG

Um die fachliche Weiterentwicklung in künftigen Versuchen nutzen zu können, müssen die Änderungen in die OA Softwarekomponente eingearbeitet werden.

Mit folgenden Aspekten beschäftigt sich die vorliegende Abeit:

AUFGABE	KAPITEL
Versionsupdate	4
Anzeige für Joystickausschlag	5
Gui-Element Steuerungsanteil	6
Netzwerkfunktionalität	7

Tabelle 1: Übersicht bearbeiteter Aufgaben

2.1 VERSIONSUPDATE

Die Implementierungen der drei Komponenten SAM, OA und MA basieren auf Squeak. Squeak selbst wird ständig weiterentwickelt. Dabei werden neue Features hinzugefügt, aber auch Fehler aus bisherigen Versionen entfernt. In vorhergenden Arbeiten ist SAM bereits auf die Version 3.9 erneuert worden. Es soll geprüft werden, inwiefern ein Update für den OA möglich, notwendig oder überhaupt sinnvoll ist.

2.2 ANZEIGE FÜR JOYSTICKAUSSCHLAG

Herman Schwarz realisierte in seinem Entwurf des OA bereits eine Anzeige für die Joystickbewegungen. Die Anzeige stellt die Ausschläge vereinfacht dar. Alle Bewegungen eines Joysticks, auch Teilbewegungen, werden immer als volle Ausschläge in die entsprechende Richtung dargestellt.

Künftig sollen Joystickbewegungen 1:1 abgebildet werden. Die Anzeige soll darüber hinaus noch um die Möglichket erweitert werden, Rückmeldung über die anderen Eingriffe des Operateurs zu geben. Derartige Eingriffe können unter anderem ein Geschwindigkeitslimit oder ein Umverteilen des Steuerungsanteils sein.

2.3 STEUERUNGSANTEIL

In kooperativen Fahrten fließen die Joystickbewegungen beider MWB in die Berechnung der Positionierung des Fahrobjektes ein. Standardmässig startet eine kooperative Fahrt mit einer ausgeglichenen Verteilung zwischen den MWB.

Um Steuerungsanteile umzuverteilen, existiert bereits ein Gui-Element in der Studie von Herman Schwarz. Dieses Element soll überarbeitet und mit erweiterter Funktionalität versehen werden. Dazu gehören beispielsweise ein Reset-Button oder eine verbesserte Art der Rückmeldungen veränderter Steuerungsanteile.

2.4 NETZWERKFUNKTIONALITÄT

Bei Beginn dieser Arbeit gab es nur in den Komponenten SAM und OA eine erste Testimplemtentierung für Netzwerkkommunikation. Aufgrund des reinen Testcharakters, umfasste sie nur einen Teil der benötigten Funktionalität.

Es soll ein Konzept erarbeitet und umgesetzt werden, das eine stabile Kommunikation ermöglicht und dabei alle anfallenden Informtionen korrekt überträgt. Die bisher nicht berücksichtigte Komponente MA zur Messung der Anstrengung soll ebenfalls berücksichtigt werden.

3

VERSUCHSAUFBAU UND BETEILIGTE KOMPONENTEN

Die folgende Darstellung soll vorrangig die technische Sicht zeigen und die Einbettung der einzelnen technischen Komponenten verdeutlichen. Fachliche Gesichtspunkte sind bewußt vernachlässigt worden, um eine knappe Beschreibung zu ermöglichen. Ein kurzer Überblick über die Komponenten:

1. SAM: stellt eine Trackingsimulation zur Verfügung und wird von zwei Probanden, den Mikroweltbewohnern (MWB), genutzt
2. OA: gibt dem Operateur die Möglichkeit das zeitgleich stattfindende Tracking der beiden Mikroweltbewohner zu überwachen und stellt verschiedene Arten von Eingriffen zur Verfügung, um beide MWB zu beeinflussen
3. MA: am Ende einer jeder Fahrt Strecken hat jeder MWB die Möglichkeit, seine Anstrengung mit auf einer Skala auszudrücken. Dieser Wert wird dem Operateur während der nächsten Strecke angezeigt.

Im Folgenden ist der Versuchsaufbau grafisch dargestellt. Der senkrechte schwarze Balken steht stellvertretend für die räumliche Trennung.

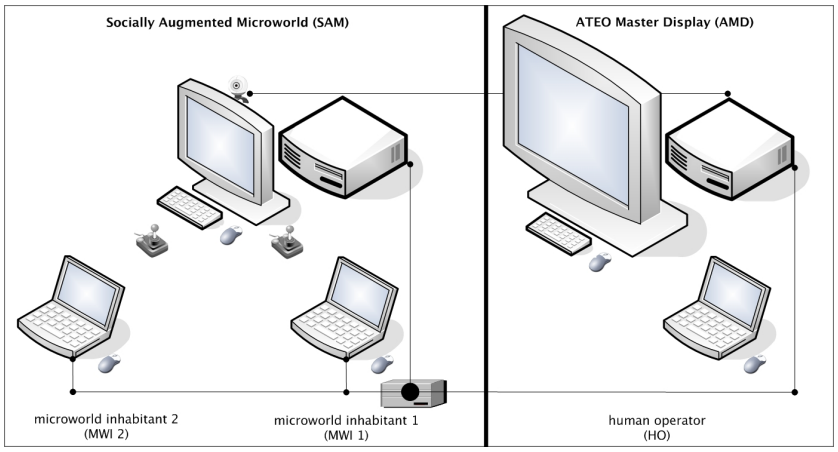


Abbildung 1: Versuchsaufbau

Die MWB steuern gemeinsam ein Fahrobjekt über eine Reihe von Fahrten. Räumlich davon getrennt überwacht ein dritter Proband, der Operateur, die Leistung und Zustände (z.B. Anstrengung) der MWB.

Abbildung 2: Entwurf des OA Version 2.6

Hierzu stellt ihm der Operateursarbeitsplatz (Abb. 2[1]) mehrere Mittel zur Verfügung:

1. Streckenansicht

   Mit einem schwarzen Rahmen ist der Teil der Streckenansicht markiert, der dem Ausschnitt entspricht den die MWB aktuell sehen. Der darüber liegende Teil bildet eine nur dem Operateur sichtbare Vorschau. Damit erhält der Operateur Einsicht in den künftigen Streckenverlauf und kann Hindernisse vor den MWB sehen.

2. Joystickauslenkungen

   Auf beiden Seiten des mit dem schwarzen Rahmen markierten Teils der Streckenansicht befinden sich die Anzeigen für die Joystickauslenkungen. Die linke repräsentiert den ersten MWB und die rechte den zweiten MWB. Die Auslenkungen der Joysticks werden in Bewegungen des grünen Objektes übersetzt.

3. Systemstatus

   Dieses Element bietet einige allgemeine Informationen über die Trackingsimulation SAM. Zum Beispiel mit welcher Strategie ein MWB instruiert wurde oder die Nummer der aktuellen Fahrt.

1 Stand des Entwurfs am Ende der Studienarbeit

4. Anstrengung

   Im gleichen Maße wie sich ein technisches System abnutzt, führt die Anstrengung des Trackings früher oder später zu einer Einschränkung der Leistung. Nach jeder kooperativen Fahrt bewerten die MWB auf einer Skala die subjektiv empfundene Anstrengung. Diese wird dann in der folgenden Fahrt dem Operateur angezeigt.

5. Videobild

   Das Videobild ermöglicht es dem Operateur in Echtzeit u. a. Gesichtsausdruck oder Joystickbewegungen zu beobachten.

6. Visuelle Hinweise

   Beim Auftreten von Hindernissen oder Gabelungen auf der Strecke kann der Operateur mit Hilfe dieses Elementes eine Warnung für die MWB anzeigen. Darüber hinaus hat er die Möglichkeit allgemeine Hinweise auszulösen, wie beispielsweise „langsamer“ oder „schneller“. Das Auslösen von Warnungen und Hinweise werden dem Operateur auch zurückgemeldet. In der Abbildung 2 ist über den Elementen für die Joystickauslenkung die Rückmeldung für den visuellen Hinweis „schneller“ zu sehen.

7. Auditive Hinweise

   Hier hat der Operateur die Gelegenheit beiden oder einem konkreten MWB mit einem auditiven Hinweis[2] zu adressieren. Die Geschwindigkeit und Richtung kann ebenso wie bei den visuellen Hinweisen beeinflusst werden. Weiterhin ist es möglich das Verhalten in Kurven (schneiden vs. genau fahren) vorzugeben und die Führung eines MWB über den anderen übernehmen zu lassen.

8. Geschwindigkeitslimit

   Der Operateur kann die Maximalgeschwindigkeit in 1% Schritten mit Hilfe des Schiebereglers einstellen. Der Reset Button stellt die maximale Geschwindigkeit auf 100% zurück.

9. Richtungsbeschränkung

   Die Freiheit in der Horizontalen für das Fahrobjekt kann mit diesem Element eingeschränkt werden. Solange der entsprechende Button aktiviert ist, ist diese Richtung blockiert. Erst ein Klick auf den Reset Button hebt die Einschränkung wieder auf.

10. Steuergewalt

2 Die auditiven Hinweise sind im Vorfeld aufgenommene mp3-Dateien.

Eine weitere Möglichkeit die Freiheit eines MWB einzuschränken, ist die Reduzierung des Anteils an der Steuerung. Jeder Klick reduziert den Anteil um 5%, die dem anderen MWB zugewiesen werden. Mit diesem Element können Steuerungsanteile nur umverteilt werden, aber nicht insgesamt den MWB entzogen werden.

4 VERSIONSUPDATE

Die Arbeiten von Herman Schwarz basieren auf der Squeak Version 3.8. Die Komponente SAM wurde bereits im Rahmen der Studienarbeit von Nicolas Niestroj [2] auf die Version 3.9 angepasst. Es gibt einige Gründe auch den Operateursarbeitsplatz in die neue Version zu überführen. Die einzelnen ATEO Komponenten wären dadurch hinsichtlich ihrer Squeak Versionen kompatibel. Die Verwendung der Weiterentwicklung von Squeak führt zu mehr Robustheit. Auch die Beseitigung von Fehlern der Version 3.8 stellt einen Vorteil dar.

Eine Untersuchung des Quellcodes und erste Tests wiesen auf keine Probleme für die erstellten Klassen hin. Ein negatives Ergebnis hingegen lieferte die Analyse des verwendeten Webcam Plugins. Eine Recherche auf der Webseite des Plugin - Projektes bestätigte, daß das bisher verwendete Plugin bei einem Versionsupdate nicht mehr eingesetzt werden kann.

Die gefundenen Alternativen waren für die Anwendung im OA nicht geeignet. Sie wurden alle für zum Teil wesentlich ältere Versionen von Squeak geschrieben und schieden daher ebenfalls aus.

Alternativ wäre es denkbar gewesen die nötigen Anpassungen am Plugin innerhalb des Projektes ATEO vorzunehmen. Dies hätte jedoch einen beträchtlichen Aufwand bedeutet, da das notwendige Wissen innerhalb des Projektes nicht vorhanden war. Daher ist dieser Weg nicht weiter verfolgt worden.

Das Videobild für den Operateur ist ein so wichtiger Teil der Versuchsumgebung, dass andere Betrachtungen, wie die Nutzung neuer Features oder die Entfernung von Bugs in der Version 3.9, vernachlässigt werden müssen. Zwischen den Versionen 3.8 und 3.9 besteht insofern Kompatibilität, als daß der Quellcode aus Version 3.8 in Version 3.9 verwendet werden kann. Dessen ungeachtet ist die Kompatibilität für das erfolgreiche Zusammenwirken von SAM und dem OA nicht erforderlich, da Kommunikation ausschließlich über ein Netzwerk erfolgt.

5 GUI-ANPASSUNGEN

5.1 ANZEIGE FÜR JOYSTICKAUSSCHLAG

Bei der Ermittlung der Position des Fahrobjektes, fließen die Joystickauslenkungen beider MWB ein. Um es dem Operateur zu ermöglichen, den individuellen Einfluß eines einzelnen MWB einzuschätzen, werden diese einzeln dargestellt.

In der Studie des OA werden Joystickausschläge abstrahiert dargestellt. Für jede Achse existieren nur drei Zustände. Die Neutralstellung und die beiden Maximalausschläge am jeweils äußeren Rand der Anzeige. Dadurch wird der gesamte Wertebereich möglicher Auslenkungen auf acht unterschiedliche Ausschläge approximiert, vergleichbar mit den Himmelsrichtungen auf einem Kompass.

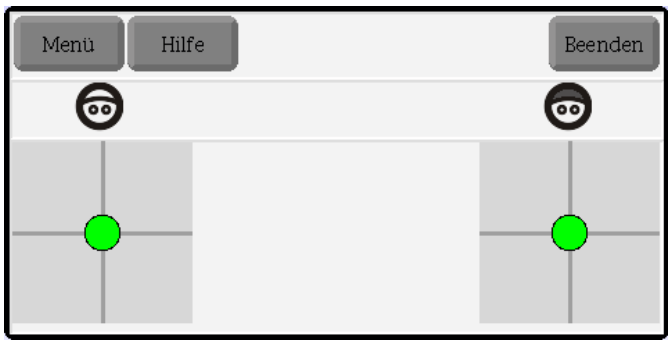


Abbildung 3: Element zur Anzeige der Joystickauslenkungen (Studie)

Dieses Anzeigeelement wurde derart umgestaltet (Abb. 4), dass auch kleine Auslenkungen, die Einfluss auf das Fahrobjekt haben, in der Anzeige ablesbar sind. Das soll den Operateur in die Lage versetzen eine genauere Einschätzung der individuellen Fahrleistung abzugeben.

Zusätzlich soll dieses Anzeigeelement in der neuen Version dem Operateur harte Eingriffe zurückmelden. Als harte Eingriffe werden die Eingriffe bezeichnet, die bestimmte Joystickbewegungen der MWB bei der Berechnung der neuen Position des Fahrobjektes nicht oder nur anteilig berücksichtigen. Im Moment sind das Geschwindigkeit, Verteilung des Steuerungsanteils und Richtungsauslenkung. Jede dieser Einschränkungen wird über eine Größenänderung des hellgrauen Anzeigeelementes innerhalb der schraffierten Fläche realisiert. Die Innenkante der schraffierten Fläche repräsentiert die maximal mögliche Ausdehnung des hellgrauen Bereichs, bei einem Steuerungsanteil von 100% und

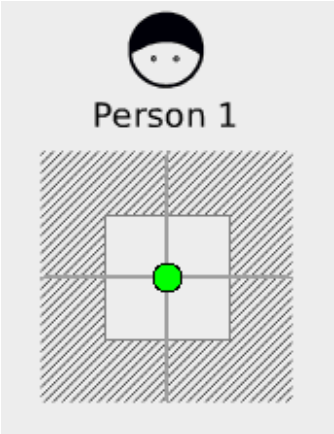


Abbildung 4: Element Joystickauslenkung aus dem OA

einem Geschwindigkeitlimit von 100%. Die Veränderungen lassen sich eindeutig den eingestellten Eingriffen zuordnen. Wird die Geschwindigkeit begrenzt, also die Joystickauslenkung in der y-Achse, so wird die Anzeige in der gleichen Achse, also in der Höhe, verändert. Die Geschwindigkeit kann in dem Wertebereich von [0, 100] eingestellt werden und fließt daher als einfacher Faktor in die Höhe des Anzeigeelementes ein. Bei einer Einschränkung der Links- oder Rechtsauslenkung wird das Element halbiert.

Auswirkungen von harten Eingriffen im Anzeigeelement:

- Umverteilung des Steuerungsanteils: Gleichmäßige Veränderung beider Achsen
- Änderung des Geschwindigkeitslimits: Veränderung in der Y-Achse
- Einschränkung der Richtung: Veränderung in der X-Achse

Die Abbildung 7 zeigt beispielhaft die Umsetzung harter Eingriffe in der Josytickanzeige in verschiedenen Varianten.

Listing 1: OPJoystick Methode `calculateExtentWithLeft: andRight: withInput: andSpeed:`

```
calculateExtentWithLeft: directSetDirectionLeft
   andRight: directSetDirectionRight withInput:
   input andSpeed: speed

        | newExtent |

        newExtent := self extent * input.
        (directSetDirectionLeft |
           directSetDirectionRight)
        ifTrue:[
                newExtent setX: ((self width *
                   input) // 2) setY: newExtent y
                   .
        ].

        (speed = 0)
        ifTrue:[ newExtent setX: newExtent x setY
           : 1. ]
        ifFalse:[ newExtent setX: newExtent x
           setY: (newExtent y * speed) ceiling.
           ].

        ^newExtent
```

Im vorhergehenden Listing ist die Methode dargestellt, welche die Auswirkungen der harten Eingriffe umsetzt. Nach dem Listing werden einzelnen Codefragmente den einzelnen Eingriffen zugeordnet und erläutert. Zum näheren Verständnis sei angemerkt, das den Instanzen der Klasse OPJoystick nicht bekannt ist, für welchen MWB sie die Joystickauslenkungen darstellt. Daher taucht diese Unterscheidung an keiner Stelle auf.

Listing 2: Anpassung bei Änderung des Steuerungsanteils

```
...

newExtent := self extent * input.

...
```

Dieser Ausschnitt zeigt die Anpassungen für den übergebenen Steuerungsanteil. Wie weiter oben bereits angemerkt, werden dadurch beide Dimensionen angepasst. Die Variable `input` ist ein übergebener Parameter mit einem Wertebereich zwischen 0.0 und 1.0. Der Aufruf `self extent` liefert die Größe des Anzeigeelementes OPJoystick zurück. Für die initiale Darstellung einer kooperativen Fahrt, die Steuerung ist gleichmäßig verteilt, würde `input` den Wert 0.5 enthalten. Der Aufruf von `self extent` liefert

130@130[1] zurück. Im Ergebnis bedeutet das eine Größe für den hellgrauen Bereich von 65@65, wie es Abbildung 4[2] bereits nahegelegt hat. Es existiert keine explizite Behandlung des initialen Zustandes. Die Methode ist so gestaltet, dass in diesem Fall einfach der Initialwert des jeweiligen Steuerungsanteil[3] übergeben wird.

Listing 3: Anpassung bei Richtungsbeschränkung

```
...

(directSetDirectionLeft | directSetDirectionRight
    )
ifTrue:[
        newExtent setX: ((self width * input) //
            2) setY: newExtent y.
].

...
```

Dieser Ausschnitt verarbeitet die Eingaben aus dem Element Richtungsbeschränkung (Abb. 2). Wenn einer der beiden Buttons[4] zum Zeitpunkt des Methodenaufrufes aktiviert ist, enthält einer der beiden Parameter `directSetDirectionLeft` oder `directSetDirectionRight` den Wert `true`. In diesem Fall wird die horizontale Ausdehnung durch den Ausdruck

```
((self width * input) // 2)
```

um die Hälfte reduziert und durch den Aufruf von `setX:` zugewiesen.

Listing 4: Anpassung bei Änderung Geschwindigkeitslimit

```
...

(speed = 0)
ifTrue:[ newExtent setX: newExtent x setY: 1. ]
ifFalse:[ newExtent setX: newExtent x setY: (
    newExtent y * speed) ceiling. ].

...
```

In diesem Codefragment werden die Auswirkungen des Parameters speed, welcher den aktuell eingestellten Wert[5] des Elementes Geschwindigkeitslimit enthält, umgesetzt. Dabei stellt

1 Das ist die Größe des Elementes, welche in der `initialize` Methode festgelegt wurde.
2 Die Abbildung zeigt das Element im initialen Zustand.
3 Bei kooperativen Fahrten immer 0.5. In einer Einzelfahrt ist der Wert 1.0 für den aktiven MWB, der andere MWB ist dann entsprechend 0.0.
4 Das Element ist so gestaltet, dass immer nur ein Button aktiviert sein kann.
5 Die Skala im Element Geschwindigkeitslimit reicht von 0 bis 100. Dieser Wert wird vorher umgerechnet, so dass er zwischen 0 und 1 liegt, um ihn direkt als Faktor einfliessen lassen zu können.

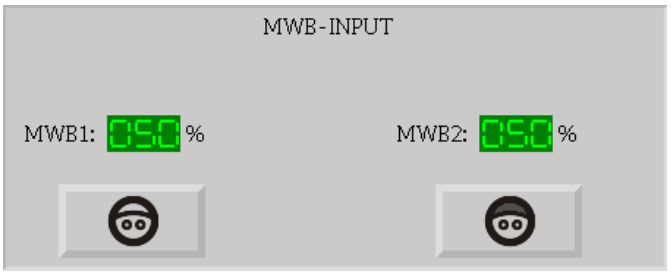
der Wert 0 aus dem Wertebereich [0...100] eine Besonderheit dar. Damit soll verhindert werden, dass die Höhe des hellgrauen Bereiches Null wird, also nicht mehr sichtbar ist. Trotzdem wären horizontale Auslenkungen weiterhin möglich und wahrscheinlich, was zur Folge hätte, dass das grüne Objekt scheinbar frei auf der schraffierten Fläche positioniert wird, was nicht gewünscht ist. Zur Vermeidung wird in einem derartigen Fall die Höhe auf den Wert 1 gesetzt. Damit ist der hellgraue Bereich immer noch sichtbar und das grüne Objekt wird darauf platziert.

Für alle Werte von `speed` ungleich 0 fließt der Parameter als Faktor in die Berechnung der vertikalen Ausdehnung des hellgrauen Bereiches ein.

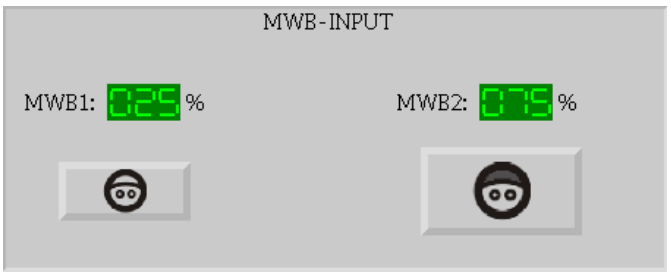
5.2 VERTEILUNG STEUERUNGSANTEIL

Einen Teil der Strecken absolvieren beide MWB kooperativ. Das bedeutet, dass sie das Fahrobjekt gemeinsam in der Vertikalen und Horizontalen steuern. Die individuellen Eingaben der MWB fließen gemäß des eingestellten Anteils an der Steuerung in die Berechnung der Position des Fahrobjektes ein. Eine kooperative Fahrt beginnt immer mit einer gleichmäßigen Verteilung von 50/50.

Der Operateur soll die Möglichkeit erhalten, diese Verteilung zu ändern. Damit kann er beispielsweise einem ungenau fahrenden MWB bei Hindernissen den Steuerungsanteil entziehen und seinem genauer fahrenden Partner zuweisen. Beide Anteile ergeben zusammen immer 100%. Es ist also nicht möglich, den MWBs insgesamt die Steuerung zu entziehen. Steuerungsanteile werden immer nur zwischen den MWBs umverteilt.



(a) Ausgangzustand mit gleichmäßiger Verteilung



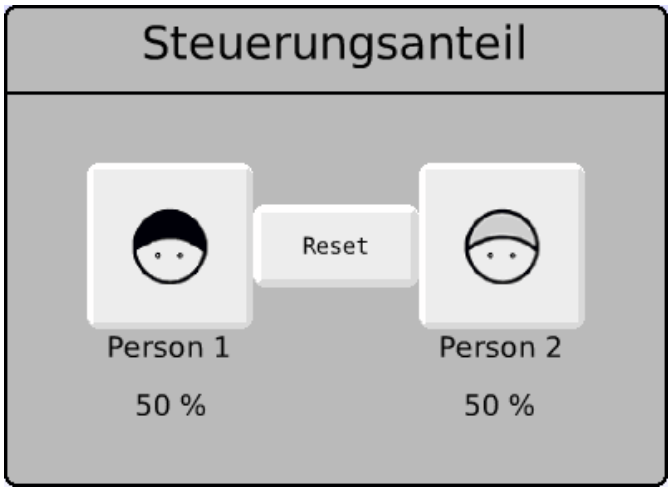
(b) Zustand nach einmaligem Klick

Abbildung 5: Element Steuerungsanteil aus der Studie

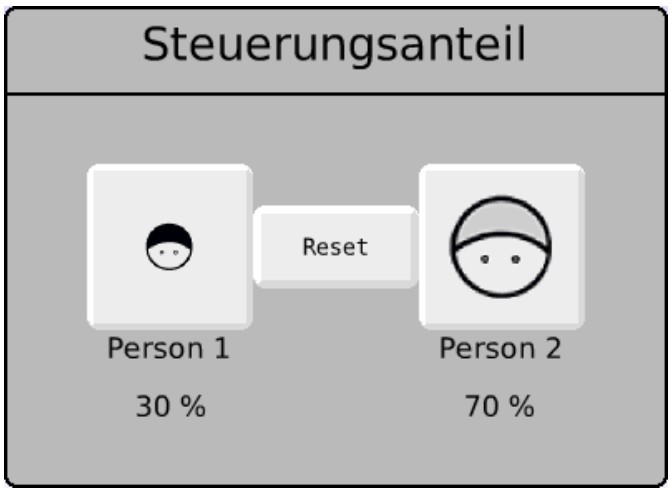
In der Studie von Herman Schwarz (Abb. 5) ist es bereits möglich, Steueranteile zu verändern. Ein Klick auf den entsprechenden Button, bewirkt eine Veränderung um 25%. Für eine feinere Einstellung, soll die Schrittweite auf 5% verringert werden. Ein „Reset"-Button wird als weiteres Steuerelement eingefügt. Dieser soll es ermöglichen schnell in die Ausgangsverteilung von 50/50 zurückzukehren. Damit sollen unnötige Klicks reduziert unf die Usability verbessert werden.

In der Studie veränderte jeder Klick auf einen der beiden Zuweisungsbuttons auch die Größe beider Buttons. Dieser Effekt diente der Rückmeldung für den Operateur und ermöglichte ihm

ein schnelleres Ablesen der eingestellten Verteilung. Dieses Verhalten soll entfallen. Stattdessen erfolgt die Rückmeldung über die Größe des MWB-Symbols in beiden Zuweisungsbuttons, welche sich entsprechend des Steuerungsanteils ändert. Die Größe der Buttons an sich bleibt in jedem Zustand konstant. Zusätzlich wird die aktuelle Verteilung in der Anzeige zur Joystickauslenkung (Abb. 7) angezeigt. Dazu wird die Fläche für jeden MWB, welche die maximal mögliche Auslenkung symbolisiert, im Verhältnis zum aktuellen Steuerungsanteil skaliert.



(a) Ausgangzustand des neuen Elementes mit gleichmäßiger Verteilung



(b) Zustand nach mehrfachen Klicks

Abbildung 6: Element Steuerungsanteil aus dem OA

Nach den konzeptionellen Erläuterungen werden im Folgenden einige Aspekte der technischen Umsetzung erläutert. Die Aufgabe der vorgestellten Methode ist es, die durch Buttonklicks notwendig gewordenen Änderungen vorzunehmen. Das sind im Kern die Größenänderungen der beiden MWB-Icons in den Buttons und das Anpassen der Prozentangaben darunter.

Listing 5: Auszug aus der Methode `updateMwiButtons` (Berechnung des neuen Steuerungsanteils)

```
...

(whichButtonClicked = 1) & ((distributionMwiOne +
    clickIncrement) <= 100)
ifTrue:[ "Button 1 was clicked"
   distributionMwiOne := distributionMwiOne +
   clickIncrement.].

(whichButtonClicked = 2) & ((distributionMwiOne -
    clickIncrement) >= 0)
ifTrue:[ "Button 2 was clicked"
   distributionMwiOne := distributionMwiOne -
   clickIncrement. ].

distributionMwiTwo := 100 - distributionMwiOne.

...
```

Dieser Teil der Methode `updateMwiButtons` berechnet den neuen Steuerungsanteil für beide MWB. Da es sich ausschließlich um eine Umverteilung der Anteile handelt, ergeben beide in Addition immer 100. Aus diesem Grund ist es ausreichend einen Wert zu speichern, da sich der andere Wert eindeutig errechnen läßt. Die hierfür genutzte Variable `distributionMwiOne` enthält den Anteil des MWB 1 und ist eine Instanzvariable. Der Anteil des MWB 2 wird in `distributionMwiTwo` gespeichert. Diese Variable existiert allerdings nur innerhalb der Methode `updateMwiButtons`, wird also beim Verlassen dieser gelöscht. Der Parameter der Methode `whichButtonClicked` zeigt an, welcher Button geklickt wurde. In der Instanzvariable `clickIncrement` ist die Schrittweite je Buttonklick hinterlegt. Dadurch ist es möglich, die Schrittweite mit wenig Aufwand anzupassen.

In Abhängigkeit welcher Button geklickt wurde, wird geprüft ob der aktuelle Wert der Instanzvariable `distributionMwiOne` um den Wert in `clickIncrement` erhöht[6] werden kann ohne die Grenze von 100% zu überschreiten. Sollte dieser Wert unter 100 liegen, wird der inhalt von `distributionMwiOne` neu berechnet und zugewiesen. Im Anschluss wird die lokale Variable `distributionMwiTwo` zugewiesen.

6 Da ein Klick auf Button 1 einer Zuweisung für den MWB 1 entspricht, muss hier nur auf Addition geprüft werden.

Listing 6: Auszug aus der Methode `updateMwiButtons` (Berechnung der neuen Ausdehnung der MWB-Icons)

```
...

"calculating the new x,y values for the extent of
    the label for both buttons"
label1 := labelMin + ((labelMax - labelMin) *
   distributionMwiOne/100).
label2 := labelMin + ((labelMax - labelMin) *
   distributionMwiTwo/100).

[...]
labelGraphicMwiOne image: ((formForButton1
   scaledToSize: label1@label1) [...].

[...]
labelGraphicMwiTwo image: ((formForButton2
   scaledToSize: label2@label2) [...].

...
```

Dadurch das die Grafiken in den Buttons in ihrer Grundform quadratisch sind, ist es ausreichend eine Kante neu zu errechnen. Zu diesem Zweck werden die lokalen Variablen `label1` und `label2` verwendet. Die Nummerierung korrespondiert mit denen der MWB.

Teil der Anforderungen war es, dass die Grafiken auch bei einem Steuerungsanteil von 0% sichtbar sind. Gleichzeitig sollten sie nicht über die Grenzen des Button hinaus wachsen. Daher wurden sowohl eine nicht zu unterschreitende Mindestgröße, als auch eine Maximalgröße festgelegt. Diese Angaben werden in den Instanzvariablen `labelMin` und `labelMax` gespeichert. Für den maximalen Wert von `distributionMwiOne`[7] von 100 ergibt der Ausdruck:

```
labelMin+((labelMax-labelMin)*distributionMwiOne/100)
```

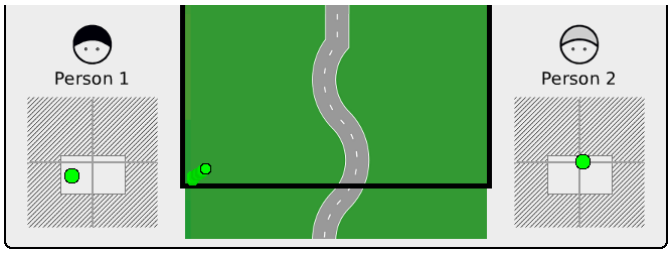
den Wert von `labelMax`.

Für den anderen Extremfall, `distributionMwiOne` ist 0, würde das Ergebnis `labelMin` sein. Da die Werte immer zwischen diesen beiden Extremen liegen, ist sichergestellt das auch die Größen der Grafiken innerhalb der vorgesehenen Grenzen verbleiben.

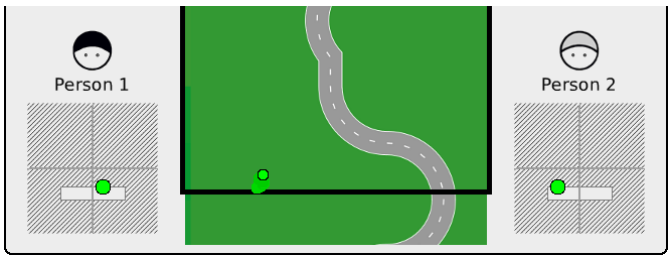
Die letzten beiden Statements in dem Methodenausschnitt zeigen die tatsächliche Skalierung der Grafiken. In den Variablen `formForButton1` und `formForButton2` sind die ursprünglichen Grafiken der Buttons, welche bei jedem Aufruf neu geladen werden. Aufgrund des Qualitätsverlustes bei wiederholten Skalierungen wird jedes Mal auf die Originale zurückgegriffen.

7 Da für `distributionMwiTwo` der gleiche Wertebereich gilt, ist die Auswertung analog vorzunehmen.

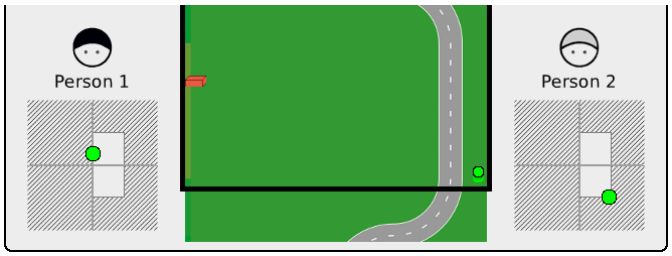
An dieser Stelle ist auch die vorher angesprochene Verwendung der lokalen Variablen `label1` und `label2` als Wert sowohl für die vertikale als auch horizontale Dimension zu sehen.



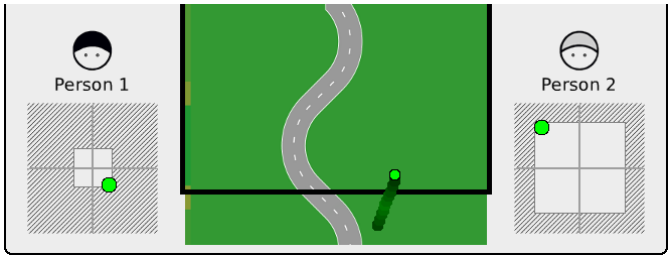
(a) Geschwindigkeit: 60%, Richtungsbeschränkung: keine, Steuerungsanteil: 50%/50%



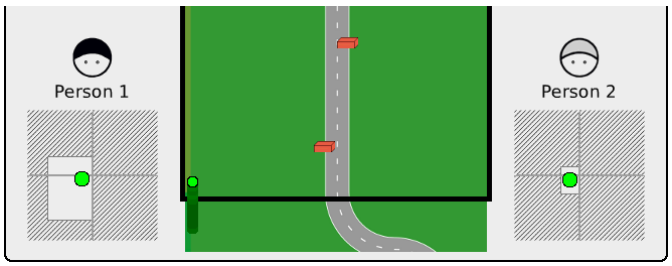
(b) Geschwindigkeit: 20%, Richtungsbeschränkung: keine, Steuerungsanteil: 50%/50%



(c) Geschwindigkeit: 100%, Richtungsbeschränkung: links, Steuerungsanteil: 50%/50%



(d) Geschwindigkeit: 100%, Richtungsbeschränkung: keine, Steuerungsanteil: 30%/70%



(e) Geschwindigkeit: 70%, Richtungsbeschränkung: rechts, Steuerungsanteil: 70%/30%

Abbildung 7: Verschiedene Arten von Rückmeldungen harter Eingriffe im Joystickelement

6 NETZWERKIMPLEMENTIERUNG

Wie in Kapitel 3 dargelegt, sind die einzelnen Komponenten durch den Versuchsaufbau räumlich getrennt. Zur Überwindung der Trennung erfolgt die Kommunikation zwischen den einzelnen Komponenten über ein Ethernet Netzwerk.

Ursprünglich wurde SAM als alleinstehendes Programm entwickelt. Eine Erweiterung um Netzwerkfunktionalität ist somit notwendig, da SAM in der Lage sein muss, mit dem OA Daten über das Netzwerk auszutauschen. Dessen ungeachtet soll sowohl SAM, als auch der OA weiterhin ohne Netzwerkanbindung funktionieren. Da beide über einen Konfigurationsdialog [1] verfügen, bietet es sich an, dort jeweils einen Button einzufügen, mit dem die Netzwerkfunktionalität an- und ausgeschaltet werden kann. Die Komponente zur Messung der Anstrengung wird ebenfalls mit Netzwerkfunktionalität versehen.

In diesem Kontext agiert der OA als Zentrale für die anderen beiden Komponenten. Sowohl SAM als auch die Komponente MA verbinden sich nur mit dem OA. Untereinander tauschen sie keine Informationen aus. Zu Beginn wurden in jeder Komponente eigene Varianten der benötigten Methoden implementiert. Für die anfangs geringe Komplexität funktionierte dieses Vorgehen gut.

Nach einer Reihe von experimentellen Implementierungen und tatsächlichen „gelebten“ Iterationen des OA, wurde klar, dass eine getrennte Betrachtungsweise nicht geeignet ist, um den steigenden Funktionsumfang befriedigend umzusetzen. Durch die jeweils eigenen Implementierungen wurden Wartung und Fehlersuche sehr aufwendig. Auch beim Umsetzen neuer Features war diese Trennung hinderlich.

Deshalb lag es nahe, eine Basisklasse zu entwerfen, von der jede Netzwerkklasse in den einzelnen Komponenten ableiten würde. Sie implementiert unter anderem Methoden, mit denen Verbindungen aufgebaut und wieder geschlossen und Daten gesendet und empfangen werden können. Durch Parameter können die Methoden individuell in jeder Komponente verwendet werden.

Zur Umsetzung der Netzwerkkommunikation werden verschiedene Protokolle verwendet. Die Gründe dafür werden im Folgenden erläutert.

1 Die Komponenten SAM und OA verfügen jeweils über einen Dialog mit dessen Hilfe die Komponenten konfiguriert werden können. Die Dialoge werden in der Diplomarbeit näher beschrieben.

Für die Kommunikation über das Netzwerk stehen unterschiedliche Protokolle zur Verfügung. In seiner Diplomarbeit [3] untersuchte Nicolas Niestroj verschiedene Netzwerkprotokolle, um festzustellen, welches den Anforderungen des Versuchsaufbaus am besten gerecht wird. Als Ergebnis seiner Analyse grenzte er die Auswahl auf die Protokolle Transmission Control Protocol / Internet Protocol (TCP/IP), User Datagram Protocol (UDP) und Simple Object Access Protocol (SOAP) ein.

Durch das Zusammenspiel von SAM, OA und MB ergeben sich verschiedene Anforderungen:

- Experimentablauf synchronisieren:
  - Starten der Streckenvorschau zum richtigen Zeitpunkt
  - Anzeige von Instruktionen auf dem Operateursarbeitsplatz in Abhängigkeit des Zustandes von SAM
  - Eingaben des Operateurs nur zulassen, wenn die MWB auch eine Strecke fahren
- Sicheres Übermitteln von Daten:
  - Buttonklicks des Operateurs erfolgen in der Regel nur einmal und sollten daher nicht verloren gehen
- Schnelles Übermitteln von Daten:
  - Geschwindigkeit des Trackingobjektes für die Streckenvorschau
  - Joystickausschläge für die Anzeige im Operateursarbeitsplatz
  - Geschwindigkeit eines dynamischen Hindernisses

TCP/IP bietet sich an, wenn das sichere Übermitteln von Information unerlässlich ist. Bei der Übertragung wird sichergestellt, dass Datenpakete auf der Gegenseite ankommen. Diese Sicherheit geht zu Lasten der Geschwindigkeit, da zusätzlich Quittungen versendet und ausgewertet werden. In einer Beispielimplementierung wurde versucht, die Anzahl der verwendeten Protokolle zu reduzieren. An allen relevanten Stellen wurde das UDP Protokoll durch TCP/IP ersetzt. Bei der Streckenvorschau stellte sich ein merkliches Ruckeln ein. Eine Analyse ergab, dass bei der Verwendung von TCP/IP eine Latenzzeit von bis zu 100 ms auftritt. Ob es am TCP/IP Protokoll selbst lag oder an der Art wie es in Squeak implementiert wurde, ließ sich nicht ohne größeren Aufwand herausfinden. Da die Antwort darauf das Problem nicht gelöst hätte, wurden in dieser Richtung keine weiteren Nachforschungen angestellt.

Für Kommunikationskanäle, bei denen es vor allem auf eine schnelle Übertragung ankommt, kann das UDP Protokoll genutzt

werden. Diese Kanäle übertragen Daten für die Streckenvorschau von SAM an den OA. Das sind beispielsweise Geschwindigkeit oder y-Position des Fahrobjektes. Hier ist es unproblematisch, wenn ausnahmsweise einzelne Pakete verloren gehen. Durch die hohe Frequenz der Datenübermittlung, fallen einzelne Fehlübermittlungen unter die Wahrnehmungsschwelle. SOAP wurde in der Diplomarbeit von Nicolas Niestroj als Protokoll für die Steuerung des Versuchsablaufs vorgesehen. Allerdings kann auch das TCP/IP Protokoll, wenn auch auf einem einfacheren Niveau, diese Aufgabe erfüllen. Der Verzicht auf SOAP bietet den Vorteil, dass eine Technologie weniger implementiert und gewartet werden muss. Die erhöhte Latenzzeit bei TCP/IP kann vernachlässigt werden, da die Versuchsablaufsteuerung nicht in dem Maße zeitkritisch ist.

6.1 KLASSE ATEONETWORK

Die Klasse ATEONetwork dient als Basisklasse. Sämtliche Netzwerkklassen der Einzelkomponenten erben von ihr. Die Klasse selbst wird nicht instanziiert. Sie ist als abstrakte Klasse zu verstehen.

Die Klasse enthält alle notwendigen Methoden, um eine funktionierende Netzwerkkommunikation zu ermöglichen. Das umfasst den Aufbau von Verbindungen sowie das Senden und Empfangen von Daten. Jede der zuvor genannten Aufgaben wurde für beiden Protokolle, TCP/IP und UDP, implementiert. In frühen Versionen existierte für jede Art von zu übermittelnder Information eine eigene Verbindung. Das bedeutete, dass beispielsweise die Geschwindigkeit, Joystickbewegungen oder Hinweise des Operateurs über eine eigene Verbindung gesendet wurden. Mit zunehmender Anzahl von Features, die nach dem alten Paradigma eine eigene Verbindung erfordern würden, zeigte sich, dass dieses Vorgehen eine effiziente Weiterentwicklung erheblich erschwerte. Aus diesem Grund wurden in der Klasse ATEONetwork die Voraussetzungen geschaffen, zwischen zwei Komponenten mit nur einer Verbindung je Protokoll auszukommen.

Nachfolgend werden die wichtigsten Methoden der Netzwerkbasisklasse beschrieben.

6.2 INITIALISIERUNG

In dieser Methode sind die Ports für die einzelnen Verbindungen festgelegt. Sie werden in Instanzvariablen gespeichert. Anhand des Namens kann im Quelltext sehr einfach erkannt werden, zu welcher Verbindung eine Portnummer gehört. Die erbenden Klassen in den Komponenten greifen nur auf die Variable zu. Damit erfordert eine Umstellung der Ports lediglich einen Eingriff

in der initialize Methode der Basisklasse ATEONetwork. Wartung und Erweiterung werden dadurch erheblich vereinfacht.

Listing 7: ATEONetwork Methode 1

```
initialize

        tcpPortOpSendSamReceive := 50000.
        tcpPortOpReceiveSamSend := 51000.
        tcpPortOpReceiveMwSend  := 52000.

        udpPortOpReceiveSamSend := 53000.
```

6.3 VERBINDUNGSAUFBAU

Zum Verbindungsaufbau stellt die Klasse vier Methoden bereit. Je zwei für das TCP/IP und das UDP Protokoll.

Eine Verbindung zwischen zwei Komponenten wird in zwei Schritten aufgebaut:

1. Komponente 1 stellt einen Verbindungsendpunkt zur Verfügung
2. Komponente 2 versucht sich aktiv mit dem bereiten Endpunkt von Komponente 1 zu verbinden

Die Methoden für die Bereitstellung eines empfangsbereiten Endpunktes:

Listing 8: ATEONetwork Methode 2

```
createListeningTcpSocketStreamOnPort: port

        | newSocketStream |

        newSocketStream := SocketStream on: (
            Socket newTCP listenOn: port).

        ^newSocketStream
```

Listing 9: ATEONetwork Methode 3

```
createListeningUdpSocketOnPort: port

        | newSocket |

        newSocket := Socket newUDP setPort: port.

        ^newSocket
```

In Squeak steht für UDP nur eine Implementierung auf Socket Ebene zur Verfügung. Sockets sind in Squeak der primitivste

Weg Netzwerkkommunikation zu realisieren. Für das Protokoll TCP/IP stellt Squeak eine Stream Implementierung zur Verfügung, welche als Wrapper für die Sockets agiert. Diese ist einfacher zu benutzen und weniger fehleranfällig.

Listing 10: ATEONetwork Methode 4

```
createConnectingTcpSocketStreamOnPort: port
   andHost: hostIP

        | newSocketStream |

        newSocketStream := SocketStream
           openConnectionToHost: hostIP port:
           port.

        ^newSocketStream
```

Listing 11: ATEONetwork Methode 5

```
createConnectingUdpSocketOnPort: port andHost:
   hostIP

        | newSocket |

        newSocket := Socket newUDP setPeer:
           hostIP port: port.

        ^newSocket
```

Wie in den obigen beiden Listings zu sehen ist, liefern die beiden Methoden erwartungsgemäß einen UDP-Socket bzw. einen TCP/IP-Socketstream zurück. Damit ist bereits in der Basisklasse sichergestellt, das für jedes Protokoll die korrekten Verbindungsendpunkte erstellt werden.

6.4 SENDEN UND EMPFANGEN

In früheren Umsetzungen wurde für jeden Informationsfluss eine eigene Verbindung genutzt. Dies hat sich, wie bereits erläutert, als nachteilig herausgestellt. Als Folge teilen sich mehrere Informationsflüsse mit verschiedenen Sendehäufigkeiten einen Kommunikationskanal. Der Empfänger der Daten muss in die Lage versetzt werden, zu erkennen welche Informationen er verarbeitet. Für diesen Zweck wurde für jeden Kommunikationskanal ein festes Format entworfen, mit dem eine Nachricht (Frame) zusammengesetzt wird. Die zu sendenden Daten sind durch vorher vereinbarte, beiden Seiten bekannte, Separatoren getrennt. Innerhalb eines Frames werden Informationen wieder durch eigene Separatoren getrennt. Mit diesen Mitteln ist es dem Empfänger

möglich, sowohl einen einzelnen Frame als auch mehrere Frames hintereinander zu verarbeiten.

So könnten, von SAM an den OA, gesendete Frames aussehen:

Listing 12: Beispielframes

```
[ data1,data2,-,-,data5# ]
[ data1,data2,-,-,data5#data1,-,data3,-,data5# ]
```

Vorher wurde bereits kurz die unterschiedliche Häufigkeit angesprochen in der verschiedene Informationen gesendet werden. So müssen z.B. Daten zur Experimentsteuerung im Vergleich zur Geschwindigkeit, nur sehr selten versendet werden. Um das Format trotzdem einzuhalten, werden in einem solchen Fall, wie im obigen Beispiel zu sehen, Platzhalter eingefügt. Dadurch weiß der Empfänger, dass diese Stelle im Frame keine Informationen enthält.

VERWENDUNG	SEPARATOR
Informationen in einem Frame	,
Zwischen zwei Frames	#
Platzhalter	-

Tabelle 2: Übersicht verwendete Separatoren

Die benötigten Methoden zum Senden und Empfangen sind nur auf ihre Kernaufgabe reduziert. Das vor dem Senden erforderliche Vorbereiten der Daten, oder das Verarbeiten nach dem Empfangen soll individuell von den einzelnen Komponenten implementiert werden. Die Verarbeitung der Frames geschieht in den ModelData-Klassen der einzelnen Komponenten. Diese Klassen dienen als zentraler Speicherort für Daten. Daneben verfügen sie über entsprechende Methoden zur Bereitstellung von Frames und deren Verarbeitung.

Die Methoden für das Versenden von Daten:

Listing 13: ATEONetwork Senden TCP/IP

```
sendTcpData: data onStream: socketStream

        socketStream sendCommand: data.
```

Listing 14: ATEONetwork Senden UDP

```
sendUdpData: data onSocket: udpSocket

        udpSocket sendData: data.
```

Die entsprechenden Gegenstücke zum Empfang:

Listing 15: ATEONetwork Empfang TCP/IP

```
receiveTcpData: socketStream

        | returnValue |

        returnValue := ''.

        [socketStream isDataAvailable]
        whileTrue:[ returnValue := returnValue, (
            socketStream nextLineCrLf). ].

        ((returnValue size) = 0)
        ifTrue:[ returnValue := nil. ].

        ^returnValue
```

Listing 16: ATEONetwork Empfang UDP

```
receiveUdpData: udpSocket

        | returnValue |

        returnValue := nil.

        returnValue := udpSocket
            receiveAvailableData.

        ^returnValue
```

Wie bereits erläutert, beschränken sich die Methoden nur auf ihre eigentlichen Aufgaben, das Senden und Empfangen von Daten.

6.5 BEISPIELIMPLEMTENTIERUNG

Zur Veranschaulichung soll an dieser Stelle eine Implementierung einer Verbindung mit dem TCP/IP-Protokoll gezeigt werden. Die Verbindung wird zwischen dem OA und SAM aufgebaut.

Auf der Seite des OA reicht der Aufruf:

Listing 17: OA: aufbauen einer TCP/IP Verbindung

```
createConnections

        ...

        tcpConnectionOpReceiveSamSend := self
            createListeningTcpSocketStreamOnPort:
            tcpPortOpReceiveSamSend.

        ...
```

Der TCP/IP Socketstream wird in der Variable *tcpConnectionOpReceiveSamSend* gespeichert. In diesem Fall wird ein Socketstream erstellt, der auf eine eingehende Verbindung wartet. Die genutzte Methode *createListeningTcpSocketStreamOnPort* stammt aus der Basisklasse ATEONetwork. Der Parameter *tcpPortOpReceiveSamSend* beinhaltet die Portnummer, die mit den übrigen Nummern in der *initialize* Methode der Basisklasse definiert wird.

In SAM wird das entsprechende Gegenstück zum oben erstellten passiven Endpunkt erstellt.

Listing 18: OA: aufbauen einer TCP/IP Verbindung

```
createConnections: ipAddress

        ...

        tcpConnectionOpReceiveSamSend := self
            createConnectingTcpSocketStreamOnPort:
             tcpPortOpReceiveSamSend andHost:
            ipAddress.

        ...
```

Die Vorgehensweise ist hier ähnlich. Zur Erstellung des Endpunktes wird eine Methode aus der Basisklasse verwendet. Die aufgerufende Methode muss eine andere sein, da die Implementierung beim OA einen passiven Endpunkt zurückliefert. Damit mit dem bereitstehenden Endpunkt eine Verbindung aufgebaut werden kann, muss an dieser Stelle die IP-Adresse und Portnummer als Parameter übergeben werden.

Diese Aufteilung gilt für die übrigen Netzwerkimplementierungen ebenfalls. Der OA stellt ausschliesslich passive, wartende Verbindungsendpunkte bereit, die anderen beiden Komponenten verbinden sich aktiv zum OA. Beim Aktivieren des Netzwerks wird daher nach einer IP-Adresse verlangt, um einen Verbindungsaufbau zu ermöglichen.

Am Beispiel kann man erkennen, das die konkreten Implementierungen ausschliesslich auf Funktionalität der Basisklasse ATEONetwork zurückgreifen. Das erleichtert die Weiterentwicklung und Wartung erheblich. Auch das Hinzufügen von weiteren Verbindungen ist dadurch unproblematisch.

7 DISKUSSION UND AUSBLICK

7.1 DISKUSSION

Beim Zusammenstellen der Arbeitspakete für die Studienarbeit zeichnete sich ab, dass der Aufwand für einige Pakete schwer einzuschätzen ist. Zudem waren die Planungen einiger betroffener Aspekte des OA noch nicht abgeschlossen. Daher konnten nicht alle Pakete begonnen und fertiggestellt werden. Folgende Pakete wurden aus Zeitmangel oder aufgrund ihres Umfangs nicht realisiert:

1. Konfiguration OA

   Da die Planungen zum OA zur Zeit der Studienarbeit noch nicht abgeschlossen waren, erschien es sinnvoller, diesen Teil in die sich anschließende Diplomarbeit zu verlagern. Denn weder die Eigenschaften des Konfigurationsdialogs, noch die der einzelnen GUI-Elemente waren zu diesem Zeitpunkt im vollen Umfang bekannt. Eine Implementierung für die bisher vorhandenen Elemente erschien daher nicht sinnvoll. Im schlechtesten Fall wäre im Rahmen der Diplomarbeit durch die Implementierung der restlichen GUI-Elemente eine Neuimplementierung notwendig gewesen.

2. Konfiguration der Popups

   Aufgrund voranschreitender Planungen wurde dieses Arbeitspaket obsolet.

3. Konfiguration der Sounds

   Aufgrund voranschreitender Planungen wurde dieses Arbeitspaket obsolet.

4. Machbarkeitsanalyse zur Anzeige der Herzratenvariabilität

   Aufgrund voranschreitender Planungen wurde dieses Arbeitspaket obsolet.

Größere Probleme ergaben sich vor allem bei Entwurf und Implementierung der Netzwerkanbindung. Die Verflechtung des OA mit der Funktionalität eines Netzwerkdienstes[1] ist eher nachteilig und sorgt für unnötige Komplexität der Software. Auch ist

1 Der OA nimmt alle Informationen entgegen und verteilt sie wieder an die Abnehmer.

die Trennung von technischen Funktionen und fachlicher Logik nicht immer stringent erfolgt. Aus heutiger Sicht wäre eine eigene Netzwerkkomponente sinnvoll. Ein erster Schritt in diese Richtung wäre die Entkopplung der Netzwerkfunktionalität von den einzelnen Komponenten. Der dafür nötige Aufwand wäre beträchtlich, würde aber Nachnutzbarkeit, Anpassbarkeit und Wartbarkeit noch einmal erheblich erhöhen.

7.2 ERGEBNISSE UND AUSBLICK

7.2.1 *Versionsupdate*

Das Update wurde aufgrund der in Kapitel 4 angesprochenen Probleme nicht durchgeführt. Bis zum Ende der Studienarbeit stellten sich dadurch keine Probleme ein. Trotzdem sollte in künftigen Arbeiten dieser Aufgabe Platz eingeräumt werden. Aktuell ist auf der Webseite des Squeak-Projektes die Version 4.1 verfügbar. Darin sind weitreichende, grundlegende Änderungen vorgenommen worden. Für diese Version steht zudem eine alternative Virtual Machine zur Verfügung, welche durch Verbesserungen eine zwei- bis zehnfache Geschwindigkeitssteigerung erreicht. Gerade mit Blick auf die Performanceprobleme in der Vergangenheit sollte man einen möglichen Versionssprung nicht aus den Augen verlieren.

7.2.2 *GUI-Anpassungen: Joystickausschlag und Verteilung Steuerungsanteil*

Die geplanten Anpassungen konnten ohne Einschränkungen umgesetzt werden. Die beiden Elemente werden im Rahmen der folgenden Diplomarbeit in den zu erstellenden OA integriert. Ein Konfigurationsdialog soll es ermöglichen den OA in verschiedenen Ausbaustufen[2] zusammenzustellen. Dazu müssen auch die beiden GUI-Elemente erweitert werden, dass sie durch den Konfigurationsdialog manipulierbar sind.

Durch die gewonnene Erfahrung, fand auch auch eine Bewertung der geleisteten Arbeiten statt. Aus heutiger Sicht wäre für die Buttons im Element Steuerungsanteil eine eigene Klasse zu empfehlen. Das würde ungefähr einer Halbierung des Codes in der Methode `updateMwiButtons:` bedeuten. Dann wären die Änderungen an den Grafiken der Buttons nur einmal in der zu entwerfenden Klasse implementiert. Das würde die Übersichtlichkeit und das Verständnis fördern. Da die aktuelle Implementierung die Anforderungen erfüllt, ist eine Umstellung nicht dringend, aber langfristig wünschenswert.

2 Ausbaustufen unterscheiden in der Anzahl der angezeigten GUI-Elemente.

7.2.3 *Netzwerkimplementierung*

Die Komponenten zur Netzwerkkommunikation wurden neu entworfen. Auch konzeptuell fand ein Umdenken statt. Es wird nicht länger eine Verbindung je Informationsfluss verwendet. Stattdessen wurde ein Vorgehen realisiert, dass mit einer konstanten Anzahl von Netzwerkverbindungen auskommt. Verschiedene Informationsflüsse mit gleichen Quellen und Zielen teilen sich eine Verbindung. Daher ist die Anzahl abhängig von den beteiligten Kommunikationspartnern und nicht von der Menge der ausgetauschten Informationen.

Möglicherweise könnte eine Analyse der momentan übertragenen Daten Einsparpotential aufzeigen. Beispielsweise werden beide Joystickausschläge der MWB, die vertikale und die horizontale Position des Fahrobjektes übertragen. Dadurch das die letzteren beiden Werte aus den Joystickauslenkungen errechnet werden können, wäre es abzuwägen ob die Auslenkungen allein nicht ausreichen würden. Die Berechnung würde dann nicht mehr nur auf Seiten von SAM erfolgen, sondern auch innerhalb des OA.

In der vorliegenden Version bildet der OA die zentrale Anlaufstelle für Netzwerkkommunikation. Das ist auch aus fachlicher Sicht durchaus korrekt. Trotzdem ist diese Zusatzfunktion im OA der Wartbarkeit und Weiterentwicklung nicht förderlich. Sollten durch künftige Planungen weitere Komponenten zu diesem Setting hinzugefügt werden, wäre es angebracht eine eigenständige Netzwerkkomponente zu entwerfen und zu implementieren.

7.3 QUELLCODE

Der im Rahmen der Studien- und Diplomarbeit entstandene Quellcode wird der Diplomarbeit in Form einer CD beigelegt. Das umfasst die Quellen von SAM, OA und MA.

LITERATURVERZEICHNIS

[1] J. Nachtwei. The many faces of human operators in process control - a framework of analogies. *Theoretical Issues in Ergonomics Science*, pages 1–21, 2010.

[2] Nicolas Niestroj. *Erweiterung des AEO-Systems zur Komplexitätserhöhung von SAM*. Studienarbeit, Berlin, 2008.

[3] Nicolas Niestroj. *Vernetzung im ATEO Projekt aus inhaltlicher und technischer Sicht*. Diplomarbeit, Berlin, 2009.

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

[5] J. Seppälä, A. S. Nissinen, H. Koivo, and M. Laitila. Intelligent visualisation of dynamic process data. *Proceedings of Control Systems 2000, 01.05–04.05 2000*. Online verfügbar unter: http://ae.tut.fi/ jms/Publications/cs2000.pdf [Stand: 21.05.2010].

[6] H. Wandke and J. Nachtwei. The different human factor in automation: the developer behind vs. the operator in action. *D. de Waard, F.O. Flemisch, B. Lorenz, H. Oberheid, and K.A. Brookhuis (Eds.), Human factors for assistance and automation*, pages 493–502, 2008.