Die Gebrauchstauglichkeit des Automatiken-GUI im Projekt Arbeitsteilung Entwickler Operateur (ATEO)

Analyse und Optimierung

Studienarbeit
im Fachgebiet Informatik
Lehrstuhl: Softwaretechnik

eingereicht im	Institut für Informatik Lehr- und Forschungsgebiet Softwaretechnik Mathematisch-Naturwissenschaftliche Fakultät II Humboldt-Universität zu Berlin
von	Nikolai Kosjar
Matrikelnummer	514905
am	19. September 2011
Gutachter	Prof. Dr. sc. nat. Klaus Bothe
Betreuerin	Dipl.-Psych. Saskia Kain

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einführung

1.1. Einbettung der Arbeit im ATEO-Projekt

1.1.1. Das ATEO-Projekt

Das ATEO-Projekt (Arbeitsteilung Entwickler-Operateur) ist ein Beitrag zum interdisziplinären Graduiertenkolleg prometei (Prospektive Gestaltung von Mensch-Technik-Interaktion), welches am Zentrum für Mensch-Maschine-Systeme der Technischen Universität Berlin organisiert ist. Wissenschaftler des Fraunhofer Instituts für Produktionsanlagen und Konstruktionstechnik unterstützen das Graduiertenkolleg, ebenso wie der Lehrstuhl für Ingenieurpsychologie und Kognitive Ergonomie der Humboldt-Universität zu Berlin mit dem ATEO-Projekt.

Im ATEO-Projekt wird die Arbeitsteilung zwischen zwei verschiedenen Gruppen experimentell variiert und empirisch untersucht. Der ersten Gruppe gehören Entwickler bzw. Designer an, die Systeme antizipieren und planen. Der zweiten Gruppe gehören Operateure bzw. Anwender an, welche die implementierten Systeme nutzen. In den Experimenten wird die Antizipation von Ereignissen durch den Einsatz der durch die Entwickler konzipierten Automatiken mit der Reaktion auf Ereignisse von Operateuren verglichen.

Für die Untersuchungen im ATEO-Projekt kommt eine Mikrowelt mit lebendigen Komponenten, den Mikroweltbewohnern, zum Einsatz. Erst die Mikroweltbewohner bringen die nötige Komplexität ins System. Die Mikrowelt ist der zentrale Bestandteil (s. Abbildung 1) des ATEO-Projekts. Sie heißt SAM (Socially Augmented Microworld). Die anderen Komponenten sind der Operateursarbeitsplatz, das AAF-Framework und das Automatiken-GUI. Während SAM, das AAF-Framework und das Automatiken-GUI in den nachfolgenden Abschnitten näher erklärt werden, sei der Operateursarbeitsplatz nur der Vollständigkeit halber erwähnt.

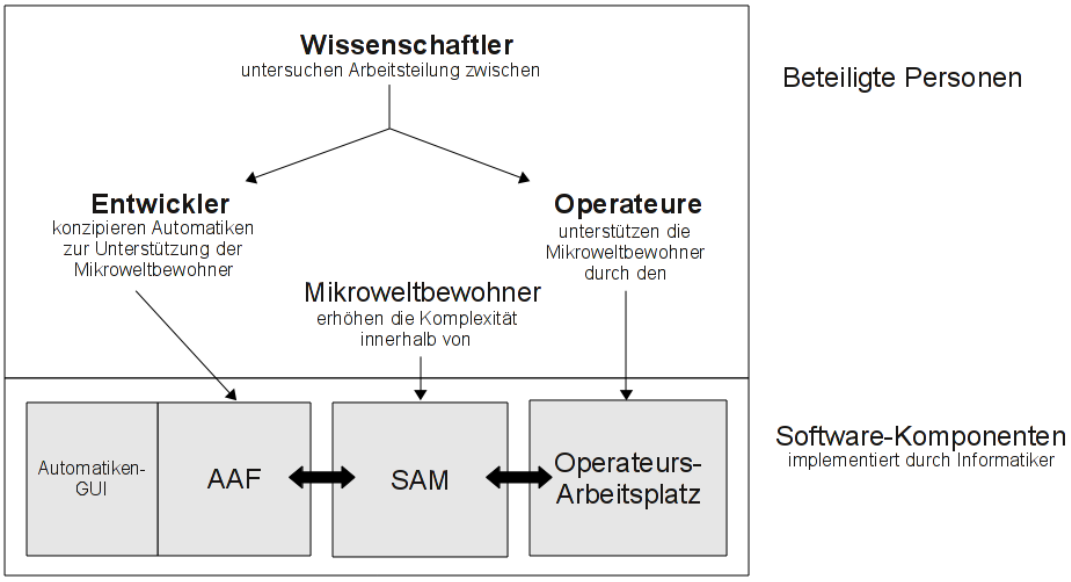

Abbildung 1: Personen und Software-Komponenten im ATEO-Projekt

1.1.2. Socially Augmented Microworld (SAM)

Bei SAM handelt es sich um eine Mikrowelt, bei der die Mikroweltbewohner die Aufgabe haben, ein rundes Objekt entlang einer vorgegebenen Strecke zu navigieren. Dabei haben beide Mikroweltbewohner Einfluss auf die Steuerung des Objekts.

Die Aufgabe besteht darin, das Objekt möglichst schnell und möglichst genau entlang der Strecke zu navigieren. Ein Mikroweltbewohner soll die Priorität auf das genaue Fahren legen, der andere auf das schnelle Fahren.

Die Tracking-Aufgabe wird zusätzlich erschwert, indem durch Gabelungen und Hindernisse auf der Strecke Konfliktsituationen hervorgerufen werden. Abbildung 2 vermittelt einen Eindruck von SAM.

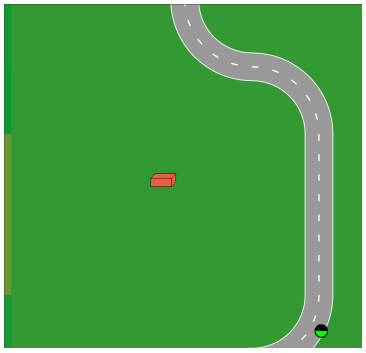

Abbildung 2: SAM - ein sich näherndes Hindernis verursacht eine Konfliktsituation

1.1.3. AAF-Framework

Um den Beitrag der Entwickler in die Untersuchungen einzubringen, konfigurieren die Forscher die von den Entwicklern konzipierten Automatiken. Diese sind im Rahmen einer Untersuchung von Psychologin Kain [11] entstanden und werden zur Zeit durch eine Gruppe von Informatikern unter ihrer Anleitung implementiert.

Eine Automatik soll die Mikroweltbewohner bei der Umsetzung ihrer Ziele unterstützen. So kann eine Automatik zum Beispiel verhindern, dass die Mikroweltbewohner das Objekt abseits der Strecke führen. Denkbar ist auch, dass auftretende Konfliktsituationen durch die Anzeige von Hinweisen entschärft werden.

Um Automatiken in SAM integrieren zu können, hat Hasselmann in seiner Diplomarbeit (Arbeit in Anfertigung) das AAF (ATEO Automation Framework) entworfen. Automatiken sind Graphen und bestehen aus Knoten, die entweder Funktionen oder andere Automatiken sind. Ein Graph enthält zwei besondere Knoten - einen Anfang und ein Ende. Jeder Knoten, Anfang und Ende ausgenommen, hat genau einen Vorgänger und genau einen Nachfolger. Über den Anfang kommt ein Teilzustand von der zentralen SAM-Datenstruktur SAMModelData in den Graphen - der SAMState. Die direkten Kinderknoten des Anfangs erhalten jeweils eine Kopie des SAMState und können ihn manipulieren und diesen weiter an ihre direkten Kinderknoten weitergeben, welche weitere Änderungen vornehmen können[1]. Dieser Vorgang setzt sich fort, bis die veränderten und weitergereichten SAMStates schließlich im Ende „münden“. Das Ende erzeugt einen SAMState, welcher in SAM eingespeist wird und dessen Werte in SAMModelData übernommen werden - auf diese Weise haben die Funktionen Einfluss auf die Verarbeitung in SAM. Eine Besonderheit des AAF ist es, dass jeder Knoten wiederum ein Graph, d.h. eine andere Automatik, sein kann. Da jeder Graph genau einen Anfang und ein Ende hat, und jeder Knoten in einem Graphen genau einen Vor- und genau einen Nachfolger hat, kann ein beliebiger Knoten durch einen Graphen ersetzt werden. Auf die Art und Weise können selbst komplexe Automatiken entwickelt werden.

Funktionen und Automatiken können demnach zu neuen Automatiken zusammengesetzt werden. Um das auch ohne Programmierkenntnisse durchführen zu können, wurde das Automatiken-GUI „AAF Graph Tool“ entwickelt.

1.1.4. Automatiken-GUI „AAF Graph Tool“

Im Rahmen ihrer Diplomarbeit [6] hat Fuhrmann eine grafische Benutzungsschnittstelle entwickelt, mit der Forscher für ihre Experimente komplexe Automatiken erstellen und konfigurieren können (vgl. Abbildung 3) - das AAF Graph Tool.

Im Einzelnen können mit dieser Anwendung Automatiken erstellt, gespeichert, geladen und bearbeitet werden. Zum Bearbeiten gehört auch das Verändern von Parametern einzelner Funktionen sowie die Aktivierung bzw. Deaktivierung von Funktionen

[1] Funktionen können demnach auch Änderungen ihrer Vorgänger verändern.

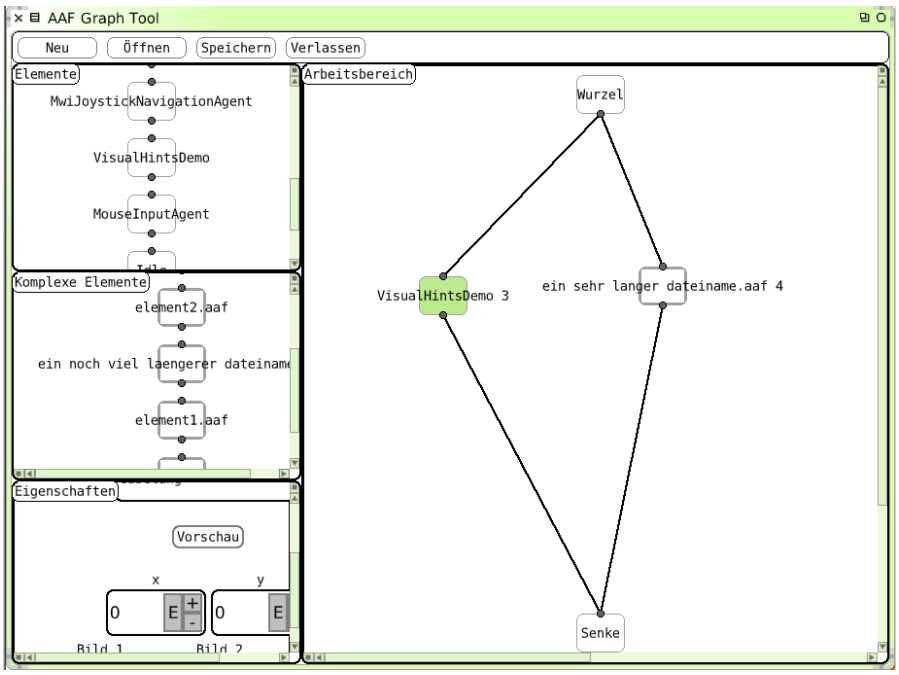

Abbildung 3: Das von Fuhrmann [6] entwickeltes Automatiken-GUI „AAF Graph Tool“

oder Automatiken für einzelne Streckenabschnitte. Das Erstellen von Automatiken wird durchgeführt, indem Funktionen und bisher erstellte Automatiken als Bausteine in einem Graphen zusammengesetzt werden.

1.1.5. Umfang dieser Arbeit

Ein Wunschkriterium bei der Anforderungsanalyse von Fuhrmann für das Automatiken-GUI „AAF Graph Tool“ war die Gestaltung des Tools nach software-ergonomischen Gesichtspunkten[2]. Durch den zeitlich begrenzten Rahmen konnte dieses Wunschkriterium nicht hinreichend umgesetzt werden.

Das Ziel dieser Arbeit ist die Umsetzung dieses Wunschkriteriums. Das Automatiken-GUI soll nach software-ergonomischen Gesichtspunkten aufgewertet werden. Das soll erreicht werden, indem einerseits allgemeine Ratschläge, Heuristiken und Prinzipien zur Verbesserung der Gebrauchstauglichkeit aus der Literatur befolgt werden und andererseits, indem Hinweise der Psychologin Kain umgesetzt werden. Zu der verwendeten Literatur sei auf Abschnitt 2.4.3 und 2.4.4 verwiesen.

Durch die Aufwertung des Automatiken-GUI werden auch zwei weitere Ideen von Fuhrmann[3] umgesetzt:

[2] vgl. [6], Abschnitt 4.2.2, S. 36
[3] vgl. [6], Abschnitt 5.2, S. 79ff.

- Implementierung von „Individuellen Element-Paletten“[4]
  Element-Paletten tragen zur Übersicht bzw. Organisation von Automatiken bei, indem sie die Gruppierung von Funktionen im AAF Graph Tool erlauben sollen. Das wurde für komplexe Elemente umgesetzt, indem diesen mehrere Kategorien zugeordnet werden können. Bei der Anzeige der Elemente kann nach den Kategorien gefiltert werden.
- Implementierung von „Einheitlichen grafischen Bedienelementen“[5]

Zum Umfang dieser Arbeit gehört auch die Aktualisierung von Dokumenten, die sich auf das Automatiken-GUI beziehen. Dazu zählt die Bedienungsanleitung des GUI (Anhang A), die Integrationsanleitung von Funktionen in die GUI (Anhang B) und die Prüfliste von manuellen Testfällen (Anhang C).

Folgender Hinweis sei noch gegeben um Verwirrung entgegenzuwirken. Fuhrmann hat im Rahmen ihrer Diplomarbeit [6] die Aktivierung bzw. Deaktivierung von Funktionen für bestimmte Streckenabschnitte implementiert. Diese Idee wird auf allgemeine Situationen ausgeweitet und wird derzeit im Rahmen einer laufenden Diplomarbeit (Nikolai Kosjar, genauer Titel noch nicht festgelegt) überarbeitet. Auf einigen Abbildungen sind Ergebnisse dieser Arbeit sichtbar, wie zum Beispiel Abbildung 14 auf Seite 40 oder 20 auf Seite 49. Der Leser soll daher nicht verwirrt sein, wenn er die alte Konfiguration für die Aktivierung bzw. Deaktivierung von Streckenabschnitten in der neuen Automatiken-GUI nicht wiedererkennt.

1.2. Gliederung dieser Arbeit

In Kapitel 2 werden die Voraussetzungen geklärt. Neben einer kurzen Vorstellung von Smalltalk und Squeak, der im ATEO-Projekt verwendeten Programmiersprache und Entwicklungsumgebung, wird auch auf einige Richtlinien zur Gestaltung von Benutzungsschnittstellen eingegangen.

In Kapitel 3 ist das Vorgehen für die Analyse beschrieben. Es werden auch die Benutzer des AAF Graph Tools charakterisiert und für die Nachvollziehbarkeit wird die untersuchte Version genannt.

Kapitel 4 ist das Ergebnis der Analyse. Hier findet sich eine Liste von allen identifizierten Problemen.

Kapitel 5 widmet sich der Behebung der in Kapitel 4 beschriebenen Probleme. Neben der Beschreibung der eigentlichen Behebung wird auch (sofern sinnvoll) erklärt, wieso diese Behebung besser ist als eine andere.

Kapitel 6 hebt die wichtigsten Änderungen dieser Arbeit hervor und zeigt Probleme auf.

Es sei auch auf den Anhang hingewiesen. Dieser enthält die erwähnten, aktualisierten Dokumente (Anahng A, B und C) und einige Hinweise zur Umsetzung (Anhang D).

[4] vgl. [6], Abschnitt 5.2.4, S. 81
[5] vgl. [6], Abschnitt 5.2.9, S. 84

2. Voraussetzungen

2.1. Smalltalk

„Smalltalk“ bezeichnet eine Programmiersprache, die zugehörige Entwicklungsumgebung sowie die virtuelle Maschine, mit deren Hilfe die Smalltalk-Programme ausgeführt werden. Je nach Literatur wird daher auch von „Smalltalk-Systemen“ geredet. Durch verschiedene Implementierungen von Smalltalk-Systemen gibt es inzwischen auch nicht eine Smalltalk-Programmiersprache, viel mehr hat sich eine Gruppe von Sprachdialekten entwickelt.[6]

Das erste Smalltalk-System wurde in den 70er Jahren im Palo Alto Research Center (PARC) von Xerox entwickelt. Nach fünf mehrjährigen Entwicklungszyklen wurde das System 1981 unter dem Namen Smalltalk-80 als erste kommerzielle Version von Xerox veröffentlicht. Adele Goldberg, Daniel Ingalls und Alan Key waren die wichtigsten Entwickler.[7]

Die Smalltalk-Programmiersprache basiert nur auf wenigen Konzepten, sodass die Sprache relativ schnell gelernt und verstanden werden kann. Es ist eine dynamisch typisierte[8] und rein objektorientierte Sprache. Die Aussage „Alles ist ein Objekt“ gilt demnach z. B. auch für Integer und Zeichen. Mit Smalltalk wurde Pionier-Arbeit an grafischen Benutzungsschnittstellen geleistet. Sehr populär wurde u. a. das „Model-View-Controller“-Konzept (MVC), welches von Trygve Reenskaug für eine klare Trennung von eigentlicher Funktionalität („Business-Logic“) und Benutzungsschnittstelle eines Programms entwickelt wurde.[9]

Zu den bekannten kommerziellen Smalltalk-Systemen zählen „VisualAge Smalltalk“ von IBM und „Visual Works“ von Cincom.[10] In der Open-Source-Welt ist „GNU Smalltalk“ und „Squeak“ verbreitet. Letzteres findet auch im ATEO-Projekt Verwendung.

2.2. Squeak[11]

Squeak ist eine Open-Source-Implementierung eines Smalltalk-Systems. Es entstand bei Apple Inc. durch eine Forschungsgruppe, welche Prototypen für Lernsoftware und Experimente mit Benutzungsschnittstellen entwickelte. Daneben hatte die Forschungsgruppe auch zum Ziel das Dynabook-Konzept von Alan Kay umzusetzen. Dieses Konzept beschreibt einen tragbaren Computer, der speziell für Kinder aller Altersklassen gedacht ist. Neben Alan Kay selbst zählen Dan Ingalls, Ted Kaehler und Scott Wallace zu den wichtigsten Entwicklern.

[6] vgl. [8], Abschnitt 1.2 (S. 1f).
[7] vgl. [17], Abschnitt 2.1 (Seite 21f.); [8], Abschnitt 1.1 (S. 1)
[8] Der Typ einer Variablen wird erst bei der Ausführung bestimmt.
[9] vgl. [7], Vorwort (S. viii); [8], Abschnitt 1.1 (S. 1)
[10] vgl. [17], Abschnitt 2.2 (Seite 22f.)
[11] vgl. [16], insbesondere die „About“-Seiten

Dem Dynabook-Konzept folgend bietet Squeak die Möglichkeit jeden Teil des Smalltalk-Systems einzusehen und zu studieren, auch die virtuelle Maschine, welche als Besonderheit ebenfalls in Smalltalk implementiert ist.

Multimedia wird von Squeak besonders unterstützt. Durch Plugins für die virtuelle Maschine stehen Features wie z. B. 2D mit Kantenglättung, beschleunigtes 3D, Echtzeit Sound- und Musiksynthese, Unterstützung für MPEG2-Videos, und viele andere mehr zur Verfügung. Insgesamt existieren über 750 einfach nachinstallierbare Pakete[12].

Die Benutzeroberfläche von Squeak heißt „Morphic“. Auf diese wird im nächsten Abschnitt eingegangen. Es lassen sich aber weiterhin Benutzeroberflächen erstellen, welche auf dem traditionellen MVC-Konzept basieren. Tatsächlich kann der Benutzer beides kombinieren.

Für eine vollständige Liste von Features sei auf die offizielle Squeak-Webseite http://www.squeak.org/ (abgerufen am 13. September 2011, 15: 16 Uhr) verwiesen.

2.3. Das Morphic-Framework von Squeak[13]

Das Morphic-Framework von Squeak gestattet es auf sehr einfache Art und Weise interaktive und lebendige Benutzeroberflächen zu erstellen. Es verfügt über vernünftige Voreinstellungen und nimmt dem Entwickler viel Arbeit ab, wenn es um Drag & Drop, Animationen, automatisches Layout von Morphen und Ereignisverarbeitung geht.

Das zentrale Element im Morphic-Framework ist der „Morph“ (griechisch für Gestalt bzw. Form). Wird ein neuer Morph erstellt und angezeigt, z. B. mit

```
Morph new openInWorld.
```

dann erscheint ein blaues Rechteck, wie in der Abbildung 4 links zu sehen ist. Dieser Morph ist jedoch kein flüchtiges Rechteck, welches verschwindet, sobald ein Fenster drüber und wieder weggeschoben wird. Es ist ein interaktives Objekt. Per Drag & Drop kann zum Beispiel seine Position verändert werden. Weitere Einstellungen zur Manipulation des Morphs erhält der Benutzer, wenn er den zugehörigen „Halo-Morph“ anzeigen lässt - z. B. durch drücken der mittleren Maustaste auf den Morph. Es erscheinen dann um den Morph verschiedene, anklickbare Buttons mit Icons (in Abbildung 4 rechts sichtbar). Verweilt der Benutzer mit der Maus über einen Button, so erscheint ein Tooltip, der die zugehörige Manipulation erklärt. Zu den Manipulationen zählen u. a. Rotation, Kollabieren, Entfernen, Menü aufrufen, Verschieben, Duplizieren, Debuggen, Farbe ändern, Vergrößern. Der grüne Morph rechts in der Abbildung 4 ist durch Duplikation und Farbänderung entstanden. Weitere Manipulationsmöglichkeiten erschließen sich dem Benutzer, wenn er das Menü aufruft (das zugehörige Halo-Icon ist rot und zeigt ein kleines Menü). Ein Entwickler der einen Morph erstellt, kann dem Menü individuelle Einträge hinzufügen.

[12] vgl. http://map.squeak.org/
[13] vgl. [12] und [13]

Abbildung 4: Links: Ein Morph in seinen Standardeinstellungen; Rechts: Morph mit veränderter Hintergrundfarbe und angezeigtem Halo

Um eigene Morphe oder Bedienelemente zu erstellen, kann der Benutzer wie eben beschrieben vorgehen: Einen Morph erzeugen und ihn solange über den Halo und das Menü verändern, bis er den Wünschen entspricht. Morphe können auch interaktiv verschachtelt bzw. zusammengesetzt werden[14]. Auf diese Art und Weise lassen sich Morphe ohne Programmierkenntnisse erstellen. Das ist z. B. zum Erstellen von GUI-Prototypen sehr nützlich. Für weiterführende Möglichkeiten, wie z. B. Animationen oder dem Verhalten bei bestimmten Mausinteraktionen oder Tastatureingaben, muss Quelltext geschrieben werden. Dazu wird eine neue Klasse erstellt, die von der Klasse Morph erbt. Die Methoden, die hinzugefügt oder überschrieben werden müssen, sind üblicherweise sehr klein, sodass mit relativ wenig Quellcode viel im Morphic-Framework erreicht werden kann. Auch lässt sich durch die Definition einer eigenen Klasse das Aussehen des Morphs sehr individuell gestalten: In der Methode #drawOn: lassen sich auf einem FormCanvas die üblichen Zeichenoperationen (Zeichnen von u. a. Polygonen, Kurven, Ellipsen, Linien, Bildern und Text) durchführen.

Die Oberfläche von Squeak ist auch in Morphic implementiert. Der Hintergrund (Klasse PasteUpMorph), das Fenster-System (Klasse SystemWindow) und der Mauszeiger (Klasse HandMorph) sind Morphe. Viele Tools von Squeak, zum Beispiel der Browser, beruhen aber teilweise noch auf dem MVC-Konzept. Diese Tools zeigen, dass sich beide Frameworks gemeinsam verwenden lassen[15]. Im MVC-Kontext betrachtet, übernimmt ein Morph die View- und Controller-Rolle. Der Morph kann zugleich aber auch das Model sein, muss es aber nicht, denn durch das Observer-Pattern kann über #changed: und #update: weiterhin eine klare Trennung der Darstellung und Interaktion (View und Controller) vom Model gewährleistet werden.

2.3.1. Layouts

Morphe lassen sich auch zueinander positionieren, indem Eltern-Kind-Beziehungen zwischen den Morphen definiert werden. Jeder Morph kann mehrere *Submorphe* (Kindknoten) enthalten und jeder Submorph hat auch Referenz auf seinen *Owner* (Elternknoten).

[14] Mehr hierzu, jedoch aus programmiertechnischer Sicht, folgt in den nächsten Abschnitten.
[15] Dazu wurden die typischen View- und Controller-Klassen in entsprechende Morphe eingebettet.

Wie die Submorphe im Owner positioniert werden, hängt von der *LayoutPolicy* des Owners ab.

Falls keine LayoutPolicy eingestellt ist (Voreinstellung), behalten die Submorphe ihre Position relativ zu ihren Ownern: Wird der Owner verschoben, verschieben sich seine Submorphe mit ihm. Bei Vergrößerung oder Verkleinerung des Owners behalten die Submorphe ihre Position bei. Dabei kann es passieren, dass die Eltern-Kind-Beziehung zwischen den Submorphen und Owner nicht offensichtlich ist, z. B. wenn der Owner kleiner als der Submorph ist oder wenn der Submorph außerhalb des Owners liegt.

Mit der LayoutPolicy *TableLayout* können die Submorphe in einem Gitter angeordnet werden. Mit nur einer Zeile werden die Morphe ausschließlich nebeneinander angeordnet, mit ausschließlich einer Spalte nur untereinander. Zeilen- und Spaltenanzahl werden nicht fest vorgegeben, sondern passen sich dynamisch an: Wird der Owner breiter gezogen, werden gegebenenfalls Submorphe aus der zweiten in die erste Zeile „rutschen“.

Die zweite LayoutPolicy ist *ProportionalLayout*. Mit dieser Policy steht einem Submorph in seinem Owner eine definierte Fläche zur Verfügung, die proportional mitverändert wird, wenn die Größe des Owners verändert wird. Mit ProportionalLayout kann aber auch die Einschränkung gemacht werden, dass ein Submorph in Höhe und/oder Breite fix bleibt bei Größenveränderung des Owners.

TableLayout und PorportionalLayout sollen an einem Beispiel verdeutlicht werden. Mit einem ProportionalLayout soll ein Morph in drei Bereiche aufgeteilt werden. Dazu wird der Morph *parent* erstellt: Seine Größe wird festgelegt, seine LayoutPolicy auf ProportionalLayout gesetzt und schließlich wird er angezeigt.

```
parent := Morph new extent: 175@100.
parent layoutPolicy: ProportionalLayout new.
parent openInWorld.
```

Der obere Bereich ist grün und hat eine fixe Höhe, passt aber seine Breite bei Größenveränderung an:

```
parent
    addMorph: (Morph new color: Color green darker)
    fullFrame: (LayoutFrame fractions: (0@0 corner: 1@0)
                            offsets: (0@0 corner: 0@25)).
```

Mit dem LayoutFrame wird der Bereich spezifiziert. „fractions:“ wird ein Rechteck übergeben, welches durch zwei Punkte definiert wird. Der erste Punkt ist die linke obere Ecke des Rechtecks, der zweite die rechte untere Ecke. Der Wertebereich ist jeweils [0:1] da hiermit die Proportionen des Bereichs festgelegt werden. Betrachten wir zuerst die X-Koordinaten der Punkte in „(0@0 corner: 1@0)“: Der Bereich spannt sich von ganz links (0) bis ganz rechts (1) auf. Da die Y-Koordinaten jeweils 0 sind, hat dieser Bereich die Höhe 0 und wird dabei bei Größenveränderung des Owners nicht angepasst. Das mit „fractions:“ übergebene proportionale Rechteck kann mit „offsets“ weiter angepasst werden. „offsets: (0@0 corner: 0@25)“ spezifiziert, dass weder die linke noch die rechte

Grenze des Rechtecks verschoben werden soll (jeweils 0 in der X-Koordinate). Vertikal soll die obere Grenze beibehalten werden (0 in der Y-Koordinate des ersten Punktes), während die untere Grenze um 25 Pixel nach unten verschoben wird - dadurch erhält das Rechteck, welches die Höhe von 0 hatte, die fixe Höhe 25.

Links kommt ein weiterer Bereich hinzu, der unterhalb des grünen Bereichs positioniert wird und rot ist. Dieser Bereich verändert horizontal seine Größe mit, er nimmt stets 20% der Breite ein:

```
parent
    addMorph: (Morph new color: Color red darker)
    fullFrame: (LayoutFrame fractions: (0@0 corner: 0.2@1)
                            offsets: (0@25 corner: 0@0)).
```

Schließlich fügen wir einen weiteren Bereich hinzu, der den Rest der zur Verfügung stehenden Fläche von parent einnimmt. Dieser Bereich hat die LayoutPolicy TableLayout und enthält fünf weitere Morphe fester Größe. Diese werden von links nach rechts im Gitter platziert (listDirection) und rutschen bei Bedarf in die nächste Zeile (wrapDirection).

```
main := (Morph new color: Color yellow darker).
main layoutPolicy: TableLayout new;
    listDirection: #leftToRight;
    wrapDirection: #topToBottom;
    layoutInset: 10;
    cellInset: 10.
    5 timesRepeat: [ main addMorphBack: Morph new ].
parent
    addMorph: main
    fullFrame: (LayoutFrame fractions: (0.2@0 corner: 1@1)
                            offsets: (0@25 corner: 0@0)).
```

Der hiermit erstellte Morph kann in der Abbildung 5 links oben betrachtet werden. Der Morph rechts oben unterscheidet sich nur durch seine Größe. Durch das ProportionalLayout behält der grüne Bereich eine konstante vertikale Höhe, während er sich horizontal anpasst. Der rote Bereich nimmt stets 20% der Breite ein. Durch die Größenveränderung passt sich auch das TableLayout des gelben Bereiches an und die blauen Morphe werden umgeordnet.

2.3.2. Skalierung

Ein mit `Morph new` erstellter Morph hat eine Größe von 50 x 40 Pixeln. Wird der Morph von einem TableLayout verwaltet, so lässt sich für diesen jeweils für die Vertikale als auch Horizontale festlegen, ob er mehr Fläche einnehmen soll, falls diese vorhanden ist.

In der Voreinstellung bleibt die Größe des Morphs bei Größenveränderung des Owners konstant. Das entspricht einem Aufruf von `aMorph hResizing: #rigid; vResizing: #rigid.`

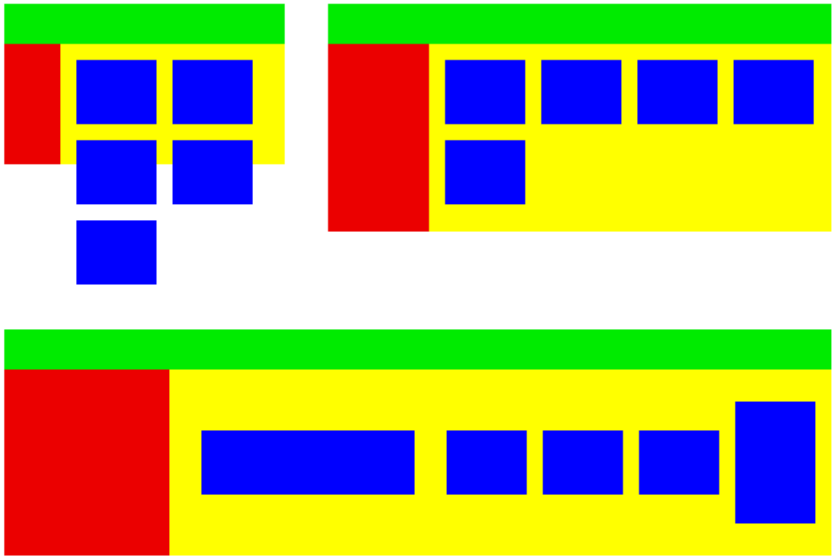

Abbildung 5: Mit ProportionalLayout und Tablelayout sowie der Skalierung von einzelnen Morphen sind dem Entwickler beim Layout kaum Grenzen gesetzt.

Wird #spaceFill übergeben, so wird der zur Verfügung stehende Platz eingenommen. Mit #shrinkWrap nimmt der Morph gerade die Größe ein, die notwendig ist, um alle seine Submorphe zu beherbergen.

In Abbildung 5 unten wurde im gelben Bereich für den ersten blauen Morph die Einstellung hResizing: #spaceFill verwendet, für den letzten blauen Morph dagegen die Einstellung vResizing: #spaceFill.

2.4. Gestaltung von und Interaktion mit Benutzungsschnittstellen

2.4.1. Software-Ergonomie

Nach Dahm ist *Ergonomie* die Lehre von der menschlichen Arbeit, ihrer Beschreibung, Modellierung und Verbesserung. Die Ergonomie stellt dabei nicht die Arbeit in den Vordergrund, sondern den Menschen, seine (Arbeits-)Aufgaben und seine Bedürfnisse. Dabei geht es nicht nur um den Schutz vor negativen Einwirkungen (Verletzungen, unzumutbaren Arbeitsbedingungen, psychische Einwirkungen wie z. B. Stress, Anpassung oder Überforderung), sondern auch darum, den Menschen bei seiner Arbeit zu unterstützen, zu fördern und zufrieden zustellen.[16]

Übertragen auf Software, bedeutet *Software-Ergonomie* nach Heinecke das Streben nach

[16] vgl. [4], Abschnitt 2.1, S. 28

der Anpassung der Eigenschaften eines Software-Systems an die physischen und psychischen Eigenschaften der damit arbeitenden Menschen.[17]

2.4.2. Gebrauchstauglichkeit und ihre Leitziele

Im Rahmen der Software-Ergonomie ist der Begriff der Gebrauchstauglichkeit (engl. Usability) von zentraler Bedeutung. Die Gebrauchstauglichkeit wird in der EN ISO 9241-11 definiert:

> Das Ausmaß, in dem ein Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen.

Die Gebrauchstauglichkeit ist also abhängig von den Benutzern, ihren Zielen und dem Nutzungskontext. Eine Software, die von anderen Benutzern als intendiert eingesetzt wird, wird von diesen höchstwahrscheinlich mit einer geringen Gebrauchstauglichkeit bewertet. Aus diesem Grund ist es bei der Software-Entwicklung sehr wichtig, die zukünftigen Benutzer zu kennen.

Effektivität, Effizienz und Zufriedenheit seien nochmal genauer betrachtet:

Effektivität *Die Genauigkeit und Vollständigkeit, mit der Benutzer ein bestimmtes Ziel erreichen.*

Effizienz *Der im Verhältnis zur Genauigkeit und Vollständigkeit eingesetzte Aufwand, mit dem Benutzer ein bestimmtes Ziel erreichen.*

Zufriedenheit *Freiheit von Beeinträchtigungen und positive Einstellung gegenüber der Nutzung des Produkts.*

Ist eine Software effektiv, dann bietet es dem Benutzer alle notwendigen Funktionen um seine Aufgabe zu erfüllen. Ist die Software zusätzlich effizient, dann kann der Benutzer sein Ziel mit relativ geringem Aufwand, z. B. in kurzer Zeit, erreichen. Damit eine Software den Benutzer zufrieden stellt, muss sie auf ihn und seine Aufgaben zugeschnitten sein.

2.4.3. Grundsätze der Dialoggestaltung

In der Norm „DIN EN ISO 9241: Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten“ werden in Teil 110 die Grundsätze der Dialoggestaltung beschrieben. Ein Dialog wird definiert als

> Interaktion zwischen einem Benutzer und einem interaktiven System in Form einer Folge von Handlungen des Benutzers (Eingaben) und Antworten des interaktiven Systems (Ausgaben), um ein Ziel zu erreichen.

[17] vgl. [9], Abschnitt 2.1.2, S. 40

Dazu zählen auch navigierende Handlungen des Benutzers.

Die Grundsätze sind allgemein gehalten und sind somit unabhängig von „spezieller Technologie oder Technik“. Sie werden im Folgenden zitiert und die wichtigen Aspekte werden hervorgehoben.

Aufgabenangemessenheit *Ein Dialog ist aufgabenangemessen, wenn er den Benutzer unterstützt, seine Arbeitsaufgabe effektiv und effizient zu erledigen.*

Hierzu zählt, u. a. dass nur relevante Informationen angezeigt werden, Ein- und Ausgaben beispielsweise regionalen oder nationalen Konventionen entsprechen (z. B. Währungen), sinnvolle Voreinstellungen getroffen werden und dass keine unnötigen Dialogschritte[18] vom Benutzer gemacht werden müssen. Es sind vernünftige Voreinstellungen zu wählen und wiederkehrende Aufgaben sollte das System unterstützen.

Selbstbeschreibungsfähigkeit *Ein Dialog ist selbstbeschreibungsfähig, wenn jeder einzelne Dialogschritt durch Rückmeldung des Dialogsystems unmittelbar verständlich ist oder dem Benutzer auf Anfrage erklärt wird.*

Beispielsweise sollte der Benutzer während des Dialogs möglichst nicht eine Information erst extern nachschlagen müssen. Für numerische Eingaben sollte demnach die Einheit angezeigt werden. Benutzern sollte klar sein, wie viele Schritte im Dialog noch vor ihm liegen und es sollte eine einheitliche Terminologie verwendet werden.

Erwartungskonformität *Ein Dialog ist erwartungskonform, wenn er konsistent ist und den Merkmalen des Benutzers entspricht, z. B. seinen Kenntnissen aus dem Arbeitsgebiet, seiner Ausbildung und seiner Erfahrung sowie den allgemein anerkannten Konventionen.*

Der Benutzer sollte möglichst nicht (negativ) überrascht werden. Hierzu gehört, dass sprachliche und kulturelle Konventionen eingehalten werden, dass das Vokabular des Benutzers Verwendung findet und dass der Benutzer eine Rückmeldung bekommt, wenn das System länger benötigt als er erwartet. Die Art und Länge von Rückmeldungen sollte angemessen sein. Konsistenz bei ähnlichen Arbeitsschritten führt ebenfalls zu erhöhter Erwartungskonformität.

Lernförderlichkeit *Ein Dialog ist lernförderlich, wenn er den Benutzer beim Erlernen der Nutzung des Dialogsystems unterstützt und anleitet.*

Der Benutzer sollte sein Ziel mit minimalem Lernaufwand erreichen. Hier sind

[18] In [3] werden diese unnötigen Dialogschritte als „Rüstaufgaben“ bezeichnet. Dazu zählt z. B. das Manipulieren von Fenstern - es bringt den Benutzer seinem Ziel nicht näher, sondern ist nur ein notwendiger Schritt um eine Aktion im Programm auszuführen. Zu den Rüstaufgaben zählt auch die Navigation innerhalb des Programms.

u. a. sinnvolle Voreinstellungen, ein UNDO[19]- und ein Hilfe-System zu empfehlen. Aus Rückmeldungen und Erläuterungen sollte der unerfahrene Benutzer ebenfalls lernen können[20].

Steuerbarkeit *Ein Dialog ist steuerbar, wenn der Benutzer in der Lage ist, den Dialogablauf zu starten sowie seine Richtung und Geschwindigkeit zu beeinflussen, bis das Ziel erreicht ist.*

Der Benutzer sollte die Möglichkeit haben seine Aufgabe zu unterbrechen (z. B. durch Abspeichern einer „Sitzung“) und zu einem späteren Zeitpunkt wiederaufzunehmen. Sinnvolle Voreinstellungen sollten durch den Benutzer ebenfalls veränderbar sein und bei vielen anzuzeigenden Daten sollte die angezeigte Menge regulierbar sein.

Fehlertoleranz *Ein Dialog ist fehlertolerant, wenn das beabsichtigte Arbeitsergebnis trotz erkennbar fehlerhafter Eingaben entweder mit keinem oder mit minimalem Korrekturaufwand seitens des Benutzers erreicht werden kann.*

Als Mittel für Fehlertoleranz zählt Fehlererkennung und –vermeidung. Wenn es zu einem Fehler kommt (Fehlererkennung), sollte der Benutzer ausreichend informiert werden, um den Fehler zu beseitigen. Dazu gehört, dass der Benutzer an die fehlerhafte Stelle verwiesen wird. Fehler im Zusammenhang mit dem Wertebereich einer Eingabe können bei numerischen Eingaben beispielsweise über ein Spinbox-Widget verhindert werden. Durch die vorgegebenen Werte hat der Benutzer keine Möglichkeit eine fehlerhafte Eingabe zu tätigen. Eine automatisch durchgeführte Fehlerkorrektur sollte dem Benutzer kenntlich gemacht werden und er sollte die Möglichkeit haben diese rückgängig zu machen.

Individualisierbarkeit *Ein Dialog ist individualisierbar, wenn das Dialogsystem Anpassungen an die Erfordernisse der Arbeitsaufgabe sowie an die individuellen Fähigkeiten und Vorlieben des Benutzers zulässt.*

Sprache sowie kulturelle Eigenheiten verlangen nach Einstellungsmöglichkeiten. Farben und Schriftgrößen sollten ebenfalls anpassbar sein. Wo möglich, sollten mehrere Wege existieren um eine Aktion auszuführen (für neue Benutzer z. B. ein Menüeintrag, für erfahrene Benutzer eine Tastenkombination). Format und Typen von Eingabe- und Ausgabeoperationen sollten ebenfalls einstellbar sein. Schließlich sollten alle Einstellungen (auf Voreinstellungen) zurückgesetzt werden können.

Diese Grundsätze überlappen sich. In einigen Fällen schließen sie sich sogar teilweise aus. Es sollte immer von der Benutzergruppe, dem Kontext und der eingesetzten Dialogtechnik abhängig gemacht werden, welcher Grundsatz vorzuziehen ist.

[19] Mit einem UNDO-System kann der Benutzer durchgeführte Aktionen rückgängig machen.

[20] vgl. [10], Kapitel 1, S. 19ff: Hier wird von einem „Conceptual Model“ gesprochen. Es ist das Modell der Anwendung, die der Benutzer verstehen soll. Rückmeldungen und Erläuterungen helfen dem Benutzer dieses Modell zu erfassen.

2.4.4. Richtlinien

Von den ersten Tagen an haben Designer von Benutzungsschnittstellen Richtlinien aufgeschrieben um ihre Einsichten festzuhalten. Die frühen Richtlinien von Apple und Microsoft haben viele Designer beeinflusst und finden teilweise heute im Web und auf mobilen Geräten Anwendung.

Inzwischen gibt es eine beachtliche Anzahl von Richtlinien, Regeln, Heuristiken und Prinzipien. Obwohl sich vieles überschneidet, auch mit den Grundsätzen der Dialoggestaltung (vgl. 2.4.3), haben alle ihren Wert, da sie unterschiedliche Aspekte betonen. Die für diese Arbeit durchgesehenen Richtlinien seien im Folgenden aufgelistet. Wo anwendbar, wird in nachfolgenden Teilen der Arbeit auf einzelne Richtlinien und Grundsätze der Dialoggestaltung verwiesen.

Die durchgesehenen Richtlinien sind:

1. „Eight golden rules of interface design“ von Ben Shneiderman und Catherine Plaisant (vgl. [15], Abschnitt 2.3.4)
2. „Ten Usability Heuristics“ von Jakob Nielsen (vgl. [14])
3. Dahms sieben Ergänzungen zu Nielens Heuristiken und Shneidermans acht goldenen Regeln (vgl. [4], Abschnitt 8.4, S. 157)
4. Designprinzipien von Alan Cooper, Robert Reimann und David Cronin (vgl. [3], Anhang A)
5. „Basic Principles“ von Jeff Johnson (vgl. [10], Kapitel 1)

3. Analyse

Das Ziel der Analyse ist es Probleme zu identifizieren, die nicht mit der Software-Ergonomie, insbesondere einem hohen Maß von Gebrauchstauglichkeit des Automatiken-GUI, vereinbar sind.

Der nachfolgende Abschnitt beschreibt kurz das gewählte Vorgehen für die Analyse. Um die „richtigen“ Probleme zu identifizieren, müssen die „richtigen“ Benutzer soweit wie möglich charakterisiert werden. Das geschieht in Abschnitt 3.2. Schließlich wird zur Nachvollziehbarkeit in Abschnitt 3.3 die untersuchte Version des Automatiken-GUI genannt.

Im Abschnitt 4 sind die Ergebnisse der Analyse als eine Liste von festgestellten Problemen zusammengefasst.

3.1. Vorgehen

Für die Analyse wurden einerseits die in Abschnitt 2.4.4 genannten Ratschläge, Heuristiken und Prinzipien aus der Literatur durchgesehen und auf Anwendbarkeit überprüft. Andererseits hat die Expertin Kain viele Anmerkungen beigesteuert, sodass insgesamt über 20 Probleme identifiziert werden konnten (Abschnitt 4).

3.2. Die Benutzer

Software wird für bestimmte Benutzer entwickelt. Sind bei der Software-Entwicklung die Benutzer allerdings unbekannt oder unzureichend charakterisiert, so kann die Software nicht auf die Benutzer abgestimmt werden. Das Leitziel der „Zufriedenheit“ (nach Gebrauchstauglichkeit in EN ISO 9241-11) ist dann nur schwer zu erreichen.

Die Anforderungsanalyse von Fuhrmann hat gezeigt, was die Benutzer mit der Software erledigen müssen[21], aber nicht wer sie sind. Aus diesem Grund werden die Benutzer des Automatiken-GUIs soweit wie möglich charakterisiert (nach Kain [11]):

> Die Benutzer des AAF Graph Tools sind deutschsprachige Wissenschaftler, vornehmlich Psychologen, Soziologen und Verhaltensforscher. Es ist davon auszugehen, dass diese Computerkenntnisse eines durchschnittlichen Akademikers aufweisen. Keinesfalls kann davon ausgegangen werden, dass Programmierkenntnisse vorhanden sind oder dass sich die Benutzer mit Squeak auskennen.
>
> Die Benutzer erhalten eine kurze Beschreibung des ATEO-Systems. Einzelne Komponenten und deren Funktions- und Bedienweise werden in dieser erklärt, einschließlich des AAF Graph Tools, um erfolgreiche Experimente durchführen zu können.

[21] vgl. [6], Abschnitt 4.2, S. 34ff.

Die Benutzer erhalten also eine gewisse Dokumentation bzw. Anleitung, werden aber an keinem Workshop oder ähnlichem teilnehmen. Daraus ergibt sich, dass die Dokumentation keine Fragen offen lassen sollte und dass das AAF Graph Tool sich so intuitiv wie nur möglich verhalten sollte.

3.3. Untersuchte Version des Automatiken-GUI

Um die Nachvollziehbarkeit der erkannten Probleme zu ermöglichen, wird an dieser Stelle die untersuchte Version des Automatiken-GUI genannt.

Der Quelltext des ATEO-Projekts wird auf der Assembla-Plattform[22] verwaltet. Diese stellt als Versionsverwaltung git[23] zur Verfügung.

Die letzte Version, die Änderungen an dem AAF Graph Tool enthält und hier als die zu untersuchende Version referenziert wird, ist:

```
commit b60de7ac8cdd22e78c06962e7d7c54a688e8f5a6
Author: Esther Fuhrmann <efuhrman@informatik.hu-berlin.de>
Date: Thu Dec 2 15:33:06 2010 +0100
changed positioning of root and sink element in GUI: now
they are calculated from display size instead of being constant
```

Um eine lokale Kopie dieser Version zu erhalten, muss der aktuelle Quelltext heruntergeladen werden (Zeile 1) und die genannte Version eingestellt werden (Zeile 2):

```
$ git clone git://git.assembla.com/ATEO.git
$ git checkout b60de7ac8cdd22e78c06962e7d7c54a688e8f5a6
```

Nachdem die Anweisungen in der Datei INSTALL befolgt wurden, um die Quelltexte von SAM, dem AAF-Framework und dem AAF Graph Tool zu importieren, kann das Automatiken-GUI schließlich über den folgenden Befehl in einem Workspace gestartet werden:

```
AAFGTMainWindow new openInWorld.
```

Da zu dieser Version keine abgespeicherten Automatiken existierten, wurden einige sehr einfache erstellt, da sonst diverse Probleme nicht gezeigt werden könnten.

[22] vgl. http://www.assembla.com/code/ATEO/git/nodes (abgerufen am 16. September, 12:50 Uhr)
[23] vgl. http://git-scm.com/ (abgerufen am 13. September, 15:43 Uhr)

4. Erkannte Probleme

Die nachfolgenden Abschnitte beschreiben die identifizierten Probleme. Die sind nach Ähnlichkeit und betreffendem Anwendungsbereich gruppiert. Die Gruppen sind:

- Bezeichner (Abschnitt 4.1, 6 Probleme)
- GUI-Elemente (Abschnitt 4.2, 3 Probleme)
- Bereiche für Elemente (Abschnitt 4.3, 5 Probleme)
- Arbeitsbereich (Abschnitt 4.4, 5 Probleme)
- Eigenschaftenbereich (Abschnitt 4.5, 2 Probleme)
- Verschiedenes (Abschnitt 4.6, 8 Probleme)

Die Gruppe „Verschiedenes“ enthält alle Probleme, die keiner anderen Gruppe zugeordnet werden konnten.

Mit „Anwendungsbereichen“ sind die vier benannten Bereiche des Automatiken-GUI gemeint: Elemente, Komplexe Elemente, Eigenschaften und Arbeitsbereich (Abbildung 6). Im Folgenden sei der oberste Bereich, der die Buttons „Neu“, „Öffnen“, „Speichern“, „Verlassen“ enthält, „Menübereich“ genannt.

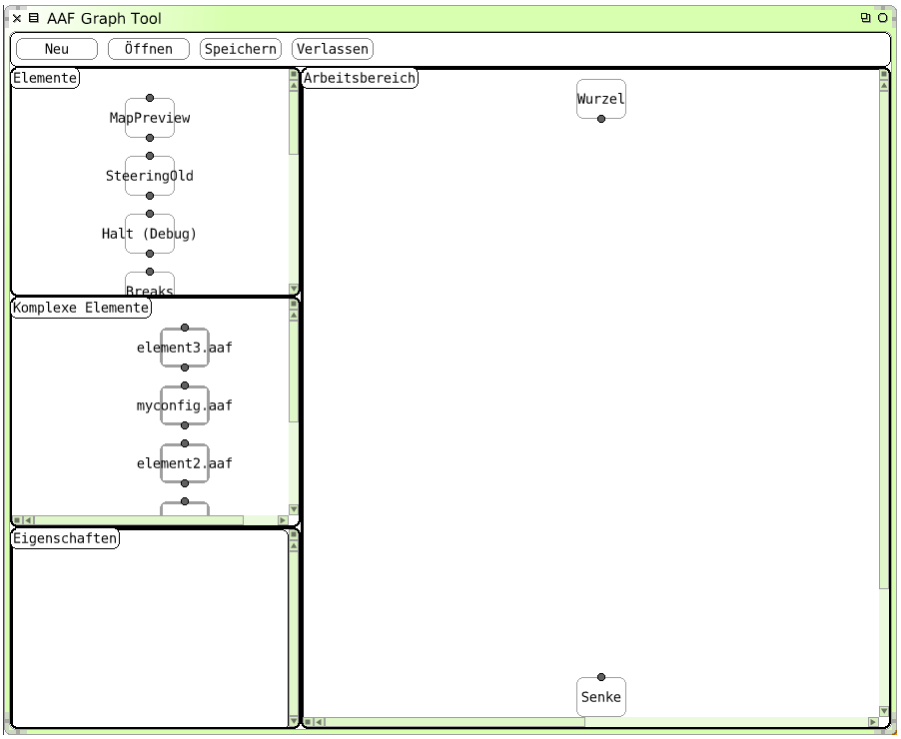

Abbildung 6: Gestartetes Automatiken-GUI „AAF Graph Tool“ von Fuhrmann [6].

Es sei noch angemerkt, dass Fuhrmann [6] die Funktionen in dem Automatiken-GUI „Elemente“ nennt, daher wird an dieser Stelle auch noch diese Wortwahl verwendet. Die

Korrektur zu „Funktionen“ ist eine Problembehebung, die in Abschnitt 5.1 durchgeführt wird.

4.1. Bezeichner (6 Probleme)

Problem 1.1 *Der Name „AAF Graph Tool“ sagt wenig für die Benutzer aus.*

Problem 1.2 *„Inneren Graphen anzeigen“ und „Inneren Graphen schließen“ sind ungünstige Bezeichner.*
Für Informatiker und Mathematiker ist der Begriff Graph geläufig. Für die Benutzer ist er unpassend. Das Konzept einer „inneren“ und implizierten „äußeren“ Komponente führt zu einem unnötig komplizierten konzeptionellen Modell.

Problem 1.3 *„Verlassen“ ist ein unkonventioneller Bezeichner zum Beenden von Software.*
Konventionell findet sich in fast jeder Anwendung der Menüeintrag Datei - Beenden. Der Bezeichner „Verlassen“ im Button oben ist nicht falsch, entspricht aber nicht den üblichen Konventionen. „Verlassen“ könnte den Eindruck erwecken, dass nur eine Ansicht verlassen wird.

Problem 1.4 *Die Mischung aus deutschen und englischen Begriffen ist unangemessen.*
Die Benutzer sind in erster Linie deutschsprachige Benutzer und dementsprechend sollten alle sichtbaren Texte in Deutsch sein. Das gilt insbesondere für die Namen der Elemente.

Problem 1.5 *„Elemente“ und „Arbeitsbereich“ sind zu allgemeine Begriffe.*
Hierfür lassen sich treffendere Begriffe finden.

Problem 1.6 *„Wurzel“ und „Senke“ sind verwirrende Begriffe.*
Für Benutzer, denen die Graphentheorie unbekannt ist, wovon auszugehen ist, sind die Begriffe „Wurzel“ und „Senke“ verwirrend: Sollten „Wurzeln“ nicht eher im unteren Bereich zu finden sein?

4.2. GUI-Elemente (3 Probleme)

Problem 2.1 *Die Bezeichner der Bereiche sind von den Buttons kaum unterscheidbar.*
Beispielsweise ist das Bezeichnerwidget für „Arbeitsbereich“ von dem darüberliegenden Button „Verlassen“ schwer zu unterscheiden. Der einzige Unterschied besteht in der Rahmenfarbe (Grau beim Button, Schwarz beim Bezeichner). Das ist irreführend für den Benutzer, da dieser Ausprobieren könnte die Bezeichnerwidgets anzuklicken. Zwar sind die Anwendungsbereiche voneinander klar gegliedert, aber innerhalb eines Bereiches wird durch den Bezeichner ein „Fragment“ erzeugt, welches der Benutzer wahrnehmen und verarbeiten muss.

Problem 2.2 *Die Bedienelemente sind weder in Form noch Größe einheitlich.*
Ein Beispiel hierfür ist der Button „Vorschau“, welcher nicht mit der verwendeten

Spinbox zusammenpasst (Abbildung 9, links unten). Die Spinbox ist überdimensioniert.

Provlem 2.3 *Die GUI wird von zu viel weißer Farbe dominiert.*

4.3. Bereiche für Elemente (5 Probleme)

Problem 3.1 *Der Bereich für komplexe Elemente erweckt den Eindruck, dass der Benutzer mit Dateien arbeitet.*
Während im Bereich für Elemente die Namen als Bezeichner verwendet werden, werden im Bereich für komplexe Elemente die zugehörigen Dateinamen verwendet (Inkonsistenz). Dateinamen sind in diesem Fall ungünstige Bezeichner für Objekte, mit denen der Benutzer direkt interagieren kann. Sie enthalten überflüssige Informationen (Dateiendung „.aaf“) und der Benutzer ist bei der Vergabe eines Dateinamens eingeschränkt. So können beispielsweise einige Sonderzeichen nicht verwendet werden[24].

Problem 3.2 *Den Bereichen für Elemente steht zu wenig Fläche zur Verfügung.*
Bei hinreichend vielen Elementen werden im Bereich für Elemente und komplexe Elemente vertikale Scrollbalken notwendig. Haben die Elemente zusätzlich einen langen Namen, so erscheinen auch horizontale Scrollbalken[25].

Problem 3.3 *Die Organisation der Elemente in ihren Bereichen ist unklar.*
Die Elemente werden in ihren Bereichen untereinander in einer nicht erkennbaren Reihenfolge aufgelistet. Die Reihenfolge entspricht weder der alphabetischen Anordnung der Bezeichner, noch sind die Elemente gruppiert. Das Suchen eines bestimmten Elements kostet den Benutzer wertvolle Zeit.

Problem 3.4 *Die Elemente sind undokumentiert.*
In den Bereichen für Elemente und kombinierte Elemente stehen keine weiteren Informationen über die Elemente zur Verfügung. Ist dem Benutzer ein Element unbekannt, kann der Bezeichner nicht ausreichend sein, um zu wissen, welche Funktionalität das Element zur Verfügung stellt. Hier würde der Benutzer das Element in den Arbeitsbereich ziehen, es anklicken und dann dessen Eigenschaften betrachten müssen, um eine genauere Vorstellung von der Funktionalität zu bekommen.

Problem 3.5 *Elemente aus dem Arbeitsbereich können nicht zurück in den Bereich für (komplexe) Elemente gezogen werden.*

[24] Betriebssystemabhängig, der Benutzer sollte jedoch von dieser technischen Einschränkung nichts merken. Siehe auch Designprinzip „Die Verwaltung von Festplatten und Dateien ist kein User-Ziel“ in [3], Abschnitt 17.2, S. 331.

[25] „Scrollen Sie Text niemals horizontal“ heißt es in [3], Abschnitt 21.3.5, S. 419f. In den Bereichen geht es zwar nicht um Fließtext, dennoch passiert es, dass durch das horizontale Scrollen der Anfang von einigen Bezeichnern nicht sichtbar ist. Ebenfalls durch das horizontale Scrollen wird gegen das „Gesetz der Fortsetzung“ (Gestaltgesetz) verstoßen: die vertikale Achse an der die Elemente und komplexe Elemente in ihren Bereichen angeordnet werden, wird zu zwei Achsen aufgebrochen (vgl. Abbildung 9). Zwar handelt es sich um unterschiedliche Bereiche, dennoch wird das als unharmonisch wahrgenommen.

Via Drag & Drop können Elemente aus den zwei linken oberen Bereichen in den Arbeitsbereich gezogen werden. Einige Benutzer versuchen Elemente in ihre jeweiligen linken Bereiche zurückzuziehen, in der Hoffnung, diese Elemente aus dem Arbeitsbereich zu löschen.

4.4. Arbeitsbereich (5 Probleme)

Problem 4.1 *Der Arbeitsbereich skaliert bei Größenveränderung des Fensters nicht mit.*
Die Anwendung startet stets im maximierten Fenster. Wird das Fenster verkleinert, erscheinen automatisch Scrollbalken, da die Größe des Arbeitsbereiches und die Positionen der Elemente nicht angepasst werden. Durch Scrollen kann es passieren, dass die Wurzel ode Senke nicht mehr sichtbar ist. Würde der Benutzer in Abbildung 9 nach unten scrollen, wäre die Wurzel nicht mehr sichtbar.

Problem 4.2 *Die Größe des Arbeitsbereichs bei konstanter Fenstergröße ist nicht veränderbar.*
Für kleine Bildschirmauflösungen wäre es wünschenswert den Bereich für Elemente kleiner zu ziehen, damit dem Arbeitsbereich mehr Fläche zugestanden wird.

Problem 4.3 *Wurzel und Senke im Arbeitsbereich sind nur scheinbar interaktive Elemente.*
Ein Element im Arbeitsbereich (kein kombiniertes Element), welches konfiguriert, verschoben und gelöscht werden kann, ist optisch nicht von der unveränderlichen Wurzel und Senke unterscheidbar. Der Benutzer wird erst durch einen Versuch entdecken, dass er mit Wurzel und Senke nicht interaktiv arbeiten kann.[26]

Problem 4.4 *Im Arbeitsbereich kann die Auswahl eines Elements nicht mit Links-Klick aufgehoben werden.*
Wenn ein Element mit Links-Klick ausgewählt werden kann, sollte es auch mit Links-Klick wieder aufgehoben werden können.

Problem 4.5 *Die Weitergabe der Daten von Wurzel zu Senke ist unzureichend visualisiert.*

4.5. Eigenschaftenbereich (2 Probleme)

Problem 5.1 *Dem Eigenschaftenbereich steht zu wenig Fläche zur Verfügung.*
Die dem Eigenschaftenbereich zugewiesene Fläche ist zu klein (vgl. Abbildung 7 oben und 9). Auch wird der Platz nicht optimal genutzt. Es gibt zu viel ungenutzte Fläche.

Problem 5.2 *Der Eigenschaftenbereich ist anfangs leer und nicht hinreichend erklärt.*
Beim Starten des AAF Graph Tools ist dieser Bereich leer (Abbildung 6, links

[26] vgl. auch Designprinzip „Unterscheiden Sie Elemente visuell, die sich unterschiedlich verhalten.“ in [3], Abschnitt 14.3.3, S. 287

unten), was bei dem Benutzer einen Eindruck der Unvollständigkeit erzeugen kann. Erst wenn die Maus in diesen Bereich bewegt wird, erscheint ein Tooltip mit dem Inhalt „Anpassen von Eigenschaften“. Hier ist nicht klar, um welche Eigenschaften es sich handelt.[27]

4.6. Verschiedenes (8 Probleme)

Problem 6.1 *Die Bezeichner der Elemente überlagern das Element-Objekt.*
Die meisten Bezeichner sind breiter als ihr „Element-Objekt“ (abgerundetes Rechteck). Dadurch entsteht ein unharmonischer Eindruck.

Problem 6.2 *Die Navigation in den inneren bzw. äußeren Graphen erfolgt an unterschiedlichen Stellen.*
Um den inneren Graphen eines komplexen Elements anzeigen zu lassen, klickt der Benutzer im Eigenschaftenbereich auf „Inneren Graphen anzeigen“ (Abbildung 7 oben). Wird der innere Graph angezeigt und möchte der Benutzer zurück zum äußeren Graphen navigieren, klickt er dazu im Arbeitsbereich auf den links oben hinzugefügten Button „Inneren Graphen schließen“ (Abbildung 7 unten).

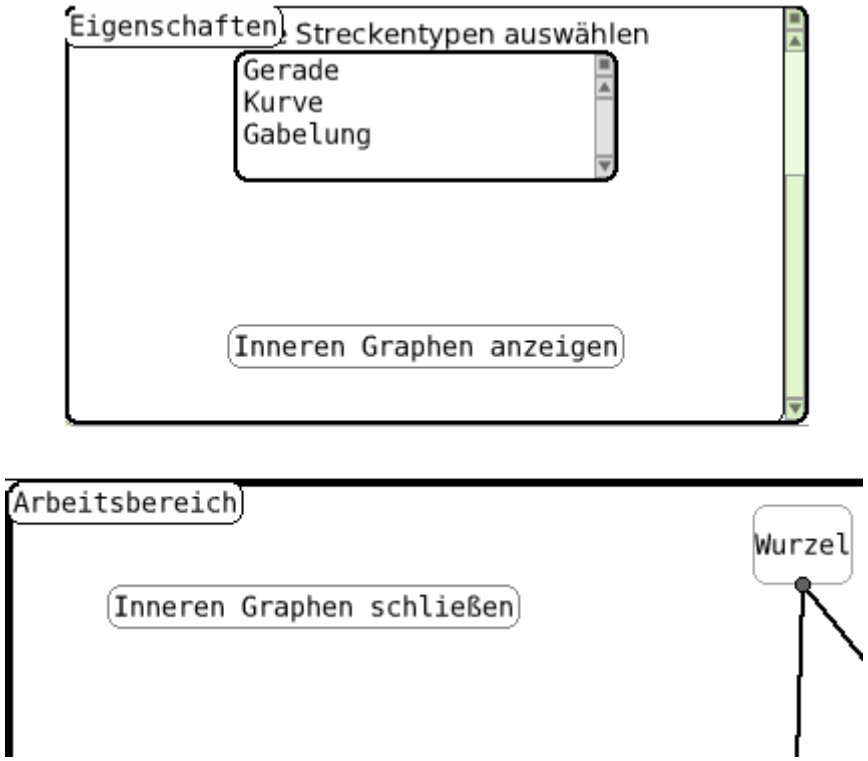

Abbildung 7: Navigation in den inneren Graphen aus dem Eigenschaftenbereich (oben) und äußeren Graphen aus dem Arbeitsbereich (unten)

[27] vgl. Grundsatz der Selbstbeschreibungsfähigkeit in Abschnitt 2.4.3

Problem 6.3 *Falls ein komplexes Element geöffnet ist, dann ist unklar welche Datei zugehörig ist.*
Öffnet der Benutzer über den „Öffnen“-Button ein komplexes Element, dann wird an keiner Stelle angezeigt, welche Datei er gerade geöffnet hat.

Problem 6.4 *Benutzer wird nicht gewarnt bei der Speicherung eines ungültigen Graphen.*
Der Umgang mit ungültigen[28] Graphen ist problematisch. Zum einen wird der Benutzer beim Speichern eines ungültigen Graphen nicht gewarnt, zum anderen wird der ungültige Graph nicht als komplexes Element links im linken mittleren Bereich angezeigt. Das erweckt den Eindruck, dass die Arbeit umsonst gemacht wurde. Tatsächlich existiert die gespeicherte Datei und kann auch über „Öffnen“ weiterbearbeitet werden.

Problem 6.5 *Anwendungsstart und Speichervorgang dauern zu lange.*
Sind viele komplexe Elemente vorhanden, so dauert der Start vom AAF Graph Tool sowie der Speichervorgang dessen länger als erwartet.[29]

Problem 6.6 *Der Menübereich verändert seine Höhe abhängig vom Anwendungsfenster.*
Vergrößert der Benutzer in der Vertikalen das Anwendungsfenster, so wächst auch der Menübereich in der Vertikalen mit. Dieses Verhalten hat keinen funktionalen Effekt und verwirrt den Benutzer (Abbildung 8).

Abbildung 8: Der Menübereich bei relativ kleinem (oben) und großem (unten) Anwendungsfenster.

Problem 6.7 *Der „Verlassen“-Button passt logisch nicht zu „Neu“, „Öffnen“, „Speichern“*
„Neu“, „Öffnen“ und „Speichern“ sind Aktionen, die der Benutzer öfter verwendet. „*Verlassen*“ ist dagegen eine einmalige Aktion. Zudem liegt „Speichern“ direkt neben „*Verlassen*“ - verklickt sich der Benutzer, wird die Anwendung entweder sofort beendet (sehr überraschend) oder fragt nach ob ungesicherte Änderungen gespeichert werden sollen (verwirrend). Weiterhin ist der „*Verlassen*“-Button nicht dort positioniert, wo ihn der Benutzer erwarten würde.

[28] Ein Graph sei in diesem Kontext ungültig genannt, wenn er mindestens einen Knoten enthält, der keinen Vor- oder Nachfolger hat.

[29] vgl. [10], Prinzip 8 „Design for responsiveness“, S. 45 - Die wahrnehmbare Reaktionsfähigkeit bzw. Geschwindigkeit einer Anwendung ist der wichtigste Faktor bezüglich der Benutzerzufriedenheit. Benutzer hassen es, wenn Anwendungen sie warten lassen.

Problem 6.8 *Es existiert keine Möglichkeit eine Kopie einer abgespeicherten Automatik zu erstellen, um mit dieser weiterzuarbeiten.*

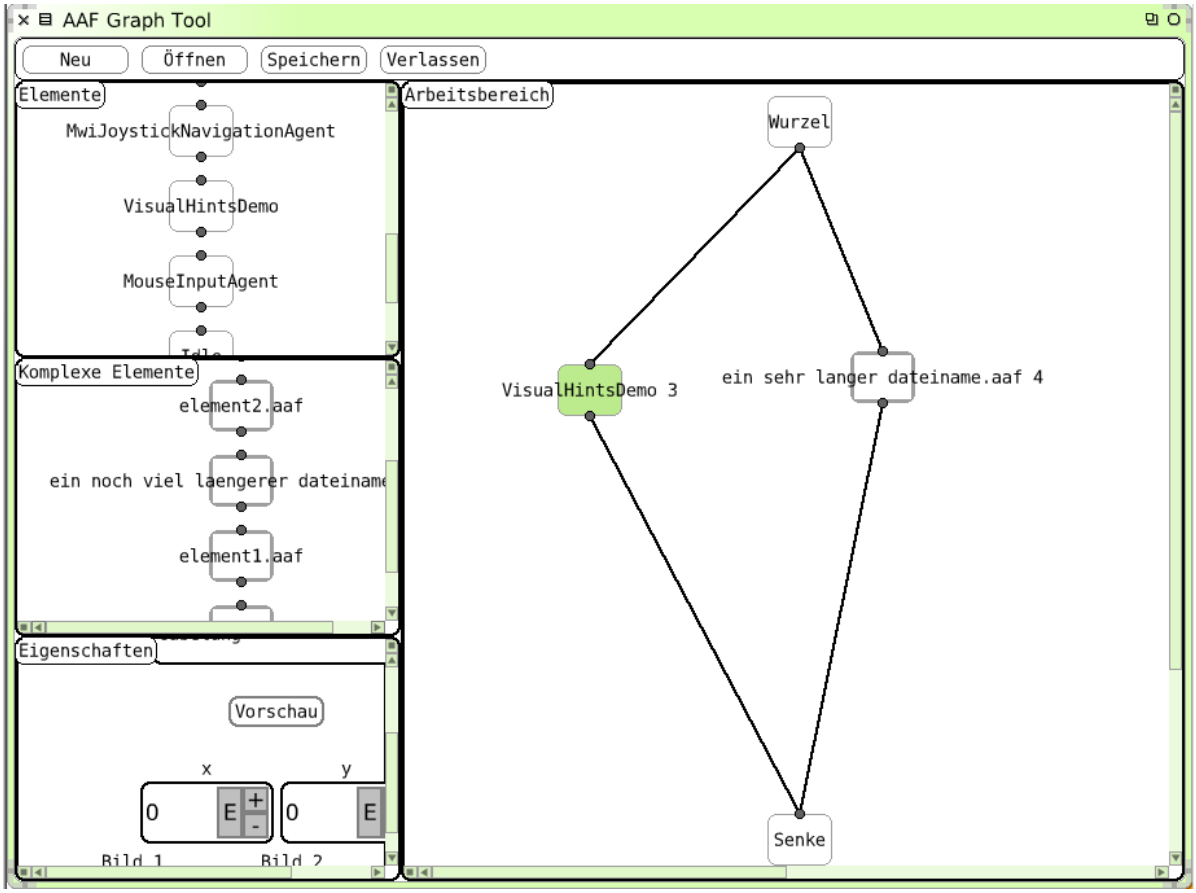

Abbildung 9: Eine einfache kombinierte Funktion

5. Behebung der Probleme

In diesem Abschnitt wird beschrieben, wie die identifizierten Probleme behoben werden. Die Struktur der nachfolgenden Unterabschnitte richtet sich nach der Struktur von Abschnitt 4.

5.1. Bezeichner

Ein wichtiger Aspekt der Benutzerschnittstelle ist die Sprache. Wie in [4] erläutert, steht bei der Umsetzung von ergonomischen Kriterien der Benutzer im Vordergrund. Dazu gehört auch, dass bei Bezeichnern und Texten (Erläuterungen, Rückmeldungen) in der Anwendung eine für den Benutzer angemessene Sprache gewählt wird. Insbesondere sollte darauf geachtet werden, dass keine Begriffe der Entwickler bzw. Implementierung bis zum Benutzer „durchscheinen“. Dies entspricht dem Grundsatz der Aufgabenangemessenheit sowie der Erwartungskonformität. Auch Nielsen unterstreicht in [14] diesen Aspekt:

> Match between system and the real world - The system should speak the users' language...

Bezeichner kommen üblicherweise mehrmals in der Anwendung vor. Hier sollte die Konsistenz gewahrt werden: Für das selbe Konzept sollte der selbe Bezeichner verwenden werden. Es wird empfohlen einen Wortschatz zu definieren, nach dem sich die Entwickler richten[30]. Dieser erweist sich auch bei der Dokumentation der Anwendung als sehr nützlich.

Nachdem im Folgenden die einzelnen Behebungen der Probleme vorgestellt werden, findet sich in Abschnitt 5.1.1 der erstellte Wortschatz – Begriffe in ihrem Kontext.

Dieser Abschnitt sollte sehr sorgfältig gelesen werden, da in nachfolgenden Problembehebungen die neuen Begriffe bereits verwendet werden.

Problem 1.1 *Der Name „AAF Graph Tool“ sagt wenig für die Benutzer aus.*

„AAF Graph Tool“ ist der bisherige Name der Anwendung, welcher in der Titelleiste des Anwendungsfensters angezeigt wird. So treffend dieser Anwendungsname aus Sicht der Entwickler ist, so verwirrend ist er für den Benutzer. Die Anwendung wird in *Konfigurationstool der Automatik-Funktionen* umbenannt, damit auch neue Benutzer grob wissen, was mit der Anwendung gemacht werden kann.

Problem 1.2 *„Inneren Graphen anzeigen“ und „Inneren Graphen schließen“ sind ungünstige Bezeichner.*

Das einfachste konzeptionelle Modell dieses Sachverhalts entspricht einer Zoom-Funktion. Weitere Einstellungsmöglichkeiten von Funktionen können durch das Erhöhen der Detailstufe eingesehen und verändert werden. Die Begriffe „Reinzoomen“ und „Rauszoo-

[30] vgl. [10], Kapitel 1, S. 23 und 29

men“ erinnern aber zu stark an die Ansicht eines Bildes in einem Grafik- oder Navigationsprogramm. Daher wird *Detailstufe erhöhen* und *Detailstufe verringern* verwendet.

Problem 1.3 *„Verlassen“ ist ein unkonventioneller Bezeichner zum Beenden von Software.*

„Verlassen“ wird umbenannt in *Beenden*.

Problem 1.4 *Die Mischung aus deutschen und englischen Begriffen ist unangemessen.*

Die Elemente erhalten alle deutsche Bezeichner. Dieses Problem konnte allerdings nicht vollständig gelöst werden, siehe Abschnitt 6.2.

Problem 1.5 *„Elemente“, „Arbeitsbereich“ und „Eigenschaften“ sind zu allgemeine Begriffe.*

Die Elemente der Anwendung, das worum es geht und womit der Benutzer direkt interagiert, sind Funktionen. Die bisherigen Elemente heißen nun *elementare Funktionen*, die komplexen Elemente heißen nun *kombinierte Funktionen*. Wird sich in der Anwendung auf die Funktionen bezogen, kann durch „elementar“ bzw. “kombiniert“ nun unmissverständlich Bezug zu einer bestimmten Funktionsart genommen werden[31]. *Funktionen* meint sowohl die elementaren als auch die kombinierten Funktionen.

Der „Arbeitsbereich“ wird umbenannt zu *Ablaufplan*. Dieser Begriff ist wesentlich konkreter und den Benutzern vertraut.

Der Begriff „Eigenschaften“ ist recht verständlich und wird in vielen anderen Anwendungen ebenfalls benutzt. Er klingt jedoch recht passiv. *Konfiguration* betont, dass *Parameter* von Funktionen aktiv verändert werden können. Diese Begriffe sind Wissenschaftlern vertraut.

Problem 1.6 *„Wurzel“ und „Senke“ sind verwirrende Begriffe.*

Die Begriffe „Wurzel“ und „Senke“ werden in *Anfang* und *Ende* umbenannt. Zwar sind diese auf den ersten Blick auch etwas allgemein (Problem 1.6), aber sie passen gut zum *Ablaufplan*, der ja einen Anfang und ein Ende haben muss.

5.1.1. Begriffe in ihrem Kontext

Aufbauend auf den Umbenennungen seien im Folgenden die wichtigsten Begriffe der Anwendung in ihren Kontexten zusammengefasst.

Die Anwendung erlaubt *elementare und kombinierte Funktionen zu konfigurieren* und zu *neuen* kombinierten Funktionen *zusammenzustellen*. Diese können in einer Datei *gespeichert* und aus dieser *geöffnet* werden.

Die Anwendung ist in drei wichtige Bereiche eingeteilt:

[31] Vorher wurde nur zwischen „Elementen“ und „komplexen Elementen“ unterschieden. Mit dem Begriff„Elemente“ war nicht ersichtlich, ob alle Elemente gemeint waren, oder nur die, die keine komplexen Elemente sind.

- Links: *Anzeige von elementaren und kombinierten Funktionen*
- Mitte: *Ablaufplan*
- Rechts: *Konfigurationsbereich für eine (ausgewählte) Funktion*

Im linken Bereich können entweder die elementaren oder die kombinierten Funktionen *angezeigt* werden. Sowohl die Anzeige von elementaren als auch kombinierten Funktionen lässt sich nach *Kategorien filtern*[32]. Die kombinierten Funktionen lassen sich aus dieser Anzeige direkt *öffnen*[33] um *bearbeitet* zu werden. *Ungültige* kombinierte Funktionen (d.h. nicht alle Funktionen der kombinierten Funktion sind im Ablaufplan verbunden) werden bei der Anzeige rot *hervorgehoben*. Jede Funktion in der Anzeige lässt sich via Drag & Drop in den Ablaufplan *ziehen*.

Jeder Ablaufplan hat einen *Anfang* und ein *Ende*. Funktionen im Ablaufplan können *gelöscht* und *umbenannt* werden. Bereits bestehende Verbindungen können *gelöscht* werden. Von kombinierten Funktionen im Ablaufplan lässt sich die *Detailstufe erhöhen*, um weitere Parameter anzuzeigen. Die *Detailstufe* kann wieder *verringert* werden.

Die Anwendung kann jederzeit *beendet* werden.

5.2. GUI-Elemente

Teile der Anwendung und insbesondere die Konfiguration von Funktionen sollten einheitliche Bedienelemente benutzen, da diese maßgeblich zur visuellen Konsistenz einer Anwendung beitragen.

Problem 2.2 *Die Bedienelemente sind weder in Form noch Größe einheitlich.*

Dazu werden neue, konsistente Bedienelement entwickelt. Sie können in der *AAFGT Widget Gallery* betrachtet werden (Abbildung 10). Dabei handelt es sich um eine Demonstration der verfügbaren Bedienelemente, auf die die Entwickler zugreifen können. Abschnitt D.2 liefert mehr Informationen zu den Bedienelementen und dokumentiert die zugehörigen Klassen.

Die bestehenden Funktionen mit ihren Konfigurationsschnittstellen wurden auf die neuen Bedienelemente portiert.

Die Implementierung von „Einheitlichen grafischen Bedienelementen“ ist eine Idee von Fuhrmann[34] zur Weiterentwicklung des Automatiken-GUIs und ist mit der Einführung der neuen Widgets umgesetzt.

Problem 2.1 *Die Bezeichner der Bereiche sind von den Buttons kaum unterscheidbar.*

Die Bereiche sind nun in den AAFGTPane-Containern der neuen Widgets organisiert und ihre Überschriften unterscheiden sich nun deutlich von den Buttons.

[32] Entspricht dem Grundsatz der Steuerbarkeit.
[33] Anstatt über „Öffnen“ im Menü zu gehen und die entsprechende Datei auszuwählen.
[34] vgl. [6], Abschnitt 5.2.9, S. 84

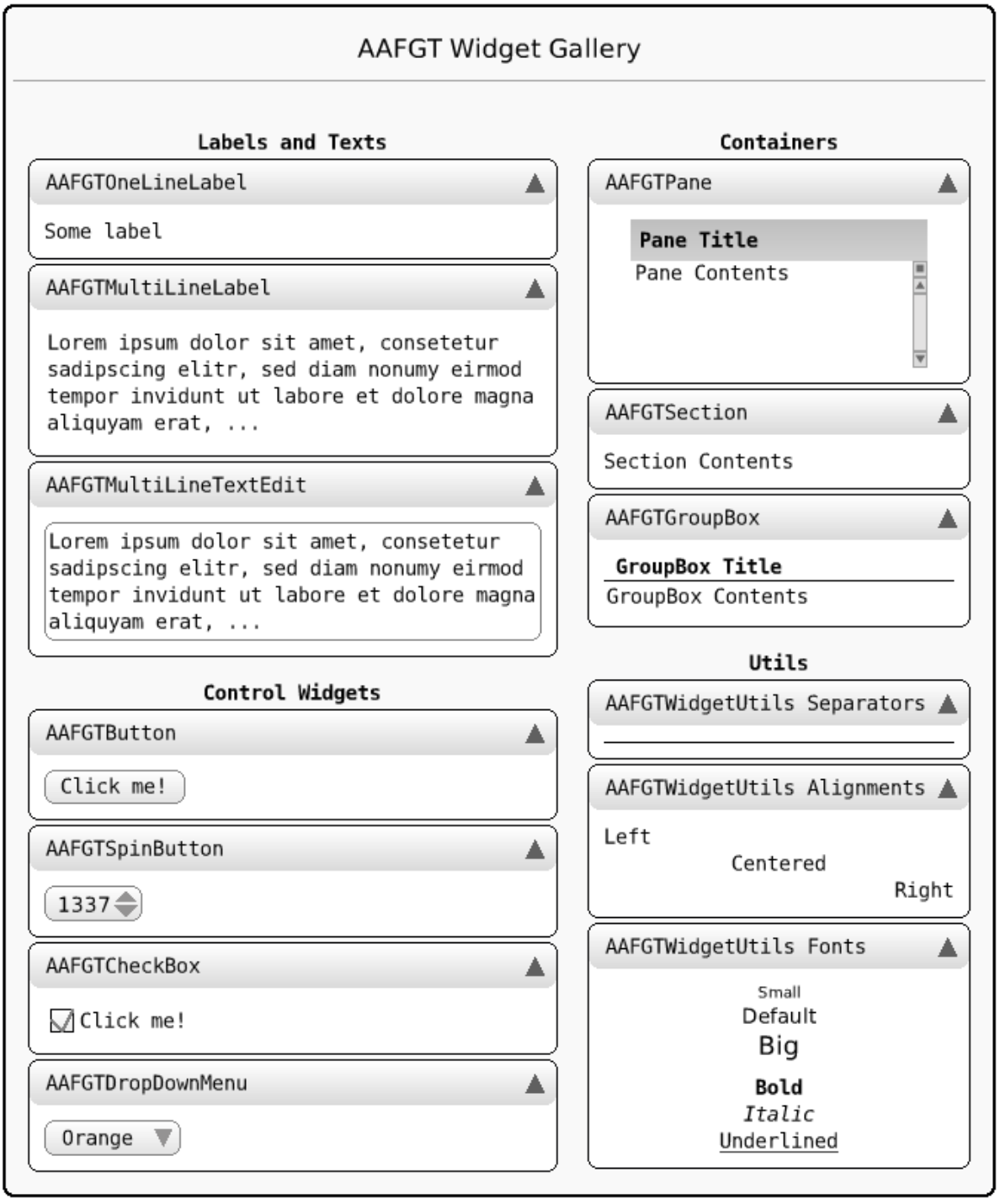

Abbildung 10: Die *AAFGT Widget Gallery* demonstriert die Benutzung der Bedienelemente.

Provlem 2.3 *Die GUI wird von zu viel weißer Farbe dominiert.*

Allgemeinen Empfehlungen nach sollten Farben sparsam verwendet und gut aufeinander abgestimmt werden. Das Automatiken-GUI ist nun vorwiegend in Grautönen realisiert. Grau ist neutral, eignet sich damit für Hintergründe besonders und ist dem Benutzer aus anderen Anwendungen vertraut. Der Hintergrund des Menüs und des Ablaufplans sind allerdings in Weiß gehalten, um einen nicht zu tristen Gesamteindruck zu bewirken. Eine ausgewählte Funktion hat eine weiße Schrift auf blauen Hintergrund[35]. Ansonsten ist die Farbe der Schrift immer schwarz auf hellem (Grau) oder sehr hellem Hintergrund (Weiß) um die gute Lesbarkeit zu gewährleisten. Ungültige, kombinierte Funktionen sind durch einen roten, dicken Rahmen gekennzeichnet.

5.3. Bereiche für Elemente

Dieser Abschnitt widmet sich der Überarbeitung der Bereiche für die Funktionen (ehemals Elemente).

Problem 3.1 *Der Bereich für komplexe Elemente erweckt den Eindruck, dass der Benutzer mit Dateien arbeitet.*

Problem 3.1 wird behoben, indem der Benutzer die Möglichkeit erhält, einen frei wählbaren Namen für eine kombinierte Funktion zu vergeben, der keinen Einschränkungen unterliegt. Beim Erstellen einer neuen kombinierten Funktion findet der Benutzer im Konfigurationsbereich ein Eingabefeld um einen Namen zu vergeben. Durch Abspeichern der Funktion wird diese sofort im Funktionsbereich für kombinierte Funktionen angezeigt.

Problem 3.2 *Den Bereichen für Elemente steht zu wenig Fläche zur Verfügung.*

Durch die Fülle der Funktionen die hinzukommen werden, werden die Funktionsbereiche unübersichtlich und scrollbar bleiben müssen. Aus diesem Grund wird folgender Ansatz verfolgt: Der Funktionsbereich zeigt entweder nur elementare oder nur kombinierte Funktionen an. Mit einem Dropdown-Menu kann der Benutzer die Anzeige auf die entsprechenden Funktionen anpassen. Das erhöht für den Benutzer zwar den Navigationsaufwand, aber auch die Übersicht. Alle Anwendungsbereiche, bis auf das Menü, werden in einem AAFGTPane-Container angezeigt. Für den Funktionsbereich wurde der Kopf des AAFGTPane-Containers so angepasst, dass das Dropdown-Menü hier platziert werden kann. Dies spart zusätzlich vertikalen Platz (siehe Abbildung 11, oberes Dropdown-Menü).

Problem 3.3 *Die Organisation der Elemente in ihren Bereichen ist unklar.*

Problem 3.3 wird auf zwei Ebenen begegnet. Zum einen werden die Funktionen alphabetisch sortiert angezeigt[36]. Zum anderen lässt sich eine Funktion einer oder mehreren

[35] Der blaue Hintergrund ist Benutzern vertraut - ausgewählte Elemente werden in üblichen Dateimanagern ebenfalls blau dargestellt.

[36] Anfangs spielte bei der Sortierung die Groß- und Kleinschreibung eine Rolle. 'Zander' wurde vor 'aal' eingeordnet, weil in Zeichentabellen die Groß- vor den Kleinbuchstaben angeordnet sind. Das

Kategorien zuordnen und die Anzeige von elementaren oder kombinierten Funktionen lässt sich nach einer Kategorie filtern. Die Kategorien können ebenfalls beim Anlegen einer neuen kombinierten Funktion vergeben werden. Das passiert, indem der Benutzer Schlagwörter angibt. Jedes neue Schlagwort ist eine neue Kategorie. Um den Navigationsaufwand für den Benutzer hier nicht unnötig zu erhöhen, wird die künstliche Kategorie '(Alle Funktionen)' als Voreinstellung eingeführt. Die Standardkategorie bei Erstellung einer kombinierten Funktion ist '(Unkategorisiert)'. Durch die runden Klammern ist sicher gestellt, dass diese künstlichen Kategorien an erster Stelle im Dropdown-Menü erscheinen, und nicht zwischen Kategorien die vom Benutzer angelegt worden sind. Die Abbildung 11 zeigt den oberen Funktionsbereich mit Filtermöglichkeiten.

Die Kategorien und Filtermöglichkeiten implementieren „Individuelle Element-Paletten", eine Idee von Fuhrmann[37] zur Weiterentwicklung des Automatiken-GUIs.

Abbildung 11: Die Anzeige der verfügbaren Funktionen kann über Dropdown-Menüs gefiltert werden.

Problem 3.4 *Die Elemente sind undokumentiert.*

Nicht für alle Funktionen, insbesondere kombinierte Funktionen, kann ein kurzer und dennoch aussagekräftiger Name gefunden werden. Daher ist für den Benutzer wichtig zu wissen, über welche Funktionalität (und in welchem Umfang) eine Funktion tatsächlich verfügt. Dazu sollte jede Funktion mit einer Beschreibung ausgestattet sein. Bei elementaren Funktionen verfassen die Entwickler die Beschreibung, weil sie die Funktion entworfen haben. Bei kombinierten Funktionen kann der Benutzer selbst eine Beschreibung vergeben. Dies erfolgt an gleicher Stelle, an der bei einer kombinierten Funktion Name und Schlagwörter vergeben werden. Durch diese Beschreibungen wird Problem 3.4 gelöst. Damit der Benutzer die Beschreibung ohne großen Interaktionsaufwand einsehen

ist für den Benutzer jedoch irreführend. Daher wird bei der Sortierung nicht zwischen Groß- und Kleinschreibung unterschieden.

[37] vgl. [6], Abschnitt 5.2.4, S. 81

kann (Effizienz), wird die Beschreibung im Funktionsbereich als Tooltip angezeigt, sobald der Benutzer mit der Maus über einer Funktion verweilt.

Zusätzlich können die Entwickler im Bereich für Parameter eine ausführlichere Hilfestellung bereitstellen. Ein Beispiel zeigt Abbildung 19 (Sektion Beschreibung).

Problem 3.5 *Elemente aus dem Arbeitsbereich können nicht zurück in den Bereich für (komplexe) Elemente gezogen werden.*

Problem 3.5 wird gelöst durch Aufhebung der Einschränkung. Durch den überarbeiteten Funktionsbereich stellt sich jedoch die Frage, wie sich die Anwendung verhalten soll, wenn eine einfache Funktion aus dem Ablaufplan in den Funktionsbereich gezogen wird, der gerade jedoch die kombinierten Funktionen anzeigt (analog auch andersherum). In diesem Fall wird der Funktionsbereich umgeschaltet auf die elementaren Funktionen. Dadurch, dass elementare und kombinierte Funktionen gut unterscheidbar[38] sind, registriert der Benutzer das Umschalten und kann sich darauf einstellen. Wenn der Benutzer eine elementare Funktion löscht, dann liegt die Vermutung nahe, dass er danach eine andere elementare Funktion wieder hinzufügen möchte.

Kombinierten Funktionen kann durch die oben beschriebenen Änderungen ein Name, eine Beschreibung und eine Liste von Schlagwörtern vergeben werden. Die Eingabemaske für diese Informationen wird in Abschnitt 5.5 beschrieben.

5.4. Arbeitsbereich

Dieser Abschnitt widmet sich der Überarbeitung des Ablaufplans (ehemals Arbeitsbereich).

Problem 4.1 *Der Arbeitsbereich skaliert bei Größenveränderung des Fensters nicht mit.*

Problem 4.1 wird angegangen, indem die Positionen der Funktionen im Ablaufplan nicht mehr absolut, sondern relativ abgespeichert werden. Für Benutzer, die die Anwendung im maximierten Squeak-Fenster bei gleicher Bildschirmauflösung benutzen, ändert sich dadurch nichts. Benutzer, die eine auf einem größerem Bildschirm abgespeicherte kombinierte Funktion aufmachen, müssen hingegen nicht mehr Scrollen. Nachteilig ist jedoch, dass sich Funktionen teilweise im Ablaufplan überlappen können, wenn der Ablaufplan nur klein genug gemacht wird. Das kann nur dadurch verhindert werden, indem der Benutzer nicht zu viele Funktionen in einen Ablaufplan packt und dessen Namen kurz hält.

Problem 4.2 *Die Größe des Arbeitsbereichs bei konstanter Fenstergröße ist nicht veränderbar.*

Zwischen den Bereichen für Funktionen, dem Ablaufplan und dem Konfigurationsbereich, sind sogenannte „Splitter“ eingeführt worden. Dabei handelt es sich um vertikale

[38] Kombinierte Funktionen haben fettgedruckte Namen, elementare Funktionen nicht.

Streifen, die, an der hervorgehobenen Stelle gezogen, die Aufteilung zwischen den Bereichen in der Horizontalen anpassen können (s. Abbildung 12). Dadurch kann dem Ablaufplan mehr Platz zugewiesen werden.

Problem 4.3 *Wurzel und Senke im Arbeitsbereich sind nur scheinbar interaktive Elemente.*

Der nichtfunktionale Anfang (ehemals Wurzel) und das nichtfunktionale Ende (ehemals Senke) werden mit einem dunkleren Hintergrund belegt, um sie von Funktionen im Ablaufplan unterscheiden zu können (s. Abbildung 12).

Problem 4.4 *Im Arbeitsbereich kann die Auswahl eines Elements nicht mit Links-Klick aufgehoben werden.*

Problem 4.4 ist eine Interaktions-Inkonsistenz, die den Erwartungen des Benutzers widerspricht. Die Behebung dieses Problems ist nicht nur für die Konsistenz wichtig, sondern auch, damit der Benutzer zu der Eingabemaske von der äußersten kombinierten Funktion gelangen kann[39].

Problem 4.5 *Die Weitergabe der Daten von Wurzel zu Senke ist unzureichend visualisiert.*

Aus den Verbindungslinien sind *Verbindungspfeile* geworden. Dies verdeutlicht die Weitergabe der Daten vom Anfang (ehemals Wurzel) zum Ende (ehemals Senke) besser.

Abbildung 12 zeigt den überarbeiteten Ablaufplan (ehemals Arbeitsbereich).

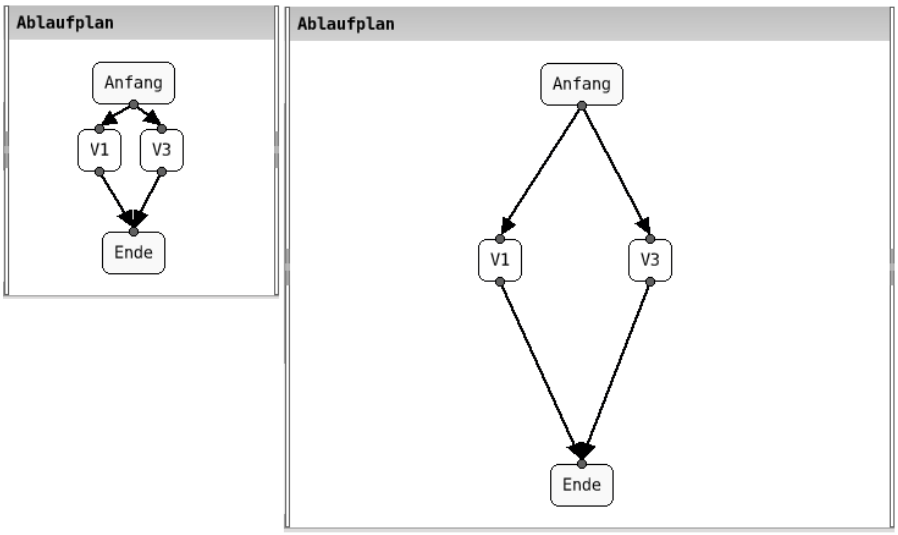

Abbildung 12: Der skalierbare Ablaufplan mit einem dunkleren Anfang und Ende. Links und rechts sind die Splitter zum Vergrößern oder Vekleinern des Ablaufplans zu sehen.

[39]Die Navigation zu dieser Eingabemaske ist noch nicht optimal gelöst. Der Benutzer sollte zu jedem Zeitpunkt ohne Umwege dahin navigieren können.

5.5. Eigenschaftenbereich

Dieser Abschnitt widmet sich der Überarbeitung des Konfigurationsbereichs (ehemals Eigenschaftenbereich).

Problem 5.1 *Dem Eigenschaftenbereich steht zu wenig Fläche zur Verfügung.*

Der Konfigurationsbereich wird daher nach rechts verschoben. Von links nach rechts ist damit die Anordnung der drei Bereiche unterhalb des Menübereichs wie folgt: Bereiche für Funktionen, Ablaufplan, Konfigurationsbereich. Abbildung 13 zeigt die neue Anordnung der Bereiche.

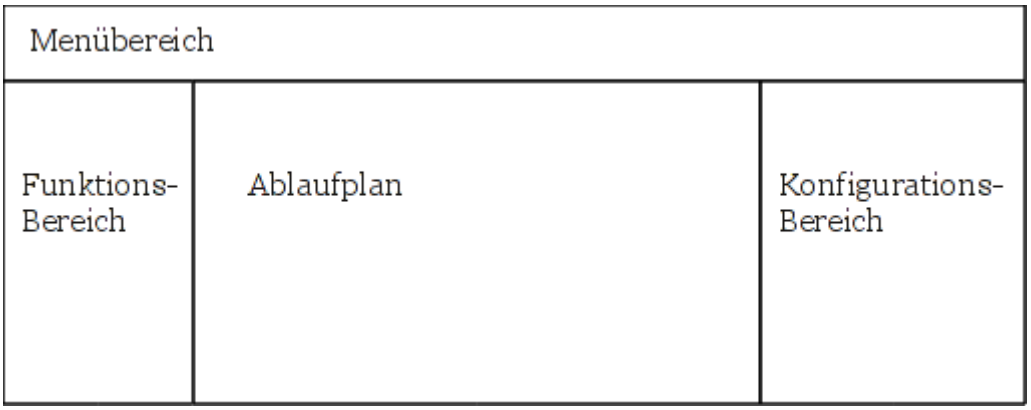

Abbildung 13: Das überarbeitete Layout für das Anwendungsfenster

Problem 5.2 *Der Eigenschaftenbereich ist anfangs leer und nicht hinreichend erklärt.*

Der Benutzer kann beim Erstellen oder Bearbeiten einer kombinierten Funktion Name, Beschreibung und Schlagwörter für die Kategorien im Konfigurationsbereich angeben. Die Abbildung 15 auf Seite 41 (links) zeigt hierfür die Eingabemaske. Die Vergabe von Schlagwörtern wurde sehr einfach gelöst: Der Benutzer trägt in jeder Zeile einen Kategorienamen ein. Der Nachteil dabei ist, dass bei Vergabe von bereits existierenden Kategorien der Benutzer darauf achten muss, dass der Kategoriename richtig geschrieben wird. Auch das Umbenennen einer Kategorie kann nur durchgeführt werden, indem alle betroffenen kombinierten Funktionen geändert werden. Dies wird jedoch als seltener Usecase angesehen. Der Vorteil besteht allerdings darin, dass keine unnötigen Extra-Bedienelemente notwendig sind (z. B. ein Button um eine neue Kategorie hinzuzufügen, zu ändern oder zu löschen).

Je nach Situation zeigt der Funktionsbereich Folgendes an:

1. *Ist eine elementare Funktion ausgewählt*, dann kann der Benutzer die Aktivierung und weitere Parameter anpassen. Falls ein Ablaufplan nicht auf oberster Ebene angezeigt wird, kann der Benutzer den Detailgrad verringern (Abbildung 14, links).
2. *Ist eine kombinierte Funktion ausgewählt*, dann kann der Benutzer die Aktivierung anpassen und den Detailgrad erhöhen. Falls ein Ablaufplan nicht auf oberster Ebe-

ne angezeigt wird, kann der Benutzer den Detailgrad auch verringern (Abbildung 14, rechts).

3. *Ist keine Funktion ausgewählt und...*

   a) *...der Ablaufplan auf oberster Ebene wird angezeigt,* dann kann der Benutzer über eine Eingabemaske Name, Beschreibung und Kategorien der zum obersten Ablaufplan gehörenden kombinierten der Funktion eingeben (Abbildung 15, links).

   b) *...der Ablaufplan nicht auf oberster Ebene angezeigt wird,* dann kann der Benutzer den Detailgrad verringern. (Abbildung 15, rechts).

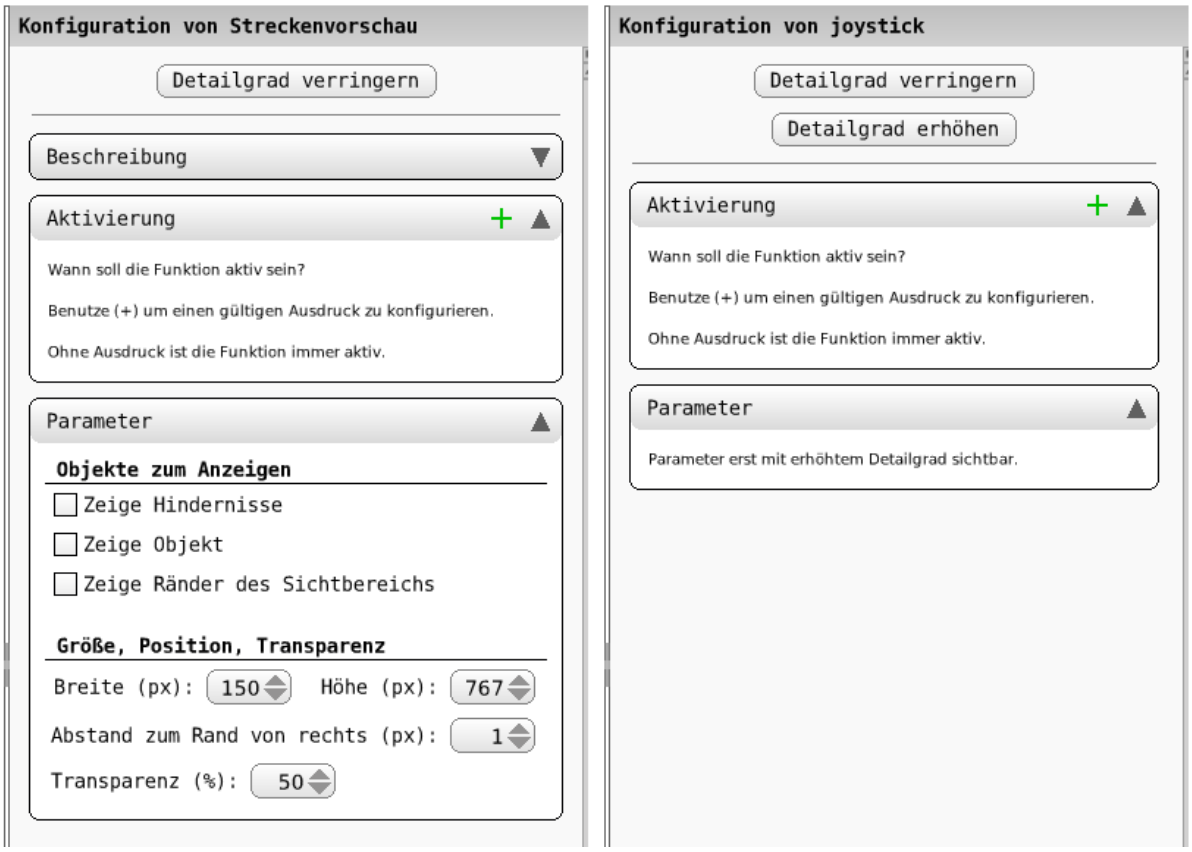

Abbildung 14: Konfigurationsbereich einer elementaren Funktion (links) und einer kombinierten Funktion (rechts) in einem Ablaufplan auf nicht oberster Ebene.

5.6. Verschiedenes

In diesem Abschnitt werden die restlichen Probleme bearbeitet.

Problem 6.1 *Die Bezeichner der Elemente überlagern das Element-Objekt.*

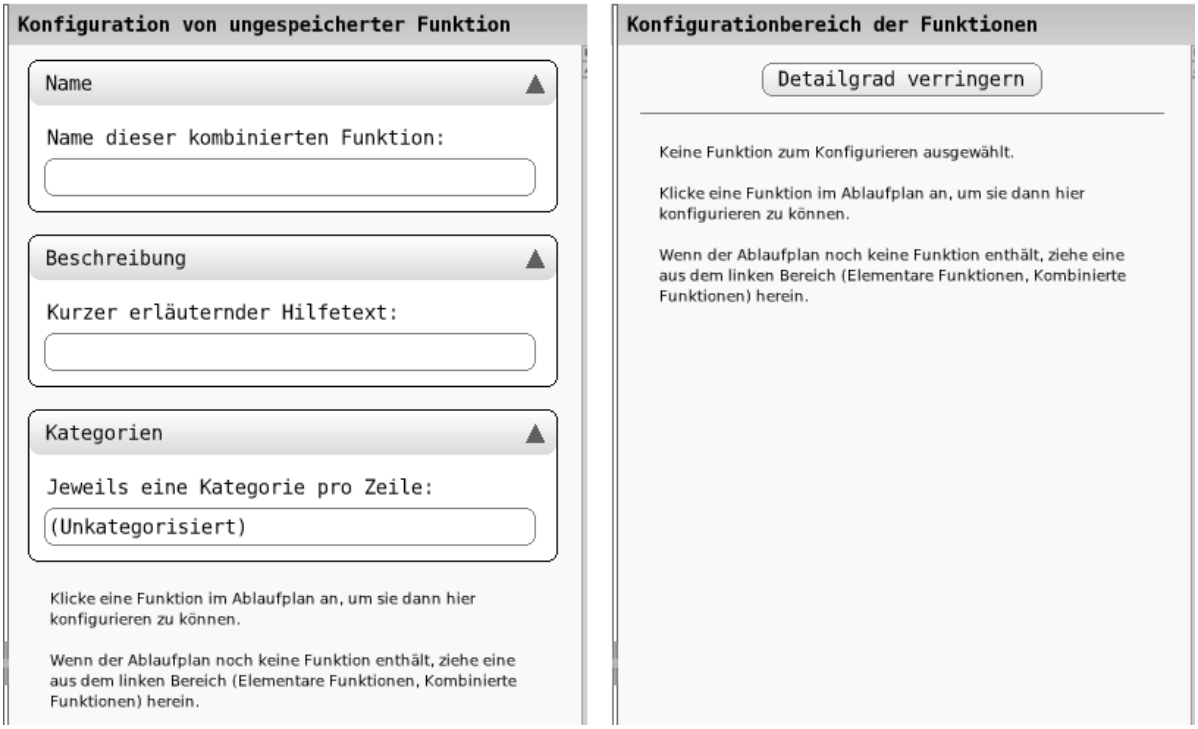

Abbildung 15: Konfigurationsbereich falls keine Funktion ausgewählt ist und der oberste Ablaufplan angezeigt wird (links); ...und ein nicht oberster Ablaufplan angezeigt wird (rechts)

Der visuelle, abgerundete Rahmen umfasst nun auch den Namen der Funktion. Im Bereich für elementare und kombinierte Funktionen weisen die Funktionen die gleiche Breite auf. Sehr lange Funktionsnamen werden vergekürzt dargestellt, wobei der vollständige Name über einen Tooltip eingesehen werden kann. Im Ablaufplan passt sich die Funktion so an, dass ihr ganzer Name angezeigt wird (Abbildung 16).

Abbildung 16: Der sehr lange Funktionsname wird mit „...“ abgekürzt. Zieht der Benutzer die Funktion in den Ablaufplan, nimmt sie die Größe ein, die notwendig ist, um den Funktionsnamen vollständig darzustellen.

Elementare und kombinierte Funktionen können jetzt leichter unterschieden werden. Kombinierte Funktionen haben einen fettgedruckten Namen, elementare Funktionen nicht. Weiterhin kodiert ein roter, dicker Rahmen einer kombinierten Funktion einen

unvollständigen Ablaufplan. Ausgewählte Funktion sind nun mit einem kräftigem blauen Hintergrund versehen (Abbildung 17).

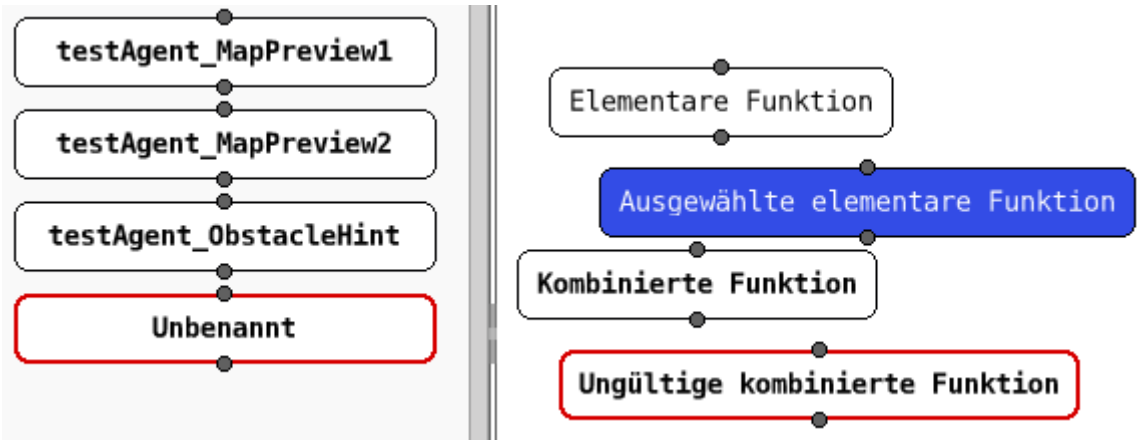

Abbildung 17: Alle visuellen Kodierungen der Funktionen im Überblick

Problem 6.2 *Die Navigation in den inneren bzw. äußeren Graphen erfolgt an unterschiedlichen Stellen.*

Das Problem wird behoben, indem der Detailgrad über entsprechende Buttons im Konfigurationsbereich engestellt werden kann (siehe Abbildungen 14 auf Seite 40 und 15 auf der vorherigen Seite). Der bisherige Button im Ablaufplan wird dazu entfernt.

Problem 6.3 *Falls ein komplexes Element geöffnet ist, dann ist unklar welche Datei zugehörig ist.*

Dieses Problem ist behoben. Die Abbildung 18 zeigt den Namen der geöffneten Datei in der Titelleiste[40].

Problem 6.4 *Benutzer wird nicht gewarnt bei der Speicherung eines ungültigen Graphen.*

Dieses Problem ist behoben. Die Abbildung 18 zeigt die Warnung.

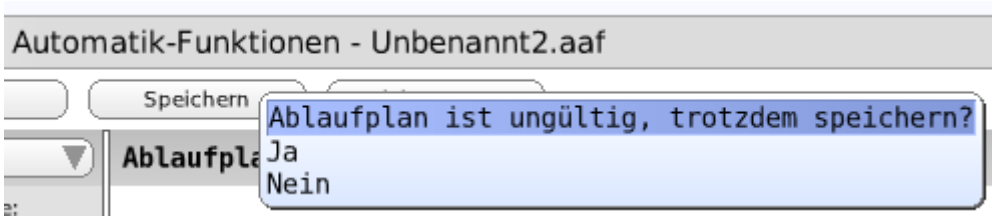

Abbildung 18: Beim Speichern eines ungültigen Ablaufplans erhält der Benutzer eine Warnung. Auch wird in der Titelleiste des Anwendungsfensters der Dateiname angezeigt.

[40]Die Anzeige des aktuellen Datei- oder Dokumentennamens in der Titelleiste ist weit verbreitet.

Problem 6.5 *Anwendungsstart und Speichervorgang dauern zu lange.*

Dieses Problem ist nun ebenfalls behoben. Die Ursache des Problems lag daran, dass u. a. von allen Automatiken die Funktionen vollständig für die Bearbeitung geladen wurden, obwohl das nicht nötig war. Für eine nähere Beschreibung sei auf den Abschnitt D.1 verwiesen.

Problem 6.6 *Der Menübereich verändert seine Höhe abhängig vom Anwendungsfenster.*

Der Menübereich weist nun stets eine konstante Höhe auf.

Problem 6.7 *Der „Verlassen"-Button passt logisch nicht zu „Neu", „Öffnen", „Speichern"*

Der „Beenden"-Button (ehemals Verlassen) wird nach ganz rechts in der Menüleiste verschoben. Rechts oben findet der Benutzer in der Regel auch einen Fenster-Button, um eine Anwendung zu beenden. Da dieser Button in Squeak gerade auf der linken Seite der Fenstertitelleiste ist, kommt dies dem Benutzer etwas entgegen.

Problem 6.8 *Es existiert keine Möglichkeit eine Kopie einer abgespeicherten Automatik zu erstellen, um mit dieser weiterzuarbeiten.*

Dem Menübereich wurde ein weiterer Button hinzugefügt: „Speichern unter...". Diese Bezeichnung und die damit verbundene Funktionalität ist dem Benutzer aus anderen Anwendungen vertraut.

6. Zusammenfassung

6.1. Wichtige Änderungen hervorgehoben

Mit dieser Arbeit wurden 29 Probleme in dem Automatiken-GUI „AAF Graph Tool“ identifiziert und bearbeitet. Einige Änderungen sollen an dieser Stelle hervorgehoben werden:

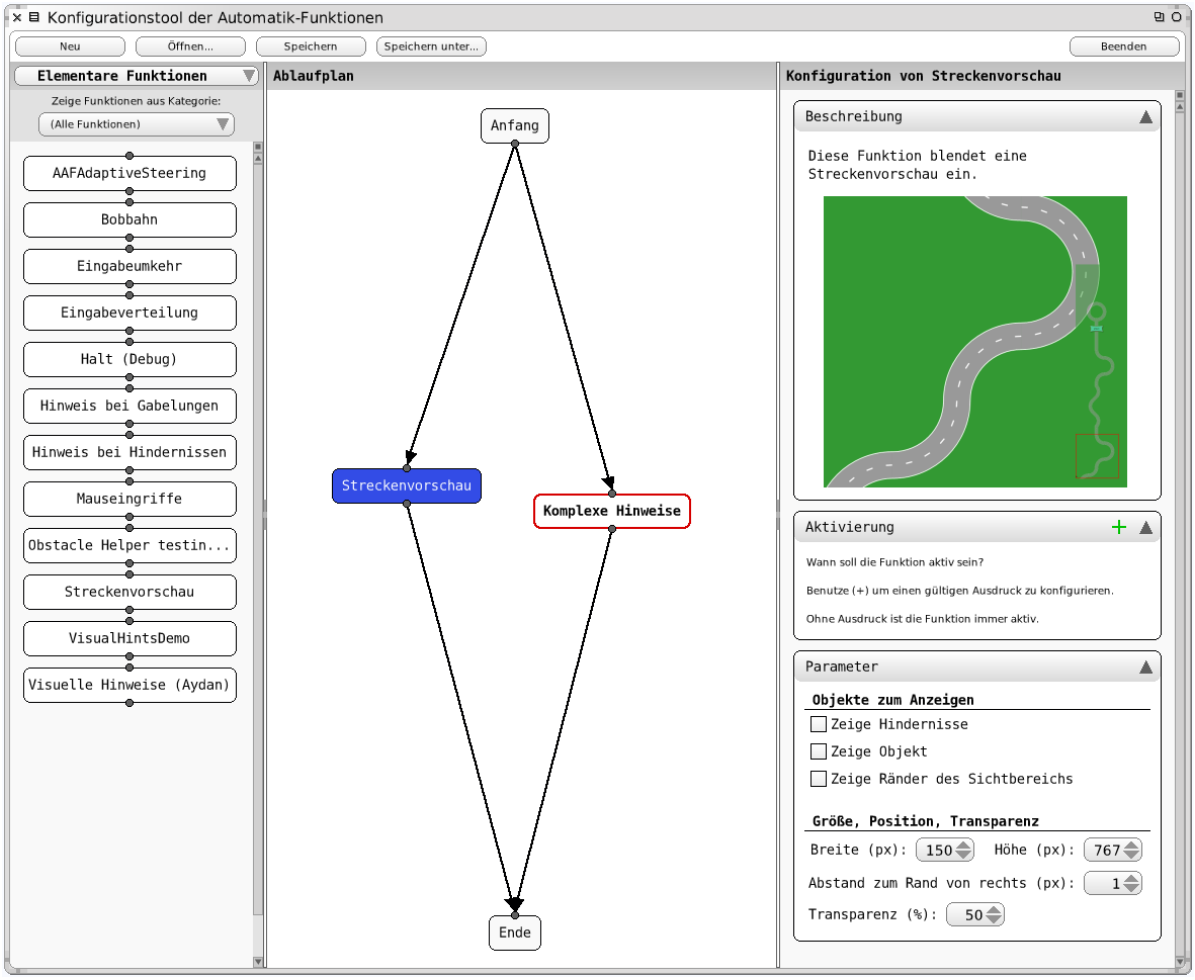

Abbildung 19: Das überarbeitete Autmatiken-GUI in Verwendung.

- Die Automatiken-GUI startet nun schneller und der Speichervorgang ist ebenfalls beschleunigt.
- Durch einheitliche Bedienelemente und eine dezente Farbauswahl wirkt die GUI viel konsistenter (s. Abbildung 19).
- Die Anwendung ist bis auf wenige Ausnahmen (s. Abschnitt 6.2) nun komplett in Deutsch gehalten.

- Die Funktionen können dem Benutzer nun besser erklärt werden - durch kurze Beschreibungen, die via Tooltips im Funktionsbereich erscheinen und durch längere Beschreibungen, die im Konfigurationsbereich aufklappbar sind (s. Abbildung 19 rechts).
- Der Konfigurationsbereich nimmt jetzt mehr Platz ein - diesen braucht er auch, wie in Abbildung 19 rechts zu sehen ist.
- Der Übersicht halber können Funktionen nun mehreren Kategorien zugeordnet werden. Elementare Funktionen werden vom Entwickler zugeordnet. Kombinierte Funktionen von Benutzern.
- Der Ablaufplan ist skalierbar.

6.2. Probleme

Im Folgenden sind Probleme aufgelistet, die noch bestehen und behoben werden sollten. Wegen des zeitlichen Rahmens wurden sie bisher nicht bearbeitet.

- *Englische Begriffe in blauen Popups*
  Es werden einige Standard-Widgets von Squeak benutzt. Diese sind per Voreinstellung blau gehalten. Abbildung 18 auf Seite 42 zeigt ein Beispiel. Problematischer sind aber Widgets, die englische Bezeichnungen haben, wie in Abbildung 26 auf Seite 52 gezeigt. Diese Widgets verstoßen gegen farbliche und sprachliche Konsistenz.

Größere Änderungen an Software verursachen auch neue, ungewollte Probleme. Die folgenden beiden Probleme sind während dieser Arbeit entstanden:

- *Ungenaue vertikale Scrollbalken*
  Horizontale Scrollbalken wurden zwar beseitigt, aber ein Problem mit vertikalen Scrollbalken ist hinzugekommen: Die Fläche im Funktionsbereich ist größer als nötig, weswegen zwar gescrollt werden kann, aber ohne, dass neue Funktionen zum Vorschein kommen (s. Abbildung 19 links unten).
- *Erkennung von Änderungen*
  Folgendes Vorgehen zeigt das Problem auf: *Öffne eine kombinierte Funktion ohne eine Änderung an dieser durchzuführen. Klicke „Neu“ in der Menüleiste an.* Es wird gefragt, ob die Änderungen an der aktuellen Funktion gespeichert werden, obwohl keine Änderungen durchgeführt worden sind.

Literatur

[1] Prof. Dr. Klaus Bothe, Michael Hildebrandt, and Nicolas Niestroj. *ATEO-SYSTEM Komponente: SAMS*. 2009. https://www2.informatik.hu-berlin.de/swt/lehre/PR_MTI_0910/restricted/literature/Verhaltenspezifikation%20SAMs_20-10-09_commentable.pdf.

[2] Johannes Brauer. *Grundkurs Smalltalk - Objektorientierung von Anfang an*, volume 3. Vieweg+Teubner, 2009.

[3] Alan Cooper, Robert Reimann, and David Cronin. *About Face - Interface und Interaction Design*. mitp, 2010.

[4] Markus Dahm. *Grundlagen der Mensch-Computer-Interaktion*. Pearson Studium, 2006.

[5] DIN. *Software-Ergonomie - Empfehlungen für die Programmierung und Auswahl von Software*. Beuth, 1 edition, 2003.

[6] Esther Fuhrmann. *Entwicklung eines GUI für die Konfiguration der Software-Komponente zur Systemprozessüberwachung und -kontrolle in einer psychologischen Versuchsumgebung*. 2010. Diplomarbeit.

[7] Adele Goldberg and David Robson. *Smalltalk-80 - The Language*, volume 1. Addison Wesley, 1989.

[8] Philip D. Gray and Ramzan Mohamed. *Smalltalk-80 - A Practical Introduction*, volume 1. Pitman, 1990.

[9] Andreas M. Heinecke. *Mensch-Computer-Interaktion*. Fachbuchverlag Leipzig, 2004.

[10] Jeff Johnson. *GUI Bloopers 2.0 - Common User Interface Design Don'ts and Dos*. Elesevier, 1 edition, 2008.

[11] Saskia Kain. *Entwickler in komplexen Mensch-Maschine-Systemen: Analyse des Einflusses von Entwicklerressourcen auf den Entwicklungsprozess und das -ergebnis*. 2011. Dissertation in Vorbereitung.

[12] John Maloney. An introduction to morphic: The squeak user interface framework, 2000. http://stephane.ducasse.free.fr/FreeBooks/CollectiveNBlueBook/morphic.final.pdf.

[13] John Maloney. Tutorial: Fun with the morphic graphics system, 2011. http://static.squeak.org/tutorials/morphic-tutorial-1.html.

[14] Jakob Nielsen. Ten usability heuristics, 2005. http://www.useit.com/papers/heuristic/heuristic_list.html.

[15] Ben Shneiderman and Catherine Plaisant. *Designing The User Interface*. Pearson Education, 2005.

[16] SqueakCommunity. Squeak smalltalk, 2011. http://www.squeak.org/.

[17] Alois Stritzinger. *Komponentenbasierte Softwareentwicklung*, volume 1. Addison Wesley Longman Verlag GmbH, 1997.

Alle Verweise wurden am 17. September 2011, 18:14 Uhr überprüft.

A. Aktualisierte Bedienungsanleitung der Automatiken-GUI

Die aktuelle Version dieser Bedienungsanleitung befindet sich im GIT-Repository des ATEO-Projekts im Verzeichnis doc[41].

Die Aktualisierung umfasst im Wesentlichen folgende Änderungen:

- Aktualisierung der Quelltexte
- Neuer Unterabschnitt „Einführung“
- Neuer Unterabschnitt „Kategoien des Agenten“
- Neuer Unterabschnitt „Den Agenten dem Benutzer erklären“
- Kleine Korrekturen (Eindeutigkeit von Formulierungen, Rechtschreibung, Kommasetzung)

A.1. Allgemeines

Diese Bedienungsanleitung ist für die Benutzungsschnittstelle *Konfigurationstool der Automatik-Funktionen (ATEO)*, im Folgenden Automatiken-GUI genannt, gedacht. Abbildung 20 zeigt die Automatiken-GUI in Verwendung.

[41] http://www.assembla.com/code/ATEO/git/nodes/doc (abgerufen am 17. September 2011, 16:47 Uhr)

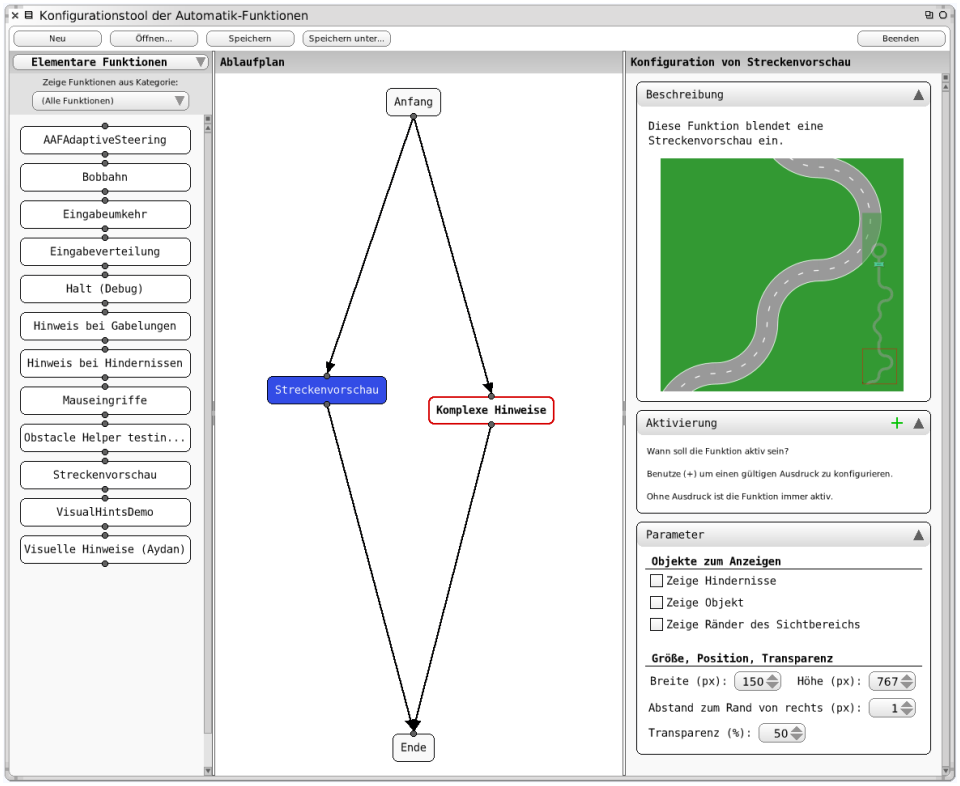

Abbildung 20: Die Automatiken-GUI in Aktion

A.2. Starten des Automatiken-GUI

Das Automatiken-GUI wird gestartet, indem in einem Workspace-Fenster in Squeak Folgendes ausgeführt wird:

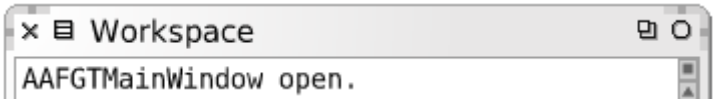

Abbildung 21: Starten des Automatiken-GUI

Abbildung 22 zeigt die gestartete Automatiken-GUI. Oben befindet sich eine Menüleiste mit wichtigen Programmfunktionen. Links ist der Funktionsbereich aus dem elementare und kombinierte Funktionen ausgewählt und in den Ablaufplan gezogen werden können. In der Mitte werden in dem Ablaufplan die Funktionen angeordnet. Eine mit der Maus ausgewählte Funktion (Links-Klick) kann rechts im Konfigurationsbereich angepasst werden.

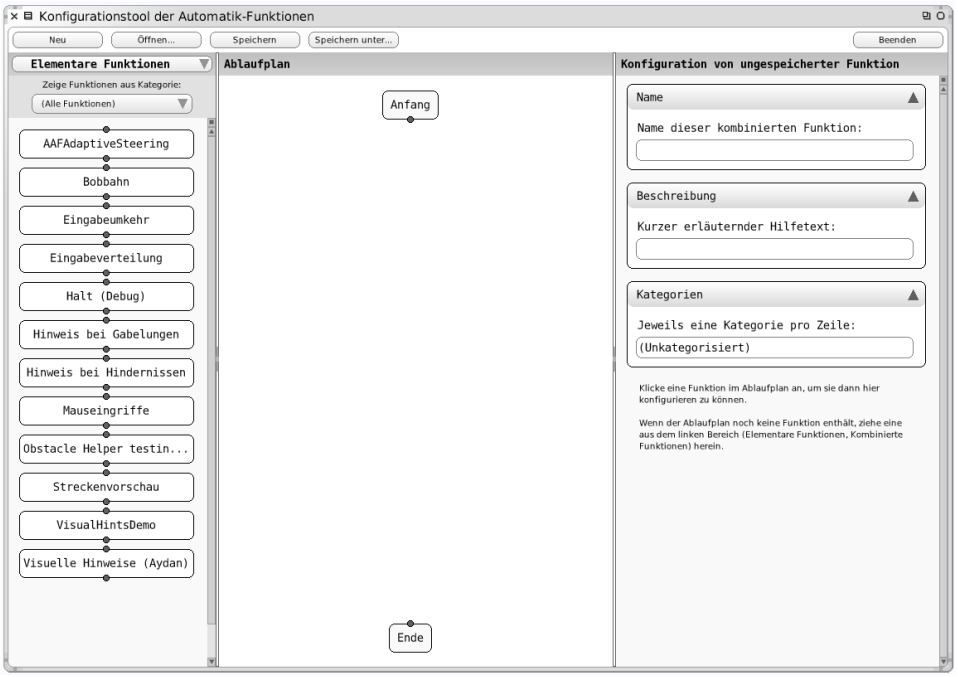

Abbildung 22: Die gestartete Automatiken-GUI

A.3. Aufbau einer Automatik

A.3.1. Funktionen hinzufügen

Funktionen werden zur Automatik hinzugefügt, indem sie aus einem der beiden Funktionsbereiche auf der linken Seite in den Ablaufplan im mittleren Bereich der Anwendung gezogen und dort fallen gelassen werden (siehe Abbildung 23).

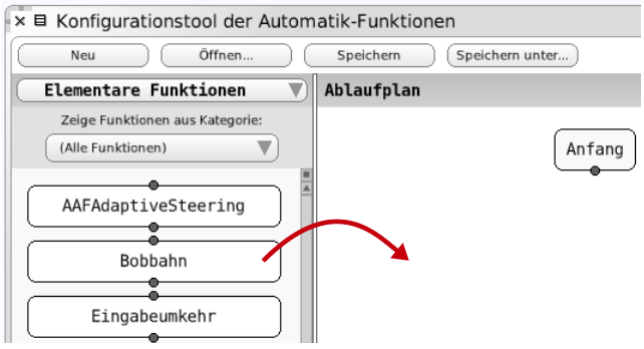

Abbildung 23: Eine Funktion kann aus dem Funktionsbereich durch anklicken und ziehen dem Ablaufplan hinzugefügt werden.

A.3.2. Verbinden von Funktionen

Zum Verbinden von Funktionen wird folgendermaßen vorgegangen:

1. Einen Verbindungspunkt einer Funktionen anklicken. Er ist nun aktiviert und wird orange hervorgehoben (s. Abbildung 24).

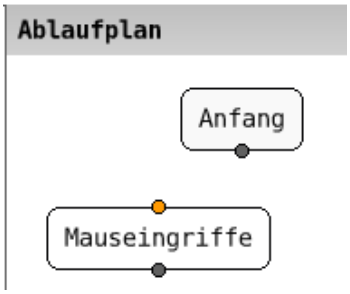

Abbildung 24: Der erste Verbindungspunkt ist aktiviert.

2. Anschließend den Verbindungspunkt anklicken, mit dem die Verbindung hergestellt werden soll. Die Verbindungslinie wird gezogen (s. Abbildung 25).

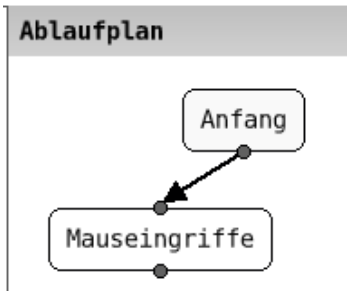

Abbildung 25: Die Verbindung zwischen zwei Funktionen wurde hergestellt.

Soll ein bereits aktivierter und rot angezeigter Verbindungspunkt doch nicht verbunden werden, so wird er durch erneutes Anklicken deaktiviert.

Beim Ziehen von Verbindungen ist zu beachten, dass nur jeweils ein Eingang mit einem Ausgang verbunden werden kann. Dabei ist es gleichgültig, ob die Verbindung vom Eingang zum Ausgang oder umgekehrt gesetzt wird. Eingang und Ausgang der selben Funktion können nicht verbunden werden.

Jede Funktion mit Ausnahme von Anfang und Ende kann nur eine eingehende und eine ausgehende Verbindung herstellen.

Der Anfang kann beliebig viele ausgehende, das Ende beliebig viele eingehende Verbindungen haben.

A.3.3. Benennung einer Funktion ändern

Um den Namen einer Funktion zu ändern, wird mit der rechten Maustaste das Kontextmenü aufgerufen und “Umbenennen” ausgewählt. In das erscheinende Texteingabefeld kann nun ein neuer Name eingegeben werden (s. Abbildung 26).

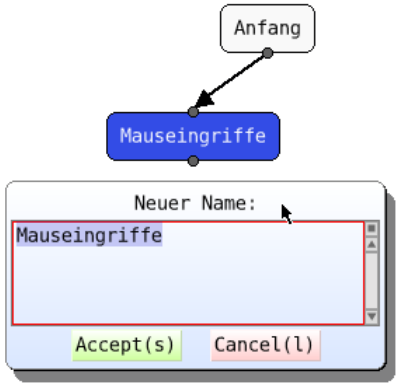

Abbildung 26: Eine Funktion kann umbenannt werden.

A.3.4. Löschen einer Verbindung

Um eine existierende Verbindung zwischen zwei Funktionen wieder zu löschen, wird die Verbindung mit der rechten Maustaste angeklickt. Im erscheinenden Kontextmenü wird nun der Punkt “Verbindung löschen” ausgewählt (s. Abbildung 27).

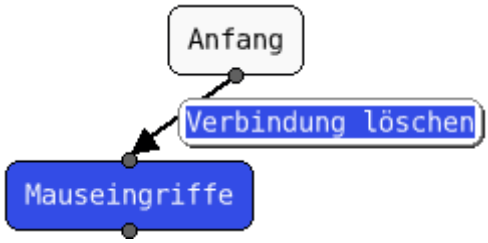

Abbildung 27: Über das Kontextmenü können Verbindungen wieder gelöscht werden.

A.3.5. Löschen einer Funktion

Soll eine Funktion aus der Automatik entfernt werden, so wird diese mit der rechten Maustaste angeklickt. Im nun erscheinenden Kontextmenü wird der Punkt “Löschen” ausgewählt (s. Abbildung 28). Ist die Funktion mit anderen Funktionen verbunden, so werden die Verbindungen ebenfalls gelöscht.

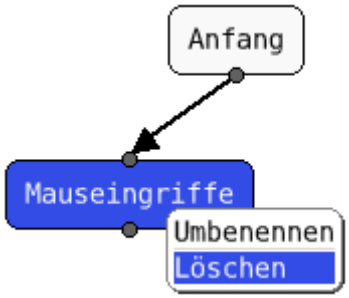

Abbildung 28: Über das Kontextmenü kann eine Funktion gelöscht werden.

Alternativ lässt sich eine Funktion auch löschen, indem sie “zurück” in den Funktionsbereich gezogen wird. Dabei wird die Anzeige gegebenenfalls zum passenden Funktionsbereich wechseln.

A.3.6. Bearbeitung der Parameter einer Funktion

Um die Einstellungen einer Funktion zu bearbeiten, wird die Funktion per Links-Mausklick ausgewählt. Die zugehörigen Einstellungen erscheinen nun im Konfigurationsbereich auf der rechten Seite und können bearbeitet werden (s. Abbildung 29). Nicht alle Funktionen haben Einstellungen.

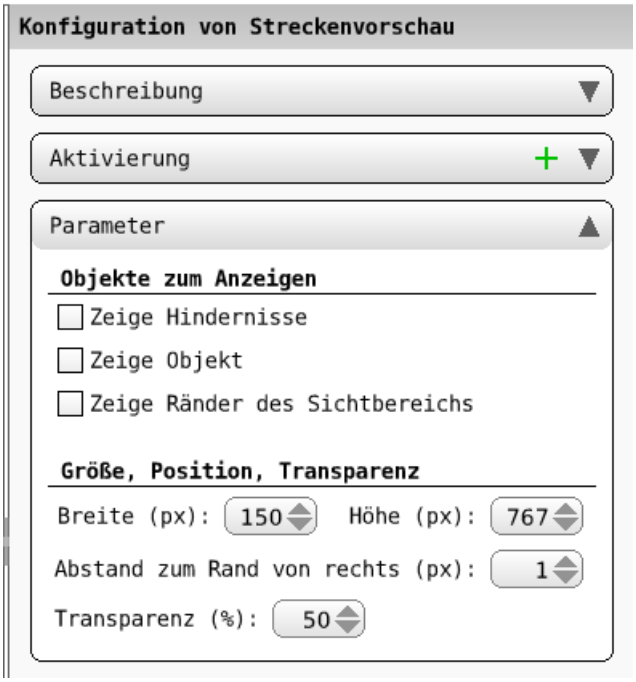

Abbildung 29: Im Konfigurationsbereich können die Einstellungen einer ausgewählten Funktion bearbeitet werden.

A.3.7. Bearbeitung der Aktivierung einer Funktion

Für jede Funktion kann festgelegt werden, wann diese aktiviert werden soll. Dies wird über die Aktivierung im Konfigurationsbereich eingestellt (s. Abbildung 30). Durch einen booleschen Ausdruck wird die Aktivierungszeitspanne spezifiziert. Wird kein Ausdruck erstellt, so ist die Funktion während der ganzen Ausführung aktiv.

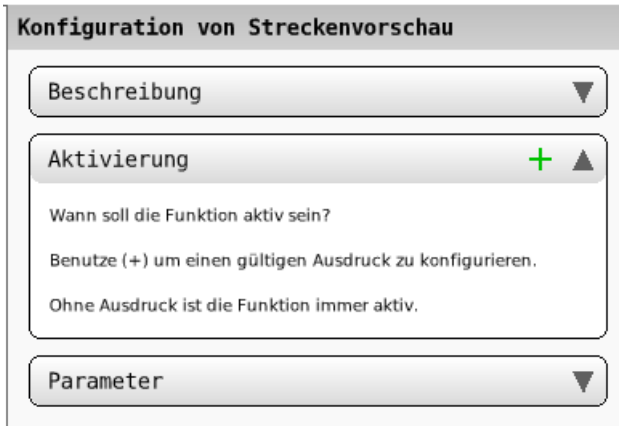

Abbildung 30: Über die Aktivierung kann eingestellt werden, wann eine Funktion aktiviert wird.

A.3.8. Detailgrad erhöhen und verringern

Kombinierte Funktionen sind an den fett gedruckten Bezeichnern erkennbar. Ihr Detailgrad im Ablaufplan kann erhöht werden, um weitere Einstellungen einzusehen und zu verändern.

Ist eine kombinierte Funktion angeklickt, so wird im Konfigurationsbereich ein Button "Detailgrad erhöhen" angezeigt, über den der Ablaufplan der angeklickten, kombinierten Funktion angezeigt werden kann (s. Abbildung 31). Dieses erscheint dann anstelle des bis dahin sichtbaren Ablaufplans und kann auf die gleiche Weise bearbeitet werden. Alternativ lässt sich das der Detailgrad über ein Doppelklick mit linker Maustaste auf die Funktion im Ablaufplan erhöhen. Eine weitere Möglichkeit bietet das Kontextmenü der Funktion.

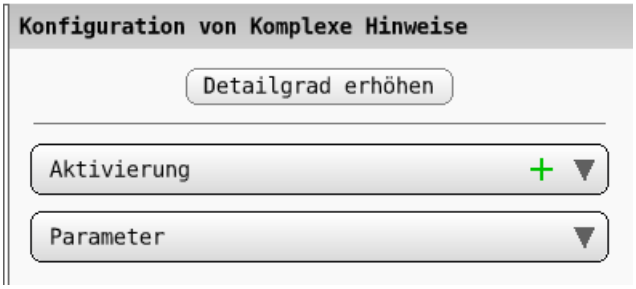

Abbildung 31: Kombinierte Funktionen bestehen haben einen eigenen Ablaufplan. Dieser lässt sich durch den Button "Detailgrad erhöhen" anzeigen und bearbeiten.

Nach der Bearbeitung kann der vorherige Ablaufplan angezeigt werden. Dazu klickt der Benutzer auf den Button "Detailgrad verringern" an der gleichen Stelle im Konfigurationsbereich. Alternativ lässt sich auch hier über das Kontextmenü des Ablaufplans (rechte Maustaste auf freien Bereich innerhalb des Ablaufplans) sowie über einen Doppelklick auf eine freie Fläche des Ablaufplans der Detailgrad verringern.

Achtung: Wird eine Funktion gespeichert, während ein erhöhter Detailgrad eingestellt ist, so werden alle Funktionen der aktuell bearbeiteten Datei (Siehe Titelleiste des Fensters) gespeichert.

A.4. Einbinden bestehender Automatiken

Bereits bestehende Automatiken können als Funktionen neuer Automatiken verwendet werden. Dazu stellt das Automatiken-GUI alle Automatiken, die im Standardverzeichnis für gespeicherte Automatiken liegen und lauffähig sind, im Funktionsbereich für kombinierten Funktionen zur Verfügung (s. Abbildung 32). Sie können genauso wie elementare Funktionen verwendet werden.

Abbildung 32: Existierende Automatiken stehen als kombinierte Funktionen in neuen Automatiken zur Verfügung.

A.5. Speichern einer Automatik

Um eine Automatik in eine Datei abzuspeichern, wird in der Menüleiste (s. Abbildung 33) der Button Speichern ausgewählt. Enthält die Automatik Funktionen, die nicht oder nur teilweise mit anderen Funktionen verbunden sind, erscheint eine Warnung die den Benutzer informiert. Fährt der Benutzer mit der Speicherung fort, wird diese Automatik mit dem ungültigen Ablaufplan im Funktionsbereich für kombinierte Funktionen mit einem roten Rahmen hervorgehoben.

Abbildung 33: Über die Menüleiste sind die Programmfunktionen Neu, Öffnen, Speichern, Speichern unter und Beenden zugänglich.

Über einen Texteingabe-Dialog kann der Benutzer den Namen der Datei angeben. Dabei ist zu beachten, dass der Dateiname mit .aaf enden sollte und dass die Speicherung automatisch in das Standardverzeichnis für Automatik-Dateien [42] erfolgt. Existiert der gewählte Dateiname bereits, so erhält der Benutzer eine entsprechende Warnung und hat die Möglichkeit, einen anderen Namen anzugeben.

Um eine Kopie der aktuellen Datei zu erzeugen und mit dieser weiterzuarbeiten, kann der Button "Speichern unter..." verwendet werden. Im darauf folgenden Texteingabe-Dialog wird nach dem Dateinamen für die Kopie gefragt.

[42] Das Verzeichnis, in dem abgespeicherte Automatiken abgelegt werden, heißt config.agents

A.6. Laden einer Automatik

Es gibt zwei Möglichkeiten eine früher in eine Datei abgespeicherte Automatik zur erneuten Bearbeitung zu öffnen:

1. Der Button Öffnen in der Menüleiste (Abbildung 33) wird ausgewählt. Es erscheint ein Auswahldialog, der alle im Standardverzeichnis vorhandenen Automatiken als Dateien zur Auswahl stellt (s. Abbildung 34).

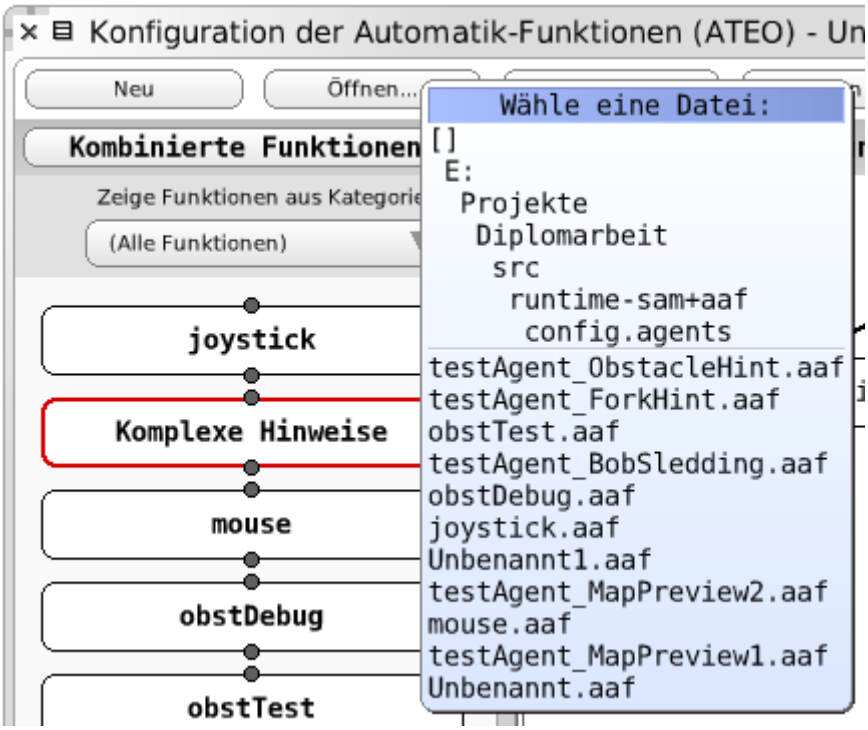

Abbildung 34: Beim Laden von Automatiken werden die verfügbaren zur Auswahl gestellt.

2. Über das Kontextmenü einer Funktion im Funktionsbereich für kombinierte Funktionen kann Öffnen ausgewählt werden.

A.7. Neue Automatik erstellen

Soll eine neue Automatik begonnen werden, so ist der Button Neu in der Menüleiste (Abbildung 33) zu verwenden. Der Ablaufplan wird geleert und die Bearbeitung kann von Neuem beginnen (s. Abbildung 22). Befinden sich in der aktuellen Automatik noch ungespeicherte Änderungen, so erhält der Benutzer beim Klick auf Neu eine entsprechende Warnung sowie die Möglichkeit, sie noch abzuspeichern.

Der Name, die Beschreibung und die Kategorien der neu zu erstellenden Automatik können im Konfigurationsbereich festgelegt werden (s. auch nachfolgenden Abschnitt).

A.8. Allgemeine Eigenschaften einer Automatik

Allgemeine Eigenschaften einer Automatik, wie ihr Name, ihre Beschreibung und die zugeordneten Kategorien können im Konfigurationsbereich bearbeitet werden, wenn im Ablaufplan auf oberster Ebene (geringster Detailgrad) keine Funktion ausgewählt ist (s. Abbildung 35).

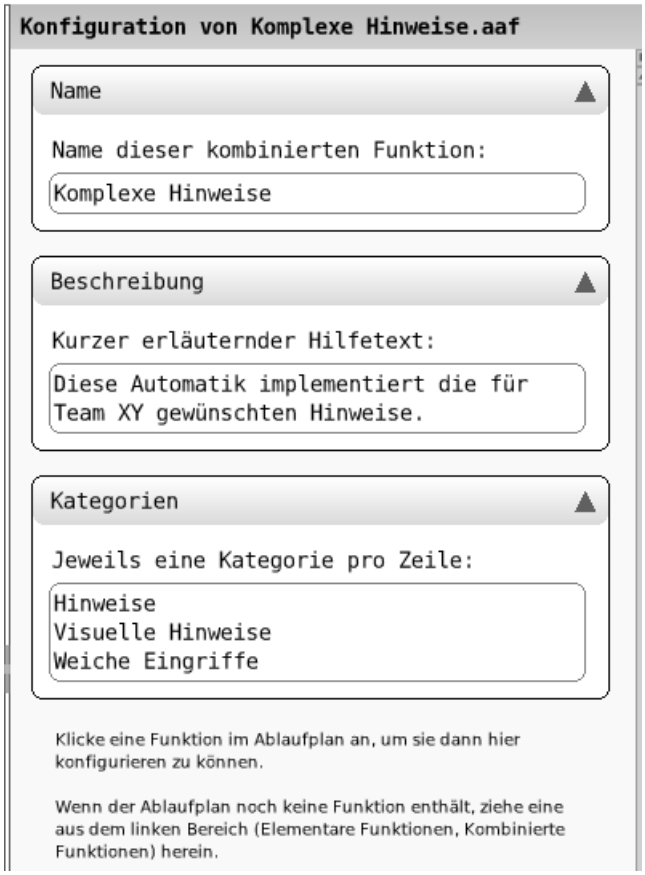

Abbildung 35: Im Konfigurationsbereich können ein Name, eine Beschreibung sowie verschiedene Kategorien vergeben werden.

Der Name der Automatik bzw. der kombinierten Funktion ist nicht der Dateiname, sondern der Name, wie er im Automatiken-GUI angezeigt wird, wenn abgespeichert wird.

Die Beschreibung soll für den Benutzer einen kleinen Hilfetext zur Verfügung stellen. Sie erscheint als Tooltip, wenn die Maus im Konfigurationsbereich für kombinierte Funktionen über einer Funktion verweilt.

Mit Hilfe von Schlagwörter können jeder Funktion mehrere Kategorien zugeordnet werden. Die Auswahl vorhandener Automatiken bzw. kombinierter Funktionen kann nach den vergebenen Kategorien gefiltert werden (s. auch nachfolgenden Abschnitt).

A.9. Auswahl elementarer und kombinierter Funktionen

Im linken Bereich des Fensters werden die zur Verfügung stehenden Funktionen angezeigt. Über ein Dropdown-Menü kann zwischen der Anzeige von elementaren und kombinierten Funktionen gewechselt werden. Ein weiteres Dropdown-Menü lässt nur Funktionen einer bestimmten Kategorie anzeigen. Die Kategorien für kombinierte Funktionen können vom Benutzer vergeben werden (vgl. Abbildung 35).

Im Funktionsbereich für kombinierte Funktionen werden ungültige Funktionen mit einem roten Rahmen hervorgehoben - diese enthalten im Ablaufplan Funktionen, die nicht oder nur teilweise verbunden sind.

A.10. Verlassen des Automatiken-GUI

Das Automatiken-GUI kann alternativ über den Beenden-Button in der Menüleiste (Abbildung 33) oder das Kreuz in der Titelleiste verlassen werden. In beiden Fällen erhält der Benutzer eine Warnung, falls noch nicht gespeicherte Änderungen vorliegen, sowie die Möglichkeit, sie noch abzuspeichern.

B. Aktualisierte Anleitung zum Integrieren von Agenten in die GUI

Die aktuelle Version dieser Integrationsanleitung befindet sich im GIT-Repository des ATEO-Projekts im Verzeichnis doc[43].

Die Aktualisierung umfasst im Wesentlichen folgende Änderungen:

- Anpassung an die neuen Begriffe (Ablaufplan statt Arbeitsbereich, Funktionen statt Elemente, ...)
- Aktualisierung der Screenshots
- Neuer Unterabschnitt „Allgemeines“
- Neuer Unterabschnitt „Allgemeine Eigenschaften einer Automatik“
- Kleine Korrekturen (Eindeutigkeit von Formulierungen, Rechtschreibung, Kommasetzung)

B.1. Einführung

In dieser Anleitung werden die einzelnen Schritte beschrieben, die durchgeführt werden müssen, um einen neuen AAF-Agenten im Automatiken-GUI als Funktion zur Verfügung zu stellen.

Abbildung 36 zeigt das Automatiken-GUI mit den für diese Anleitung zwei wichtigen Bereichen. Links ist der Funktionsbereich, in dem elementare und kombinierte Funktionen angezeigt werden. Rechts ist der Konfigurationsbereich einer Funktion.

[43] http://www.assembla.com/code/ATEO/git/nodes/doc (abgerufen am 17. September 2011, 16:47 Uhr)

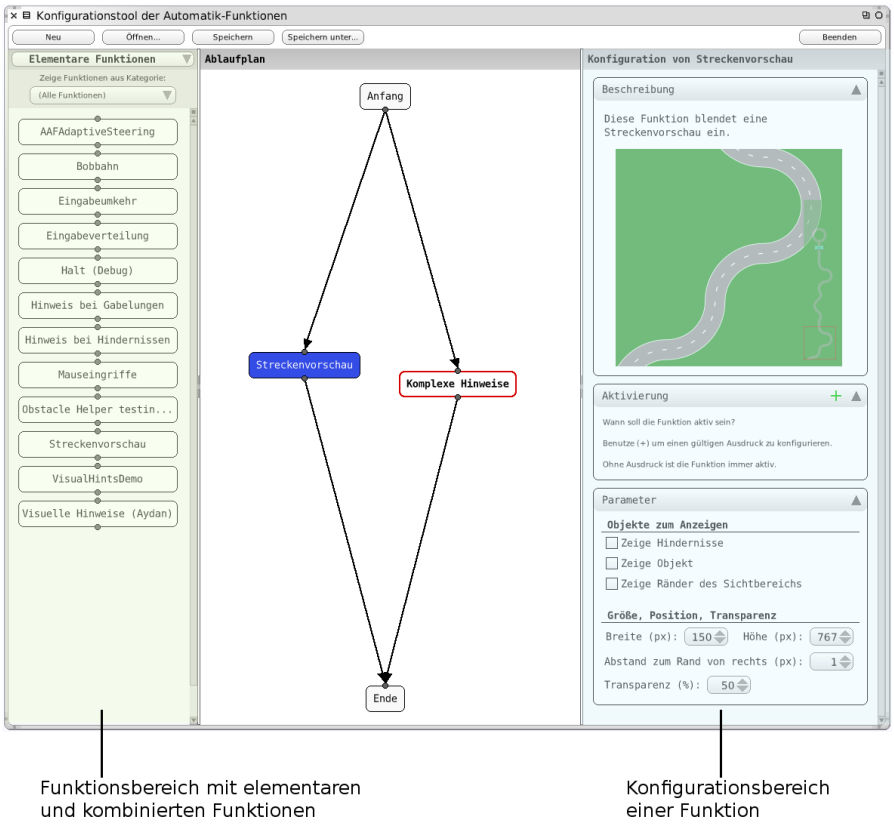

Abbildung 36: Das Automatiken-GUI mit hervorgehobenen Funktions- und Konfigurationsbereich.

B.2. Erstellen eines passenden Agenten

Um einen AAF-Agenten im Automatiken-GUI integrieren zu können, muss er zunächst erstellt werden. Hierbei sind einige Bedingungen zu erfüllen, damit der fertig implementierte Agent ins Automatiken-GUI eingebunden und später in einer Automatik verwendet werden kann.

Die Quelltext-Beispiele sind hauptsächlich aus den Klassen AAFMouseInputAgent und AAFMouseInputAgentDialog entnommen. Dabei wurde die Formatierung geringfüging verändert um die Darstellung in diesem Dokument zu optimieren.

B.2.1. Ableiten von AAFAgent

Der Agent muss eine Ableitung von AAFAgent sein. Dabei muss die Ableitung nicht direkt sein, es ist ebenso möglich, über eine oder mehrere Zwischenklassen von AAFAgent zu erben.

```
AAFAgent subclass: #AAFMouseInputAgent
```

```
instanceVariableNames: 'mwi scaleFactorX scaleFactorY'
classVariableNames: ''
poolDictionaries: ''
category: 'AAF-Agents-Dev'
```

B.2.2. Klartextname des Agenten

Damit das Automatiken-GUI die Funktion mit einer passenden Aufschrift versehen kann, muss der Klartextname des Agenten von der Klassenmethode plaintextName als Zeichenkette zurückgeliefert werden.

```
plaintextName
    ^ 'Mauseingriffe'
```

B.2.3. Einsatzbereitschaft anzeigen

Ist der Agent fertig implementiert und prinzipiell als Teil einer Automatik einsatzfähig, so muss dies angezeigt werden. Für diesen Zweck haben AAF-Agenten eine Klassenmethode readyForUse, welche einen booleschen Wert zurückgibt. Die geerbte Standardimplementierung liefert den Wert false. Soll der Agent für die Verwendung freigegeben werden, so muss der Programmierer diese Methode überschreiben und den Wert true zurückgeben.

```
readyForUse
    ^ true
```

Für Details zu den Anforderungen, die ein AAF-Agent erfüllen muss, um als Teil eines AAF-Graphen einsetzbar zu sein, sei hier auf die Dokumentation des AAF verwiesen.

Sind die beiden bis hierher genannten Voraussetzungen erfüllt, so wird im Automatiken-GUI bereits eine Funktion für den Agenten erstellt und im Funktionsbereich bereitgestellt.

Besitzt der Agent keinerlei Eigenschaften, die über die geerbten hinausgehen, so ist die Integration in das Automatiken-GUI an dieser Stelle abgeschlossen und die erstellte Funktion kann verwendet werden. Besitzt er weitere Eigenschaften, so sind noch weitere Schritte nötig, damit diese gespeichert, geladen und angepasst werden können.

B.2.4. Kategorien des Agenten

In der Automatiken-GUI kann die Anzeige von elementaren Funktionen im Funktionsbereich durch vergebene Kategorien gefiltert werden. Einem Agenten können mehrere Kategorien zugeordnet werden. Diese werden als eine SequenceableCollection, zum Beispiel ein Array, in der Klassenmethode tags des Agenten festgelegt:

```
tags
    ^ #('(Entwicklung)')
```

B.2.5. Abfrage und Setzen spezieller Eigenschaften

Besitzt der Agent Eigenschaften, die über die geerbten hinausgehen, so muss er die Instanzmethoden getAllProps und setAllProps erweitern. Diese bilden die Schnittstelle zum Auslesen und Setzen aller Eigenschaften, ohne die einzelnen Zugriffsmethoden kennen zu müssen. Benötigt wird diese Schnittstelle z. B. zum Speichern und Wiederherstellen von Automatiken.

getAllProps muss ein Dictionary zurückgeben, welches die Eigenschaften und ihre aktuellen Werte als Schlüssel-Wert-Paare enthält. Hierbei ist zu beachten, dass gleich zu Beginn die entsprechende Methode der Basisklasse aufgerufen werden muss, sodass auch die Eigenschaften enthalten sind, die geerbt wurden.

```
getAllProps
    | dict |
    dict := super getAllProps.
    dict at: 'mwi' put: self mwi.

    ^ dict
```

setAllProps erhält ein Dictionary als Parameter. Dieses enthält alle Eigenschaften des Agenten sowie ihre zu setzenden Werte als Schlüssel-Wert-Paare. Die Methode setAllProps ist nun dafür zuständig, die Werte über die entsprechenden Zugriffsmethoden den Eigenschaften zuzuweisen. Auch hier ist zu beachten, dass ein Teil der Werte durch die setAllProps-Methode der Basisklasse verarbeitet werden muss, um die geerbten Eigenschaften zu setzen.

```
setAllProps: aPropertyDictionary
    (aPropertyDictionary size >= 2)
    ifTrue: [
        super setAllProps: aPropertyDictionary.
        self mwi: (AAFUtils convert:
            (aPropertyDictionary at: 'mwi') type: 'Number').
    ].
```

An dieser Stelle ist der Agent so weit, dass er tatsächlich im Automatiken-GUI verwendet werden kann, auch Speichern und Laden sind möglich. Allerdings kann er bis hierher nur mit seinen Standardeigenschaften verwendet werden, welche sinnvoll vorbelegt sein sollten, da es noch keine Möglichkeit zum Konfigurieren gibt.

B.3. Erstellen des Konfigurationsdialogs

Damit die Eigenschaften des Agenten angezeigt und konfiguriert werden können, ist es notwendig, für den Agenten einen passenden Dialog zu erstellen. Die hierfür durchzuführenden Schritte werden im Folgenden erläutert.

B.3.1. Ableiten von AAFAgentDialog

Der Konfigurationsdialog für den Agenten muss eine Ableitung von AAFAgentDialog sein. Auch hier spielt es wieder keine Rolle, ob die Ableitung direkt oder über Zwischenklassen erfolgt.

```
AAFAgentDialog subclass: #AAFMouseInputAgentDialog
    instanceVariableNames: 'dropDownMenu'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'AAFGT-Dialog'
```

B.3.2. Benennung der Dialogklasse

Damit die Dialogklasse vom Automatiken-GUI gefunden und verwendet werden kann, muss sie den gleichen Namen tragen, wie der zugehörige Agent, erweitert durch die Zeichenkette „Dialog“. Heißt also die Agenten-Klasse beispielsweise MouseInputAgent, so heißt die zugehörige Dialogklasse MouseInputAgentDialog.

B.3.3. Hinzufügen der Bedienelemente für die speziellen Eigenschaften

Um Eigenschaften anzeigen und editieren zu können, die der Agent über die geerbten hinaus besitzt, muss die Methode addSpecialElements entsprechend erweitert werden. In dieser Klasse werden die entsprechenden Bedienelemente erzeugt und anschließend dem dafür vorgesehenen Bereich buttonArea zugefügt. Zuletzt werden die angezeigten Werte aktualisiert, indem sie vom Agenten abgefragt werden.

```
addSpecialElements
    "Sets up parts which are unique for this type of dialog."
    | container |
    dropDownMenu := AAFGTDropDownMenu
        value: 'MWB1' items: #('MWB1' 'MWB2')
        target: self action: #userChose:.

    container := AlignmentMorph newRow color: Color transparent.
    container
        addMorphBack: (AAFGTOneLineLabel contents:
            'Maussteuerung aktivieren für: ');
        addMorphBack: dropDownMenu.

    self buttonArea
        addMorph: (AAFGTWidgetUtils centered: container).

    self updateFromDelegate.
```

Die Erstellung der Bedienelemente kann nicht allgemein beschrieben werden, da sie stark davon abhängig ist, um welchen Datentyp es sich handelt und welche Art von Bedienelement zum Einsatz kommen soll. Die Bedienelemente auf die zurückgegriffen werden kann, finden sich in der *AAFGT Widget Gallery*. In einem Workspace kann diese via AAFGTWidgetGallery open. aufgerufen werden.

B.3.4. Aktualisieren der Anzeige bei Änderungen am Agenten

Damit der Dialog sich im Falle von Änderungen am Agenten korrekt aktualisiert, muss nun noch die Methode updateFromDelegate implementiert werden. Hierfür müssen die aktuellen Werte vom Agenten abgefragt und den entsprechenden Anzeigeelementen zugewiesen werden. Wichtig ist auch hier wieder der Aufruf der updateFromDelegate-Methode der Basisklasse, da nur so die Aktualisierung der Eigenschaften erfolgen kann, die der Agent geerbt hat.

```
updateFromDelegate
    | value |
    super updateFromDelegate.

    value := delegate mwi = 1
        ifTrue: [ 'MWB1' ]
        ifFalse: [ 'MWB2' ].

    dropDownMenu label: value.
```

Nun ist der Agent vollständig einsatzbereit und kann nicht nur gespeichert und geladen, sondern über den Dialog auch in seinen Eigenschaften angepasst werden.

B.3.5. Den Agenten dem Benutzer erklären

Durch die Implementierung von weiteren Methoden kann der Agent für den Benutzer besser erklärt werden.

Über die Klassenmethode shortDescription *der Agentenklasse* kann ein kleiner Hilfetext zurückgegeben werden:

```
shortDescription
    ^'Diese Funktion erlaubt die Steuerung eines',
     'Mikroweltbewohners mit der Maus'.
```

Verweilt der Benutzer mit der Maus über der Funktion im Funktionsbereich der Automatiken-GUI, dann erscheint dieser Hilfetext als Tooltip.

Über die Klassenmethode longDescription *der Dialogklasse* kann eine Hilfestellung zurückgegeben werden. Diese erscheint im Konfigurationsbereich der Funktion als eigener kleiner Abschnitt mit dem Titel "Beschreibung". Damit die Möglichkeit gegeben ist Bilder

anzuzeigen, wird nicht nur ein einfacher String, wie bei shortDescription zurückgegeben, sondern ein Morph. Sei hierfür ausnahmsweise die Methode longDescription der Klasse AAFMapPreviewAgentDialog betrachtet:

```
longDescription
    | container |
    container := AlignmentMorph newColumn color: Color transparent.
    container cellInset: 10.
    container
        addMorphBack: (
            AAFGTMultiLineLabel contents:
                'Diese Funktion blendet eine Streckenvorschau ein.'
        );
        addMorphBack: (
            (Form fromFileNamed: SAMConfig pathImagesHints,
                'doc_mapPreview.png') asMorph
        ).

^ container
```

Der zurückgegebene Morph container in diesem Beispiel umschließt einen Text sowie ein Bild.

C. Aktualisierte und ergänzte Prüfliste von manuellen Testfällen

Um die korrekte Arbeitsweise der Automatiken-GUI sicherzustellen, hat Fuhrmann eine Prüfliste von 19 manuellen Tests definiert[44]. Weil mit dieser Arbeit viele Änderungen am Quelltext durchgeführt worden sind, war die Durchführung der manuellen Tests notwendig. Alle Tests konnten erfolgreich ausgeführt worden, bis auf:

- *Testfall 2 nicht mehr anwendbar*
  Testfall 2 soll sichergehen, dass eine Funktion, die aus aus dem Funktionsbereich für elementare Funktionen in den Funktionsbereich für kombinierte Funktionen gezogen wird, verworfen wird. Weil nun nur noch ein Funktionsbereich sichtbar ist, ist dieser Testfall nicht mehr durchführbar und somit irrelevant.
- *Testfall 18 und 19 aktuell nicht mehr anwendbar*
  Testfall 18 und 19 testen die Aktivierung von Funktionen für bestimmte Streckenbereiche. Diese Funktionalität wird aktuell im Rahmen der Diplomarbeit von Nikolai Kosjar (genauer Titel bisher nicht festgelegt) überarbeitet und steht daher nicht zur Verfügung.

Die Testfälle wurden aktualisiert, um den Umbenennungen durch diese Arbeit (s. Abschnitt 5.1 auf Seite 31) gerecht zu werden. Testfall 11 musste auch inhaltlich angepasst werden. Testfall 11 sollte sichergehen, dass eine Funktion im Ablaufplan mit Doppelklick umbenannt werden kann. Das Umbenennen durch Doppelklick wurde aus Konsistenzgründen entfernt. Stattdessen kann eine Funktion im Ablaufplan über ihr Kontextmenü umbenannt werden.

Außerdem wurden fünf neue Testfälle (Testfall 20 - 24) hinzugefügt, die wichtige Änderungen dieser Arbeit abdecken:

- Testfall 20 testet ob die Vergabe von Name, Beschreibung und Kategorien einer Automatik bzw. kombinierten Funktion ordnungsgemäß gespeichert und geladen wird.
- Testfall 21 und 22 testen das Löschen von Funktionen im Ablaufplan durch ziehen in den Funktionsbereich.
- Testfall 23 testet die Skalierung des Ablaufplans.
- Testfall 24 testet die Filterung der Funktionen nach Kategorie im Funktionsbereich.

Im Folgenden werden alle Testfälle aufgeführt.

[44] vgl. [6], Anhang E, S. 137f.

Vorbedingung:	VB
Test:	T
Nachbedingung:	NB

Testfall 1	
VB	• Es befinden sich außer Anfang und Ende keine Funktionen im Ablaufplan. • Der zugehörige `AAF`-Graph besteht nur aus Anfang- und Endeknoten. • Anfang- und Endeknoten haben keine Verbindungen.
T	• Eine elementare Funktion wird von links aus dem Funktionsbereich in den Ablaufplan gezogen. • Die Funktion wird mit Anfang und Ende verbunden.
NB	• Die Funktion befindet sich im Ablaufplan. • Die Funktion ist mit Anfang und Ende verbunden. • Der zuhörige AAF-Graph besteht aus Anfang- und Endeknoten und einem normalen Knoten. • Der Typ des Agenten im normalen Knoten des AAF-Graphen passt zur Funktion. • Im `AAF`-Graphen hat der Anfangsknoten den normalen Knoten als Kindknoten. • Im `AAF`-Graphen hat der Endeknoten den normalen Knoten als Elternknoten. • Im `AAF`-Graphen hat der normale Knoten den Anfangsknoten als Elternknoten und den Endeknoten als Kindknoten.

Testfall 2 - NICHT MEHR ANWENDBAR	
VB	–
T	• Eine Funktion wird aus dem Funktionsbereich für elementare Elemente in den Funktionsbereich für komplexe Funktionen gezogen.
NB	• Die Funktion wurde dem Funktionsbereich für komplexe Funktionen nicht hinzugefügt. • Die Funktion wurde verworfen und existiert nicht mehr.

Testfall 3	
VB	• NB aus Testfall 1.
T	• Die Funktion wird im Ablaufplan verschoben.
NB	• Die Funktion wird an der neuen Position angezeigt. • Die Koordinaten des zugehörigen `AAF`-Knotens haben sich angepasst.

Testfall 4	
VB	• NB aus Testfall 1.
T	• Die Funktion wird aus dem Graphen gelöscht.
NB	• Die Funktion ist aus dem Graphen gelöscht. • Die Verbindungen zu Anfang und Ende sind aus dem Ablaufplan gelöscht. • Der zuhörige AAF-Graph besteht nur aus Anfangs- und Endeknoten. • Anfang- und Endeknoten haben keine Verbindungen.

Testfall 5	
VB	• NB aus Testfall 1. • Es existiert keine gespeicherte Automatik mit dem Namen *test.aaf*. • Es existiert keine kombinierte Funktion mit der Aufschrift *test.aaf* im Funktionsbereich für komplexe Funktionen.
T	• Abspeichern als *test.aaf*.
NB	• Die Datei *test.aaf* wurde im Standardspeicherverzeichnis geschrieben. • Die Datei *test.aaf* enthält den zum Graphen passenden korrekten XML-Code. • Im Funktionsbereich für kombinierte Funktionen wird eine neue Funktion mit der Aufschrift *test.aaf* angezeigt.

Testfall 6	
VB	• Es befinden sich außer Anfang und Ende keine Funktionen im Ablaufplan. • Der zugehörige `AAF`-Graph besteht nur aus Anfang- und Endeknoten. • Anfang- und Endeknoten haben keine Verbindungen. • Es existieren keine ungespeicherten Veränderungen.
T	• Öffnen von *test.aaf*.
NB	• NB wie Testfall 1. • Das einfache Element trägt die Aufschrift, die es beim Abspeichern trug. • Diese Aufschrift ist auf `AAF`-Ebene gesetzt.

Testfall 7	
VB	• NB aus Testfall 1.
T	• Klick auf einen der Verbindungspunkte am normalen Element.
NB	• Der Status des Verbindungspunkt ist *nicht aktiviert*. • Der Verbindungspunkt ist nicht rot.

Testfall 8	
VB	• Es befinden sich außer Anfang und Ende keine Funktionen im Ablaufplan. • Der zugehörige `AAF`-Graph besteht nur aus Anfang- und Endeknoten. • Anfang- und Endeknoten haben keine Verbindungen.
T	• Zwei Elemente A und B aus dem Funktionsbereich in das Ablaufplan ziehen. • Ausgang von A anklicken und so aktivieren. • Ausgang von B anklicken.
NB	• Der Ausgang von A ist rot. • Der Ausgang von B ist grau. • Der Status des Ausgangsverbindungspunktes von A ist *aktiviert*. • Der Status des Ausgangsverbindungspunktes von B ist *nicht aktiviert*. • Zwischen den beiden Ausgängen wird keine Verbindung angezeigt. • Zwischen den zu A und B gehörenden AAF-Knoten besteht keine Verbindung.

Testfall 9	
VB	• Es befinden sich außer Anfang und Ende keine Funktionen im Ablaufplan. • Der zugehörige `AAF`-Graph besteht nur aus Anfang- und Endeknoten. • Anfang- und Endeknoten haben keine Verbindungen.
T	• Zwei Elemente A und B aus den Funktionsbereich in das Ablaufplan. • Eingang von A anklicken und so aktivieren. • Eingang von B anklicken.
NB	• Der Eingang von A ist rot. • Der Eingang von B ist grau. • Der Status des Eingangsverbindungspunktes von A ist *aktiviert*. • Der Status des Eingangsverbindungspunktes von B ist *nicht aktiviert*. • Zwischen den beiden Eingängen wird keine Verbindung angezeigt. • Zwischen den zu A und B gehörenden AAF-Knoten besteht keine Verbindung.

Testfall 10	
VB	• Es befinden sich außer Anfang und Ende keine Funktionen im Ablaufplan. • Der zugehörige `AAF`-Graph besteht nur aus Anfang- und Endeknoten. • Anfang- und Endeknoten haben keine Verbindungen.
T	• Ein Element aus einem der Funktionsbereiche in das Ablaufplan. • Eingang des Elements anklicken und so aktivieren. • Ausgang desselben Elements anklicken.
NB	• Der Eingang ist rot. • Der Ausgang ist grau. • Der Status des Eingangsverbindungspunktes ist *aktiviert*. • Der Status des Ausgangsverbindungspunktes ist *nicht aktiviert*. • Zwischen Ein- und Ausgang wird keine Verbindung angezeigt. • Der zugehörige AAF-Knoten ist nicht mit sich selbst verbunden.

Testfall 11	
VB	• NB aus Testfall 1.
T	• Im Kontextmenü der Funktion Umbenennen wählen. • Neuen Namen vergeben.
NB	• Für die Funktion wird der neue Name angezeigt. • Der neue Name wurde als Label auf `AAF`-Ebene gesetzt.

Testfall 12	
VB	• Es befinden sich außer Anfang und Ende keine Funktionen im Ablaufplan. • Der zugehörige `AAF`-Graph besteht nur aus Anfang- und Endeknoten. • Anfang- und Endeknoten haben keine Verbindungen.
T	• Zwei Elemente A und B aus dem Funktionsbereich in den Ablaufplan ziehen. • Eingang von A anklicken. • Ausgang von B anklicken. • Ausgang von A anklicken. • Eingang von B anklicken.
NB	• Eingang und Ausgang von B werden grau angezeigt. • Die entsprechenden Verbindungspunkte haben den Status *nicht aktiviert*. • Eingang von A wird grau angezeigt. • Der entsprechende Verbindungspunkt hat den Status *nicht aktiviert*. • Ausgang von A wird rot angezeigt. • Der entsprechende Verbindungspunkt hat den Status *aktiviert*. • Zwischen dem Eingang von A und dem Ausgang von B wird eine Verbindung angezeigt. • Zwischen dem Ausgang von A und dem Eingang von B wird keine Verbindung angezeigt. • Die zu A und B gehörenden `AAF`-Knoten haben nur eine Verbindung, bei dieser ist A Kind- und B Elternknoten.

Testfall 13	
VB	• NB aus Testfall 1.
T	• Element im Ablaufplan verschieben. • *Neu*-Schaltfläche anklicken.
NB	• Der Hinweis auf ungespeicherte Änderungen wird angezeigt.

Testfall 14	
VB	• NB aus Testfall 1.
T	• Funktion im Ablaufplan verschieben. • *Öffnen*-Schaltfläche anklicken.
NB	• Der Hinweis auf ungespeicherte Änderungen wird angezeigt.

Testfall 15	
VB	• NB aus Testfall 1.
T	• Funktion im Ablaufplan verschieben. • *Verlassen*-Schaltfläche anklicken.
NB	• Der Hinweis auf ungespeicherte Änderungen wird angezeigt.

Testfall 16	
VB	• Es existiert eine kombinierte Funktion.
T	• Die kombinierte Funktion in das Ablaufplan ziehen. • Die kombinierte Funktion mit der Maus anklicken und es so markieren. • Im Konfigurationsbereich den Button mit Aufschrift *Detailgrad erhoehen* anklicken.
NB	• Das innere Ablaufplan wird angezeigt.

Testfall 17	
VB	• Es ist eine Automatik geladen, die mindestens ein komplexes Element enthält. • Es wird der Graph aus einem komplexen Element angezeigt.
T	• Abspeichern als *test2.aaf*.
NB	• Der abgespeicherte Graph ist der auf oberster Ebene.

Testfall 18 - AKTUELL NICHT ANWENDBAR	
VB	• Es befinden sich mindestens 2 Elemente A und B im Ablaufplan. • Funktion A ist für Kurven aktiviert. • Funktion B ist für Kurven und Gabelungen aktiviert.
T	• Beide Elemente markieren.
NB	• Im Eigenschaftenbereich wird für Kurve die Farbe für *aktiviert* und für Gabelung die Farbe für *gemischter Zustand* angezeigt. Die übrigen Listeneinträge sind als *inaktiviert* eingefärbt.

Testfall 19 - AKTUELL NICHT ANWENDBAR	
VB	• NB aus vorigem Testfall.
T	• In der Auswahlliste auf Gabelung klicken. • In der Auswahlliste auf Kurve klicken.
NB	• Für beide Funktionen wird Kurve *inaktiv* angezeigt. • Für beide Funktionen wird Gabelung *aktiv* angezeigt. • Diese Zustände sind in den zugehörigen Feldern auf `AAF`-Ebene korrekt abgebildet.

Testfall 20	
VB	–
T	• *Neu*-Schaltfläche anklicken. • Im Konfigurationsbereich rechts werden ein Name, eine Beschreibung und drei beliebige Kategorien vergeben. • Über die *Speichern*-Schaltfläche wird die kombinierte Funktion abgespeichert. • Es wird eine beliebige andere Automatik geöffnet. • Es wird die eben abgespeicherte Automatik geöffnet.
NB	• Name, Beschreibung und Kategorien sind wie vergeben vorhanden. • Im Konfigurationsbereich für kombinierte Funktionen wird die abgespeicherte Funktion unter dem vergebenem Namen angezeigt. • Im Konfigurationsbereich für kombinierte Funktionen wird die abgespeicherte Funktion mit einem roten, dicken Rand angezeigt. • Im Tooltip der Funktion im Funktionsbereich für kombinierte Funktionen ist die vergebene Beschreibung zu lesen, ebenso wie ein Hinweis, dass die kombinierte Funktion ungültig ist.

Testfall 21	
VB	• Es befindet sich eine elementare Funktion im Ablaufplan. • Der Funktionsbereich zeigt kombinierte Funktionen an.
T	• Die elementare Funktion in den Konfigurationsbereich ziehen.
NB	• Die elementare Funktion wird verworfen (gelöscht). • Der Ablaufplan ist leer. • Der Funktionsbereich hat die Anzeige zu elementaren Funktionen gewechselt.

Testfall 22	
VB	• Es befindet sich eine kombinierte Funktion im Ablaufplan. • Der Funktionsbereich zeigt elementare Funktionen an.
T	• Die kombinierte Funktion in den Konfigurationsbereich ziehen.
NB	• Die kombinierte Funktion wird verworfen (gelöscht). • Der Ablaufplan ist leer. • Der Funktionsbereich hat die Anzeige zu kombinierten Funktionen gewechselt.

Testfall 23	
VB	• Es befindet sich eine beliebige Funktion im Ablaufplan.
T	• Das Fenster wird vergrößert oder verkleinert.
NB	• Nach der Vergrößerung oder Verkleinerung hat sich die Position der Funktion relativ zum Ablaufplan nicht verändert.

Testfall 24	
VB	• Es existieren mindestens 3 elementare Funktionen die alle verschiedenen Kategorien zugeordnet sind.
T	• (a) Im Konfigurationsbereich für elementare Funktionen wird nach der Kategorie für Funktion 1 gefiltert. • (b) Im Konfigurationsbereich für elementare Funktionen wird nach der Kategorie für Funktion 2 gefiltert. • (c) Im Konfigurationsbereich für elementare Funktionen wird nach der Kategorie für Funktion 3 gefiltert. • (d) Im Konfigurationsbereich für elementare Funktionen wird nach der Kategorie '(Alle Funktionen)' gefiltert.
NB	• (a) Es wird u. a. Funktion 1 angezeigt, aber nicht Funktion 2 und 3. • (b) Es wird u. a. Funktion 2 angezeigt, aber nicht Funktion 1 und 3. • (c) Es wird u. a. Funktion 3 angezeigt, aber nicht Funktion 1 und 2. • (d) Es wird u. a. Funktion 1, 2 und 3 angezeigt.

D. Hinweise zur Implementierung

D.1. Ladezeit der Funktionsbereiche

D.1.1. Das Problem

Es wurde bemerkt, dass das Starten der Anwendung sowie das Speichern einer Datei ungewöhnlich lange dauert. Dieses Problem wird offensichtlicher, je mehr Funktionen existieren.

Der Grund hierfür liegt in den Funktionsbereichen. Wenn der Funktionsbereich für elementare Funktionen initialisiert wird, dann wird jeder Agent, der `readyForUse` mit `^true` implementiert, auch initialisiert. Einige Agenten müssen Daten von der Festplatte laden, sodass die Initialisierung bei diesen Agenten relativ lange dauert. Initialisierte Agenten sind aber nur notwendig, wenn auch ihre Konfigurationen verändert werden sollen. Die meisten Agenten werden also beim Start unnötig geladen. Ab und zu tritt auch der Fall ein, dass die Initialisierungsroutine eines Agenten fehlerhaft ist und ein Debug-Fenster verursacht. Das ist beim Start der Anwendung natürlich unschön, zu mal das nichts mit der Anwendung an sich, sondern den dahinterliegenden Agenten zu tun hat.

Beim Laden des Funktionsbereichs für kombinierte Funktionen wird das Problem gravierender: Das Laden einer kombinierten Funktion zieht die Initialisierung jedes Agenten in seinem Graphen nach sich. Je mehr kombinierte Funktionen existieren und je mehr Agenten diese enthalten, desto größer wird die Ladezeit.

Beim Speichern einer kombinierten Funktion wird der Funktionsbereich der kombinierten Funktionen vollständig neu geladen, um die Änderungen zu reflektieren. Hierbei entsteht für den Benutzer eine spürbare Wartezeit - die Anwendung wirkt träge. Das ist sehr ungünstig, da der Benutzer vermutlich durch diese Erfahrung in Zukunft weniger oft abspeichern wird.

D.1.2. Die Lösung

Das Problem kann gelöst werden, wenn die Agenten nur initialisiert werden, wenn sie tatsächlich gebraucht werden. Das ist dann der Fall, wenn der Benutzer eine Funktion bzw. den Agenten aus dem Funktionsbereich in den Ablaufplan zieht.

Aus zeitlichen Gründen wurde das Problem nur für den gravierenderen Funktionsbereich, nämlich den für kombinierte Funktionen, behoben.

Der Preis für den schnelleren Anwendungsstart und den Speichervorgang ist eine Verzögerung beim Drag & Drop. Klickt der Benutzer eine umfangreichere kombinierte Funktion im Funktionsbereich an (hier findet die Initialisierung statt), so dauert es etwas länger, bis sich die Funktion abhebt und an den Mauszeiger haftet.

Es folgen ein paar Hinweise zur Implementierung.

- In der Methode AAFNode delegate ist *Lazy Evaluation* umgesetzt: Der Delegate wird erst instantiiert, wenn er zum ersten mal referenziert wird.
- Dazu hat AAFBaseNode eine Instanzvariable delegateInit erhalten, die die Initialisierungsparameter des Agenten enthält. Diese Instanzvariable wird beim Laden des Graphen für jeden AAFNode befüllt.
- Die Methode AAFNode delegate sieht dementsprechend wie folgt aus:

```
delegate
    delegate isNil ifTrue: [
        delegate := (self delegateInit at: 'class') new.
        delegate setAllProps: (self delegateInit at: 'properties').
    ].

    ^ delegate.
```

D.1.3. Vergleich der gemessenen Ladezeiten

Die Messung der Ladezeit wurde durch folgende Zeile in einem Workspace via Strg+P (für Print[45]) drei mal durchgeführt:

```
Time millisecondsToRun: [ AAFGTComplexElementArea new ].
```

Wie erwähnt, sind die Ladezeiten abhängig von der Anzahl und des Umfangs der kombinierten Funktionen. Für die Nachvollziehbarkeit sei auf die nachfolgenden Versionen verwiesen. Diese haben die gleichen elementaren und kombinierten Funktionen, unterscheiden sich im Wesentlichen nur die Behebung der Ladezeit.

- Alte Version: commit c5045cec7004df7e779605351bcbb9f728da021e
  Ladezeiten (in ms): 1402, 1447, 1366
- Aktuelle Version: commit 3c528d490199e0d28f6905f3f453506c9d5a420c
  Ladezeiten (in ms): 169, 166, 167

Es konnte also mehr als eine Sekunde Ladezeit eingespart werden. Wie erwähnt, wird der Unterschied gravierender, je mehr kombinierte Funktionen hinzukommen werden.

D.2. Dokumentation der Klassen der Bedienelemente

Fuhrmann hat in ihrer Arbeit angeregt[46] einheitliche Bedienelemente zu implementieren, alternativ auch andere GUI-Frameworks auszuwerten, wie z. B. wxSqueak[47] oder SqueakGtk[48]. Das Framework sollte kompatibel mit dem bisherigen Quellcode sein.

[45] Das „Printen“ einer Zeile im Workspace führt zur Ausgabe des Rückgabewertes.
[46] vgl. [6], Abschnitt 5.2.9, S. 84
[47] vgl. http://www.wxsqueak.org/
[48] vgl. http://squeakgtk.pbworks.com/

Denkbar wäre es zum Beispiel nur die eigentlichen Bedienelemente dieser Frameworks in den Konfigurationsdialogen der Funktionen zu benutzen. An dieser Stelle scheidet SqueakGtk als GUI-Framework allerdings schon aus (siehe FAQ auf der Webseite). wxWidgets ist problematisch, weil es noch über einige kritische „Known Issues“ verfügt, darunter Speicherlecks.

Abgesehen davon sprechen auch andere Gründe gegen die Verwendung von wxSqueak und SqueakGtk. Keine der beiden Projekte hat bisher die vollständige API der offiziellen GUI-Frameworks implementiert. Weil sich wxWidgets und Gtk+ auch ständig weiterentwickeln, werden wxSqueak und SqueakGtk immer hinterherhängen. Weiterhin sei angemerkt, dass Fuhrmann schon angefangen hat einige Bedienelemente selbst zu implementieren.

Aus diesen Gründen wurde eine eigene kleine Bibliothek von Bedienelementen in den Kategorien AAFGT-Widgets und AAFGT-Widgets-Support implementiert. AAFGT-Widgets-Support enthält Hilfsklassen. Alle Klassen beginnen mit dem Prefix „AAFGT“. Die AAFGT Widget Gallery (Abbildung 10) zeigt alle Bedienelemente in Verwendung.

Im Folgenden sind die einzelnen Klassen, auf die die Entwickler zurückgreifen können, dokumentiert. Sollte die Dokumentation unzureichend sein, empfiehlt es sich die AAFGT Widget Gallery zu studieren - Klasse AAFGTWidgetGallery.

D.2.1. Klasse AAFGTOneLineLabel

Diese Klasse implementiert einen einfachen Bezeichner für eine Zeile.

Instanziierung:

```
contents: aString
    "Create an one line label with the contents aString."
```

D.2.2. Klasse AAFGTMultiLineLabel

Diese Klasse implementiert einen einfachen Bezeichner der über mehrere Zeilen gehen kann.

Instanziierung:

```
contents: aString
    "Create a multi line label with the contents aString."
```

D.2.3. Klasse AAFGTMultiLineTextEdit

Diese Klasse implementiert einen Texteditor der mit mehrzeiligem Text umgehen kann. Bei der Instanziierung kann via target/action ein Callback ermöglicht werden (zweite Instanziierungsmöglichkeit).

Instanziierung:

```
contents: aString
    "Create a multi line text editor with the default
     contents aString."


contents: aString target: aTarget action: anAction
    "Create a multi line text editor with the default
     contents aString.
     If the user changes the text, execute anAction on
     aTarget."
```

D.2.4. Klasse AAFGTButton

Diese Klasse implementiert einen Button mit einem Bezeichner. Bei der Instanziierung wird via target/action ein Callback ermöglicht.

Instanziierung:

```
label: aLabel target: aTarget action: anAction
    "Create a button with label aLabel.
     On click execute anAction on aTarget."
```

D.2.5. Klasse AAFGTSpinButton

Diese Klasse implementiert einen editierbaren SpinButton. Bei der Instanziierung kann via target/action ein Callback ermöglicht (zweite Instanziierungsmöglichkeit) werden. Als Tooltip wird der Wertebereich angezeigt.

Instanziierung:

```
value: aValue range: aRange climbRate: aClimbRate
    "Create a spin box with default value aValue,
     a range of aRange and climbRate of aClimbRate.

     aRand is an Array with the smallest and biggest
     number allowed."


value: aValue range: aRange climbRate: aClimbRate
  target: aTarget action: anAction
    "Create a spin box with default value aValue,
     a range of aRange and climbRate of aClimbRate.

     aRand is an Array with the smallest and biggest
     number allowed.
```

```
    Execute anAction on aTarget if user modifies the
    value.

    anAction must take one argument - the new value."
```

D.2.6. Klasse AAFGTCheckBox

Diese Klasse implementiert eine CheckBox mit einem Bezeichner. Bei der Instanziierung wird via target/action ein Callback ermöglicht.

Instanziierung:

```
label: aLabel target: aTarget action: anAction
    "Create a a check box with a label aLabel.
     On click execute anAction on aTarget.

     anAction must take one argument - the
     boolean, wheter the check box is checked
     (true) or not (false)."
```

D.2.7. Klasse AAFGTDropDownMenu

Diese Klasse implementiert ein Dropdown-Menü. Bei der Instanziierung wird via target/action ein Callback ermöglicht. Als Tooltip werden die möglichen Alternativen angezeigt.

Instanziierung:

```
value: defaultValue items: items target: aTarget action: anAction
    "Create a drop down menu with a default
     selection defaultValue. All menu items
     are provided as a SequenceableCollection
     of strings in parameter items. On click
     execute anAction on aTarget.

     anAction must take one argument - the new
     value (selected from user). "
```

D.2.8. Klasse AAFGTPane

Diese Klasse implementiert einen Container ähnlich zu AAFGTSection. AAFGTPane ist als großangelegter Container gedacht um Bereiche einer Anwendung zu beheimaten. Ein AAFGTPane kann auch mehrere Inhalte verwalten (siehe zweite Instanziierunsmethode).

Instanziierung:

```
label: aLabel contents: aWidget
    "Create a pane container with aWidget as
     contents and label aLabel."


panes: panesDict activePane: activePaneLabel
    "Create a switchable pane container with
     given panes and the current active pane label
     activePaneLabel.

     panesDict is a Dictionary: keys are the pane
     labels, values the widgets"
```

D.2.9. Klasse AAFGTSection

Diese Klasse implementiert einen umrahmten Container. Der Kopfbereich ist ein- und ausklappbar.

Instanziierung:

```
label: aLabel
    "Create a section container with label
     aLabel and without contents.

     Widgets can be added later via the area getter"


label: aLabel contents: aWidget
    "Create a section container with aWidget as
     contents and label aLabel."
```

D.2.10. Klasse AAFGTGroupBox

Diese Klasse implementiert einen einfachen Container für andere Bedienelemente, indessen Kopfbereich die Überschrift fett gedruckt und unterstrichen wird.

Instanziierung:

```
label: aLabel
    "Create a group box container with label
     aLabel and without contents.

     Widgets can be added later via the area getter"


label: aLabel widget: aWidget
```

```
    "Create a group box container with label
     aLabel and widget aWidget as contents."


label: aLabel widgets: widgets
    "Create a group box container with label
     aLabel and the widgets in parameter widgets
     (SequenceableCollection) as contents."
```

D.2.11. Klasse AAFGTWidgetUtils

Diese Klasse stellt nützliche Hilfsfunktionen zur Verfügung. Unter anderem existieren Klassenmethoden zum

- Ausrichten von Morphen bzw. Widgets in anderen Morphen (Kategorie *alignments*)
- Anpassen von Schriftgröße, Fett- und Kursivdruck und Unterstreichungen (Kategorie *fonts*)
- Anzeigen von Bezeichnern vor oder über anderen Widgets (Kategorie *prepending labels*)
- Anzeigen von vertikalen oder horizontalen Trennstrichen in beliebiger Farbe (Kategorie *separators*)
- Anzeigen von leeren Zwischenraum (Kategorie *spacer*).

D.2.12. Klasse AAFGTWidgetGallery

Diese Klasse stellt eine Galerie aller Bedienelemente bereit. Mit

```
AAFGTWidgetGallery open.
```

wird die Galerie (s. Abbildung 10) angezeigt.

Erklärungen

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Berlin, 19. September 2011