ionesoft

comem+

Accessibility für digitale Lehrmittel

Barrierefreiheit erhöhen in eclipse RCP-Applikationen

Travail de Bachelor 2015

Berne, 30 juillet 2015

Christian Heimann
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud - HEIG-VD
Département comem+
Filières Ingénierie des Médias
Classe MIT41

Répondant interne
Jean-Pierre Hess

Avenue des Sports 20
CH-1400 Yverdon-les-Bains
www.comem.ch

Répondant externe
Daniel Stainhauser
Chef d'entreprise
ionesoft GmbH

Hes·so

heig-vd
HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD
www.heig-vd.ch

Management Summary

Die Vision «*beook* komplett barrierefrei machen» ist gesetzt. Die Plattform für digitale Lehrmittel *beook* soll für blinde und visuell eingeschränkte Personen zugänglich werden.

Die Windows-Version von *beook* basiert auf *eclipse* RCP, einem vielseitigen und flexiblen Framework zum Erstellen von Desktop-Applikationen. Accessibility wird damit zu einem Teil abgedeckt, da es auf Betriebssystem-eigenen Benutzeroberflächen-Komponenten basiert. Jedoch laufen viele vorhandene Accessibility-Elemente ins Leere, da sie nicht von Screenreadern interpretiert werden. Die richtigen Ausgabemöglichkeiten anzusprechen ist daher wichtig.

Um Erfahrungen zu sammeln wurde eine Test-Applikation erstellt. So konnte in einer übersichtlichen Applikation mit kleinem Umfang die essenziellen Techniken überprüft und die Finessen des Benutzeroberflächen-Aufbaus entdeckt werden.

Mit den gewonnen Erkenntnissen konnte die Barrierefreiheit betreffend Sprachausgabe und Tastaturnavigation verbessert werden. Dafür wurde ein wiederverwendbares Modul entwickelt, das in alle RCP-Applikationen implementiert werden kann. Dieses *Accessible Modul* bringt die Möglichkeit mit, über eine XML-Datei die Screenreader-Texte wie auch die Tastaturnavigations-Abfolge zu definieren. Damit lassen sich diese Elemente zentral verwalten.

Die Integration des *Accessible Moduls* in *beook* führte zu den Erkenntnissen, dass die Dynamik und Vielfältigkeit von RCP im Modul noch nicht genügend abgebildet war. Deshalb wurden in einem letzten Schritt eine Erweiterungsmöglichkeit und die Unterstützung von Template-Struktur-Beschreibungen eingebaut, um das Modul auf eine stabilere Basis für die Zukunft zu stellen. Nötige, tiefgreifendere Anpassungen der grafischen Benutzeroberfläche von *beook* sehen noch an, um der Vision einen guten Schritt näher zu kommen.

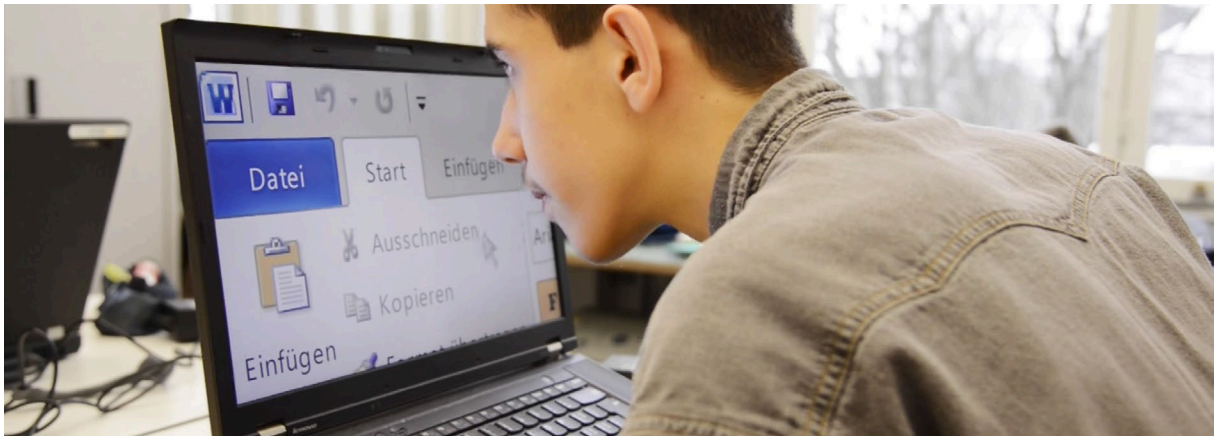
Ein Schwergewicht bei der Entwicklung wurde auf Wiederverwendbarkeit gelegt und dadurch weniger Zeit in die effektive Integration investiert. Die RCP- und Accessibility-Community soll von der ausführliche Dokumentation und der Bereitstellung des Quellcodes des *Accessible Moduls* profitieren. So bleibt zu hoffen, dass eine verbesserte Barrierefreiheit bald zum Standard von RCP-Applikationen gehört.

heig-vd
HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Travail de Bachelor 2015

Accessibility für digitale Lehrmittel

Barrierefreiheit erhöhen in eclipse RCP-Applikationen



Eine bestehende Desktop-Applikation barrierefrei zu machen verlangt den Spagat zwischen den Ansprüchen an eine optimale User-Experience und dem Zusammenspiel bestehender, eingesetzter Technologien. Nebst einer Navigation, die komplett über die Tastatur möglich sein muss, ist die «Lesbarkeit» der Applikation für Screenreader zentral.

Phase «Proof of Concept» – Open Source Beispiel-Applikation

In einem spärlich dokumentierten Bereich, der Implementation von Accessibility Features in eine RCP-Applikation, galt es zuerst Bestehendes auszuprobieren und Mögliches auszuloten. Entstanden ist eine Beispiel-Anwendung, deren Code öffentlich zugänglich ist und als Anregung für weitere barrierefreie Applikationen dienen soll.

Eine Hauptkomponente ist für die reibungslose und effiziente Steuerung via Tastatur zuständig. Diese ist so konzipiert, dass sie auch in andere Projekte integriert werden kann. So kann die grundlegende Tastaturnavigation unabhängig von Applikations-spezifischen Dateien gehalten werden.

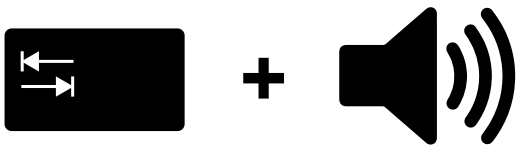
Der Screenreader – JAWS – soll in gewünschten Momenten Informationen mitteilen. Diese Momente bzw. Stellen müssen gesetzt und möglichst clever benannt werden, um für den Benutzer eine eindeutige, unverwechselbare Landmarks, also Orientierungspunkte, darzustellen.

Als Konzept hinter den umgesetzten Massnahmen diente die Bachelor-Vorarbeit «Blind am Computer».

Phase «Integration» – beook wird zugänglicher

In der RCP-Applikation «beook», einer Plattform für digitale Lehrmittel der ionesoft GmbH, konnte durch die gewonnen Erkenntnisse die Barrierefreiheit erhöht werden. Pflichtelemente wie Tooltips und Labels wurden konsequent gesetzt. Zusätzlich wurde als erste Priorität die Tastaturnavigation wie im Proof of Concept umgesetzt.

Die Arbeit zeigt, durch welche Möglichkeiten und Implementationsschritte man die Barrierefreiheit von eclipse RCP-Applikationen verbessern kann. Diese sind der Grundstein, um bestehende Applikationen blinden und visuell eingeschränkten Personen zugänglich zu machen. Dank einer durchdachten Tastatursteuerung profitieren aber auch alle Power-User von dieser Erweiterung.



Auteur : **Christian Heimann**
Répondant externe : **Daniel Stainhauser**
Prof. responsable : **Jean-Pierre Hess**
Sujet proposé par : **ionesoft GmbH**

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences
Western Switzerland

heig-vd
HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

comem+

Danksagung an …

Daniel Stainhauser, für die fortwährende Unterstützung, den regen Wissensaustausch und -transfer und dem ganzen Team von *ionesoft* für einen angenehm intensiven Sommer im Marzili.

Jean-Pierre Hess, für die vielen guten Inputs, die motivierenden Feedbacks und das echte Interesse, welches er für diese Arbeit aufgebracht hat.

Selamet Aydogdu, für den Einblick in seine Welt, einer Live-Coding-Demo, und der Bereitschaft fürs Testing, zu dem es noch nicht kam.

Ouli-Minna Elgorriaga und all ihre Schüler, die mir einen Einblick in den Schul-Alltag von Blinden und Sehbehinderten ermöglicht haben. Das zuerst während einem kompletten Tag an der Schule für Sehbehinderte in Zürich, und dann während einem Vormittag am Gymnasium in Wetzikon mit der Gymnasiastin Laura.

Luc Fontolliet, der mit seinen Erfahrungen und Kenntnissen im Bereich «Software für Blinde» einen der Grundsteine für diese Arbeit legte, Mouskie demonstrierte und und den Kontakt mit Jean-Marc Meyrat herstellte.

Alexandre Hauser, für die Auskunft zur Herstellung der Mouskie-Software.

Jean-Marc Meyrat, der während zwei Lektionen das Basiswissen über Accessibility in der IT teilte und bei einem anschliessenden Mittagessen mit seinem Lebenslauf wie auch mit seinen aktuellen Projekten beeindruckte.

Robert W. Kingett, für einen regen Informationsaustausch, der via Youtube und Facebook angefangen hat, und seine fordernde Haltung. Auch scheute er sich keiner kritischen Rückfragen.

Bryan Obright von Oracle, für Tipps im Bereich Tooling und API.

Anton Bolfing von *Access for All*, für das Publizieren meiner Dokumente auf dem Blog von access4all.ch/blog.

Urs Hiltebrand von Accesstech, für eine reichhaltige Auskunft zu Screenreadern und Navigation direkt ab der Quelle.

Michael Schall von *Freedom Scientific*, für das Zurverfügungstellen einer 90-Tage-Lizenz des Screenreader *JAWS*.

Saskia Faulk, Françoise Châtelain, Yves Germanier, und alle anderen Lehrkräften, die uns Absolventen bis an diesen Punkt hin begleitet haben.

Und meine zukünftige Ehefrau Céline, die während der Intensiv-Phase dieser Bachelor-Arbeit Entbehrungen ertragen musste, jederzeit für mich da war und mich trotzdem ab und zu von der Arbeit weglocken konnte, um gemeinsam ein paar schöne Stunden am Wasser zu verbringen.

Vorwort

Diese Bachelor-Arbeit wurde im Rahmen des Studiums zum Medieningenieur an der *HEIG-VD*, Departement comem+, verfasst. Die Dauer der Arbeit war auf 8 Wochen beschränkt und lief von Anfang Juni bis Ende Juli 2015.

Methodik

Vorgabe

Als konzeptionelle Vorgabe für die Umsetzung diente der Empfehlungskatalog der Bachelor-Vorarbeit «Blind am Computer»[1]. Dieser gab eine klare, theoretische Stossrichtung vor.

Technische Erarbeitung

Um sich in der *eclipse-RCP*-Umgebung zurechtzufinden, dienten vor allem Lars Vogels Tutorials[2], welche enorm viele Aspekte zur Erstellung von *RCP*-Applikationen abdecken. Für einfache, klare Code-Beispiele war die Website von Java2s[3] hilfreich.

Da auf die spezifische Problemstellung (*eclipse RCP*-Applikation barrierefrei machen) passende Dokumentationen fehlen, war viel Trial&Error nötig. Ebenso war von *Freedom Scientific*, dem Hersteller des Screenreaders *JAWS*, keine Entwickler-Dokumentation erhältlich, welche die Funktionsweise mit den Angriffspunkten beschreiben würde.

Technische Analyse

Um die Software-Vorgänge und deren Element-Struktur zu analysieren kamen folgende Hilfsprogramme zum Einsatz:

- **AccEvent**: Accessible Event Watcher des Windows *SDK*
- **Inspect**: Inspektor der *MSAA*-Hierarchie des Windows *SDK*
- **aDesigner**: auf *eclipse* basierter «Accessibility Checker»
- **eclipse IDE**: Debugging

Anmerkungen

Gender

Werden Personenbezeichnungen aus Gründen der besseren Lesbarkeit lediglich in der männlichen oder weiblichen Form verwendet, so schliesst dies das jeweils andere Geschlecht mit ein.

Begriff «Zielgruppe»

Um die Wiederholung des Begriffs «blinde und visuell eingeschränkte Personen» zu vermeiden, wird in dieser Arbeit von «Zielgruppe» gesprochen. Diese Zielgruppe besteht nicht nur aus Personen mit Geburtsgebrechen, sondern auch all jenen – und das ist die Mehrheit – welche eine Sehbehinde-

1 Christian Heimann – Blind am Computer: Bachelor-Vorarbeit, Seite 23 ff.
2 Lars Vogel – Free Tutorials to program Eclipse RCP and Plug-ins – www.vogella.com > Tutorials > Eclipse RCP/Plug-ins
3 SWT JFace Eclipse « Java – www.java2s.com > Java > SWT JFace Eclipse

rung erst im Laufe der Zeit entwickeln oder deren Sehkraft mit zunehmendem Alter nachlässt.

Begriffe «Modul» und «Applikation»

Um zwischen den beschriebenen Softwarekomponenten gut unterscheiden zu können, wird von Modul und Applikation gesprochen. Das *Accessible Modul* beschreibt ein *eclipse*-RCP-Plugin-Projekt, welches an jede RCP-Applikation angeschlossen werden kann. Mit Applikation wird von einem «*eclipse* 4 Application Project» gesprochen. Das hat im Kern den gleichen Aufbau wie ein Plugin-Projekt, es besitzt aber zusätzlich noch die nötigen Elemente, um die Applikation konfigurieren und direkt starten zu können.

Wenn von «leerem» *Proof of Concept* gesprochen wird, ist damit die RCP-Applikation ohne jegliche zusätzlich gesetzten Accessibility-Features und ohne eingefügtes *Accessible Modul* gemeint.

Begriff «Label»

Wenn es um die Sprachausgabe von Screenreader geht und dabei von «Label» gesprochen wird, bezeichnet dies die Textattribute, welche vom Screenreader beim jeweiligen Element vorgelesen werden. Dies sind nicht immer Labels, sondern manchmal auch gesetzte Texte auf einem Widget, Tooltips oder durch AccessibleListener gemeldete Strings.

Diagramme

Für die Veranschaulichung von Zusammenhängen wurden zwei verschiedene Typen von Diagrammen in dieser Arbeit integriert. Die farbig gehaltenen Diagramme zeigen konzeptionelle Zusammenhänge auf, die in Schwarz-Weiss gesehaltenen sind nach UML-Standard gesetzte Design- und Sequenz-Diagramme. Um sich mit den UML-Diagrammen einen schnellen Überblick verschaffen zu können, wurden diese an einer Stelle gebündelt.

Empfohlene Lektüre

Um die gegangenen Wege während dieser Arbeit noch besser zu verstehen, wird eine vorangehende Lektüre der Bachelor-Vorarbeit «Blind am Computer»[4] empfohlen.

4 Christian Heimann – Blind am Computer: Bachelor-Vorarbeit

Inhaltsverzeichnis

Einleitung

Rahmen

Rechtliche Vorgaben

Seit dem Beitritt der Schweiz zur UN-Behindertenrechtskonvention am 15. April 2014[1] steht die Bildungslandschaft definitiv in der Pflicht: Darin wird im Artikel 24 als zentraler Punkt «die Möglichkeit der gemeinsamen Beschulung behinderter und nicht behinderter Kinder in allgemeinbildenden Schulen und Besuch von Universitäten»[2] festgehalten. Inklusive Bildung wird gefordert, das heisst, behinderte und nicht behinderte Schüler gemeinsam zu Unterrichten und sie zum Beispiel auch nach Möglichkeit mit den gleichen Lehrmitteln zu versorgen.

EPUB 3 als zukünftiger Standard

Damit kommt auch die Verlags- und Lehrmittel-Industrie etwas unter Druck: Die Machbarkeitsstudie «Barrierefreie elektronische Lehrmittel im *EPUB* 3» von *Access for All*, welche im Oktober 2014 publiziert wurde, geht stark davon aus, dass sich *EPUB* 3 unter anderem dank seiner Semantik als Standard für digitale Bücher und Lehrmittel durchsetzen wird.

Doch ein gut strukturierter Inhalt genügt zur barrierefreien Verwendung nicht: Die Anwendung, welche *EPUB*-Dokumente anzeigt, also der Container um den eigentlichen Inhalt herum, muss seine Funktionen ebenfalls zugänglich machen. Ein solcher «Container» ist *beook*, die Plattform für digitale Lehrmittel der *ionesoft*.

Zielgruppe: blinde und visuell eingeschränkte Personen

Die in dieser Hinsicht «erste» Zielgruppe für eine barrierefreie Anwendung sind blinde und visuell eingeschränkte Personen. Diese arbeiten zu einer Mehrheit mit Windows-Laptops und den zusätzlichen, assistiven Technologien «Screenreader» und «Braillezeile».[3]

ionesoft und beook

Die *ionesoft GmbH* hat, mit einer zu grossen Teilen zugänglichen iOS-Version ihrer Applikation *beook*, bereits einen Schritt gemacht, eine möglichst barrierefreie Plattform für digitale Lehrmittel zu erstellen. Da eine Vielzahl von blinden und visuell eingeschränkten Personen als Arbeitsinstrument Laptops mit Windows als Betriebssystem benutzen, ist nun die nächste Bestrebung, die Windows Version von *beook* zugänglich zu machen. Die *beook*-Version für Windows ist mit *eclipse RCP* aufgebaut.

Zeitgleich ist die *ionesoft GmbH* als Projektpartner in einem Projekt zur Überprüfung von «Barrierefreiheit mit *EPUB*» von *Access for All* involviert. Diese Bachelor-Arbeit stellt also auch einen Beitrag an dieses Projekt dar.

---

1 Convention on the Rights of Persons with Disabilities – United Nations Treaty Collection – treaties.un.org > Databases > Status of Treaties > Chapter 4 > 15

2 Übereinkommen über die Rechte von Menschen mit Behinderungen – Wikipedia

3 Christian Heimann – Blind am Computer: Bachelor-Vorarbeit, Seite 7

eclipse-Community

Dokumentationen und Beispiele im Bereich «Accessibility» für *eclipse* RCP Applikationen sind zum jetzigen Zeitpunkt rar. Die Ergebnisse dieser Arbeit sollen deshalb auch öffentlich bereit gestellt werden. Besonders im deutschsprachigen Raum wird die *eclipse*-Plattform von einer breiten Basis getragen und weiterentwickelt.[4] Daher ist die mögliche Reichweite trotz der sprachlichen Barriere dieser in Deutsch verfassten Arbeit recht gross.

Problemstellung

Durch die vorangehende Analyse, die Vorarbeit zu dieser Bachelor-Arbeit mit dem Titel «Blind am Computer», konnte aufgezeigt werden, wie die Applikation *beook* optimal barrierefrei werden könnte. Als zwei Hauptelemente hatten sich die Sprachausgabe mit dem Screenreader JAWS und ein klares Navigationskonzept herauskristallisiert. Vision war es, diese Empfehlungen möglichst weit umzusetzen.

Die *beook*-Applikation für Windows basiert auf dem Framework *eclipse* RCP. Um sich in der *eclipse*-RCP-Umgebung zurechtzufinden, wurde der Weg über ein *Proof of Concept* gewählt. Damit sollte geklärt werden, mit welchen technischen Mitteln eine RCP-Applikation möglichst einfach «barrierefrei» gemacht werden kann. Die Eingriffe in die bestehende Applikation sollen zudem möglichst gering gehalten werden.

Die geforderte Implementation in der *beook*-Applikation soll zeigen, dass die entwickelten Methoden in einer echten Umgebung ebenfalls eingesetzt werden können und *beook* dem Ziel «komplett barrierefrei» etwas näher rückt.

Ziele dieser Arbeit

Als Vision und Triebfeder dieser Bachelor-Arbeit wurde die vollständige Barrierefreiheit der RCP-Applikation «*beook*» gesetzt. Möglichst nah dran zu kommen, mit möglichst universell einsetzbaren Lösungen war das globale Ziel. In konkreten Etappen ausgeführt hiess dies:

- Erstellen einer barrierefreie Beispielanwendung mit GUI- und Steuerungs-Elementen, die auch in der *beook*-Applikation enthalten sind.
- Veröffentlichung von Erkenntnissen und des Quellcodes der Beispielanwendung, um in diesem Nischenbereich einen Beitrag zu leisten.
- Die RCP-Applikation «*beook*» für den Screenreader JAWS lesbar machen und eine barrierefreie Navigation ermöglichen.

4 EclipseCon: Eclipse Community Award mit vielen deutschen Gewinnern – www.heise.de > heise Developer > News > 2012 > KW 13

1. Entwickeln mit eclipse RCP

1.1. Plattformunabhängige Entwicklungsumgebung

beook ist eine Applikation zum Lesen und Bearbeiten von digitalen Lehrmitteln. Diese Lehrmittel sind digitale Bücher, deren Funktionalität dank *beook* weit über ein einfaches «Lesen» hinaus geht. iOS war die erste Plattform, für welche *beook* entwickelt wurde. Als auf Januar 2014 die Entwicklung einer Version für Windows und Mac gewünscht wurde, hat sich *ionesoft* entschieden, diese auf dem Java-basierten *RCP*-Framework aufzubauen.

RCP ist ein Bestandteil von *eclipse*, einer plattformunabhängigen Entwicklungsumgebung von *IBM*. Der Quellcode von *eclipse* wurde im Jahr 2001 freigegeben. Seit 2004 ist die rechtlich eigenständige Eclipse Foundation für die Entwicklung von *eclipse* zuständig.[5] Die Entwicklungsumgebung *eclipse* selbst basiert ebenfalls auf *RCP*.

Der Vorteil von *RCP* ist, dass es standardmässig die nötigen strukturellen Elemente für eine komplette Desktop-Applikation mitbringt. Dadurch kann man die programmatische Erstellung von Fenstern, Unterfenstern und deren Unterteilungen *RCP* überlassen, und die Definition ebendieser über eine *GUI*-Oberfläche setzen.

1.2. Einführung in RCP-Applikationen

1.2.1. Beispiele von GUI-Elementen

Um von Beginn weg mit der *RCP*-eigenen Begrifflichkeit besser klar zu kommen, sind hier einige Beispiele von *GUI*-Elementen, wie sie in *RCP*-Applikationen vorkommen, abgebildet und gelb gekennzeichnet.

Perspektive

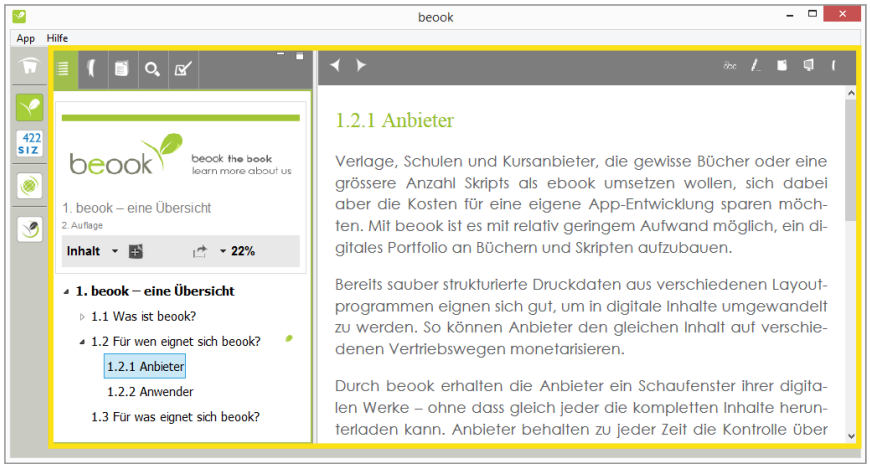


fig. 1 – *GUI-Element Perspektive*

5 Eclipse (IDE) – Wikipedia

Eine Perspektive *(fig. 1)* ist eine «Komplettansicht» einer Applikation(). Mehrere aktive Perspektiven im gleichen Programmfenster sind nicht möglich. Die Perspektiven können aber hintereinander gelegt werden. Die Menü-Leiste und die links liegende Toolbar sind global und gehören nicht zu einer Perspektive.

Part und View

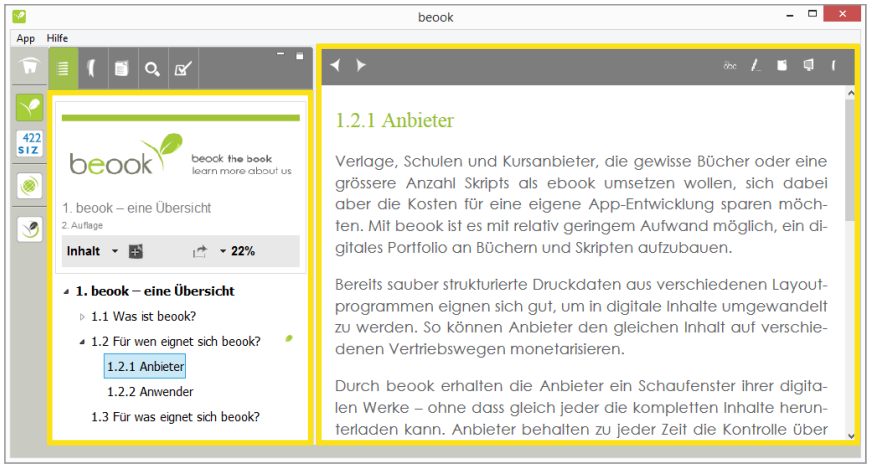


fig. 2 – *GUI-Element Part, dessen Inhalt von einer View definiert wird*

Parts sind die Unterbestandteile von Perspektiven *(fig. 2)*. Ein Part wird Ausgefüllt von einer View. Diese definiert, was im Part angezeigt wird.

Composite

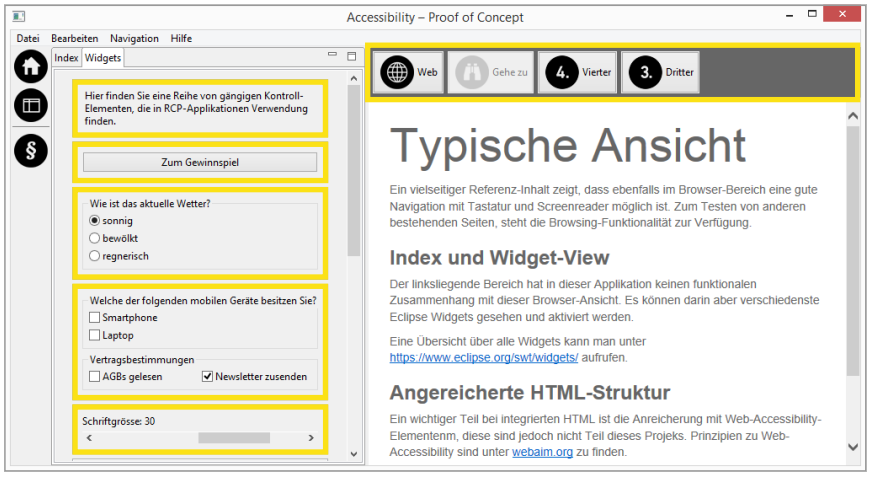


fig. 3 – *GUI-Element Composite*

Composite ist ein Element zur Strukturierung, welches transparent angewendet werden kann *(fig. 3)*. Eine Verschachtelung ist möglich, da ein Composite sowohl Widgets wie auch andere Composites beinhalten kann. Das umfassendste Element innerhalb einer View ist ebenfalls ein Composite.

Widget

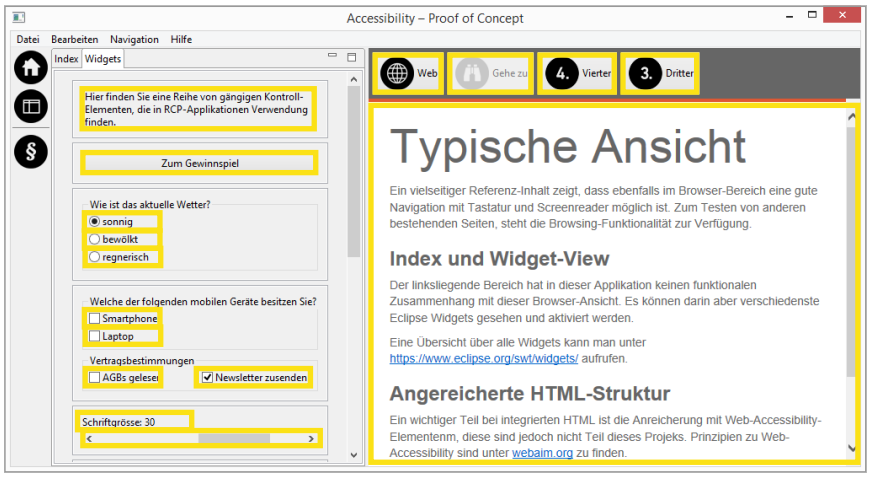


fig. 4 – *GUI-Element Widget*

Steuer-, Aktions-, Label-, Text- und Browser-Elemente *(fig. 4)* werden Widgets oder auch Controls genannt.

1.2.2. Zusammenhänge der Haupt-GUI-Elemente

Um sich eine noch klarere Vorstellung machen zu können, wann von Perspektiven, Parts, Widgets und anderen *eclipse*- und *SWT*-Objekten die Rede ist, werden im folgenden Schema *(fig. 5)* die Zusammenhänge und Verbindungen dargestellt.

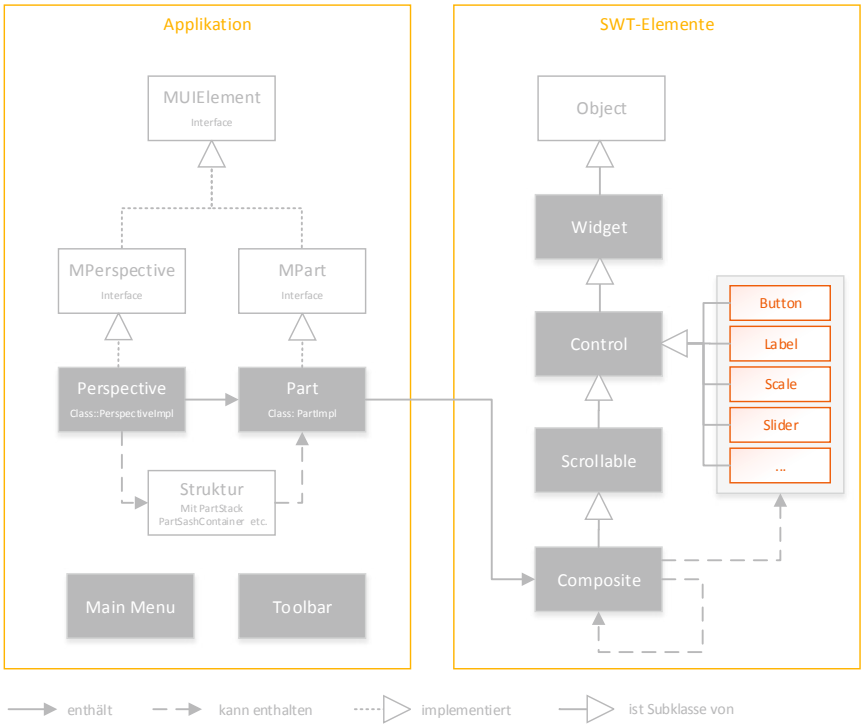


fig. 5 – *Schematische Übersicht der Haupt-GUI-Elemente von RCP-Applikationen*

Die Darstellung *(fig. 5)* hat keinen Anspruch auf Vollständigkeit, sondern soll einen Überblick über die für diese Arbeit wichtigen Elemente geben.

Die *GUI*-Bestandteile können klar in zwei Ebenen geteilt werden: Applikation und *SWT*. Mit den Elementen der Ebene «Applikation» wird die Struktur der

Applikation definiert. Diese setzt sich in der Regel aus einer oder mehreren Perspektiven zusammen. Jede Perspektive kann einen oder mehrere Parts enthalten, diese können zusätzlich durch Struktur-Elemente wie PartStack und PartSashContainer verschachtelt sein. Jedes Part-Element ist ein «Blatt» im Baum der Applikations-Struktur.

Von Perspektiven unabhängige Elemente wie das Hauptmenü (Menüleiste) und Toolbars (deren übergeordnetes Element Trimbar heisst), gehören ebenfalls zur Applikations-Struktur.

Die *SWT*-Ebene enthält die konkreten, in Programmfenster- und Programmteilen enthaltenen Unterbereiche, Schaltflächen, Labels und weiteren *GUI*-Elemente. Dabei stellt Composite das Control dar, welches selbst wieder andere Controls – sich selbst miteingeschlossen – enthalten kann. Die für diese Arbeit wichtigsten Widgets fallen unter die Klasse «Control», dies ist aber nicht für alle Widgets der Fall.

Die Verbindung zwischen der Applikations- und der *SWT*-Ebene ergibt sich durch das Setzen einer Klassen-URI im Part. Diese bestimmt, welche Programm-Klasse den Part erzeugen wird. In der Regel wird eine «View Part»-Klasse gesetzt, welche den Code für die Befüllung des Parts liefert. Nach der Part-Erstellung wird die in der gesetzten Klasse mit @PostConstruct annotierte Methode «createComposite» aufgerufen. Darin wird der programmierte *GUI*-Aufbau gemacht.

@PostConstruct-Annotation für createComposite-Methode
Auszug aus: ch.chrissharkman.accessibility.rcp.application.poc.parts.BrowserViewBuilder.java

```
/**
 * Function to create the complete composite of the part.
 * Argument composite parent comes from framework, so it is the Part
 * which loads this classURI in application.e4xmi
 * @param parent
 */
@PostConstruct
public void createComposite(Composite parent) {
     // keep the parent composite for AccessibleView implementation
     this.viewComposite = parent;
     // Layout of the composite element
     GridLayout gridLayout = new GridLayout();
     gridLayout.numColumns = 1;
     gridLayout.marginWidth = 0;
     gridLayout.marginHeight = 0;
     gridLayout.verticalSpacing = 0;
     parent.setLayout(gridLayout);
     GridData gridData = new GridData();
     gridData.grabExcessHorizontalSpace = true;
     parent.setLayoutData(gridData);

     // add components
     createToolbar(parent);
     this.browser = BrowserManager.instance().createBrowser(parent);

     // display content
     BrowserManager.instance().display(this.browser, this.actualContent);
}
```

1.2.3. Technische Strukturübersicht

Wie bereits erwähnt ist *RCP* ist ein Framework zur vereinfachten Herstellung von Java-Desktop-Applikationen – daher auch der Name «Rich Client Plat-

form». In *eclipse* wird mit Plugin-Projekten gearbeitet, die sich einfach in weitere Plugin-Projekte einbinden lassen.

Ein e4-Application-Project – so heisst der Typ einer kompletten Desktop-Applikation – kann durch einen Wizard angelegt werden. Es ist ein erweitertes Plugin-Projekt und enthält folgende Haupt-Dateien:

- **Application.e4xmi**: Strukturdatei der RCP-Applikation. Darin können Perspektiven, Parts, deren Verschachtelung und Inhaltsklassen gesetzt, Toolbars definiert und zu ladende Add-ons[6] hinzugefügt werden. Auch Commands, Handler, Keybindings und Binding Contexts können so direkt konfiguriert werden.
- **bundlename.product**: Launch-Konfiguration der Applikation, Auflistung der Plugins, welche das Produkt vervollständigen.
- **plugin.xml**: Darin können Extension-Points definiert werden, sodenn diese nicht über das Applicaiton.e4xmi bereits definiert sind.
- **manifest.mf**: Definition von benötigten Bundles, importierten und exportierten Packages und dem Classpath für die zu kompilierenden Elemente.

fig. 6 – *Demonstration zum einfachen Erstellen einer RCP-Applikations-Basis unter: tb.chrissharkman.ch/einfach*

Durch das korrekte Anlegen dieser Dateien kann eine e4-Application von Beginn an gestartet werden. Dass man wirklich schnell eine RCP-Applikation zum Laufen bringt zeigt das Video *(fig. 6)*.

In der *eclipse*-Umgebung können diese Application-Project-Dateien bequem über eine grafische Benutzeroberfläche strukturiert werden. So lässt sich zum Beispiel der komplette Applikations-Struktur-Baum anzeigen *(fig. 7)*. Die GUI-Elemente sind bis maximal auf Niveau «Part» zu sehen.

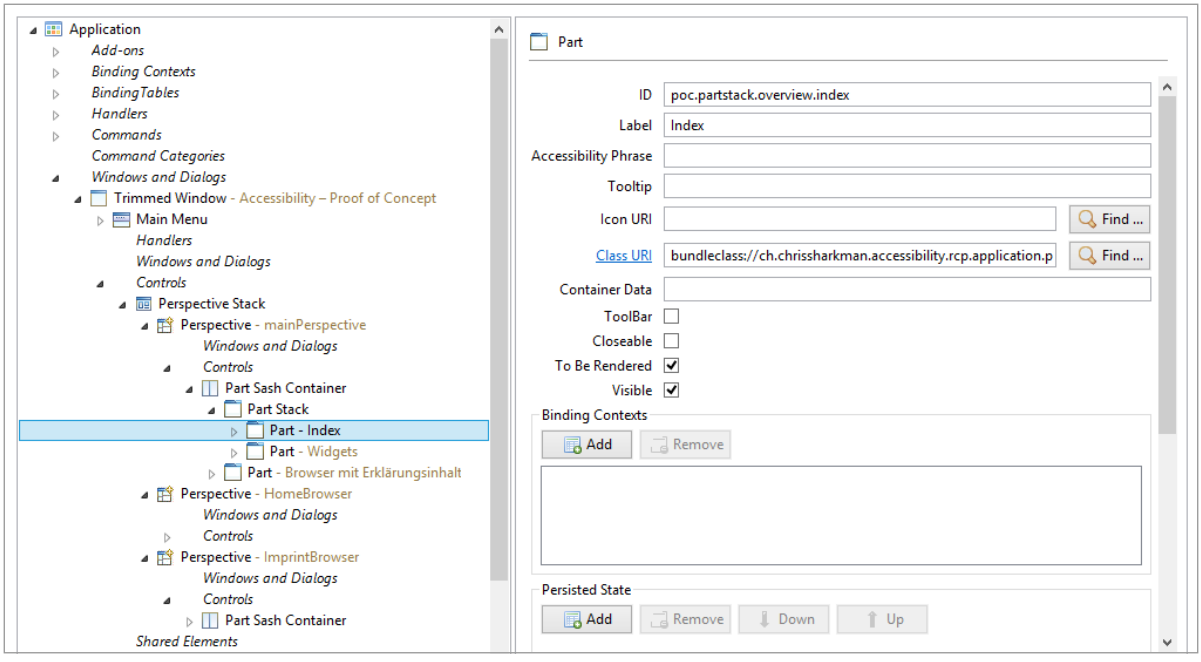


fig. 7 – *Grafische Benutzeroberfläche von eclipse zum Bearbeiten der Application.e4xmi-Datei mit aufgeklapptem Struktur-Baum*

6 Klassen, die vor Applikationsstart bestimmte Funktionen ausführen sollen. Automatisch wird die mit der Annotation @PostConstruct und init() gekennzeichnete Methode aufgerufen.

1.2.4. Applikation aufsetzen

Zum Aufsetzen eines *RCP*-Plugin-Projekt können mit dem «e4 Application Project Wizard» die nötige Root-Struktur und alle erforderlichen Dateien angelegt werden. Anschliessend kann die *GUI*-Struktur der Applikation bis auf Niveau «Part» über die Application.e4xmi-Datei festgelegt werden. Durch diesen Ansatz ist der Applikationsaufbau statisch. Natürlich können zusätzlich auch dynamische Perspektiven und Parts erzeugt werden, diese sind jedoch im Application.e4xmi nicht ersichtlich.

Hier alle Schritte wie in einem Tutorial zu erklären, wie am einfachsten die Basis für eine *RCP*-Applikation erstellt wird, wäre nicht zielführend: Diese Anleitungen existieren schon. Grundsätzlich wurden die ausführlichen Anleitungen und Tutorials von Lars Vogel[7], einem *eclipse*-*RCP*-Spezialisten, befolgt.

1.2.5. Nachteile einer RCP-Applikation als barrierefreie Anwendung

Die für die Bachelor-Vorarbeit gemachte Analyse hatte gezeigt, dass die *RCP*-Basis der Applikation schon einige Elemente zu einer barrierefreien Nutzung mitbringt.[8] Doch wo liegen die Nachteile, um eine *RCP*-Applikation barrierefrei zu machen? Als Nachteile sind folgende Elemente zu beachten:

- **Implementierte Standards**: Durch die Verwendung von nativen *GUI*-Elementen kommt oft deren Standard-Verhalten zum Zug und trägt dadurch schon etwas zur Barrierefreiheit bei. Doch für Erweiterungen oder Anpassungen dieser Standards wird es deswegen umso schwieriger.
- **Spärliche Dokumentation**: Arbeiten und Dokumentationen zu barrierefreien *RCP*-Applikationen sind rar. Auch wenn Accessibility Features beziehungsweise das in SWT vorhandene Accessible-Objekt dokumentiert sind, gibt es bisher wenig schriftlich festgehaltene Erfahrungen.
- **Einschränkungen durch Schnittstellen zu Dritt-Komponenten**: Man bewegt sich in einem Framework und dieses muss über das Betriebssystem mit einer weiteren Komponente, dem Screenreader, kommunizieren. Jede Schnittstelle bedeutet Einschränkungen *(fig. 8)*. Gegenüber einer nativen Programmierung, die auch direkt eine native Assistive Technologie aufrufen kann, ist der Ansatz *RCP*-Windows-*JAWS* klar im Nachteil. XAML/Narrator (Windows) und iOS/VoiceOver (Apple) harmonieren viel besser. In Windows ist für diese Schnittstelle *MSAA*, Microsoft Active Accessibility, zuständig. Dieses Addon des Betriebssystems hilft Assistiven Technologien normale Benutzeroberflächen zugänglich zu machen.[9]

---

7 Lars Vogel – Free Tutorials to program Eclipse RCP and Plug-ins – www.vogella.com > Tutorials > Eclipse RCP/Plug-ins

8 Christian Heimann – Blind am Computer: Bachelor-Vorarbeit, 3.3.2 Bereits funktionierende Elemente, Seite 21

9 Microsoft Active Accessibility – Wikipedia

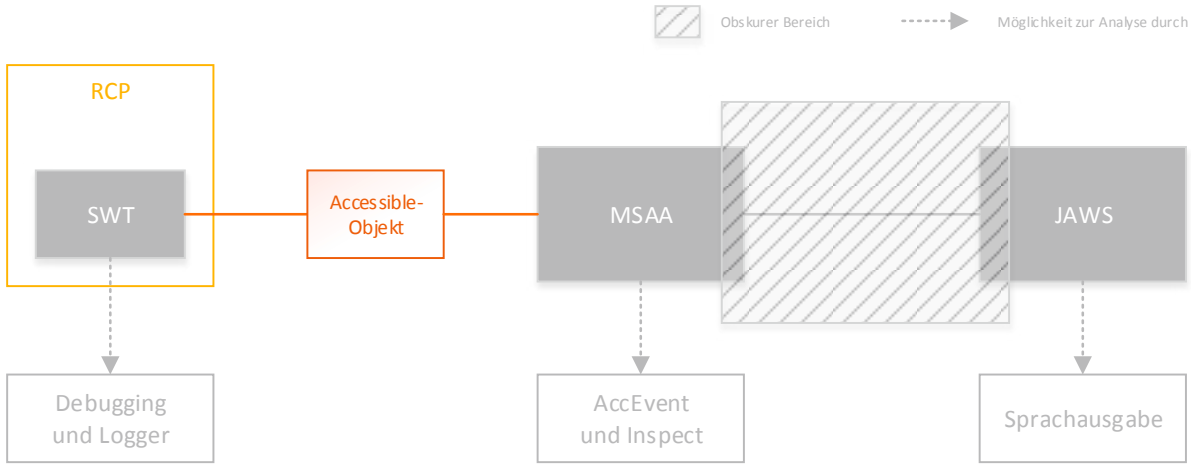


fig. 8 – *Accessible Technologie-Abhängigkeiten von der RCP-Applikation bis zur Sprachausgabe*

1.3. Details von verwendeten eclipse-Funktionalitäten

Auf drei Funktionalitäten der *eclipse-RCP*-Umgebung lohnt es sich kurz einzugehen: Annotationen, Dependency Injection und Eventbroker.

1.3.1. Annotationen

Annotationen sind mit @ beginnende Auszeichnungen vor Methoden, Konstruktoren und Feldern. Dadurch wird der Zeitpunkt, respektive der entsprechende abzufangende Event, bestimmt, welcher zum Ausführen des Codes führt. Durch Annotationen, die verhaltensabhängig Methoden aufrufen, können viele zu setzende Listener und dadurch eine starke Koppelung von Klassen vermieden werden. Bedingung zum korrekten Funktionieren von Annotationen ist, dass die Klasse vom *eclipse*-Framework initialisiert wurde.[10]

In der *Proof of Concept*-Applikation wurden folgende Annotationen gesetzt:

- **@Inject**: Zur Dependency Injection, siehe unten.
- **@PostConstruct**: Gekennzeichnete Methode wird nach dem Aufruf des Konstruktors ausgeführt, oft verwendet zum Aufruf der Methode zur Befüllung von Parts.
- **@Focus**: Bei Focus auf den entsprechenden Part, welcher die Annotation enthält, wird die mit @Focus gekennzeichnete Methode aufgerufen.
- **@Execute**: Annotation für Handler, mit welcher die auszuführende Methode des Handlers gekennzeichnet wird.[11]

1.3.2. Dependency Injection

Dependency Injection, durch die Annotation @Inject erzeugt, erhält bei *eclipse* 4 einen wichtigen Stellenwert, um die verschiedenen Teile einer Applikation möglichst ungekoppelt zu lassen.

10 Marc Teufel, Dr. Jonas Helming – Eclipse 4, Rich Clients mit dem Eclipse SDK 4.2, Seite 157
11 Marc Teufel, Dr. Jonas Helming – Eclipse 4, Rich Clients mit dem Eclipse SDK 4.2, Seite 158

Wichtig ist dabei: Eine selbst erstellte Klasse, die andernorts injiziert werden soll, muss seit *eclipse* 4.3 unbedingt mit der Annotation @Creatable markiert sein, ansonsten kann keine neue Instanz der Klasse erstellt werden.

Dependency Injection von MApplication, MPartService und EModelService injizieren immer die gleichen Instanzen, da diese pro Applikation nur einmal vorkommen. Bei Injektionen eigener Klassen kommt es bei mehrfacher Injektion in verschiedenen Klassen zu unterschiedlichen Instanzen.

1.3.3. Eventbroker

Um Events senden und empfangen zu können, steht ein globaler Eventbus zur Verfügung. Dieser wird über die Instanz des IEventBroker Objekts bedient. Durch Dependency Injection kann dieser leicht in jede beliebige Klasse der Applikation eingebracht werden.

Dem Eventbroker ist es möglich, mit der Methode send() synchrone und mit der Methode post() asynchrone Events zu schicken. Mit dem zweiten Attribut beider Methoden können dem Event auch gleich noch Daten mitgegeben werden falls nötig.

Injizierung des Eventbrokers und posten eines Events
Auszug aus: ch.chrissharkman.accessibility.rcp.application.poc.handler.EditHandler

```
public class EditHandler {

     private static Logger logger = Logger.getLogger(EditHandler.class);

     @Inject
     IEventBroker broker;

     @Execute
     public void execute(MApplication app, EModelService modelService,
     EPartService partService) throws InvocationTargetException,
     InterruptedException {
          broker.post("edit", null);
          logger.debug("broker posted edit");
     }

}
```

1.4. Unterschied SWT/Swing

In der Evaluations-Phase vor dem Entwicklungsstart der gesamten *eclipse*-Umgebung im Jahr 1998 war ursprünglich die Verwendung des damals brandneuen *Swing*-Toolkits zum Erstellen von GUI-Komponenten geplant. Aufgrund früherer Erfahrungen wurde diese Entscheidung aber revidiert: *Swing* verwendet direkt in Java implementierte Widgets anstatt die des jeweils darunterliegenden Betriebssystem zu verwenden. Um die Grundelemente optisch möglichst nahe am jeweils ausgeführten Betriebssystem des Benutzers zu halten, Performance-Vorteile gegenüber Java-implementierten Widgets zu haben und so eine für Entwickler attraktive Möglichkeit zur GUI-Erstellung zu bieten, wurde das Standard Widget Toolkit entwickelt: Eine Plattform unabhängige Programmschnittstelle, welche die Verwendung von Betriebssystem-eigenen GUI-Komponenten implementiert.[12]

12 FAQ: Why does Eclipse use SWT? – Eclipsepedia – wiki.eclipse.org

1.4.1. Barrierefreiheit: Vorteile auf beiden Seiten

Was bedeutet diese damalige Entscheidung betreffend Barrierefreiheit? Der Hauptunterschied zwischen Java *Swing* und *SWT* liegt in deren Ansatz: *SWT* unterstützt «out-of-process assistive technology», *Swing* unterstützt «in-process assistive technology». Das heisst, dass bei *Swing* die Kommunikation von Assistiven Technologien und dem Interface durch die gleiche darunterliegende virtuelle Maschine gemacht wird und dadurch sehr leichtgewichtig ist. Durch die Prozess-übergreifende Kommunikation, die für *SWT* nötig ist, werden die möglichen abfragbaren States und empfangbaren Events eingeschränkt. Ein Accessible-Objekt in jedem *SWT*-Control erlaubt die Modifikation der an die ATs gegebenen Werte.[13]

Bedauerlicherweise ist die Accessibility-Fähigkeit von Windows auf die *MSAA* «default accessibility» beschränkt, da sie komplett von *MSAA* abhängig ist.[14] Welche Werte bis zur Ausgabe eines Screenreaders kommen, wird unter «3.2. JAWS zum Sprechen bringen» auf Seite 23 beschrieben.

Interessant bei diesem Vergleich ist, dass eine *eclipse* RCP-Applikation, deren *GUI* mit *SWT* konstruiert werden, von Beginn an eine gute Zugänglichkeit aufweisen, wohingegen *Swing*-basierte Applikationen oft als «weisses Blatt»[15] daherkommen, aber eigentlich mehr Möglichkeiten zur Barrierefreiheit bieten könnten. Dieser Vorteil zugunsten von *SWT* lässt sich durch die Verwendung von Betriebssystem-eigenen Komponenten erklären.

1.4.2. SWT-Starrheit durch Top-Down-Ansatz

In *Swing* kann zu jedem Moment ein beliebiges *GUI*-Element instanziiert und erst später dem gewünschten Kontext zugeordnet werden. So kann zum Beispiel zuerst ein Button erzeugt werden und erst nach dessen Erstellung definiert werden, wohin der Button platziert werden soll.

SWT ist durch den gezwungenen Top-Down-Ansatz deutlich unflexibler. Jedes zu erstellende Control muss im Moment der Instanziierung bereits sein Parent-Element kennen.[16]

1.4.3. Swing-Verwendung in RCP

Es ist durchaus möglich, das *GUI* einer *RCP*-Applikation mit *Swing*-Komponenten zu erstellen. Es wird aber immer mindestens ein *SWT*-Composite als «Rahmen» für jegliche *Swing*-Elemente benötigen.[17] Standardmässig wird in der Community auf *SWT* gesetzt. Die Applikation *beook* wurde ebenfalls mit *SWT*-Komponenten erstellt, daher wurden die Accessibility-Möglichkeiten von *Swing* nicht weiter verfolgt.

---

13 Barry Feigenbaum, updated by Sueann Nichols – Accessibility Design and Coding Guidelines for Java Swing and SWT GUI Development, Seite 7

14 Barry Feigenbaum, updated by Sueann Nichols – Accessibility Design and Coding Guidelines for Java Swing and SWT GUI Development, Seite 14

15 Christian Heimann – Blind am Computer: Bachelor-Vorarbeit, Anhang, Interview mit Selamet Aydogdu

16 Barry Feigenbaum, updated by Sueann Nichols – Accessibility Design and Coding Guidelines for Java Swing and SWT GUI Development, Seite 8

17 FAQ: How do I embed AWT and Swing inside SWT? – Eclipsepedia – wiki.eclipse.org

2. «Proof of Concept»-Applikation

2.1. Absicht und Ziele

fig. 9 – *beook-Logo*

2.1.1. beook schonend erweitern

beook (fig. 9) hat sich als Plattform für digitale Lehrmittel in der Deutschschweiz bereits gut etabliert. Die Applikation existiert für die Plattformen iOS, Android und Mac/Windows. Die Java-basierte *eclipse-RCP*-Applikation, welche im Januar 2014 zum Angebot der *beook*-Plattform dazu kam, deckt gleichzeitig die Betriebssysteme Mac und Windows ab.

Wie die *beook*-Plattform als Ganzes, so wurde auch die *RCP*-Applikation stetig weiterentwickelt und steht heute auf einem soliden und getesteten Fundament. Die Architektur ist äusserst modular aufgebaut, damit die gleichen Technologien auch für Kunden-Adaptationen verwendet werden können, ohne unnötigen, redundanten Unterhalts-Aufwand zu generieren.

Zuerst war die Idee, einfach die nötigen Bausteinchen zur Unterstützung von Barrierefreiheit einsetzen zu können. Es wurde aber bald klar, dass dies nicht einfach so geht. Zudem sollte die Integration von Accessibility-Elementen dem modularen Aufbau von *beook* gerecht werden. Die verträglichste Art der Integration sollte mit dem *Proof of Concept* gefunden werden.

2.1.2. Erfahrungen sammeln

Im Wissen, dass die Arbeit an einer *eclipse-RCP*-Applikation Kenntnisse des Frameworks verlangt, war die Wahl für das Vorgehen während dieser Arbeit schnell gefällt: Zuerst ein *Proof of Concept* herstellen und anschliessend die Integration durchführen. Durch das Erstellen einer eigens zu Demonstrations-Zwecken konzipierten *RCP*-Applikation konnte der Applikationsaufbau im *RCP*-Framework genauer angeschaut werden. Gleichzeitig entstand damit ein Test-Bed, auch Test-Vehikel genannt, in dem ein reges Ausprobieren, Löschen, Editieren und Kopieren stattfinden konnte.

Der Weg über ein *Proof of Concept* erlaubte es, Erfahrungen zu sammeln, damit bei der Integration – der OP am offenen Herz – alles möglichst schonend und reibungslos abläuft. Es galt daher, ein möglichst klares Vorgehen zu finden – auch wenn die Integration der Barrierefreiheits-Elemente letztendlich nicht an den Kern der Applikation ging.

Erfahrungen zu sammeln war dafür umso wichtiger, da viele Aspekte der Barrierefreiheit nur spärlich dokumentiert sind. Das *Proof of Concept* bot dafür einen sicheren Rahmen für viel Trial & Error.

2.1.3. Öffentlich zugänglicher Code erstellen

Da es sich bei *beook* um Software mit geschütztem Quellcode handelt, bietet sich das *Proof of Concept* zur Veröffentlichung an. Dank einem frei zugänglichen Beispiel-Code auf tb.chrissharkman.ch/github können die gemachten Erfahrungen so ganz einfach mit der *eclipse* community geteilt werden, die gerade im deutschsprachigen Raum sehr lebendig und gross ist.

Als einer der wenigen, der sich intensiv mit Barrierefreiheit für RCP-Applikationen auseinandergesetzt hat, konnte Bryan Obright ausfindig gemacht werden. Er ist Mitarbeiter von Oracle und hatte sich im Jahr 2014 mit der Thematik beschäftigt. Er war als Referent zu dieser Thematik gelistet an der *eclipse*con 2014 in San Francisco; zu einem Vortrag kam es aber nie, da er nie ins definitive Programm aufgenommen wurde. Aus diesem Grund hat er seine Aufzeichnungen, Informationen und viele seiner Kenntnisse nie aufbereitet für ein breites Publikum – zu viele Oracle-proprietäre Informationen stecken noch darin. Ein Teilen dieses Wissens wird so, ohne übermässigen Aufwand zu betreiben oder Betriebsgeheimnisse zu verraten, sehr schwierig, ja gar unmöglich. Das Schicksal, wegen proprietären Informationen geheim bleiben zu müssen, sollte den während diese Arbeit erlangten Kenntnisse nicht widerfahren.

2.2. Grundstein: Leere «Proof of Concept»-Applikation

2.2.1. Konzept-Idee

Die grundlegende Konzept-Idee für die *Proof of Concept*-Applikation wurde von der bestehenden *beook*-Applikation abgeleitet *(fig. 10)*. Die Benutzeroberfläche des Test-Bed sollte deshalb der Hauptperspektive der Applikation, der Readeransicht mit Inhaltsspalte, ähnlich sein.

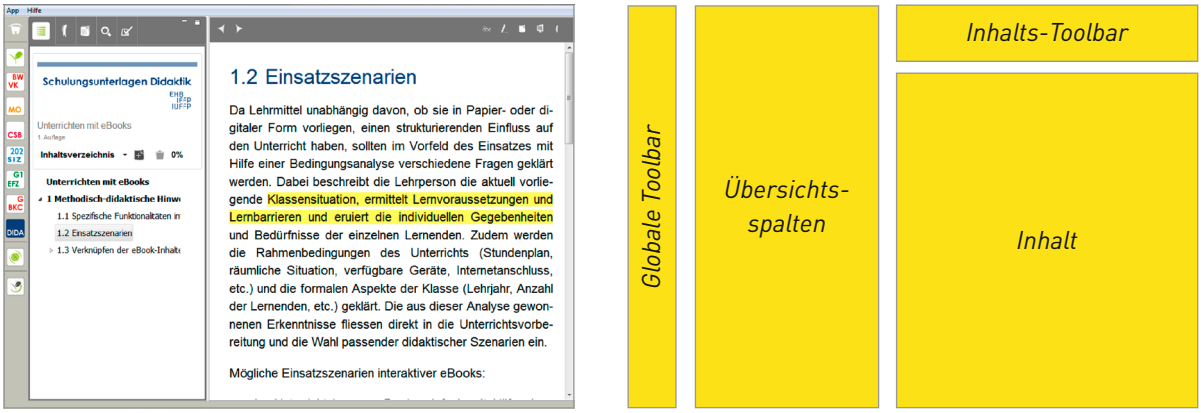


fig. 10 – *Ableitung der Struktur der Readeransicht von beook*

Zusätzlich sollte das *Proof of Concept* folgenden Punkten Rechnung tragen:

- Vielfältigkeit der Struktur simulieren
- Erfahrung mit unterschiedlichen Widgets zulassen
- Test-Aktionen bereitstellen

2.2.2. «Fertige» Proof of Concept-Applikation

Die Basis der Applikation wurde mit dem e4-Application-Project-Wizard erstellt. Die Struktur der *Proof of Concept*-Applikation wurde komplett in der Application.e4xmi-Datei festgelegt, da keine programmatische Perspektiven-, respektive Part-Erzeugung stattfindet.

Um die Beispiel-Applikation möglichst leicht lesbar zu halten, wurde auf abstraktionen verzichtet, die in einer «echten» Applikation zur effizienten Be-

wirtschaftung nötig wären. Gleiches galt für das Layout des *Proof of Concept*: Ein aufwendiges GUI wird schnell sehr wortreich. «So klar wie möglich mit so wenig wie nötig» war deshalb die Devise, um sich auf die Elemente zur Demonstration von Accessibility-Funktionalitäten zu beschränken.

Auch die Logik der Applikation wurde auf das Nötigste reduziert. Dadurch sind zum Beispiel zwischen der Übersichtsspalte und dem Inhaltsbereich keine Interaktionen implementiert.

Bestandteile

Das Test-Bed, die «fertige» RCP-*Proof of Concept*-Applikation, enthält drei verschiedene Perspektiven: Home, Main und Imprint.

Die Home-Perspektive *(fig. 11)* enthält einen einzigen Part mit der Browser-View. Diese enthält ein Composite als Toolbar, welche vier Buttons und ein Browser-Widget beinhaltet.

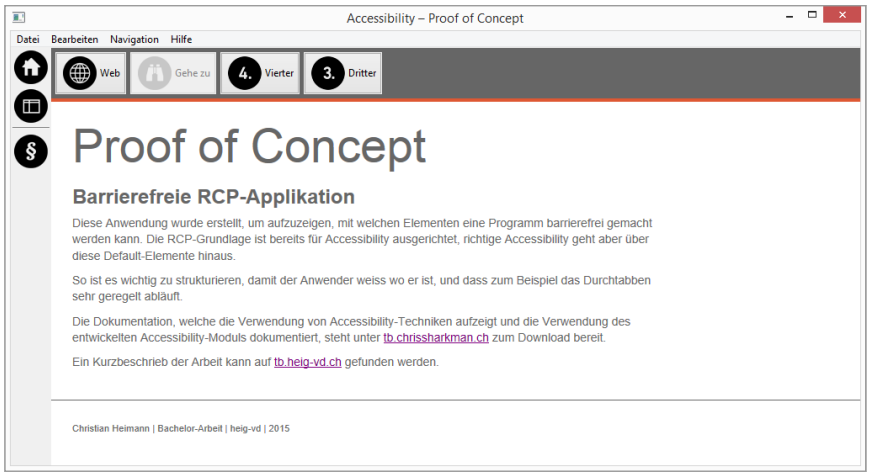


fig. 11 – *Home-Perspektive des Proof of Concept: Dunkelgrau hinterlegt ist die Toolbar mit den vier Buttons, ab der orangen Linie beginnt das Browser-Widget*

Die Main-Perspektive *(fig. 12)* wiederspiegelt die Anordnung der *beook*-Reader-Ansicht. Sie enthält auf der linken Seite einen Bereich mit PartStack und zwei darin gesetzten Parts (Index und Widgets), auf der rechten Seite erneut einen Part mit der BrowserView.

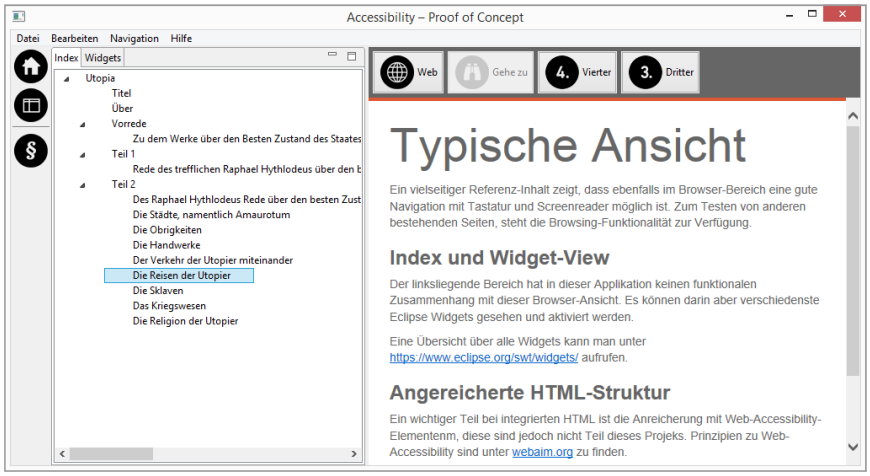


fig. 12 – *Main-Perspektive des Proof of Concept*

Für die dritte, die Imprint-Perspektive *(fig. 13)*, war zuerst eine Kopie der Home-Perspektive vorgesehen: Um einfache Elemente zu schaffen und beim Navigieren eine kleine Auswahl zu haben. Beim Testing – bereits mit *Accessible Modul* – wurde festgestellt, dass es zur Überprüfung der Funktionalität zuträglich ist, der Imprint-Perspektive noch einen weiteren Part hinzuzufügen. So enthält diese Perspektive ein Part mit BrowserView und zusätzlich ein weiteres Part mit einem Composite das zwei Buttons beinhaltet (OptionView).

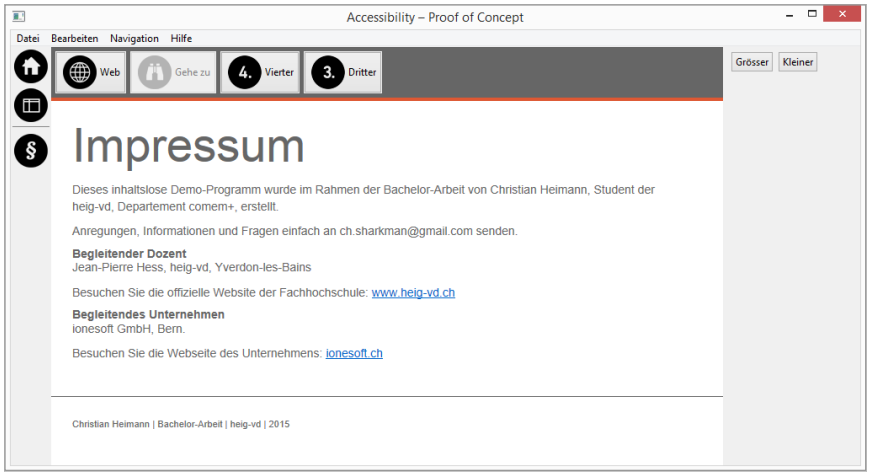


fig. 13 – *Imprint-Perspektive des Proof of Concept*

Als globale Elemente wurden eine Menüleiste und eine globale Toolbar gesetzt. Die Menüleiste weist nebst dem Befehl zum Beenden der Applikation auch noch die Menüs «Bearbeiten», «Navigation» und «Über» auf. Diese wurden vorbereitender Natur gesetzt, um gewisse Aktionen über die Menüleiste zugänglich zu machen. Dieser Ansatz wurde für diese Arbeit aber nur zu Beginn kurz verfolgt, bleibt aber eine Ausbaumöglichkeit um alle Funktionalitäten verfügbar zu machen. Der Grundidee, ein Touchfähiges Interface zu haben, welches auf allen Plattformen nahezu identisch aussieht, widerspricht die Aktionsführung über die Menüleiste aber. Zusätzlich würde dies Redundanzen ergeben.

Durch diesen Aufbau kann die Vielfältigkeit der *GUI*-Struktur simuliert werden.

Unterschiedliche Widgets

In der Main-Perspektive ist der Widgets-Part mit einer Auflistung der verschiedensten Widgets bestückt. Durch diese Übersicht konnten Erfahrungen mit einer breiten Palette von Widgets gesammelt werden.

Zudem wurde die Tree-Erstellung, die Baum-Struktur im Index-Part, programmatisch erzeugt. Dazu wurde eine externe Ressource eingebunden, auch um das korrekte Verwalten externer Ressourcen zu festigen. Bei der Ressource handelt es sich um eine ncx-Datei, dem Inhaltsverzeichnis-Format von *EPUB*.

Um sich in der restlichen Applikation auf den Navigationsfluss zu konzentrieren, wurde in der Inhalts-Toolbar und in der OptionView die Realität ver-

einfacht: Darin wurden ausschliesslich Composites und Button-Controls gesetzt.

Test-Aktion erstellen

Um einen Funktionsaufruf zu simulieren, wurde direkt im Application.e4xmi ein Command auf alle Menü-Punkte des Bearbeiten-Menüs gesetzt. Diese führen zur mit @Execute annotierten Methode des EditHandlers. Darin wird als einzige Aktion der «edit» Event via Eventbroker gesendet. Dieser Event wird im BrowserViewPart empfangen. Dort wird über ein dem Browser-Widget injiziertes Javascript der selektierte Text ausgelesen und anschliessend ein normales Dialogfenster mit dem selektierten Text erstellt *(fig. 14)*. Dies ist eine Anlehnung an die Markierfunktionalität in *beook*. Das Interessante dabei war das Zusammenspiel von Java in *RCP* und Javascript im Browser-Widget. Das folgende Code-Beispiel zeigt den Empfänger des «edit» Events und die Injizierung des Javascript-Codes in das Browser-Widget.

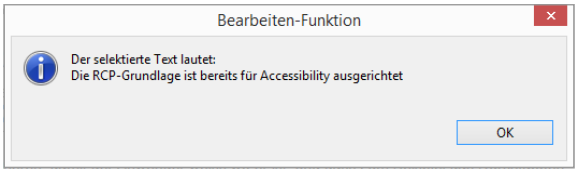


fig. 14 – *Dialog mit selektiertem Text als Nachricht*

Abfangen eines Events und Injizierung einer Javascript-Funktion in das Browser-Widget
Auszug aus: ch.chrissharkman.accessibility.rcp.application.poc.parts.BrowserViewPart

```
/**
 * Function to read the actual selected text in browser part.
 * This function is called when edit event is sent.
 * @param string
 */
@Inject
@Optional
public void editText(@UIEventTopic("edit") String string) {
     if (this.browser.isVisible()) {
          logger.info("call in edit");
          this.browser.setJavascriptEnabled(true);
          String script = new String("return getSelectedText(); "
               + "function getSelectedText () {if (window.getSelection)"
               + "{var range = window.getSelection (); "
               + "return (range.toString ());} else {"
               + "if (document.selection.createRange) {var range = "
               + "document.selection.createRange ();"
               + "return (range.text);}}}");
          String result = (String) this.browser.evaluate(script);
          MessageDialog.openInformation(this.browser.getShell(),
               "Bearbeiten-Funktion", "Der selektierte Text lautet:\n" +
               result);
     }
}
```

2.3. Generelle Problemstellungen

Während der Erstellung der «leeren» *Proof of Concept* Applikation traten einzelne Schwierigkeiten auf, zu denen es unter anderem auch durch Unerfahrenheit gekommen ist.

2.3.1. Wiederverwendbarkeit des Browser-Parts

Da im *Proof of Concept* dreimal der gleiche Part verwendet wird war der initiale Ansatz, die gleiche Instanz des Parts für mehrere Perspektiven zu verwenden. Das entsprach aber nicht dem Gedanken, dass eine Perspektive für «eine Anwendung» steht. So wird jetzt für jede der drei Perspektiven eine eigene Browser-Part-Instanz erzeugt. Da die Events, welche von den But-

tons propagiert, auch direkt von der jeweiligen Instanz gehandhabt werden, kommen sich diese nicht in die Quere.

Anders ist es bei dem ausgelösten Event vom «Bearbeiten»-Menü. Dieser wird über den Event-Broker von Eclipse gemacht. Da dieser gesendete Event aber von allen Browser-Part-Instanzen abgefangen wird, muss in der auszuführenden Methode zuerst geklärt werden, ob es die jeweilige Instanz überhaupt betrifft. In diesem Fall wird dies durch die Überprüfung von isVisible() gemacht: Es wird kontrolliert, ob die jeweilige Browser-Part-Instanz derzeit sichtbar ist – oder eben verdeckt von anderen Perspektiven.

2.3.2. Fragilität der Application.e4xmi-Datei

Application.e4xmi ist fragil was die Handler und Commands angeht. Es kommt zu Fehlern beim Hinzufügen und Entfernen von Knoten, welche die Datei korrumpieren. Vor dem Schliessen der Datei – die von Eclipse nicht mehr geöffnet werden kann, sobald sie korrupt ist – ist ein Überprüfen der Datei im Reiter «xmi» zu empfehlen, in dem strukturelle Fehler angezeigt werden. Ein regelmässiges Backup des Projekts ist generell empfohlen.

Das Problem ist, dass beim Entfernen von Handlern und Commands in den <bindingTables> zum Teil «Bindings» bestehen bleiben. Wenn eine Referenz-Id nicht mehr gefunden werden kann, scheint die Datei für *eclipse* korrupt zu sein. In diesem Fall soll man die Datei nicht schliessen, sondern durch die Info des Error-Logs das mögliche Binding eruieren und manuell in der Code-Ansicht des Application.e4xmi löschen.

2.3.3. Export der RCP-Applikation

Für die Paketierung einer *eclipse-RCP*-Applikation steht ein Export-Wizard zur Verfügung. Es ist wichtig im Vorfeld zu kontrollieren, ob alle verwendeten Elemente, die sich ausserhalb des Quellen-Ordners (src) befinden, auch miteingebunden werden beim Export. Das Betrifft unter anderem Plugin-Projekte und Inhalte aus anderen Ordnern. Dies sind zum Beispiel das «base» Plugin-Projekt, welches das *Accessible Modul* enthält, und der Ordner «content» mit den *HTML*-Dateien.

An den Orten poc.product > Dependencies, in den build.properties > Binary Build, im plugin.*XML* > Dependencies und plugin.*XML* > Runtime > Exported Packages/Classpath wird definiert, was beim Export alles gesammelt werden muss. Eine gute Übersicht zu behalten ist durch Redundanzen in der grafischen Benutzeroberfläche und der vielen verschiedenen Orte für Plugins und Abhängigkeiten nicht einfach.

3. Konkrete Mittel und Wege zur Barrierefreiheit

3.1. Vorgehen

3.1.1. Setup

Während der Versuchsphase wurde in der *Proof of Concept*-Applikation ausprobiert und ausgiebig nach Lösungen gesucht, die Accessibility auf eine Stufe über dem vorhandenen Standard zu heben. Die Recherche nach bestehenden, spezifischen Dokumenten zum Thema Accessibility in Zusammenhang mit *RCP* war schwierig.

Das Versuchs-Setup war dabei bestimmt durch die in der Schweiz am häufigsten unter der Zielgruppe verbreiteten Ausstattung: Windows Betriebssystem und *JAWS*-Screenreader.[18]

Um System-Events zu detektieren wurde mit der Anwendung «AccEvent», dem Accessible Event Watcher des Windows SDK, gearbeitet. Der Strukturaufbau der Applikation und die Accessibility-Fähigkeit konnte mit dem aDesigner, einem auf *eclipse* basierten «Accessibility Checker», überprüft werden. Beide Tools sind auf die Verbindung von der Applikation zu *MSAA* ausgerichtet, nicht aber auf die Funktionsweise und Events von verwendeten ATs.

3.1.2. Dokumentationen

Ein Dokument, das viele Hinweise liefert zur konkreten Problemstellung und das zusätzlich anhand von Code-Beispielen auf Details eingeht, ist die Arbeit von Barry Feigenbaum, *IBM* Worldwide Accessibility Center und Sueann Nichols, *IBM* Human Ability and Accessibility Center mit dem Titel « Accessibility Design and Coding Guidelines for Java *Swing* and *SWT GUI* Development». Ein weiteres, globaleres Dokument ebenfalls aus der Hand von *IBM* ist das «Creating Accessible Applications in Eclipse (2nd Edition)». Hier klärt ein genereller Teil gut auf über die konzeptionellen Aspekte von Accessibility, komplette Code-Beispiele von Widget-Details geben viele Inputs zu *SWT*.

Die Suche nach Dokumentationen über Accessibility-Schnittstellen und Funktionalitäten betreffend Betriebssystem Windows, endete grundsätzlich auf den Microsoft-Seiten zu *MSAA*. Eigentlich genau richtig, da *MSAA* ja die Brücke schlägt zwischen der *RCP*-Applikation und dem Screenreader. Nur befindet man sich dabei bereits auf einem Niveau im System, das com-basiert ist und somit über den *SWT*-Mapping-Layer mit der Java-*SWT*-Umgebung verbunden wird.[19]

Für die Entwicklung von barrierefreien Applikationen für Windows gibt es derzeit viel neues, aktuelles Material, wie zum Beispiel Video-Präsentationen zu Accessibility-Techniken. Diese sind aber grundsätzlich für das Binom

18 Christian Heimann – Blind am Computer: Bachelor-Vorarbeit, 1.2 Gängiges Hardware-Setup, Seite 7

19 Alex Blewitt – The difference between SWT and AWT – alblue.bandlem.com

«Narrator mif XAML» gemacht, um die Entwicklung von Windows-Apps die auch den nativen Screenreader benötigen zu begünstigen.

Dokumentationen zum Screenreader *JAWS*: da war die Suche noch erfolgloser. Selbst die Anfrage beim Europa-Hauptsitz von *Freedom Scientific* brachte keine anderen Dokumente zutage, als jene, die schon auf der Website zur Verfügung stehen. Somit bleibt die Verbindung und die Funktionsweise zwischen *MSAA* und *JAWS* komplett im Dunkeln. Bereits vor einigen Jahren war es nicht möglich, an diese Informationen zu kommen.[20]

3.1.3. Accessibility Phrase und Accessible-Objekt

Das Studium der bestehenden Möglichkeiten in *RCP* und *SWT* hatte hingegen vielversprechend begonnen. So ist bereits in der *GUI*-Maske zum Setzen von Perspektiven und Parts ein Feld mit dem Label Accessibility Phrase markiert *(fig. 15)*.

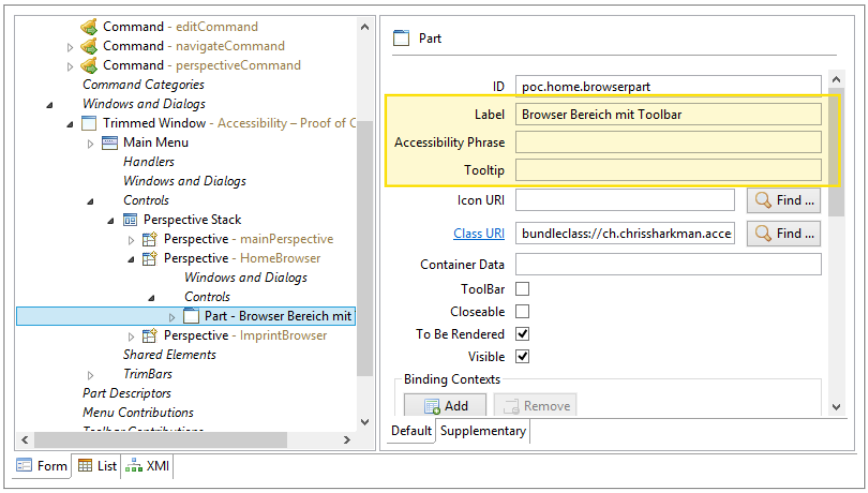


fig. 15 – *Die Maske zur Konfiguration eines Parts im Application.e4xmi: Die Felder Label, Accessibility Phrase und Tooltip scheinen der Barrierefreiheit dienlich zu sein.*

Sowohl in den Perspektiven wie auch in den Parts hat ein gesetzter Text im «Accessibility Phrase»-Feld keine akustische Ausgabe mit *JAWS* erzeugt. Der einzige ausgesprochene Text ist jener des Labels. Dieser kann aber nicht frei für Accessibility-Zwecke definiert werden, da zum Beispiel die Reiter von Parts in PartStack-Elemente mit dem gesetzten Label beschriftet werden.

Ein anderes, vielversprechend klingendes Element war das Accessible-Objekt, welches die Brücke schlägt zwischen *SWT*-Widgets und Assistiven Technologien. Jedes Control enthält damit ein Proxy zur Betriebssystemeigenen Accessibility API. Die meisten Betriebssysteme liefern bereits ein Standardverhalten für die meisten Widgets, mit dem Accessible-Objekt kann dieses Standardverhalten überschrieben, beziehungsweise ergänzt werden.[21] Das Accessible-Objekt kann für jedes Objekt des Typs *SWT*-Widget über die Methode getAccessible() aufgerufen werden.

20 Fragen an Alexander Hauser – Telefon-Interview

21 Class Accessible – Documentation Eclipse Platform Release 4.2 – help.eclipse.org

Aufrufen des Accessible-Objekt eines Buttons und setzen des AccessibleListeners
Auszug aus Versuchsdatei

```
Composite compButton = getWidgetComposite(compControls);
Button button = new Button(compButton, SWT.PUSH);
button.setText("Hier klicken");
button.getAccessible().addAccessibleListener(new AccessibleAdapter() {

     @Override
     public void getName(AccessibleEvent e) {
          super.getName(e);
          e.result = "Schiesst die Rakete zum Mond";
     }
}
```

Schaut man sich das Interface «AccessibleListener» an, dann sieht man, dass die Implementation vier Methoden vorsieht:

- getDescription(AccessibleEvent e)
- getHelp(AccessibleEvent e)
- getKeyboardShortcut(AccessibleEvent e)
- getName(AccessibleEvent e)

Alle vier Methoden werden von Windows aufgerufen, dies konnte durch simples Debuggen getestet werden. Sobald der Fokus auf das entsprechende *SWT*-Widget fällt, werden die Methoden ausgeführt. Das Betriebssystem ist aber nur der halbe Weg zum Benutzer mit Screenreader *JAWS*: Von *JAWS* wird der in der getName-Methode gesetzte e.result-Wert ausgesprochen, dies anstelle eines allenfalls existierenden Labels oder eines gesetzten Textes auf dem Widget. Hilfetext (getHelp), Beschreibung (getDescription) und Kurzbefehl (getKeyboardShortcut) konnten hingegen nie mit *JAWS* ausgegeben werden, auch nicht nach intensivem Studium der *JAWS* Benutzer-Dokumentation.

Für die Klassen, welche das MUIElement-Interface implementieren, steht die Methode getAccessible nicht zur Verfügung. Das heisst, die Möglichkeit über getName die Sprachausgabe zu definieren fällt für Parts und Perspektiven weg.

3.1.4. Ausgangslage verschoben

Nach einer intensiven Phase der Recherche stellte sich die Gewissheit ein, dass ein ausführliches Testing mit der Methode Trial & Error Klarheit bringen muss, ob wirklich nicht mehr mit *JAWS* ausgegeben werden kann. Damit beim Ausprobieren möglichst nichts übergangen wurde, musste der Zeitplan umgestellt werden. Der Phase *Proof of Concept* wurde mehr Zeit eingerechnet, was unweigerlich bei der Implementations-Phase zu Kürzungen führte.

3.2. JAWS zum Sprechen bringen

3.2.1. Einstellungen

Die Screenreading-Software *JAWS* lässt eine Vielzahl von Einstellungen zu, um sich den vielen verschiedenen Bedürfnissen seiner Benutzer anpassen zu können. Um bei der Arbeit mit *JAWS* alle verfügbaren Sprachausgaben

auszugeben, wurden in den *JAWS*-Grundeinstellungen die folgenden Einstellungen gesetzt:

- **Tutormeldungen**: Hilfe für Menüs und Elemente ansagen
- **Zugriffstasten**: Alle sprechen

In den Stimmeinstellungen, welche eigentlich keine Auswirkungen haben auf «was» ausgesprochen wird sondern lediglich auf das «wie», wurde folgende Einstellung gesetzt:

- **Stimme**: Satzzeichen: Alle

In der *JAWS*-Einstellungsverwaltung wurden folgende Einstellungen gesetzt:

- **Sprachausführlichkeit**: Ausführlichkeitsstufe: Einsteiger, höchste; In den Einsteiger-Einstellungen wurden als «zu sprechenden Elemente» alle Elemente aktiviert und die Länge von *JAWS*-Meldungen auf Lang gesetzt.

Durch dieses Setting war die Grundvoraussetzung gegeben, alles mitzubekommen, was *JAWS* über die Sprachausgabe ausgeben werden kann. Dass ein fortgeschrittener Benutzer die Sprachausgabe «reduziert», um am Computer effizienter und schneller arbeiten zu können, wurde für diese Arbeit nicht berücksichtigt.

3.2.2. JAWS verwenden

Die korrekte Verwendung von *JAWS* erfordert Übung. Es genügt deshalb nicht, einfach *JAWS* zu installieren und «loszutabben» um wirklich zu erfahren, ob die Verwendung einer Applikation «barrierefrei» ist.

Tastaturbefehle

Aus den etlichen Seiten an Screenreader-eigenen Tastaturbefehlen zum Aufruf von Funkionalitäten müssen die wichtigsten herausgepickt werden. Denn seitenweise Tastenkombinationen können, gerade zu Beginn, wenn man den Screenreader noch nie verwendet hat, nur schwer behalten werden. Eine Auswahl:

- **[tab]** Sprung zum nächsten Link oder fokussierbaren Element
- **[Insert + Pfeil nach unten]** Alles vorlesen
- **[Insert + Pfeil nach links]** Vorheriges Wort vorlesen
- **[Insert + Pfeil nach rechts]** Nächstes Wort vorlesen
- **[ctrl]** Sprachausgabe unterbrechen
- **[Insert + F4]** *JAWS* beenden

Sobald man sich in einem Browser-Kontext befindet, stehen zusätzliche, praktische Kurztasten zur Verfügung. Eine Auswahl:

- **[B]** Sprung zum nächsten Button
- **[H]** Sprung zum nächsten ausgezeichneten Titel
- **[R]** Sprung zur nächsten *ARIA*-Region
- **[U]** Sprung zum nächsten, nicht besuchten Link
- **[V]** Sprung zum nächsten, bereits besuchten Link

Die vollständige Liste von Tastaturbefehlen steht auf der Webseite von *Freedom Scientific* zur Verfügung.[22]

Geschwindigkeit

Zu Beginn wird man als ungeübter Benutzer mit der akustischen Informationsflut gefordert. Um die Sprachausgabe gut zu Verstehen, kann die Geschwindigkeit heruntergesetzt werden. Auf der Skala von 40 bis 150 ist ein Wert von 60 für Anfänger gut. So kann auch das verwendete Vokabular erlernt werden. Mit regelmässigem Gebrauch ist bald ein Wert von 110 möglich – «normal Sehende» kommen man da aber an die Grenze der verarbeitbaren, akustischen Information.

10 Silben pro Sekunde, so hoch ist die absolute Limite für sehende Personen. Das ist in etwa so schnell, wie am Ende von Medikamenten-Werbungen die «hyperaktiven» Sprecher ihren Standardsatz «... und lesen Sie die Packungsbeilage ...» herunterspulen. Blinde können hingegen bis zu 25 Silben pro Sekunde verstehen. So schnell kann zwar kein Mensch sprechen, aber künstliche Stimmen können extrem schnell eingestellt werden – so, dass es für Ungeübte nur noch nach Rauschen tönt.[23]

Die Stimmgeschwindigkeit von *JAWS* kann nicht nur über das Menü sondern auch mit **[alt + windows + ctrl + Page Up]** verschnellert beziehungsweise mit **[alt + windows + ctrl + Page Down]** verlangsamt werden.

Blind navigieren

Um als «sehender» Tester die Knackpunkte wirklich wahrzunehmen, braucht es den kleinen Schritt – man könnte beinahe von ein bisschen Mut sprechen – sich mit einer Augenbinde die Sicht zu nehmen und sich darauf einzulassen, blind zu navigieren. Dabei hat man in der Regel aber einen Vorteil gegenüber Nicht-Sehenden: Oft hat man das Layout der Applikation bereits gesehen. Man hat also bereits vor dem Start ein Bild im Kopf, wo man sich ungefähr befindet, respektive zu welchem Punkt man navigieren will.

Im Moment, wenn einmal kein Feedback vom Screenreader mehr kommt, fühlt man sich verloren. Dies kann zum Beispiel passieren, wenn ein Element nicht beschriftet ist oder der Fokus ins Nichts springt. Wichtig ist da, sich über einen Standardbefehl zu einem nächsten Orientierungs-Punkt, einer «Landmark», bewegen zu können. Dazu mehr unter «3.3.2. Der Navigation Struktur verleihen» auf Seite 31.

3.2.3. Tooltip, Label, getName – andere Stufe, anderer Zugriffspunkt

Durch die gemachten Versuche konnte geklärt werden, in welchen Situationen und mit welchen Elementen man *JAWS* zum Sprechen bringt.

Beim Öffnen/Anzeigen einer neuen Perspektive, wird das Label des Parts, welches den Fokus bekommt, vorgelesen. Beim normalen «Durchtabben» ist dies nicht der Fall. Aufgepasst: In PartStacks werden die Labels zur Be-

22 JAWS Keystrokes – www.freedomscientific.com > Support > Documentation > JAWS

23 R. Douglas Fields – Why Can Some Blind People Process Speech Far Faster Than Sighted Persons? – Scientific American – www.scientificamerican.com > Mind & Brain > News

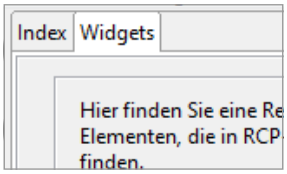


fig. 16 – *Tab-Reiter mit der Beschriftung analog Labels der Parts «Index» und «Widgets»*

schriftung der Tab-Reiter verwendet *(fig. 16)* und erfüllen deshalb nicht in erster Linie einen Accessibility-Zweck.

Bei Buttons wird der Text vorgelesen, wenn nur ein setText() gemacht wird. Dies ist häufig der Fall, daher funktioniert dieser Standard gut. Existiert ein zugewiesenes Label, dann überlagert das Label den gesetzten Text. Wird ein AccessibleListener mit getName-Methode gesetzt, so überlagert dessen getName-Wert den Text und mögliche Labels.

Bei Widget-Gruppen wird immer der auf der Gruppe gesetzte Text plus zusätzlich das Label des Widgets oder der auf dem Widget gesetzte Text vorgelesen. Bei Radiobutton-Gruppen bekommt beim «Durchtabben» lediglich das erste Element den Fokus, bei Checkboxen jedes.

Im Trimbar-Bereich (der globalen Toolbar) werden von den HandledToolItems die Tooltip-Angaben vorgelesen. Den HandledToolItems gesetzte Labels werden von *JAWS* nicht berücksichtigt.

Kurz zusammengefasst müssen für diese Elemente für Screenreader folgende Felder beziehungsweise Funktionen gesetzt werden:

- **Part**: Label
- **Widget**: getName() vor setText() respektive Label
- **HandledToolItem**: Tooltip

3.2.4. Technische Finessen beim Erstellen des GUIs

Schon bei der programmatischen Erstellung des *GUI* kann die Barrierefreiheit unterstützt werden. Grundsätzlich wird von Screenreadern die Programmstruktur durchleuchtet. Die Reihenfolge von Elementen ist dabei wichtig, um Bestandteile einander zuordnen zu können und die Sprungreihenfolge zu definieren.

Label setzen für Widgets

Bei *SWT*-Komponenten passiert die Zuordnung standardmässig sequenziell: Ein Label, das direkt vor einem anderen *SWT*-Widget erzeugt wird, wird automatisch das beschreibende Label für dieses andere *SWT*-Widget.

Wenn die Zuordnung nicht durch den Programmablauf passieren kann oder soll, dann gibt es bei allen *SWT*-Widgets die Möglichkeit, über das Accessible-Element und der Methode addRelation() die Widgets in Relation zu setzen. Dazu muss man mit der Accessible-Konstante «relation label for» respektive «labelled by» und dem jeweils dazugehörenden Widget die Funktion addRelation() aufrufen. Wird im String des Labels mit dem Vorzeichen «&» ein Mnemonic definiert, so fällt beim Drücken der entsprechenden Kurztaste der Focus auf das dem Label folgende Control.[24]

24 Class Label – Documentation Eclipse Platform Release 4.2 – help.eclipse.org

addRelation-Methoden-Aufruf um Label einem Text-Widget zuzuordnen, mit gesetztem Mnemonic
Auszug aus Versuchsdatei

```
Label label = new Label(composite, SWT.NONE);
label.setText("&E-Mail");
Text emailField = new Text(composite, SWT.BORDER);
emailField.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
Accessible accLabel = label.getAccessible();
Accessible accEmailField = emailField.getAccessible();
// add Relation between text and label
accLabel.addRelation(ACC.RELATION_LABEL_FOR, accEmailField);
accEmailField.addRelation(ACC.RELATION_LABELLED_BY, accLabel);
```

Aber Achtung: Label werden nur bei den Widgets «Text», «Slider», «Scale», «Spinner» und «Combo/Dropdown» vorgelesen. Alleinstehende Labels werden grundsätzlich übergangen und sind daher zu vermeiden, da Labels den Fokus nicht bekommen. Bei Button-Controls wird ein gesetzter Text erwartet oder allenfalls die Bezeichnung einer eingefügten Grafik ausgesprochen – was unbedingt über einen gesetzten AccessibleListener übersteuert werden sollte mit der getName-Methode.

Text-Elemente – übergangene Texte und suboptimale Hacks

eclipse empfiehlt für die Verwendung von Text-Elementen das Text-Widget mit dem Style-Bit «read only» anstatt Labels zu verwenden. So wird das Widget vom Fokus erfasst und dadurch vom Screenreader vorgelesen.

Text-Widget mit Style-Bit «read only»
Auszug aus: ch.chrissharkman.accessibility.rcp.application.poc.parts.WidgetViewPart.java

```
// Introduction Text
Composite compIntro = getWidgetComposite(compControls);
Text textIntro = new Text(compIntro, SWT.WRAP | SWT.READ_ONLY);
textIntro.setLayoutData(new GridData(SWT.FILL, SWT.CENTER, true, true));
textIntro.setText("Hier finden Sie eine Reihe von gängigen Kontroll-Elementen, "
     + "die in RCP-Applikationen Verwendung finden.");
```

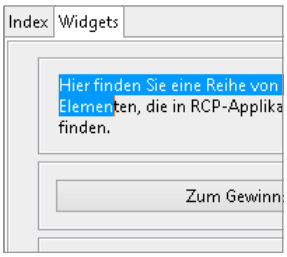


fig. 17 – *Selektierbarer Text in Text-Widget*

Für den «Normal Benutzer» stellt es allerdings ein ungewohntes Verhalten dar, wenn Beschreibungstext in grafischen Benutzeroberflächen selektiert werden kann und beim «Durchtabben» ein Cursor gesetzt wird *(fig. 17)*.

Da nützt auch ein im Web oft zitierter Trick[25] nichts, um beim SWT-Text-Widget über die Funktion setEnabled(*false*) den Cursor zu verstecken. Dies führt dazu, dass der Fokus das Element überspringt und der Screenreader das Element gar nicht erfasst. Enabled graut auch den Text per default aus. Es bleibt also die Entscheidung zwischen «Optischer Cleanness» und «Barrierefreiheit».

Bei dieser Problematik und anderen Entweder-Oder-Entscheidungen wäre eine Möglichkeit, man würde einen Modus-Wechsler in die Applikation einfügen. Dieser würde dann eine Label-Factory beeinflussen, welcher Widget-Typ gesetzt würde.

Sprachfluss für Fragen mit Check- und Radioboxen

Bei Fragestellungen in Formular-Situationen, die mit Check- oder Radioboxen beantwortet werden, ist es besser, diese in ein Group-Widget zu setzen

25 Cursor visibility in Text SWT – stackoverflow.com – Question 23985519

und nicht mit einem vorgängigen Label zu beschreiben. So wird die Frage für jede Variante wiederholt. Bei Checkboxen kann man so sicher sein, auf den richtigen Kontext zu antworten. So stimmt der Sprachfluss optimal für den Benutzer. Dies verändert zwar leicht die Optik (gegenüber ohne Group), ist aber für eine gute Accessibility der richtige Ansatz.

Erstellen eines Group-Widgets mit Radio Buttons
Auszug aus: ch.chrissharkman.accessibility.rcp.application.poc.parts.WidgetViewPart.java

```
// Create parent composite element
Composite compRadio = getWidgetComposite(compControls);

// Create group, place into composite without particular styling
Group groupRButtons = new Group(compRadio, SWT.NONE);

// Definition of the layout of the radio button group: 1 column grid layout,
// filling the complete height and width
GridData gdGroupRButtons = new GridData(SWT.FILL, SWT.FILL, true, true);
groupRButtons.setLayoutData(gdGroupRButtons);
groupRButtons.setLayout(new GridLayout(1, true));

// Set the question text of the radio button group
groupRButtons.setText("Wie ist das aktuelle Wetter?");

// Creation of radio buttons
Button rButtonA = new Button(groupRButtons, SWT.RADIO);
Button rButtonB = new Button(groupRButtons, SWT.RADIO);
Button rButtonC = new Button(groupRButtons, SWT.RADIO);

// Set text of radio buttons
rButtonA.setText("sonnig");
rButtonB.setText("bewölkt");
rButtonC.setText("regnerisch");

// Set default selection
rButtonA.setSelection(true);
```

3.2.5. Eventbasierte Sprachausgabe

Auf dem Betriebssystem iOS (ab iOS 7) stellt die AVSpeechSynthesizer-Klasse viele Möglichkeiten zur Verfügung. Die Sprachausgabe kann damit gezielt gesteuert werden[26] und zudem können auch spezifische, eventbasierte Ausgaben erzeugt werden. Diese tolle Möglichkeit, den Screenreader «etwas sagen zu lassen», scheint für *JAWS* (von Java über *MSAA*) nicht zu existieren – zumindest konnten keine dokumentierten Spuren auf eine solche Funktionalität gefunden werden.

3.2.6. Gute Bezeichnungen setzen

Labels beziehungsweise Bezeichnungen ausgesprochen bekommen ist eine Sache, gute Bezeichnungen setzen eine andere. Es gibt dazu drei wichtige Punkte zu beachten:[27]

- **Kurz und prägnant**: Auch wenn bei geübten Screenreader-Benutzern die Sprachgeschwindigkeit oft sehr hoch eingestellt ist, so sollten Labels kurz und prägnant sein. So können sie schnell wiedererkannt werden. Dieser Aspekt ist auch zuträglich, wenn das Ausgabegerät kein Screenreader sondern eine Braillezeile ist, die im Standardfall zwischen 12 und 40 Zeichen darstellt.

26 AVSpeechSynthesizer – iOS Developer Library – AVFoundation Framework Reference > AVSpeechSynthesizer Class Reference

27 Doug Kirschner – Tips and Tricks for Designing a Great Accessibility Experience for Your App – Channel 9 – ca. 0:24:50

- **Einzigartig, deutlich, nicht repetitiv**: Die Vermeidung von Wiederholungen ist wichtig, damit Screenreader-Benutzer die Elemente auch gut auseinanderhalten können. Kontextbezogene Pre- oder Suffixe sollten deshalb weggelassen werden. Die durch diese Einzigartigkeit geförderte Deutlichkeit macht es den Benutzern einfacher, sich zurechtzufinden.
- **Konsistenz**: Ein gesprochenes Label kann mehr enthalten, als was auf einem Control oder Element stehen würde – aber es muss konsistent sein mit dem, was auf dem Bildschirm zu sehen ist.

Eine gute Bezeichnung sagt genau, was passieren wird oder in welchem Zustand sich das bezeichnete Control derzeit befindet. Es gilt dabei zwischen Adverb und Adjektiv zu unterscheiden: Wird ein Adjektiv verwendet, so zeigt die Beschriftung den Zustand des bezeichneten Controls an, zum Beispiel «an/aus». Dabei muss klar werden, was «an» oder «aus» ist. Bei einem Verb wird erwartet, dass passiert, was im Label gesetzt ist. Gute Beispiele sind: Ausschalten, Abbrechen, Speichern.[28]

Bei der Verwendung von Adjektiv oder Verb gibt es kein entweder/oder, sondern es gilt, das Passende zum Passenden zu setzen. Handelt es sich um einen Button, der eine Funktion auslöst, die dadurch den Zustand des Buttons nicht ändert, ist ein Verb zu setzen. Hat der Button die Funktion eines Schalters, dann sollte der Zustand angegeben werden.

Zu berücksichtigen ist auch, welche Attribute oder Eigenschaften durch das Erkennen des Control-Objekts bereits vorgelesen werden. So macht das Label «Einschalt-Knopf» zum Beispiel keinen Sinn, weil die Rolle des Control-Objekts in der Regel vorgelesen wird: Dies würde dann etwa so ausgegeben: «Einschalt-Knopf-Schalter».

Beispiel aus dem Proof of Concept

Als Beispiel aus der *Proof of Concept*-Applikation kann dafür das Button-Pärchen «Web» und «Gehe zu» genommen werden. Im Code sind beide als Button instanziert, «Web» allerdings mit dem Style-Bit «toggle», «Gehe zu» mit dem Style-Bit «push».

Button-Instanzierung mit unterschiedlichen Style-Bits
Auszug aus: ch.chrissharkman.accessibility.rcp.application.poc.parts.BrowserViewPart

```
buttonWeb = new Button(toolbar, SWT.TOGGLE);
buttonWeb.setText("Web");
buttonGoto = new Button(toolbar, SWT.PUSH);
buttonGoto.setText("Gehe zu");
buttonGoto.setEnabled(false);
buttonGoto.setGrayed(true);
```

Das Style-Bit ändert die Aussage vom Screenreader: So wird beim Button «Web» folgendes ausgesprochen: «Surfmodus Kontrollfeld nicht aktiviert» respektive «Surfmodus Kontrollfeld aktiviert». Sobald der Button «Gehe zu» aktiv ist, wird folgendes ausgegeben: «U R L eingeben Schalter». Bei

28 Gute Labels setzen – Mail-Wechsel mit Selamet Aydogdu

beiden wird noch ergänzend gesagt, dass mit der Leertaste die Aktion ausgeführt werden kann.

Bei Abkürzungen – wie *URL* – ist es wichtig, diese mit Leerzeichen zwischen den Buchstaben zu schreiben, damit sie auch ausgesprochen werden wie die uns bekannten Abkürzungen.

Gute Labels gesetzt via accessibleName
Auszug aus: ch.chrissharkman.accessibility.rcp.application.poc > accessible.poc-accessible-properties.xml

```
<widget elementId="poc.browserpart.toolbar.btnweb"
     accessibleName="Surfmodus" defaultElement="true" />
<widget elementId="poc.browserpart.toolbar.btnurl"
     accessibleName="U R L eingeben" />
```

3.2.7. Kurz: Viele kleine Elemente ergeben ein Ganzes

Die Versuche haben gezeigt, dass kleine Details stimmen müssen, dass die *GUI*-Komponenten gut von einem Screenreader wiedergegeben werden können:

- Richtige Properties für MUIElemente, Widgets und HandledToolItems setzen
- AccessibleListener für Widgets mit getName-Methode setzen
- *GUI*-Aufbau soll Accessibility-Aspekte berücksichtigen
- Fokus-Eigenschaften der Widgets müssen stimmen
- Clevere, aussagekräftige Bezeichnungen wählen

3.3. Navigation verbessern

3.3.1. Ausgangslage

Ein Standard-Verhalten der Zielgruppe, um sich durch eine Applikation hindurch zu bewegen, ist das «Durchtabben». Durch das Drücken von **[tab]** springt der Fokus von Element zu Element, grundsätzlich in der Reihenfolge der Programmstruktur. Das funktioniert auf Windows in den meisten Programmen und in Browsern, beziehungsweise den dargestellten Webseiten, kann aber mitunter sehr aufwendig sein, wenn es viele fokussierbare Elemente gibt.

Auch in *RCP*-Applikationen ist diese Basis-Funktionalität zum «Durchtabben» vorhanden. Der Fokus in der «leeren» *Proof of Concept*-Applikation springt dadurch vom ersten Element der globalen Toolbar auf den ersten Part der aktiven Perspektive, setzt darin den Fokus auf den Reiter des PartStack, anschliessend auf den «Minimieren»-Button, dann auf den Browser-Part und dessen erstes Widget, welches den Fokus nehmen kann. Danach tabbt man über alle folgenden Widgets, kommt in das Browser-Widget und tabbt da über alle fokussierbaren Elemente im Browser-Widget, bis der Fokus am Ende wieder in der globalen Toolbar auf dem zuvor schon fokussierten HandledToolItem landet *(fig. 18)*.

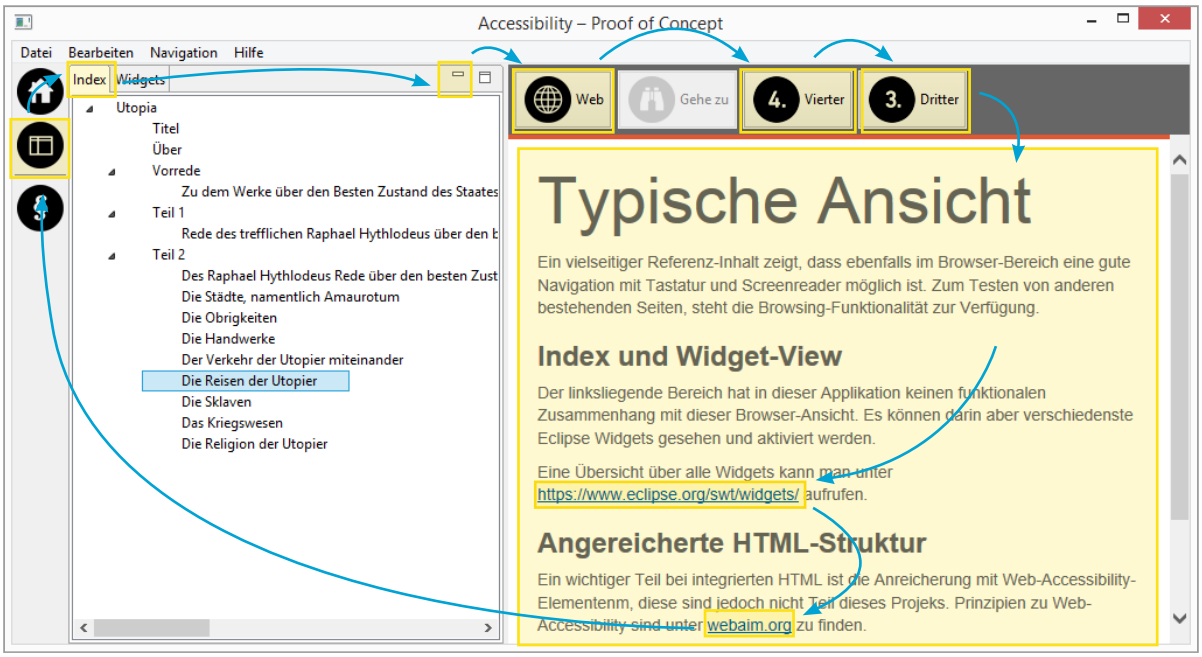


fig. 18 – *Standard-Tab-Reihenfolge im «leeren» Proof of Concept*

Die Problematik dieser Navigation liegt darin, dass für einen Benutzer nie ganz klar ist, wann er in einen neuen Kontext kommt und wann er einen verlässt. Selbst für einen Sehenden ist die in der Darstellung eingezeichnete Reihenfolge nicht zu 100% klar.

Wäre die Applikation absolut statisch mit immer dem gleichem Inhalt, so könnte man durch Routine irgendwann wissen, was wann kommt. Allein durch die Integration des Browser-Widgets, das dynamisch viele verschiedenen Inhalte und Inhaltsstrukturen annehmen kann, kann sich diese Routine oder Gewöhnung gar nicht einstellen.

3.3.2. Der Navigation Struktur verleihen

Einer gut strukturierten Navigation wurde deshalb eine hohe Priorität gegeben. Um sich in einer Applikation zurechtzufinden braucht die Zielgruppe «Landmarks», Orientierungspunkte.

Sinnbild

Als gutes Sinnbild kann man sich eine lange Linie vorstellen, auf der man entlanggeht. Unterwegs kommt man an Begriffen vorbei, die einem das erste Mal noch sehr fremd sind. Man läuft weiter und irgendwann kommt wieder der erste Begriff, dann der zweite und so weiter. Als der erste Begriff zum dritten Mal kommt, kann man sagen, jetzt wäre wieder der Anfang. Unterwegs kommen aber zu viele Begriffe, und dazu noch zum Teil verschiedene, sodass eine Orientation unterwegs auf dieser langen Linie schwierig bleibt.

Schneidet man diese lange Linie in mehrere Teile, gewinnt man zusätzliche «Startpunkte», die helfen, sich zu orientieren. Jede Linie wird kürzer, besser überblickbar. Zudem kann man sicher sein, dass man auf seiner Linie bleibt.

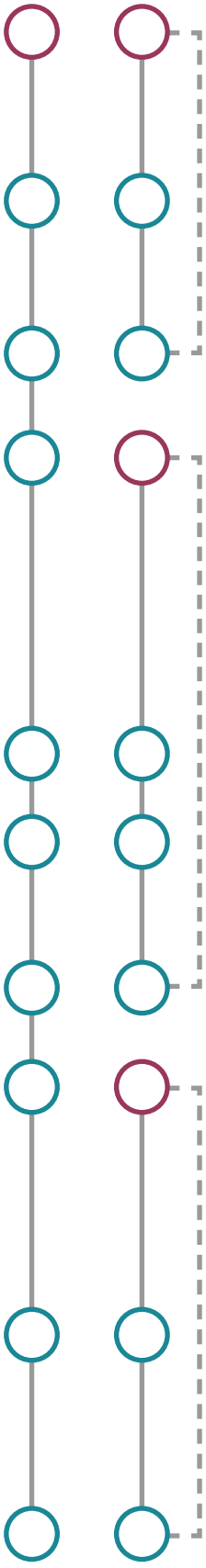


fig. 19 – *Sinnbild: Unterteilung der Linie schafft neue Orientierungspunkte*

Es wird ein Navigations-Niveau hinzugefügt, das nötige Landmarks erzeugt *(fig. 19)*.

Regionen, transparent für «Normal Benutzer»

Jede Perspektive der Benutzeroberfläche soll für das «Durchtabben» in Stücke zerlegt werden, auf eine Weise, die transparent bleibt für den «Normal Benutzer». Dies dient der Benutzerfreundlichkeit der Anwendung. Dieser Ansatz ist ähnlich wie die Verwendung von Regionen im *ARIA*-Konzept, nur dass die Regionen in *ARIA* noch detaillierter wahrgenommen werden können durch die Verteilung von Rollen wie «main», «navigation» oder «contentinfo»[29].

Eine saubere Iteration innerhalb eines solchen Stücks ist anzustreben: Gerade bei einem Browserbereich, der mit verschiedensten Inhalten gefüllt sein kann, sollte ein «Rausspringen» des Fokus vermieden werden. Das Prinzip gilt aber für jeden Bereich.

Primär-Tastaturnavigation nicht über Shortcuts

Ein anderer technisch möglicher Ansatz, den verschiedensten Funktionen oder Parts einzelne Tastenkombinationen zuzuordnen, wurde nicht verfolgt. Dies würde schnell zu einer Flut von Tastenkombinationen führen und wäre nicht zielführend, ja sogar abschreckend für neue Benutzer. Tastenkombinationen können ergänzend gesetzt werden, sie dürfen jedoch nicht die primäre Tastaturnavigations-Möglichkeit darstellen.[30]

3.3.3. Key Bindings und Binding Context

Um die Tastaturnavigation zu leiten, das «Tabben» und das Springen von Teil zu Teil zu kontrollieren, führte die erste Herangehensweise zu den Key Bindings und den Binding Contexts. Diese beiden Tabellen befinden sich im Application.e4xmi. In den Key Bindings können Kurztasten und Tastaturkombinationen gesetzt und mit einem Command verknüpft werden. Zur Steuerung, in welchem Kontext welche Tastaturkombinationen wie verarbeitet werden, können Binding Contexts definiert und zum Beispiel den Perspektiven oder Parts zugeordnet werden.

Der Weg über die Definition im Application.e4xmi ist die einfachste Art, globale Keylistener zu setzen – was ja für die Tastaturnavigation nötig ist. Es gab nur drei Probleme:

type:user-Tag

Einfach ist etwas, wenn man weiss, wie es geht: Damit gesetzte Key Bindings überhaupt funktionieren, muss unbedingt jedem Key Binding als Tag «type:user» gesetzt werden. Ansonsten werden Key Bindings im «BindingToModelProcessor» entfernt[31]. Zum Glück hatte Ralph Schuster einmal die Geduld aufgebracht, nach der Ursache zu suchen, die seit *eclipse* 4.3 besteht.

29 Using ARIA landmarks to identify regions of a page – www.w3.org
30 Christian Heimann – Blind am Computer: Bachelor-Vorarbeit, 1.5.2 Shortcut-Verwendung, Seite 11
31 Ralph Schuster – Eclipse/E4: Problem with Key Bindings – techblog.ralph-schuster.eu

Als diese Hürde genommen war, zeigte sich im Debugging das definierte Set von Key Bindings als erstes Element im Array der bindingTables der Applikations-Instanz (MApplication-Objekt). Ist die gleiche Kurztaste oder Tastenkombination in mehreren Elementen der bindingTables vorhanden, dann kommt einfach der erste Empfänger zum Zug. Findet sich kein Empfänger, im ersten Element, so wird der Befehl hierarchisch durchgereicht.

Modifikation während der Laufzeit nicht möglich

In der Applikations-Instanz sind nebst den Key Bindings auch die Binding Contexts ersichtlich. Eigentlich ideal – könnte man während der Laufzeit diese Arrays ergänzen oder modifizieren, was nicht möglich ist. Die jeweiligen Arrays sind durch den Zugriffsmodifikator *protected* geschützt. Ein nachträgliches Injizieren ist so nicht mehr möglich, ist aber nötig, wenn man von «ausserhalb» Tastaturkombinationen einfügen möchte.

Taste ist nicht gleich Taste

Waren die ersten zwei Schwierigkeiten umschifft, wurde klar, dass nicht alle Tasten und Tastenkombinationen über die Key Bindings abgefangen werden können. So werden unter anderem **[tab]** und **[alt + F4]** bereits vorher – wahrscheinlich von einer der Applikation übergeordneten *RCP*-Schicht – abgefangen und kommen deshalb gar nie bei den Key Bindings an.

Das Debugging hat zumindest gezeigt, dass sich diese Befehle nicht in den von *eclipse* bereits vorhandenen bindingTables befinden, die in der Hierarchie eigentlich auch unterhalb der eigenen Key Bindings stehen würden. Die gesamten Tastaturkombinationen, welche in *eclipse* unter Preferences > General > Keys zu finden sind, werden standardmässig jeder *RCP*-Applikation mitgegeben, auch wenn die dazugehörigen Commands nicht implementiert sind.

Der **[tab]**, die essenzielle Taste für die Tastaturnavigationsfähigkeit konnte also nicht über die Key Bindings abgefangen werden.

3.3.4. Traverse-Keys

In *SWT* gibt es die Bezeichnung «Traverse Key». Gemäss *SWT*-Dokumentation fallen unter anderem die folgenden Tasten unter diese Kategorie, direkt aufgeführt mit den dazugehörigen *SWT*-Integer-Bezeichnungen:

- **[tab]** traverse tab next
- **[shift + tab]** traverse tab previous
- **[Pfeil nach rechts oder unten]** traverse arrow next
- **[Pfeil nach links oder oben]** traverse arrow previous
- **[enter]** traverse return
- **[esc]** traverse escape

Die *SWT*-Klasse «Display» enthält die Methode addFilter, welche einen Listener jeglicher Art hinzufügen kann, der dann bei den entsprechenden Events reagiert. Um all die Traverse-Keys abzufangen, kann als Event-Type «traverse» gesetzt werden.

Setzen eines globalen Filters zum detektieren von Traverse-Key-Events.
Auszug aus Versuchsdatei

```
Display.getCurrent().addFilter(SWT.Traverse, new Listener() {
     public void handleEvent(Event e) {
          System.out.println("Traverse Key Event: " + e.keyCode);
     }
});
```

Zum Verarbeiten des Events in der Methode handleEvent() war es jedoch nicht möglich, die Traverse-Keys sauber über die *SWT*-Integer zu trennen. **[shift + tab]** wird mit «traverse tab previous» nicht erkannt. Eine normale Überprüfung mit keyCode- und stateMask-Wert ist deshalb nötig um zum Beispiel **[shift + tab]** sauber zu empfangen.

3.3.5. Event handling

In *eclipse RCP* werden Events von oben nach unten behandelt, vom Grössten zum Kleinsten. Hat zum Beispiel ein Composite ein darin liegendes Widget und weisen beide einen Key Listener auf, so wird bei einem Tastenanschlag nur der Key Listener vom ummantelnden Composite aufgerufen. Deshalb wird ein Event, der von einem Filter auf dem Display-Objekt abgefangen wird, ebenfalls nicht weitergegeben.

Immer wenn **[tab]** ins Spiel kommt, dann wird das Eventhandling jedoch vom Standard-Verhalten beeinflusst. Geplant war, dass der Wechsel von Navigations-Teil zu Navigations-Teil mit **[ctrl + tab]** gemacht werden kann – analog dem Sektionswechsel in Microsoft Word. Je nach Widget wurde aber das **[ctrl + tab]** nur als normaler **[tab]** interpretiert. Oder sogar als doppelter **[tab]** interpretiert. Woher genau die Events kommen, beziehungsweise wo sie «falsch» abgefangen werden, konnte nicht eruiert werden. Um dieser Problematik auszuweichen wurde als Taste für den Regions-Wechsel **[F6]** gesetzt, da diese Taste bereits für «Unterfensterwechsel» eine ähnliche Funktionsbelegung in anderen Applikationen hat. Da **[F6]** jedoch keinen Traverse-Key darstellt, wurde ein Filter für den KeyUp-Event gesetzt mit einem Listener für den Regions-Wechsel.

Um einem Widget das Standardverhalten bei **[tab]** zu unterbinden, kann ein TraverseListener gesetzt werden, der auf **[tab]** den doit-Wert *false* zurückgibt. Mit diesem Flag wird angegeben, dass diese Aktion nicht erlaubt ist.[32] Der **[tab]** wird somit nicht standardmässig interpretiert.

Unterbinden der TraverseEvent-Aktion mit TraverseListener
Auszug aus: ch.chrissharkman.accessibility.rcp.base.handler.SuppressTraverseListener.java

```
public class SuppressTraverseListener implements TraverseListener {

     @Override
     public void keyTraversed (TraverseEvent e) {
          if (e.detail == SWT.TRAVERSE_TAB_NEXT
               || e.detail == SWT.TRAVERSE_TAB_PREVIOUS) {
               e.doit = false;
          }
     }
}
```

32 Class KeyEvent – Documentation Eclipse Platform Release 4.4 – help.eclipse.org

3.3.6. Fokus setzen

Will man die Tastaturnavigation und deren Tab-Reihenfolge steuern, dann muss auch das Fokus-Setzen gemacht werden. Dazu gibt es folgende Punkte zu beachten:

- Generell kann der Fokus nur auf Controls gesetzt werden. Gewisse Controls wie «Label» sind davon ausgeschlossen.
- Beim Aktivieren eines Parts wird, sodenn das Argument grantFocus als true gesetzt wird, der Fokus automatisch auf das erste mögliche Control gesetzt.

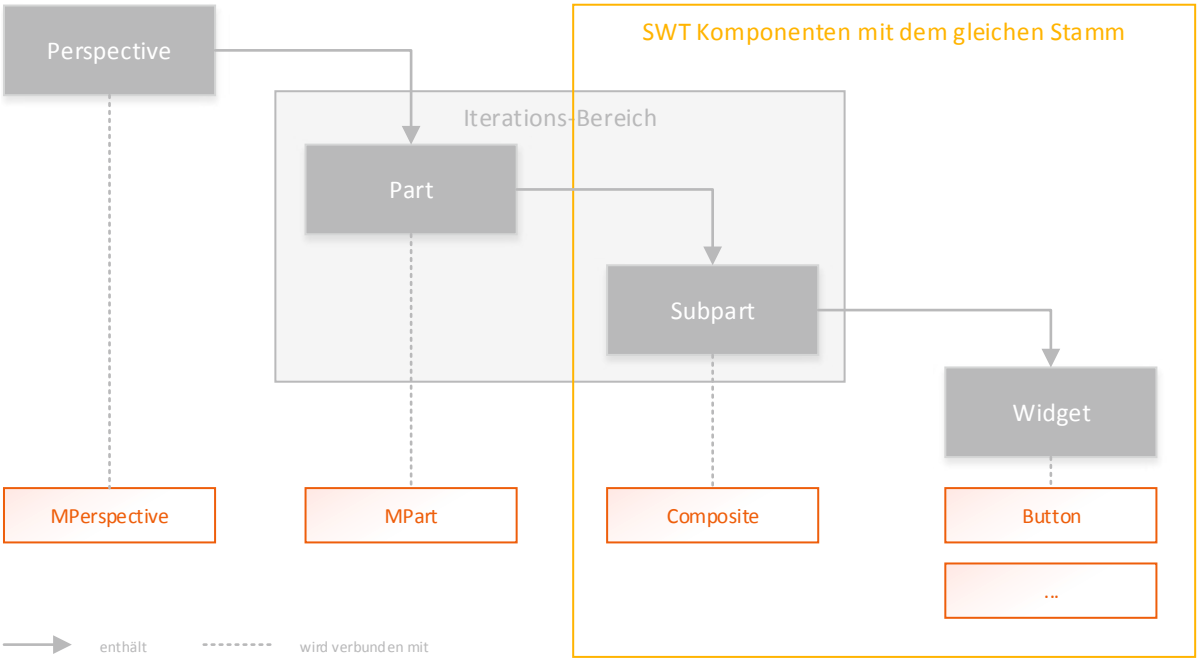
3.3.7. Niveau-Anpassung für Parts, Composites und Widgets

Ganz zu Beginn dieser Arbeit wurde unter «1.2.2. Zusammenhänge der Haupt-GUI-Elemente» auf Seite 8 aufgezeigt, dass die GUI-Struktur in einen Applikations- und einen SWT-Bereich aufgeteilt werden kann.

Zerlegt man nun eine Perspektive in verschiedene Bereiche um die Tastaturnavigation durch «Teilung der Linie» zu verbessern, kommt man schnell zur Problematik, dass ein Bereich manchmal ein ganzes Part ausmacht, dann wiederum nur aus einem Composite besteht. Für den Benutzer sollte es dabei transparent sein, ob eine logische Einheit in einem Part oder nur in einem Composite zusammengefasst ist, solange die Iteration darin Sinn macht und zu einer klareren Navigation beiträgt.

Da ein Composite zum gleichen Stamm wie zum Beispiel ein Button gehört, also beides Widgets sind, aber in der angedachten Navigation komplett verschiedene Rollen übernehmen, wurde der Nicht-*eclipse*-Begriff «Subpart» geschaffen. Er definiert die Struktur im SWT-Bereich zwischen Part und dem effektiv fokussier- oder klickbaren Element.

fig. 20 – *Die Niveaus des ViewTree mit den durch Benutzerfreundlichkeit bzw. Klassen-Stamm gegebenen Überschneidungen*



Das Schema *(fig. 20)* visualisiert die Überschneidung betreffend Iterations-Bereich und dem gleichen Klassen-Stamm von zwei in diesem Konzept unterschiedlich genutzten Widgets. Die Perspektiven werden von der Applikation wie gewohnt verwaltet. Parts und Subparts stellen die Bereiche dar, innerhalb welchen die Fokus-Iteration stattfindet, beziehungsweise welche mit einem Sprung zur nächsten «Region» gewechselt werden.

3.3.8. Tab-Order-Möglichkeit

Um innerhalb einer Shell oder eines Composite-Elements die Tab-Reihenfolge zu bestimmen, gibt es die Möglichkeit, die Controls in eine eigens definierte Reihenfolge zu setzen. Mit der definierten «tab list» wird bestimmt, in welcher Folge der Fokus von Control zu Control springt.

Setzen einer Tabreihenfolge von Controls auf das Shell-Element mit der Methode setTabList()
Auszug aus Versuchsdatei

```
Control[] tabOrder = new Control[] { buttonOne, buttonThree, buttonTwo };
shell.setTabList(tabOrder);
```

Dieser Ansatz wirkt schwerfällig und wird der Niveau-Angleichung von Part, Composite und Control auch nicht gerecht. Zugunsten einer «globaleren» Lösung wurde eine Verwendung dieser Möglichkeit nicht weiterverfolgt.

3.3.9. Handlichkeit und Einfachheit

All die in den vorangehenden Absätzen erwähnten Elemente und Techniken handlich zu machen war also das Ziel. Um nicht nur *beook* ein bisschen in Richtung Barrierefreiheit zu bringen, sondern auch andere RCP-Applikationen einfach ergänzen zu können. Der Weg dazu war nach einer Diskussion auch klar: Ein *Accessible Modul* sollte entwickelt werden, das mit geringem Aufwand das *Proof of Concept* wie auch *beook* mit den erprobten Accessibility-Techniken ausrüsten kann.

4. Das Accessible Modul

4.1. Übersicht

Die Suche nach Mitteln und Wegen zur Erzeugung von Barrierefreiheit hat gezeigt, dass Eingriffe und programmatische Elemente an den verschiedensten Orten nötig sind. Um diese möglichst effizient und automatisiert in einer RCP-Applikation zu verwalten, wurde der Ansatz über ein Plugin-Projekt gewählt.

Ein Plugin-Projekt kann bei jedem beliebigen e4-Application-Project hinzugefügt und entsprechend problemlos und mit wenig Aufwand auch wieder entfernt werden. Es ist für sich eine Einheit, die von der eigentlichen RCP-Applikation keine Kenntnisse hat. Die nötigen Eingaben ins Modul werden über

Dateien, Properties oder Parameter bei der Initialisation des Moduls gemacht, da diese von Applikation zu Applikation unterschiedlich sein können.

Mit dem *Accessible Modul* werden die in dieser Arbeit gesetzten Hauptaspekte – Sprachausgabe und strukturierte Navigation – zusammengeführt und organisiert. Betreffend Sprachausgabe kümmert es sich darum, für die *GUI*-Elemente die passenden «Label» zu setzen. Für die strukturierte Navigation sorgt es, in dem es die Reihenfolge von Elementen kennt und so die Fokus-Folge administriert.

Mit dem *Accessible Modul* kann die Barrierefreiheits-Fähigkeit an einem Ort gebündelt werden und zukünftige Erweiterungen können in allen *RCP*-Applikationen verwendet werden, welche das Modul schon integriert haben.

Um all die nötigen Zusatzinformationen aufzuführen, ist eine Struktur-Datei nötig. Für die zentrale Datei wurde das *XML*-Format gewählt. Dieses hat eine gute Lesbarkeit und kann mit in Java bestehenden Komponenten einfach ausgelesen werden. Zudem fügt sich eine *XML*-Datei gut in *beook* ein, da in vielen Programm-Teilen mit diesem Format gearbeitet wird.

4.1.1. Hauptbestandteile

Es wurden zwei Haupt-Verwaltungs-Klassen geschaffen, über die die Funktionalität des Moduls geregelt wird.

- **AccessibleManager**: Diese Klasse initialisiert die Eingabequelle für die Struktur-Datei, den ViewMediator und die globalen, für das *Accessible Modul* nötigen Listener. Über diese Klasse läuft auch das Setzen des ViewTrees und das Setzen von Accessibility Features – derzeit der «Labels» für den Screenreader. Der AccessibleManager selbst wird vom Framework *RCP* instanziert.
- **ViewMediator**: Eine Instanz dieser Klasse wird im AccessibleManager gesetzt. In diese Instanz wird das ViewTree-Objekt gesetzt, welches die Verbindung zum GUI enthält. Auch koordiniert diese Instanz mit Hilfe des ViewTrees die Navigation, das korrekte Fokus-Setzen, in Subparts und über Widgets.

4.1.2. Ablauf

Das Schema *(fig. 21)* stellt in groben Zügen den Ablauf dar, welcher zur Erstellung der gewünschten Screenreader-Sprachausgabe und einer strukturierten Navigation nötig ist.

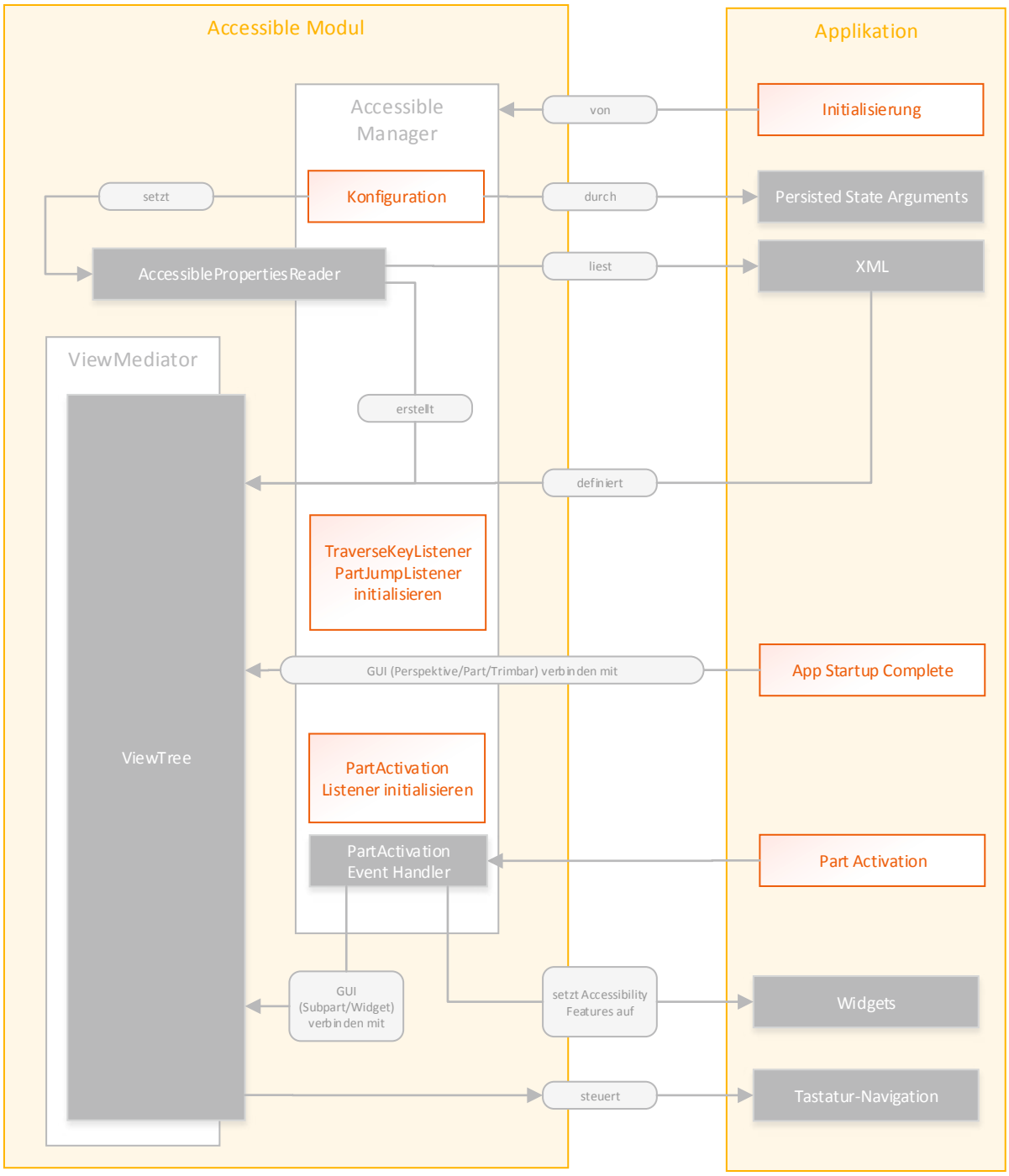


fig. 21 – *Nötiger Ablauf von der Initialisierung der Applikation bis zur strukturierten Navigation über Widgets mit gesetzten «Labels».*

UML

UML-Diagramm «4.6.4. Ablauf Initialisierung Accessible Modul» auf Seite 52

Initialisierung und Konfiguration

Beim Start der Applikation wird die AccessibleManager-Klasse instanziert und anschliessend die mit @PostConstruct annotierte init()-Methode aufgerufen.

Die init()-Methode holt die Konfigurations-Parameter von der Applikation und kann dadurch unter anderem den AccessiblePropertyReader initialisieren. Dieser ist zuständig für die Transformation der Struktur-Datei in ein Objekt-Modell. Der AccessiblePropertyReader kann je nach Datei-Typ der

Struktur-Datei unterschiedlich sein. Aktuell existiert zwar nur ein AccessiblePropertyReader für *XML*-Dateien, aber andere könnten an dieser Stelle hinzugefügt werden. Eine automatische Schaltung (switch) würde automatisch den korrekten Typ instanzieren.

«ViewTree» erstellen und verbinden

Durch das Parsen der Struktur-Datei kann das Objekt-Modell, der in diesem Modul sogenannte ViewTree, erstellt und im ViewMediator gesetzt werden.

Die entsprechenden Elemente können im ViewTree-Objekt nach dem Empfangen vom «App Startup Complete»-Event mit den globalen, bereits bestehenden *GUI*-Elementen verbunden werden. Perspektiven, Parts und die globalen Toolbars stehen ab diesem Moment bereit.

Um die restlichen Teile des *GUI* mit dem ViewTree zu verbinden, muss die Part-Aktivierung abgewartet werden. Denn Composites und Widgets werden erst beim Öffnen beziehungsweise Aktivieren eines Parts erzeugt. Das *RCP*-Framework sendet beim Aktivieren eines Parts einen Event, welcher vom AccessibleManager gehört wird. Darauf gibt dieser dem ViewMediator den Befehl, die restlichen *GUI*-Elemente mit dem jeweiligen Teil im Model zu verbinden. Ab diesem Moment ist der ViewTree komplett, um die strukturierte Navigation korrekt zu leiten.

UML

UML-Diagramm «4.6.5. Ablauf Part-Activation» auf Seite 52

Accessible-Features setzen

Durch den Part-Activation-Event wird, sobald der entsprechende Part im ViewTree komplett verbunden ist, die Methode addAccessibleFeaturesWithin() aufgerufen. Diese fügt den Subparts und Widgets, welche im ViewTree Bestandteil des mitgegebenen Parts sind, folgende Elemente hinzu:

- **Für Widgets**: AccessibleListener für die Sprachausgabe des Screenreaders und ein Listener zur Unterdrückung der Standard-Aktion bei **[tab]** sofern in der Struktur-Datei für das *GUI*-Element kein Attribut «needDefaultTabAction» mit dem Wert *true* gesetzt ist.
- **Für Subparts**: FocusListener zum Detektieren, wann ein Part verlassen wird, um die Iteration innerhalb des Subparts sicher zu stellen.

Bereits nach der Verbindung von globalen *GUI*-Elementen wurden die Accessible-Features für Parts und HandledToolItems gesetzt:

- **Für Parts**: Label für die Sprachausgabe des Screenreaders. Ein bestehendes Label wird überschrieben.
- **Für HandledToolItems**: Tooltip für die Sprachausgabe des Screenreaders. Ein bestehender Tooltip wird überschrieben.

Globale Listener setzen

Die globalen Listener für das Abfangen von Traverse-Keys und **[F6]** werden nach der Erstellung des ViewTrees gesetzt. Diese werden, um global zu wirken, der aktuellen Display-Instanz als «Filter» hinzugefügt. Um zu steuern, welche wirklich aktiv sind – wenn man dieses Verhalten zum Beispiel für

eine bestimmte Perspektive nicht haben möchte – kann man diese Listener aktivieren beziehungsweise deaktivieren.

4.1.3. ViewTree – das Objekt-Modell der gewünschten Navigations-Struktur

Eine *RCP*-Applikation kann vielschichtig aufgebaut sein betreffend Perspektiven, PartStacks, Parts, SashParts und anderen strukturellen *GUI*-Elementen für eine *RCP*-Applikation. Dieser Baum kann im Application.e4xmi gesehen werden, sodenn er in der Application.e4xmi-Datei definiert ist und nicht programmatisch aufgebaut wird.

UML-Diagramm
«4.6.1. ViewTree-Modell» auf Seite 49

Für den ViewTree, das Hauptelement im ViewMediator, wurde diese Struktur massiv vereinfacht und wie unter «3.3.7. Niveau-Anpassung für Parts, Composites und Widgets» auf Seite 35 beschrieben auf vier Niveaus reduziert. Dieses Objekt-Modell stellt gleichzeitig die gewünschte Navigations-Struktur dar, enthält aber auch die nötigen Attribute zum Setzen der korrekten Accessible-Features. Für jeden Fokus-Sprung wird auf dieses Modell zurück gegriffen, um das nächste, zu fokussierende Element ausfindig zu machen. Die Verbindung zwischen *GUI* und dem Modell befindet sich im jeweiligen ViewTree-Element selbst.

Der ViewTree ist aus folgenden Klassen zusammengestellt:

- **AccessiblePerspective**: enthält eine ArrayList mit AccessibleParts
- **AccessiblePart**: enthält eine ArrayList mit AccessibleSubpart
- **AccessibleSubpart**: enthält eine ArrayList mit AccessibleWidgets
- **AccessibleWidgets**: sind die «Blätter» des ViewTree

4.1.4. Konstanten

Alle Konstanten – über das Modul hinweg gleichbleibende Werte – für Events, Klassenbezeichnungen und Attribute wurden in der Accessible-Constants-Klasse gesammelt. Die mit public static final ausgezeichneten Werte können von überall her abgerufen werden. Dieses Vorgehen erlaubt rasches und saubere Änderungen, sollte einer dieser Werte eine Anpassung verlangen.

Um sich die Möglichkeit auf die Verwendung von eigenen Methoden in den AccessibleConstants offen zu lassen, wurde die Entscheidung gefällt, die AccessibleConstants als Klasse und nicht als Interface zu definieren.

4.2. XML-Input – die Struktur-Datei

4.2.1. XML-Beispiel aus dem Proof of Concept

Um den Aufbau des XML direkt zu verdeutlichen, folgt eine Abbildung der für die *Proof of Concept*-Applikation erstellten *XML*-Datei. Zur besseren Lesbarkeit wurden die Perspektiven auf eine reduziert.

Gekürztes XML-Beispiel
Auszug aus: ch.chrissharkman.accessible.rcp.application.poc > accessible > poc-accessible-properties.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<accessible>
     <keybindings>
          <!-- Global key bindings are described here: parsing would be possible extension in future -->
          <keybinding action="focusOnNextWidget" keySequence="TAB" />
          <keybinding action="focusOnPreviousWidget" keySequence="SHIFT+TAB" />
          <keybinding action="focusOnNextPart" keySequence="F6" />
          <keybinding action="focucsOnPreviousPart" keySquence="F6+TAB" />
          <keybinding action="escape" keySequence="ESC" />
     </keybindings>
     <dialogs>
          <!-- Possibility to set dialogs and its sequence: possible extension in future -->
     </dialogs>
     <perspectivestack>
          <perspective elementId="poc.perspective.home" defaultElement="true">
               <part elementId="poc.home.browserpart" label="Browserbereich mit Einleitung">
                    <subpart elementId="poc.browserpart.toolbar">
                         <widget elementId="poc.browserpart.toolbar.btnweb"
                              accessibleName="Surfmodus" defaultElement="true" />
                         <widget elementId="poc.browserpart.toolbar.btnurl" accessibleName="U R L eingeben" />
                         <widget elementId="poc.browserpart.toolbar.btnTest3"
                              accessibleName="Dieser Button kommt als drittes Element in der Tabreihenfolge und
                                   hat keine Funktion." />
                         <widget elementId="poc.browserpart.toolbar.btnTest4" accessibleName="Dieser Button
                              kommt in der Tabreihenfolge als letztes Element und hat keine Funktion." />
                    </subpart>
                    <subpart elementId="poc.browserpart.browser" defaultTabAction="true" />
               </part>
          </perspective>
          <!-- more perspectives -->
     </perspectivestack>
     <trimbars>
          <toolbar elementId="poc.toolbar" defaultElement="true">
               <handledtoolitem elementId="poc.handledtoolitem.home"
                    tooltip="Home: Willkommenstext" defaultElement="true" />
               <handledtoolitem elementId="poc.handledtoolitem.content"
                    tooltip="Inhalt: Übersichts- und Lesebereich" />
               <handledtoolitem elementId="poc.handledtoolitem.imprint"
                    tooltip="Impressum: Angaben zur Urheberschaft" />
          </toolbar>
     </trimbars>
</accessible>
```

4.2.2. Struktur

Eine Baumstruktur lässt sich in *XML* gut abbilden, darum fiel die Wahl als Struktur-Datei auf dieses Format. In der Datei werden auf dem ersten Niveau vier Bereiche abgebildet:

- Keybindings
- Dialogs
- PerspectiveStack
- Trimbars

Davon sind derzeit nur die Teile «PerspectiveStack» und «Trimbars» in Gebrauch, die anderen zwei Elemente stehen zum zukünftigen Funktionsausbau bereit.

Im PerspectiveStack wird die Navigations-Struktur abgebildet, wie sie auch unter «3.3.7. Niveau-Anpassung für Parts, Composites und Widgets» auf

Seite 35 beschrieben ist. Dabei können die Elemente folgende Attribute enthalten:

- **Perspective**: elementId, defaultElement
- **Part**: elementId, label, defaultElement
- **Subpart**: elementId, defaultTabAction
- **Widget**: elementId, accessibleName, defaultTabAction, defaultElement

Unter dem Knoten «Trimbars» werden die globalen Toolbars aufgeführt – im Fall des *Proof of Concept* ist dies nur eine. Dabei können die Elemente folgende Attribute enthalten:

- **Toolbar**: elementId, defaultElement
- **HandledToolItem**: elementId, tooltip, defaultElement

Einziges, obligatorisches Attribut für alle oben aufgeführten Elemente ist die elementId, ohne welche die *GUI*-Elemente nicht mit dem Modell verbunden werden könnten.

Die Bezeichnungen «PerspectiveStack» und «Trimbars» wurden übernommen, um den Entwicklern, die in der *RCP*-Umgebung arbeiten, sofort Anhaltspunkte zu geben, um welches Pendant es sich handelt.

4.2.3. Matching über ID

Perspektiven und Parts des *GUI* besitzen standardmässig bereits eine Identifikationskennzeichnung, eine *ID*. Dadurch können die existierenden *GUI*-Elemente den im ViewTree vorhandenen AccessiblePerspective beziehungsweise AccessibleParts zugeordnet werden.

Composites und Widgets haben hingegen grundsätzlich kein klares Identifikationsmerkmal wie die Perspektiven und Parts. Um das *Matching* zu ermöglichen, muss dehalb bei der Erstellung der Widgets diesen eine *ID* gegeben werden. Dazu wird das Data-Objekt, welches sich in jedem Widget befindet, verwendet. Darin wird das Schlüssel-Wert-Paar «accessibleId: *ID*-String» gesetzt. Dieser Eingriff in den Code der Applikation ist unumgänglich.

Setzen der AccessibleId in das Data-Objekt des Widgets
Auszug aus Versuchsdatei

```
Button button = new Button(composite, SWT.PUSH);
button.setData("accessibleId", "poc.browserpart.button");
```

Zur einfachen Verwendung wird vom Modul im AccessibleHelper die Methode «bindAccessibleId» zur Verfügung gestellt. Als Argumente werden das *GUI*-Element und die gewünschte Id als String mitgegeben.

Aufruf der bindAccessibleId-Methode zum Setzen der ID
Auszug aus: ch.chrissharkman.accessible.rcp.application.poc.parts.BrowserViewPart.java

```
// set the id of the subpart and its widgets
AccessibleHelper.bindAccessibleId(toolbar, "poc.browserpart.toolbar");
AccessibleHelper.bindAccessibleId(buttonWeb, "poc.browserpart.toolbar.btnweb");
AccessibleHelper.bindAccessibleId(buttonGoto, "poc.browserpart.toolbar.btnurl");
```

4.3. Funktionalität des Moduls

4.3.1. «Labels» setzen

Für die Sprachausgabe der Screenreader werden direkt an den richtigen Stellen die nötigen und nützlichen «Labels» gesetzt. Diese sind je nach *GUI*-Element Label (Part), Tooltip (HandledToolItem) oder «getName»-Methode eines AccessibleListeners (Widgets).

Um Labels und Tooltips zu setzen, werden beim entsprechenden *GUI*-Element die bestehenden Werte einfach überschrieben. Part und HandledToolItem bieten dafür bereits die entsprechenden Methoden setLabel() und setTooltip().

Bei den Widgets werden – um die höchste Priorität für die Ausgabe des Screenreader zu haben – AccessibleListener gesetzt.

Hinzufügen eines AccessibleListeners zum Control eines gegebenen AccessibleWidgets
Auszug aus: ch.chrissharkman.accessible.rcp.base.AccessibleManager.java

```
/**
 * Method to set an AccessibleListener to the connected GUI object control
 * that contains the getName method. This method returns the given accessibleName
 * description to screenreaders.
 * @param widget
 */
private void setAccessibleListenerGetNameTo(AccessibleWidget widget) {
     logger.info("in getNameTo: "
          + widget.getGuiObject().getClass().getSimpleName());
     if (widget.getGuiObject() instanceof Control) {
          Control control = (Control) widget.getGuiObject();
          control.getAccessible()
               .addAccessibleListener(new AccessibleAdapterWidget(widget));
     }
}
```

Für die Instanz des AccessibleListener wurde eine neue Klasse «AccessibleAdapterWidget» erstellt. Diese erweitert die AccessibleAdapter-Klasse, somit müssen die unnötigen Methoden «getHelp», «getDescription» und «getKeyboardShortcut» nicht aufgeführt werden.

Bei der Instanzierung des AccessibleAdapterWidget-Objekts wird das AccessibleWidget mitgegeben und als Feld gesetzt. Die Methode getName ruft daher immer den dem AccessibleWidget gesetzten accessibleName auf und bleibt somit dynamisch, sollte dieser nachträglich einmal geändert werden.

getName-Funktion, welche jedem Widget mit accessibleName zugeordnet wird
Auszug aus: ch.chrissharkman.accessible.rcp.base.handler.AccessibleAdapterWidget.java

```
/**
 * Method to set as result the accessible name of the
 * widget set as property of this instance. The property
 * "accessibleName" is validated with the accessibleExtension of actual
 * accessibleManager.
 */
@Override
public void getName(AccessibleEvent e) {
     String fullAccessibleName = AccessibleManager.instance()
          .getAccessibleExtension()
          .getAccessibleNameString(this.widget.getAccessibleName());
          e.result = fullAccessibleName;
     }
```

UML-Diagramm «4.6.2. Navigations-Listener» auf Seite 50

4.3.2. Tastaturnavigation leiten

Die Tastaturnavigation, sprich das kontrollierte Springen des Fokus von Control zu Control wie auch das gesteuerte Springen von Bereich zu Bereich, stellt die zweite Funktion des *Accessible Moduls* dar.

Die global gesetzten Listener für **[tab]**, **[F6]** und **[esc]** senden die folgenden Events:

- focus next widget
- focus previous widget
- focus next subpart
- focus previous subpart
- focus escape

Im ViewMediator werden diese Events empfangen. Dann wird ermittelt, wo der Fokus derzeit gesetzt ist und anschliessend die Methode aufgerufen, um den Fokus auf das gewünschte Widget, bezeihungsweise den gewünschten Part zu setzen.

Handling des «focus next widget» Events
Auszug aus: ch.chrissharkman.accessibility.rcp.base.ViewMediator.java

```
// Set handler for focus on next widget event: then set focus on next widget in the viewTree,
// if accessibleId is set on control. If not, default behavior produced.
AccessibleManager.getBroker().subscribe(AccessibleConstants.FOCUS_NEXT_WIDGET, new EventHandler() {

     @Override
     public void handleEvent(Event event) {
          lastFocusWidgetEventTimestamp = System.currentTimeMillis();
          try {
               AccessibleElement activePart = getAccessibleElementBy(partService.getActivePart()
                    .getElementId());
               Control control = Display.getCurrent().getFocusControl();
               if (control.getData(AccessibleConstants.KEY_ID_GUI_ELEMENT) != null) {
                    AccessibleElement focusedWidget = getAccessibleElementBy(
                         control.getData(AccessibleConstants.KEY_ID_GUI_ELEMENT).toString(),
                         activePart.getAccessibleChildren());
                    setFocusOnNextWidget(activePart, focusedWidget);
               }
          } catch (Exception e) {
               logger.warn("Exception in focus next widget handleEvent: " + e
                    + " - Default eventhandler will handle it.");
          }
     }
});
```

Damit das Fokus-Setzen korrekt funktioniert, muss das Standard-Verhalten von Controls beim Betätigen von **[tab]** unterdrückt werden. Nur so kommt der gesetzte globale Filter für den Traverse-Key **[tab]** zum Zug. Der SuppressTraverseListener wird gesetzt, sofern das AccessibleWidget-Modell kein TabAction-Wert mit *true* aufweist.

SuppressTraverseListener mit dem keyTraversed-Eventhandler, der die Standardaktion unterdrückt.
Auszug aus: ch.chrissharkman.accessibility.rcp.base.handler.SuppressTraverseListener.java

```
public class SuppressTraverseListener implements TraverseListener {

     @Override
     public void keyTraversed (TraverseEvent e) {
          if (e.detail == SWT.TRAVERSE_TAB_NEXT || e.detail == SWT.TRAVERSE_TAB_PREVIOUS) {
               e.doit = false;
          }
     }
}
```

So wird die Verantwortung der Tab-Steuerung vom Control entnommen und der globale Traverse-Key-Filter kommt zum Zug.

4.3.3. Ebene über der Applikation

Die Funktionalität des *Accessible Moduls* platziert sich als Schicht über der Applikation: Tastenanschläge werden gefiltert, Listener auf Controls und Parts gesetzt. Dies vereinfacht die Implementation und senkt die Hürde zur Verwendung. Einzig das Ändern von Tooltips und Labels passiert auf der Applikations-Ebene.

Wie sich in der Suche nach Mitteln zur Barrierefreiheit gezeigt hat, fördert eine bestimmte GUI-Struktur die Barrierefreiheit. Das *Accessible Modul* macht aber keine Eingriffe in den Aufbau der GUI-Struktur. Es avanciert in dieser Form also nicht zum Wundermittel für alle Fälle.

4.4. «How to» – Anleitung zur Implementation

4.4.1. Grundsatz

Um die Implementation des *Accessible Moduls* so einfach und losgelöst wie möglich zu gestalten, wurde versucht, die Eingriffe in die bestehende Applikation so gering wie möglich zu halten.

Die Hauptkonfiguration findet über mehrere Parameter im Application.e4xmi statt. Durch die gegebene Struktur der Benutzeroberflächen von RCP-Applikationen ist eine volle Integration aber nur durch Ergänzungen in den bestehenden Klassen der Applikation möglich. Wiedererkennungs-Merkmale und Zugänglichkeit müssen geschaffen werden.

Um das Modul zu verwenden, beginnt man damit, es im plugin.XML als Required Plug-in zu setzen. Anschliessend kann die Konfiguration im Application.e4xmi vorgenommen werden.

4.4.2. Konfiguration im Application.e4xmi

Durch die Konfiguration über die Application.e4xmi-Datei können alle nötigen Parameter gesetzt werden. Dazu müssen in den Add-ons der Applikation die beiden Klassen AccessibleManager und ViewMediator des Modul-Bundles hinzugefügt werden *(fig. 22)*.

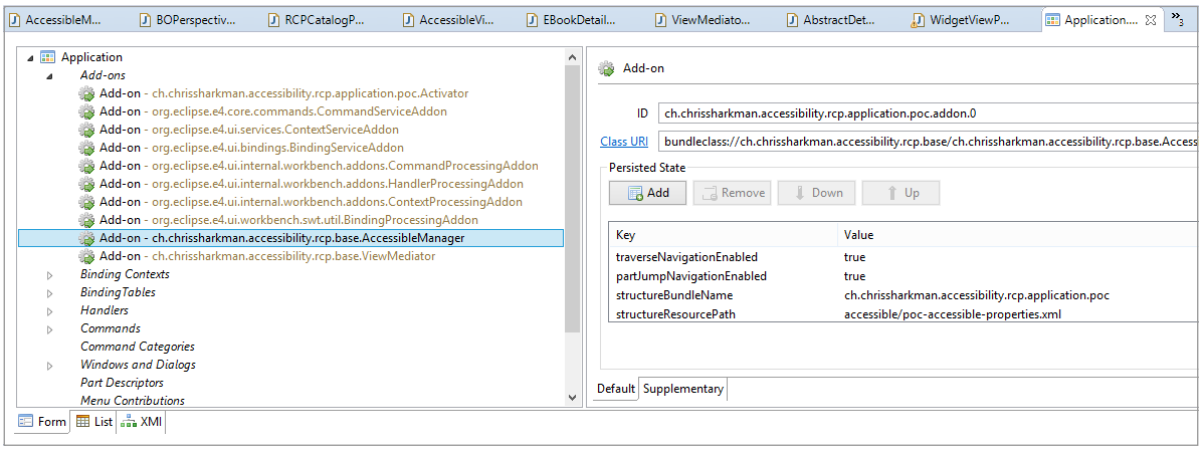


fig. 22 – *Application.e4xmi mit hinzugefügten Add-ons und gesetzten «Persisted State»-Werten*

Unter «Persisted State» des Add-on «AccessibleManager» sind folgende Parameter zu setzen:

- **structureBundleName**: Der Name des Bundles worin sich die Struktur-Datei befindet. Dieser Wert ist obligatorisch.
- **structureResourcePath**: Die Pfadangabe im Bundle (ab Root) zur Struktur-Datei. Dieser Wert ist obligatorisch.
- **traverseNavigationEnabled**: Wenn das Modul die komplette TraverseNavigation (Subpart-Berücksichtigung, Reihenfolge Widgets) berücksichtigen soll, dann kann hier der Wert *true* gesetzt werden. Wird dieser Wert nicht gesetzt, so ist er automatisch *false*.
- **partJumpNavigationEnabled**: Wenn das Modul Part-Jumps ermöglichen soll, dann wird kann hier der Wert *true* gesetzt werden. Wird dieser Wert nicht gesetzt, so ist er automatisch *false*.
- **accessibleExtensionBundleName**: Zum Erstellen einer Instanz der AccessibleExtension-Klasse wird der komplette Bundle-Name benötigt. (Details unter «5.2. Extension-Mechanismus: Anpassung durch Subklasse» auf Seite 54)
- **accessibleExtensionCanonicalClassName**: Der kanonische Klassen-Name wird für den Extension-Mechanismus ebenfalls benötigt.

Sind sowohl traverseNavigation wie auch partJump aktiviert, dann wird zum Springen von Part zu Part die gegebene Modell-Struktur mit den Subparts berücksichtigt. Ist nur partJump aktiviert, dann wird anhand der vorhandenen Parts die programmatische Folge von Parts durchlaufen. Subparts, auch wenn sie im Modell definiert wären, hätten so keinen Einfluss auf das Navigationsverhalten.

Der ViewMediator muss zu den Add-ons hinzugefügt werden, um die Klasse an den EventBus zu koppeln.

4.4.3. Eingriffe in bestehende Klassen

Das *Accessible Modul* versteht sich als Ebene über der Applikation welche für eine verbesserte Barrierefreiheit sorgt. Ganz ohne Eingriffe in bestehende Klassen der Applikation ist ein Fukitonieren aber nicht möglich.

Identifikations-Attribut setzen auf Widgets und Composites

Bei der Erzeugung von *GUI*-Elementen in der Applikation muss ein Identifikationsmerkmal gesetzt werden, damit ein Verbinden mit dem ViewTree möglich ist. Der AccessibilityHelper stellt dafür die Methode bindAccessibleId zur Verfügung, damit man sich diesbezüglich nur um diesen einen Aufruf kümmern muss.

Interface «AccessibleView» implementieren

Zwischen Parts und deren Inhalt (Composites, Widgets) gibt es einen Graben, über den man mit den verfügbaren Funktionen nicht hinweg kommt. Es handelt sich dabei um Klassen mit komplett unterschiedlichem Ursprung. Die Brücke schlägt die Implementation des AccessibleView-Interfaces in den ViewPart-Klassen.

Die ViewPart-Klassen enthalten die Konstruktionsanweisungen der *GUI*-Elemente. Wird ein Part aufgerufen, wird der darin gesetzten ViewPart-Klasse die @PostConstruct-Methode createComposite(Composite parent) aufgerufen. Um nun den Inhalt eines Parts zugänglich zu machen, muss das dabei als Parameter erhaltene parent als Property (viewComposite) der Klasse gesetzt werden.

So kann die *GUI*-Struktur im Modell sauber abgebildet werden. Es besteht ein Zugang über das Part zu dessen Inhalt.

Log4j in Activator-Klasse initialisieren

Da im Modul mit dem Log4j-Logger (Version 1.2) geloggt wird, muss dieser in der Activator-Klasse der Applikation initialisiert werden. Dies muss geschehen, bevor das Add-on «AccessibleManager» aufgerufen wird.

Die Log4j-Programmbibiliothek gehört nicht zu den Standardbibliotheken von *eclipse-RCP*-Applikationen. Vorgängig bedingt es also, Log4j zu importieren. Sollte in der Applikation eine andere Logging-Implementation gesetzt sein, so ergeben sich dadurch unnötige Parallelen. Eine Commons-Logging-Implementation wäre daher eine Möglichkeit, dieses Element in einem nächsten Schritt flexibler zu machen.

Sollten keine Logs ausgegeben werden, kann zur Überprüfung der korrekten Konfiguration die Initialisierung des log4j in der Konsole ausgegeben werden. Dazu muss in den VM arguments der Launch-Konfiguration der Applikation das Argument -Dlog4j.debug gesetzt werden.

4.5. Proof of Concept mit integriertem Accessible Modul

4.5.1. Resultat

In dieser geschlossenen, auf die Kern-Elemente reduzierten Applikation, funktioniert das *Accessible Modul* gut. Labels, Tooltips und getName-Listener werden wie in der Struktur-Datei vorgegeben gesetzt. Ebenso wird die dort beschriebene Reihenfolge berücksichtigt.

fig. 23 – *Proof of Concept mit integriertem Accessible Modul in Verwendung: tb.chrissharkman.ch/poc*

Durch die «neue» Tastaturnavigation lässt sich wie vorgesehen von Part zu Part springen. In komplett definierten Parts sind auch die Iterationen innerhalb der Subparts korrekt. Dies ist besonders beim Browsen ein wichtiges Element. Rückwärts-Sprünge zu vorhergehenden Parts und Widgets wurden als zweite Priorität eingestuft und sind daher noch nicht implementiert.

Wie die Anwendung des *Proof of Concept* aussieht und vor allem wie sie mit JAWS tönt, kann im Video erlebt werden *(fig. 23)*. Zusätzlich steht die Applikation als .exe-Datei auf der beiliegenden CD bereit und kann getestet werden. Von JAWS gibt es eine Demo-Version, die kostenlos installiert werden kann.

4.5.2. Navigation: Unterschiede zur konzeptionellen Vorlage

Betreffend Navigation konnten zwei Elemente aufgrund der GUI-Struktur nicht so umgesetzt werden, wie diese im Empfehlungskatalog beschrieben wurden.

Platzierung der globalen Toolbar in der Navigationsabfolge

Die Leiste am linken Rand ist in *beook* für die Auswahl der Bücher, des Portals oder des Webbrowsers zuständig – in *eclipse*-Terminologie ausgedrückt: für die Auswahl der Perspektive.

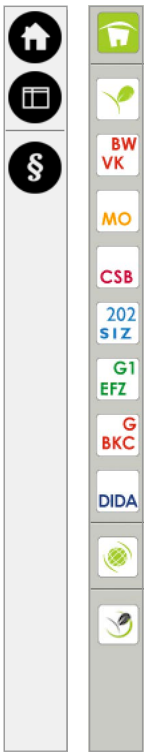


fig. 24 – *Die globale Toolbar des Proof of Concept (links) und von beook (rechts)*

Programmatisch hat diese Auswahl durch die Klassen MToolbar und MHandledToolItem einen anderen Aufbau als die im GUI definierten Subparts, welche Composites sind. Beim Setzen der XML-Datei für das *Proof of Concept* wurde auch eine störende Redundanz ersichtlich. Obwohl sich die globale Toolbar ausserhalb der Perspektiven befindet, wurde sie zuerst als Subpart dann doch in jeder Perspektive aufgeführt.

Dadurch wurde ersichtlich, dass dieses Element in der Subpart-Reihenfolge nicht korrekt ist. Es ist, wie der Name schon sagt, ein globales Element, und steht ein Niveau über den Perspektiven. Dazu kommt, dass im normalen *beook*-Gebrauch das Wechseln der Perspektive ein eher seltener Usecase ist. Darum war es auch vom Benutzerstandpunkt her korrekt, die globale Toolbar aus der Subpart-Reihenfolge zu entfernen.

Verdeutlicht hat sich dadurch aber auch, dass die globale Toolbar der Ort ist, welcher über die escape-Taste angesteuert werden muss. Es ist die «rettende», übergeordnete Stufe.

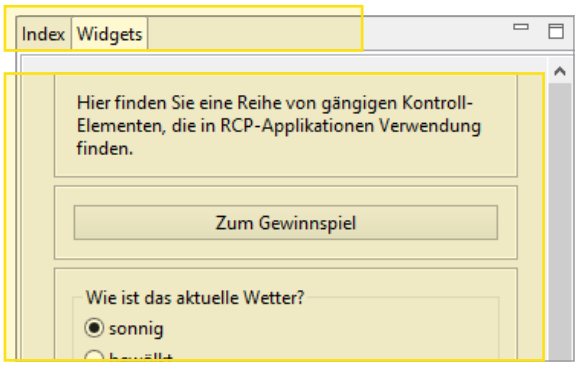


fig. 25 – *Im Empfehlungskatalog angedachte Unterteilung, die so nicht realisiert werden kann.*

Kein Iterationsbereich über Register

Registerkarten werden durch ein PartStack-Element erzeugt, das einem oder mehreren Parts übergeordnet ist. Die verschiedenen, eingefügten Parts werden automatisch in Form gebracht. Die Reiter, auch Tabs genannt, sind deshalb ein Anhängsel des Parts und können nicht als separates Element erfasst werden. Die im Empfehlungskatalog[33] angedachte Unterteilung *(fig. 25)* kann deshalb so nicht umgesetzt werden.

33 Christian Heimann – Blind am Computer: Bachelor-Vorarbeit, 4.2.3 Tab-Navigation von Element zu Element, Seite 27

4.6. Überblick mit UML

4.6.1. ViewTree-Modell

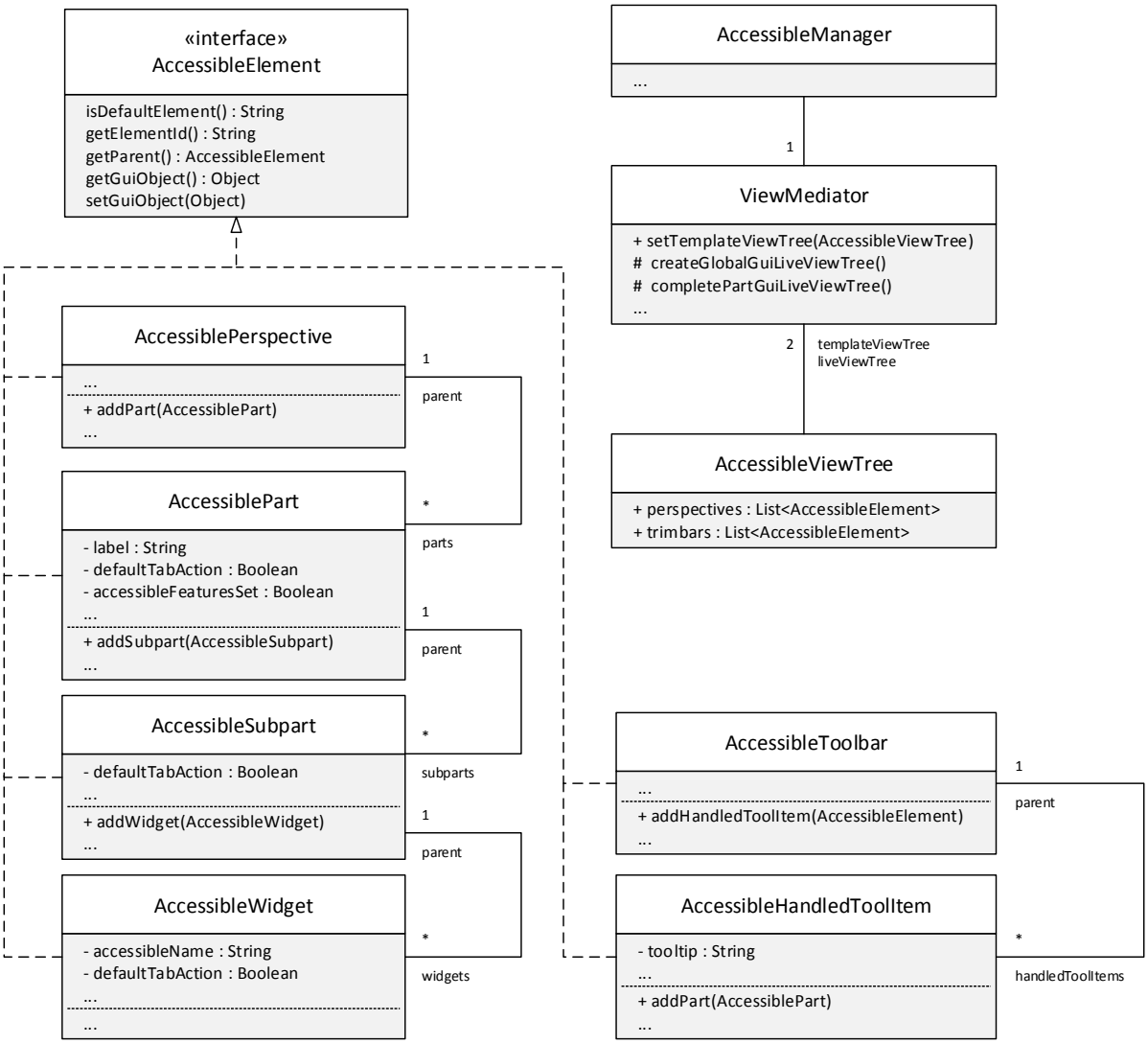


fig. 26 – *ViewTree-Modell (UML)*

Die im ViewMediator gesetzten AccessibleViewTrees sind ein zentrales Element zur Steuerung des Fokus via Tastaturnavigation. Weitere Ausführungen zum ViewTree unter «4.1.3. ViewTree – das Objekt-Modell der gewünschten Navigations-Struktur» auf Seite 40.

4.6.2. Navigations-Listener

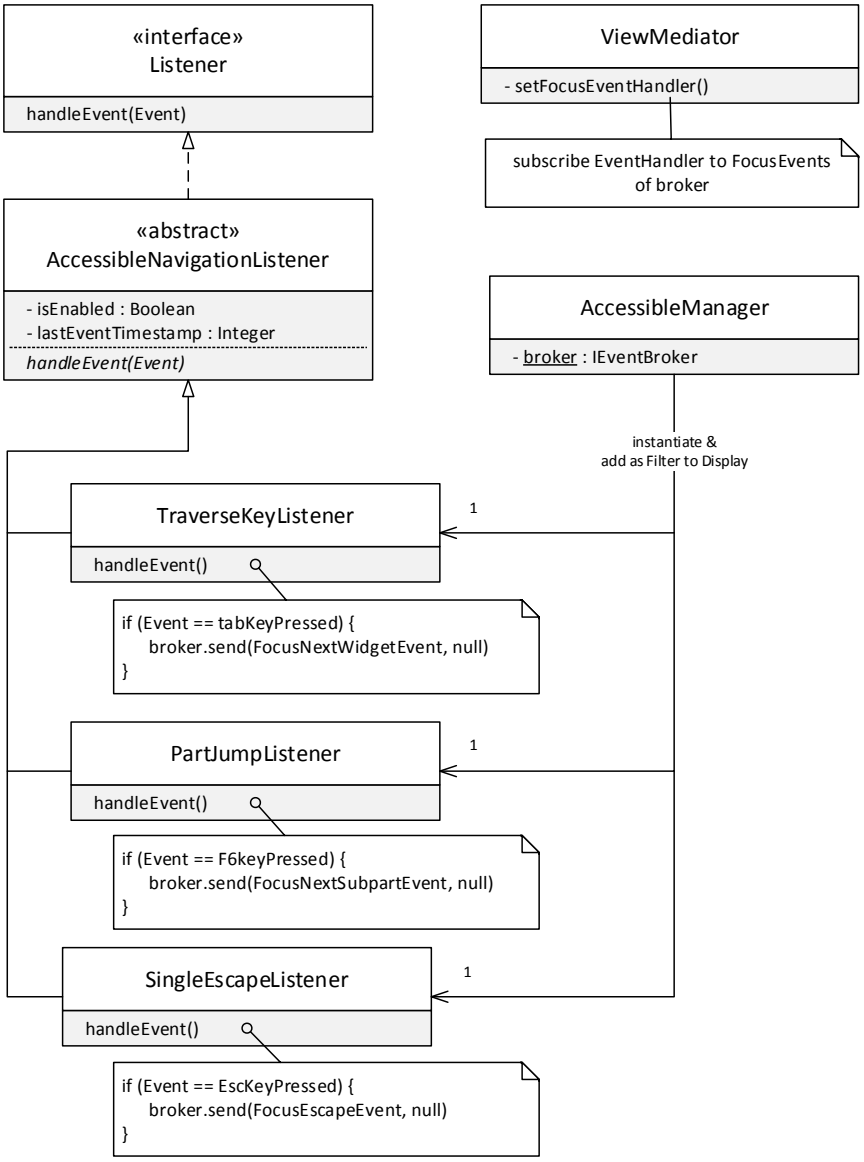


fig. 27 – *Navigations-Listener (UML)*

Die Navigations-Listener sind global gesetzte Listener. Alle Events werden über die im AccessibleManager gesetzte Instanz des IEventBrokers gesendet respektive verwaltet.

Der im AccessibleNavigationListener gesetzte Boolean «isEnabled» kann zum Aktivieren/Deaktivieren des jeweiligen Listeners verwendet werden. So müssen diese nicht entfernt werden, sollten sie für einen gewissen Moment ihre Funktion nicht wahrnehmen müssen. Der Timestamp ist nötig, um die zum Teil mehrfach geschickten, zeitgleichen Events ignorieren zu können.

Weitere Ausführungen zur Tastaturnavigation und deren Events unter «4.3.2. Tastaturnavigation leiten» auf Seite 44.

4.6.3. Accessible Extension

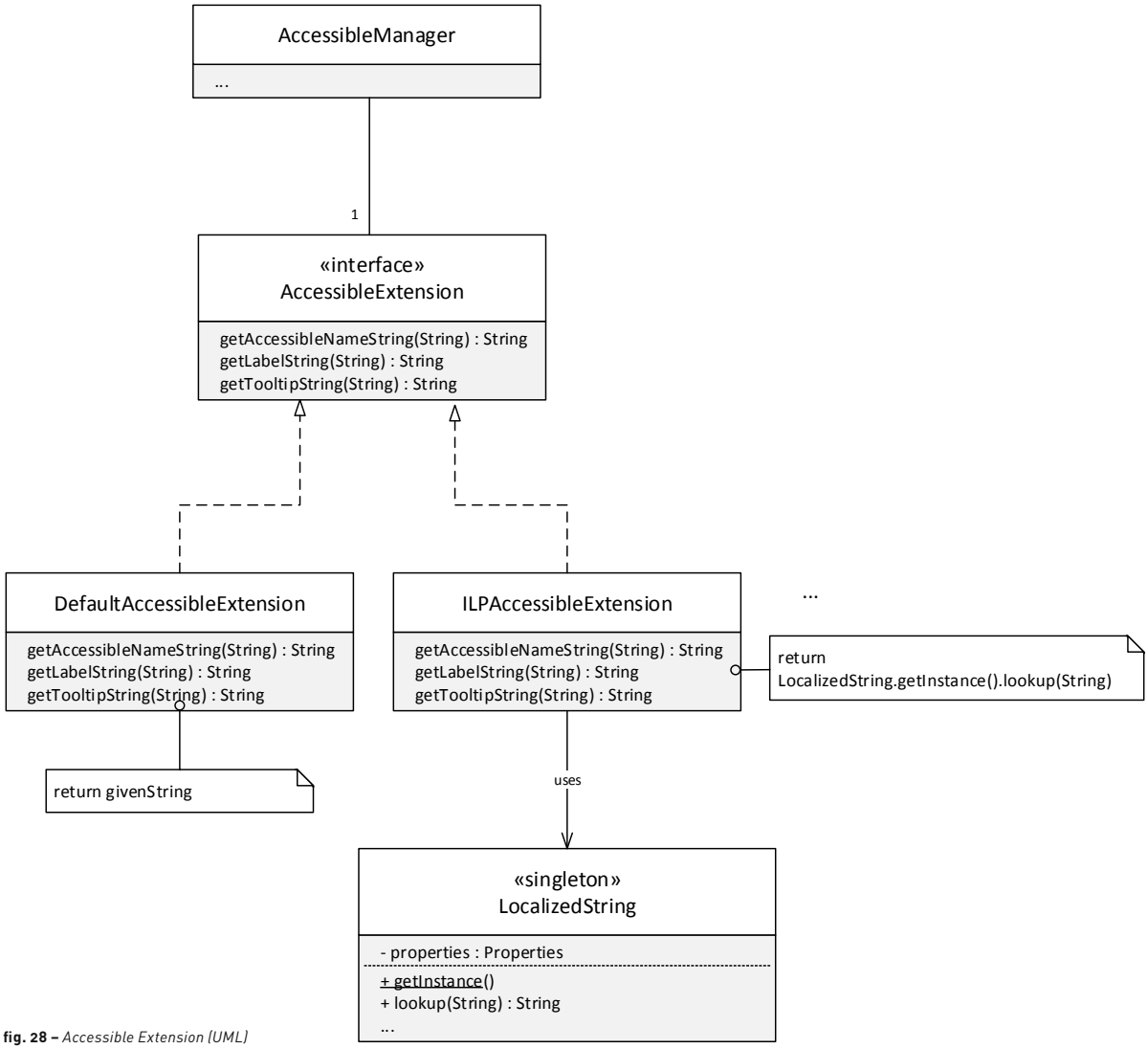


fig. 28 – *Accessible Extension (UML)*

Dieses Diagramm greift dem Kapitel «5.2. Extension-Mechanismus: Anpassung durch Subklasse» auf Seite 54 vor. Die Accessible Extension-Möglichkeit wurde erst während der Implementation entwickelt. Sie dient dazu, Methoden extern des *Accessible Moduls* unterschiedlich implementieren zu können.

Wird keine eigene AccessibleExtension-Klasse definiert, so kommt die DefaultAccessibleExtension-Klasse zum Zug.

4.6.4. Ablauf Initialisierung Accessible Modul

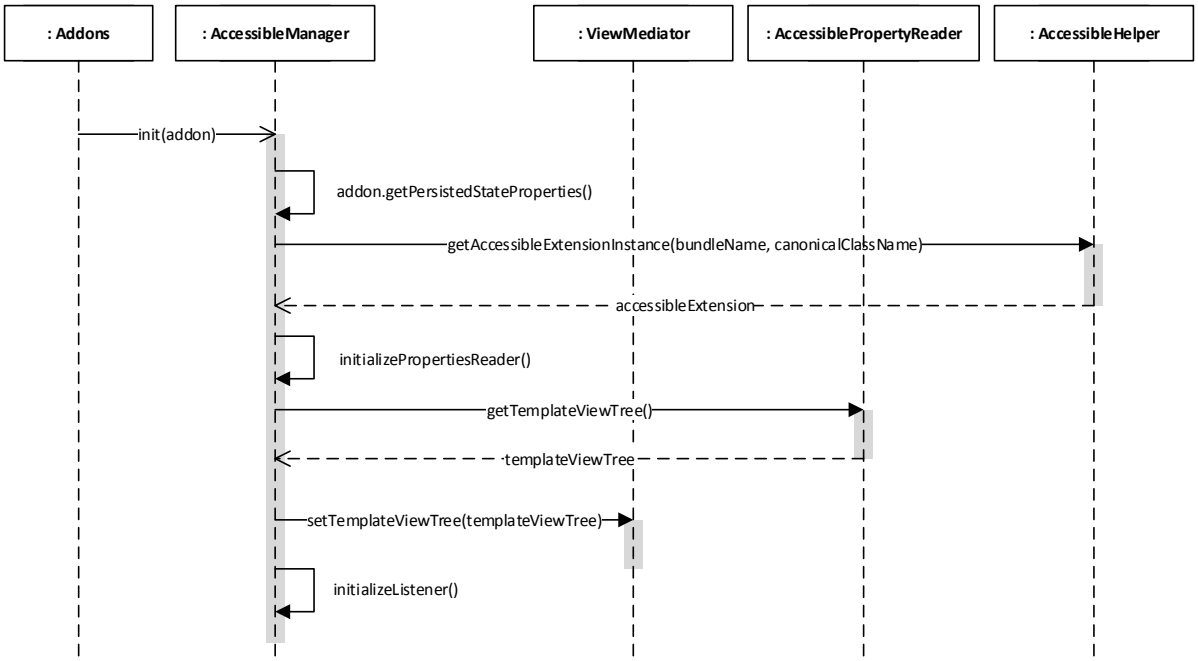


fig. 29 – *Ablauf Initialisierung Accessible Modul (UML)*

Weitere Details zur Initialisierung des *Accessible Moduls* unter «Initialisierung und Konfiguration» auf Seite 38.

4.6.5. Ablauf Part-Activation

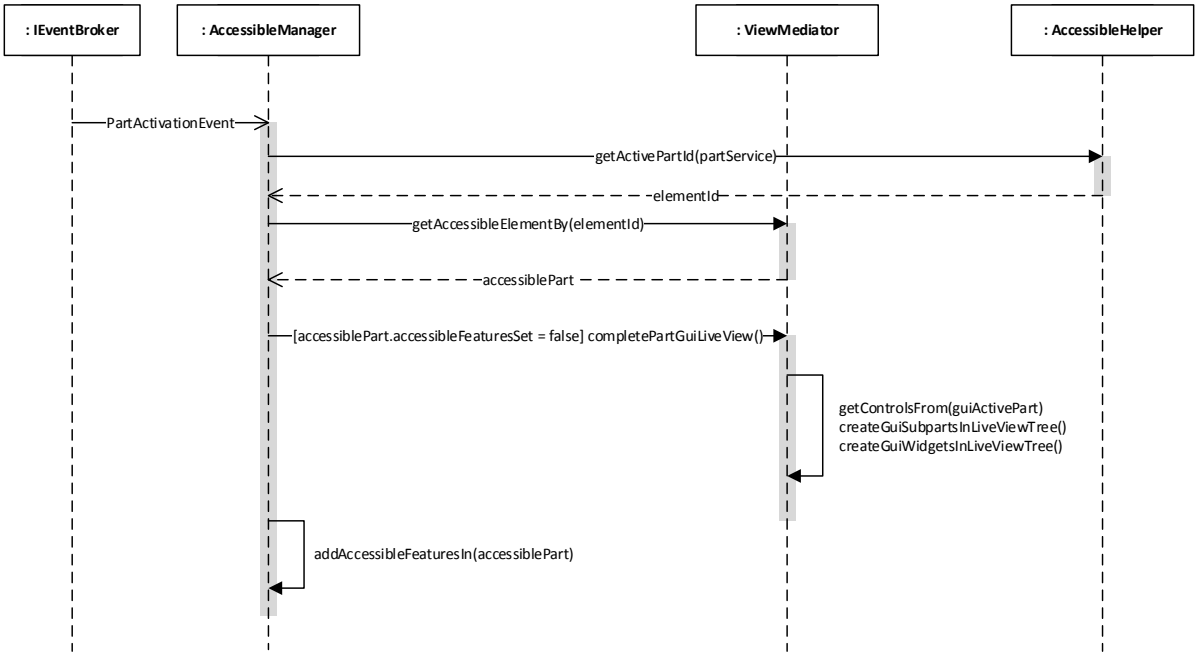


fig. 30 – *Ablauf Part-Activation (UML)*

Weitere Details zur Part-Activation unter «Accessible-Features setzen» auf Seite 39.

4.7. Veröffentlichung des Quellcodes

Im Bereich Accessibility/*JAWS*/*RCP* konnte kein Dokument gefunden werden, das ganz konkrete Implementationsansätze zeigt. Umso wichtiger ist es deshalb, wenigstens die durch diese Arbeit erlangten Kenntnisse zu teilen, damit andere Entwickler ein Feld weiter vorne starten kann.

Der Quellcode der *Proof of Concept*-Applikation und des *Accessible Moduls* wurden dafür in einem öffentlichen Ordner auf *GitHub*[34] bereitgestellt. Somit kann jeder die beiden Projekte in die eigene *eclipse*-Entwicklungsumgebung importieren und das *Accessible Modul* verwenden.

Bei den aktuell über 25 Millionen[35] öffentlichen *GitHub*-Repositories ist die Chance klein, als Entwickler bei der Suche genau darauf zu stossen. Darum wurde ein Blog-Artikel verfasst, der sowohl auf das *GitHub*-Repository verweist wie auch auf eine Möglichkeit zum Download dieser Arbeit.

Im deutschsprachigen Raum ist der Blog von *Access for All* einer der führenden betreffend aktuellen, barrierefreien Technologien. Der Empfehlungskatalog, welcher aus der Vorarbeit hervorgegangen ist, durfte auf dieser Plattform bereits veröffentlicht werden. Der neue Artikel reiht sich daher gut in die Folge ein und zeigt auch gleich auf, dass Theorie und Praxis weiter auseinanderliegen, als man das gerne hätte.

Für eine zusätzliche Bekanntmachung innerhalb der *eclipse* Community konnte noch keine passende Plattform gefunden werden. Um das *Accessible Modul* prominent zu platzieren, fehlt auch noch die programmatische Qualität.

5. Implementation des Accessible Moduls in beook

5.1. Schwieriger Einbau

Eines vorneweg: Die Implementation des *Accessible Moduls* stellte sich als viel schwieriger heraus, als dies in der Konzeptionsphase erwartet wurde. Die Dynamik der Applikation und die Unterschiede im *GUI*-Aufbau der Applikation wurden im Vorfeld nicht genügend bedacht. Trotz intensivem Arbeitseinsatz kann der derzeitige Stand der Implementation des *Accessible Moduls* nur als «knapp befriedigend» bezeichnet werden.

34 tb.chrissharkman.ch/github
35 GitHub Features – www.github.com > Features

Die vorhandenen Mittel wurden darauf konzentriert, das *Accessible Modul* generischer zu machen, um der Dynamik und der Umgebung einer bestehenden Applikation besser gerecht zu werden.

5.2. Extension-Mechanismus: Anpassung durch Subklasse

5.2.1. Möglichkeiten schaffen

Das *Accessible Modul* war für die *Proof of Concept*-Applikation funktionell genügend. Beim Herangehen an die Implementation in *beook* zeigte sich aber umgehend, dass die sehr komplexe und flexibel konfigurierbare Applikation *beook* mehr verlangt: Um das *Accessible Moduls* integerieren zu können mussten zuerst mehrere Adapter und Schnittstellen entwickelt werden, damit dieses eingesetzt werden kann. Dies war insofern ein gewünschter Aspekt, um die Praxistauglichkeit des Moduls zu beweisen und zu verbessern.

Durch einen Extension-Mechanismus wird eine Möglichkeit geschaffen. In der derzeitigen Version des Moduls wird der Extension-Mechanismus über eine einzelne Instanz eines Interfaces geführt. Damit kann in *beook* zum Beispiel die Anforderung zur Internationalisation, dem sprachabhängigen Beschriften, abgedeckt werden.

5.2.2. Mehrsprachigkeit ohne Doppelspurigkeit

beook kann derzeit in vier verschiedenen Sprachen verwendet werden. Zusätzlich sind gesetzte Texte auf Labels, Schaltflächen und anderen Elementen der Programmumgebung nicht nur sprach- sonder auch inhaltsabhängig. Um all diese Beschriftungen effizient zu steuern, steht in der *beook*-Applikation die Klasse «LocalizedString» zur Verfügung. Alle statischen Texte werden über diese Klasse aus entsprechenden Properties-Dateien geladen.

Eine parallele Funktionalität im Modul aufzubauen wäre nicht optimal: Eine mögliche vorgegebene Variante für sprachabhängige Texte würde nicht allen Anwendungen gerecht. Vor allem würden bestehende, in der Applikation vorhandene Programm-Elemente, die bereits einen Grossteil der Funktionalität sauber abdecken, konkurrenziert. Diese Doppelspurigkeit muss vermieden werden.

5.2.3. AccessibleExtension-Interface mit Default-Klasse

UML-Diagramm «4.6.3. Accessible Extension» auf Seite 51

Das «AccessibleExtension»-Interface wurde geschaffen, um durch applikationseigene Klassen die Modulfunktionalität zu erweitern – wie im Fall der Mehrsprachigkeit. Dieses führt alle Methoden auf, welche eine Möglichkeit zur Einflussnahme bieten sollen. Eine Instanz des Interface wird im AccessibleManager gesetzt und dieses definiert dann, wie die aufgerufenen Funktionen effektiv ausgeführt werden.

So werden Werte beim Setzen von Strings in getName-Handler, Labels und Tooltips durch die Interface-Instanz validiert. Für *beook* heisst das: Die im viewTree-Modell gesetzten Schlüssel-Werte können über die bestehende

«LocalizedString»-Funktionalität gefunden und übersetzt werden. Retourniert wird der effektiv zu setzende Text.

Diese Funktionalität muss natürlich zuerst geschaffen werden. Eine in der Applikation neu erstellte Klasse – ILPAccessibleExtension – implementiert das AccessibleExtension-Interface und alle dazugehörigen Methoden. So kann jede Applikation die Logik für die Methoden im AccessibleExtension Interface selbst bestimmen.

Setzen des ILPAccessibleExtension-Objekts

Das Setzen des ILPAccessibleExtension-Objekts kann, um eine nötige Flexibilität zu wahren, über zwei Arten geschehen:

- Im Application.e4xmi wird unter dem Addon «AccessibleManager» der Parameter accessibleExtension mit dem Wert des Klassennamens und des Bundle-Namen gesetzt. Ist dieser Wert vorhanden, so wird die Instanzierung des AccessibleExtension-Objekts im AccessibleManager mit der gegebenen Klasse ausgeführt.
- Eine Setter-Methode in der AccessibleManager-Klasse zum nachträglichen Setzen, beziehungsweise Verändern der AccessibleExtension-Instanz, erlaubt den programmatischen Zugriff aussen.

Bequemer Standard

Um keinen unnötigen Aufwand für die Implementation von der im Interface zugesicherten Methoden zu schaffen, wird standardmässig ein Default-AccessibleExtension-Objekt gesetzt. Dieses enthält ganz simple Methoden, die den normalen Programmfluss mit der geringsten, nötigen Logik herstellen. Dieser Standard wird verwendet, wenn kein Klassenname im Addon gesetzt ist und auch nachträglich keine andere neue Instanz gesetzt wird.

5.2.4. Ergänzung im Manifest.mf

Durch das Erstellen einer Klasse in der Applikation, welche aber im Modul gebraucht wird, stellen sich Fragen zur Abhängigkeit zwischen Applikation und Modul. Im Modul soll nie eine direkte Abhängigkeit erstellt werden müssen, sprich zum Beispiel das Applikations-Plugin in die Dependencies gesetzt werden.

Trotzdem muss die neu erstellte ILPAccessibleExtension dem Modul bekannt sein. Sie muss also im Classpath des Moduls gesetzt sein, sonst kann diese bei der Instanzierung nicht gefunden werden. Eclipse bietet dafür einen «Kumpel» an: den Eclipse-RegisterBuddy. Der Schlüssel «Eclipse-RegisterBuddy» muss in der Manifest-Datei der Applikation eingetragen werden. Als Wert muss das der Bundle-Name des Moduls angegeben werden.

Modul dem Eclipse-RegisterBuddy angeben, um eigene Klasse verfügbar zu machen
Auszug aus: ch.ionesoft.ilp.rcp.application.beook/META-INF/MANIFEST.MF

```
Eclipse-RegisterBuddy: ch.chrissharkman.accessibility.rcp.base
```

Damit das Modul den RegisterBuddy auch aktiviert, muss in der Manifest-Datei des Moduls die Eclipse-BuddyPolicy gesetzt werden. Im *Accessible Modul* ist der Wert auf «registered» gesetzt. Der Wert «registered» gibt

an, dass alle Bundles, die sich explizit als Buddy registrieren, vom Modul konsultiert werden können.[36]

Eclipse-BuddyPolicy festlegen
Auszug aus: ch.chrissharkman.accessibility.rcp.base/META-INF/MANIFEST.MF

```
Eclipse-BuddyPolicy: registered
```

5.3. Flexibilitätssteigerung der Struktur-Vorgabe durch Wildcard-Matching

5.3.1. Struktur-Datei dynamisieren

Bei der Implementation traf die im *Proof of Concept* noch einfach gehaltene Realität auf die echte Realität: Eine dynamische Erzeugung von Perspektiven und Parts in einer RCP-Applikation ist jederzeit möglich. Dies ist in *beook* der Fall und macht es dadurch unmöglich, durch eine statische Struktur-Datei ein ViewTree-Modell zu erzeugen, welches die Applikation zur Laufzeit korrekt abbildet.

Es benötigte deshalb zum einen in der Struktur-Datei die Möglichkeit, Elemente Template-basiert beschreiben zu können. Zum anderen musste das ViewTree-Modell diesen Template-Elementen Rechnung tragen können.

5.3.2. Struktur-Datei mit Template-Elementen

Durch eine Template-basierte Beschreibung wird die Struktur-Datei dynamisiert. So können Elemente mit partiell unbekannten IDs in der Struktur-Datei beschrieben werden, wenn zum Beispiel strukturgleiche Parts mehrfach vorkommen. In *beook* ist dies mit den Buch-Perspektiven der Fall. Jedes Buch erhält seine eigene Perspektive, welche die gleiche Struktur hat wie die anderen Buch-Perspektiven: Gleiche Parts, gleiche Subparts, gleiche Widgets. Was sie unterscheidet ist daher einzig die ID der Perspektive, die einem Muster folgt, aber nicht vollständig bekannt ist. Dieses Muster kann mit Wildcards beschrieben werden und macht damit aus dem Struktur-Element ein Template für eine Vielzahl «gleicher» Elemente.

Template-Element mit partiell unbekannter ID
Auszug aus: beookAccessible.xml

```
<handledtoolitem elementId="ch.ionesoft.ilp.rcp.application.?.handledtoolitem.*" />
```

Wildcards

Wildcards sind Platzhalter-Zeichen verschiedenster Art, die es erlauben unbekannte Sachverhalte zu beschreiben. Für das Wildcard-*Matching* des *Accessible Moduls* können die folgenden Wildcard-Zeichen verwendet werden:

- **Asterisk *** : Der Stern, welcher am Ende einer ID gesetzt werden kann, steht für «alle weiteren des gleichen Stamms».
- **Fragezeichen ?** : Das Fragezeichen, welches innerhalb einer ID gesetzt werden kann, steht für «unbekannter Einschub».

36 Context Class Loader Enhancements – wiki.eclipse.org

Auf das oben im Code aufgeführte Template-Element würde zum Beispiel folgende IDs matchen:

- ...tion.heig.handledtoolitem.login
- ...tion.heig.handledtoolitem.save.all
- ...tion.hes.handledtoolitem.login
- ...tion.hes.handledtoolitem.course.book

Die Überprüfung von Wildcard-Matchs wird von der Helper-Klasse «StringMatcher» übernommen. Der StringMatcher kommt aber erst zum Zug, wenn keine komplette ID, ohne Wildcard-Zeichen, gefunden werden konnte.

5.3.3. Erweiterung des ViewTree-Modells

Auch das von der Struktur-Datei definiterte ViewTree-Modell musste deshalb mit einem LiveViewTree um eine Stufe erweitert werden *(fig. 31)*: Aus der Struktur-Datei wird immer noch direkt ein ViewTree erzeugt, dieser dient aber als Template-Modell. Die Zuordnung der GUI-Elemente passiert anhand des *Matchings*.

Ausgangspunkt für die Zuordnung sind die in der Applikation effektiv vorhandene GUI-Elemente. Finden diese im TemplateViewTree ihre Vorlage, so werden sie an den für sie bestimmten Platz in den LiveViewTree gesetzt.

Die TraverseKey-Navigation nimmt für die Fokus-Sprünge Bezug auf den LiveViewTree. So ist sichergestellt, dass die darin vorhandenen Elemente auch wirklich existieren.

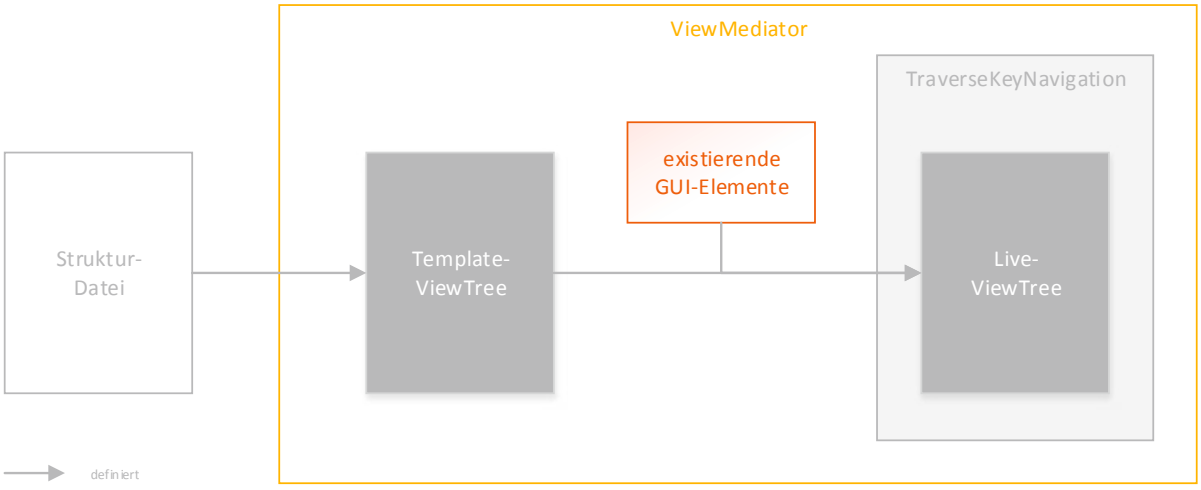


fig. 31 – *Erweiterung des ViewTree-Modells mit LiveViewTree*

5.4. Zusätzliche Eingriffe in beook

5.4.1. Switch zum einfachen Ein- und Ausschalten

Um das implementierte *Accessible Modul* mit einem einzigen Flag aktivieren, beziehungsweise deaktivieren zu können, wurde im AccessibleManager nachträglich ein öffentlicher Boolean gesetzt. Dieser bestimmt, ob das *Accessible Modul* überhaupt initialisiert wird oder zu Beginn der init()-Methode

des AccessibleManagers abgebrochen wird. In der Activator-Klasse, welche beim Start von *beook* aufgerufen wird, kann so dieses Flag gesetzt werden, um das Starten des Moduls zu unterbinden. Damit besteht die Möglichkeit das *Accessible Modul* in den Einstellungen der Applikation zu aktivieren.

5.4.2. CLabel – «Button-Hacks»

Eine Problematik, welche Eingriffe in *beook* unabdingbar macht, sind die in der Toolbar gesetzten CLabels, welche die Funktion von Buttons einnehmen. Es sind «Customized Labels», die mit einem Listener gehört werden und mit einem Bildwechsel eine Selektion simulieren.

Die Verwendung dieser Elemente hat optische Gründe. Diese Elemente zugänglich zu machen heisst, dass sie den Fokus empfangen können. Dies ist so nicht der Fall. Somit würde ein Neu-Aufbau dieses Teils anstehen. Ohne diese Anpassung, ist eine korrekte Funktionalität des Moduls nicht machbar.

Ein nötiger Eingriff für den Neu-Aufbau ganzer GUI-Elemente wurde derzeit als zu kritisch angesehen, da im Rahmen dieser Bachelor-Arbeit das Zeitbudget nicht zugelassen hätte. Zudem wäre dann auch die Umsetzung des unter «3.2.4. Technische Finessen beim Erstellen des GUIs» auf Seite 26 bereits erwähnten Ansatzes zu verfolgen, über Factories je nach Fall «das richtige Element» zu erzeugen.

Dieser Umbau könnte in einer nächsten Phase im Rahmen des Projekts von *Access for All* stattfinden.

5.4.3. Portal

Die Portal-Perspektiven wurden ebenfalls von der Implementation ausgenommen. Diese steht zum jetzigen Zeitpunkt gerade vor einer Neukonzeption: Weg von den System-GUI-Komponenten, hin zu einem auf allen Plattformen gleich verwendbaren HTML-Webportal.

5.5. Resultat

Viele der Hintergrundfunktionalitäten des implementierten *Accessible Moduls* arbeiten korrekt:

- **ViewTree**: Die Erstellung des TemplateViewTrees ist korrekt, das Zuordnen von GUI-Elementen und Erstellen des LiveViewTree funktionieren bis auf Stufe Part. Das Template-Matching von IDS greift.
- **Labels**: Labels und Tooltips werden korrekt gesetzt. Da der LiveViewTree nicht komplet erstellt werden kann, können die Subparts und Widgets nicht vervollständigt werden.
- **AccessibleExtension**: Der Zugriff über die Extension auf die gewünschten localized Strings funktioniert.

Betreffend Navigation ist es bedauerlicherweise für die Zielgruppe noch nicht anwendbar. Aufgrund der fehlenden Funktionalität und noch anzupassendem GUI konnten auch keine Tests mit der Zielgruppe gemacht werden, was zu in einer späteren Phase nachgeholt werden muss.

- **TraverseKey-Navigation**: Aufgrund der GUI-Elemente wie CLabels, aber wegen dem unvollständigen LiveViewTree kann die TraverseKey-Navigation ihre Wirkung noch nicht entfalten.
- **PartJump**: PartJumps sind möglich, die Iteration innerhalb eines Parts findet damit aber nicht statt.
- **Escape**: Das Escapen funktioniert, es handelt sich dabei aber auch lediglich um einen Fokus-Sprung auf die globale Toolbar.

Die *beook*-Applikation wird aus Gründen der Betriebsgeheimnisse nicht mitgeliefert. Ab dem nächsten Release von *beook* besteht die Möglichkeit, dass das *Accessible Modul* Bestandteil von *beook* sein wird.

Die Resultate zeigen, dass in einem Applikations-Umfeld wie RCP ein Modul eine starke generische Wirksamkeit haben muss, um bestehen zu können. Dieser Aspekt muss noch weiterentwickelt werden.

5.6. Mögliche Erweiterungen für die Zukunft

Die Integration des Moduls wie auch schon dessen Entwicklung hat auch viele Erweiterungsmöglichkeiten aufgezeigt. Diese werden für eine noch bessere Anwendbarkeit des Moduls bei zukünftigen Entwicklungsschritten in Betracht gezogen.

5.6.1. Komplettes Parsen der Struktur-Datei

In der Struktur-Datei sind bereits die Tastenkombinationen, beziehungsweise die Kurztasten angegeben, welche für das Navigieren durch die Applikation nötig sind. Auch für Dialoge der Applikation ist eine Sektion der Datei vorgesehen. Derzeit werden diese «Keybindings» und Dialoge aber nicht geparst und können somit auch keinen Einfluss auf die Accessibility nehmen.

5.6.2. GUI-Element-Unterstützung vervollständigen

Die Wirkung des *Accessible Moduls* ist derzeit erst auf eine beschränkte Anzahl von Widgets (Composites, Buttons) ausgerichtet. Tests für andere Widgets sind nötig, um auch deren Fokus-Verhalten noch genauer zu kennen und steuern zu können.

5.6.3. Commons Logging anstatt Log4j

Das *Accessible Modul* verlangt derzeit die Einbindung des Log4j-Loggers. Dies kann in einer Applikation, in der mit einem anderen Logger gearbeitet wird, störend sein. Denn vielleicht möchte man die gleiche Ausgabe-Konfiguration für das Loggen nutzen.

Um dies möglich zu machen, wäre der Apache Commons Logging Component geeignet. Er würde die Brücke zwischen Modul und der eigentlich implementierten Logging-Lösung der Applikation machen. Eine Konfiguration auf Seiten der Applikation wäre aber auch in diesem Fall nötig.

5.6.4. Flexibilität zur Laufzeit

Derzeit fehlen dem Modul noch Methoden, um während der Laufzeit Änderungen in das ViewTree-Modell der Applikation einspielen zu können. So ist

die Struktur von Perspektiven und Parts noch statisch und wird nur zu Beginn initialisiert. Gerade in *beook* gibt es auch den Fall, dass nachträglich, zum Beispiel beim Laden von zusätzlichen Büchern, neue Perspektiven hinzukommen. Dazu gehört auch die Möglichkeit, Elemente, Listener und den ViewTree wieder abbauen oder entfernen zu können.

6. Einfach barrierefrei?

Einfach barrierefrei – eine Vision, die während der ganzen Zeit dieser Arbeit hochgehalten wurde, aber mit fortschreitender Arbeit gefühlsmässig immer weiter in die Ferne rückte. Letztendlich steht ein ausbaufähiges Element, welches den Weg zur Barrierefreiheit verkürzt und vor allem «Accessibility» an einem Ort bündelt.

6.6.1. Nachträgliches Aufrüsten ist schwierig

Ob bei realen Bauvorhaben oder in der IT: Es ist bekannt dass ein nachträgliches Aufrüsten oft Schwierigkeiten mit sich bringt und dazu neue Wege gegangen werden müssen. Dies gilt auch für Barrierefreiheit.

Wird eine Applikation von Beginn an für die Zielgruppe konzipiert und werden die Anforderungen für Barrierefreiheit zumindest beim Entstehungsprozess berücksichtigt, so können die grössten Stolpersteine direkt vermieden werden. Schaut man sich jedoch gerade die in *beook* umfangreiche Arbeitsumgebung an, so wäre es mit einer «schnellen Lösung» unweigerlich zu Kompromissen in der Funktionalität oder zu Einbussen in der Gestaltung gekommen.

fig. 32 – *«Mouskie» Mini-Braillezeile mit zwei Braillezeichen*

Beispiel Mouskie

Als Beispiel für eine Applikation, welche Barrierefreiheit von Beginn weg zum Ziel hatte, steht die in sechs Monaten entwickelte Applikation «Mouskie». Sie ist für blinde und sehbehinderte Kinder und Jugendliche erstellt worden und dient im Zusammenspiel mit einer Art Computer-Maus zum Erlernen der Brailleschrift. Diese Computer-Maus ist keine Maus im gewöhnlichen Sinn, sondern eine Mini-Braillezeile mit nur zwei Braillezeichen *(fig. 32)*.

Durch die klare Zielgruppe musste Barrierefreiheit von Anfang kompromisslos angestrebt werden. Herausgekommen ist eine klare, einfach zu bedienenden Applikation *(fig. 33)*, die punkto akustische Hinweise über jene des Screenreaders hinaus geht. Zusätzlich sind auch individuelle Kontrast-Einstellungen möglich. Für die Kontroll-Elemente mussten eigene Klassen entwickelt werden – was auch erklärt, warum grundsätzlich nur Buttons ein-

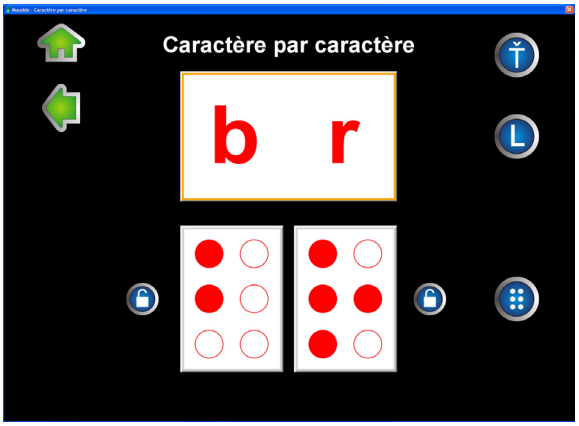


fig. 33 – *Die Benutzeroberfläche der Applikation «Mouskie» zum Erlernen von Braillezeichen*

gesetzt wurden. Während der Entwicklung wurde mit den Screenreadern *JAWS* und Dolphin getestet.

Natürlich ist Mouskie ein Beispiel des anderen Extrems, eine Applikation die als Benutzer nur die Zielgruppe im Fokus hat. Es zeigt mit deren Einfachheit aber gut, dass auch diese einen wichtigen Teil zur Barrierefreiheit beiträgt.

Dynamik fordert generische Eingriffe

Bei einem kleinen, übersichtlichen System wie dem *Proof of Concept*, schafft man es aufgrund seiner statischen Aufbauweise, die Accessibility zu kontrollieren. Ein grosses, dynamisches, bereits bestehendes System verlangt von einem Modul eine stark generische Art, um auf jeden erdenklichen Fall reagieren zu können.

6.6.2. Die Krux der Verbindung MSAA – JAWS

Sie bleibt bis am Ende dieser Arbeit in einer schwarzen Kiste: Die Verbindung zwischen *MSAA* und *JAWS*. Die Phase zum Herausfinden, was alles mit dem Screenreader ausgegeben werden kann, war absolut nötig, auch wenn die Resultate eigentlich bescheiden wirken. Nur mit Wissen kann man die Elemente komplett kontrollieren. Bei der Verbindung von *MSAA* und *JAWS* bleiben es blosse Erfahrungen.

6.6.3. Aus «ausführend» wird «entdeckerisch»

Beim Herangehen an diese Aufgabe war die Vorstellung da, relativ schnell an einen ausführenden Teil gehen zu können. Sobald die Zusammenhänge und Funktionalitäten klar sind, hätte die Umsetzung zuerst für das *Proof of Concept* und anschliessend für *beook* stattfinden sollen. Bald zeigte sich, dass sich aus der «ausführenden» eine vor allem «entdeckerische» Aufgabe entwickelt. Und diese Aufgabe war sehr fordernd. Die Richtigkeit, ein *Proof of Concept* anzufertigen hat sich gezeigt: Es gab viel zu entdecken, es musste vieles ausprobiert werden.

Das sehr Positive daran war, dass sich auch dank den Schwierigkeiten eine «globalere» Lösung mit dem *Accessible Modul* aufdrängte, als einfach an vielen Orten «Steinchen» zu setzen um die Barrierefreiheit zu verbessern. Schwieriger daran war, einzusehen, dass die Barrierefreiheit zum Abschluss der Arbeit noch nicht genug weit sein würde, um die nötigen Anforderungen gemäss Empfehlungskatalog zu erfüllen.

Natürlich wäre es schön gewesen, die gesetzte Vision realisieren zu können, um den blinden und visuell eingeschränkten Endbenutzern eine komplett barrierefreie *beook*-Applikation bieten zu können. Ein erster Schritt ist aber gemacht, viele wichtigen Erkenntnisse konnten gemacht werden, unter anderem:

- Welche Finessen im *GUI*-Aufbau für Accessibility wichtig sind.
- Wie für welches Element eine Sprachausgabe erzeugt werden kann.
- Wie sich die Navigation aus von «aussen» strukturieren lässt.
- Und dass Dokumentation sehr wichtig ist, um eine Technologie gut nutzbar zu machen.

6.6.4. Zwei Bausteine Richtung Ziel

Die vielen für diese Bachelor-Arbeit kontaktierten blinden und visuell eingeschränkten Personen haben immer wieder Verblüfft: Mit ihren Geschichten, mit ihrer differenzierten Wahrnehmung und mit ihrem ungebrochenen Tatendrang. Eine barrierefreie Plattform für digitale Lehrmittel würde diesem Tatendrang Rechnung tragen. Die nötigen, technischen Mittel stehen heutzutage alle zur Verfügung, sie müssen nur richtig zusammengeführt werden.

Mit dem *Accessible Modul* können zwei wichtige Aspekte der Barrierefreiheit abgedeckt werden: die Sprachausgabe und die Navigationsstrukturierung und -führung. Realistisch gesehen ist man damit dem Ziel, der vollständigen Barrierefreiheit zwei Bausteine näher, aber eben: Man ist noch nicht am Ziel.

Fazit

Die Herangehensweise über ein *Proof of Concept* war nötig, um in einem abgesteckten Rahmen herauszufinden, wie in *eclipse*-RCP-Applikationen Barrierefreiheit erzeugt werden kann. Fehlende Dokumentationen und scheinbare Möglichkeiten verlangten viel Entdeckersinn, um schliesslich aufzeigen zu können, was wirklich funktioniert.

Mit diesem Wissen wurde ein Modul entwickelt, welches zwei Funktionen abdeckt: an den korrekten Orten Labels für Screenreader setzen und die Navigation koordinieren und strukturieren. Die Implementation in das *Proof of Concept* zeigte, das Funktionieren des Moduls und dass es praktisch ist, wenn Accessibility-Aspekte an einem Ort konzentriert verwaltet werden können.

Die Implementation des Moduls in *beook*, einer bestehenden RCP-Applikation, war schwierig und bleibt bis zum Schluss dieser Arbeit nur knapp befriedigend: Einen ganz einfachen Weg, Barrierefreiheit in einer bestehenden Applikation zu erzeugen, ist auch mit dem *Accessible Modul* noch nicht gebahnt. Die Dynamik der Applikations-Struktur und der Einsatz von GUI-Elementen, die für Accessibility nicht geeignet sind, verhindern die Verwendung der kompletten Funktionalität. Diese stellt ein Minimum dar für blinde und visuell eingeschränkte Personen, damit Ihnen wirklich gedient ist.

Für *ionesoft* wird im Rahmen des Projekts von *Access for All* auf eine Weiterführung dieser Arbeit gehofft, um von den wertvollen, gewonnenen Erfahrungen profitieren zu können. Die Vision bleibt, eine komplett barrierefreie Plattform für digitale Lehrmittel zu erschaffen.

Literaturverzeichnis

Monographien, Bücher, Beiträge und Publikationen

Barry FEIGENBAUM, updated by Sueann NICHOLS – *Accessibility Design and Coding Guidelines for Java Swing and SWT GUI Development*
IBM Worldwide Accessibility Center/IBM Human Ability and Accessibility Center,
17. Dezember 2008

> Leitfaden für Java-Entwickler, die im Bereich Swing und SWT mit Anforderungen an Barrierefreiheit konfrontiert werden. Ergänzendes Dokument zu den zwei anderen Dokumenten von IBM «Software accessibility checklist» und «IBM Guidelines for Writing Accessible Applications Using 100% Pure Java». Das Wertvolle an dieser Publikation sind der Vergleich Swing – SWT und die abgebildeten Code-Beispiele.

Christian HEIMANN – *Blind am Computer: Bachelor-Vorarbeit*
heig-vd, comem+, Bern, 2015

> Erarbeitung und Publikation eines Empfehlungskatalogs zur Erstellung von barrierefreien Desktop-Applikationen mit dem Praxisbeispiel beook

Marc TEUFEL, Dr. Jonas HELMING – *Eclipse 4, Rich Clients mit dem Eclipse SDK 4.2*
entwickler.press, Frankfurt am Main, 2012

> Arbeitsbuch mit starkem Bezug auf die Neuheiten der Eclipse 4 Plattform im Vergleich mit Eclipse 3.x und mit leichtem Tutorial-Charakter. Dies ist nützlich zum Komplettieren bestehender eclipse-RCP-Kenntnisse, aber auch um den Werdegang der Eclipse-Plattform seit Version 1.0 nachzuvollziehen.

Internetquellen

Access Keys – WinDevCenter – Common UI Controls and Text Guidelines – msdn.microsoft.com
https://msdn.microsoft.com/en-us/library/windows/desktop/bb226831(v=vs.85).aspx

AVSpeechSynthesizer – iOS Developer Library – AVFoundation Framework Reference > AVSpeechSynthesizer Class Reference
https://developer.apple.com/library/ios/documentation/AVFoundation/Reference/AVSpeechSynthesizer_Ref/index.html – Stand 18.09.2013, Abfrage am 23.07.2015

Class Accessible – Documentation Eclipse Platform Release 4.2 – help.eclipse.org
http://help.eclipse.org/juno/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fswt%2Faccessibility%2Fpackage-summary.html –
Stand 2012, Abfrage am 21.07.2015

Class KeyEvent – Documentation Eclipse Platform Release 4.4 – help.eclipse.org
http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fswt%2Fevents%2FTraverseEvent.html –
Stand 2014, Abfrage am 23.07.2015

Class Label – Documentation Eclipse Platform Release 4.2 – help.eclipse.org
http://help.eclipse.org/juno/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fswt%2Fwidgets%2FLabel.html –
Stand 2012, Abfrage am 23.07.2015

Context Class Loader Enhancements – wiki.eclipse.org
http://wiki.eclipse.org/index.php/Context_Class_Loader_Enhancements#Eclipse-BuddyPolicy_Header – Stand 29.03.2009, Abfrage am 16.07.2015

Convention on the Rights of Persons with Disabilities – United Nations Treaty Collection – treaties.un.org > Databases > Status of Treaties > Chapter 4 > 15
https://treaties.un.org/Pages/ViewDetails.aspx?src=TREATY&mtdsg_no=IV-15&chapter=4&lang=en – Stand 13.07.2015, Abfrage am 13.07.2015

Cursor visibility in Text SWT – stackoverflow.com – Question 23985519
http://stackoverflow.com/questions/23985519/cursor-visibility-in-text-swt –
Stand 23.07.2015, Abfrage am 23.07.2015

Ralph SCHUSTER – *Eclipse/E4: Problem with Key Bindings* – techblog.ralph-schuster.eu
http://techblog.ralph-schuster.eu/2013/10/13/eclipsee4-problem-with-key-bindings/comment-page-1/#comment-35773 – Stand 13.10.2013, Abfrage am 20.06.2015

EclipseCon: Eclipse Community Award mit vielen deutschen Gewinnern –
www.heise.de > heise Developer > News > 2012 > KW 13
http://www.heise.de/developer/meldung/EclipseCon-Eclipse-Community-Award-mit-vielen-deutschen-Gewinnern-1484682.html
Stand 28.03.2012, Abfrage am 13.07.2015

FAQ: How do I embed AWT and Swing inside SWT? – Eclipsepedia – wiki.eclipse.org
https://wiki.eclipse.org/FAQ_How_do_I_embed_AWT_and_Swing_inside_SWT%3F –
Stand 19.11.2012, Abfrage am 21.07.2015

FAQ: Why does Eclipse use SWT? – Eclipsepedia – wiki.eclipse.org
https://wiki.eclipse.org/FAQ_Why_does_Eclipse_use_SWT%3F –
Stand 16.06.2006, Abfrage am 21.07.2015

Lars VOGEL – *Free Tutorials to program Eclipse RCP and Plug-ins* –
www.vogella.com > Tutorials > Eclipse RCP/Plug-ins
http://www.vogella.com/tutorials/eclipse.html – Stand 2015, Abfragen März bis Juli 2015

> Vogella ist einer der eclipse rcp Gurus. Er bietet auf seiner Webseite eine grosse Anzahl an Tutorials gratis an. Die meisten Tutorials bestehen jeweils aus einer theoretischen Einführung und einer praktischen «Step by Step»-Anleitung.

GitHub Features – www.github.com > Features
https://github.com/features – Stand 2015, Abfrage am 26.07.2015

JAWS Keystrokes – www.freedomscientific.com > Support > Documentation > JAWS
Auf der Webseite von Freedom Scientific kommt man auf eine Version der Keystrokes von 2008. Über den direkten Link ist es möglich, eine neuere Variante aufzurufen:
http://doccenter.freedomscientific.com/doccenter/archives/training/jawskeystrokes.htm –
Stand 10.09.2014, Abfrage am 23.07.2015

SWT Event Propagation – stackoverflow.com – Question 7225756
http://stackoverflow.com/questions/7225756/swt-event-propagation –
Stand 30.03.2015, Abfrage am 23.07.2015

SWT JFace Eclipse « Java – www.java2s.com > Java > SWT JFace Eclipse
http://www.java2s.com – Stand 2015, Abfragen Mai bis Juli 2015

SWT Widgets – www.eclipse.org > Project > SWT > Widgets
https://www.eclipse.org/swt/widgets/ – Stand 2015, Abfrage am 03.06.2015

Alex BLEWITT – *The difference between SWT and AWT* – alblue.bandlem.com
http://alblue.bandlem.com/2005/02/eclipse-difference-between-swt-and-awt.html –
Stand 10.02.2005, Abfrage am 21.07.2015

The Event Model – www.java2s.com – Java Tutorial > SWT > SWT Event
http://www.java2s.com/Tutorial/Java/0280__SWT/TheEventModel.htm –
Stand 2015, Abfrage am 23.07.2015

Using ARIA landmarks to identify regions of a page – www.w3.org
http://www.w3.org/WAI/GL/wiki/Using_ARIA_landmarks_to_identify_regions_of_a_page –
Stand 03.01.2015, Abfrage am 23.07.2015.

R. DOUGLAS FIELDS – *Why Can Some Blind People Process Speech Far Faster Than Sighted Persons?* – Scientific American – www.scientificamerican.com > Mind & Brain > News
http://www.scientificamerican.com/article/why-can-some-blind-people-process/ –
Stand 13.12.2010, Abfrage am 23.07.2015

Wikipedia

Accessible Rich Internet Applications – Wikipedia
http://de.wikipedia.org/wiki/Accessible_Rich_Internet_Applications –
Stand 15.04.2015, Abfrage am 23.05.2015

Classpath (Java) – Wikipedia
https://en.wikipedia.org/wiki/Classpath_(Java) – Stand 04.03.2015, Abfrage am 15.07.2015

Eclipse (IDE) – Wikipedia
https://de.wikipedia.org/wiki/Eclipse_(IDE)

Java Virtual Machine – Wikipedia
https://de.wikipedia.org/wiki/Java_Virtual_Machine – Stand 01.02.2015, Abfrage am 23.05.2015

Microsoft Active Accessibility – Wikipedia
https://en.wikipedia.org/wiki/Microsoft_Active_Accessibility –
Stand 31.05.2015, Abfrage am 28.07.2015

Microsoft Narrator – Wikipedia
https://en.wikipedia.org/wiki/Microsoft_Narrator – Stand 15.11.2014, Abfrage am 21.07.2015

Übereinkommen über die Rechte von Menschen mit Behinderungen – Wikipedia
https://de.wikipedia.org/wiki/Übereinkommen_über_die_Rechte_von_Menschen_mit_Behinderungen – Stand 13.07.2015, Abfrage am 13.07.2015

Persönliche Aufzeichnungen

Fragen an Alexander Hauser – Telefon-Interview
Transkription im Anhang, geführt am 8. Juli 2015

Gute Labels setzen – Mail-Wechsel mit Selamet Aydogdu
Auszug im Anhang, 20. Juli 2015

Videoquellen

Doug KIRSCHNER – *Tips and Tricks for Designing a Great Accessibility Experience for Your App* – Channel 9
https://channel9.msdn.com/Events/Build/2014/2-539 – Build 2014, Microsoft,
Veröffentlicht am 04.04.2014, Abfrage am 19.06.2015

Video mit Tipps und Tricks zum barrierefrei machen von XAML-Applikationen, mit einer generellen Einleitung zum Thema «Accessibility».

Abbildungsverzeichnis

Code-Auszugsverzeichnis

Glossar

Accessibility **Barrierefreiheit** Barrierefreiheit bezieht sich nicht nur auf blinde und visuell eingeschränkte Personen, sondern auf alle, die zum Beispiel in einer Tätigkeit, einer Bewegung, logischen Denkfolgen unter anderem durch die Umgebung, bauliche Installtionen oder gesellschaftliche Normen gewisse Dinge nicht tun können, weil Barrieren und Hürden, bestehen. Accessibility steht dafür, diese Barrieren abzubauen um Zugang zu schaffen.

Accesskey auch manchmal Mnemonic oder Hotkey genannt, sind alphanumerische Tasten die verwendet werden, um ein Menüpunkte anzuwählen.[37] Nicht zu verwechseln mit Shortcuts, die einen direkten Aufruf eines Befehls erzeugen.

ARIA **Accessible Rich Internet Applications** Ein von der Web Accessibility Iniative entwickelter Webstandard, der die Zugänglichkeit von Webseiten und Webanwendungen für Blinde, die Screenreader verwenden, erhöht. Seit März 2014 ist ARIA ein empfohlener Webstandard des World Wide Web Consortiums (W3C).[38]

ARIA-Regions Definierte Bereiche in Webseiten, deren Bezeichnung von Screenreadern erkannt wird und zwischen denen man durch Screenreader-Software-Funktionalitäten hin- und herspringen kann.

AT **Assistive Technology oder Assistive Technologien** Dies ist ein Sammelbegriff, mit dem alle Technologien bezeichnet werden, die Personen mit Einschränkungen helfen, diese Einschränkungen zu überwinden. AT umfasst ein breites Spektrum, vom Hörgerät über die in dieser Arbeit oft erwähnten Screenreadern bis hin zu Prothesen oder speziell für Behindertensport entwickelte Sportgeräte.

beook Applikation für digitale Lehrmittel der ionesoft GmbH

Bug Ein kleiner Programmierungsfehler, der zu Funktionsstörungen in einer Software führt.

Bundle Eine Ordner, in dem sich in der Regel verschiedene Packages mit Programmklassen befinden. Dieser ist in der Regel das Root-Verzeichnis eines eclipse Plugin-Projekts .

Classpath Ein bestimmter Ort, an dem die JVM nach benutzerdefinierten Klassen sucht.[39]

CSS **Cascading Style Sheet** Ein Dokument, dass die Stile zur Formatierung von HTML-Elementen enthält. Dies ist so konzipiert, um Form (CSS) und Inhalt (HTML) zu trennen.

eclipse Entwicklungsumgebung, die auf dem RCP-Framework basiert.

EPUB **Electronic Publication** Ist ein auf XML basierter, global verwendeter Standard für digitale Bücher und Publikationen.

GitHub Eine Plattform zur kollaborativen Entwicklungs-Zusammenarbeit für Software.

GUI **Graphic User Interface** Grafische Benutzeroberfläche, die mit optischen Elementen Daten, Funktionen und Struktur sichtbar macht.

IE **Internet Explorer** Windows Standard Internet-Browser

Interface Java-Klassentyp der vorgibt, welche Methoden eine Klasse haben muss, die das Interface implementiert. Ein Interface selbst implementiert keine Methoden.

Java Programmiersprache, mit der man systemunabhängige Programme schreiben kann, die auf einer virtuellen Maschine, einer Art «Übersetzer», ausgeführt werden.

---

37 Access Keys – WinDevCenter – Common UI Controls and Text Guidelines – msdn.microsoft.com
38 Accessible Rich Internet Applications – Wikipedia
39 Classpath (Java) – Wikipedia

Java Swing Eine Programm-Bibliothek zum Erzeugen von Benutzeroberflächen-Elementen, die nicht vom Betriebssystem abhängig sind, sondern immer gleich aussehen, da sie JAVA-nativ sind.

JAWS **Job Access with Speech** Name der derzeit weltweit am stärksten verbreitete Screenreader-Software, hergestellt von Freedom Scientific.

JFace Eine Programm-Bibiliothek, welche die Verwendung von SWT- und weiteren Benutzeroberflächen-Komponenten vereinfacht. Bestandteil der RCP.

JVM **Java Virtual Machine** Die Java Virtual Machine ist der Teil der Java-Laufzeitumgebung (Java Runtime Environment, JRE) für Java-Programme, der für die Ausführung des Java-Bytecodes verantwortlich ist. Hierbei wird im Normalfall jedes gestartete Java-Programm in seiner eigenen virtuellen Maschine (VM) ausgeführt.[40]

Listener Salopp ins Deutsche übersetzt ein sogenannter «Hörer». Ein programmatisches Element, das die gesendeten Events überprüft und schaut, ob einer davon für sich den Listener selbst bestimmt ist. Ist dies der Fall, wird in der Regel ein zugewiesener Event Handler aufgerufen, der dann eine Aktion ausführt.

Matching **to match** Der englische Begriff «Matching» heisst soviel wie zusammenpassend. Es verlangt keine totale Gleichheit.

Mnemonic siehe Accesskey

MSAA **Microsoft Active Accessibility** Technologie, die das Zusammenspiel von Microsoft Windows, Applikationen und den technischen Hilfsmitteln wie Screenreader verbessert.

Narrator **Microsoft Narrator** Windows-Proprietärer und nativ verfügbarer Screenreader, der mit jedem Microsoft Windows Betriebssystem mitgeliefert wird.[41]

NVDA **Non Visual Desktop Acces** Ein gratis Screenreader der Firma NV Access, der von einer breiten Community unterstützt wird und zu deren Hauptsponsoren Google und Adobe gehören.

Proof of Concept Eine der Praxis nahe Testanordnung zum Überprüfen der Umsetzbarkeit und Funktionalität eines Konzepts.

Property Eine Eigenschaft, ein Attribut. In einer Properties-Datei in der Java-Umgebung sind Properties als einfache Schlüssel-Wert-Paare beschrieben, die mit Doppelpunkt zwischen Schlüssel und Wert gesetzt werden, zum Beispiel: «version: 1.0»

RCP **Rich Client Platform** Programm-Basis, die es erlaubt mit vordefinierten Komponenten komplette Java-Applikationen zu erstellen.

Screenmagnifier Software, die Bildschirmausschnitte vergrössern kann. Beispiele sind ZoomText, MAGic und die Systemeigenen Bildschirmlupen auf Windows und Mac.

Screenreader Software, die den Inhalt eines Bildschirms erfassen und vorlesen kann. Beispiele sind JAWS, NVDA, VoiceOver (iOS) und TalkBack (Android).

String Ein Java-Objekt, das eine Zeichenkette aus Buchstaben, Zahlen, Leerzeichen usw. enthält.

SWT **Standard Widget Toolkit** Eine Programm-Bibliothek, welche eine systemunabhängige Verwendung von Betriebssystem-Eigenen Fenstern und Benutzeroberflächen-Komponenten ermöglicht.

Tag Englisch ausgesprochen, bezeichet ein Tag eine in der Regel kleine Markierung oder Kennzeichnung eines Elements.

---

40 Classpath (Java) – Wikipedia
41 Microsoft Narrator – Wikipedia

Trial & Error **Ausprobieren** Trial & Error ist ein angewandtes Prinzip zum Entdecken von Regelmässigkeiten und Funktionalitäten, die nicht dokumentiert sind, aber deren Vorhanden sein angenommen wird.

TTS Text-To-Speech Auch bekannt als Sprachsynthese, eine künstliche Stimme, welche geschriebenen Text akustisch ausgeben kann.

Windows SDK **Windows Standard Development Kit** Eine Zusammenstellung von Programmen, die zum Entwickeln von Windows-Applikationen nötig beziehungsweise hilfreich sind.

Wizard Wörtlich ins Deutsche übersetzt heisst Wizard «Zauberer». In der Programmierung bezeichnet ein Wizard in der Regel ein Script, welches oft über einen grafischen Dialog konfiguriert werden kann und eine automatische Erzeugung von gewünschten, respektive benötigten Elementen bewerkstelligt.

XML **Extensible Markup Language** Eine Auszeichnungssprache, die für Menschen lesbar ist und zum Datenaustausch entworfen wurde.

Eigenständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit ohne fremde Hilfe und ohne Verwendung anderer als der angegebenen Hilfsmittel verfasst habe und dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt zitiert habe.

Bern, 30. Juli 2015

HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

comem+

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

Diplôme 2015
Travail de Bachelor

Département Comem+
Ingénierie des Médias
Heimann Christian
christian.heimann@heig-vd.ch

29.06.2015

Accessibility für digitale Lehrmittel

Blind am Computer

Répondant externe

Daniel Stainhauser
ionesoft GmbH
Sandrainstrasse 17
3007 Bern
dstainhauser@ionesoft.ch
+41 32 513 39 91

Lieu du travail

ionesoft GmbH
Sandrainstrasse 17
3007 Bern

Le candidat

Heimann Christian

Le répondant interne

Hess Jean-Pierre

Le répondant externe

Daniel Stainhauser

Le chef du département Comem+

Jean-Marc Seydoux

Explication générale (motivation, définition stricte du sujet)

Ausgangslage Die ionesoft GmbH ist in der Schweiz derzeit führend mit «beook», ihrer Technologie für elektronische Lehrmittel (Workflow, Umwandlung, Plattform, Urheberrechtsschutz). Im Rahmen des Projektes «Accessibility goes Mainstream» in Zusammenarbeit mit der Stiftung «Access for all» wird die Technologie erweitert, um die Inhalte auch Blinden und visuell Beeinträchtigten zugänglich zu machen.

Mit den heutigen technischen Möglichkeiten liegen alle Elemente bereit, die es visuell eingeschränkten Menschen ermöglichen, Inhalte zu erfassen, die bisher nur Sehenden vorbehalten waren. Jetzt gilt es, diese Möglichkeiten auszunutzen!

Cadrage du sujet (problématique, hypothèses de travail, méthodologie, ressources, etc.)

TPB: Konzeptionelle Überlegungen zur optimalen Navigation in der beook Applikation und zur barrierefreien Anwendung des kompletten Funktionsumfanges. Die erarbeiteten Definitionen können bei einer Implementierung von Accessibility Features als Grundlage dienen. Damit ein grösstmögliches Publikum davon profitiert, soll das Dokument öffentlich zugänglich gemacht werden.

TB: Die Bachelorarbeit konzentriert sich in einem ersten Schritt auf das Herstellen eines Proof of Concepts. An einer inhaltsfreien Beispielanwendung (RCP, Windows, mit JAWS als Screenreader) wird die Verwendung von Accessibility-Techniken erkundet, aufgezeigt und dokumentiert. Diese Elemente und der dazugehörige Quellcode wird veröffentlicht.
In einem zweiten Schritt wird das gewonnene Wissen zur Implementierung von nötigen programmierungstechnischen Elementen in der beook Applikation umgesetzt. Dadurch soll die RCP Version (Windows) der beook Applikation soll einen grossen Schritt in Richtung Barrierefreiheit machen. Vom Quellcode der beook Applikation werden nur Auszug- und Screenshot-artige Teile abgebildet, mit Kennzeichnung der während dieser Arbeit eingefügten Elementen.

Objectifs TPB

- Empfehlungskatalog mit den erarbeiteten Definitionen.
- Veröffentlichung des Dokuments.

Objectifs TB

- Eine barrierefreie Beispielanwendung mit GUI- und Steuerungs-Elementen, die auch in der beook Applikation enthalten sind.
- Veröffentlichung von Erkenntnissen und des Quellcodes der Beispielanwendung.
- RCP Applikation «beook» für den Screenreader JAWS lesbar machen und eine barrierefreie Navigation ermöglichen.

Tâches TPB

- Analyse der aktuellen Windows RCP Applikation «beook» unter dem Aspekt «Barrierefreie Bedienung».
- Analyse von Standard-Keyboard-Belegungen von Computern für visuell eingeschränkte Personen, deren Mnemonics-Verwendung und Präferenzen von Tab-Reihenfolgen.
- Definition eines Keyboard-Access-Schemas unter Berücksichtigung von Standard-Belegungen und Mnemonics, festlegen von Tab-Reihenfolgen.
- Definition von Anpassungen und Implementationen, die zur barrierefreien Bedienung nötig sind.

Tâches TB

- Proof of Concept: Erstellen einer Beispielanwendung mit GUI-Elementen der beook Applikation.
- Erstellen einer Dokumentation mit Muster-Vorgehensweisen zum Erstellen von barrierefreien RCP-Applikationen.
- Erläuterungen von Problemen und deren Lösungen, die beim Erstellen des «Proof of Concepts» erfasst werden.
- Implementation von Elementen zum Lesbarmachen für Screenreader und zur hindernisfreien Navigation der beook Applikation.

heig-vd

« Confidentialité »

Confidentialité liée au Travail de Bachelor (TB)

Diplômant	Christian Heimann
Titre du Travail de Bachelor	Accessibility für elektronische Lehrmittel
Question de recherche du TB (ou thématique)	RCP Applikation für Windows barrierefrei machen bzw. lesbar machen für Screenreader. Navigation via Shortcuts and Mnemonics ermöglichen anhand Resultaten von gemachter Analyse. Erstellen einer «Proof of Concept»-Applikation mit entsprechender Dokumentation. Implementation von Code-Elementen in der beook-App: Von dieser Implementation werden nur Screenshot-artige Auszüge in der Arbeit kommentiert, der Source Code ist nicht Bestandteil der Bachelor Arbeit.
Professeur responsable du TB	Jean-Pierre Hess
Entreprise partenaire	ionesoft GmbH
Personne de référence *(non-publié)*	Daniel Stainhauser

Tous les TB sont déposés à la Bibliothèque de la HEIG-VD qui en gère l'archivage et la consultation. Quel que soit le niveau de confidentialité du TB, le nom du diplômant, le nom du professeur responsable, le titre du TB et la question de recherche figurent dans tous les documents de présentation des TB ainsi que dans la base de données des TB (http://tb.heig-vd.ch). Le professeur responsable veille à ce que le titre du TB et le libellé de la question de recherche soient rédigés conformément au niveau de confidentialité voulu.

Les TB peuvent être soumis à un logiciel anti-plagiat. Dans ce cas, leur contenu sera traité de manière confidentielle.

☒ **Le TB n'est pas confidentiel :**

Outre les informations mentionnées ci-dessus, les documents de présentation du TB contiennent également le nom de l'entreprise partenaire, un résumé et une affiche descriptive.

Le TB peut être consulté ou emprunté librement à la Bibliothèque par le corps enseignant et les étudiants. Si une personne externe à la HEIG-VD souhaite consulter ou emprunter un TB, elle dépose une demande motivée auprès de la Bibliothèque, laquelle sollicite l'accord du professeur responsable et du doyen du département concerné.

☐ **Le TB est confidentiel** : les conditions suivantes de diffusion des informations sont à appliquer :

Oui ☐ Non ☐ *Nous acceptons que **le nom de l'entreprise partenaire** figure dans les documents publiés ainsi que dans la base de données consultable sur http://tb.heig-vd.ch.*

Oui ☐ Non ☐ *Nous acceptons que, une fois validés par l'entreprise partenaire, **le résumé et l'affiche descriptive** du TB figurent dans la base de données consultable sur http://tb.heig-vd.ch.*

Oui ☐ Non ☐ *Nous acceptons que le TB soit consultable et empruntable par le corps enseignant, les étudiants ou une personne externe à la HEIG-VD sous condition de l'obtention de l'accord de l'entreprise partenaire, du professeur responsable du TB et du doyen du département concerné. Le TB porte la mention « **confidentiel, consultable sous condition** ».*

Oui ☐ Non ☐ *Nous demandons qu'aucune consultation ou emprunt du TB ne soit permis hormis par le professeur responsable du TB qui s'engage à ne pas faire usage des informations mises à sa disposition. Le TB porte la mention « **confidentiel, non consultable** ».*

Dans tous les cas, un accord de confidentialité doit être signé par l'étudiant, l'expert et toutes les personnes participant à l'évaluation du TB.

Nous déclarons accepter les conditions de diffusion du Travail de Bachelor indiquées ci-dessus.

Date : 29.06.2015 signature du diplômant : Christian

Date : 29.06.2015 signature du professeur responsable : Jean-Pierre Hess

Date : 29.06.2015 signature de l'entreprise partenaire : Daniel Stainhauser

N.B. : Ce document fait partie intégrante du cahier des charges du TB.
La forme masculine est utilisée comme genre neutre et désigne à la fois les hommes et les femmes.

Protokoll – PV Bilan Intermédiaire

In den Lokalitäten der ionesoft GmbH, Bern
29. Juni 2015, Beginn 09.00 Uhr

Teilnehmer: Jean-Pierre Hess (JHS), Daniel Stainhauser (DS), Christian Heimann (CH)

Agenda

1. Einleitung
2. Aktueller Stand
3. Anstehende Arbeiten, Verbesserungen, Ziele
4. Anpassung des Pflichtenhefts
5. Validierung von Plakat und Zusammenfassung
6. Festlegen der nächsten Fixpunkte und Pendenzen
7. Varia

1. Einleitung

DS stellt JHS kurz die ionesoft vor. Entstehungsgeschichte mit dem Wachstum auf heute 10 Mitarbeiter. CH ergänzt, wie er vor zwei Jahren während der «HES d'été» zur ionesoft gekommen ist.

2. Aktueller Stand

CH führt den aktuellen Stand aus: RCP-Applikation-Basis erstellt, funktionierende Screenreader-Access-Punkte, Detail-Probleme hat es gegeben, Elemente die nicht funktonieren wie sie eigentlich dokumentiert sind.
Eines der Probleme sind getAccessible()-Methoden getDescription und getHelp. Diese werden zwar aufgerufen, aber es findet keine Ausgabe über JAWS statt. Betreffend «fehlenden» Screenreader-Ausgaben rät DS deshalb zum Debuggen im Stack. Kontrolle, wo das in SWT die für Windows wichtigen Flags bzw. Properties gesetzt werden. Ein Live-Kurztest zeigt aber gleich, dass sich das Debuggen schnell System-Integer-Werten verliert. Eine Kontrolle, ob in den JAWS-Einstellungen auf dem Testgerät alle möglichen Werte ausgegeben werden, soll vorgenommen werden. (CH)

JHS findet generell, dass die Ampel noch im grünen Bereich ist, noch nicht im Orangen. Er merkt aber an, dass eine Detail-Versessenheit manchmal den Blick auf das Ganze gefährdet.

3. Anstehende Arbeiten, Verbesserungen, Ziele

Durch eine Diskussion unter den Teilnehmern werden mögliche Lösungsansätze entwickelt, wie das angestrebte Ziel möglichst erreicht werden kann, wovon einer als Prioritär gesetzt wird:

Eine XML-Struktur soll definieren, welches die Reihenfolge der Parts und Unterelemente ist und sogleich auch deren getName, getDescription, getHelp, Tooltip, Mnemonic bzw. Shortcut beschreiben. Die Parts werden aufgrund ihrer IDs mit einer Initialisierungs-Methode gesammelt und auf diesen Parts werden dann die nötigen Elemente gesetzt.
Das XML wird geparst und in Java-Objects geformt oder in ein JDOM im Memory gelegt.
Eine «Facade» soll den Zugriff auf die XML-Daten generisch halten, es sollen keine «DOM»-Objekte rausgeführt werden.
DS fügt an, dass die Initialisierung nicht nur bei App start sondern auch über ein callback ausgelöst werden können sollten, wenn Komponenten nicht am Anfang instanziert werden. Ebenfalls sollten Abbaumethoden vorhanden sein! Ein ViewMediator soll sich dann um die Navigation kümmern. Zentral soll so die Navigation wie auch die Listener abgehandelt werden.
Übereinstimmend ist man sich aber einig, dass das XML nicht eine Kopie der bereits bestehenden Application.e4xmi sein soll, sondern nur die ergänzenden Elemente beinhalten soll.

JHS findet wichtig, möglichst bald den ganzen Weg – vom XML bis zum Endpunkt – durchzuspielen, um zu sehen, ob es funktionieren kann. Anschliessend ist die komplette XML-Parse-Integration ein «nice to have», es könnte auch mit Hardcode-Werten weitergemacht werden.

JHS merkt an, dass die entwickelte Lösung offen, generisch, möglichst modular bleiben soll. Er denkt dabei direkt an die anderen, existierenden Plattformen iOS und Android, für die es ebenfalls eine Version von beook gibt. Auch sollen Konstanten und ähnliches immer möglichst lesbar bleiben, z.B. im Bereich von Key-Codes.

Durch die Schilderungen von CH verschiedener Problematiken wird auch klar, dass ihm noch Wissen im Bereich von Handlern/Listenern fehlt – was sich in diffusen Gedanken widerspiegelt. Link- und Literatur-Tipps von JHS und DS sollen helfen, möglichst schnell Klarheit zu schaffen. Probleme benennen und wie hier in diesem Fall das gewonnene, wichtige Wissen möglichst direkt in der Dokumentation festhalten.

4. Anpassung des Pflichtenhefts

Die Aufgabe «Erstellen und implementieren eines Keyboard-Access-Konfigurators» wird aus dem Pflichtenheft entfernt, da sich in der Analyse der Bachelor-Vorarbeit gezeigt hat, dass dieses Element kein Bedürfnis der Zielgruppe darstellt.
JHS ist damit einverstanden, DS auch.

Das Pflichtenheft wird online angepasst, ausgedruckt und vor Ort von den drei Parteien unterzeichnet.

5. Validierung von Plakat und Zusammenfassung

Plakat und Zusammenfassung werden von JHS und DS als aussagekräftig und genug detailliert angesehen. Es muss nicht unbedingt die technische Umsetzung erwähnt sein. Diese Dokumente können so eingereicht werden.

6. Festlegen der nächsten Fixpunkte und Pendenzen

- Einstellungen-JAWS kontrollieren (werden Schalter-Elemente und Hilfen komplett ausgesprochen, siehe Punkt 1)
- Aneignen von Wissen zu Listener/Handler in RCP eclipse.
- Aufzeichnen in möglichst klarem und gut lesbarem Schema, wie das Accessibility-Modul aussehen soll: Welche Klassen, welche Methoden, welche Properties, wo eingesetzt, Struktur des XML.
- Bis am 5. Juli sollte das Accessibility-Modul stehen und in das Proof of Concept integriert sein.
- Treffen am 9. Juli zu einer nächsten Besprechung mit JHS in Bern. Regelmässige Besprechungen bis ans Ende der Arbeit sind vorgesehen.
- Die Implementation des Moduls in beook sollte anschliessend in kurzer Zeit (2 bis 3 Tage) möglich sein.

7. Varia

Die Bachelor-Arbeit wird als Nicht-Vertraulich klassiert, da der Source Code von beook nicht Bestandteil der Bachelor-Arbeit ist. Von den Implementations-Schritten werden nur Screenshot-artige Auszüge in der Arbeit kommentiert. Das Dokument «Confidentialité liée au Travail de Bachelor» wird von den drei Parteien unterzeichnet.

Der Protokollführer und Organisator CH dankt dem begleitenden Dozent JHS und der begleitenden externen Fachperson DS herzlich für die genommene Zeit zu dieser Standortbestimmung.

Fragen an Alexander Hauser zu Accessibility, Telefon-Interview am 8. Juli 2015

Der Kontakt zu Alexander Hauser kam über Luc Fontolliet zu stande. Sie haben gemeinsam an einer Applikation gearbeitet, die für blinde und visuell eingeschränkte Kinder und Jugendliche entwickelt wurde. Die Applikation «Mouskie» dient zusammen mit einer Maus-artigen Zwei-Zeichen-Braillezeile zum Erlernen der Braille-Schrift.

Quel étaient les éléments sur lesquelles vous aviez fais le plus d'attention? Navigation, audio, possibilités d'output?
L'application a été prévu d'être accessible dès le début, alors c'était l'accessibilité en générale.

Avec quelles lecteurs d'écran avait vous travaillé?
Nous avons testé avec JAWS et Dolphine, deux Screenreader qui ne se comportaient pas toujours égale.

Concernant JAWS: Est-ce que tu avais trouvé une documentation pour développeurs de JAWS ou d'autres screenreaders qui t'as montrer les points d'accès?
En ligne nous n'avons pas trouvé des informations profondes sur JAWS. Aucune Documentation de développeurs pour des screenreader n'a été trouvé.

Principalement il fallait étudier les specifications de Microsoft. Malheureusement, ni Dolphin ni JAWS respectent ces spécifications – par contre les ATs native oui.

Alors c'était lié avec beaucoup d'essais, quels textes sont lus ou pas. Et nous sommes même tombés sur des effets bizarres: par exemple, si la taille de la police d'un button était trop grand (>24) JAWS ne lisait plus ce texte.

Une problématique étaient aussi des textes trop longues. JAWS s'arrêtait après 100 lettres sur certaines éléments. Alors dans la configuration de l'application nous avons fait attention que ces textes ne dépassaient pas les 100 charactères.

Nous avons laissé faire la géstion du focus par microsoft. Alors les événenemts sont pris correctement et les evenements «accessibility» sont appelés correctement.

Est-ce que tu te souviens d'avoir trouvé une méthode, pour JAWS faire dire qqch à des moments particuliers, disons des events interceptés...?
Ceci n'existe pas selon mes connaissances.

Pour le logiciel «Mouskie», quel était le langage programmé, quel étaient les composantes GUI utilisés?
J'avais etendu des classes d'éléments *GUI* de base. Pour regler les couleurs, tailles et ces particules «accessibilité» ceci était nécessaire. Je n'avait pas utilisé une librairie pour l'accessibilité. L'application a été écrit en C#.

J'ai vraiment les problèmes: apart du «setName» attribut et les labels, je n'ai j'amais eu un output vocale d'un texte comme par exemple «accessibilityPhrase», getDescription, getHelp ... Comment c'était fait chez vous?
Toute indication été «static» dans l'application avec des textes sur les bouttons et autres éléments. Comme déjà mentionné, à aucun moment un événement pouvait triggé une sortie vocale.

L'interface rélativement simple de Mouski: Est-ce que c'était du aux limitations?
Le client voulait une interface simple, pas seulement pour les enseignants mais aussi pour les utilisateurs effectifs, aveugles et malvoyantes. L'interface simple alors n'a pas été un résultat des limitations de possibilité. On évitait toute indication textuelle non nécessaire qui aurait pu distraire.

Integration des sons: Est-ce que c'était fait à travers les screenreader ou dans le logiciel?
Tous les sons sont gérés par l'applications. Ceci demandait beaucoup de travail au niveau applicatif.

Beaucoup investit dans la mise en forme, surtout les Couleurs, changement de contrastes: comment c'était fait?
La gestion des couleurs/contrastes a été géré dans l'application aussi. Avec des propriétés l'application contrôle le schema de couleur. Les couleurs de primaryBackground et de primaryText du Système sont lu et pris en compte. Mais elles peuvent être écrasées.

Merci d'avoir répondu aux questions et de confirmer la situation des possibilités limités d'accès aux Screenreaders.

Gute Labels setzen – Mail-Wechsel mit Selamet Aydogdu, 20. Juli 2015

Fragen per E-Mail an Selamet Aydogdu
Betreffend Labels, also gesetzten Texten, die vom Screenreader vorgelesen werden um Schaltflächen oder ähnliches zu beschreiben hätte ich folgende Fragen:

- Was macht für dich ein gutes Label aus?
- Ist es dienlich, wenn in dieser Information beschrieben wird, was durch drücken passiert?
- Würdest du mir ein paar gute und ein paar schlechte Beispiele notieren? Es spielt keine Rolle, wenn der Wortlaut nicht zu 100% stimmt.

Antwort von Selamet
Gerne beantwort ich deine Fragen:

Für mich ist ein gutes Label, wenn es genau sagt, was passiert, bevor man darauf klickt. Dabei unterscheide ich zwischen Adverb/Adjektiv und Verb. Wird ein Adjektiv verwendet, so sagt die Beschriftung den Zustand des Labels an. Zum Beispiel: An/Aus. Bei einem Verb jedoch, was passiert, wenn man darauf klickt: Ausschalten, Anschalten, Abbrechen, Speichern, usw.

Gute Beispiele sind die vorangehenden Beispiele. Schlecht wäre aber zum Beispiel: Ausgeschaltet, Aktiviert, Verbindung getrennt.

Gut wäre aber: Ausschalten, aktivieren, Verbindung trennen.

Noch ein weiteres Beispiel: Nehmen wir an, der Zustand eines Schalters ist «on». Dann kämen folgende Beschriftungen in Frage: «an», «ausschalten». Wäre der Zustand des Schalters «off», so kämen folgende Beschriftungen in Frage: «aus», «einschalten».

CD mit Daten

Auf der beigelegten CD befindet sich die gesamte Arbeit mit den kompletten Anhängen als PDF. Profitieren Sie von einer angenehmen Lektüre dank Lesezeichen, Querverweisen und Hyperlinks.

Zusätzlich sind die folgenden Daten auf der CD:

- **Proof of Concept Anwendung**: *Proof of Concept* als ausführbare .exe-Datei für Windows.
- **Proof of Concept Quellcode**: Der komplette Quellcode des *Proof of Concepts*. Dieses Projekt kann in die *eclipse* Entwicklungsumgebung geladen und von dort ausgeführt werden.
- **Accessible Modul Quellcode**: Der komplette Quellcode des *Accessible Modul* Plugin-Projekts.
- **beook**: Das erstelle *beook*Accessible.xml, welches die Struktur für die Funktion des AccessibleModuls vorgibt.

Zusammenfassung

Eine bestehende Desktop-Applikation barrierefrei zu machen verlangt den Spagat zwischen den Ansprüchen an eine optimale User-Experience und dem Zusammenspiel bestehender, eingesetzter Technologien. Nebst einer Navigation, die komplett über die Tastatur möglich sein muss, ist die «Lesbarkeit» der Applikation für Screenreader zentral.

Die Arbeit zeigt Erkenntnisse, Möglichkeiten und Implementationsschritte, durch welche man die Barrierefreiheit von *eclipse* RCP-Applikationen verbessert. Diese sind der Grundstein, um die bestehende Applikation für digitale Lehrmittel «*beook*» blinden und visuell eingeschränkten Personen zugänglich zu machen. Dank einer durchdachten Tastatursteuerung profitieren aber auch Power-User von dieser Erweiterung.