WebMerlin - Ergonomie

Verbesserung der Usability einer Anwendung zur Erstellung von Hörfunk-Rahmenplänen

Software-Projekt
von
Martina Blank
Wintersemester 2005/2006

HOCHSCHULE DER MEDIEN

Inhalt dieser Dokumentation

externe Dokumente

SourceCode & Co. WebMerlinSources.zip
Da es sich bei den Quellen von WebMerlin um Eigentum des SWR handelt, habe ich lediglich die von mir direkt bearbeiteten Klassen entnommen und als Zip-Archiv zusammengestellt.

1. Allgemeines

Studiengang:	Medieninformatik, 7. Semester
Kurs:	13139 - Praktikum Softwaretechnik 2
Zeitraum:	Wintersemester 2005/2006
Teilnehmer:	Martina Blank - mb068@hdm-stuttgart.de
betreuender Dozent:	Prof. Jens-Uwe Hahn
Kurzbeschreibung des Projekts:	WebMerlin ist eine Anwendung zur Erstellung von Hörfunkrahmenplänen. Im Rahmen des Software-Projekts sollten Verbesserungen an der Ergonomie durchgeführt werden, z.B. Ergänzung von Tastatur-Shortcuts. Vorab sollte eine User-Befragung durchgeführt werden.
Betriebsmittel:	NetBeans, Java (AWT, Swing), JIRA, JBoss, CVS, ANT, Oracle, Java Web Start (Nutzung der Betriebsmittel und Räume des SWRs)
zeitlicher Ablauf des SW-Projekts:	bis KW 48/05 • Einarbeitung in die Anwendung WebMerlin und in Java Swing • Userbefragung • Erstellen einer Ergonomie-Studie • SWR-interne Abklärung der umzusetzenden Änderungen 03.11.2005 • Besuch des World Usability Day in Stuttgart mit SWR-Kollegen. Im Rahmen dieser Veranstaltung hatte ich Gelegenheit einige sehr interessante Vorträge zu hören, u.a. einen von unserem Herrn Burmester (Fakultät 3, Studiengang Informationsdesign) und einen von Frau Prof. Astrid Beck von der FH für Technik in Esslingen zum Thema "Wie entwickelt man benutzerfreundliche Software? Eine kurze Einführung in das Thema Software-Usability" Sehr interessant zu sehen was andere FHs mit Informatik-Studiengängen in der Richtung machen. bis KW 03/06 • Implementierung der Änderungen KW 04/06 • Bericht und Präsentation für Projekt erstellen vorgesehene Gesamtarbeitszeit: 11 Tage à 8 Stunden im Betrieb des SWR (tatsächlich habe ich 25 Tage an dem Projekt gearbeitet)

2. Einführung

Die Anwendung WebMerlin ist ein Programm des Südwestrundfunks (SWR) zur Planung und Erstellung von Hörfunk-Sendeplänen. WebMerlin befindet sich bereits seit 2001 im Betrieb, wird aber noch weiterentwickelt (letzte Version 4.0 ging Ende 2005 in Test-Betrieb und soll Ende Januar 2006 in Produktiv-Betrieb gehen) und den Anwenderbedürfnissen angepasst.

Im Rahmen des Software-Projekts sollten Verbesserungen an der Ergonomie durchgeführt werden. Die verschiedenen Masken sollten um Mnemonics bzw. weitere Shortcuts ergänzt werden. Bereits vorhandene Tastaturkürzel sollten vereinfacht/vereinheitlicht werden. (Bsp. Taste ENTER für alle Speichern/Ja/OK-Aktionen).
Vorab sollte eine User-Befragung (Ergonomie-Studie) durchgeführt werden, da im Juni 2005 ein neuer Anwenderkreis hinzugekommen ist, der bereits in Schulungen einige Änderungswünsche geäussert hatte. Die Ergebnisse der Studie mussten mit den verschiedenen Rundfunkanstalten (RFAs) abgestimmt werden. Die Ergebnisse sollten dann implementiert werden.

3. Anwendung

Die Anwendung WebMerlin wurde erstmals im Jahr 2001 (damals noch unter der Bezeichnung "Radio mit System" (RMS) beim SWR eingeführt. Es handelte sich um eine Eigen-/ Auftragsprogrammierung in Zusammenarbeit mit der Firma GFT Solutions.

GFT hat auf ihrer Webseite eine Liste ihrer Kunden und den bei ihnen realisierten Projekten hinterlegt. Dort beschreiben sie WebMerlin folgendermassen:

Herausforderung
- Unterstützung des vollständigen Hörfunk-Prozesses: Planung, Produktion, Abrechnung, Auswertung und Dokumentation
- Integration bestehender Systeme

Lösung
- Integriertes Planungs- und Workflowmanagementsystem
- Java-Applikation als Benutzer-Interface, Web-basierte Administration
- 3-Schicht Architektur mit Applikationsserver
- SOAP-Ansatz ab Version 2

Leistung
- Anforderungsmanagement
- Design & Informations-Architektur
- IT-Consulting & - Implementierung
- Projektplanung und Organisation
- Anwendung CMMI Qualitätsmodell

GFT arbeitet auch heute noch mit einigen Entwicklern an WebMerlin weiter, die Hauptentwicklungstätigkeit findet jedoch im Hause SWR selbst statt, genaugenommen in der Abteilung Anwendungssysteme der IKS (Hauptabteilung Informations- und Kommunikationssysteme), in der ich auch meinen Arbeitsplatz habe.

Die Quellcode-Dateien der Anwendung WebMerlin bilden ein sehr umfangreiches und für den Laien völlig unüberschaubares Konstrukt von Verzeichnissen, Unterverzeichnissen und Dateien. Wenn man das erste Mal die Quellen aus dem abteilungseigenen CVS auscheckt, hat man prompt 500 MB mehr auf der Festplatte. Insgesamt gibt es ca. 6.000 verschiedene Dateien, natürlich werden es täglich mehr. Davon sind ca. 1.600 Java-Klassen, die in etwa 170

verschiedenen Packages und Unter-Packages angeordnet sind. Über die Jahre hat sich natürlich auch viel Dokumentations-Material angesammelt.

Grob untergliedern lässt sich die Anwendung in die folgenden Komponenten:

Die **Client**-Komponente beinhaltet im wesentlichen sämtliche Model-View-Controller-Konstrukte, mit denen die Funktionalität zur Darstellung von Daten und der Weitergabe von Interaktionen der Benutzer abgebildet wird. Zu jedem Fachobjekt auf dem Server existiert ein korrespondierendes Model auf dem Client. Für die Darstellung der Objektinhalte auf dem Bildschirm existiert mindestens eine View-Klasse, optional auch weitere. Die auftretenden Ereignisse (Laden von Daten , Benutzerinteraktionen) werden durch entsprechende Listener-Objekte verarbeitet. Hinzu kommen lokales Session Management, Fehlerbehandlung, u.ä. Die Kommunikation zu den serverseitigen Objekten (Geschäftsprozesse, Fachobjekte) erfolgt über SOAP (Simple Object Access Protocoll). Um eine Entkopplung von cientseitigen Objekten von einem bestimmten Server zu erreichen, erfolgt der clientseitige Zugriff auf Attribute und Methoden über eine Schnittstelle die verschiedene Implementierungen besitzt (Bridge). Dadurch wird eine einfache Möglichkeit geschaffen, zwischen verschiedenen Anbietern von Services (meist Applikations-Server), aber auch zwischen simuliertem (offline) und realem (online) Betrieb umzuschalten.

In der **Server**-Komponente erfolgt die eigentliche Verarbeitung aller zu bewegenden Daten, d.h. die Serverkomponente ist für die gesamte Geschäftslogik mit Transaktionssteuerung und Persistenzsicherung zuständig. Dazu gehören alle Mechanismen für die Bereitstellung und Kontrolle der Geschäftsprozesse sowie aller Fachobjekte. Die Serverkomponente gliedert sich in eine Persistenzschicht, die die Fachobjekte beinhaltet und Methoden für das Speichern und Lesen dieser Objekte bereitstellt und die Logikschicht, die die Geschäftsprozesse beinhaltet. Die Logikschicht ist nach fachlicher Zusammengehörigkeit strukturiert.

Die **Datenbank**-Komponente beinhaltet das Datenbankschema mit allen zugehörigen Zwangsbedingungen. Der Zugriff darauf erfolgt über SQL Statements. Verarbeitungslogik kann in der Datenbank in Ausnahmefällen, z.B. zur Steigerung der Performance, implementiert werden.
Geschäftsprozesse sind als Enterprise Java Beans (EJBs) realisiert. Damit erfolgt eine Trennung zwischen z.T. komplexer Low Level Programmierung (State Management, Multi-Threading) und der eigentlichen Implementierung der Geschäftsprozesse, wodurch leichter wartbarer Code und ein höherer Grad an Wiederverwendbarkeit erreicht wird.

Da sich meine Änderungen lediglich auf die Client-Komponente bezogen haben, will ich im Folgenden nur den Client etwas genauer betrachten.

Beim Client wurde die fachliche Trennung als Basis der Package-Struktur gewählt, die Trennung von Model/View/Controller wurde mittels der Namensgebung der Klassen erreicht. Dies sieht man später auch anhand von Beispielen, die ich bei der Implementierung meiner Änderungen eingefügt habe. View-Klassen tragen die Endung "View/Panel/Dlg", Model-Klassen die Endung "Model", alle weiteren Klassen werden entsprechend ihrer Eigenschaften benannt, z.B. DataAccess, Action, Factory, Item, Node, etc.

Im folgenden werden die vorhandenen Client-Packages kurz erläutert:

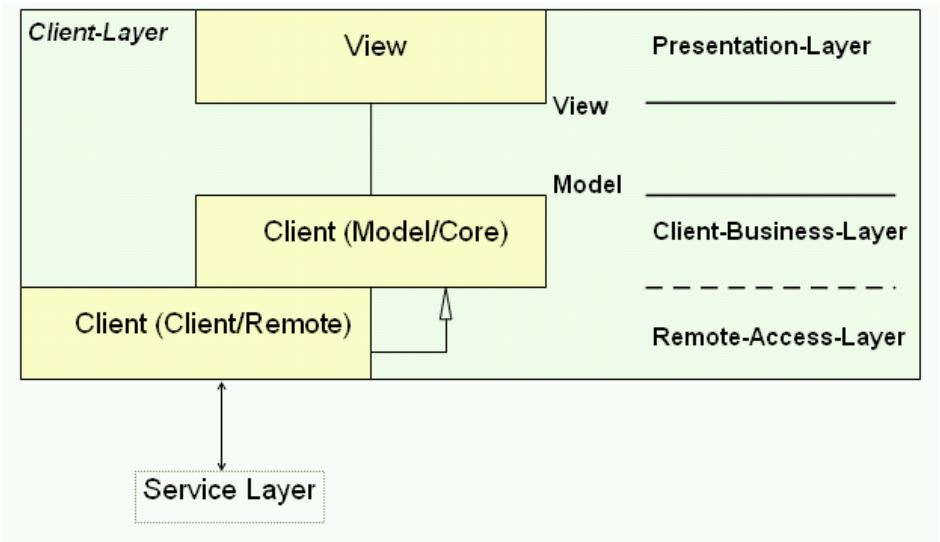
`de.ard.sad.client`	Workbench Klassen, wiederverwertbare, fachlich unabhängige Klassen und Interfaces zur Wiederverwendung in verschiedenen Anwendungen (Basiskomponenten)
`de.ard.sad.client.actions`	Workbench Actions, Navigator Actions
`de.ard.sad.client.com`	Klassen zum Kommunikationsaufbau, Kommunikations-Exceptions
`de.ard.sad.client.beitrag`	Klassen des Moduls *Beitrag*
`de.ard.sad.client.beitrag.actions`	Actions des Moduls *Beitrag*

de.ard.sad.client.laufplan	Klassen des Moduls *Laufplan*
de.ard.sad.client.laufplan.actions	Actions des Moduls *Laufplan*
de.ard.sad.client.raster	Klassen des Moduls *Raster*
de.ard.sad.client.raster.actions	Actions des Moduls *Raster*
de.ard.sad.client.recherche	Klassen der Recherche-Funktionalität
de.ard.sad.client.tester	Test Klassen (DB, SOAP)
de.ard.sad.utilities	Klassen, die keine fachlichen Abhängigkeiten besitzen, und dennoch nicht zu den Basiskomponenten zählen, da sie überall eingesetzt werden können. Dies sind Klassen die eine hohe Wiederverwertbarkeit mit sich bringen. Oft handelt es sich hierbei um Klassen, die wiederholende Funktionalitäten zusammenzufassen (oft verwendete Oberflächenkomponenten oder interne Programmfunktionalitäten, wie z. B. das Error-Handling).
de.ard.sad.utilities.admin	über WebMerlin hinaus nutzbare Administrations-Klassen (z.B. Login)
de.ard.sad.utilities.security	über WebMerlin hinaus nutzbare Klassen rund um Sicherheitseinstellungen
de.ard.sad.utilities.soap	über WebMerlin hinaus nutzbare Klassen zur Serialisierung

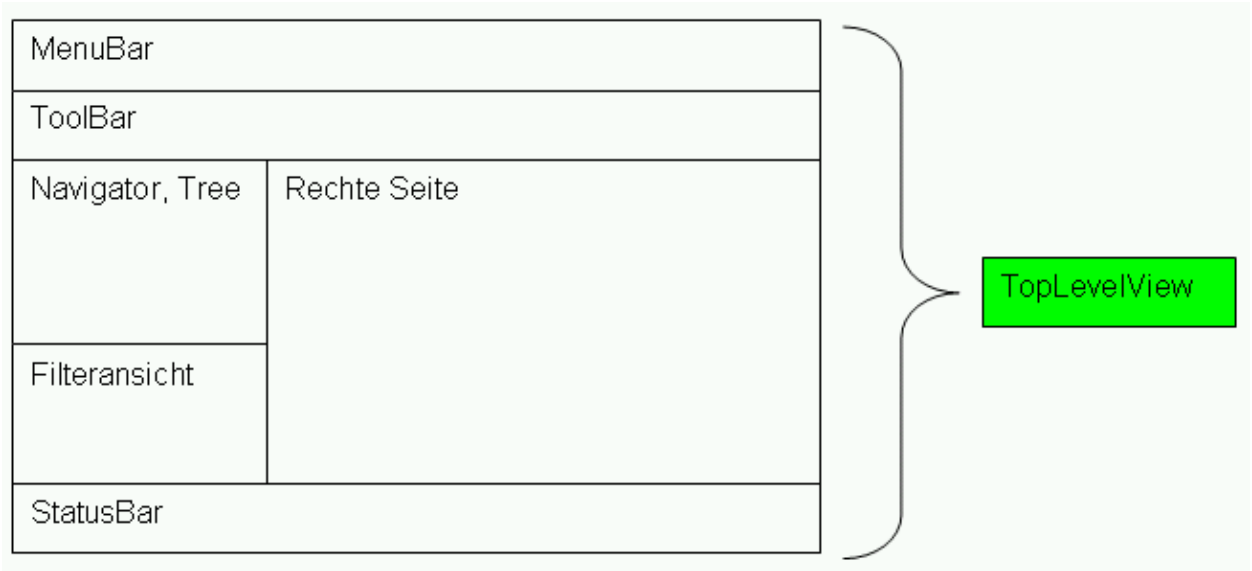
Abstrakte Klassen tragen das Präfix *A*, Interfaces das Präfix *I*.

Schichten
Innerhalb des Clients sind 3 Schichten identifizierbar (siehe auch in der Grafik):

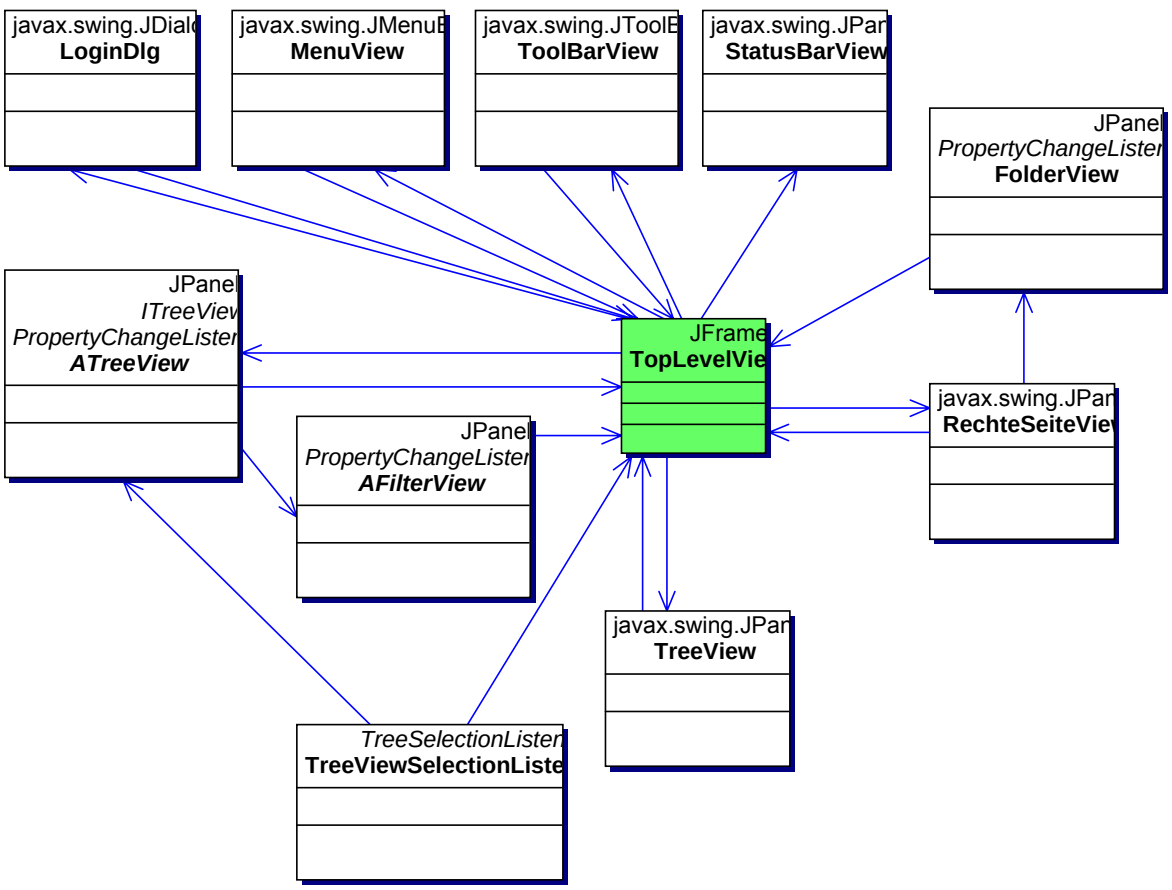
- View- oder Präsentations-Schicht
- Model- oder Business-Schicht
- Data Access oder Zugriffs-Schicht



Die Einteilung der Workbench (der WebMerlin-Anwendungsoberfläche) aus Sicht der Präsentationsebene:



Diese Einteilung entspricht in etwa den folgenden Klassen:



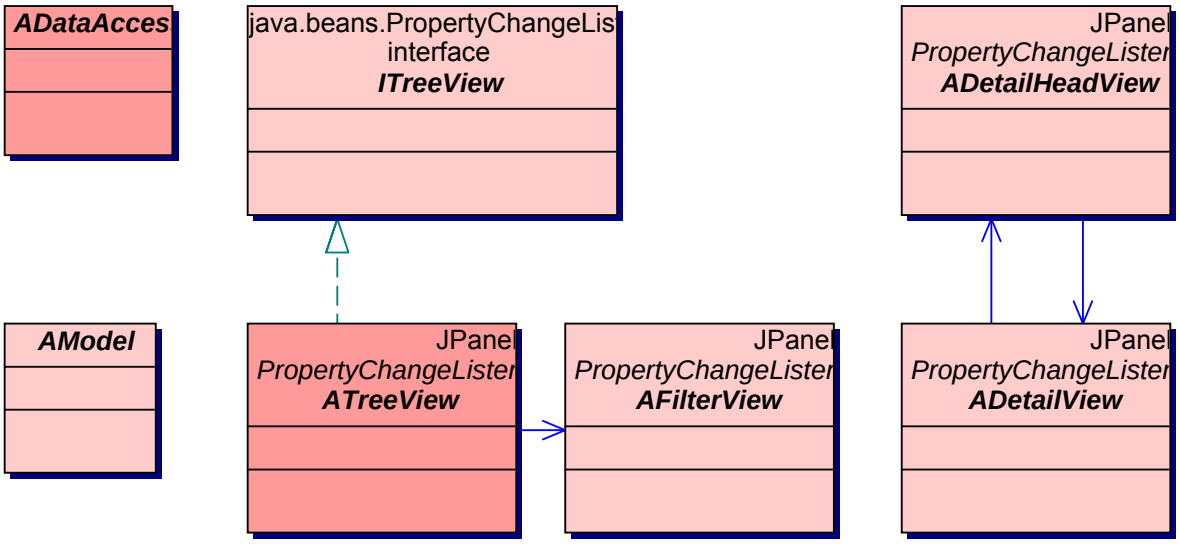
Die Klasse TopLevelView ist keine View-Klasse im herkömmlichen Sinn, da sie selbst keine Oberfläche darstellt, sondern lediglich alle "Haupt"-GUI-Komponenten der Workbench aggregiert.

Wichtig: Alle Komponenten aggregieren umgekehrt auch die einzig existierende Instanz der TopLevelView und geben diese auch wiederum an ihre Unter-Komponenten weiter, so dass jede View-Klasse diese Instanz kennt und benutzen kann.

Aufgaben der TopLevelView:
- Bestimmung der Reihenfolge der Instanziierung
- initiales Laden des Navigators, entsprechend dem eingestellten Filter
- Kommunikation zwischen selektiertem Objekt im Navigator und rechter Seite
- Instanziierung und Verwaltung der Module
- Verwaltung der Model-Instanzen
- Kontext-Verwaltung
- Laden und Verwalten von Stammdaten

Die Workbench stellt aber auch noch andere Klassen zur Verfügung, welche von den Modulen benutzt werden sollten:
- de.swr.rms.client.AModel – Superklasse für Models
- de.swr.rms.client.AFilterView – Superklasse für ein Panel mit Filtereinstellungen
- de.swr.rms.client.ADetailView – eine rechte Seite mit mehreren Ordneransichten
- de.swr.rms.client.ADetailHeadView – eine Ordneransicht mit Kopfdaten

Diese Grafik zeigt eine Übersicht aller API-Klassen der Client-Komponente:



View-Klassen bauen die Präsentationsschicht des Clients auf. Erkennbar sind sie durch die Erweiterung "View" am Ende des Dateinamens. Sie sind auf Komponentenbasis aufgebaut, d.h. grössere Komponenten bauen sich aus kleineren auf. Typischerweise importieren sie die Packages

```
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;
import java.awt.event.*;
```

Viele innerhalb der Module eingesetzten View-Klassen leiten sich von der Klasse `ADetailHeadView` ab, welche folgende, typischen Methoden vorschreibt:

```
public abstract boolean updateModel();
public abstract void propertyChange(PropertyChangeEvent e);
public abstract void removeDlgAsModelListener();
public abstract void addBearbeitenMouseListener(MouseListener ml);
public abstract void clearAll();
```

Die `updateModel()`-Methode liest die Daten der GUI Eingabefelder, instanziiert ein Daten-Objekt und steckt es in das entsprechende Model. Diese Methode wird immer dann aufgerufen, wenn der User z.B. "Speichern", "OK" o.ä. aufgerufen hat.
Die `propertyChange()`-Methode bringt die Daten des Models zur Ansicht. Diese Methode wird immer dann aufgerufen, wenn sich das Model geändert hat.

Weiterhin fangen die View-Klassen auch ihre Oberflächen-Events ab. Hierbei wird allen Oberflächenelementen mit Events eine eigene Klasse zur Verfügung gestellt.

Bsp.: Eine OK-Schaltfläche

```
public class JbtOkListener implements ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {
       try{
             .......

             } catch( Exception x ) {

                ErrorHandler.getInstance().handleException(x);
             }
        }

    }
```

Diese Klasse wird als innere Klasse in die View-Klasse eingefügt.

Actions wurden als eigene Klassen ausgelagert. Der Zugriff auf die Actions darf nur über die Klasse „ActionFactory“ mittels *`ActionFactory.getInstance()`*... erfolgen. Die Actions sind abgeleitet von javax.swing.AbstractAction.

Property-Dateien dienen zur variablen Einstellung des Clients über Textdateien. Die Property-Dateien in WebMerlin bestehen ausschliesslich aus Paaren der Form: key = value. Im Client werden die folgenden Property-Dateien unterschieden:

system.properties	Pfade, Bildschirm Defaults, ...
server.properties	Serveradresse, DB Treiber, Ports, ...
errorMessages.properties	Warnungen, Fehlermeldungen, ...
general.properties	sprachabhängige Einstellungen der Workbench
menus.properties	Mnemonics, Accelerators, ...
beitrag.properties	sprachabhängige Einstellungen des Moduls Beitrag
laufplan.properties	sprachabhängige Einstellungen des Moduls Laufplan
raster.properties	sprachabhängige Einstellungen des Moduls Raster

Möchte man den Text z.B. für ein Label setzen zu einem Key k1 in der Property-Datei p1, so lautet der Aufruf/Pfad im Quellcode letztendlich so:

```
myLabel.setText( java.util.ResourceBundle.getBundle("lib/p1").getString("k1") );
```

Shortcuts und weitere Navigation

Shortcuts sind ein Hilfsmittel, um eine Benutzeroberfläche allein durch Tastaturbefehle, d.h. ohne Verwendung einer Maus, bedienen zu können. Grundlage für die in WebMerlin realisierten Shortcuts sind einerseits Windows-übliche Shortcuts

Gebräuchliche Shortcuts in Windows-Anwendungen

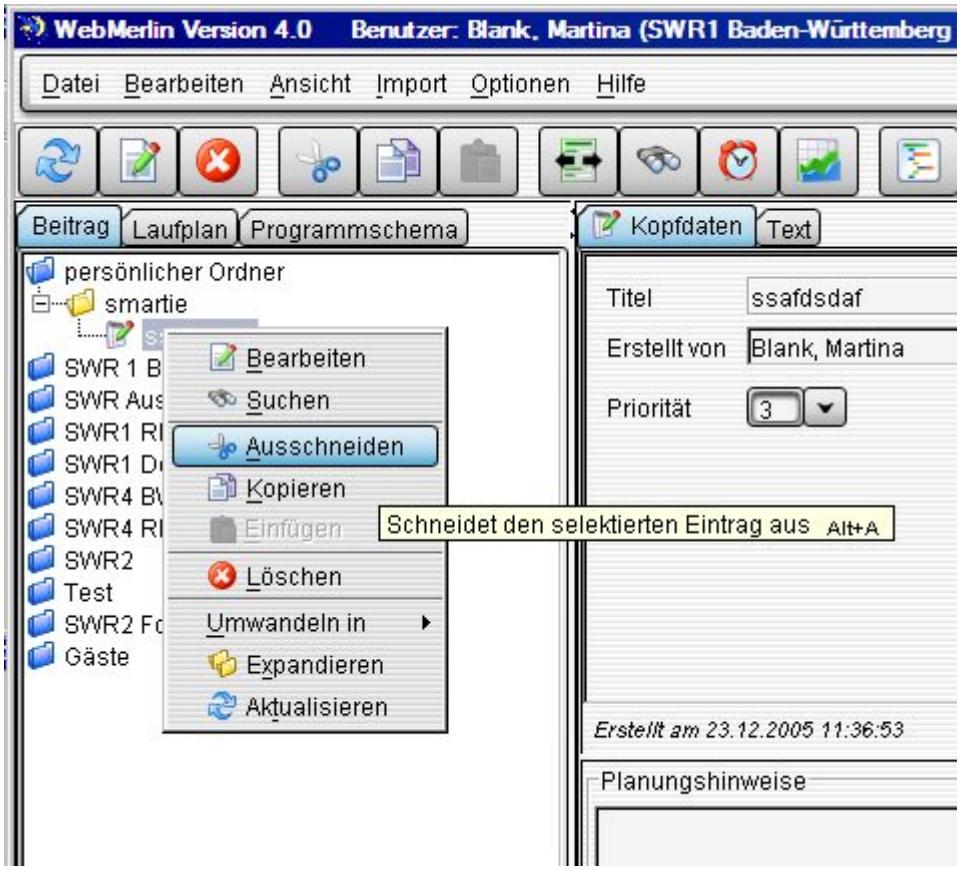
Tastenkombination	Bedeutung
[F1]	Hilfe
[F2]	Umbenennen
[F3]	Suchen
[F4]	Klappt die Dropdown-Liste im Focus auf
[Alt] [F4]	Schliesst die Anwendung
[F5]	Aktualisiert die Ansicht
[F6]	Zwischen Fensterausschnitten wechseln
[F10]	Wechselt in den Menümodus
[Strg] [X]	Ausschneiden
[Strg] [C]	Kopieren
[Strg] [V]	Einfügen
[Strg] [G]	Gehe zu
[Strg] [Ende]	Zum Ende eines Dokuments springen
[Strg] [Pos1]	An den Anfang eines Dokuments springen
[Strg] [Z]	Rückgängig
[Strg] [A]	Alles markieren
[Alt] [Tab]	Wechselt zu einem anderen laufenden Programm

sowie der Wunsch, häufig genutzte Funktionen direkt über Tastenkombinationen aufrufen zu können. Grundsätzlich sind alle Hauptmenüs und alle Menüunterpunkte durch Tastaturbefehle aufrufbar, wesentliche Funktionen besitzen auch einen direkten Shortcut.

Beispiel: Das Hautpmenü Datei wird durch die Tastenkombination Alt + D aufgerufen, durch Eingabe von N erreicht man das Untermenü Neu, indem man schlussendlich durch Eingabe von O die Funktion "Ordner" (neuen Ordner anlegen) zur Ausführung bringen kann. Alternativ startet man diese Funktion durch den Shortcut Strg + Alt + O von der Menüstruktur in der Anwendung. Die implementierten Shortcuts erkennt man somit durch die unterstrichenen Buchstaben im Menüeintrag bzw. durch die am Ende des Eintrags angegebene Tastenkombination:

Bei der Festlegung der Tastenkombinationen wurden nach Möglichkeit die Anfangsbuchstaben des entsprechenden Menüeintrages verwendet, Shortcuts zur Neuanlage von Objekten sind üblicherweise von der Form Strg + Alt + Anfangsbuchstabe, die übrigen Dialoge werden meistens durch Strg + Umschalt + Anfangsbuchstabe aufgerufen. Die Shortcuts für die Funktionen Copy, Cut und Paste von Baumeinträgen beinhalten zusätzlich die Alt-Taste; dies ist notwendig, um sie von den üblichen Windows-Shortcuts für Copy, Cut und Paste von Texteinträgen zu unterscheiden.

Im Beitragsbaum bzw. Laufplanbaum (siehe linker Rahmen im folgenden Screenshot) kann man das Kontext-Menü entweder mit der rechten Maustaste oder dem Shortcut Umschalt + F10 öffnen.



Tastaturbefehle zur Textbearbeitung
Markierte Textteile innerhalb von Eingabefeldern können wie auch bei Windows üblich mit Strg + X, Strg + C bzw. Strg + V ausgeschnitten, kopiert bzw. eingefügt werden. Mit Pos1 bzw. Ende gelangt man an den Anfang bzw. das Ende einer Textzeile, mit Strg + Pos1 bzw. Strg + Ende an den Anfang bzw. das Ende des jeweiligen Textelements. Für die Formatierung markierter Textteile sind die Shortcuts Alt + b (fett), Alt + i (kursiv) und Alt + u (unterstrichen) verfügbar.

Tastaturbefehle zur Navigation
Der Wechsel zwischen den Modulen Beitrag, Laufplan und Programmschema (siehe Tabs im obigen Screenshot im linken Rahmen) kann durch die Shortcuts Alt + 1, Alt +2 und Alt +3 erfolgen (Voraussetzung: Focus liegt auf einer der drei entsprechenden Registerkarten).

Allgemein kann zwischen zusammengehörenden Tabs mit den Pfeiltasten ← und → gewechselt werden (sofern der Focus „oben“ auf dem Reiter sitzt), analog wechselt man zwischen den Hauptmenüs. Durch Hinzunahme der Pfeiltasten ↑ und ↓ kann man sich komplett durch die Menüstruktur bzw. die Beitrags- und Laufplanbäume hangeln. Mit Hilfe von TAB bzw. Shift + TAB kann man den Focus innerhalb eines Dialogs von links nach rechts und oben nach unten (bzw. rückwärts) wandern lassen, Komboboxen werden mit den Pfeiltasten bedient, Checkboxen mit der Leertaste, Schaltflächen mittels Return. Können in einem Textfeld Tabulatoren gesetzt werden (falls es sich also im Sinne von Java um eine TextArea und nicht um ein TextField handelt), so ist dieses mit Strg + TAB zu verlassen! Beide Arten der Text-Eingabefelder wurden in der Anwendung verwendet.

Die Schaltflächen "Speichern", "Speichern & Schliessen" und "Abbrechen" in allen Bearbeiten-Dialog-Fenstern sind mit dem Shortcut ALT + unterstrichener Buchstabe versehen; entsprechend sind bei den übrigen Schaltflächen in der Regel Shortcuts eingebaut. ALT + F4: schliesst den Dialog, der gerade den Focus besitzt.

4. Ergonomie-Studie

Vorbedingungen:
Die von mir in der Ergonomie-Studie zu betrachtende Anwendergruppe umfasste die Redaktionsassistenten der verschiedenen Radio-Sender des SWR, da diese während der Datenerfassung und Nachbearbeitung der Sendepläne sehr intensiv mit der Benutzeroberfläche von WebMerlin arbeiten. Die von mir besuchte Abteilung in Mainz besteht aus etwa 20 Anwendern (12 vom SWR1, 9 vom SWR4). Weiteren Input für meine Studie lieferten die direkt aus dem User-Umfeld stammenden Mitglieder der WebMerlin-Projektgruppe.

Bei der Untersuchung war spezielles Augenmerk gelegt auf:
- detaillierte Dokumentation des Workflows, speziell bei Bedienung über die Tastatur
- Festhalten des (bisherigen/gewünschten) Focus-Wechsels für alle Masken

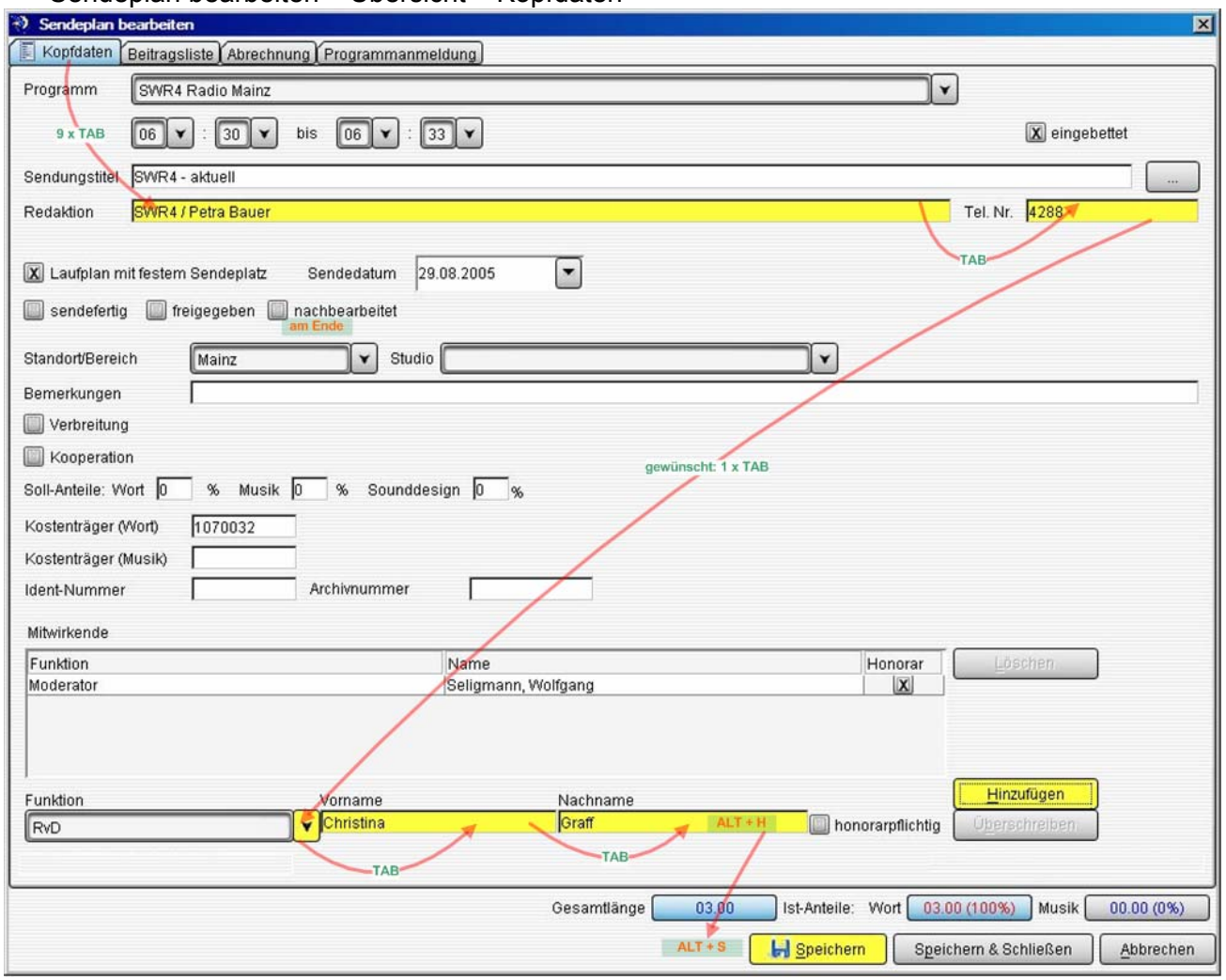
Primär relevante Masken:
- Sendeplan bearbeiten – Übersicht – Kopfdaten
- Sendeplan bearbeiten – Übersicht – Beitragsplan (mit/ohne Beitragsbaum/Laufplan)
- Neuer Wortbeitrag – Kopfdaten
- Kostenträger-Auswahl
- Neue Moderation – Kopfdaten
- Sendeplan bearbeiten – Übersicht – Abrechnung
- Honorarposition bearbeiten – Sendedaten/Vertragsdaten, Beschäftigungszeiten bearbeiten
- Nachrichtenbeitrag

Ich habe hier nicht alle Aussagen der Anwender gelistet, sondern nur die wirklich repräsentativen. Eine Anwenderin äusserte z.B., dass sie sich für ein bestimmtes Eingabefeld in der Maske Wortbeitrag eine Default-Vorgabe einer Kostenträgernummer (1070131 = Schlagzeilen) wünschen würde. Allerdings gelte dies nur für die SWR4Sendung vormittags ab 05:00h. Hier wird sicher klar, dass man bei der Auswahl von zu implementierenden Änderungen schon filtern muss.

Ergebnisse:
Wie erwartet wird das Programm WebMerlin von den Redaktionsassistenten sehr unterschiedlich bedient. Manche Anwender benutzen grösstenteils die Maus und navigieren lediglich bei direkt hintereinanderliegenden Eingabefeldern mit der TAB-Taste. Andere hingegen setzen fast nur die Tastatur ein (oder versuchen es zumindest) und greifen nur, wenn es gar nicht mehr anders geht zur Maus. Dazwischen liegt eine fliessende Grauzone.
Vor allem die Tastatur-Liebhaber haben eine Reihe Anregungen geliefert, an welchen Stellen neue Tastaturkürzel (Shortcuts, Mnemonics) oder eine Änderung der (Cursor-) Focusierung sinnvoll wäre.
Interessant war für mich, dass die Meinungen der Entwickler und Anwender darüber, was sinnvoll oder wirklich erforderlich wäre, sehr auseinandergehen. Einige der Ideen der Entwickler, die ich in Mainz vorgebracht habe, wurden dort als nicht notwendig angesehen. Dafür tauchten dort andere Wünsche auf, die die Entwickler bisher noch nicht berücksichtigt hatten.

Es folgt eine Betrachtung der gebräuchlichsten Eingabemasken aus der vorhergehenden Liste im einzelnen und ein allgemeiner Teil mit weiteren Ideen und Anregungen.

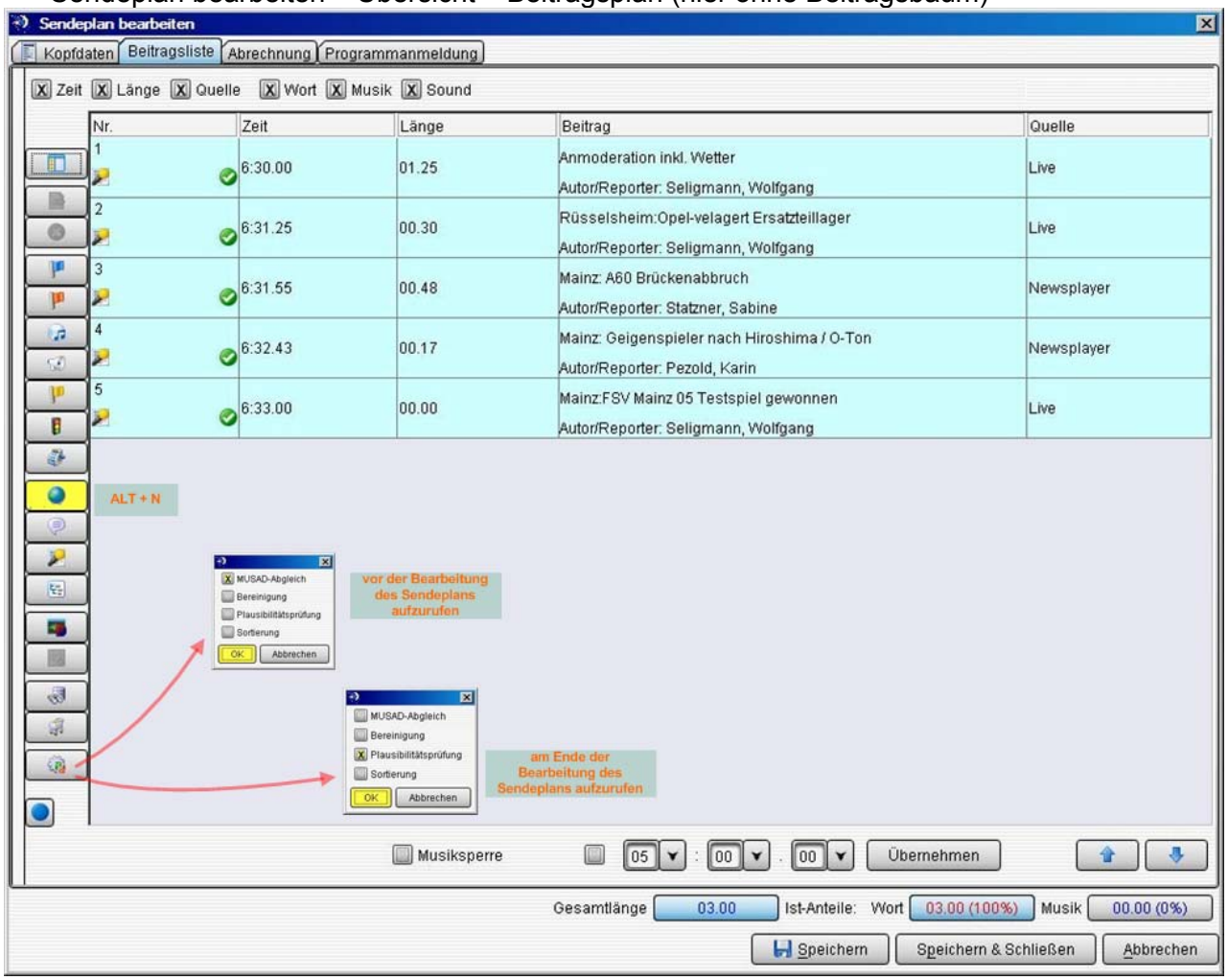
- Sendeplan bearbeiten – Übersicht – Kopfdaten



Zu dieser Maske wurde vorgeschlagen, dass der Cursor nach Eingabe der Telefonnummer direkt auf das Auswahlfeld "Funktion" springt. Die anderen Eingabefelder dazwischen werden in der Regel nicht bearbeitet.

Hinweis einer Anwenderin: Wenn bereits ein Mitwirkender eingegeben wurde (dieser ist im Feld Mitwirkende noch markiert und dessen Name wird im unteren Bereich noch angezeigt) und man unten beginnt einen neuen Mitwirkenden zu erfassen in dem man die "Funktion" abändert, steht im Namensfeld immer noch der Name des oben markierten Mitwirkenden. Eigentlich hätte sie erwartet, dass die Auswahl einer Funktion eine Neueingabe initialisiert und die Felder rechts davon gelöscht werden. Solange der obige Eintrag markiert ist, wird vom Programm aber wohl vorausgesetzt, dass dieser überschrieben werden soll.

- Sendeplan bearbeiten – Übersicht – Beitragsplan (hier ohne Beitragsbaum)



Aus dieser Ansicht heraus (im Bearbeiten-Modus des Sendeplans) werden die verschiedenen Beiträge der Sendepläne nachbearbeitet.

Der Vorschlag Shortcuts für die Icons (die Button-Reihe im folgenden Screenshot links) einzuführen wurde durchweg positiv aufgenommen (Vorschläge siehe nächstes Bild).

weiterhin sind vergeben:

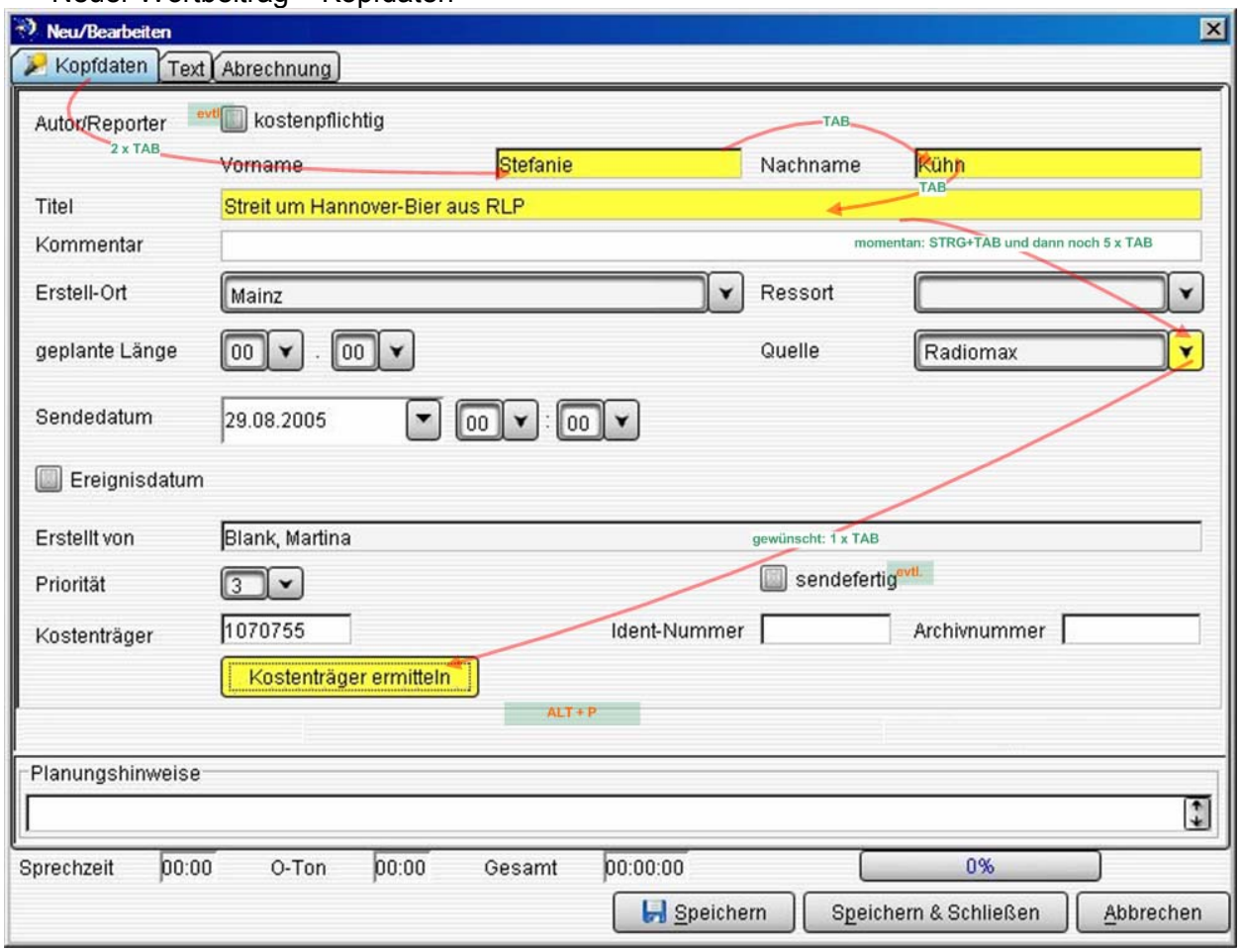
ALT + S **Speichern**

ALT + P **Speichern & Schließen**

ALT + A **Abbrechen**

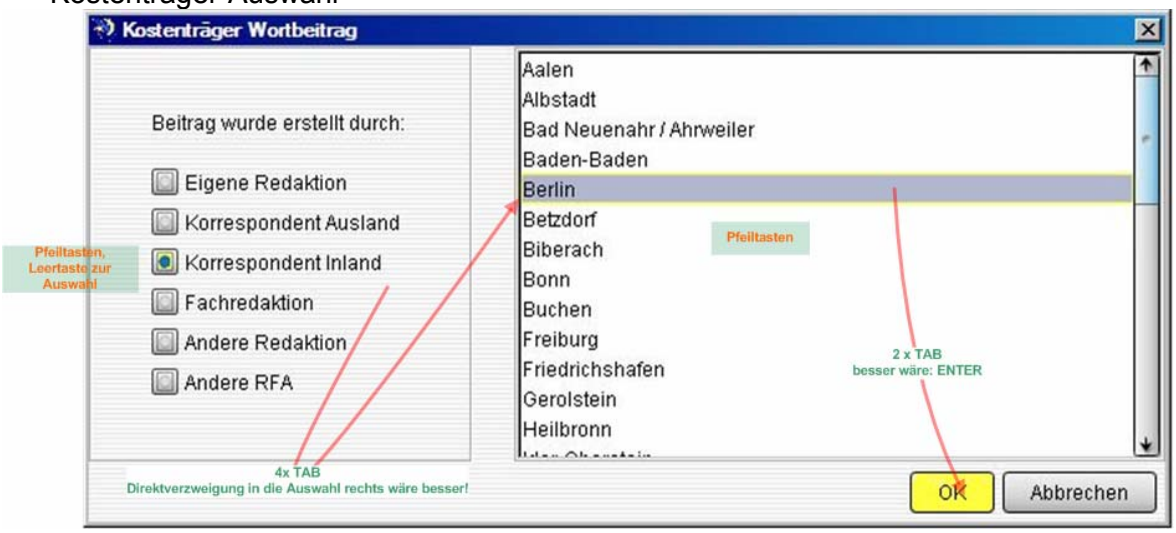
Hier muss geprüft werden, ob in jeder Ansicht/Maske, in der der Beitragsbaum auftaucht und ansprechbar ist, die links gewählten Shortcuts noch frei sind!

- Neuer Wortbeitrag – Kopfdaten



In dieser Maske werden nur sehr wenige Felder tatsächlich benötigt. Daher kam der Wunsch auf, dass man bei der regulären Navigation mit TAB-Taste nur in die markierten Felder springen kann.
Im Feld "Titel" würde eine Zeile reichen, darüber waren sich alle befragten Anwender einig. Hier ist es also nicht unbedingt erforderlich eine mehrzeilige TextArea zu haben.
Das Feld "Kommentar" wird so gut wie gar nicht genutzt.

- Kostenträger-Auswahl

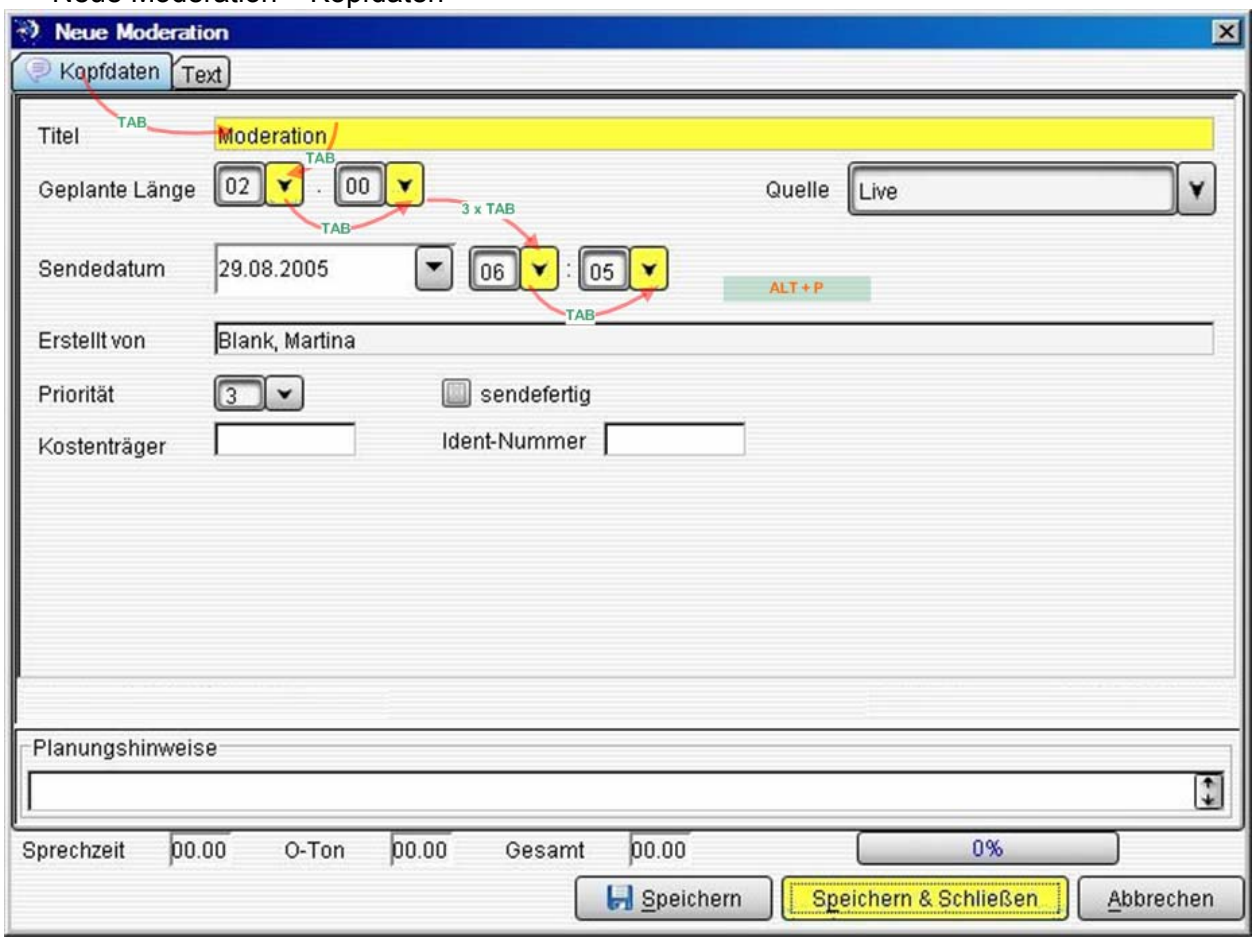


Hier wurde die mangelnde Bindung zwischen Kostenträger-Nummer und ihrer Bedeutung angesprochen. Bei der Auswahl (z.B. in der vorhergehenden Maske "Wortbeitrag") steht nur die Nummer. In der Auswahlmaske sieht man die Bedeutung, aber die Nummer nicht mehr.

Eine Anwenderin wies speziell darauf hin, dass wenn man z.B. die Eingaben eines anderen Kollegen prüfen will und dazu nochmals in die Auswahlmaske geht, ist hier nicht der Kostenträger markiert, der vorne eingetragen ist. Wählt man "Abbrechen" ist der vorher eingetragene Kostenträger vorne verschwunden. Hier sollte also dringend eine stärkere Bindung sowie eine "Merkfunktion" eingeführt werden.

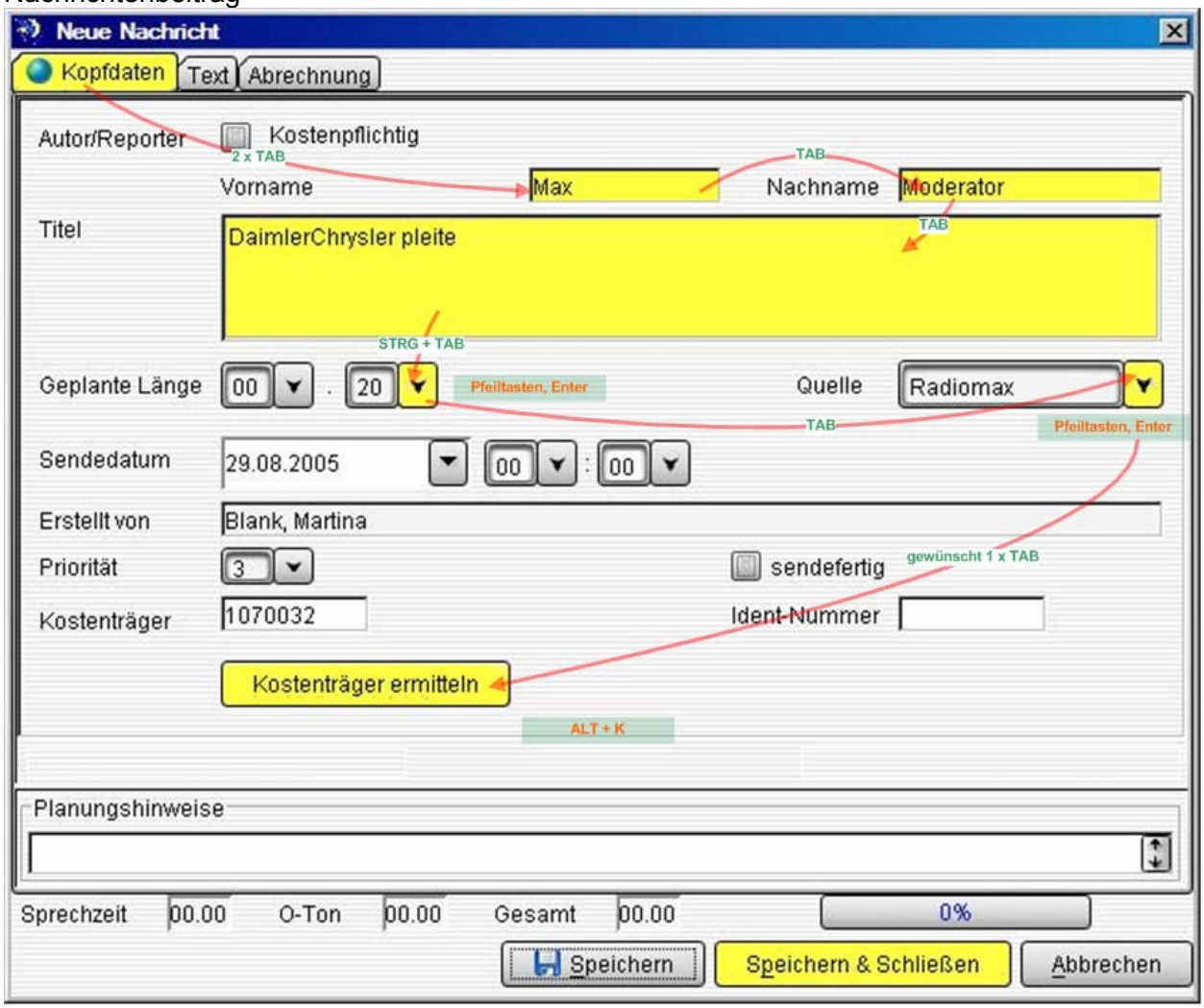
Die Anwenderin schlug weiterhin vor: Die Auswahlfunktion wäre mit der Tastatur deutlich leichter nutzbar, wenn man nach Anwahl eines Hauptpunktes links mit der rechten Pfeiltaste direkt in die Liste rechts gelangen könnte, wie es z.B. in jedem MS Office Menü bei Unterverzweigungen üblich ist.

- Neue Moderation – Kopfdaten



Diese Maske wird bei der Nachbearbeitung aufgerufen um Lücken im Sendeplan zu füllen.Auch hier werden nur wenige Felder der Maske genutzt, die aber alle leicht mit der Tastatur erreichbar sind.

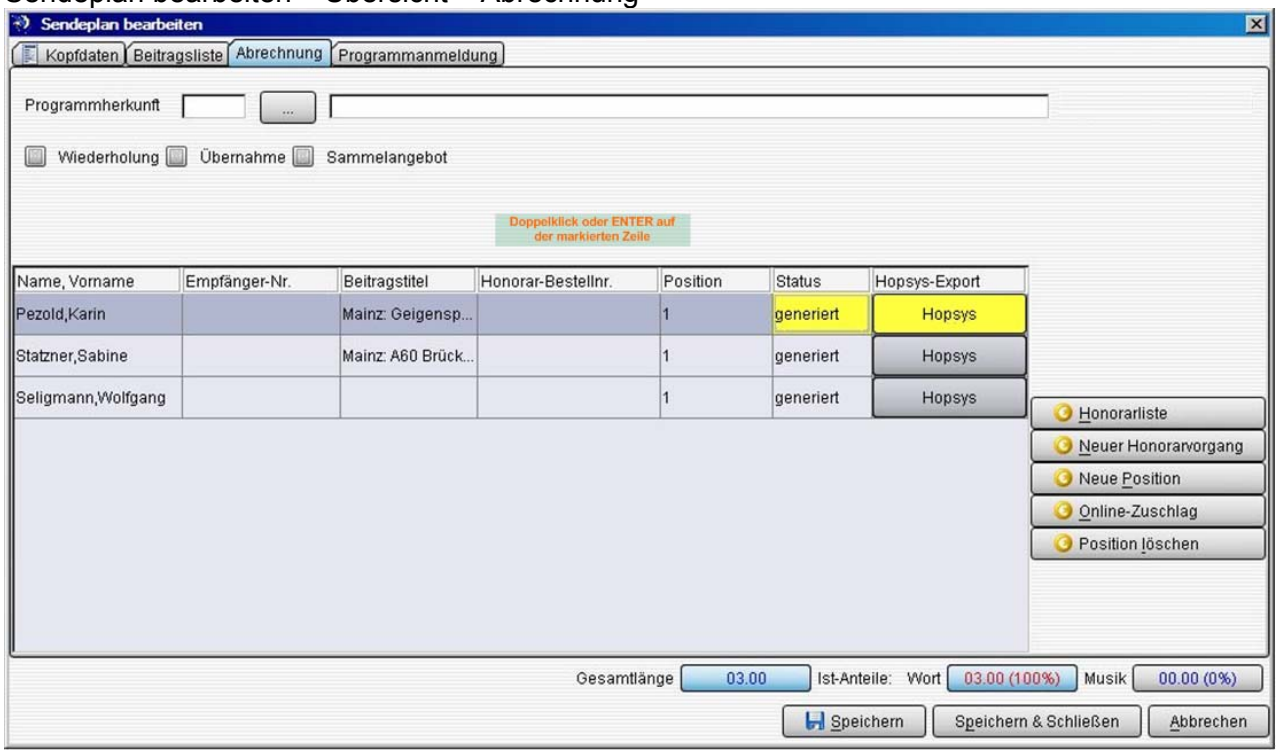
- Nachrichtenbeitrag



Auch in dieser Maske wurde wieder darauf hingewiesen, dass die Felder im mittleren Bereich nicht bearbeitet werden und daher im TAB-Focus übersprungen werden könnten.

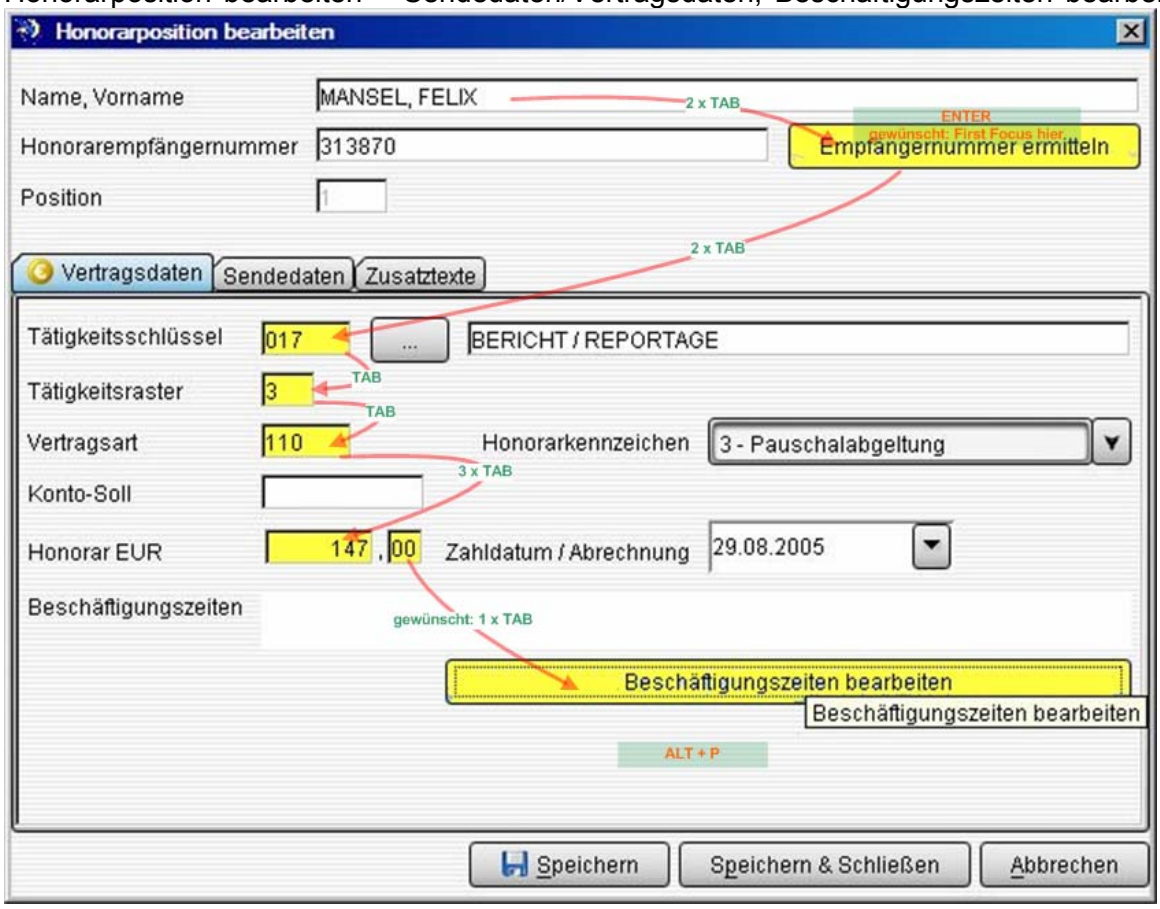
Auch hier wird im "Titel"-Feld nicht mehr als eine Zeile Platz benötigt.

- Sendeplan bearbeiten – Übersicht – Abrechnung



In dieser Maske wird hauptsächlich mit der Maus navigiert, da keine Tastatur-Eingaben zu erfolgen haben. Der Button "HOPSYS" überträgt nur die korrekten Daten, wenn zuvor nach Beendigung der Nachbearbeitung der Button "Speichern" betätigt wurde. Ist das so auch in der Hilfe dokumentiert?

- Honorarposition bearbeiten – Sendedaten/Vertragsdaten, Beschäftigungszeiten bearbeiten



Durch Doppelklick auf die jeweilige Honorarposition in der vorhergehenden Maske kommt man in diese hier. Zunächst erfoglt üblicherweise eine kurze Prüfung der Daten in der Lasche "Sendedaten", dann werden in der Lasche "Vertragsdaten" einige Zahlen erfasst. Auch hier sollten möglichst nur die markierten Felder im TAB-Focus sein.

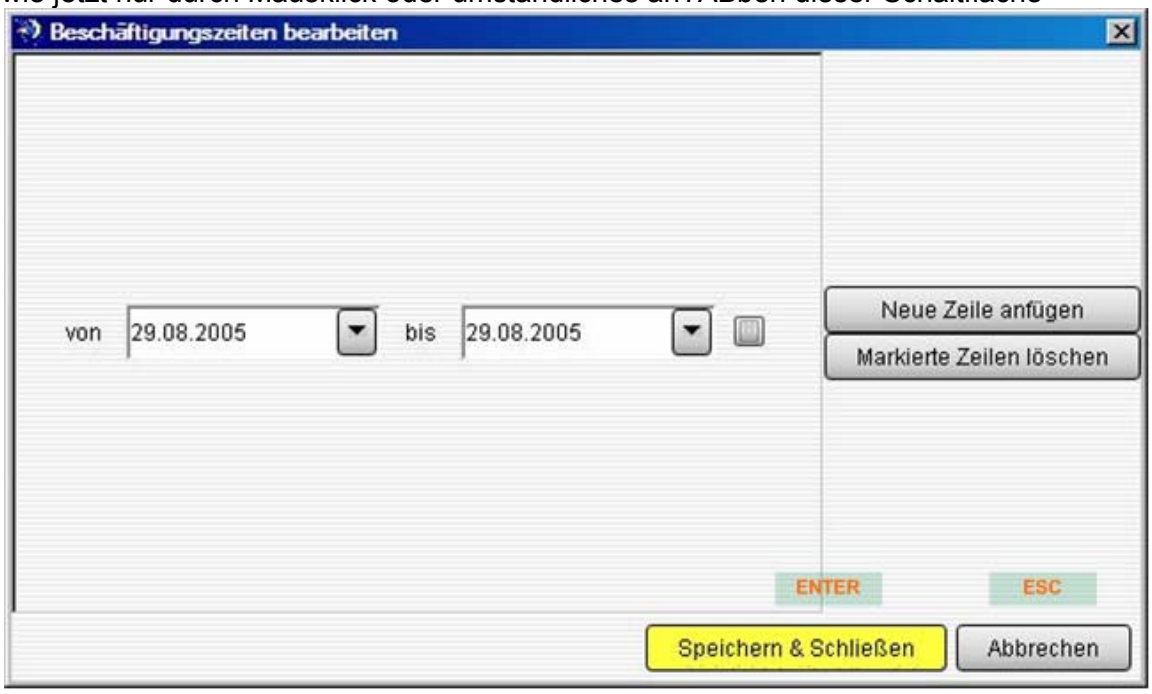
weitere Ideen und Anregungen der Anwender:

- In Dialogen wie diesen sollte ein Focus-Wechsel zwischen den Tasten auch mit den Pfeiltasten und nicht nur mit TAB erzielt werden können.

- Bei Abbrechen eines Vorgangs kommt in WebMerlin an keiner Stelle ein Hinweis, ob man sich wirklich sicher sei. Dies war ein Hinweis einer Anwenderin, die dadurch aber noch nie einen Datenverlust erlitten hat. Jedoch ist es in jeder Anwendung üblich, dem User zu sagen "Wenn du jetzt abbrichst, gehen alle deine gemachten Änderungen verloren. Bist du sicher, dass du abbrechen willst?"
- Über die Filterfunktion im Programmschema kann man von der Einzelansicht in die Wochenübersicht der Sendepläne wechseln. Eine Anwenderin äusserte den Wunsch, dass bereits bearbeitete Sendepläne hier eine gesonderte Markierung erhalten sollten.

weitere Ideen und Anregungen von mir:

- Überall sollte ESC zum "Abbrechen" oder "Nein" oder Dialog schliessen möglich sein statt wie jetzt nur durch Mausklick oder umständliches anTABben dieser Schaltfläche



- Speichern+Schliessen nicht nur über ALT+P sondern (wo möglich) über ENTER

- bei Sprung mit TAB in ein bereits ausgefülltes Textfeld sollte der eingegebene Text markiert sein (eine Anwenderin war übrigens unabhängig von mir zu der gleichen Ansicht gekommen)

<u>abschliessende Beurteilung (meine Empfehlungen an das Projekt-Team)</u>

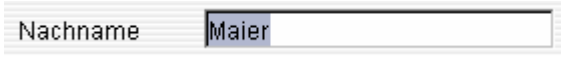
- spezieller (TAB-)Focus-Wechsel für ausgewählte Eingabemasken
- Umwandeln der TextAreas in TextFields oder Abstellen des TAB-Vorschubs in diesen Feldern
- ENTER für alle Speichern/Ja/OK-Aktionen
- ESC für alle Abbrechen/Nein-Aktionen
- Shortcuts für die Icons des Beitragsbaumes
- Überarbeitung der Koppelung Kostenträger-Kostenträgernummer
- Überarbeitung der Navigation in der Kostenträger-Auswahlmaske
- Eine Vereinfachung der bereits bestehenden Tastatur-Kürzel für Kopieren, Einfügen, Ausschneiden STRG+C/V/X statt STRG+ALT+C/V/X schien mir zu dem Zeitpunkt nicht erforderlich zu sein, da diese Funktionen im aktuellen Arbeitsfluss selten genutzt werden. Zumindest nicht von den befragten Personen. Einer der Chefentwickler meinte jedoch, diese Änderung sei unbedingt erforderlich.

Diese Liste hatte ich bei einem unserer üblichen Wochen-Meetings vorgestellt. Es verging einige Zeit bis letztendlich klar war, welche dieser Änderungen nun umgesetzt werden sollte. Also auf welche Änderungen sich alle Beteiligten einigen konnten. Dabei muss man wissen, dass WebMerlin eben nicht nur von diesen 20 Mitarbeitern in Mainz genutzt wird, sondern dass die Anwendung inzwischen auch in einigen anderen Rundfunkanstalten (RFAs) Deutschlands eingesetzt wird.

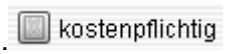
Die folgenden Änderungsvorschläge standen daher zunächst noch zur Diskussion und sollten noch mit den anderen RFAs abgestimmt werden:

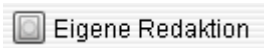
- Abstellen des TAB-Vorschubs in den Feldern Titel/Kommentar u.ä. (alle TextAreas)

  und Ersetzen durch STRG+TAB (Verlassen des Feldes durch TAB, Tabulator im Feld setzen durch STRG+TAB).
  Anmerkung einer Anwenderin aus dem Projekt-Team: In dem Tab "Programmanmeldung" des Sendeplanes wird der Tabulator im Freitextfeld sehr viel benötigt und sollte hier nicht durch STRG+TAB ersetzt werden.
- Markieren des Inhalts eines Eingabefeldes bei erneutem Ansteuern (statt Cursor ans Feldende)



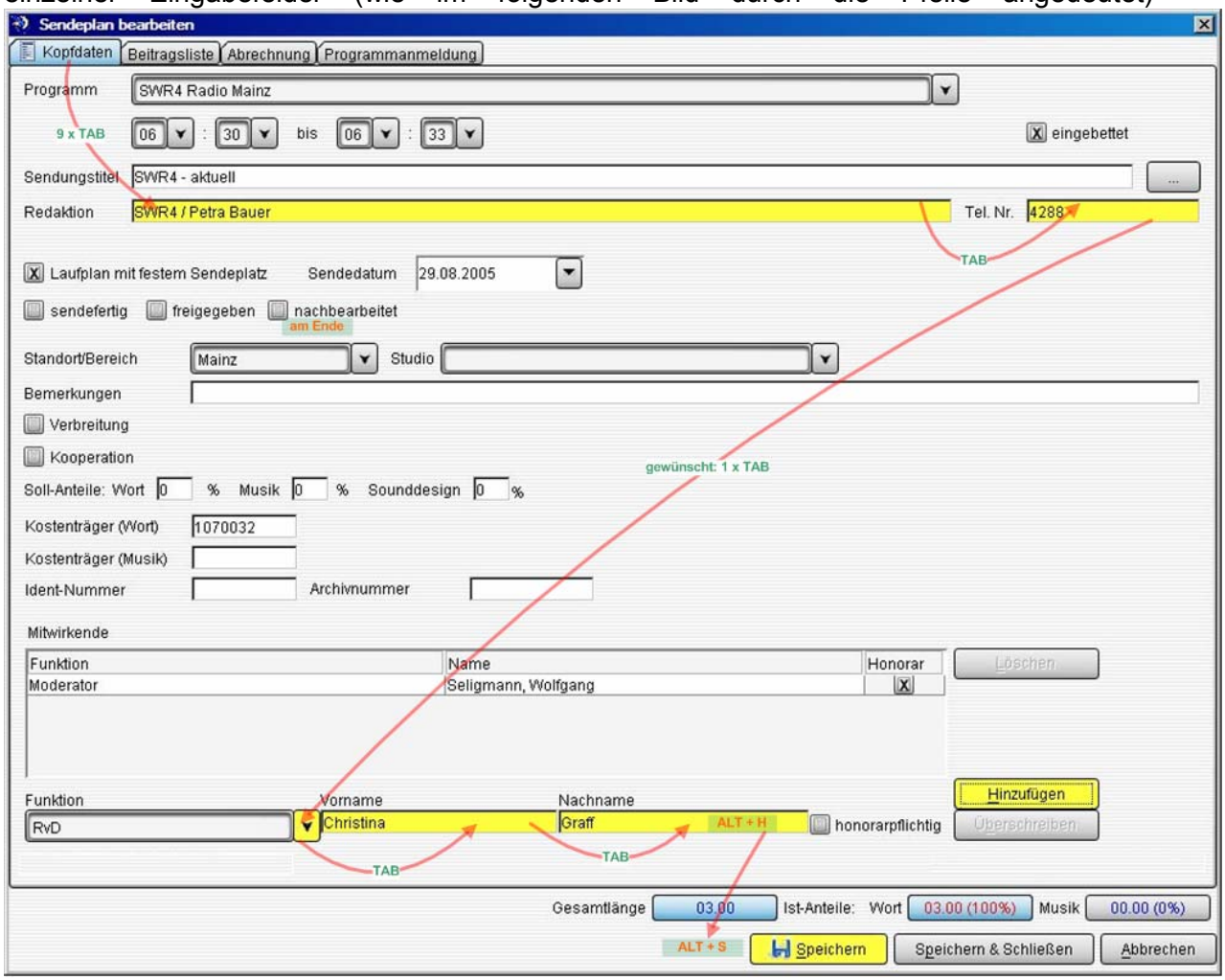
- Focus-Wechsel zwischen Schaltflächen (einfache JA/NEIN-Dialoge) auch mit Pfeiltasten statt nur mit TAB

- sichtbare Markierung (dunklere oder gestrichelte Umrandung) von Radio-Buttons, wenn sie den Focus erhalten aber nicht aktiviert (angekreuzt/ausgewählt) sind Bsp. 

oder 

- Einführung einer "Wirklich abbrechen? Ihre Änderungen gehen verloren!"-Meldung bei allen Abbrechen-Aktionen, vor denen etwas am Datenbestand geändert wurde

- spezieller (TAB-)Focus-Wechsel für ausgewählte Eingabemasken, also Deaktivierung einzelner Eingabefelder (wie im folgenden Bild durch die Pfeile angedeutet)



- Anmerkung einer Anwenderin aus dem Projekt-Team: Bei den Zeiteingabefeldern (Beitragslänge, Uhrzeiten) sollte ein einziges Feld mit hinterlegter Formatierung eingerichtet werden, um dem Nutzer an der Stelle nicht aufzuhalten, weil er über die Schreibweise nachdenken und auch in verschiedene Felder springen muss.

Im Gespräch mit den Entwicklern hatte ich noch die folgenden Punkte notiert:

- Die WebMerlin Zwischenablage wird nach einmal verwenden geleert, man kann einen dort enthaltenen Inhalt nur einmal an einer anderen Stelle einfügen.
- Momentan ist WebMerlin aus modalen Dialogen (wenn mehrere Fenster geöffnet sind, kann nur im vordersten gearbeitet werden) zusammengesetzt. Ein Entwickler meinte, dies solle geändert werden, erfordere jedoch eine komplette Überarbeitung der Client-Komponente.
- In manchen Masken verschwindet der Focus nach ein paar TAB-Sprüngen einfach ins Nichts.

5. Umsetzung der Änderungen

Nach Rücksprache mit der Projektgruppe WebMerlin und Abklärung der vorgeschlagenen Änderungen mit den anderen RFAs stand ich vor der folgenden Liste der umzusetzenden Änderungen, die ich hier in der Form aufführe, wie sie mir vorlag, nämlich als Einträge in die webbasierte Fehlerdatenbank JIRA.

Für Software Projekte ist eine Fehlerdatenbank ein unverzichtbares Werkzeug. Wer Software im Team entwickelt, braucht eine Fehlerdatenbank. Ein ganz hervorragendes Produkt für diese Aufgabe ist der JIRA Issue Tracker.
seine Vorteile:

- webbasiert, d.h. keine Installation irgendwelcher Software auf dem Arbeitsplatz-PC. Es ist lediglich ein Webbrowser erforderlich. Auch der Zugriff via Internet ist möglich.
- deutsche und englische Benutzeroberfläche verfügbar
- auch ohne längere Einweisung leicht zu bedienen: JIRA ist übersichtlich gestaltet und sehr benutzerfreundlich und intuitiv
- leicht an die eigenen Anforderungen anpassbar
- kommerzielles Produkt: Support, Updates, neue Releases

Projekte : WebMerlin (ID: WEBMERLIN)

Projektleitung: Miriam Brielmann
URL: http://sade.swr.cn.ard.de/rms/webstart

- Neuen Vorgang in Projekt WebMerlin anlegen
- Projekt administrieren
- Release-Notizen

Wählen: Offene Vorgänge	Versionsplanung	Änderungsprotokoll	Populäre Vorgänge
Anbindung Sendeautomation (ext.)	9	WM 3.3.5	14
Anwendung	131	WM 4.0	70
Betrieb	8	WM 4.1	44
Drucken(B2W)	3	keine Version zugeordnet	128
Druckvorlagen und Reports	12		
Fachliche Erweiterungen	32		
Honorierung	10		
Online-Hilfe	2		
Quality Assurance (QA)	1		
Selaus-Schnittstelle	3		
Terminverwaltung	9		
keine Komponente zugeordnet	38		

In JIRA kann man (sofern der Datenbestand gepflegt wird!) jederzeit nachschauen, was für Issues offen sind und den Bearbeitungsstand und zuständigen Entwickler abfragen. Änderungen des Datenbestandes werden per Mail allen Beteiligten mitgeteilt.

Im Nu hatte ich mehrere Issues, die mir in JIRA zugewiesen worden waren und während der Projekt-Laufzeit kamen auch immer mal wieder welche dazu, da ich ab sofort die Zuständige für alle GUI-Issues war.

Auswertungen
Aufwände pro Projekt
Entwicklerauslastungsbericht
gruppierter Bericht
Aufwände pro Version

Vordefinierte Filter

- ALLE
- unerledigt
- nicht zugeordnet
- eigene
- selbst erstellt
- vor kurzem erledigt
- vor kurzem hinzugefügt
- vor kurzem aktualisiert
- am wichtigsten

Projektzusammenfassung

open	37	2%
in progress	4	
reopened	3	
resolved	76	4%
closed	1387	80%
to approve	20	1%
pending	78	5%
to handle	60	3%
in test	39	2%
not approved	10	1%
to specify	16	1%

Offene Vorgänge

Nach Priorität

Blocker	2	1%
Critical	21	8%
Major	159	62%
Minor	61	24%
Trivial	12	5%

Nach Bearbeiter

[redacted]	1	
[redacted]	1	
[redacted]	1	
[redacted]	89	35%
[redacted]	2	1%
[redacted]	2	1%
[redacted]	4	2%
[redacted]	9	4%
[redacted]	1	
[redacted]	1	
[redacted]	3	1%
[redacted]	25	10%
[redacted]	2	1%
[redacted]	9	4%
Martina Blank	5	2%
[redacted]	34	13%
[redacted]	15	6%
[redacted]	1	
[redacted]	1	
[redacted]	1	
[redacted]	12	5%
[redacted]	5	2%
[redacted]	7	3%
[redacted]	24	9%

Meine JIRA-Issues

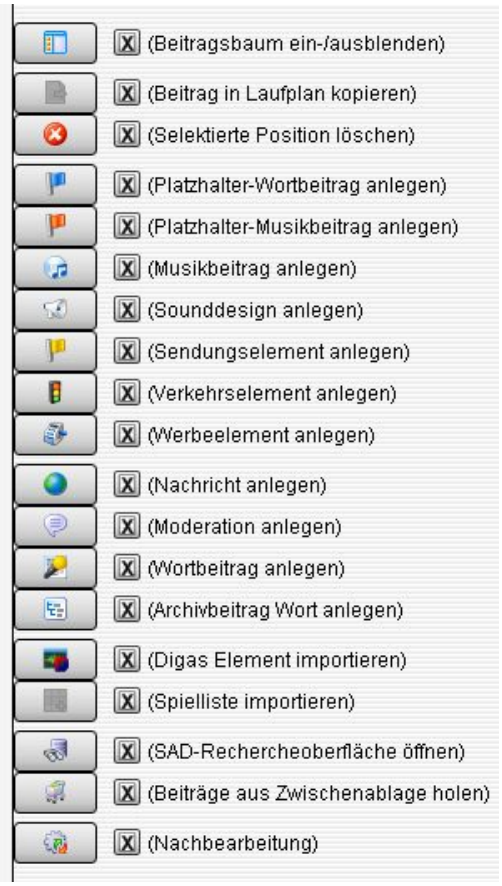
JIRA [#WEBMERLIN-1564] Überarbeitung Shortcuts und Texteingabefelder

Vorbemerkung:
Dies war der umfangsreichste JIRA-Eintrag, den ich bearbeitet habe, da hier viele Wunsch-Änderungen gesammelt wurden. Etwas was man nicht unbedingt machen sollte. Eigentlich sollte jede Änderung in einen separaten Eintrag gestellt werden. Ich werde sie hier zunächst auflisten und dann der Reihe nach erklären, wie ich vorgegangen bin. Kernpunkt dieses Eintrages war jedoch die Aussage "WebMerlin sollte nach Möglichkeit auch ohne Maus bedienbar sein".

Liste der gewünschten Änderungen:
(1) Shortcuts für Icons des Beitragsbaumes
(2) Shortcut für den Button "Kostenträger ermitteln", in allen Masken, in denen dieser auftaucht
(3) Shortcuts in Sendeplanliste für Buttons "Vor", "Zurück", "Bearbeiten", "Schliessen"
(4) Vereinfachung der Shortcuts im Menü: STRG+C/V/X statt STRG+ALT+C/V/X
(5) ENTER für alle Speichern/Ja/OK-Aktionen
(6) ESC für alle Abbrechen/Nein-Aktionen (ohne Hinweis, also als ob man Abbrechen drückt)

Einzelbetrachtung:
(1) Shortcuts für Icons des Beitragsbaumes

Ausgangssituation:
Die hier gemeinten Icons (rechts im Bild) konnten via Tastatur bisher nur durch vielfaches Drücken der Tabulator-Taste angesteuert und dann mit ENTER ausgelöst werden. Das Anklicken mit der Maus war an dieser Stelle also deutlich zeitsparender. Die User-Gruppe, die aber lieber die Tastatur benutzt, war hier benachteiligt.



Lösung:
Nach Rücksprache mit den Usern habe ich jeden der Buttons mit einer möglichst naheliegenden Tastenkombination von ALT und einem Zeichen hinterlegt. Gleichzeitig erscheinen die jeweiligen Shortcuts hinter dem Button-Namen (die Beschreibung kann ein- und ausgeblendet werden) und im Tooltip, wenn man mit der Maus über den Button geht.

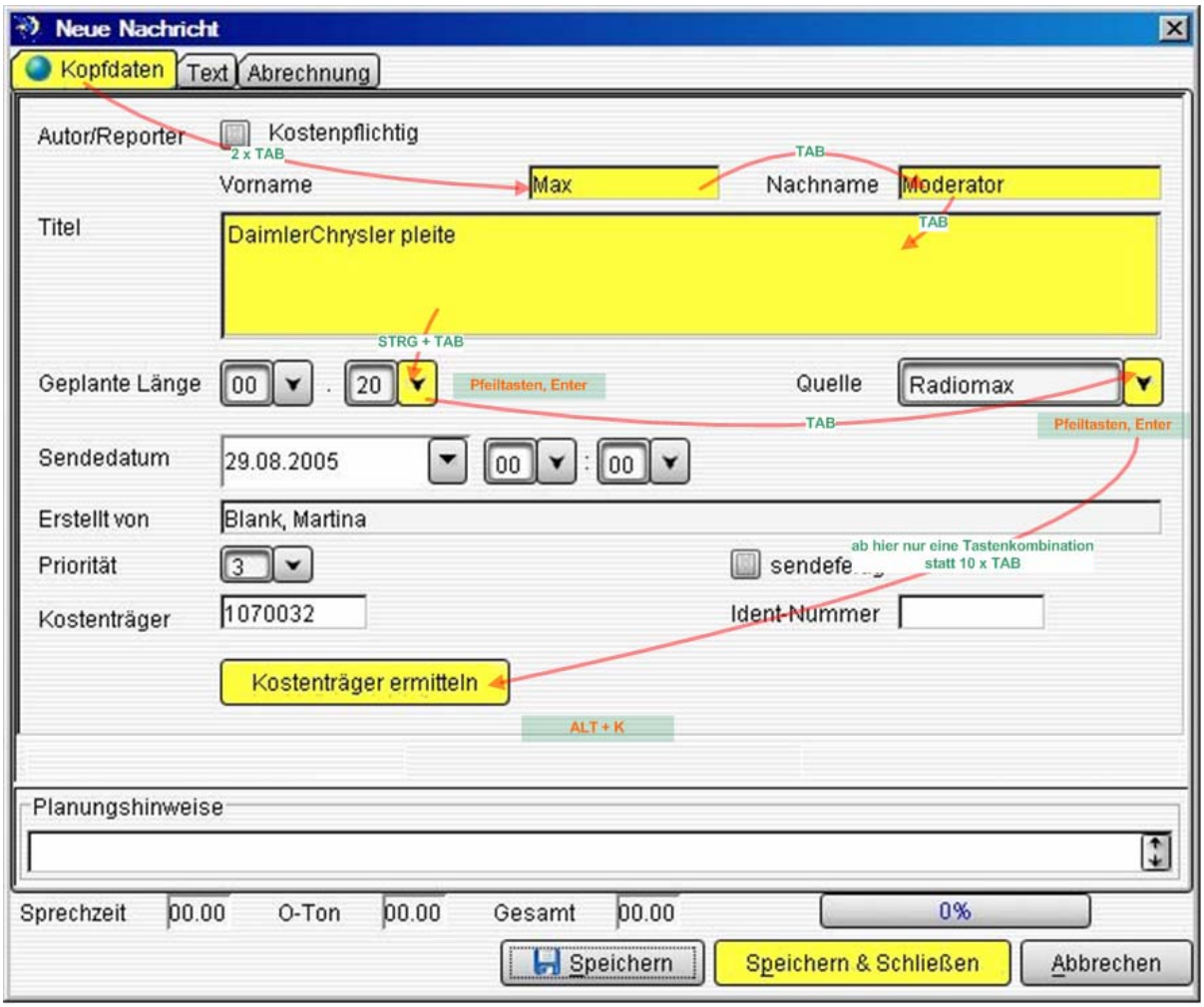
Bemerkungen:
Die User wurden auf mehreren Wegen über diese Änderung informiert. Einige haben darauf schon sehnsüchtig gewartet, wie auch ein anderer JIRA-Eintrag eines Kollegen aus dem Projekt-Team zeigte. Der Eintrag wurde dann mit diesem zusammengeführt.

Mit der Auswahl von ALT+[Taste] bin ich der Konvention von WebMerlin gefolgt bei Buttons dieser Art stets ALT als Kombitaste zu verwenden. Warum soll man etwas komplett umkrempeln, woran sich die User schon gewöhnt haben?

beteiligte Klassen:
de.ard.sad.rms.client.laufplan.PlanView.java

(2) Shortcut für den Button "Kostenträger ermitteln" (in allen Masken, in denen dieser auftaucht)

Ausgangssituation:
In diesem Screenshot sieht man eine der betroffenen Masken (Neue Nachricht) mit dem Kostenträger-Button. Die roten Pfeile stellen den bisherigen "Focus"-Wechsel bzw. die Navigation durch die Maske dar, wie sie üblicherweise bei den Usern erfolgt.



Problem:
Tastatur-User sind hier benachteiligt, da sie bis zu 10 mal die Tabulator-Taste betätigen müssen, bis sie am Kostenträger-Button angelangt sind.

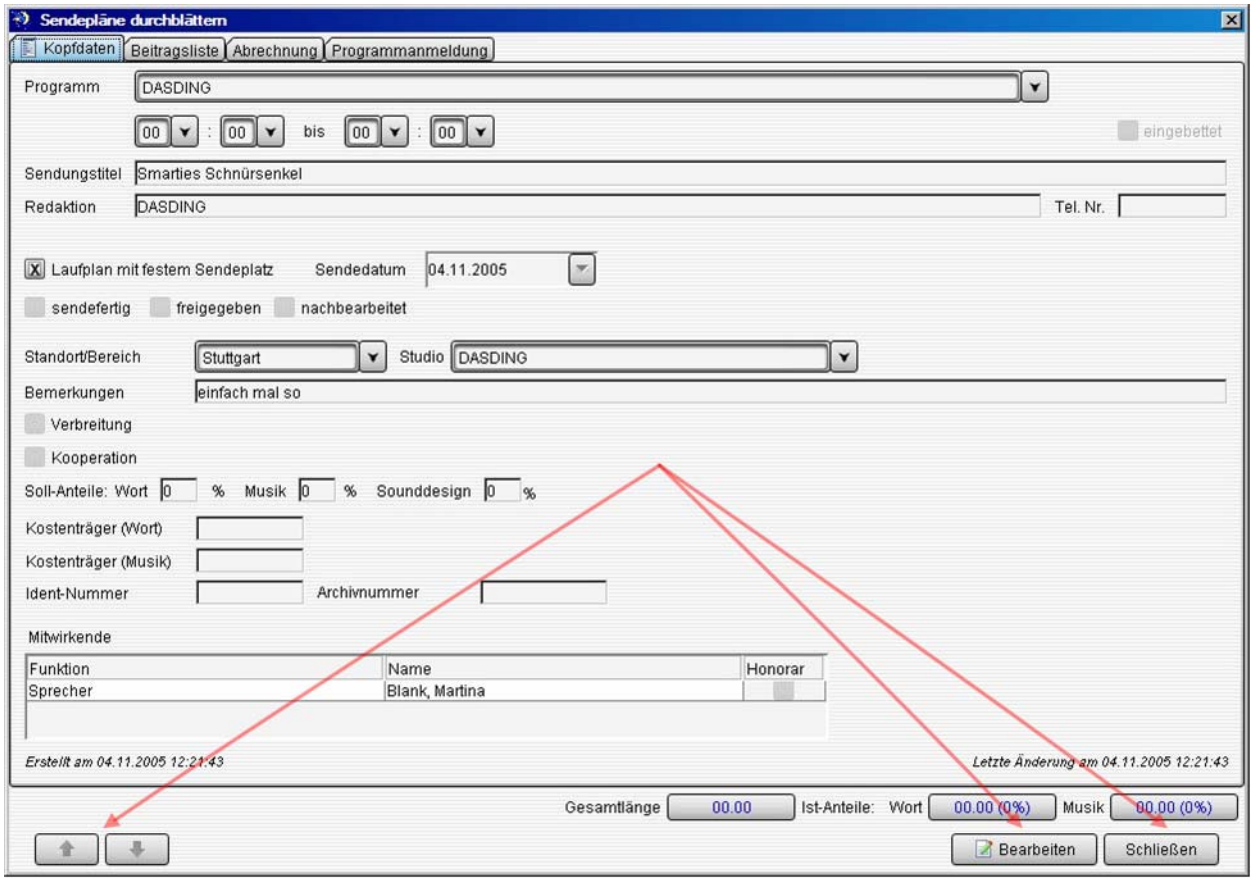
Lösung:
Der Button wurde mit einem Mnemonic-Shortcut hinterlegt: ALT+K (K wie Kostenträger, relativ naheliegend!). Das K ist unterstrichen, so dass die Funktion ins Auge fällt.

Bemerkungen:
Hier wurde - wenn auch nicht direkt, so doch in einer Annäherung - eine Anregung aus der Ergonomie-Studie umgesetzt. Der für Tastatur-User lange weg vom letzten Eingabefeld (Quelle) zum Button "Kostenträger ermitteln" kann nun durch eine Tastenkombination überbrückt werden.

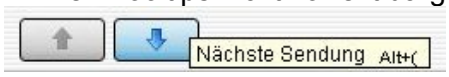
beteiligte Klassen:
de.ard.sad.rms.client.beitrag.NachrichtDetailHeadView.java
de.ard.sad.rms.client.beitrag.WortbeitragDetailHeadView.java

(3) Shortcuts in Sendeplanliste für Buttons "Vor", "Zurück", "Bearbeiten", "Schliessen"

Ausgangssituation:
Es geht um die hier im Bild mit roten Pfeilen gezeigten Buttons. Die Maske dient dazu, durch eine Liste von Sendeplänen durchzublättern, sie ggf. zu bearbeiten und die Liste wieder zu schliessen.

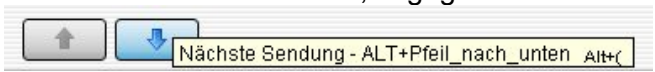


Problem:
Für die Buttons rechts gibt es gar keine Shortcuts. Die Buttons links haben Shortcuts, die aber in ihren Tooltips nicht korrekt dargestellt werden.



Lösung:
Die Buttons "Bearbeiten" und Schliessen" habe ich mit Mnemonics hinterlegt: ALT+B und ALT+S. Die jeweiligen Buchstaben sind nun unterstrichen, sodass die Funktion ins Auge fällt.

Für die Navigations-Buttons sollten die Pfeiltasten (↑ und ↓) zusammen mit ALT verwendet werden können. Dies funktionierte auch bereits. Die leicht abgesetzte kleinere Beschriftung im Tooltip, die den Shortcut beschreibt, wird von Java direkt eingespielt, wenn man dem Button einen Mnemonic hinzufügt. Dabei wurde wohl die Beschriftung falsch umgesetzt, da es sich nicht um Tasten aus dem Standard-Zeichensatz handelt. Ich habe keine Möglichkeit gefunden, dies zu unterdrücken oder zu berichtigen. Die Kollegen wussten sich leider auch keinen Rat. Daher habe ich über ein Properties-File nochmals eine zusätzliche Beschriftung hinzugefügt. Ein unschöner Workaround, zugegeben.



Bemerkungen:
Herauszufinden, wo die Mnemonics für die Buttons einzusetzen waren, war in dieser Maske etwas aufwendig, da sie verschachtelt aufgebaut ist. Die zwei Bearbeiten/Schliessen-Buttons unten rechts liegen im äussersten Rahmen, in den die Sendepläne incl. der Vor- und Zurück-Buttons eingebettet sind. Mehr dazu im Abschnitt Dialoge (5)+(6).

beteiligte Klassen:
de.ard.sad.rms.client.ViewDlg.java
lib.laufplan.properties

(4) Vereinfachung der Shortcuts im Menü: STRG+C/V/X statt STRG+ALT+C/V/X

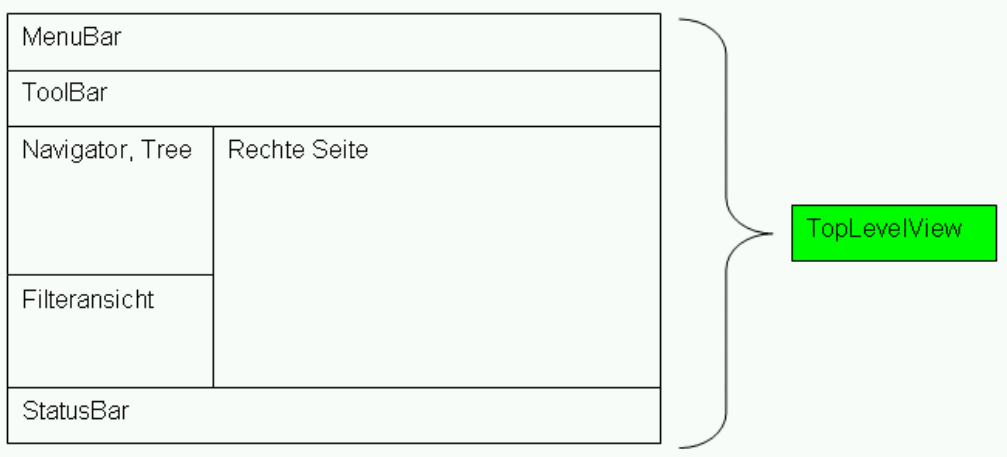
Ausgangssituation:
Die Shortcuts bzw. Menüeinträge dienen vor allem dazu, Einträge in der Beitragsliste oder im Laufplan zu kopieren und einzufügen. Dies erleichtert den Anwendern die Arbeit, da sie so bereits vorhandene Beiträge etc. an anderer Stelle wiederverwenden können. Wie bereits im Kapitel 3 - Anwendung (Seite 10) erwähnt, wurden hier nicht die üblichen Windows-Shortcuts für Cut, Copy und Paste gewählt, um sie von eben diesen zu unterscheiden.

Problem:
Wie sich bei meiner Suche nach einer Änderungsmöglichkeit herausstellte, war dies wohl nicht der einzige Grund. Aufgrund der geschichteten Architektur der Swing-Panes war es so, dass nicht alle Key Events (Tastatur-Anschläge) korrekt abgefangen werden konnten, also letztendlich wie gewünscht eine bestimmte Action auslösen konnten.

Lösung:
Ich habe die hinterlegten Shortcuts abgeändert, jedoch löste jede erdenkliche Tastaturkombination die gewünschte Action aus, ausser STRG+C/V/X. Der Menüeintrag "Ausschneiden" heisst in der Anwendung `jmiCut` und ist als JMenuItem angelegt. Die Tastenkombination ist im Properties-File menu.properties mit `AcceleratorCut=control alt X` hinterlegt. Dem Menüeintrag ist der Shortcut folgendermassen zugeordnet:

```
jmiCut.setAccelerator(KeyStroke.getKeyStroke(ResourceBundle.getBundle("lib/menus").get
String("AcceleratorCut")));
```

Der Ansatz, diesen Eintrag auf `control X` zu ändern blieb leider erfolglos. Meine Kollegen und ich nehmen an, dass die gängigen Windows-Shortcuts STRG+C/X/V innerhalb der verschachtelten Struktur der Anwendung irgendwo auf einer vorherigen Ebene abgefangen werden, bevor sie dazu kommen die von uns gewünschte Action auszulösen. Hier nochmals die geschachtelte Struktur der GUI (siehe auch Seite 7). Unter "Navigator, Tree" liegen die Beitragsliste und der Laufplan, in denen die Actions z.B. zur Wirkung kommen sollten. In der MenuBar sind die JMenuItems hinterlegt.



Im Endeffekt habe ich die alten Shortcuts beibehalten. Ich hatte noch in einem Swing-Buch die Idee aufgeschnappt, die FocusTraversalPolicy der Anwendung zu ändern (`setFocusTraversalKeysEnabled(false)`) und damit zu verhindern, dass KeyEvents an ungewünschter Stelle "consumed" werden. Dies würde bedeuten, dass die gesamte

FocusTraversalPolicy manuell zu Behandeln wäre. Das auszuprobieren habe ich aber zeitlich leider nicht mehr geschafft. Ich werde es im März versuchen, wenn ich wieder beim SWR weiterarbeite.

beteiligte Klassen:
de.ard.sad.rms.client.MenuView.java
lib.menu.properties
de.ard.sad.rms.client.actions.CutAction.java
de.ard.sad.rms.client.actions.CopyAction.java
de.ard.sad.rms.client.actions.PasteAction.java
de.ard.sad.rms.client.laufplan.LaufplanTreeView.java
de.ard.sad.rms.client.beitrag.BeitragTreeView.java
u.a.

(5) ENTER für alle Speichern/Ja/OK-Aktionen
(6) ESC für alle Abbrechen/Nein-Aktionen (ohne Hinweis, also als ob man Abbrechen drückt)

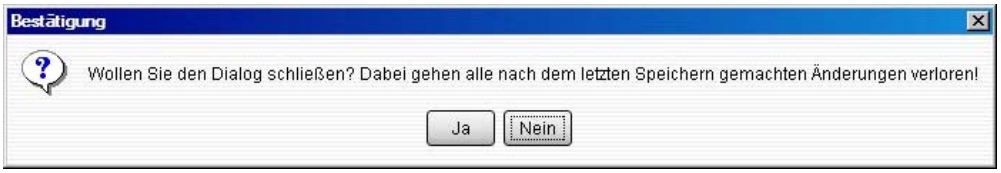
Ausgangssituation:
Wer aus der Windows-Welt kommt und eher die Tastatur als die Maus benutzt, ist es gewohnt, dass er diese kleinen Rückfragen des Rechners (à la "Sind Sie wirklich sicher, dass Sie das tun wollen?") mit einmal ENTER oder ESC schliessen kann. Die von mir befragten User sahen das ähnlich.

Problem:
Da die meisten Dialoge in WebMerlin aber nicht auf ENTER oder ESC reagieren, müssen sie dann zur Maus greifen. Daher sollte ich diese überarbeiten.

Lösung:
Meine Analyse der Client-Komponenten ergab, das es im Grossen und ganzen drei Hauptgruppen von Dialogen gibt. Diese stelle ich jetzt nacheinander vor und beschreibe wo sie vorkommen und was ich an ihnen geändert habe.

(1) einfache Dialoge
   a. simple SWING JOptionPane

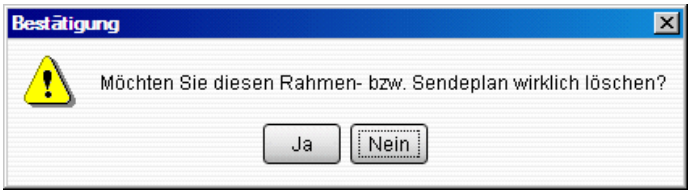
verwendete Klasse	`javax.swing.JOptionPane.showConfirmDialog()`
Funktionsweise	• Fragen/Texte werden als Parameter beim Aufruf an diese Klasse übergeben • Anwender hat JA/NEIN-Buttons zur Auswahl • dient zur Sicherheitsabfrage
Einsatz	1. de.ard.sad.rms.client.BearbeitenDlg.java (close_dialog) 2. de.ard.sad.rms.client.laufplan.actions.GenerateAction (overwrite_question) 3. de.ard.sad.rms.client.prganmeldung.PeriodPanel (file_overwrite)
Ausgangs-situation	NEIN-Button hat den Default-Focus, reagiert auf ENTER Wechsel zu JA-Button über TAB
meine Änderung	Focus wurde bei 1 und 3 auf JA umgesetzt, indem der Parameter `initialValue` von NO auf YES geändert wurde. Der Dialog kann jetzt mit ENTER bestätigt werden. Umsetzung in dieser Form bei 2 nicht möglich, da andere Methode zur Erzeugung des Dialogs verwendet wird.
weitere Ideen	Klären, ob hier ein Abändern der aufrufenden Methode bei 2 gewünscht ist.

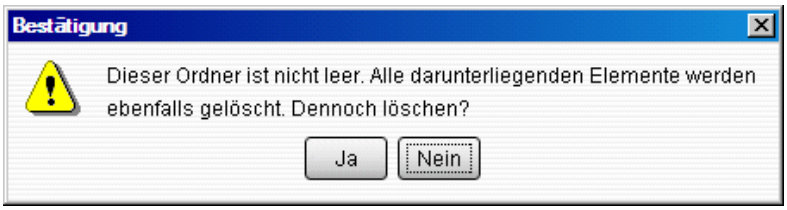


de.ard.sad.rms.client.BearbeitenDlg.java (close_dialog)

b. angepasste SWING JOptionPane

verwendete Klasse	de.ard.sad.rms.utilities.CustomizedJOptionPane.showOptionDialog()
Funktionsweise	• Klasse erbt von den SWING-Dialogen • Fragen/Texte werden beim Aufruf als Parameter an die Klasse übergeben • Anwender hat JA/NEIN-Buttons zur Auswahl • dient zur Sicherheitsabfrage bei Löschen-Aktionen
Einsatz	1. de.ard.sad.rms.client.actions.DeleteNodeAction (delete, delete_folder, delete_laufplan = 3 verschiedene Dialoge) 2. de.ard.sad.rms.client.programmschema.actions.PSDeleteAction (delete) 3. de.ard.sad.rms.client.actions.DeleteBeitragAction (delete)
Ausgangs-situation	NEIN-Button hat den Default-Focus, reagiert auf ENTER Wechsel zu JA-Button über TAB
meine Änderung	Focus bei allen auf JA umgesetzt, indem der Parameter `initialValue` von NO auf YES geändert wurde





1. de.ard.sad.rms.client.actions.DeleteNodeAction (delete, delete_folder, delete_laufplan)

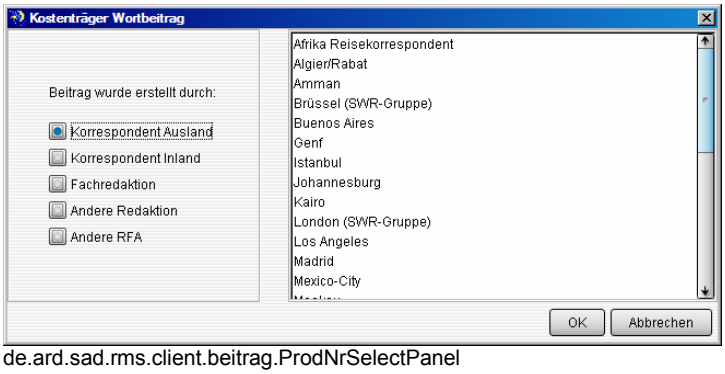
2. de.ard.sad.rms.client.programmschema.actions.PSDeleteAction (delete)
3. de.ard.sad.rms.client.actions.DeleteBeitragAction (delete)

(2) SWR-eigene Dialog-Klasse

verwendete Klasse	de.ard.sad.rms.utilities.GeneralDialog
Funktionsweise	• Fragen/Texte, Beschriftung und Anzahl der Buttons werden als Parameter an die Klasse übergeben • teilweise werden weitere Panels, etc. in den Dialog eingebettet • ca. 50 andere Klassen verwenden diese Klasse und rufen hierüber ihre speziellen Dialoge auf
Einsatz	nur ein Auszug: Kostenträger-Auswahlmaske, Beschäftigungszeiten bearbeiten, Programmanmeldung anzeigen/sperren, Sendepläne durchblättern, Digas-Element importieren, … werden alle darin eingebettet
Ausgangs-situation	kein spezieller Focus

meine Änderung	Die Anzahl und Beschriftung der Buttons variiert, eine einheitliche Festlegung welcher Button den Default-Focus haben soll oder als ENTER-Button festzulegen ist, ist nicht einfach.
	Dennoch habe ich ENTER für den jeweiligen jbtButton1 eingeführt.
	Aus zeitlichen Gründen konnte ich nicht mehr alle 50 Dialoge abprüfen, ob dadurch evtl. ein bereits vorhandener Focus für ENTER verloren geht.
	
	Eine Integration von ESCAPE habe ich nicht erreichen können. Der KeyEvent VK_ESCAPE geht scheinbar verloren. Folgende Varianten habe ich versucht:
	
	`this.jbtCancel.setMnemonic(KeyEvent.VK_ESCAPE);`
	
	`if (KeyEvent.getKeyCode() == KeyEvent.VK_ESCAPE) this.dispose()`
	
	`this.jbtCancel.addKeyListener(new KeyAdapter() {`
	`  public void keyPressed(KeyEvent evt) {`
	`  KeyStroke ks = KeyStroke.getKeyStrokeForEvent(evt);`
	`  KeyStroke EscapeStroke = KeyStroke.getKeyStroke(27, 0);`
	`  if (ks.getKeyCode() == KeyEvent.VK_ESCAPE) { dispose(); }`
	`}});`

weitere Ideen	Alternativ sollte der Einsatz von KeyBindings geprüft werden. Mithilfe von InputMaps kann man Tasten und Aktionen aneinander binden. Ein weiterer Versuch wäre, die FocusTraversalPolicy zu ändern.
Anmerkungen	Ähnlich war es bei de.ard.sad.rms.utilities.GeneralFrame. Hier habe ich für den "Schliessen"-Button ein Mnemonic (ALT+S) angelegt. Anwendung z.B. in der Terminverwaltung.
Links	http://java.sun.com/docs/books/tutorial/uiswing/misc/keybinding.html http://java.sun.com/docs/books/tutorial/uiswing/events/keylistener.html http://java.sun.com/docs/books/tutorial/uiswing/events/example-1dot4/KeyEventDemo.java http://java.sun.com/docs/books/tutorial/uiswing/misc/focus.html



de.ard.sad.rms.client.beitrag.ProdNrSelectPanel

de.ard.sad.rms.client.actions.EpgExportAction

de.ard.sad.rms.client.beitrag.Terminverwaltung.TerminView.java

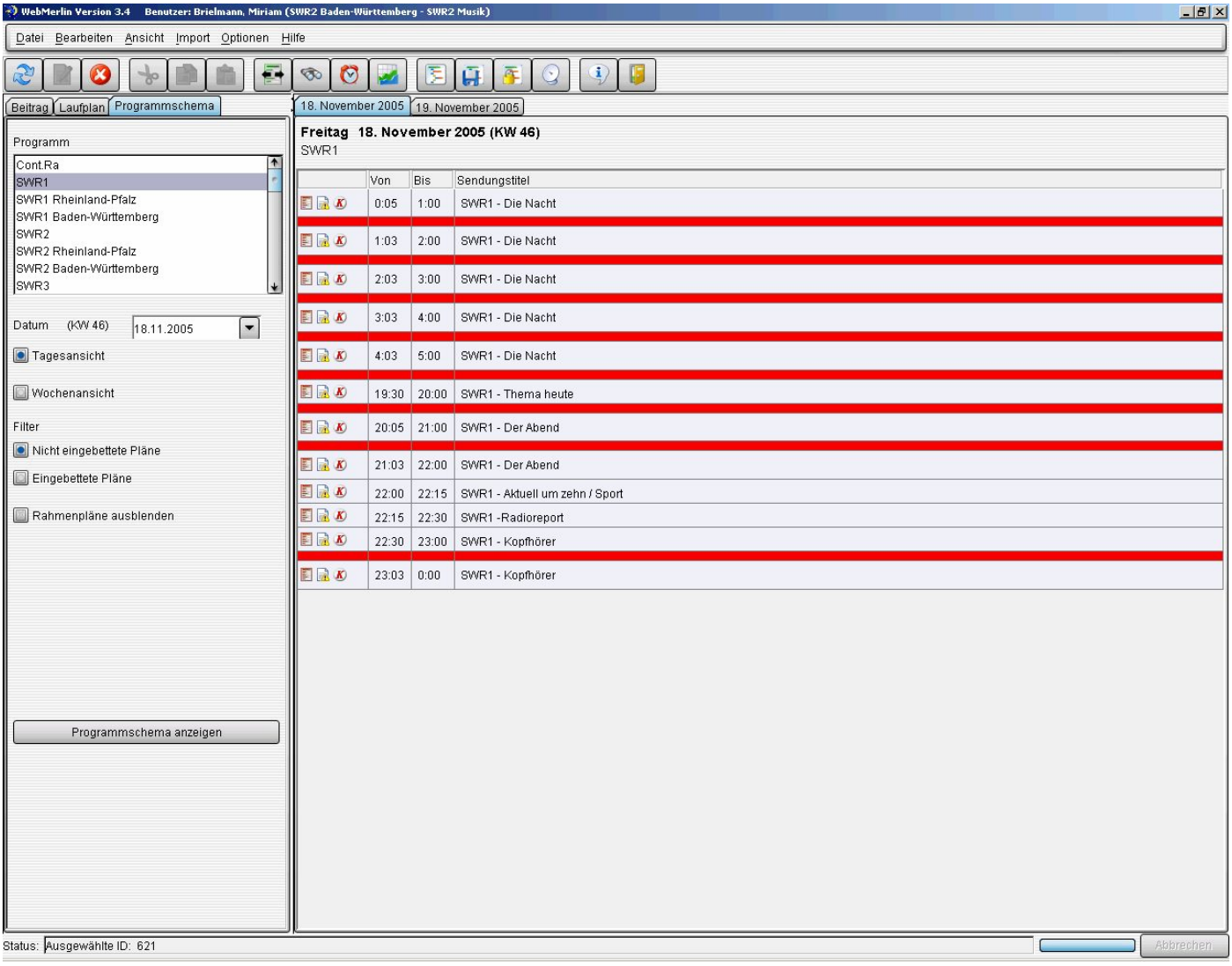
(3) sonstige Dialoge

verwendete Klasse	keine spezielle	
Funktionsweise	Die jeweiligen Fenster enthalten Buttons zum Speichern und Abbrechen oder andere Funktionen.	
Einsatz	de.ard.sad.rms.client.DataDeletedDlg	(Kopie anlegen, Abbrechen)
	de.ard.sad.rms.client.InsertOTonDlg	(Abbrechen)
	de.ard.sad.rms.client.TimeStampConflictDlg	(Abbrechen)
	de.ard.sad.rms.client.BearbeitenDlg	(Abbrechen)
	de.ard.sad.rms.client.ViewDlg	(Bearbeiten, Schliessen)
	de.ard.sad.rms.client.pgranmeldung.PrgAScrollDlg	(Bearbeiten, Schliessen)
Ausgangs-situation	keine Shortcuts	
aktueller Stand	Mnemonics für die meisten Dialoge eingeführt, siehe unterstrichene Buchstaben oben	
weitere Ideen	de.ard.sad.rms.utilities.ExceptionMessageDlg (Details, Schliessen) Hier wäre die Umsetzung mit ENTER für Details wohl sinnvoller, habe ich aber zeitlich nicht mehr hingekriegt. ESCAPE funktioniert hier aber schon (über Mnemonics), ist ein JDialog.	

de.ard.sad.rms.utilities.ExceptionMessageDlg

JIRA [#WEBMERLIN-1592] Shortcut für Drucken

Ausgangssituation:
Bei dieser Änderung geht es ums Drucken von Laufplänen bzw. einzelner Komponenten von Laufplänen. Ein Laufplan kann sein…
… ein *Rahmenplan*, d.h. ein Raster, das darstellt was längerfristig in einem bestimmten Turnus (monatlich, täglich, wöchentlich, jeden 2. Freitag…) an dem jeweiligen Tag für Sendungen und Beiträge laufen sollen.
… ein *Sendeplan*, d.h. ein konkreter Plan was für Sendungen und Beiträge laufen sollen. Dabei gibt es keinen Turnus, dafür aber ein festes Sendedatum.
Je nach Auswahl im Tab "Programmschema" links wird rechts der jeweilige Rahmenplan/Sendeplan angezeigt.



Problem:
Drucken sollte man nun entweder das gesamte Programmschema (den Rahmenplan) eines Laufplanes oder des markierten Elements in der aktuellen Ansicht im rechten Rahmen (den Sendeplan) können. Für beide Varianten gibt es im Menü Optionen. Aber der Shortcut STRG+P bewirkt nur das Drucken der aktuellen Ansicht. Im Menü steht er bei keiner der zwei Unter-Druckoptionen dabei.

Lösung:
Den Menüeinträgen habe ich die Shortcuts richtig zugeordnet und die Anzeige der Shortcuts hinter den Menüoptionen eingestellt. Bisher wurde bei Drücken von STRG+P nur der aktuell markierte Eintrag gedruckt. Dies habe ich beibehalten.
Die Funktionalität "Programmschema drucken" war schon als Methode hinterlegt, jedoch konnte sie bisher nur durch Auswahl des Menüpunkts ausgelöst werden. Diesen habe ich zusätzlich mit einem weiteren Shortcut STRG+S hinterlegt.

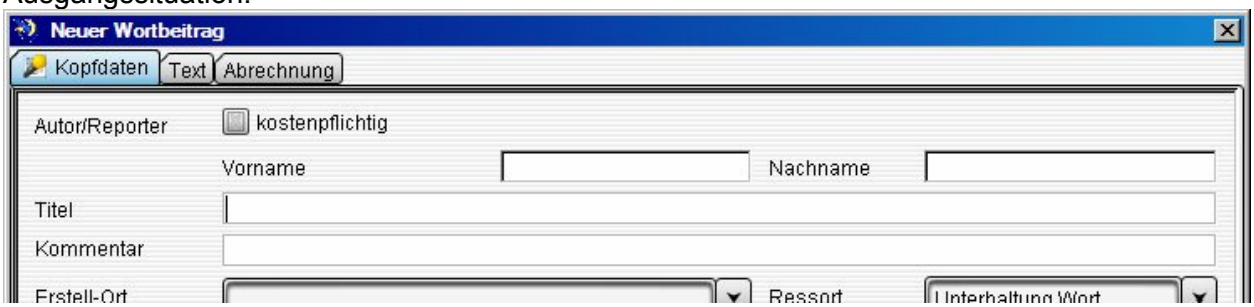
Bemerkungen:
Wenn man sich nicht in der Ansicht "Programmschema" befindet, sieht die Druckfunktion im Menü anders aus, da dann etwas ganz anderes ausgedruckt wird. Der Tooltip-Text aller drei Druckoptionen im Menü lautete vor meinen Änderungen lediglich "Drucken des Dokuments". Ich habe drei Varianten angelegt um den Hinweis mehr der Funktionalität anzupassen.

- "Drucken des Dokuments"
- "Drucken des Sende-/Rahmenplans"
- "Drucken des Programmschemas"

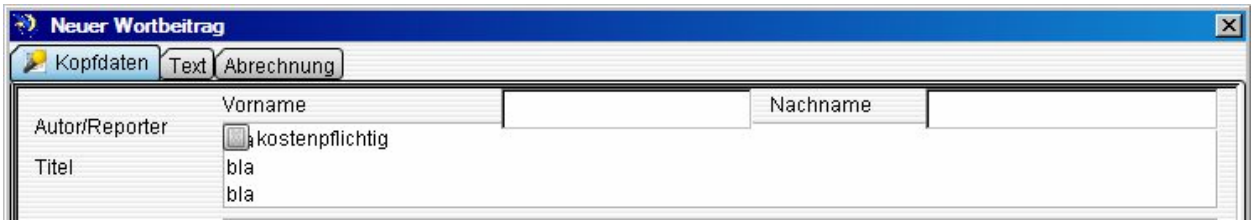
beteiligte Klassen:
de.ard.sad.rms.client.MenuView.java
de.ard.sad.rms.client.actions.PrintSchemaB2WAction.java
de.ard.sad.rms.client.actions.PrintPlanB2WAction.java
lib.menus.properties

JIRA [#WEBMERLIN-1478] Erfassung Wortbeitrag: zu viele Titelzeilen zerstören das Layout

Ausgangssituation:

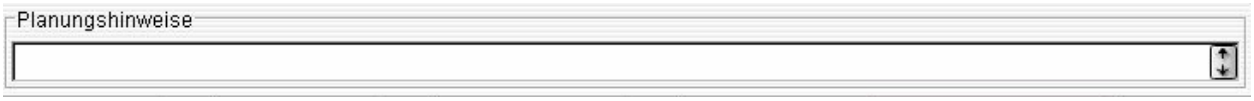


Die Eingabefelder "Titel" in den Masken "Wortbeitrag" und wie auch "Moderation" sind als einzeilige JTextAreas angelegt. Bei Zeilenvorschub mit ENTER vergrössert sich die TextArea innerhalb der Maske. Die Maske selbst hat ein festes Grössenverhältnis.



Problem:
Ab der dritten zusätzlichen Zeile schieben sich dann die anderen Komponenten der Seite übereinander.

Lösung:
Die Eingabefelder "Titel" und "Kommentar" wurden nicht in allen Masken einheitlich umgesetzt. Zwar sind es überall TextAreas, aber sie haben manchmals eine und manchmals drei Zeilen und in den oben genannten Masken haben sie keine Scrollfunktion. Da diese verhindern würde, dass das Gesamtlayout beim Vergrössern des Feldes zerstört wird, habe ich den Feldern die Scrollfunktion hinzugefügt. Sobald die initial vorgegebene Zeilenzahl überschritten wird, erscheinen rechts die Scroll-Buttons. Die Grösse des Feldes ändert sich nicht.



Bemerkungen:
Bereits in der Ergonomie-Studie hatte ich ja erwähnt, dass hier einige Mitarbeiter die Funktionen einer TextArea (Tabulatoren, Zeilenvorschub) nutzen und andere sie gar nicht benötigen. Hier kam es leider nicht zu einer einheitlichen Regelung, so dass ich nur die störendste Nebenwirkung der alten Situation umgesetzt habe.

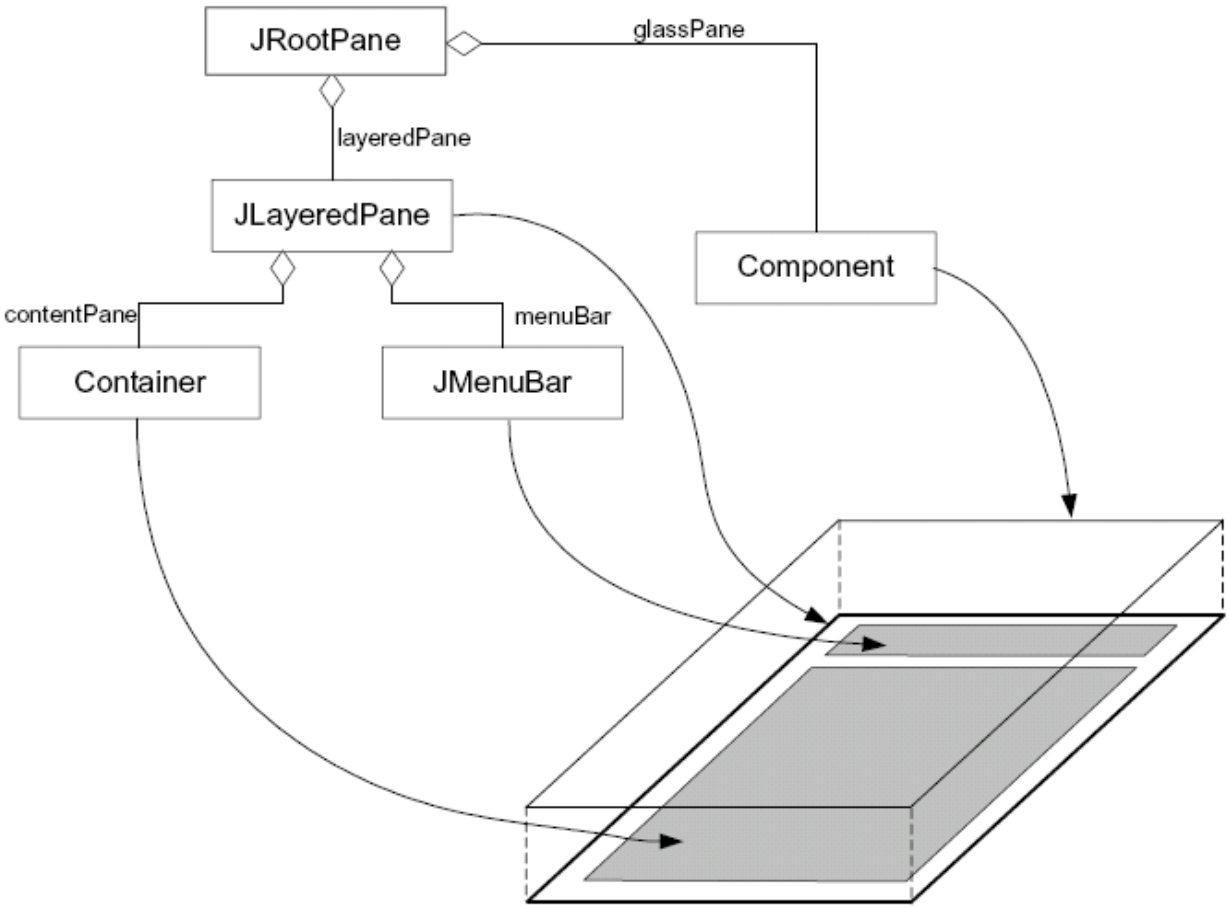
beteiligte Klassen:
de.ard.sad.rms.client.beitrag.ModerationDetailHeadView.java
de.ard.sad.rms.client.beitrag.WortbeitragDetailHeadView.java

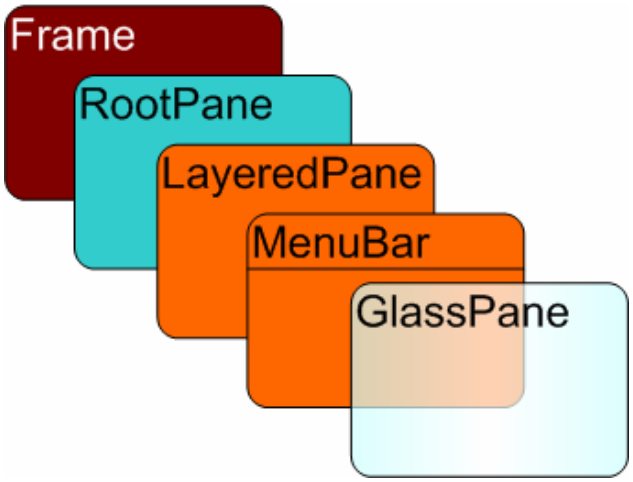
6. Java Swing und WebMerlin

Wie bereits mehrfach erwähnt, ist die GUI von WebMerlin aus diversen einzelnen Komponenten des Java AWT bzw. SWING zusammengesetzt. Hier als SWING-Neuling (in Info3 hatten wir das Thema seinerzeit ausgelassen) einen Überblick zu gewinnen ist recht aufwändig. Dennoch hatte ich zum Ende meines Projekts den Eindruck, dass ich die Struktur soweit verstanden habe. Ich will hier nun noch einmal ein paar grundlegende Strukturelemente sowie Klassen und Methoden aufführen, die mir bei WebMerlin immer wieder begegnet sind. Ausserdem will ich beschreiben, wie ich die Probleme, die sich mir stellten, angegangen bin.

Swing-Komponenten-Schachtelung

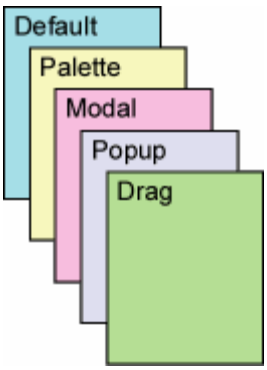
Swing beruht hauptsächlich auf Lightweight-Komponenten, die sich mittels eines Grafikkontextes selbst rendern. Alle Swing-Komponenten sind Container, können also mehrere Komponenten "aufnehmen", z.B. Text und Bilder (Image-Buttons).
Swing bietet neue Komponenten: Bäume (siehe Laufplan, Beitragsplan), Tabellen, Fortschrittsanzeige, innere Rahmen, Registerkarten (siehe z.B. in Wortbeitrag S. 16: Kopfdaten, Text, Abrechnung) verschiedene Rahmen und Separatoren, Tooltips, einfacher Mechanismus für parallele Menü- und Werkzeugleisten (GUI-Standard), transparente GUI-Elemente (z.B. bildbasierte Buttons). Die AWT-Mechanismen für die Dynamik und Statik bleiben (fast) erhalten.
Alle Swing-Komponenten sind von der MVC-Architektur motiviert. Trennung von View, Model (= Daten, Zustand) und Control (= Abfangen von Ereignissen).
Die Swing-Toplevel-Komponenten verwalten mehrere "Ebenen", die sich grafisch überlappen können (mit Transparenz).





Dabei enthält die RootPane eigentlich die vier anderen, bzw. sind sie von ihr abgeleitet! Die MenuBar ist die Pane, die den Content enthält.

Die LayeredPane kann wiederrum überlappend wie ein Container die folgenden Komponenten enthalten. Default steht hier für den eigentlichen FrameContent.



Am Beispiel der Titel-JTextAreas, denen ich die Scrollfunktion hinzugefügt habe, kann ich das nochmal beispielhaft erklären. Um der JTextArea den Scroll-Modus hinzuzufügen, muss diese in einer JScrollPane liegen, welcher wiederum in einem übergeordneten Container liegt.

Hier nun nochmal beispielhaft eine Übersicht der Änderungen die ich in WebMerlin vorgenommen habe:

de.ard.sad.rms.client.laufplan.PlanView.java

```
javax.swing.JButton jbtMusikbeitrag.setMnemonic(KeyEvent.VK_U);

javax.swing.JCheckbox jchMusikbeitrag.setLabel("(Musikbeitrag anlegen) - ALT+U");

javax.swing.JButton jbtModeration.addActionListener(new
   java.awt.event.ActionListener() {
       public void actionPerformed(java.awt.event.ActionEvent evt) {
              jbtModerationActionPerformed(evt);
       }
});

javax.swing.JCheckbox jchContribution.setActionCommand("(Beitragsbaum ein-
   /ausblenden)");
jchContribution.setLabel("(Beitragsbaum ein-/ausblenden) - ALT+B");
jchContribution.addActionListener(new java.awt.event.ActionListener() {
       public void actionPerformed(java.awt.event.ActionEvent evt) {
              jchContributionActionPerformed(evt);
       }
});

javax.swing.JButton jbtViewCheckBoxes.getAccessibleContext().setAccessibleDescription(
   "Konfigurationsmen\u00fc ein-/ausblenden - ALT+0");
```

de.ard.sad.rms.client.actions.DeleteNodeAction.java

```
result = de.ard.sad.rms.utilities.CustomizedJOptionPane.showOptionDialog(
       this.parent,
       ResourceBundle.getBundle("lib/errorMessages").getString("delete_folder"),
       ResourceBundle.getBundle("lib/general").getString("Titel_OptionPane"),
       JOptionPane.YES_NO_OPTION, JOptionPane.WARNING_MESSAGE, null, yesNoButtons,
       yesNoButtons[0]);
}
```

de.ard.sad.rms.client.BearbeitenDlg.java

```
int result = javax.swing.JOptionPane.showOptionDialog(
       this,
       ResourceBundle.getBundle("lib/general").getString("close_dialog"),
       ResourceBundle.getBundle("lib/general").getString("Titel_OptionPane"),
       JOptionPane.YES_NO_OPTION,
```

```
        JOptionPane.QUESTION_MESSAGE,
        null,
        new String[] {
        ResourceBundle.getBundle("lib/general").getString("Yes"),
        ResourceBundle.getBundle("lib/general").getString("No")
        },
        ResourceBundle.getBundle("lib/general").getString("Yes")
);
```

de.ard.sad.rms.utilities.GeneralDialog.java

```
this.getRootPane().setDefaultButton(this.jbtOk);
```

de.ard.sad.rms.client.MenuView.java

```
javax.swing.JmenuItem jmiPrintPlan.setAccelerator(
    KeyStroke.getKeyStroke(
         ResourceBundle.getBundle("lib/menus").getString("AcceleratorPrintPlan")
    )
);
```

de.ard.sad.rms.client.actions.PrintPlanB2WAction.java

```
this.putValue(Action.SHORT_DESCRIPTION, "Drucken des Sende-/Rahmenplans");
```

de.ard.sad.rms.client.beitrag.WortbeitragDetailHeadView

```
import javax.swing.ScrollPaneConstants;

private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JScrollPane jspTitel;

jScrollPane1 = new javax.swing.JScrollPane();
jspTitel = new javax.swing.JScrollPane();
jtaTitel = new javax.swing.JTextArea();

jspTitel.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
```

Meine Vorgehensweise:
Mein erster Auftrag waren die Shortcuts für die Beitragsbaum-Buttons. Ich hatte nur die laufende Anwendung vor mir und den Sourcecode in zahlreichen Packages in meiner IDE. Wie bin ich nun vorgegangen um herauszufinden, welche Komponente in welcher Klasse die Actions der Buttons auslöst? Zu dem Zeitpunkt wusste ich nur soviel, dass die Bilder auf den Buttons irgendwo in den Quellen hinterlegt sein müssen. Ich habe also nach den Bild-Dateien dieser Buttons gesucht. Als ich das lib.images-Verzeichnis gefunden hatte, habe ich mit dem Dateinamen eines Bildes weitergesucht. Verwendet wurde dieser Name in der ImageFactory, in der er einem javax.swing.ImageIcon zugewiesen wurde. Dieses wurde dann wiederum in der Klasse PlanView verwendet, die den ganzen Button-Baum mitaufbaut.

Meist habe ich mich buchstäblich von Klasse zu Klasse gehangelt. Vom Einstiegspunkt (ein Icon, eine Button- oder MenuItem-Beschriftung) bis zur Zielklasse.

Nachdem ich wochenlang immer wieder so vorgegangen bin, merkte ich es dann auch relativ schnell, wenn sich mir wieder ein ähnliches Problem stellte. Die Suche nach der relevanten Komponente oder Klasse ging dann auch von mal zu mal schneller. Während ich für die Shortcuts für die Icons des Beitragsbaumes (JIRA Issue #1564) noch einige Tage gebraucht hatte, war die Sache mit den Scrollbalken in den Titel-Feldern (JIRA Issue #1478) innerhalb eines Nachmittages abgehandelt.

7. Probleme

Das Einarbeiten in die Entwicklung einer komplexen Anwendung war sehr aufwändig für mich. Die meiste Zeit ging eigentlich dafür drauf, überhaupt erstmal herauszufinden, welche Java-Klasse nun dafür zuständig ist, dass in der Anwendung WebMerlin eine bestimmte Aktion ausgelöst wird. Ich habe mir dabei dann einen Weg angeeignet, dem auf die Spur zu kommen.

Da ich nur ein oder zweimal die Woche da war, habe ich nicht direkt alle Änderungen des Systems mitbekommen. Hin und wieder musste ich nachträglich einen Versionswechsel nachvollziehen oder mein Datenbank-Schema neu anlegen, um wieder mit der Anwendung testen zu können.

Einmal wurde die Datenbank komplett neu aufgesetzt (neue Version) und mein User-Schema wurde in der neuen DB nicht angelegt. In einer solchen Situation ist man natürlich von anderen Mitarbeitern abhängig, die die Datenbank verwalten und einem dann (nach einer gewissen Leerlaufzeit) das Schema wieder anlegen.

Die initiale Einrichtung der Entwicklungs- und Testumgebung war relativ aufwändig. Die Entwicklung erfolgte in der IDE NetBeans. Die Quelldateien werden in einem CVS gehalten. Es gibt ein ANT-Build-File. Einmalig musste ich sämtliche Quellen auf den lokalen Rechner auschecken. Dann musste ich "fehlende" JAR-Archive zu den lokalen Bibliotheken hinfügen. Diese JARs sind auf einem zentralen Server bereitgestellt. Mein lokaler JBoss musste ebenfalls in seinen Einstellungen angepasst werden. Zum Glück sass ich direkt bei einem wahren Meister seines Fachs, der die Anwendung von Anfang an mit begleitet hatte und mir in (fast) jeder Situation mit Rat und Tat zur Hilfe stehen konnte.

Nicht zuletzt war der Netbeans Form-Editor auch eine grosse Hilfe bei der Entwicklung mit SWING. Naja, eine Hilfe und teilweise auch ein Hindernis. Einerseit kann man z.B. direkt neue Buttons per Drag & Drop in die Komponente einfügen, sie dort verschieben oder direkt über das Properties-Fenster editieren. Andererseits ist der aus dem Form generierte Code in der Java-Klasse für die Bearbeitung gesperrt. Wenn man dort etwas ändert, wird es spätestens beim nächsten Öffnen durch Neugenerierung aus dem Form überschrieben. Es gab so ein paar Situationen, in denen ich aber eben am liebsten im "blauen Code" was geändert hätte…

8. Fazit

Die WebMerlin-Aufgabe reizte mich, weil ich selbst einen starken User-Focus habe, wenn ich Software entwickle. Ich konnte durch das Projekt einmal beide Seiten sehen. Die Entwicklung am einen Ende und die Nutzung der Anwendung am anderen Ende. Meine Absicht ist die, dass beide Enden wirklich zusammenlaufen, dass also nicht irgendwo hinentwickelt wird sondern der Nutzer bei der Entwicklung stets im Focus steht. Jeder Nutzer hat auch stets eine Meinung zu dem Programm. Und diese sollte auch berücksichtigt und angehört werden, weil daraus oftmals wichtige Impulse für die Entwicklung hergeleitet werden können.

Dass ich während der Projekt-Laufzeit nicht alle Änderungen wie von den Usern gewünscht umsetzen konnte, lag vor allem daran, dass so viele Parteien an der Anwendung beteiligt sind. Nicht nur die kleine User-Gruppe des SWR in Mainz hatte da ein Wörtchen mitzureden, auch die anderen RFAs, deren User und die WebMerlin-Projektgruppe sind an der Entscheidungsfindung beteiligt. Ich habe zum grössten Teil umgesetzt, was allgemein als sinnvoll erachtet wurde. Dies wiederspricht natürlich wiederrum meinem obigen Prinzip, aber es ist nunmal nicht alles Gold was glänzt.

Ausserdem hatte ich die Gelegenheit einmal andere Abteilungen der Firma SWR kennenzulernen. Während meines Praxissemesters beim SWR (September 2004 - Februar 2005) in der Abteilung Server und Netze hatte ich schon eine Führung durch die Fernsehstudios in Baden-Baden erlebt. Bei dieser Projektarbeit durfte ich nun die Hörfunk-Redaktionen in Mainz besuchen. Wenn man in einer Firma arbeitet (sei sie auch noch so gross oder klein), finde ich es unheimlich wichtig dass man sich auch mal andere Abteilungen anschaut und mit den Leuten dort spricht.

Wie schon eingangs erwähnt habe ich dreimal soviel Zeit für das Projekt aufgewendet, wie ursprünglich geplant war. Das hatte ich aber nach meinen Erfahrungen von meinem ersten Software-Projekt auch nicht anders erwartet. Es hat Spass gemacht und ich habe viel Neues gelernt, das ist die Hauptsache.

9. Begriffsklärung

Im Verlauf des Projekts habe ich mich parallel auch intensiv mit den theoretischen Grundlagen von Usability und Software-Ergonomie beschäftigt. Hier deshalb ein kleiner Abriss zu diesen Themen.

Die **Mensch-Computer-Interaktion** (**MCI**; englisch *Human-computer interaction*, *HCI*) ist ein Teilgebiet der Informatik und beschäftigt sich mit der benutzergerechten Gestaltung von interaktiven Systemen. Dabei werden neben Erkenntnissen der Informatik auch solche aus der Psychologie, der Arbeitswissenschaft, der Kognitionswissenschaft, der Ergonomie, der Soziologie und dem Design herangezogen. Ein wichtiger Aspekt in der Mensch-Computer-Interaktion ist die Gebrauchstauglichkeit von Soft- und Hardware.

Software-Ergonomie (SE) ist die Wissenschaft von der Benutzbarkeit und Gebrauchstauglichkeit von Computer-Programmen. Sie ist ein Teilgebiet der Mensch-Computer-Interaktion. Gegenstandsbereich der SE ist der arbeitende Mensch (Softwarenutzung an Arbeitsplätzen). Es wird die Benutzung von beziehungsweise die Interaktion mit Computern (Arbeitssoftware, WWW, Spiele, ...) betrachtet. Dies bedeutet die Berücksichtigung psychologischer Aspekte beim Software-Entwurf, um eine optimale Mensch-Maschine-Schnittstelle zur Verfügung zu stellen. Im Bereich der SE existieren formale Richtlinien für die Gestaltung von Bildschirmarbeitsplätzen, für die Darstellung von Informationen am Bildschirm sowie deren Manipulation durch Eingabegeräte. Diese Richtlinien sind in der Bildschirmarbeitsverordnung (BildscharbV) sowie im Standard ISO 9241 der International Organization for Standardization festgehalten und sollten damit bei der Erstellung von Anwendungssoftware berücksichtigt werden.
Als Grundlage für die Definition ergonomischer Prinzipien eignet sich das **ABC-Modell**, das den Nutzungskontext der Ergonomie als Beziehungen zwischen **Aufgabe**, **Benutzer** und **Computer** beschreibt. Ein System ist angemessen, wenn es die zur Lösung der Arbeitsaufgabe erforderlichen Funktionen bereitstellt. Ein System ist handhabbar, wenn es dem Benutzer eine leichte Erlernbarkeit, Bedienbarkeit und Verständlichkeit ermöglicht. Ein System ist persönlichkeitsförderlich, wenn es den Fähigkeiten und Kenntnissen des Benutzers (Benutzermodell) angepasst ist und den Prinzipien der Arbeitsgestaltung entspricht.

In Teil 10 der Normenreihe **ISO 9241** "Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten" *Grundsätze der Dialoggestaltung* werden folgende Qualitätskriterien definiert, die sich durch Verfeinerung der Kriterien des ABC-Modells ergeben:

1. Aufgabenangemessenheit - geeignete Funktionalität, Minimierung unnötiger Interaktionen
2. Selbstbeschreibungsfähigkeit - Verständlichkeit durch Hilfen / Rückmeldungen
3. Steuerbarkeit - Steuerung des Dialogs durch den Benutzer
4. Erwartungskonformität - Konsistenz, Anpassung an das Benutzermodell
5. Fehlertoleranz - Vermeidung schwer wiegender Fehler, leichte Korrektur
6. Individualisierbarkeit - Anpassbarkeit an Benutzer und Arbeitskontext
7. Lernförderlichkeit - Anleitung des Benutzers, Metaphern

Usability bedeutet die Gebrauchstauglichkeit oder Benutzungsfreundlichkeit von Produkten, hier im speziellen Fall die Gebrauchtstauglichkeit einer Anwendung. Die Usability ist ein Mass dafür, wie leicht etwas zu benutzen ist. Je besser die Nutzer damit zurechtkommen, desto höher ist die Usability der Anwendung.
Die Usability einer Software ist von ihrem Nutzungskontext abhängig. Im Teil 11 der ISO 9241 werden drei Leitkriterien für die Usability einer Software bestimmt:

- Effektivität zur Lösung einer Aufgabe
- Effizienz der Handhabung des Systems
- Zufriedenheit der Nutzer einer Software

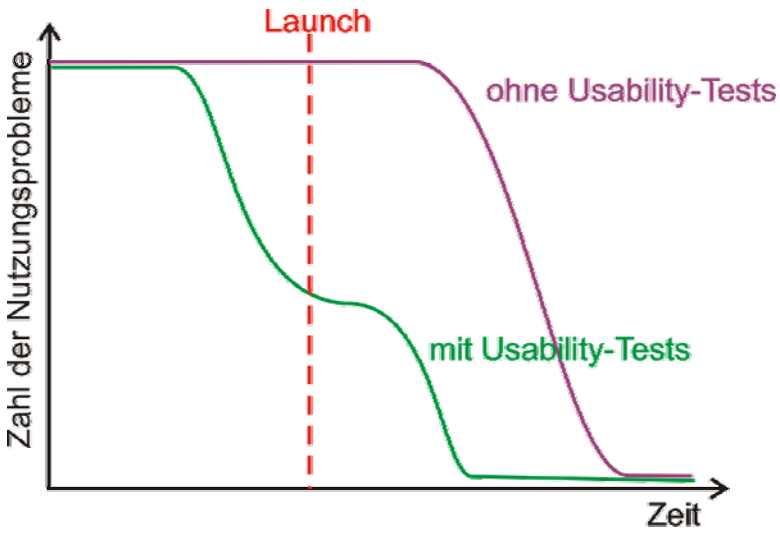
Um die Usability einer Anwendung festzustellen, müssen Untersuchungen (Usability-Tests) durchgeführt werden. Dabei versucht man herauszufinden, was die Benutzer erwarten, was sie kennen und was sie gewohnt sind. Wenn man dies dann in der Anwendung integriert, macht man sie benutzerfreundlich. Im weiteren Sinne kann man sagen, dass dadurch

- eine bessere Qualität
- eine intensivere, häufigere Nutzung
- eine grössere Nutzergruppe

erreicht werden kann.

Kennzeichen einer benutzerfreundlichen Anwendung:

- barrierefrei
- tolerant gegen Benutzungsfehler
- verständlich
- technologisch robust
- individuell zugeschnitten
- steuerbar
- den Erwartungen entsprechend
- den Aufgaben angemessen
- dem Erlernen förderlich



Ein Usability-orientierter Entwicklungsprozess deckt Probleme frühzeitig auf. Er macht es wahrscheinlicher, dass das Produkt vom Nutzer angenommen wird. Wenn der erste Test durch den User negativ ausfällt, ist der Aufwand für den Herstellenden erheblich höher, dem Produkt noch zum Erfolg zu verhelfen.

Usability hat viele Vorteile:

- schafft Erfolgssicherheit und Planbarkeit
- spart Zeit
- spart Kosten
- bringt neue Kunden
- erhöht Marktwert und Loyalität
- schafft Vermarktungs-Vorteile
- Usability-Test generieren neue Ideen
- bringt Spass
- ist ein Qualitätsmerkmal

10. Quellen

Beteiligte und Betriebsmittel
http://www.swr.de
http://www.gft.com
http://jira.atlassian.com/

Usability
HEINSEN/VOGT (2003), *Usability praktisch umsetzen*, Hanser Verlag
KRUG, STEVE (2002), *Don't make me think*, mitp
STRUNK/WHITE (1999), *The Elements of Style*, Longman

http://www.usability-umsetzen.de/ (Seite zum Buch)
http://www.sensible.com/ (Seite von Steve Krug)
http://www.worldusabilityday.de/
http://www.uidesign.de/ (User Interface Design GmbH, Burmester)
http://www.gui-design.de/ (gui Design, Beck, FHTE)
http://www.sozialnetz.de/ca/pq/mcu/ (Artikel von Herrn Burmester)
http://www.hci.iao.fraunhofer.de (Usability-Testing)

Java Swing
LOY/ECKSTEIN/WOOD (2002), *Java Swing*, O'Reilly
MARINACCI/ADAMSON (2005), *Swing Hacks*, O'Reilly
WALRATH/CAMPIONE/HUML (2004), *The JFC Swing Tutorial*, Addison-Wesley

http://java.sun.com/docs/books/tutorial/uiswing/index.html
http://java.sun.com/j2se/1.4.2/docs/api/index.html