Whitepaper zur Literaturverwaltung

„Architektur der GUI"

von Jurij Henne

erstellt im Rahmen der Projektgruppe
„Open Source und virtuelle Unternehmen"

unter der Leitung von
Prof. Uwe Schneidewind und Hendrik Eggers (Lehrbeauftragter)

Inhaltsverzeichnis

Kapitel 1

GUI-Architektur

1.1 Einleitung

Ein wichtiger Bestandteil von Liabolo ist sicherlich seine GUI. Der Begriff GUI stammt aus dem Englischen und ist eine Abkürzung für Graphical User Interface. Übersetzt ins Deutsche bedeutet eine GUI nichts anderes als „Grafische Benutzerschnittstelle“. In der Softwareergonomiewelt hat sich jedoch der korrekte Begriff der „Grafische Benutzungsschnittstelle“ durchgesetzt. Eine GUI ist also eine „Mensch-Maschine-Schnittstelle“.

Die Bedienung einer GUI ist ein interaktiver Prozess. Der Benutzer kann mit Hilfe einer Maus oder Tastatur grafische Elemente auf dem Computerbildschirm bedienen und die GUI somit zur Ausführung definierter Befehle veranlassen. Die GUI ihrerseits überwacht die Aktionen des Benutzers und leitet die Anweisungen an die Verarbeitungseinheit weiter. Die geänderten Zustände der Anwendung teilt sie dem Benutzer in Form aktualisierter Bildschirminhalte mit.

Bei der Gestaltung heutiger GUIs wird oftmals die so genannte Desktop-Metapher verwendet. Diese definiert einen virtuellen Schreibtisch(Desktop), deren Elemente in ihrer Darstellung und Funktion den der realen Welt nachempfunden sind. Der Desktop definiert aber auch das Anwendungsfenster.

Die wichtigsten Funktionen der Anwendung werden entweder in Form von grafischen Symbolen(Icons) direkt auf dem Desktop dargestellt oder sie finden ihren in einer Menuleiste Platz. Die eigentlichen funktionalen Inhalte eine Anwendung werden jedoch meist in den so gennanten Fenstern in Form von Masken bzw. Formularen präsentiert. Wie die einzelnen Fenster auf dem

Desktop positioniert und dargestellt werden hängt meist mit dem gewählten Architektur-Konzept zusammen(siehe Konzeptioneller Entwurf).

Bei der Konzeption und Umsetzung der Liabolo-GUI werden viele der modernen Ansätze und Standards aufgegriffen und berücksichtigt. Die Anwendung soll dem Benutzer eine möglichst komfortable Benutzungsoberfläche präsentieren und eine intuitive Bedienung der zahlreichen Funktionen ermöglichen. Die nachfolgenden Kapiteln beschreiben einige Überlegungen und Entscheidungen im Bezug auf die gewählte Architektur und skizzieren den aktuellen Stand der Umsetzung.

1.2 Konzeptioneller Entwurf

In diesem Kapitel wird eine passende Darstellungsarchitektur für die Liabolo-GUI untersucht. Darüber hinaus erfolgt eine Spezifikation der notwendigen Maskentypen. Anschließend werden einige Überlegungen zur grafischen Aufwertung der Darstellungsqualität zweckst komfortableren und intuitiveren Bedienung aufgestellt.

1.2.1 Darstellungsarchitektur

Wie die Inhalte einer Anwendung auf dem Bildschirm präsentiert werden, hängt meist von der gewählten Architektur ab. Grundsätzlich wird es zwischen zwei verschiedenen Ansätzen unterschieden, der Single Document Interface(SDI) Architektur und der Multiple Document Interface(MDI) Architektur. Die Beiden werden zunächst gegenübergestellt und verglichen.

Single Document Interface(SDI)

Die Singledokumente erlauben zur gleichen Zeit die Darstellung nur eines geöffnetes Formular bzw. einer Eingabe-/Einstellmaske. Dies bedeutet aber nicht zwangsläufig, dass alle anderen, zuvor geöffneten, Masken dafür geschlossen werden müssen. Nicht selten wird für die Darstellung neuer Masken eine weitere Instanz der Hauptanwendung gestartet.

Viele der heutigen SDI-basierten Anwendungen aber realisieren die Umschaltung zwischen allen geöffneten Masken mittels einer Registerkartenansicht in einem Hauptfenster. Diese Darstellungsweise sorgt für einen gewissen Maß an Übersicht und Ordnung, bringt jedoch eine Reihe von designtechnischen Nachteile mit sich. Die Abmessungen der geöffneten Masken müssen zwang-

läufig an die Abmessungen des Anwendungsfensters angepasst werden. Dabei gibt es mindestens zwei Möglichkeiten dies zu bewerkstelligen:

- Die Größe des Anwendungsfensters wird an die Größe des jeweils aktiven Formulars bzw. der aktiven Eingabenmaske angepasst. Dies sollte nach Möglichkeit vermieden werden, da eine mausbegleitende Änderung des Anwendungsfensters den Benutzer verwirren könnte. Darüber hinaus werden ausgewählte Bedienelemente, wie Menuleiste oder Werkzeugleiste bei einem zu kleinen Fenster mitskaliert und sind so unter Umständen nicht mehr vollständig sichtbar bzw. bedienbar.

- Die Größe des aktiven Formulars bzw. der aktiven Eingabenmaske an die feste Standardgröße des Anwendungsfensters anpassen. Dies vermeidet falsche Darstellung der Bedienelemente. Die Darstellung und Positionierung der formular- bzw. maskeninternen Elemente müssen aber explizit für jedes Formular auch für eventuelle Maximierungsmöglichkeit hin optimiert werden. Denn sonst kann die vom Benutzer erzwungene Grössenänderung des Anwendungsfensters die vorher festgelegte Layouts „verzerren“, indem es der internen Elemente der Maske auf dem Bildschirm neu orientiert und die Darstellung nicht mehr optimal werden kann.

  Um die Problematik anschaulich darstellen zu können, wird an dieser Stelle ein Beispiel präsentiert. Der Benutzer schaltet zwischen zwei unterschiedlichen Masken. Die beiden Masken beanspruchen unterschiedlich viel Platz auf dem Bildschirm. Es ergeben sich drei mögliche Vorgehensweisen für den Entwickler:

  - Die Maskenelemente bleiben aneinander gebunden, in ihrer Gesamtheit müssen sie jedoch entweder zentriert oder in einem definierten Bereich des Anwendungsfensters positioniert werden. Können die Maskenelemente den ganzen Bereich des Hauptfensters nicht ausfüllen, bleiben unter Umständen große Bereiche ungenutzt. Diese können leider auch nicht zur Darstellung anderer Informationen verwendet werden. Bei den Masken, die die Größe des Hauptfensters überschreiten, kommen Scrollleisten zum Einsatz.
  - Die Maskenelemente verteilen sich gleichmäßig auf dem gesamten Darstellungsbereich des Hauptfensters. Allerdings werden dadurch die jeweils zusammen gehörende Elementenpaare (z.B. Beschriftung des Texteingabefeldes und das eigentliche Texteingabefeld) auseinander gerissen. Bei den Masken, die die Größe des Hauptfensters überschreiten, kommen Scrollleisten zum Einsatz.

  - Die Maskenelemente verteilen sich gleichmäßig auf dem gesamten Darstellungsbereich des Hauptfensters und werden je nach Grösse des Hauptfensters mitskaliert. Dies hat zu Folge, dass die einzelnen Elemente bei kleineren Masken relativ grosse Bereiche ausfüllen, bei größeren Masken sehr stark herunterskaliert werden müssen, um im sichtbaren Bereich des Hauptfensters dargestellt werden zu können.

Die beiden letzten Szenarien haben höchstens eine theoretische Bedeutung, denn sie kommen wegen der ersichtlichen Nachteile fast nie zum Einsatz. Im Allgemeinen ist es sehr schwierig zu entscheiden, welche Vorgehensweise für gestellte Anforderungen optimal seien dürfte. Die tatsächlichen Darstellungsverhältnisse sollen daher an einem geeigneten Demonstrationsprototyp untersucht werden.

Multiple Document Interface (MDI)

Diese Art der GUI-Architektur erlaubt eine parallele, weitgehend voneinander unabhängige Darstellung und Verwaltung von mehreren internen Unterfenstern innerhalb des Hauptfenster der Anwendung(Desktop). Der Desktop verhält sich dabei als ein universeller Container, der sowohl die Bedienelemente wie Menu-, Werkzeugleisten und sonstiges, als auch die eigentlichen Eingabemasken aufnehmen und darstellen kann.

Die einzelnen Masken existieren als eigenständige Unterfenster. Sie können standardmäßig minimiert, maximiert und manuell vergrößert bzw. verkleinert und geschlossen werden. Im Folgenden werden einige weitere Vorteile der MDI-Architektur dargestellt:

- Für den Wechsel zwischen zwei parallel geöffneten Unterfenstern genügt oft ein Mausklick. Dieser Vorteil wird jedoch durch die Registerkartenansicht bei den SDI-Anwendungen relativiert.

- Die Formulare und Masken können ihre optimale Darstellungsgröße einnehmen und behalten und zwar unabhängig von der aktuellen Größe des Anwendungsfensters.

- Geöffnete Formularmasken können von dem Benutzer innerhalb des Anwendungsfensters frei positioniert werden. Dies ermöglicht ein angenehmes Arbeiten und schnellen Zugriff auf die oft verwendeten Funktionen.

Aber auch der MDI-Ansatz hat einige Nachteile. So wird oftmals die fehlende Ordnung bei vielen geöffneten Masken bemängelt. Oftmals werden einige Masken von anderen verdeckt,

Wahl des geeigneten Architekturansatzes

Das Liabolo-Team hat insgesamt drei Demonstrationsprototypen erstellt und erprobt, zwei nach dem SDI-Ansatz und einen nach dem MDI-Ansatz. Die Vorteile und Nachteile der jeweiligen Architektur wurden im Hinblick auf die Liabolo-Anforderungen auf diese Weise analysiert. Die Darstellungsflexibilität der SDI-Prototypen hat jedoch nicht überzeugen können. Da die Liabolo-GUI viele unterschiedlich große Masken verwalten, diese gleichzeitig geöffnet halten und darstellen soll, scheint der MDI-Ansatz geeigneter zu sein. Weitere wichtige Aspekte, die berücksichtigt wurden, lassen sich dem Whitepaper zu Softwareergonomie entnehmen.

1.2.2 Maskentypen

Nachdem der geeignete Darstellungsansatz feststeht, sollen die notwendigen Formular und Dialogtypen und ihre Eigenschaften spezifiziert werden. Wie bereits angedeutet, soll die Liabolo-GUI eine Vielzahl an unterschiedlichen Masken verwalten können.

Um eine optimale Darstellung garantieren zu können, wird entschieden die Größe der Masken nicht explizit festlegen zu lassen. Die meisten Containertypen der JAVA Swing-Grafikbibliothek erlauben komfortable Pack-Funktionalität[1], die äußerst zuverlässig funktioniert und keine expliziten Anpassungen erfordert. Es erfolgt nur die Spezifikation der Maskentypen.

Formulare

Mit Formularen werden Masken bezeichnet, die in erster Linie zu Erfassung von Daten dienen. Auch Liabolo macht ausgiebigen Gebrauch von Formularen, da nach Anwendungsspezifikation große Datenmengen erfasst werden sollen, um deren Verwaltung erst ermöglich zu können. Allerdings werden auch bei Liabolo auch Masken zur Auflistung der Datensätze und deren Bearbeitung als Formulare bezeichnet. Letztendlich haben alle Liabolo-Formulare folgende gemeinsame Eigenschaften, welche sie als solche identifizieren:

[1] Hierbei werden alle darstellbaren Elemente der Maske erfasst, deren bevorzugte Größe und Position(gemäß dem festgelegten Layout) ermittelt und daraus die optimale Maskengesamtgröße ermittelt und festsetzt.

- Jedes Formular kann zur selben Zeit nur eine offene Instanz besitzen. Befindet sich ein Formular im minimierten Zustand bzw. wird durch andere Formulare verdeckt, so wird es wieder maximiert und ausgewählt, sobald der Benutzer versucht, eine neue Instanz davon zu öffnen.

- Alle Formulare können frei auf dem Desktop positioniert und durch den Benutzer mit der Maus verschoben werden.

- Unterschiedliche Formulare können parallel auf dem Desktop geöffnet verwaltet werden(MDI-Ansatz)

Die Liabolo-GUI unterscheidet grundsätzlich zwischen drei unterschiedlichen Formulartypen:

- **„Neu/Bearbeiten"-Formulare** dienen zur Erfassung von neuen Daten und deren nachträglichen Bearbeitung, sofern es erlaubt ist[2].

- **„Browse"-Formulare** dienen zur tabellarische Auflistung aller verfügbaren Datensätze. Diese Formulare erlauben nicht nur die Bearbeitung oder Löschung der aufgelisteten Datensätze, sondern stellen auch weitere nützliche Funktionen zu deren Verwaltung[3]. Die Browse-Formulare können untergeordnete Browse-Formulare aufrufen. Sie sollen keine Daten erfassen können.

- **„Suche"-Formulare** dienen dazu die gewünschten Suchergebnisse zusammenzustellen und an die „Browse"-Formulare zwecks der Darstellung zu übertragen. Um möglichst komfortable Bedienung der Suche zu gestalten, sollen die Suchformulare einige Funktionalitäten der „Browse"-Formulare aufweisen, indem sie grobe Datenstukturen dem Benutzer präsentieren und detaillierte Informationen auf Anfrage zusammenstellen.

Die „Neu/Bearbeiten"-Formulare und die meisten „Browse"-Formulare und ebenso die „Suche"-Formulare sollten aus softwareergonomischen Gründen vom Benutzer in ihrer Darstellungsgröße nicht modifiziert werden können. Allerdings sollte dem Benutzer die Möglichkeit gegeben werden, ausgewählte „Browse"-Formulare auf die Desktopgröße maximieren zu können, um die

[2] z.B. ein Bearbeitung eines Metadatensatzes. Oftmals wird nur eine Untermenge der ursprünglich eingetragenen Daten zur Bearbeitung freigegeben, um eine gewisse Datenintergrität garantieren zu können.

[3] z.B können können ausgewählte Individuallisten exportiert werden

Fülle an eventuell vorhandenen Informationen[4] optimal betrachten und verwalten zu können.

Alle Formulare sollen die gleichen Darstellungsmerkmale aufweisen. Dieses soll durch den Einsatz einer generischen Oberklasse(DefaultForm) sichergestellt werden. Diese Klasse soll einen einheitlichen Rahmen für alle Formulare zur Verfügung stellen. Erst die abgeleitete Klasse soll diesen, zunächst leeren Rahmen mit Inhalten füllen.

Die Oberklasse soll alle notwendigen Attribute und Methoden bereitshalten, um jeder Unterklasse sinnvollen Anpassungen an dem Grundgerüst ermöglichen zu können. Die ableitende Klasse bestimmt über die Inhalte der Titelleiste[5] oder Aktualiesierungszeiträume. Die beiden Formulartypen sollen je eine wohl definierte Oberklasse(AddForm und BrowseForm)[6] erhalten, die alle gemeinsamen Attribute und Methoden kapseln und somit zu einer sauberen Source-Struktur und besseren Perfomanz beitragen.

Dialoge

Dialoge[7] sollen immer dann zum Einsatz kommen, wenn der darzustellende Inhalt keine vollwertige Formularmaske voraussetzt und/oder eine sequenzielle Abarbeitung der Benutzereingaben vorausgesetzt wird. Je nach Einsatzbereich solle zwischen zwei Typen unterschieden werden:

- **Eingabedialoge** dienen zur Erfassung benötigter Parameter(Benutzerabfragen oder Eingaben). Oft wird vom Benutzer entweder eine Bestätigung oder Ablehnung erwartet, in seltenen Fällen kommen einige weitere Auswahloptionen hinzu. Wie bereits erwähnt, sollten die Dialoge standardmäßig keine Formularfunktionalität anbieten.
- **Benachrichtigungsdialoge** können in Bestätigungsdialoge und Warnungsdialoge unterteilt werden. Bestätigungsdialoge benachrichtigen den Benutzer über erfolgreich ausgeführte Aktionen. Die Warnungsdialoge weisen auf die eventuell fehlende Eingabeparameter oder falsch ausgeführte Bedienschritte hin.

[4]Die Auflistung von Metadatensätzen und ihrer Attribute dürfte in der optimalen Darstellung den Ausgabebereich eines jeden gewöhnlichen Desktops sprengen.

[5]Ein indivueller Titel und evtl. Icon(siehe auch Grundlagen der Softwareergonomie).

[6]Diese Anforderung wurde noch nicht erfüllt,da die Notwendigkeit derselben sich erst in der späten Implementierungsphase ergab. Sie soll aber in Zukunft umgesetzt werden

[7]Ein Satz zusammengehöriger Optionen, die auf dem Bildschirm in Form eines Fensters oder einer Box angezeigt werden, in der man Einstellungen vornehmen kann(http://www.glossar.de/glossar/amglos_d.htm)

Die Dialoge haben gegenüber Formularen eine besondere Eigenschaft: sie sind modal[8]. Das heisst, dass aktive Dialogfenster die Benutzung des Programmfensters bis zur Beendigung der Eingabe unterbrechen. Also logische Konsequenz dessen können keine mehrere Dialogfenster parallel dargestellt werden. Die Abarbeitung der Dialogfenster durch den Benutzer soll also strikt sequenziell erfolgen.

Der Einsatz von Dialogen richtet sich an den allgemeinen softwareergonomischen Richtlinien[9] und heute üblichen Programmierpraktiken.

1.2.3 Grafische Gestaltung

Um die GUI optisch ansprechend gestalten zu können und dem Benutzer eine möglichst intuitive Bedienung der verfügbaren Funktion zu ermöglichen wird das Liabolo-Team einen „Third Party Look and Feel“[10]und zahlreiche Icons[11] einsetzen. Je nach Verfügbarkeit sollen im Internet bereits vorhandene Icons verwendet werden. Hierzu wurde eine ausführliche Recherche durchgeführt und u.a. folgende Adressen als mögliche Quelle ermittelt:

- Access Paradies(URL: http://www.access-paradies.de/icon/index.php) bietet etwa 14000 Icons zum Download an. Da die Sammlung in erster Linie für die MS-Access-Programmierung angedacht wurde, sind die Lizenzbestimmungen nicht eindeutig.

- Icon Factory (URL: http://www.iconfactory.com/) bietet einen unüberschaubaren Archiv an Freeware Icons für alle Betriebssysteme an. Leider haben die meisten Icons eine Mindestgröße von 32x32 Pixel.

- PixelMagick (URL:http://jimmac.musichall.cz/ikony.php3) bietet eine sehr komfortable Übersicht über die vielen verfügbaren Icons, die ursprünglich für das GNOME-Project entworfen wurden und zum grössten Teil unter der GPL-Lizenz stehen.

Da die Icons unter JAVA-Swing als gewöhnliche Grafiken eingebunden werden(JPG,GIF,PNG) ist ihr, oft betriebssystemspezifische, Quellformat nicht problematisch. Die Grafiken lassen sich mit den meisten Grafikbearbeitungsprogrammen in gewünschten Format konvertieren.

---

[8]Umgangsprachlich übersetzt: “die Art und Weise bezeichnend“

[9]Lesen sie dazu bitte das Whitepaper zu Softwareergonomie

[10]Mit Look and Feel (LAF) werden, meist durch Hersteller oder Konsortien standardisierte Design-Aspekte bezeichnet. Quelle:http://de.wikipedia.org/wiki/Look%26Feel

[11]von griechisch: eikon Bild

Viel umständlicher ist es die Standard-Icons(32x32Pixel) auf die gewünschte Größe von 16x16 Pixel herunter zu skalieren. Die zahlreichen Skalierungsversuche brachten leider nur ungenügende Ergebnisse mit sich, da die meisten Icons in dem Bitmap-Format vorliegen. Vektorgrafiken lassen sich zwar verlustfrei herunter skalieren, die dargestellten Motive sind trotzdem nicht mehr eindeutig erkennbar. Die Icons mit den Originalabmessungen von 16x16 Pixel haben bei der Auswahl deshalb die höchste Priorität.

1.2.4 Funktionale Anforderung

Die Liabolo-GUI soll dem Benutzer folgende Funktionen zur Verfügung stellen:

- **Bibliothekstatus anzeigen**
  Der Benutzer soll eine baumartige Darstellung aller, in der lokalen Bibliothek abgelegten, Strukturen abrufen können. Die Darstellung der Daten soll in einer Formularmaske erfolgen.

- **Metadaten hinzufügen**
  Das Anlegen neuer Metadaten ist eine Kernanforderung. Die Eingabe der Daten soll über eine Formularmaske erfolgen. Der Benutzer soll zur gleichen Zeit jedoch nur einen Datensatz eintragen können. Die Eingabe der Daten soll jedoch in einem Dialogfenster erfolgen.

- **Referenz hinzufügen**
  Beim Anlegen neuer Metadaten soll auch ein Referenzdatensatz angelegt werden können. Der Inhalt soll der „Metadaten hinzufügen“-Maske entsprechen. Die Eingabe der Daten soll jedoch in einem Dialogfenster erfolgen.

- **Metadaten bearbeiten**
  Der Benutzer darf bereits gespeicherte Metadaten bearbeiten. Die Bearbeitung eines Metadatensatzes soll aber in einer von „Metadaten hinzufügen“ verschiedenen Formularmaske erfolgen.

- **Metadaten suchen**
  Der Benutzer soll eine Suchmöglichkeit bekommen, in den gespeicherten Metadatensätzen nach ausgewählten Attributen zu suchen. Der Benutzer muss unter Umständen zwischen unterschiedlichen Suchmodis wechseln können. Die Eingabe der Daten soll über eine Formularmaske erfolgen.

- **Metadaten auflisten**
  Gesuchte Metadatensätze sollen zusammengefasst zu weiteren Verwaltung aufgelistet werden können. Die Verwaltung der Datensätze soll über eine Formularmaske erfolgen.

- **Inhalt der Zwischenablage anzeigen**
  Lokal abgelegte Datensätze sollten zu deren Verwaltung aufgelistet werden können. Die Verwaltung der Datensätze soll über eine Formularmaske erfolgen.

- **Kategorien hinzufügen**
  Bei der Auslieferung des Endprodukts wird eine vordefinierte Liste mitgeliefert. Diese kann jedoch mit weiteren Einträgen erweitert werden. Die Eingabe der Daten soll über eine Formularmaske erfolgen.

- **Kategorien auflisten**
  Der Benutzer soll eine Möglichkeit erhalten, die Kategorien auflisten und diese verwalten können. Die Eingabe der Daten soll über eine Formularmaske erfolgen.

- **Standort hinzufügen**
  Der Benutzer soll neue Standorte anlegen können. Die Eingabe der Daten soll über eine Formularmaske erfolgen.

- **Standort bearbeiten**
  Der Benutzer soll verfügbare Standorte auflisten und verwalten können. Die Eingabe der Daten soll über eine Formularmaske erfolgen.

- **Individuallisten hinzufügen**
  Der Benutzer soll neue Individuallisten anlegen können. Die Eingabe der Daten soll über eine Formularmaske erfolgen.

- **Individuallisten auflisten**
  Der Benutzer soll verfügbare Individualisten auflisten und verwalten können. Die Eingabe der Daten soll über eine Formularmaske erfolgen.

- **Verbindungen hinzufügen**
  Die Client-Anwendung muss sich zur Datensynchronisation und der netzumfassenden Suche mit einem verfügbaren Server verbinden können. Die Verbindungsdaten liefert der Benutzer. Die Eingabe der Daten soll über eine Formularmaske erfolgen.

- **Verbindungen auflisten**
  Der Benutzer soll verfügbare Verbidung auflisten und verwalten können. Die Eingabe der Daten soll über eine Formularmaske erfolgen.

- **Formularen-Editor**
  Der Benutzer muss für einzelne Medientypen[12] die aktiven Eingabefelder bestimmen können. Die Eingabe der Daten soll in einem Dialogfenster erfolgen.

- **Optionen/Einstellungen**
  Der Benutzer soll ausgewählte Eigenschaften des GUI-Erscheinungsbildes den eigenen Bedürfnissen anpassen können. Genaue Beschreibung der verfügbaren Schalter erfolgt im Handbuch. Die Eingabe der Daten soll in einem Dialogfenster erfolgen.

- **Hilfe**
  Dem Benutzer soll ein umfassendes Hilfesystem zur Verfügung stehen. Die Ausgabe der Daten soll in einem Dialogfenster erfolgen.

- **Info**
  Das Info-Bildschirm soll die wichtigsten Informationen über das Liabolo-Produkt darstellen. Die Ausgabe der Daten soll in einem Dialogfenster erfolgen.

1.3 Imlementierung

Entsprechend dem konzeptionell Entwurf wurde eine GUI implementiert, welche auf der MDI-Darstellungsarchitektur basiert und die JAVA Swing Grafikbibliothek nutzt.

Das Anwendungsfenster dient als ein Desktop, welcher mehrere parallel geöffnete Formularmasken darstellen kann und deren flexible Bearbeitung erlaubt. Hierzu wurden zwei, speziell für diesen Zweck, vordefinierte Swing-Klassen eingesetzt:
javax.swing.JDesktopPane.java und javax.swing.JInternalFrame.java. Die JDesktop-Klasse definiert den Anwedungsrahmen, die JInternalFrame-Klasse dient als ein Maskencontainer. Die Dialogmasken werden von der Klasse javax.swing.JDialog.java abgeleitet, welche die modale Darstellung mit sich bringt.

[12] siehe Handbuch S.XX

Der native „Look and Feel“ von Swing wurde durch den optisch ansprechenden Plastic3D-Look and Feel in der SkyKrupp-Theme von JGoodie.com[13] ausgetauscht. Die verwendeten Icons stammen überwiegend aus dem GNOME-Projekt[14]. Einige Icons wurden dagegen neu entworfen, wie z.B. für den Mediatypen-Editor. Somit wurde das Aussehen von Liabolo dem allgemeingültigem Standard weitgehend angepasst.

Nun wird kurz das Zusammenspiel der einzelnen Klassenpakete beschreiben und abschließend kurz die Aufgaben einzelner wichtigsten Klassen skizziert.

1.3.1 Klassenarchitektur

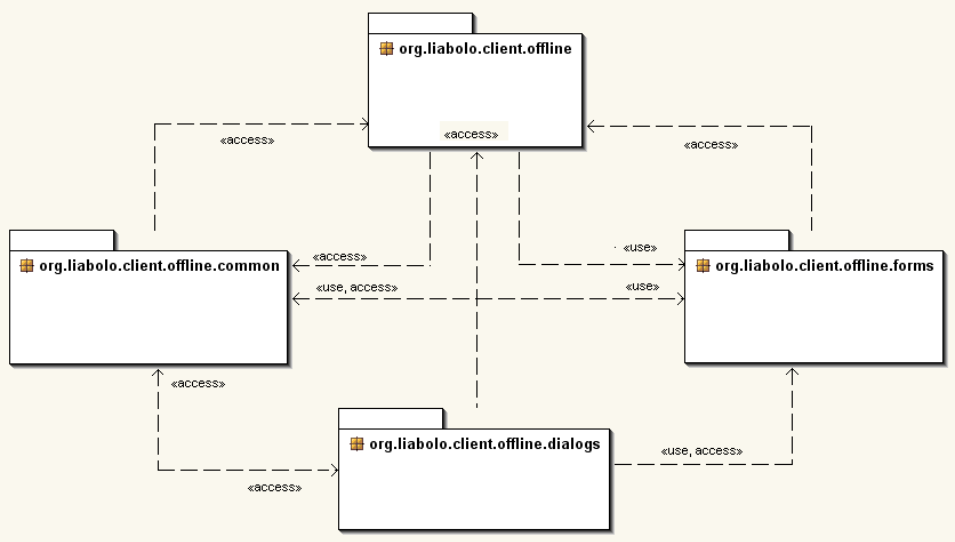

Abbildung 1.1: Package Diagramm

Das „offline“-Package enthält die Starterklassen, welche die GUI initialisieren und das Backend anbinden. Das Package „offline.forms“ enthält alle Klassen, die für die Formularmasken zuständig sind. Alle Dialogmasken befinden sich im „offline.dialogs“-Package. Das „offline.common“-Package enthält alle die so genannte Utility-Klassen. Hier werden z.B Schaltflächengeneratoren oder Dateifilter untergebracht. Alle benötigten Grafiken(u.a. die Icons) befinden sich im „offline.images“-Verzeichnis.

[13] http://www.jgoodies.com/

[14] Die ursprüngliche URL ist leider nicht mehr vorhanden. Die Icons lassen sich aber auch unter http://jimmac.musichall.cz/i.php3?ikony=43 betrachten

Die Maskenklassen werden erst nach den entsprechenden Benutzeraktionen instanziiert. Das bedeutet aber nicht, dass einzig die Klassen des „offline“-Package Formulare und Dialoge aufrufen können. Es kommt relativ häufig vor, dass ein Dialog eine Formularklasse instanziiert oder umgekehrt. Es existieren also keine definierten Schnittstellen für den Maskenaufruf. Auch der Zugriff auf die „offline.common“-Klassen erfolgt individuell. Die Kommunikation der GUI Klassen mit Backend erfolgt über eine Vermittlungsklasse des Backend(Dispatcher.java). Die GUI hat keine gesonderte Klasse, die Maskenanfragen kapselt und bevorzugt mit Backend kommuniziert.

1.3.2 Klassenbeschreibung

Es erfolgt nun eine etwas ausführlichere Beschreibung der Klassen einzelner Pakete. Für detaillierte Informationen zu jeder Klasse verweisen wir auf die vohandene JAVADOC-Dokumentation.

org.liablo.client.offline-Package

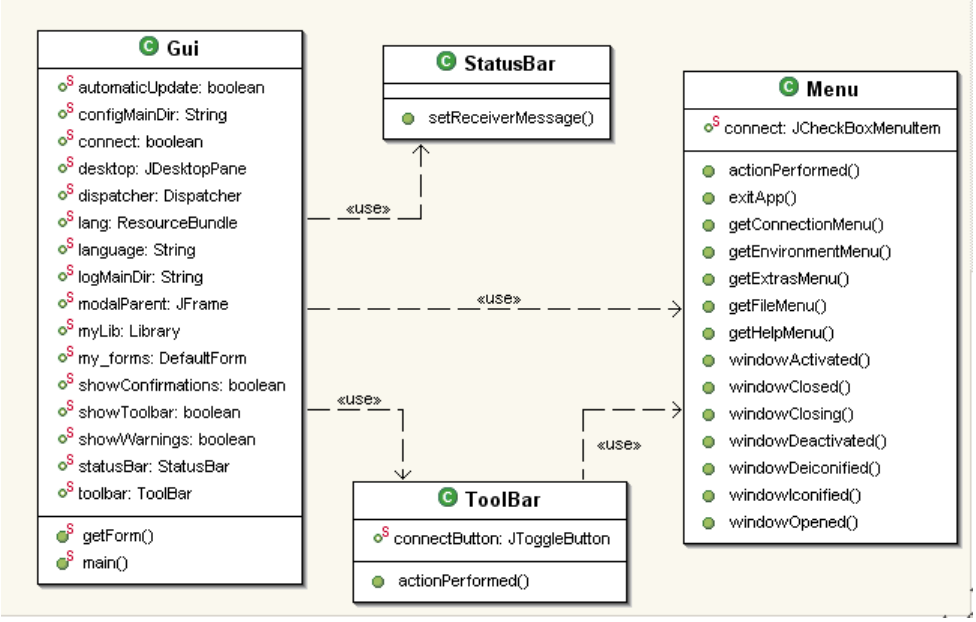

Abbildung 1.2: Klassen des org.liablo.client.offline-Package

- **org.liablo.client.offline.Gui.java**
  Die Liabolo-Anwendung wird von der Gui.java-Klasse aus initialisiert.

Die Klasse stellt das Anwendungsdesktop dar, überwacht geöffnete Masken und hält die Instanz einer Schnittstelle zur Kommunikation mit dem Backend bereit.

- **org.liablo.client.offline.Menu.java**
  Die Klasse erlaubt dem Benutzer einen schnellen Zugriff auf die meisten verfügbaren Funktionen. Auch die Benutzeraktionen aus der ToolBar.java-Klasse werden hier behandelt.

org.liablo.client.offline.forms-Package

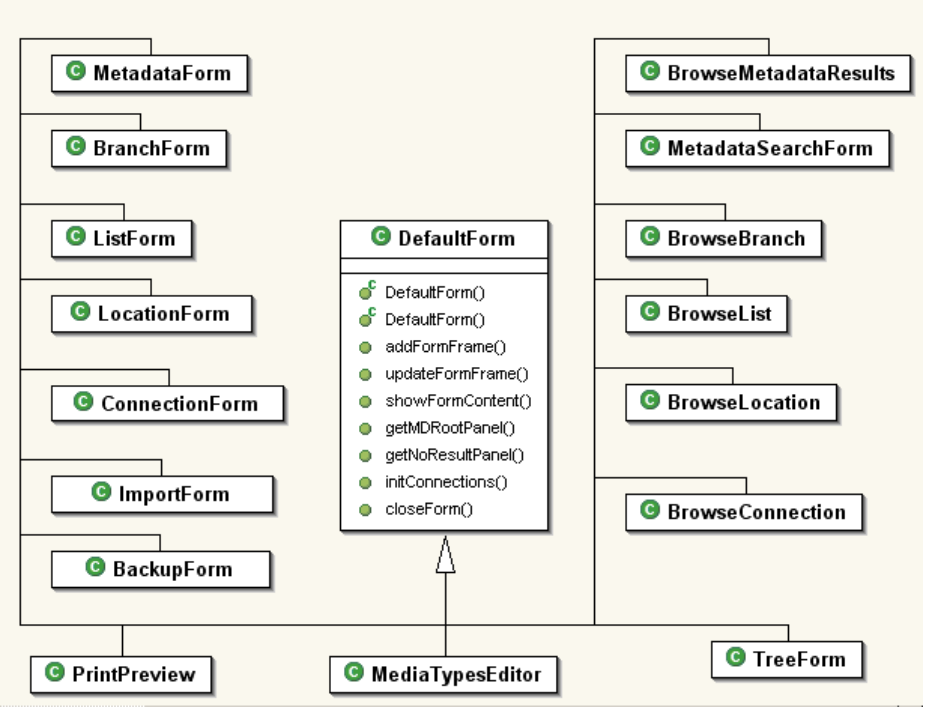

Abbildung 1.3: Klassen des org.liablo.client.offline.forms-Package

- **org.liablo.client.offline.forms.DefaultForm.java**
  DefaultForm ist die Oberklasse für alle weiteren Formularklassen. Sie implementiert einen einheitlichen Formularrahmen, welchen alle vererbenden Klassen mit spezifischen Inhalten füllen.

- **org.liablo.client.offline.forms.MetadataForm.java**
  Die Klasse implementiert einen „Neu/Bearbeiten“-Formular zur Erfassung und Bearbeitung von Metadaten.

- **org.liablo.client.offline.forms.BranchForm.java**
  Die Klasse implementiert einen „Neu/Bearbeiten"-Formular zur Erfassung und Bearbeitung von Fachbereichen.

- **org.liablo.client.offline.forms.ListForm.java**
  Die Klasse implementiert einen „Neu/Bearbeiten"-Formular zur Erfassung und Bearbeitung von Individuallisten.

- **org.liablo.client.offline.forms.LocationForm.java**
  Die Klasse implementiert einen „Neu/Bearbeiten"-Formular zur Erfassung und Bearbeitung von Standorten.

- **org.liablo.client.offline.forms.ConnectionForm.java**
  Die Klasse implementiert einen „Neu/Bearbeiten"-Formular zur Erfassung und Bearbeitung von Verbindungen.

- **org.liablo.client.offline.forms.BrowseMetatdataResuls.java**
  Die Klasse implementiert einen „Browse"-Formular zur Auflistung und Verwaltung von vorhanden bzw. gesuchten Metadatensätzen. Auch online gespeicherte(globale) Datensätze werden berücksichtigt.

- **org.liablo.client.offline.forms.BrowseBranch.java**
  Die Klasse implementiert einen „Browse"-Formular zur Auflistung und Verwaltung von lokal gespeicherten Fachbereichen.

- **org.liablo.client.offline.forms.BrowseList.java**
  Die Klasse implementiert einen „Browse"-Formular zur Auflistung und Verwaltung von lokal gespeicherten Individuallisten.

- **org.liablo.client.offline.forms.BrowseLocation.java**
  Die Klasse implementiert einen „Browse"-Formular zur Auflistung und Verwaltung von lokal gespeicherten Standorten.

- **org.liablo.client.offline.forms.BrowseLocation.java**
  Die Klasse implementiert einen „Browse"-Formular zur Auflistung und Verwaltung von lokal gespeicherten Verbindungen.

- **org.liablo.client.offline.forms.TreeForm.java**
  Die Klasse implementiert eine Maske zur Anzeige der aktuell gespeicherten Bibliothekstrukturen. Sie implementiert auch eine Suchfunktionalität, welche sich allerdings ledigleich auf die Ausgabe der Inhalte von angeklickten Knoten beschränkt.

- **org.liablo.client.offline.forms.SearchForm.java**
  Die Klasse implementiert einen „Suche“-Formular zur einer vielseitigen Suche auf der lokalen Datenbank, aber auch auf den globalen Datenbeständen.

- **org.liablo.client.offline.forms.ImportForm.java**
  Die Klasse implementiert eine Maske zum Import von Excel-Daten. Sie erlaubt u.a gezieltes Mapping von Spalten.

- **org.liablo.client.offline.forms.PrintPreview.java**
  Die Klasse implementiert eine Maske für das Druckvorschau der ausgewählten Metadatensätze.

- **org.liablo.client.offline.forms.MediaTypeEditor.java**
  Die Klasse implementiert eine Maske zur Bearbeitung von vordefinierten Medientypen. Der Benutzer kann bestimmen, welche Felder zur Datenerfassung angezeigt und welche ausgeblendet werden sollen.

org.liablo.client.offline.dialogs-Package

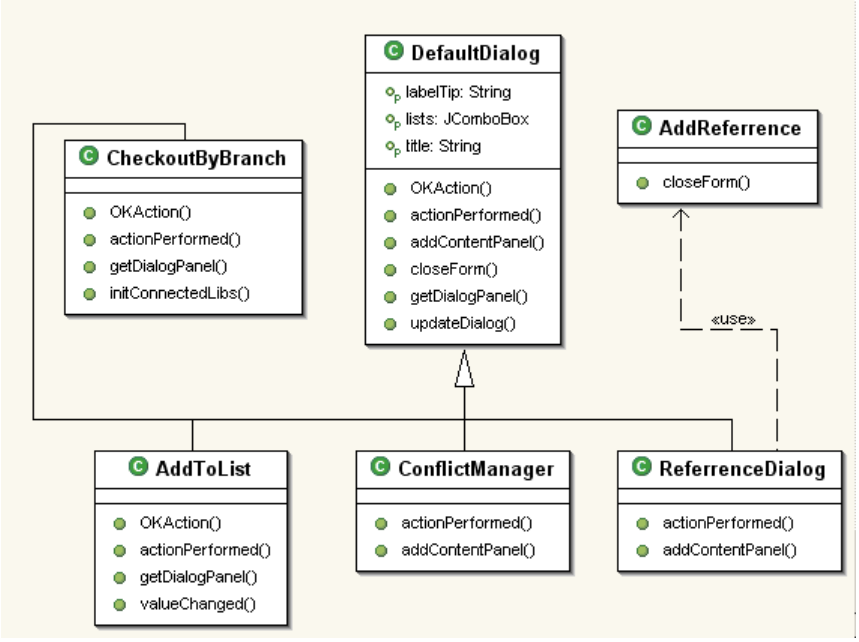

Abbildung 1.4: Klassen des org.liablo.client.offline.dialogs-Package

- **org.liablo.client.offline.forms.DefaultDialog.java**
  DefaultDialog ist die Oberklasse für alle weiteren Dialogklassen. Sie implementiert ein einheitliches Dialogfenster, welches alle vererbenden Klassen mit spezifischen Inhalten füllen. Alle Dialogmasken werden modal(und relativ zum Liabolo-Desktop) dargestellt.

- **org.liablo.client.offline.forms.AddReferrence.java**
  Die Klasse implementiert eine Dialogmaske zur Erfassung von Referenz-Metadaten. Die Signatur des hiermit abgelegten Datensatzes wird in das ebenfalls geöffnete Formularfenster zur Erfassung von neuen Metadaten übertragen. Somit wird von einem Datensatz auf einen anderen referenziert.

- **org.liablo.client.offline.forms.ReferrenceDialog.java**
  Die Klasse implementiert eine Dialogmaske zur Auswahl der möglichen Optionen bei der Zuweisung eines Rerefenzdatensatzes einem anderen Metadatensatz. Eine Option führt zur Ausführung von org.liablo.client.offline.forms.AddReferrence.java

- **org.liablo.client.offline.forms.PreferrencesDialog.java**
  Die Klasse implementiert eine Dialogmaske zur Darstellung und Bearbeitung der allgemeinen Einstellungen der Anwendung.

- **org.liablo.client.offline.forms.CheckoutByBranch.java**
  Die Klasse implementiert eine Dialogmaske zur Auswahl der gewünschten Verbindung(globalen Datenbank) für einen Checkout gewünschter Datensätze. Alle, in den zuvor markierten Fachbereichen enthaltene Datensätze werden somit in dem lokalen Repository(auch Zwischenablage genannt) abgelegt.

- **org.liablo.client.offline.forms.AddToList.java**
  Die Klasse implementiert eine Dialogmaske zur Auswahl von einer oder mehreren Individuallisten, denen die zuvor markierten Metadatensätze zugewiesen werden sollen.

- **org.liablo.client.offline.forms.AddToList.java**
  Die Klasse implementiert eine Dialogmaske zur Behandlung von Konflikten, die bei einem Datenabgleich(Commit) mit einer globalen Datenbank aufgetreten sind.

- **org.liablo.client.offline.forms.Info.java**
  Die Klasse implementiert eine Dialogmaske zur Anzeige der allgemeinen Informationen über das Liabolo-Projekt.

org.liablo.client.offline.common-Package

Die Klassen des common-Package werden in ausführlicher Form auf den JAVADOC-Seiten beschrieben.

Literaturverzeichnis

[1] Objektorientierte Softwareentwicklung Oestereich, B., R. Oldenburg Verlag, 1998

[2] Lehrbuch der Software-Technik, Balzert, H., Spektrum Akademischer Verlag, 2000

[3] Entwurf graphischer Benutzungsschnittstellen, Ziegler, J., Oldenburg, 1993

[4] Graphic Design for Electronic Documents and User Interfaces, Marcus, A., ACM, 1992

[5] Designing the User Interface. Strategies for Effective Human Computer Interaktion, Shneiderman B., Addison-Wesley, 1992

[6] Designing the User Interface. Strategies for Effective Human Computer Interaktion, Shneiderman B., Addison-Wesley, 1992

[7] Software Engineering Glossary, Ian Somerville, `http://www.comp.lancs.ac.uk/computing/resources/IanS/SE6/PDF/SEGlossary.pdf`, 2003

[8] Wikipedia, Wikipedia. Die freie Enzyklopädie, `http://de.wikipedia.org/wiki/`,

[9] GoTo Java 2, Guido Krüger, Addison Wesley, 2001

Index