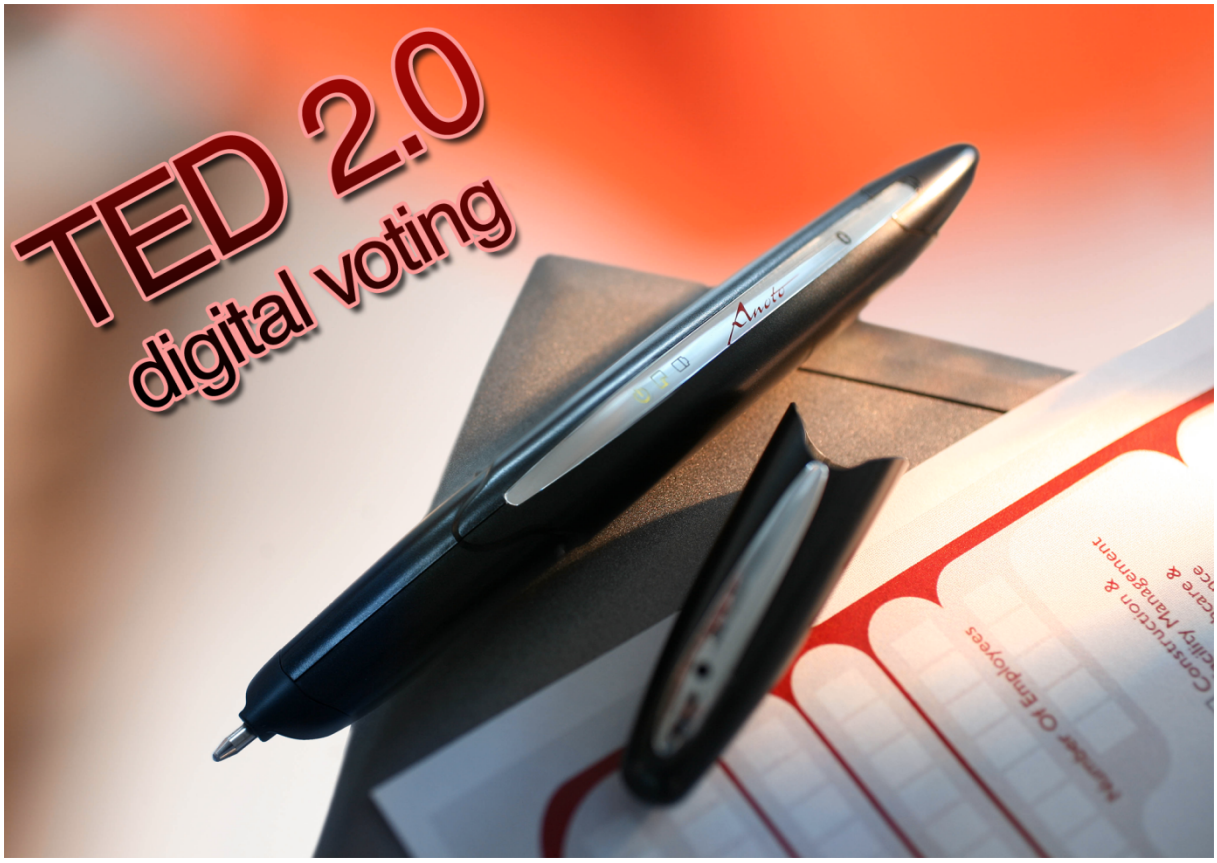


Funktionsbeschreibung 2
Anoto Technologie *2*
Einschränkungen *2*
Architektur 3
Editor *5*
Fetcher/Parser *5*
MailFetcher *5*
Config.xml *5*
Flowchart *6*
Directory Poller *6*
Flowchart *6*
Visualizer *7*
Vorgehen 8
Verwendete Tools *8*
Technische Planung *9*
Probleme *9*
Calibrator *11*

Funktionsbeschreibung

Die Benutzer erhalten die Möglichkeit, Fragebögen zu erstellen. Diese Fragebögen können auf Spezialpapier ausgedruckt werden und anschließend mit Anoto-Stiften ausgefüllt werden.

Die Auswertung der Ergebnisse geschieht automatisiert über die erstellte Software und wird grafisch aufbereitet und dargestellt.

Dieses Projekt entstand in einer Zusammenarbeit mit dem Fraunhofer Institut. Dieses stellten uns folgende Anforderungen:

- Erstellung von Fragebögen
- Auswertung der von den Anoto Stifen gelieferten Daten
- Visualisierung der Umfrage-Ergebnisse

Anoto Technologie

Wir setzen digitale Stifte von Logitech ein (IO2), welche die patentierte NanoDot Technologie von Anoto verwenden.

Diese Stifte wurden uns freundlicherweise vom Fraunhofer Institut zur Verfügung gestellt.

Die Stifte bestehen grob gesagt aus einer Speichereinheit und einer Kamera, welche vorne an der Schreibmine angebracht ist.

Wird mit dem Stift auf speziell bedrucktes Papier geschrieben, speichert der Stift die Position der Striche digital ab. Diese Daten können via Bluetooth oder USB auf einen Rechner übertragen werden.

Einschränkungen

Anoto verkauft ein SDK, welches es ermöglicht auf Basis dieser Technologie eigenes Papier & Funktionen zu entwickeln.

Leider stand uns aus Kostengründen dieses SDK nicht zur Verfügung. Die Hauptmotivation bei der Umsetzung bestand also darin, trotz massiver Einschränkungen das Ziel zu erreichen.

Diese Einschränkungen betreffen unter anderem:

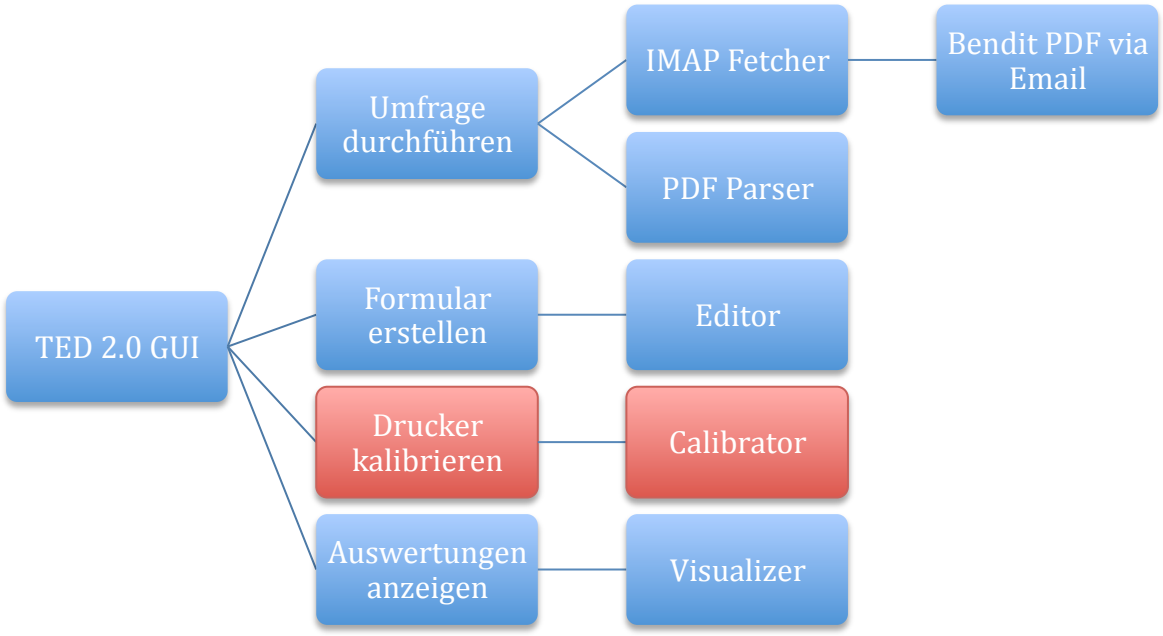
- Kein Zugriff auf die Bluetooth Funktionalität des Stiftes (nur USB möglich).
- Kein direktes Auslesen der Schreibdaten aus dem Stift möglich.

Die Lösung für dieses Problem besteht daraus, dass wir den Service einen Dienstleisters für Anoto-Produkte in Anspruch nehmen (Bendit). Dieser stellt uns Software für den

Auslesevorgang zur Verfügung und übergibt die Daten anschließend als PDF Datei, welche via Email zugestellt wird.

Architektur

Aufgrund der o.g. Einschränkungen ergibt sich folgende Architektur der erstellten Software:

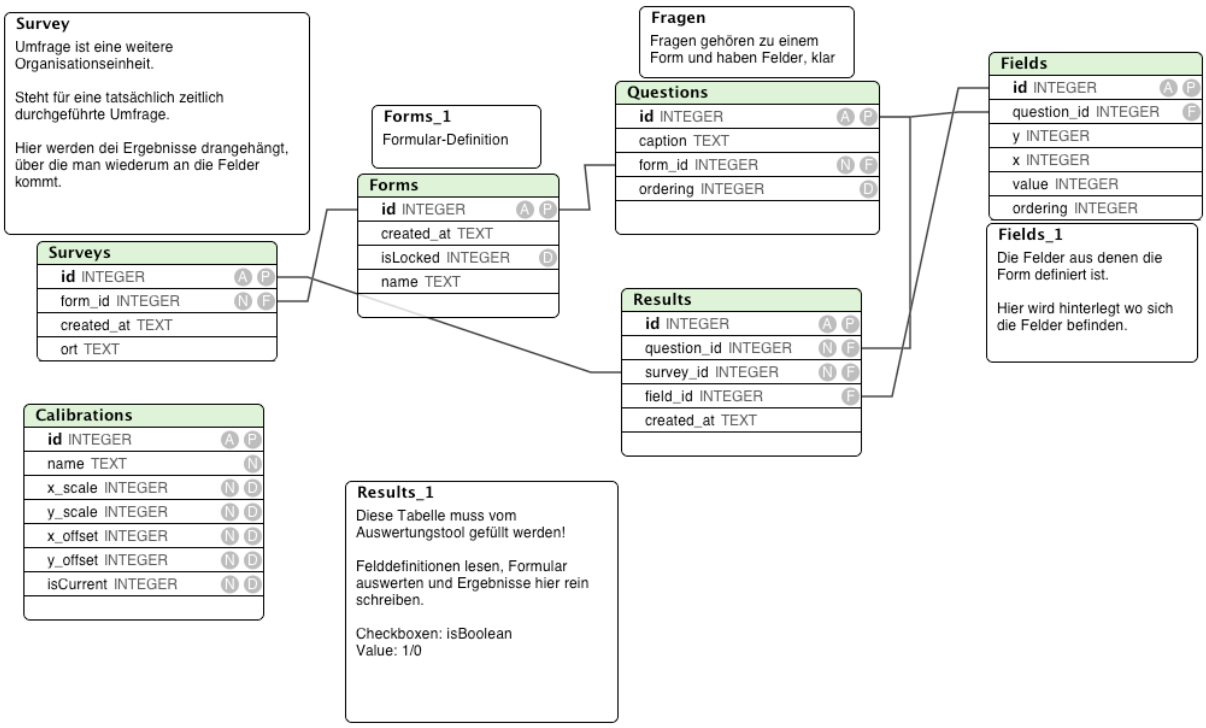


Der rot markierte Bereich wurde später wieder aus dem Programm entfernt. Eine detaillierte Beschreibung ist im Kapitel „Probleme“ zu finden.

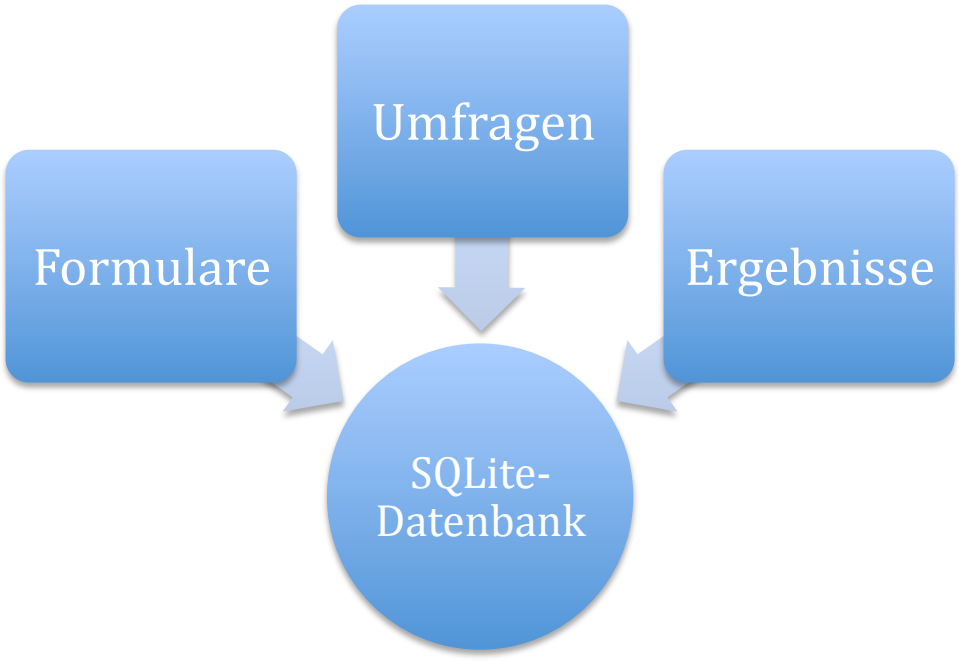
Datenbank

Wir stellten relativ schnell fest, dass der hohe Umfang der zu verarbeitenden Daten eine klare Struktur braucht. Daher entschlossen wir uns, alle Daten in einer zentralen SQLite Datenbank abzulegen.

Es wurde folgende Struktur verwendet:



Die Verwendung von SQLite als Datenbanktechnologie brachte uns den Vorteil, dass auf dem Rechner des Benutzers keine zusätzliche Datenbanksoftware installiert oder mitgeliefert werden musste.



Editor

Der Editor erlaubt das erstellen und bearbeiten von Fragebogen-Vorlagen. Aus den erstellten Vorlagen können beliebig viele Umfragen erstellt werden, zu welchen die Ergebnisse abgespeichert werden.
Fragen können in beliebiger Reihenfolge angelegt werden.

Fetcher/Parser

Die Aufgabe des Parsers ist es bei einer Live-Umfrage die Ergebnisse für den Visualizer zu erzeugen. Der Parser unterteilt sich grob in den MailFetcher und den DirectoryPoller. Diese werden bei einer Live Umfrage vom Editor beide gleichzeitig gestartet und laufen als Threads in einer Endlosschleife, bis der Editor sie durch einen Interrupt beendet.

MailFetcher

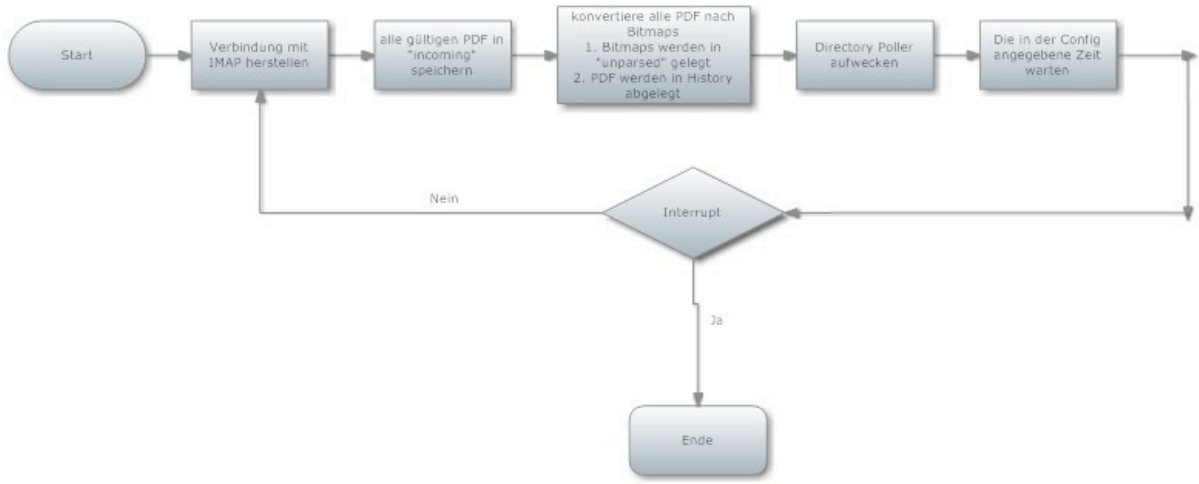
Der **MailFetcher** verbindet sich zu dem in der Config.xml angegeben IMAP Account und holt dort alle PDFs, wobei geprüft wird ob die Daten von einem gültigen Stift kommen. Alle Dateien werden in Unterordner des Ordners „files“ im Hauptverzeichnis des Programms gespeichert. Die PDF werden in den Ordner „incoming“ abgelegt und danach vom IMAP gelöscht. Falls etwas vom IMAP geladen wurde, werden anschließend alle PDF in „Incoming“ mit Pdf2Bmp in Bitmaps konvertiert. Die Bitmaps werden dann in den Ordner „unparsed“ gelegt und das konvertierte PDF in “history/Jahr/Tag/Uhrzeit“ abgelegt. Falls ein PDF mehrere Seiten hat, so wird jede Seite als einzelnes Bitmap in „unparsed“ gespeichert. Danach wird der DirectoryPoller, welcher sich zu beginn schlafen gelegt hat, aufgeweckt. Nun wartet der MailFetcher die in der Config.xml angegebene Zeit und startet danach einen weiteren Durchlauf.

Config.xml

Die im Verzeichnis „./config/config.xml“ befindliche Config, erlaubt es gewisse Einstellungen als Benutzer vorzunehmen, ohne den Programmcode neu kompilieren zu müssen.
Die Config wird, soweit nicht bereits vorhanden, aus den Parametern der Klasse MailFetcherConfig.java erzeugt. Danach können diverse Einstellungen vorgenommen werden.

- allowedPenIds: ID der gültigen Stifte hier eintragen
- host,username,password: hier die Verbindungsdaten für den IMAP eintragen
- fetchingIntervalSeconds: Intervall, in welchem Abstand der IMAP gepolt wird
- incomingFolder: Ordner, in dem die PDF vom IMAP abgelegt werden
- unparsedFolder: Ordner, in dem die konvertierten Bitmaps abgelegt werden
- historyFolder: Ordner, in dem ausgewertete PDF abgelegt werden

Flowchart

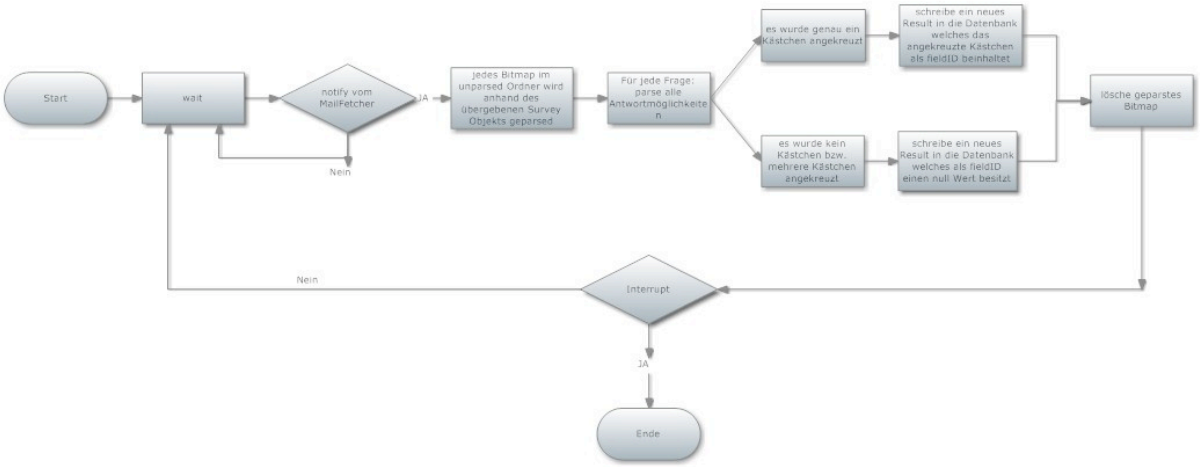


Directory Poller

Der **DirectoryPoller** wird zu Beginn schlafen gelegt. Nachdem der MailFetcher ihn aufweckt beginnt er eine Iteration. Er analysiert nun jedes Bitmap im „unparsed“ Ordner. Anhand der survey, welche dem DirectoryPoller als Parameter übergeben wird, wird nun jedes Field einer Frage anhand der Koordinaten, welche in der Datenbank liegen geprüft ob es angekreuzt ist oder nicht. Ein Feld kann auch wieder durchgestrichen sein.

Wurde genau ein Field einer Frage angekreuzt so wird ein neues Result in die Datenbank geschrieben, welches das angekreuzte Field als Wert bei „Field“ beinhaltet. Sollte bei einer Frage kein Field angekreuzt sein oder gar mehrere, so wird ein Result erzeugt, welches für den Wert bei „Field“ null beinhaltet. Dies stellt dann eine ungültige Stimme dar. Danach wird das Bitmap gelöscht.

Flowchart



Visualizer

Der Visualizer realisiert die Darstellung der Ergebnisse der Umfrage durch jFreeChart. Er bedient sich hierbei der Results, welche vom Parser erzeugt wurden und kann in zwei Modi betrieben werden. Ihm wird dabei ein boolscher Wert übergeben, wodurch er erkennen kann in welchem Modus er sich befindet. Ihm wird auch das aktuelle survey-Objekt übergeben, womit die Charts erzeugt werden können.

1. **Live-Modus**

   Wird eine Umfrage gestartet so erscheint ein neues Fenster. Nun kann durch einen Klick auf „Auswertung betrachten“ der Live-Mode des Visualizer gestartet werden. Es wird nun einmalig pro Frage ein Chart erzeugt und dieses anschließend in ein Panel eingefügt. Danach wird ein Swing-Timer gestartet, welcher jede Sekunde eine refresh Methode aufruft. Diese Methode holt sich alle zur Frage gehörenden Results. Falls neue Results vorhanden sind, so werden alle Charts aktualisiert.

2. **Normaler Modus**

   Wird im Editor eine alte Umfrage geladen(Auswerten), so läuft der gleiche Prozess wie beim Live-Modus ab, nur dass der MailFetcher, DirectoryPoller und der Swing-Timer nicht gestartet werden. Es werden nur die zur survey gehörigen Results einmalig aus der Datenbank ausgelesen und daraus die Charts erzeugt.

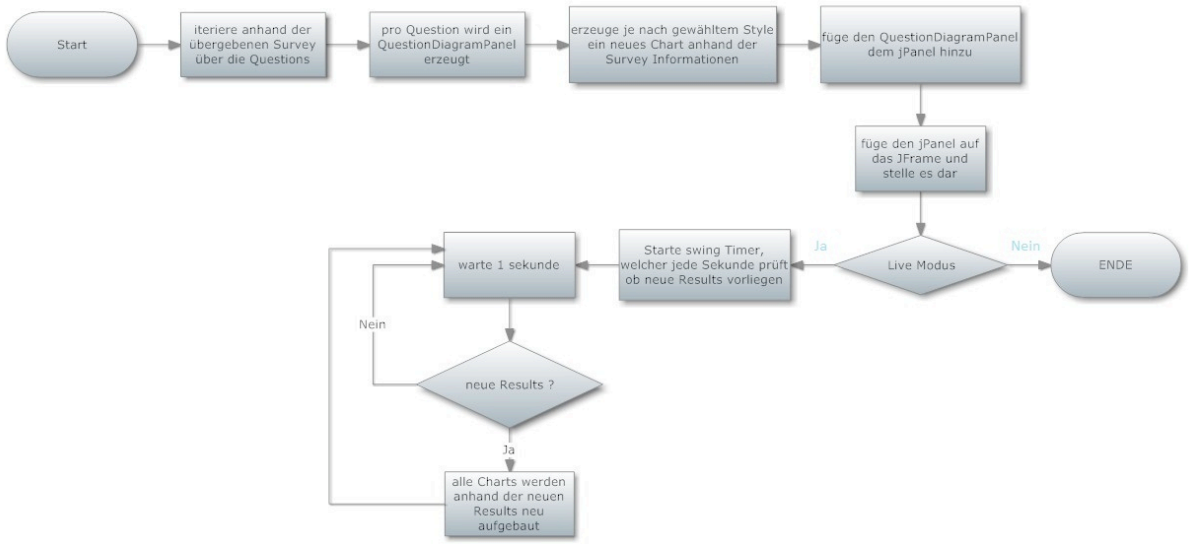
Technisches

Pro Chart gibt es ein QuestionDiagramPanel, welches das Chart als Referenz hält, eine Methode zur Aktualisierung des Charts beinhaltet(setStyle) und eine Methode zur Erzeugung eines Charts(getDiagramm), welche abhängig vom gewählten Style das Chart erzeugt.

Es gibt eine abstrakte Superklasse „AbstractQuestionDiagram“, welche als Generalisierung der unterschiedlichen Chartklassen(Bar,Pie,Ring …) dient. Pro Chart-Style ist eine eigene Klasse definiert, da sie alle unterschiedliche Vorgehensweißen zur Erzeugung ihres Spezifischen Charts benötigen.

Somit ist es einfach möglich die Styles zu ändern. Es wir im Panel einfach der aktuelle Style in der Variablen „style“ gehalten. Beim Aktualisieren eines Charts wird nun nur geprüft, welcher Style ausgewählt wurde und dementsprechend das Chart erzeugt.

Flowchart



Vorgehen

Verwendete Tools

Programmiersprache

Da das entwickelte Programm weitestgehend plattformunabhängig sein soll, entschieden wir uns für die Verwendung von Java als Programmiersprache.

Entwicklungsplattform

Als Entwicklungsplattform wurde Netbeans 6.7/6.8 verwendet. Wir entschieden uns für Netbeans, da ein starker GUI-Editor und eine sehr gute Integration von Subversion vorhanden sind.

Versionsverwaltung

Zur kontrollierten Verwaltung des Source-Codes wurde Subversion verwendet.

Projektmanagment

Redmine, bereitgestellt von der HdM.

Datenbank

Ein wichtiges Kriterium für den Einsatz einer Datenbank war, dass der Programmanwender keinen Datenbank-Server laufen lassen muss, wenn er das Programm verwendet.

Deshalb entschieden wir uns für SQLite als Datenbanktechnologie, da diese dateibasiert ist.

Des weiteren wurde Hibernate als ORM-Mapper verwendet. Wir verwendeten eine modifizierte Version, welche auch mit SQLite benutzt werden kann.

Tests

Insbesondere für die Hibernate-Model-Klassen wurden von uns intensiv Tests geschrieben um uns vor eventuellen Überraschungen zu schützen.

Für die Tests wurde JUnit verwendet. Es wurde von uns ein Testframework geschaffen, welche eine einheitliche Datenbasis für alle durchgeführten Tests zur Verfügung stellt.

Technische Planung

Arbeitsteilung

Das Projekt wurde nach gründlicher Analyse-Phase und Aufbau der Datenbank in 2 Bereiche aufgeteilt.

Für den Formulareditor war Patrick Dinger verantwortlich und für den Parser & Visualizer Benjamin Heuer.

Aus Sicherheitsgründen haben wir die 3 Bereiche zu Beginn in getrennten Netbeans-Projekten entwickelt. Nach einem Gewissen Grad der Fertigstellung wurden die 3 Teile in ein gemeinsames Projekt zusammengefügt. Während der Arbeit wurde ständig Kontakt gehalten und die Zwischenergebnisse in regelmäßigen Meetings besprochen.

Zusammenführung

Nach dem die einzelnen Komponenten fertig entwickelt waren, wurden Schnittstellen für die interne Kommunikation entworfen und von den einzelnen Komponenten umgesetzt. Dies ermöglichte uns eine weitestgehende problemlose Vereinigung aller Komponenten zu einem Hauptprojekt.

Probleme

GUI Entwicklung

Als großes Problem stellte sich die Umsetzung von Drag' and Drop Funktionalität in Java dar. Das Ziel war es, dass der Benutzer mit der Maus die Reihenfolge der Fragen und Antworten leicht ändern kann.

Neben der komplizierten Umsetzung von DnD im Interface stellte sich ebenfalls heraus, dass eine sortierte Speicherung in der Datenbank nur über Umwege möglich ist.

Als Lösung wurde in der Datenbank ein neues Feld für die Sortierung angelegt und die Hibernate-Modelklassen um die Sortiermöglichkeit erweitert. Die Modelle, welche Swing intern für die GUI-Elemente benutzt, mussten ebenfalls extrem angepasst werden um die korrekten Sortierdaten in die Datenbank schreiben zu können.

Skalierungs- und Verschiebefaktor ermitteln

Das Prinzip des Parsens der Formulare beruht darauf, dass die Koordinaten der Checkboxen in der Datenbank gehalten werden.

Der Parser prüft nun anhand dieser Koordinaten, ob an dieser Stelle die Checkbox angekreuzt wurde. Beim drucken & konvertieren der Stiftdaten bei Bendit entsteht jedoch ein gewisser Versatz und ein Skalierungseffekt tritt ein. Hier mussten wir durch

systematisches Testen diese beiden Faktoren ermitteln und als fixe Konstanten in das Programm eintragen.

Beim lesen des Bildes wird dieses anhand des Skalierungsfaktors skaliert und die Koordinaten aus der Datenbank um den eingetragenen Versatz verschoben.

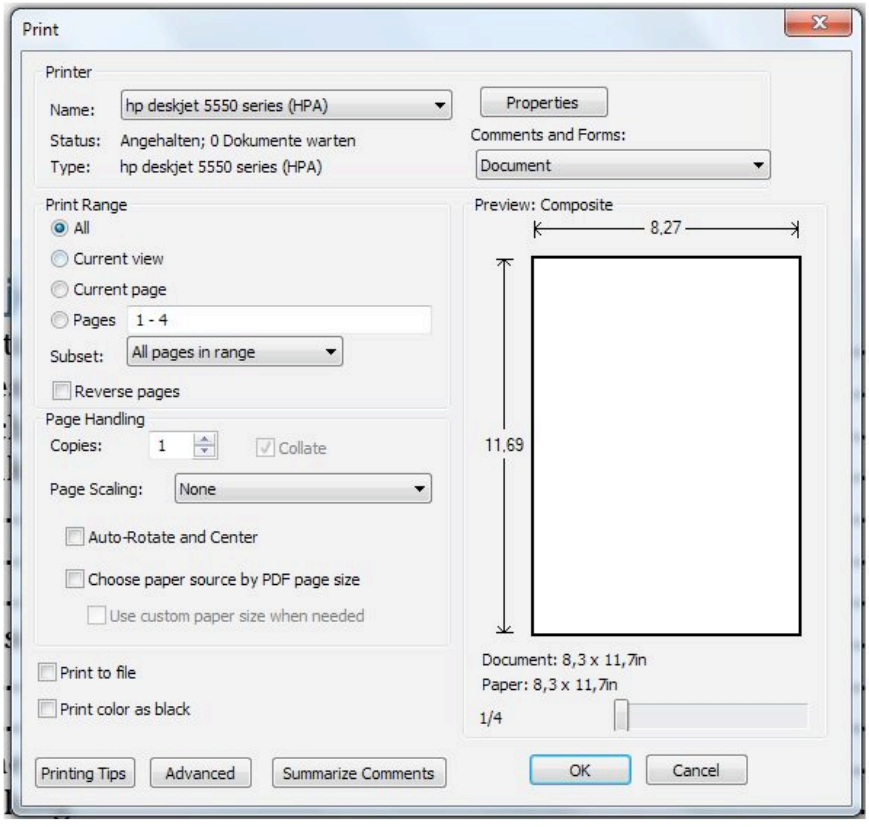
Drucker-Kalibrierung

Dieses Thema hat uns sehr viel Zeit gekostet. Es ist sehr wichtig, dass beim Drucken der Umfragebögen keinerlei Optimierungsoptionen beim Druck ausgewählt werden. Wir hatten unsere Testbögen zu Beginn jeweils mit der Auswahl „in Druckbereich einpassen“ gedruckt.

Bei dieser Option tritt jedoch das Phänomen auf, dass die Optimierungen auf jedem Drucker anders vorgenommen werden. Dies führte zu der Fehlannahme, dass keine Möglichkeit besteht einen einheitlichen Druckversatz auf allen Druckern zu erreichen.

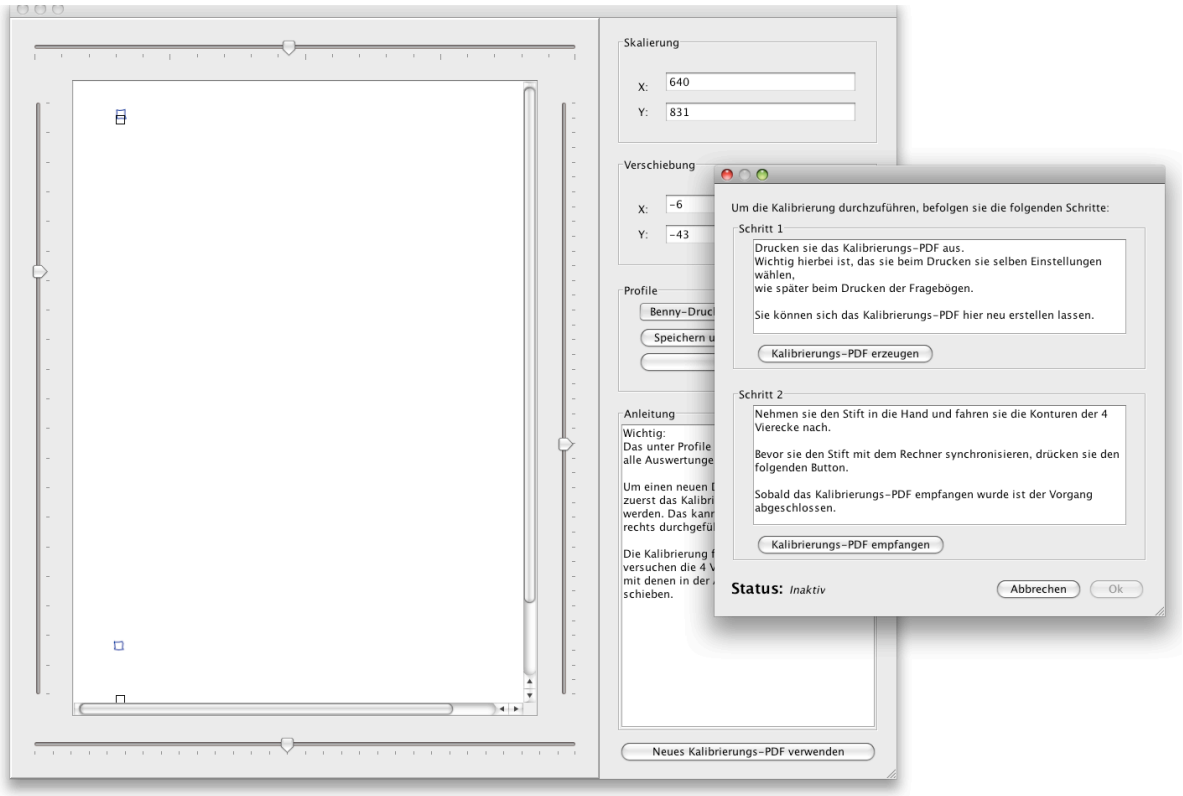
Aus diesem Grund wurde von uns ein Tool entwickelt, welches es ermöglichen sollte das Programm speziell auf die genutzten Drucker zu kalibrieren: Den Calibrator (siehe nächstes Kapitel).

Die Umfragebögen sollten daher immer mit folgenden Einstellungen gedruckt werden:



Wichtig ist, dass bei „Page Scaling“ „none“ ausgewählt wird. Dadurch führt der Drucker keine eigene Größenanpassung des Druckes durch.

Calibrator



Um das Problem mit den unterschiedlichen Versätzen der Drucker zu lösen, wurde von uns ein Kalibrierungstool entwickelt.

Leider stellte sich am Ende der Entwicklung heraus, dass eine Korrektur der Druckeinstellungen die Kalibrierung unnötig macht.

Das Prinzip des Calibrators war das folgende:

- Dem Benutzer wurde eine spezielle Kalibrierungs-PDF Datei erstellt. Diese musste auf Spezialpapier gedruckt werden. Das PDF bestand aus vier Quadraten, welche in den Ecken angeordnet waren.
- Anschließend mussten diese Quadrate ausgefüllt werden.
- Das Programm lud anschließend die Daten vom Stift und stellte sie grafisch dar.
- Die Aufgabe des Benutzers war es dann, die 4 selbst gezeichneten Quadrate mit den rot eingezeichneten Vorgaben zu vergleichen und diese mit Hilfe der Skalierungs- und Verschieberegler übereinander zu legen.
- Sobald dies geschafft war, waren die individuellen Kalibrierungsversätze ermittelt und konnten als Profil abgespeichert werden. Das aktuell ausgewählte Profil wurde anschließend für die Verarbeitung der Umfragedaten verwendet.