relaKs
Entwicklungsdokumentation

Daniel Grün

7. Juli 2005

Zusammenfassung

relaKs ist eine Datenbankanwendung für Linux, die *Soundscapes*, also künstliche Klanglandschaften, erzeugt. Diese Dokumentation gibt einen Einblick in die Struktur des Programmes und insbesondere der dazugehörigen Datenbank.

Inhaltsverzeichnis

1 Idee

Aus einzelnen Klängen soll ein Abbild natürlicher Klanglandschaften, so genannte *Themes* geschaffen werden. Dabei gibt es Klänge (z.B. Meeresrauschen

etc.), die als Hintergrundklang fungieren und andere (z.B. Vogelruf), die als Ereignis zufallsgesteuert zusätzlich abgespielt werden sollen.

Das Prinzis des Programmes ist angelehnt an Windows-Programme wie *Aire Freshener* [1], geht aber durch den Aufbau aus Hintergrund- und Ereignisklängen über bereits vorhandene Lösungen hinaus und ist zudem das vermutlich erste Programm dieser Art für die Linux-Plattform.

relaKs bindet sich dabei nahtlos in KDE [5] ein, weshalb auch der Name an die KDE-Konventionen angepasst ist.

2 Warum eine Datenbank?

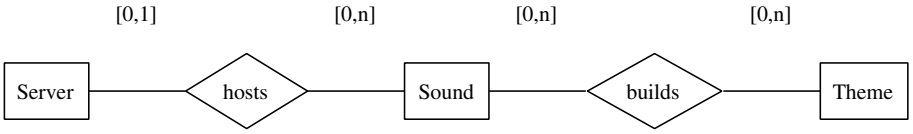
Die Klänge und Themes sollen nicht mit dem Programmpaket von relaKs ausgeliefert werden, sondern vielmehr mittels einer Datenbank und einem Internetzugang für jeden Nutzer individuell zusammenstellbar und downloadbar sein. Vorteile hiervon sind unter anderem:

- **geringeres Downloadvolumen** - das Programmpaket von relaKs enthält keine großen WAV-Dateien, die der Nutzer u.U. im Einzelnen überhaupt nich benötigt
- **Copyrightproblematik** - relaKs ist Freie Software und wird auf dementsprechenden Seiten zum Download angeboten[4]; Klänge können anders lizenziert sein (z.B. frei verfügbar für nicht-kommerzielle Nutzung) und sich trotzdem für die Benutzung mit relaKs eignen, aber nicht mit den Bestimmungen der Webhosts für Freie Software vereinbar sein, so dass sie nicht mit relaKs zusammen ausgeliefert werden können
- **Dynamik** - wenn neue Sounds gefunden und neue Themes hinzugefügt werden, sollen diese schnell allen Nutzern von relaKs zur Verfügung stehen; zudem ist die Mitarbeit bei der Erstellung von Themes erwünscht

Die Vorteile eines Datenbanksystems zur Speicherung der Theme- und Soundeigenschaften rechtfertigen also den Mehraufwand.

3 Planung der Datenbank

Zunächst war es nötig, ein ER-Diagramm der geplanten Datenbank zu erstellen. Dabei stellen *Server* jeweils einen oder mehrere *Sounds* bereit, die in verschiedenen *Themes* kombiniert werden. Ein möglichst einfacher Entwurf sieht also etwa so aus:



Nach Abbildung des ER-Modelles sind also vier Tabellen notwendig, wenn man die `[m:n]`-Relation als `builds` miteinbezieht. Im Folgenden sind diese mit ihren Attributen beschrieben:

1. `Server`

- `SID` [integer] - eindeutiger Bezeichner; Primärschlüssel
- `SURL` [string] - Basisadresse des Servers
- `SContact` [string] - e-Mail-Adresse des Betreibers
- `SPermissionGranted` [boolean] - hat der Betreiber seine ausdrückliche Genehmigung gegeben?
- ...

2. `Sound`

   - `SID` [integer] - eindeutiger Bezeichner; Primärschlüssel
   - `SFilename` [string] - lokaler Dateiname, der für jeden Sound aufgrund der Speicherung in einem Verzeichnis unterschiedlich sein muss
   - `Server` [integer] - SID des Servers; Fremdschlüssel
   - `SRelativeURL` [string] - Pfad vom Basisverzeichnis des Servers aus
   - `SFormat` [enumeration] - Kürzel des Dateiformates (momentan WAV oder MP3)
   - `SLicense` [enumeration] - Lizenz zur Nutzung (GPL, Public Domain, ...)
   - `SHomepage` [string] - Seite innerhalb des Servers, von der aus die Datei verlinkt ist
   - ...

3. `builds`

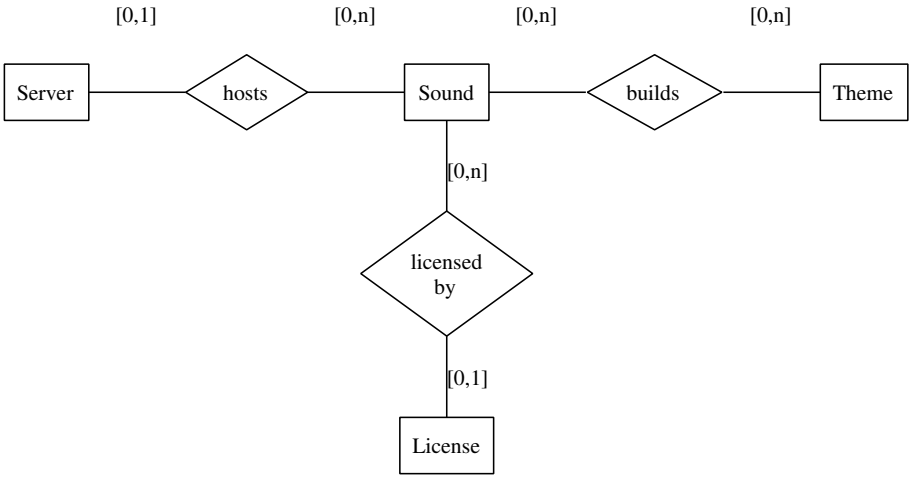
   - `SID` [int] - SID des Sounds; Fremdschlüssel; bildet mit `TID` kombinierten Primärschlüssel
   - `TID` [int] - TID des Themes; Fremdschlüssel; bildet mit `SID` kombinierten Primärschlüssel
   - `bVolume` [float] - Lautstärke (0.0-1.0)
   - `bStartOffset & bEndOffset` [float] - Zeit, die beim Abspielen am Anfang bzw. am Ende übersprungen wird
   - `bBackground` [boolean] - Ist der Klang ein Hintergrund-Klang (in endloser Wiederholung gespielt)?
   - `bFrequency` [float] - Häufigkeit eines Ereignis-Klanges
   - ...

4. `Theme`

   - `TID` [integer] - eindeutiger Bezeichner; Primärschlüssel
   - `TName` [string] - Titel des Themes
   - `TAuthor` [string] - Autor des Themes
   - `TEventBuffer` [float] - Wert in Sekunden, der die Variationsbreite der Ereignisdichte angibt (je höher, desto seltener treten Klangereignisse auf)
   - ...

Anmerkungen:

- Die als `boolean` bezeichneten Datentypen sind als Enumerationen implementiert, wie in Abschnitt 3.1 beschrieben.
- Die SID des Servers wird innerhalb der Tabelle `Sound` als Fremdschlüssel als `Server` bezeichnet, da die Bezeichnung `SID` hier schon vergeben ist.
- Der Entwurf der Datenbank erwies sich als unzureichend im Bezug auf die Lizenzeigenschaften. So muss berücksichtigt werden, dass bei manchen Lizenzierungen es nötig ist, beim Download einer Datei die Homepage anzuzeogen, von der aus sie bereitgestellt wird (sogenanntes Verbot des *deep linking*). Bei anderen Lizenzen ist dies nicht notwendig. Es ergibt sich somit eine weitere Eigenschaft der Lizenz neben dem Namen, die die Erstellung einer eigenen Tabelle nötig macht:



5. License
   - `LID` [integer] - eindeutiger Bezeichner; Primärschlüssel
   - `LName` [string] - Name der Lizenz (z.B. GPL) oder Beschreibung (z.B. non-commercial use free)
   - `LShowHomepage` [boolean] - Muss aus Lizenzgründen beim Download die Homepage aufgerufen werden?
   - ...

Durch Einführung dieser Tabelle wurde in `Sounds` das Attribut `SLicense` durch den Fremdschlüssel `LID` ersetzt.

3.1 Boolean-Variablen in SQL

Da unter SQL der Datentyp boolean nicht zur Verfügung steht, dieser aber oft und auch von relaKs benötigt wird, muss Abhilfe geschaffen werden.

Dazu wird in relaKs eine Enumeration mit den möglichen Werten *TRUE* und *FALSE* erstellt, die dann wie ein boolean-Typ verwendet werden kann. Ein Beispiel hierfür findet sich in der Tabelle `Server` unter der Bezeichnung `PermissionGranted`.

4 Datenbankerstellung und -pflege

Die Datenbank wurde auf dem lokalen Rechner mittels phpadmin[7] angelegt. Sie ist - wenn mein Rechner online ist - auf dem Host `campioni.dyndns.org` in der Datenbank `relaKs` mit Benutzernamen `relaks` und Passwort `soundscape` zu erreichen.

Zum jetzigen Zeitpunkt ist jedem mit diesen Benutzerdaten Schreibzugriff erlaubt. Dies soll anders werden, sobald die Datenbank auf einem öffentlichen Server, etwa des KDE-Projektes [5] liegt und relaKs der Öffentlichkeit zum Download zur Verfügung steht.

5 relaKs-Admin

Dieses unter Verwendung von Qt[8] geschriebene Tool ermöglicht den Zugriff auf Eigenschaften von Sounds und Themes, sowie das Löschen, Hinzufügen und den Download derselben. Zusätzlich steht eine Funktion zur Säuberung der Datenbank von verwaisten Elementen zur Verfügung.

Im Folgenden werden die wichtigsten Bestandteile der Programmoberfläche beschrieben und dabei insbesondere ihre SQL-Funktionen näher beleuchtet. Darin werden Qt-typische Begriffe verwendet, die in der Dokumentation von Trolltech[8] erklärt sind.

Das Tool relaKs-Admin findet sich im Unterverzeichnis `src/relaks-admin` der relaKs-Distribution.

5.1 RelaKsAdminMain

Das Hauptfenster enthält ein Widget vom Typ relaKsAdminMain. Nach Betätigen des QPushButtons *Load Database* wird die Datenbank geöffnet. Dabei wird die Qt-Datenbank-API angewandt:

```
// from RelaKsAdminMain::loadDatabase
db = QSqlDatabase::addDatabase("QMYSQL");
// factory function, mysql driver
db.setHostName("campioni.dyndns.org"); // my host
db.setPort(3306); // sql default port
db.setDatabaseName("relaKs");
db.setUserName("relaks");
db.setPassword("soundscape");
db.open();
```

`db` ist dabei vom Typ QSqlDatabase. Die gesetzten Attribute sind selbsterklärend. Nach Öffnen einer Datenbank wird global mit allen später erzeugen Qt-Datenbank-Objekten auf diese Datenbank zugegriffen. Hierfür sorgt die statische Funktion `QSqlDatabase::addDatabase`, die unter Qt/kdelibs als *factory function* klassifiziert wird.

Weiter zeigt das Fenster zwei QListViews für die Anzeige der `TName`s aller Datensätze von `Theme`s und der `SFilename`s von `Sound`s. Die Daten werden mittels der in Qt4 neu eingeführten Model-View-Architektur[6] geladen. Dabei erzeugt man ein Objekt, welches die eigentlichen Daten enthält (*Model*) und eines oder mehrere, das die Daten anzeigt (*view*). Im folgenden Beispiel wird ein QSqlQueryModel und als View das QListView-Objekt `QLVThemes` verwendet:

```
QSqlQueryModel *ModelThemes = new QSqlQueryModel;
ModelThemes->setQuery("SELECT TName FROM Theme");
// model data is retrieved from database
QLVThemes->setModel(ModelThemes);
// use model for this listview
```

Das Ergebnis des Querys "`SELECT TName FROM Theme`", eine Liste der vorhandenen Theme-Namen, wird somit im QListView angezeigt.

Soll ein neuer Sound oder ein neues Theme erzeugt werden, so wird zunächst ein Datensatz eingefügt:

```
QSqlQuery QSQInsertQuery
  ("INSERT INTO Sound(SFilename) VALUES(\".wav\")");
QSqlQuery QSQID("SELECT LAST_INSERT_ID()");
// returns last inserted ID with auto_increment
QSQID.next();
RelaKsAdminSound *RAS = new RelaKsAdminSound
  (QSQID.value(0).toInt(), 0);
// configure sound now!
RAS->show();
connect(RAS, SIGNAL(destroyed()), this, SLOT(loadDatabase()));
// reload database once configure is finished
```

Da das Feld `SID` mit dem Sql-Attribut `auto_increment` ausgestattet ist, wird die `SID` automatisch auf den nächsthöheren freien Wert gesetzt. Der automatisch gesetzte Wert ist mittels des Querys

```
SELECT LAST_INSERT_ID();
```

abrufbar.

Nach Initialisierung des Querys muss zunächst die Methode `QSqlQuery::next()` einmal aufgerufen werden, um mittels `QSQID.value()` auf die erste Zeile der vom Query gelieferten Tabelle zugreifen zu können. Der `value(int)` übergebene Parameter bezeichnet die gewählte Spalte dieser Zeile. Der Inhalt der Tabellenzelle wird von QSqlQuery als QVariant gespeichert. Dies ist ein universeller Datentyp für Zahlen, Strings und weitere Qt-typische Daten, der mittels Konverter-Funktionen (hier `toInt()`) in klassische Datentypen umgewandelt werden kann.

Die Funktion ruft sofort nach Erstellung des Datensatzes den Dialog RelaKsAdminSound (siehe Sektion 5.2) aus, der zur Konfiguration der Soundeigenschaften dient und als initialisierenden Parameter nur die `SID` des Datensatzes und den für `QWidget`-Objekte üblichen Pointer des *parent*-QWidgets benötigt (hier 0, da der Dialog als eigenes Fenster geöffnet und nicht in ein schon bestehendes Widget eingebunden wird).

Bei der Konfiguration eines bereits bestehenden Datensatzes kommt die Schwierigkeit hinzu, den aktuell markierten Datensatz aus der Liste herauszulesen. Die API von QListView ist hier seit der Einführung der Model-View-Architektur im Unterschied zu früheren Versionen alles andere als intuitiv:

```
QString RelaKsAdminMain::selectedThemeName()
// returns Title of currently selected theme
// exceptions: QString::null or empty QString
```

```
// when nothing is selected
{
  QItemSelectionModel *selectionModel =
    QLVThemes->selectionModel(); // selection model

  if(!selectionModel) // no selection model present
    return QString::null;

  return QLVThemes->model()->data
    (selectionModel->currentIndex()).toString();
  // selected item has index of selectionModel->currentIndex()
}

int RelaKsAdminMain::selectedThemeID()
// returns TID of currently selected theme
// exception: -1
{
  QString QSTName = selectedThemeName();
  if(QSTName.isEmpty()) // no valid selection
    return -1;
  QSqlQuery QSQID("SELECT TID FROM Theme WHERE TName=\""
    + QSTName + "\"");
  QSQID.next();
  return QSQID.value(0).toInt();
}
```

Die selbstgeschriebene Funktion `selectedThemeName()` gibt den QString des gewählten Elementes zurück. Dazu muss das *Selection Model* des jeweiligen Views erfragt und aus diesem der Index des aktuell gewählten Elementes gewonnen werden. *Selection Models* sind nützlich, um Auswahleigenschaften zwischen verschiedenen Views auszutauschen, hier allerdings eher hinderlich. Da der QString hier der `TName`-Eigenschaft entspricht, muss über den entsprechenden Query noch die `TID` gefunden und zurückgegeben werden.

Das Löschen von Datensätzen ist ebenfalls möglich. Dabei überprüft RelaKs-Admin bei zu löschenden Sounds, ob diese auch tatsächlich nicht mehr in Benutzung sind, wobei "..." für die jeweilige ID steht:

```
SELECT COUNT(*) FROM builds WHERE SID=...;
```

Wird eine Zahl größer 0 zurückgegeben, so wird das Löschen mit einer Fehlermeldung abgebrochen, sonst folgt die Löschanweisung:

```
DELETE FROM Sound WHERE SID=...;
```

Ein ähnlicher Mechanismus wird bei der `Cleanup Database`-Funktion angewandt. Hier werden alle Server aus der Tabelle gelöscht, die keinen Sound hosten.

Zuletzt ist der Download-Mechanismus aus `RelaKsAdminMain::downloadTheme()` interessant. Zusächst werden die für das gewählte Theme mit `TID=...` zu ladenden Sounds ausgewählt:

```
SELECT SID FROM builds WHERE TID=...;
```

und dann, so noch nicht lokal vorhanden (was nach Abfrage der `SFilename`s festegestellt wird), heruntergeladen, wozu der Dialog RelaKsAdminDownload dient.

5.2 RelaKsAdminSound

Dieser Dialog wird aufgerufen, um Sounds zu konfigurieren. Er lädt zunächst alle Sound-Eigenschaften. Dabei ist zum Beispiel das `License`-Attribut von Interesse, bei dem wieder ein Model, diesmal für die Anzeige in der QComboBox `QCBLicense` eingesetzt wird. Man beachte die Kommentare zum Verständnis des Ablaufes.

```
QSqlQueryModel *QSQMLicenseList = new QSqlQueryModel(this);
QSQMLicenseList->setQuery("SELECT DISTINCT LName FROM License");
// all license names are now in the model
QSqlQuery QSQLicense("SELECT License FROM Sound WHERE SID="
  + QString::number(SID));
// the sound's license ID in QSQLicense
QSQLicense.next(); // get to first data set
QSqlQuery QSQLicenseName("SELECT LName FROM License WHERE LID="
  + QSQLicense.value(0).toString());
// the sound's license name in QSQLicenseName
QSQLicenseName.next();
QCBLicense->setModel(QSQMLicenseList);
// use as model for QCBLicense
QCBLicense->setCurrentIndex
  (QCBLicense->findText(QSQLicenseName.value(0).toString()));
// set current license as current item of ComboBox
```

Darüber hinaus muss eine QComboBox (auch bekannt als Drop-Down-Liste) mit den möglichen Werten von `Format` geladen werden. `Format` ist eine Enumeration. Mittels des SQL-Befehls

```
"SHOW COLUMNS FROM Sound LIKE \'SFormat\';"
```

werden die möglichen Werte angezeigt. Das Ergebnis der Anfrage hat in der zweiten Tabellenspalte der ersten Zeile einen String, der etwa wie folgt aussieht:

```
"enum('WAV','MP3')"
```

Aus diesem können die Werte dann mittels String-Operationen gewonnen werden.

Anders wird die Server-QComboBox geladen, in der alle bereits verfügbaren Server und ein zusätzliches Element zur Definition eines neuen Servers enthalten sein sollen:

```
QSqlQuery QSQServerQuery("SELECT SURL FROM Server");
// server list
while(QSQServerQuery.next())
  QCBServer->addItem(QSQServerQuery.value(0).toString());
QCBServer->addItem("New...");
```

Beim Schließen des Dialogs mittels `OK` werden die Optionen in die Datenbank übertragen. Dazu müssen die Inhalte der Lizenz- und Server-QComboBox, die als Text vorliegen, mittels eines Querys zunächst wieder in `ID`s umgewandelt werden:

```
QSqlQuery QSQLicense("SELECT LID FROM License WHERE LName=\""
  + QCBLicense->currentText() + "\"");
QSQLicense.next();
if(QSQLicense.value(0).toInt()>0)
  QSqlQuery QSQLicenseUpdate("UPDATE Sound SET License="
    + QSQLicense.value(0).toString()
    + " WHERE SID=" + QString::number(SID));

QSqlQuery QSQServer("SELECT SID FROM Server WHERE SURL=\""
  + QCBServer->currentText() + "\"");
QSQServer.next();
if(QSQServer.value(0).toInt()>0)
  QSqlQuery QSQServerUpdate("UPDATE Sound SET Server="
    + QSQServer.value(0).toString()
    + " WHERE SID=" + QString::number(SID));
```

Es folgt die Übertragung der sonstigen Attribute:

```
QString QSUpdateQuery("UPDATE Sound SET SRelativeURL=\""
  + QLERelativeURL->text() +  "\", SFormat=\""
  + QCBFormat->currentText()
  + "\", SLength=" + QLELength->text()
  + ", SHomepage=\"" + QLEHomepage->text()
  + "\", SFilename=\"" + QLEUniqueName->text()
  + "\" WHERE SID=" + QString::number(SID));
QSqlQuery QSQUpdateQuery(QSUpdateQuery);
```

Das beim Schließen des Dialogs gesendete Signal `RelaKsAdminSound::-destroyed()` wird von einem Slot in `RelaKsAdminMain` aufgefangen, der dann die Anzeige der Datenbanktabellen neu lädt.

5.3 RelaKsAdminTheme

Dieser Dialog wird aufgerufen, um ein Theme zu konfigurieren. Die Klasse lädt mittels Querys die Daten des Themes und zeigt sie in entsprechenden Widgets an. Beim Klick auf *OK* werden die veränderten Daten, ähnlich wie in RelaKsAdminSound, übertragen. Die angewandten SQL-Techniken wurden bereits im vorhergehenden Unterkapitel beschrieben.

5.4 RelaKsAdminSoundInTheme

Der Dialog RelaKsAdminTheme enthält für jeden im Theme verwendet Sound ein Widget der Klasse RelaKsAdminSoundInTheme, die zusammen in einem QStackedWidget übereinander angeordnet werden. Dabei dient RelaKsAdminSoundInTheme zur Konfiguration der Daten der `builds`-Tabelle.

Nach Laden der Daten werden die Changed-`SIGNAL`s der die Daten anzeigenden Widgets mit eigens eingerichteten `SLOT`S verbunden:

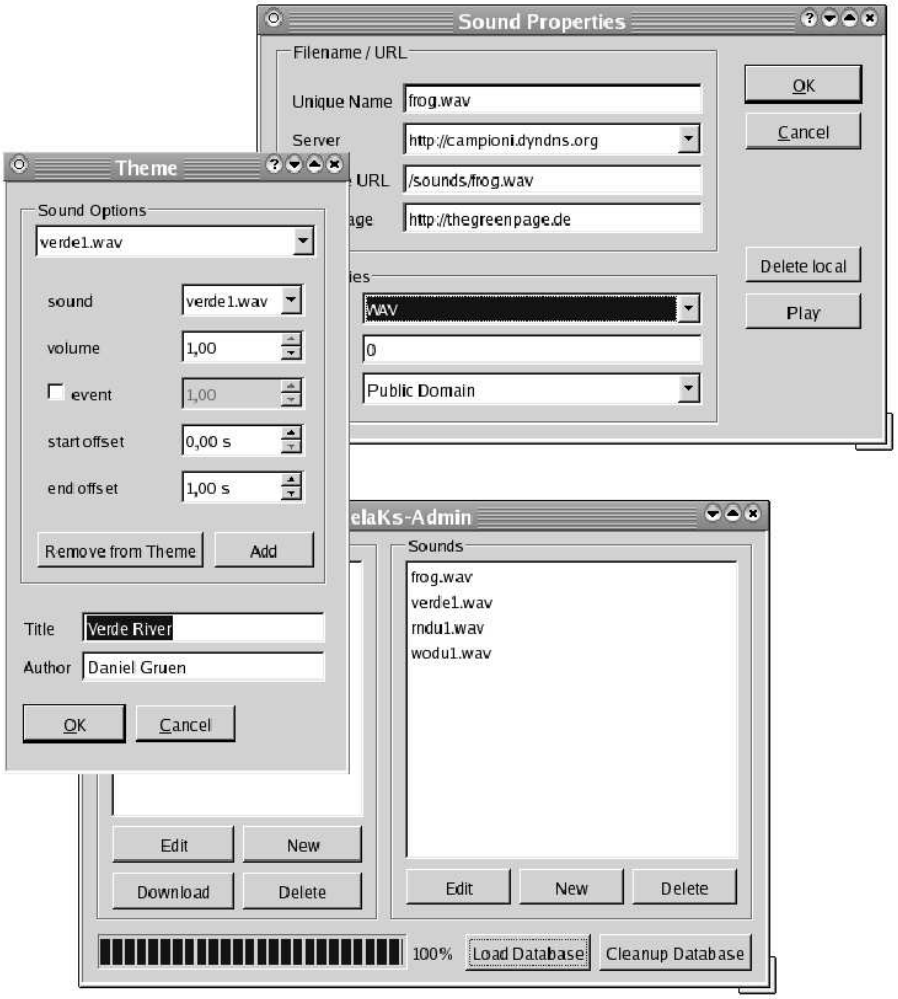
```
// from RelaKsAdminSoundInTheme::RelaKsAdminSoundInTheme()
connect(QDSBVolume, SIGNAL(valueChanged(double)),
        this,       SLOT(setVolume(double)));
```

Der angegebene Befehl verbindet das `SIGNAL valueChanged()` der QDoubleSpinBox QDSBVolume mit dem `SLOT setVolume()` der eigenen Klasse, welcher dann bei jeder Wertänderung der Spinbox aufgerufen wird:

```
void RelaKsAdminSoundInTheme::setVolume(double dv)
{
  QSqlQuery QSQUpdateQuery("UPDATE builds SET Volume="
    + QString::number(dv)
    + " WHERE TID=" + QString::number(TID)
    + " AND SID=" + QString::number(SID));
}
```

Der hier vorgenommene SQL-Query setzt den neuen Wert in der Datenbank.

5.5 Screenshots



6 relaKs-Applet

Das Abspielen von relaKs-Themes erfolgt mittels eines KDE-Applets (ein Programm, das sich in die Programmleiste von KDE einfügt). Dieses Applet wurde zum großen Teil bereits 2003 und 2004 geschrieben, wobei aufgrund des Lernaufwandes der KDE-Bibliotheken [3] und insbesondere der aRts-Biblotheken [2] und Soundfunktionen ein großer Zeitaufwand nötig war.

Im Rahmen des Datenbank-Unterrichtes erfolgte dann die Umstellung auf und Anwendung von Datenbankfuntionalitäten.

Der Quellcode des Applets findet sich im Unterverzeichnis `src` der relaKs-Distribution.

Im Folgenden werden kurz die Klassen des Applets beschrieben und ihre Sql-Funktionen erklärt.

6.1 relaKs

Die Klasse `relaKs` ist von `KPanelApplet` abgeleitet und beschreibt daher ein Miniprogramm, dass sich in das KDE-Panel einfügt. Solche Panel-Applets werden zu dynamisch ladbaren Bibliotheken kompiliert, die vom KDE-Panel-Programm `kicker` zur Laufzeit geladen werden können.

In dieser Art der Anwendung liegen die Gründe für einige Schwierigkeiten. So ist z.B. ein Debugging nur schwer möglich, da keine Konsolenausgabe des vom Panel geladenen Applets erfolgen kann und das Applet nicht ohne Panel lauffähig ist. Zudem treten Probleme auf, die bei normal kompilierten Programmen nicht ins Gewicht fallen, z.B. Absturz der Bibliothek wenn in der Klassendeklaration bereits geplante Funktionen nicht implementiert sind, auch wenn diese gar nicht aufgerufen werden.

Der Klasse `relaKs` ist ein KPopupMenu zugeordnet, das bei Rechtsklick auf das Applet erscheint. Die `activated()-SIGNAL`s des Menus sind mit verschiedenen `SLOT`s verbunden. Die folgende Anweisung erzeugt einen Menueintrag in `menu` mit dem System-Icon für `fileimport` und dem möglicherweise in einer Phrasenliste gespeicherten Übersetzung von *Load Database*, dessen `activated()-SIGNAL` mit dem `SLOT loadDatabase()` der eigenen Klasse verbunden wird:

```
menu->insertItem(SmallIcon("fileimport"),
  i18n("Load Database"), this, SLOT(loadDatabase()));
```

Wird dieser Menueintrag ausgewählt, so lädt die Funktion `loadDatabase()` die Datenbank ähnlich wie in relaKs-Admin. Bei jedem Aufruf des Kontext-Menüs wird ab sofort eine Liste der `Theme`s abgefragt und in ein Untermenü eingefügt. Hierzu wird die QAction-Klasse, ein abstraktes Interface für benutzergesteuerte Funktionen, verwendet. QAction-Objekte können in QActionGroup-Klassen zusammengefasst werden und in Menüs oder Werkzeugleisten eingefügt werden. Es ist möglich, QAction-Objekte auswählbar zu machen, so dass das Aussehen an eine QCheckBox erinnert, wie es hier für die Auswahl des aktuellen Themes gewählt wurde. Man beachte die Kommentare für ein besseres Verständnis der Funktionsweise:

```
QActionGroup *QAGThemes = new QActionGroup(this);
// a group of QAction items
```

```
QAGThemes->setExclusive( TRUE );
// only one of them is checkable
QSqlQuery QSQThemes("SELECT TID, TName FROM Theme");
while(QSQThemes.next()){
  QAction * QATheme = new QAction(QAGThemes, "ThemeItem", TRUE );
  // insert new QAction into QActionGroup
  QATheme->setText(QSQThemes.value(1).toString());
  // set QAction's text
  QATheme->addTo(themesMenu);
  // insert QAction into menu
}
connect (themesMenu, SIGNAL(activated(int)),
         this,       SLOT(themeSelected(int)));
// call themeSelected(int) when clicked
```

6.2 relaKsTheme

Die Klasse `relaKsTheme` repräsentiert ein RelaKs-Theme und enthält demzufolge Methoden zum Laden eines Themes aus der Datenbank und zusätzlich Attribute zur Speicherung der Themeoptionen und Pointer auf die zum Theme gehörigen `relaKsSound`s.

Der Datenbankzugriff erfolgt dabei allein über die Methode `relaKsTheme::load(int TID)`, der entsprechende Themeoptionen lädt und `relaKsSound`-Instanzen entsprechend der im Theme enthaltenen Sounds erzeugt.

Wird die Methode `relaKsTheme::play()` aufgerufen, was als `SIGNAL-SLOT`-Connection z.B. bei Linksklick auf das relaKs-Icon erfolgt, werden mittels `relaKsSound::play()` Hintergrundklänge gestartet und mittels QTimer immer wieder neu gestartet, so dass ein kontinuierlicher Klang entsteht und zudem als Ereignis markierte Klänge (`bBackground=FALSE`) in zufälligen Zeitabständen ebenfalls durch QTimer abgespielt.

6.3 relaKsSound

Die Klasse `relaKsSound` repräsentiert einen RelaKs-Sound und enthält demzufolge Methoden zum Laden und Abspielen von Sounddateien.

Beim Laden wird dabei auf die KDE-Ressourcenverwaltung zurückgegriffen:

```
filename = StandardDirs.findResource("sound",dfilename);
```

KDE sucht dabei in den Ressourcen-Verzeichnissen für KDE-Sounddateien (sowohl system-global als auch user-spezifisch) nach einer Datei `dfilename`.

Für das eigentliche Abspielen muss mittels einer `factory function` ein `KPlayObject` erzeugt werden, dass mittels des KDE-Soundsystems `aRts` [2] die Sounddatei ausgibt:

```
KArtsServer KASServer;
KDE::PlayObjectFactory KPOFFactory(KASServer.server());
KPlayObject KPOSound =
  KPOFFactory.createPlayObject(filename, true);
KPOSound->play();
```

Für die richtige Programmierweise, so dass keines der erzeugten Objekte frühzeitig aus dem Speicher gelöscht wird oder als *memory leak* zum Verbrauch von immer mehr Arbeitsspeicher führt war eine lange Experimentalphase notwendig.

6.4 Screenshot

Der Screenshot zeigt das Applet in einem durchsichtigen Panel vor einem Bildschirmhintergrund.

7 Entwicklung

Während die Planungsphase von relaKs weit zurückreicht, fand die Entwicklung der eigentlichen datenbankfähigen Programme in recht kurzer Zeit nach Entschluss zur Projektarbeit statt. Es folgt eine kurze chronologische Aufstellung der Arbeitsschritte und sonstigen Übungen:

bis Januar 2005 - relaKs-Applet ohne Datenbankzugriff (ca. 20 Stunden)
März 2005 - HTML-Übungen (ca. 2 Stunden)
März 2005 - SQL-Übungen (ca. 3 Stunden)
April 2005 - Datenbankentwurf-Übungen (ca. 3 Stunden)
April 2005 - Planung der relaKs-Datenbank (ca. 2 Stunden)
Mai 2005 - PHP-Übungen I und II (ca. 4 Stunden)
Mai 2005 - relaKs-Admin, erste Programmversion (ca. 10 Stunden)
Juni 2005 - relaKs-Applet, Datenbankunterstützung (ca. 3 Stunden)
Juni 2005 - neue Datenbankstruktur (ca. 7 Stunden)

Literatur

[1] Aire Freshener von Peter Hirschberg:
`www.peterhirschberg.com`

[2] aRts-Soundserver, die Sound-Bibliothek von KDE:
`www.developer.kde.org/documentation/library/3.4-api/arts/html/index.html`

[3] KDE-Bibliotheken:
`www.developer.kde.org/documentation/library/3.4-api`

[4] Gnu General Public License (Copyleft):
`www.gnu.org/copyleft/gpl.html`

[5] eine beliebte Benutzeroberfläche für Linux:
`www.kde.org`

[6] Beschreibung der von Qt4 angewandten Datenverwaltungs- und Anzeigestrategie:
`de.wikipedia.org/wiki/Model_View_Controller`

[7] PHP Admin zur Administration von MySQL-Datenbanken:
`www.phpmyadmin.net`

[8] Qt, die Bibliothek von Trolltech für GUI und mehr:
`www.trolltech.com`