RENAMER

Eine Projektarbeit von Artur Hallmann und Lennart Wagner

Inhaltsverzeichnis

Einführung

Wir mögen Fernsehserien so sehr, dass wir sie archivieren. Wenn wir sie bekommen haben die einzelnen Videos Dateinamen in denen prinzipiell schon alle Information vorhanden sind. Leider sind diese Dateinamen schlichtweg nicht schön:

```
The.Simpsons.S18E10.PDTV.XviD-NoTV.av

The.Simpsons.S18E12.PDTV.XviD-LOL.avi

The.Simpsons.S18E13.PDTV.XviD-2HD.avi

The.Simpsons.S18E14.PDTV.XviD-LOL.avi

The.Simpsons.S18E16.PDTV.XviD-LOL.avi
```

Schöner wäre:

```
The Simpsons – 18x10.avi

The Simpsons - 18x12.avi

The Simpsons - 18x13.avi

The Simpsons - 18x14.avi

The Simpsons - 18x16.avi
```

Genau dies soll das Programm „Renamer“ leisten.

Marktanalyse

Wenn man sich den Markt für Programme anschaut, die Dateien umbenennen, stellt man fest, dass es schon eine unglaubliche Menge an Programmen gibt die eine ähnliche Funktionalität besitzen. Warum noch ein Weiteres?

Hinzu kommt, dass besagter „Markt“ von kostenlosen Free- und OpenSource-Software dominiert wird. Hier ist wenig Geld zuholen. Sollte dieses Programm, dann etwa gar nicht marktfähig sein? Wir denken schon. Denn es gibt mindestens zwei Personen in diesem Markt. Wir – wir vereinen Angebot und Nachfrage in „einer“ Person. Wir glauben an unsere Idee und möglicherweise jemand anderes auch.

Ist-Analyse

Alle diese Programme funktionieren gut, wenn man alle Dateien, die man umbenennen möchte schon auf der Festplatte hat. Diese benennt man um und danach hat das Programm seinen Dienst getan.

Unsere Situation stellt sich jedoch anders dar: Es ist abzusehen, dass viele Dateien umbenannt werden sollen, jedoch sind diese noch gar nicht verfügbar. Es ist schwierig den richtigen Zeitpunkt abzupassen, wann genug Dateien da sind, dass es sich lohnen würde die in einem Rutsch umzubenennen. Und selbst dann kommt nächste Woche garantiert die nächste Folge raus.

Soll-Konzept

Unser Programm unterscheidet sich von allen anderen dadurch, dass wir eine Art Langzeitgedächtnis aufbauen. Die Regeln nach denen Dateien umbenannt werden, werden nach dem Umbenennen gespeichert.

Dies hat zur Folge, dass es keinen Unterschied macht, ob man viele Dateien auf einmal oder wenige Dateien häufig umbenennt. Der Aufwand ist der selbe.

Programmstruktur

Im Groben teilt sich das Programm in zwei Funktionen auf:

1. Informationen aus einem Dateinamen extrahieren
2. Neuen Dateinamen zusammensetzen

Dies sieht noch recht simpel aus, um es jedoch korrekt umzusetzen entsteht ein recht stattliches Objektmodell:

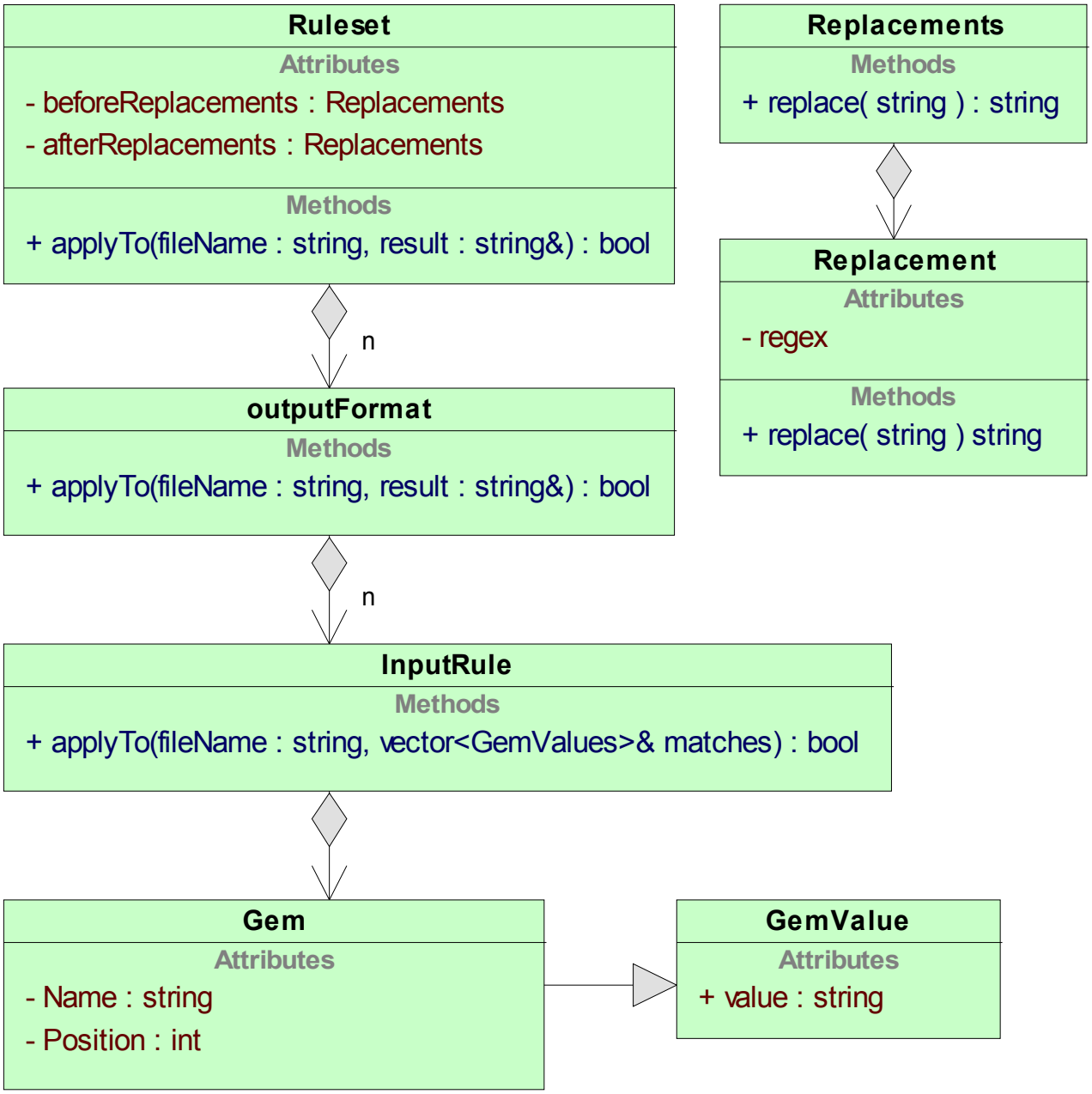

Abbildung 1: Objektmodell

Aus Gründen der Übersichtlichkeit sind nicht alle Methoden und Attribute aufgelistet.

Klasse Ruleset

Diese Klasse ist der Einstiegspunkt ins Objektmodell. Hier wird die Datenbank geöffnet. Als wichtigste Methode gibt es `ApplyTo()`. Im Endeffekt muss man dieser Methode nur einen Dateinamen übergeben und man bekommt einen verschönerten Dateinamen zurück.

Die Idee ist, dass für jede Serien ein `Ruleset` existiert.

Klasse OutputFormat

Im Eingangsbeispiel sind die resultierenden Dateinamen sehr ähnlich. Im realen Leben ist es jedoch nicht immer so einfach. Man betrachte folgende Dateinamen:

```
The Simpsons – 18x10.avi

The Simpsons – 06x16 - Bart vs. Australia.avi

The Simpsons – 06x25 - Who Shot Mr. Burns? (Part 1).avi

The Simpsons – 07x01 - Who Shot Mr. Burns? (Part 2).avi
```

Dies sind die typischen Möglichkeiten eine Serie zu benennen. Die ersten beiden Informationen die aus allen Dateinamen extrahiert werden können sind die *Staffel* und die *Episode*. Die nächste interessante Information ist der *Titel*. Leider steht diese Information nicht immer zur Verfügung. Wenn sie jedoch da ist, soll sie auch genutzt werden. In seltenen Fällen gibt es mehrteilige Folgen die inhaltlich zusammenhängen (*Part*).

Um diesen Umständen Rechnung zu tragen ist es möglich für ein `Ruleset` mehrere verschiedene Ausgabeformate zu erstellen.

Die könnte beispielsweise folgendermaßen aussehen:

```
The Simpsons – $staffel$x$episode$

The Simpsons – $staffel$x$episode$ - $titel$

The Simpsons – $staffel$x$episode$ - $titel$ (Part $part$)
```

Dies sind also die Dateinamen die entstehen können mit Platzhaltern für die Variablen Teile.

Klasse Gem

Aus Wikionary:

> gem (/ʤɛm/, plural gems)
>
> 1. A precious stone, usually of substantial monetary value or prized for its beauty or shine.
> 2. (figuratively) any precious or highly valued thing or person

Im Objektmodell ist dies zwar die letzte Klasse, jedoch auch die wichtigste. `Gems` sind die Information, die extrahiert werden (*Staffel, Episode, Titel, ...*). Wenn man die `Gems` eines Dateinamens hat ist es nur noch eine Frage von Suchen & Ersetzen, um den neuen Dateinamen zu generieren.

Klasse InputRule

Hier geschieht das eigentliche Extrahieren der `Gems` mit Hilfe von regulären Ausdrücken.

Angenommen folgende Dateien sollen umbenannt werden:

```
The.Simpsons.S18E10.PDTV.XviD-NoTV

The.Simpsons.S18E12.PDTV.XviD-LOL
```

Ein regulärer Ausdruck der dazu passt wäre beispielsweise:

```
The\.Simpsons\.S\d+E\d+\.PDTV\.XviD-(NoTV|LOL)
```

Jedes Ausgabeformat kann mehrere verschiedene `InputRules` enthalten, da die original Dateinamen teilweise so unterschiedlich sind, dass es unmöglich ist alles mit einem Ausdruck zu „erschlagen“.

Es ist möglich mit regulären Ausdrücken Information zu extrahieren. Dies geschieht dadurch, dass die entsprechenden Teile in Klammern gesetzt werden:

```
The\.Simpsons\.S(\d+)E(\d+)\.PDTV\.XviD-(NoTV|LOL)
```

Den Indizes der geklammerten Ausdrücke werden jeweils einem bestimmtes `Gem` zugeordnet.

Wie in Abbildung 1 ersichtlich ist sind die `InputRules` unterhalb der Ausgabeformate angeordnet. Dass heißt jedes Ausgabeformat hat eigene `InputRules`, die von anderen Ausgabeformaten unabhängig sind.

Klasse Replacements

Um das Ganze noch ~~komplizierter~~ einfacher zu machen gibt es noch die Möglichkeit Ersetzungen vorzunehmen. Dies vereinfacht die Sache insofern, dass bestimmte Text quasi global über das ganze `Ruleset` durch andere Texte ersetzen werden können. Dies ermöglicht es dann auch bestimmte Dinge „zu löschen“.

```
the.unit.219.repack.hdtv.xvid.notv.[VTV]

The.Unit.S02E16.HDTV.XviD-NoTV

The.Unit.S02E17.HDTV.XviD-XOR

The.Unit.S02E18.HDTV.XviD-NoTV

The.Unit.S02E20.HDTV.XviD-LOL

The.Unit.S02E21.HDTV.XviD-XOR

The.Unit.S02E22.Freefall.HDTV.XviD-FQM
```

Ersetzt man `[\.-](HDTV|XviD|NoTV|XOR|LOL|FQM)` durch Nichts erhält man:

```
the.unit.219.repack.[VTV]

The.Unit.S02E16

The.Unit.S02E17

The.Unit.S02E18

The.Unit.S02E20

The.Unit.S02E21

The.Unit.S02E22.Freefall
```

Um nicht zu monströse reguläre Ausdrücke schreiben zu müssen, können an jeder Stelle an der Ersetzungen vorkommen immer gleich mehrere Ausdrücke und Ersetzungen (vgl. Abbildung 1) angegeben werden.

Mehrere Ersetzungen deshalb, weil es auch einen Unterschied macht *wann* ersetzt wird. Deswegen kennt die Ruleset-Klasse `beforeReplacements` und `afterReplacements`. Die `beforeReplacements` werden angewendet bevor irgendetwas anderes mit den Dateinamen getan wird. Die `afterReplacements` werden angewendet kurz bevor die Datei tatsächlich umbenannt wird. An dieser Stelle bietet sich insbesondere an Whitespace am Ende zu entfernen.

Datenbank

Zur Speicherung der Daten aus dem Objektmodell nutzen wir Sqlite. Sqlite ist eine vollwertige Implementation des SQL92-Standards. Das Besondere an Sqlite ist, dass es kein Serverprozess oder überhaupt irgendein externer Prozess ist, sondern eine Bibliothek die statisch gelinkt werden kann.

Die im Weiteren mit `<<table>>` gekennzeichneten Diagramme sind keine C++-Klassen, sondern Sqlite-Tabellen. Des weiteren hat jede Tabelle eine Spalte `ROWID`, die automatisch von Sqlite hinzugefügt wird.

Im Programm Renamer übernimmt eine kleine Klasse Namens `TableRow` teilweise die Aufgabe des Datenbankzugriffs. Nachdem die Klasse `TableRow` mit dem Namen einer Tabellen initialisiert wurde, können direkt danach via `set/get` einzelne Felder verändert werden. Es repräsentiert eine Zeile in einer Tabelle.

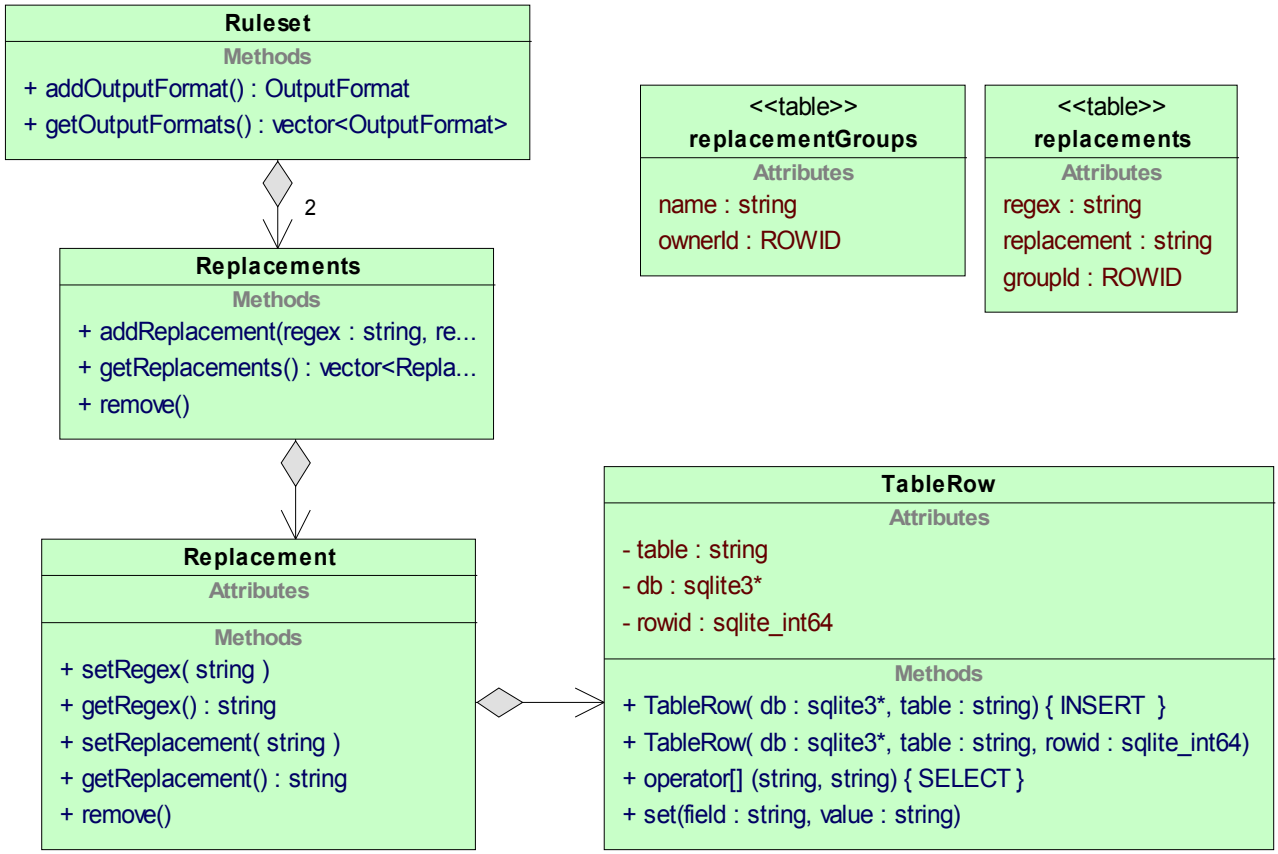

Abbildung 2: TableRow

Die Klassen `Ruleset` und `Replacement` sind hier nur der Vollständigkeit halber aufgeführt, manipulieren die Datenbank jedoch direkt.

Die Klasse Replacement *hat* ein TableRow-Objekt. Die `set/get`-Methoden sind folgendermaßen in `Replacement.h` definiert:

```
        void setRegex(boost::regex v)
            {mRow.set("regex", v.str());} ;


        boost::regex getRegex() const
            { return boost::regex(mRow.get("regex")); };
```

Es wäre vielleicht nahe liegend die Methoden von `TableRow` via Vererbung direkt öffentlich zu machen. Dies hat jedoch den entscheiden Nachteil, dass auch keine Überprüfungen stattfinden können. Der Heilige Gral der Objektorientierung – Datenkapselung – wäre dahin. Schon allein die Tatsache, dass `setRegex` einen Typ hat, verhindert in diesem Fall, dass Irgendetwas in die Datenbanken gelangen kann was kein regulärer Ausdruck ist.

Die weiteren Klassen sind gleichartig aufgebaut:

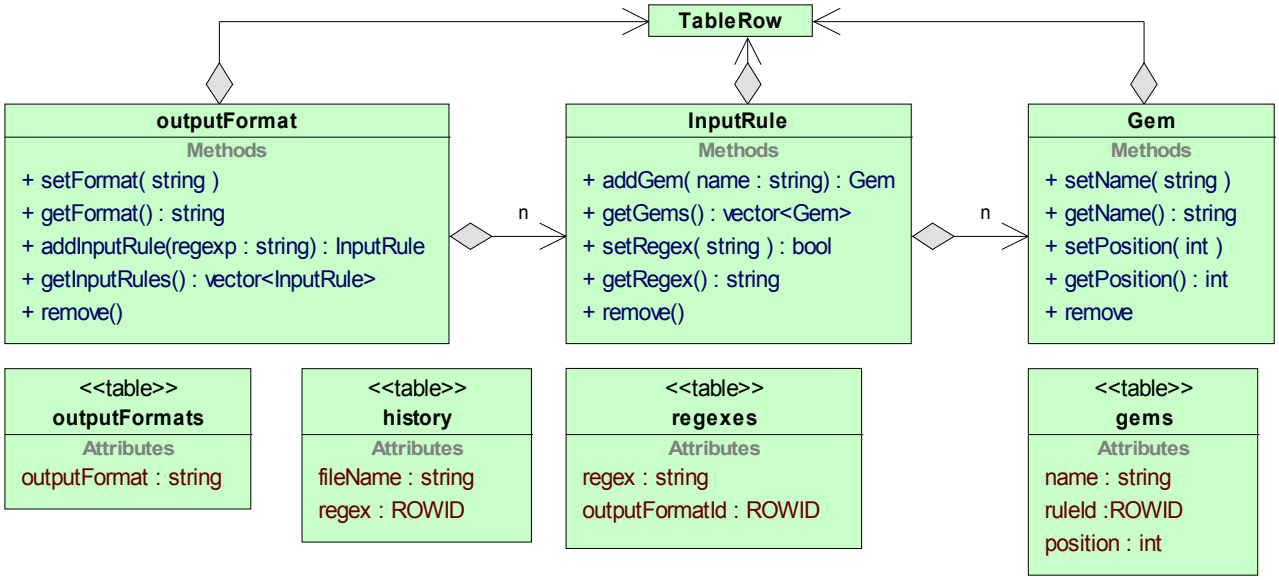

Abbildung 3: TableRow mit Objektmodell

Dadurch, dass die Objekte ihre Daten nicht in Attributen halten, sind sie letztendlich nichts anderes als eine Art Zeiger in die Datenbank. Dies hat auch den netten Nebeneffekt, dass in der Business Logik nahezu auf Pointer verzichtet werden konnte.

Die Tabelle `history` enthält alle Dateinamen die jemals konvertiert wurden. Die `setRegex`-Methode der `InputRule` stellt sicher, dass der reguläre Ausdruck nur so verändert werden kann, dass auch die historischen Dateinamen konvertiert werden können. Es soll verhindert werden, dass man eine `InputRule` im Nachhinein so verändert, dass eine schon funktionierende Konvertierung nicht mehr funktionieren würde.

Quellen

- http://en.wiktionary.org/wiki/gem
- http://www.famfamfam.com/lab/icons/
- http://www.sqlite.org/
- http://www.boost.org/