



Universität Zürich
Institut für Informatik

Entwicklung eines RDF Parsers für transaktionsbasierte Daten

**Bachelorarbeit in Informatik
abgegeben von**

Michael Imhof
Birmensdorf, Schweiz
Matrikelnummer: 04-702-130

Betreuer: Jonas Lüll
Abgabedatum: 24. August 2007

Universität Zürich
Institut für Informatik (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Schweiz

Inhaltsverzeichnis

1	Einleitung	5
1.1	<i>Inhalt und Aufbau dieser Arbeit</i>	<i>5</i>
2	Einführung in RDF	8
2.1	<i>Komponenten</i>	<i>8</i>
2.2	<i>Namespaces.....</i>	<i>10</i>
2.3	<i>Containers.....</i>	<i>10</i>
3	RDF Parser	12
3.1	<i>Jena oder VRP?</i>	<i>12</i>
3.2	<i>Die Wahl des RDF Parsers für JRP.....</i>	<i>14</i>
3.3	<i>Sesame als Alternative.....</i>	<i>14</i>
4	Die Datenbank	16
4.1	<i>Die Datenbank-Anbindung</i>	<i>16</i>
4.2	<i>Das Datenbankschema.....</i>	<i>19</i>
5	Die Logik des Programms	20
5.1	<i>Die Klasse RdfParser.....</i>	<i>21</i>
5.2	<i>Die Klasse DatabaseManager.....</i>	<i>21</i>
5.3	<i>Die Klasse DataRow.....</i>	<i>22</i>
5.4	<i>Die Klasse RdfDialog</i>	<i>22</i>
5.5	<i>Die Klasse DirectoryFilter</i>	<i>23</i>
5.6	<i>Code Refactoring</i>	<i>23</i>
5.7	<i>Einschränkungen</i>	<i>25</i>

6	Testing.....	27
6.1	<i>Testdaten.....</i>	27
6.2	<i>Testmethoden.....</i>	28
7	Probleme / Lessons Learned	29
8	Fazit.....	30
	Literaturverzeichnis.....	31
	Appendix	32
A	<i>Programmdokumentation.....</i>	32

Abbildungsverzeichnis

Abbildung 1: RDF Statement	9
Abbildung 2: Das Datenbank-Schema.....	19
Abbildung 3: Das Klassendiagramm von JRP	20
Abbildung 4: Lösung zur Speicherung mehrwertiger Attribute.....	25
Abbildung 5: JRP Logo	32
Abbildung 6: Welcome to JRP	33
Abbildung 7: Auswahl des RDF Input Files.....	34
Abbildung 8: Der File-Browser	34
Abbildung 9: Eingabe der MySQL-Settings	35
Abbildung 10: Der Folder Browser.....	35
Abbildung 11: Auswahl der Tags	36
Abbildung 12: Auswahl einer bestehenden Datenbank	37
Abbildung 13: Erzeugung einer neuen Datenbank	37
Abbildung 14: Ende des Dialogs.....	38

Tabellenverzeichnis

Tabelle 1: Vergleich zwischen VRP und ARP / Jena.....	13
---	----

1 Einleitung

Java RDF Parser (kurz: JRP) ist ein Programm zum Einlesen, Verarbeiten und Ausgeben von RDF Daten. Was hat es mit diesem Tool auf sich und wie ist dieses entstanden? Die nachfolgenden Abschnitte sollen einen Überblick über Inhalt und Aufbau dieser Arbeit vermitteln.

1.1 *Inhalt und Aufbau dieser Arbeit*

Resource Description Framework (RDF) Datensätze erfreuen sich in der modernen Wissenschaft grosser Beliebtheit. Das auf XML basierende Format wird benötigt, um Daten mit (Meta-)Daten zu beschreiben und somit Ressourcen im World Wide Web (WWW) zu repräsentieren (Xu, 2004). RDF nimmt deshalb im Rahmen des Semantic Web¹ eine zentrale Position ein.

Ziel dieser Arbeit ist es, ein eigenständiges Programm mit Java zu programmieren, welches es erlaubt, RDF Datensätze einzulesen, diese nach bestimmten Kriterien zu verarbeiten und anschliessend die gewünschten Informationen in einer Datenbank zu speichern. Bei den ausgelesenen Daten handelt es sich hierbei um Informationen über den Sender und den Empfänger einer Transaktion.

Das Programm soll hierbei so dynamisch wie möglich gestaltet sein. Das heisst, die Pfade zum RDF File und zur Datenbank sowie die entsprechenden Tags für Sender und Empfänger im RDF File sollten vom Benutzer selbst ausgewählt werden können. Das Datenbankschema soll so einfach wie möglich gehalten werden; in einer einzigen Tabelle sollen jeweils Sender und Empfänger der einzelnen Transaktionen eingetragen werden. Zudem benötigt natürlich jede dieser "Verbindungen" (entspricht einer Zeile in der Tabelle) eine eindeutige ID, welche zugleich als Primärschlüssel dienen soll.

¹ "SemanticWeb gives data well-defined and machine-understandable meaning so that they can be processed by intelligent software agents on human's behalf. Data are expressed in terms of ontologies, which define their concepts and relationships." (Dong et al., 2004)

Ziel ist es schlussendlich, die Komponente in Verbindung mit dem Programm TVIS verwenden zu können, welches zur Visualisierung von transaktionsbasierten Daten und zur Annotierung von Transaktionsmustern verwendet wird (Lüll, 2005).

Zum Parsen der Daten im RDF Format soll eine bereits vorhandene Open Source Library verwendet werden. Die Tags für Sender und Empfänger sollte der Benutzer wie bereits erwähnt selbst angeben können.

Des Weiteren soll der ganze Code natürlich so dynamisch wie möglich gehalten werden, damit allfällige Änderungen auf einfache und effiziente Art und Weise vorgenommen werden können.

Neben der Dynamik, die das Programm besitzen soll gibt es auch Teile, die immer gleich bleiben. Hierzu zählt hauptsächlich die Datenbank, weshalb der Tabellename und das Datenbankschema, welches ja nur aus einer einzigen Tabelle besteht, fix im Code verankert werden können. Vorgegeben ist zudem, dass die Daten in eine MySQL-Datenbank ausgegeben werden sollen.

In dieser Arbeit wird nicht nur das Programm selbst besprochen, sondern unter anderem auch die theoretischen Grundlagen.

Das folgende Kapitel bietet dem Leser eine kurze Einführung in RDF, was für das weitere Verständnis dieser Arbeit von zentraler Bedeutung ist.

Danach folgt eine Übersicht über die wichtigsten bestehenden RDF Parser. Xu (2003) hat in seiner Diplomarbeit verschiedene RDF Parser im Hinblick auf deren Performance untersucht und verglichen und schliesslich auch eine Evaluierung vorgenommen. Die Wahl des entsprechenden Parsers im dritten Kapitel stützt sich auf seine Resultate.

Im vierten Teil dieser Arbeit wird die Datenbank genauer erklärt. Das Kapitel enthält sowohl einen Teil über die Anbindung als auch einen kurzen Abschnitt zum einfachen Datenbankschema.

Im fünften Kapitel folgen schliesslich einige Bemerkungen zur Logik des Programms; neben den an jener Stelle besprochenen Klassen werden auch die verwendeten Refactoring-Methoden nach Fowler (2005) und das Programm betreffende Einschränkungen zur Sprache gebracht.

Das sechste Kapitel bietet schliesslich noch einen Einblick in die verschiedenen Daten und Methoden, welche im Rahmen des Testing verwendet wurden.

Der siebte Teil enthält schliesslich noch einige Bemerkungen zu Problemen, welche im Laufe des Arbeitsprozesses aufgetreten sind, und Lehren, welche aus dieser Arbeit gezogen werden können.

Abgerundet wird die Arbeit durch ein Fazit, welches die Kernpunkte dieses Papiers zusammenfasst.

2 Einführung in RDF

Für das weitere Verständnis dieser Arbeit ist ein kleines Grundwissen über RDF unerlässlich. Deshalb dient dieses Kapitel als kleine Einführung in jene Thematik.

Gemäss Selçuk Candan et al. (2001) ist das Resource Description Framework (RDF) eine auf XML basierende Sprache, um Informationen in einer Webressource zu beschreiben; bei einer Ressource könne es sich um ein beliebiges "Objekt" im Web handeln, das Informationen in irgendeiner Form beinhalte und durch einen Uniform Resource Identifier (URI) eindeutig identifizierbar sei. Weiter erklären sie, dass es RDF ermögliche, strukturierte Metadaten zu codieren, auszutauschen und wiederzuverwenden. RDF sei zudem "domänen-neutral", treffe also in keiner Weise Annahmen über eine bestimmte Domäne.

Folgende Elemente sind für ein RDF Modell zentral und werden deshalb in den kommenden Abschnitten kurz erklärt:

- Komponenten
 - Ressourcen
 - Eigenschaften
 - Statements
- Namespaces
- Containers

2.1 Komponenten

Wie bereits weiter oben erwähnt, definieren Selçuk Candan et al. (2001) eine Ressource als ein eindeutig durch einen URI identifizierbares Objekt. Weiter erklären sie, dass diese mit Eigenschaftsnamen beschrieben werden, welche die Beziehungen der Werte ausdrücken, die damit verbunden sind; diese Werte können atomar oder andere Ressourcen sein, welche wiederum eigene Eigenschaften haben können. Wie sie weiter erklären, besteht ein RDF Modell aus drei grossen Komponenten:

- **Ressourcen:** Wie eben erwähnt, ist eine Ressource ein eindeutig durch einen URI identifizierbares Objekt (zum Beispiel eine Webseite).
- **Eigenschaften:** Eine Eigenschaft (englisch: property) bezeichnet gemäss Selçuk Candan et al. (2001) einen Aspekt, eine Charakteristik, ein Attribut oder eine Relation, welche benutzt wird um eine Ressource zu beschreiben.
- **Statements:** Nach Selçuk Candan et al. (2001) besteht ein RDF Statement aus einer bestimmten Ressource mit einer Eigenschaft plus deren Wert; in dieser Reihenfolge werden die einzelnen Elemente eines Statements auch als Subjekt, Prädikat und Objekt bezeichnet. Folgendes Beispiel aus der "Introduction to RDF and the Jena RDF API" von McBride (2007) soll dies verdeutlichen:

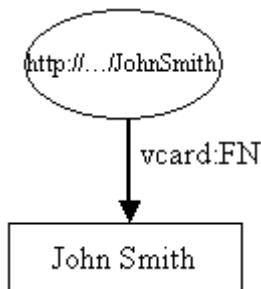


Abbildung 1: RDF Statement
(Quelle: http://jena.sourceforge.net/tutorial/RDF_API/index.html)

Abbildung 1 zeigt ein einfaches RDF Statement auf. Die Ressource ist John Smith, der eindeutig durch den URI `http://.../JohnSmith` identifiziert werden kann. Die Ressource (als Ellipse dargestellt) stellt gleichzeitig das Subjekt des Statements dar.

Der Pfeil mit der Beschriftung (`vcard:FN`) wird als Eigenschaft bezeichnet. Hier handelt es sich um den vollen Namen von John Smith. Gemäss McBride (2007) ist der Name der Eigenschaft eigentlich auch ein URI, wird jedoch im Graphen aus Gründen der Übersichtlichkeit in der Kurzform geschrieben. Der Name kann in zwei Teile aufgeteilt werden. Der Teil vor dem ':' wird als Namespace Prefix bezeichnet und repräsentiert einen Namespace (hier: `vcard`). Dazu folgt später mehr. Der Teil nach dem ':' ist der local name, welcher einen Namen innerhalb dieses Namespaces darstellt. Die Eigenschaft ist gleichzeitig das Prädikat des Statements.

Der Wert, den die Eigenschaft besitzt (hier: "John Smith", ein Literal), stellt schliesslich das Objekt des Statements dar. Statt eines Literals könnte die Eigenschaft auch wiederum eine Ressource als Wert haben (McBride, 2007).

2.2 *Namespaces*

Wie zuvor erwähnt, verwendet RDF Namespaces, um Eigenschaften korrekt zuordnen zu können. Nach Selçuk Candan et al. (2001) kann ein Namespace als Kontext verstanden werden, der einem Begriff eine bestimmte anstatt einer allgemeinen Bedeutung verleiht. Dies ermögliche es, Semantiken unter Communities zu definieren und auszutauschen, so Selçuk Candan et al. (2001) weiter.

2.3 *Containers*

Eine weitere schöne Eigenschaft von RDF ist die Möglichkeit, mehrwertige Attribute speichern zu können. Ein Beispiel dafür wäre eine Email-Nachricht, die an mehrere Empfänger gesendet wird. Die Eigenschaft Empfänger hätte hier bei einer Ressource mehrere Werte. In der RDF Version des Enron Email Datensatzes von Urs Wagner (2007) wurde dies genau so umgesetzt. Dieses Thema wird zu einem späteren Zeitpunkt dieser Arbeit nochmals aufgegriffen.

Nach Selçuk Candan et al. (2001) gibt es in RDF drei Typen von Containern, die verschiedene Gruppierungsarten erlauben:

- **Bag:** Ein Bag ist eine ungeordnete Liste von Ressourcen oder Literalen. Wie McBride (2007) weiter ausführt spielt hierbei die Ordnung keine Rolle; zwei Elemente könnten in der Reihenfolge getauscht werden, was das resultierende Modell nicht verändern würde.
- **Sequence:** Eine Sequenz ist eine geordnete Liste von Ressourcen oder Literalen. Im Gegensatz zum Bag spielt hierbei also die Reihenfolge der Elemente eine zentrale Rolle.

- **Alternative:** Eine Alternative ist eine Liste von Ressourcen oder Literalen, die Alternativen für den einzigen Wert einer Eigenschaft repräsentieren. McBride (2007) liefert hierzu folgendes schönes Beispiel:

Als Ressource kann man sich ein Software Produkt vorstellen, welches eine Eigenschaft besitzt, die angibt, woher es bezogen werden kann. Der konkrete Wert dieser Eigenschaft könnte dann gemäss McBride (2007) eine Alternative sein, die verschiedene Webseiten enthält, von denen die Software heruntergeladen werden kann.

3 RDF Parser

Eine Teilaufgabe dieser Arbeit bestand darin, eine geeignete Java Library zu finden, mit deren Hilfe die RDF Dateien korrekt geparkt und verarbeitet werden können. In seiner Diplomarbeit hat Xu (2004) RDF Parser untersucht und schliesslich eine Evaluierung vorgenommen. In diesem Kapitel wird einerseits auf die Resultate seiner Untersuchung eingegangen und darauf basierend die Wahl des entsprechenden Parsers begründet, andererseits mit Sesame noch eine weitere Alternative eines RDF Parsers präsentiert, die ebenfalls hätte in Frage kommen können.

3.1 Jena oder VRP?

In seiner Diplomarbeit hat Xu (2004) die beiden RDF Parser VRP (Validating RDF Parser) und ARP & Jena (Another RDF Parser) untersucht. Jena ist gemäss Xu (2004) ein „Java-Framework“ für die Realisierung von semantischen Webanwendungen. Es bietet eine programmatische Umgebung für „RDF“, „RDFS“, und „Web-Ontologie“ inklusiv einer Inferenzmaschine, so Xu (2004) weiter.

Um die beiden Parser im Hinblick auf Zeitkonsum und Speicherverbrauch zu vergleichen, hat er eine eigene Anwendung entwickelt. Diese weist verschiedene Arbeitsmappen auf, um Einzel- oder Vergleichs-Tests mit VRP und ARP / Jena durchführen zu können. Das für die Tests verwendete RDF Input File kann hierbei vom Benutzer beliebig ausgewählt werden.

Für die Tests können gemäss Xu (2004) verschiedene Parameter erfasst werden, so unter anderem die Anzahl Schleifendurchläufe beim Parsen der Datei oder das Aktivieren beziehungsweise Deaktivieren der expliziten Garbage Collection. Die Testergebnisse können gemäss Xu (2004) in ein Excel Sheet ausgegeben werden. Er vergleicht VRP und ARP / Jena bezüglich Erzeugung eines Modells, Auflistung der Statements und Parsen mit oder ohne Ausgabe von Triples. Die Ergebnisse seiner Tests sind in der nachfolgenden Tabelle zusammengefasst.

Vergleichsartikel		VRP	ARP/Jena
Erzeugung eines Modells	Zeitkonsum	-	+
	Speicherverbrauch ohne die explizite GC ²	-	+
	Speicherverbrauch mit der expliziten GC	-	+
Auflistung der Statements	Zeitkonsum	+	-
	Speicherverbrauch ohne die explizite GC	+	-
	Speicherverbrauch mit der expliziten GC	+	-
Parsen mit Triples	Zeitkonsum	+	-
	Speicherverbrauch ohne die explizite GC	+	-
	Speicherverbrauch mit der expliziten GC	+	-
Parsen ohne Triples	Zeitkonsum	-	+
	Speicherverbrauch ohne die explizite GC	-	+
	Speicherverbrauch mit der expliziten GC	-	+

Tabelle 1: Vergleich zwischen VRP und ARP / Jena
(Quelle: Xu, J. (2004). Die Evaluierung von Java RDF Parsern. *Diplomarbeit*. Frankfurt.)

Xu (2004) zieht aus seinen Erkenntnissen das Fazit, dass keine Aussage gemacht werden könne, welches Werkzeug dem anderen deutlich überlegen sei; je nach Bereich dominiere VRP oder ARP / Jena, was auf die unterschiedlichen Implementierungsstrategien der Autoren zurückzuführen sei. Im Gegensatz zu VRP modelliere zum Beispiel der Autor von ARP / Jena einen Arbeitsvorgang als relativ unabhängiges Modul, was zum Beispiel beim Überprüfen bestimmter Zielgruppen behilflich sei, da nicht alle Ressourcen durchsucht werden müssten, so Xu (2004) weiter.

Daraus kann der Schluss gezogen werden, dass je nach Einsatzgebiet der eine oder der andere Parser zu bevorzugen ist.

² GC = garbage collection

3.2 Die Wahl des RDF Parsers für JRP

Um eine Wahl bezüglich des Einsatzes des richtigen Parsers treffen zu können, muss man zuerst hinterfragen, welche Funktionalität schlussendlich tatsächlich benötigt wird.

JRP liest grundsätzlich RDF Daten ein und bereitet diese in einem internen Modell auf. Mittels einer SPARQL³-Abfrage werden schliesslich die gewünschten Daten aus dem Modell extrahiert.

Bezogen auf die von Xu (2004) verwendeten Vergleichskriterien scheint in erster Linie die Erstellung des Modells für diesen Anwendungsbereich eine zentrale Rolle einzunehmen. Gemäss Tabelle 1 ist ARP / Jena in diesem Bereich klar zu bevorzugen, weshalb schliesslich auch dieser Parser ausgewählt wurde.

3.3 Sesame als Alternative

Eine zusätzliche Alternative eines möglichen RDF Parsers wäre Sesame gewesen. Auf der Homepage (<http://www.openrdf.org/download.jsp>) ist darüber folgende Beschreibung zu finden: "Sesame is an open source Java framework for storing, querying and reasoning with RDF and RDF Schema. It can be used as a database for RDF and RDF Schema, or as a Java library for applications that need to work with RDF internally." (Aduna, 2007)

Für die Abwahl von Sesame können zwei Gründe vorgebracht werden:

1. **Sesame 2 ist erst als Beta-Version erhältlich:** Gemäss Aussagen von Aduna (2007) ist Sesame 2 ein grosses Update des Sesame Frameworks, welches sich jedoch noch in der Beta Phase befindet und nicht abwärtskompatibel ist. Des Weiteren warnt Aduna (2007) davor, dass die APIs vermutlich noch Fehler beinhalten würden und rät von deren Einsatz in Produktionsumgebungen ab.

³ SPARQL ist sowohl ein Protokoll wie auch eine Query-Language und kann für Abfragen auf einem RDF Modell verwendet werden.

2. **Probleme mit dem Handling:** Mehreren Versuchen, die alte Version von Sesame für JRP zu verwenden, scheiterten immer wieder. Bereits das Einlesen der Daten und die Erstellung des Modells hat mit Sesame nicht korrekt funktioniert. Ein Grund für dieses Probleme könnte laut Aussagen in diversen Online-Foren (zum Beispiel <http://www.openrdf.org/forum/mvnforum/viewthread?thread=86>) allenfalls sein, dass das Input File ein UTF-8 Encoding verwendet, was Java nicht korrekt handhaben könne. Jedoch hat auch dieser Hinweis nicht viel genutzt, weshalb der Entscheid schlussendlich gegen Sesame gefallen ist.

4 Die Datenbank

Um die gewünschten Daten aus dem RDF Input File in einer handhabbaren Art und Weise speichern zu können, verwendet der Java RDF Parser eine Datenbank. Dieses Kapitel enthält Ausführungen über die Anbindung und das Schema, welches verwendet wurde.

4.1 Die Datenbank-Anbindung

Java bietet unterschiedliche Möglichkeiten für Datenbank-Anbindungen. JRP verwendet JDBC mit MySQL.

Gemäss Konchady (1998), erlaubt JDBC Java, mit einer Datenbank eine Verbindungen aufzubauen, Abfragen zu machen und Aktualisierungen vorzunehmen. Gegenüber anderen Datenbank-Programmierungsumgebungen habe Java den Vorteil der "Plattform- und Verkäuferunabhängigkeit", so Konchady (1998) weiter; das selbe Java Programm könne auf einem PC, einer Workstation oder einem Netzwerk Computer laufen gelassen werden. Des Weiteren bietet JDBC nach Konchady (1998) den grossen Vorteil, dass das selbe Java Programm (ohne Änderungen) verwendet werden kann, wenn die Datenbank auf einen anderen Datenbank-Server verlegt wird.

Eine MySQL-Datenbank verwenden zu müssen war eine der Vorgaben in der Aufgabenstellung dieser Arbeit. MySQL ist heutzutage sehr verbreitet und bietet gemäss der folgenden Auflistung von Axmark (1999) viele Vorteile:

- MySQL kann sowohl mehrere Threads als auch mehrere User verwalten und ist dabei immer noch sehr schnell.
- Es gibt APIs zu vielen verschiedenen Sprachen.
- Der ODBC Treiber ist sowohl zuverlässig als auch gratis.
- MySQL ist sehr portabel.
- MySQL bietet viele verschiedene Spaltentypen, die sowohl alle ANSI 92 als auch alle ODBC 2.50 Typen (und ein paar neue) unterstützen.
- Die Unterstützung für fast alle ODBC 3.0 und SQL ANSI 92 Funktionen ist gewährleistet.
- Sowohl GROUP BY als auch ORDER BY werden durch MySQL unterstützt; des Weiteren sind Gruppierungsfunktionen wie COUNT, AVG, STD, SUM, MAX oder MIN möglich.
- MySQL bietet ausserdem die Möglichkeit, Tabellen aus unterschiedlichen Datenbanken in der gleichen Query zu mischen
- Es sind Einträge mit fixer wie auch mit variabler Länge möglich.
- MySQL kann mit sehr grossen Datenbanken umgehen. Bei TcX wird eine Datenbank mit über 50 Millionen Einträgen benutzt.
- MySQL ist sehr robust und weist keine undichten Stellen auf.
- Es besteht die Möglichkeit, viele verschiedene Zeichensätze zu konfigurieren
- Die Fehlermeldungen sind in vielen verschiedenen Sprachen verfügbar.
- Es gibt bereits viele Utilities und Erweiterungen.
- MySQL ist sehr gut dokumentiert, weshalb viele Fragen durch Konsultation des Manuals beantwortet werden können.

Eine Datenbank-Anbindung mit JDBC und MySQL ist einfach zu programmieren und benötigt lediglich einige wenige Zeilen Code. Eine gute und schrittweise Einführung in diese Problematik bietet zum Beispiel Baldwin (2004). Für den Java RDF Parser wurde die Datenbank-Anbindung in Anlehnung an sein Muster programmiert. Sämtliche Funktionalitäten, welche mit der Datenbank zu tun haben, sind in der Klasse *DatabaseManager* zu finden. Die Klasse bietet folgende Dienste an:

- **Zwei Konstruktoren mit unterschiedlichen Parametern**
 - Der erste wird verwendet, wenn die Datenbank bereits existiert.
 - Der zweite wird verwendet, falls eine neue Datenbank erstellt werden muss.

- **Methoden, um**
 - eine Datenbank zu erstellen.
 - eine Tabelle innerhalb einer Datenbank zu erstellen.
 - eine Verbindung aufzubauen.
 - eine Verbindung zu schliessen.
 - eine Zeile in einer Tabelle einzufügen.

Eine Verbindung wird durch den Aufruf *DriverManager.getConnection()* aufgebaut. Die Methode verlangt sowohl den Datenbank-URL als auch den MySQL Benutzernamen und das Passwort als Parameter. Diese Werte werden bereits bei der Instanzierung eines *DatabaseManager*-Objektes an den Konstruktor übergeben und anschliessend als Instanzvariablen gespeichert, sodass sie in der Methode *setUpConnection()* der Klasse *DatabaseManager* wiederverwendet werden können.

Auffällig ist vielleicht noch, dass die Methode *createTable()* keine Parameter enthält, wo doch zum Beispiel der Name der Tabelle hätte übergeben werden können. Dies wurde jedoch bewusst einfach gehalten, da das Datenbankschema einerseits nur aus einer Tabelle besteht und andererseits durch die fixe Verankerung des Tabellennamens im Code sichergestellt ist, dass alle Benutzer das gleiche Schema verwenden. Zumindest von Benutzerseite her sollte demnach der Tabellennamen nicht wählbar sein.

4.2 Das Datenbankschema

Der Java RDF Parser verwendet eine Datenbank mit einem sehr einfachen Schema. Diese enthält nämlich nur eine einzige Tabelle, die wiederum lediglich drei Spalten besitzt. Die Tabelle mit dem Namen "transactions" gibt Auskunft über Verbindungen zwischen zwei Parteien in einer Transaktion.

transactions	
PK	<u>edge_id</u>
	originator beneficiary

Abbildung 2: Das Datenbank-Schema

Zu diesem Zweck wird in der Tabelle pro Zeile ein Absender (englisch: originator) und ein Empfänger (englisch: beneficiary) gespeichert. Die Werte dürfen nicht leer sein, da eine gültige Transaktion sinnvollerweise einen Sender und einen Empfänger benötigt. Um die Verbindungen eindeutig identifizieren zu können, wird ein Primärschlüssel benötigt. Dies ist ein automatisch erzeugter numerischer Wert und wird in dieser Datenbank als "edge_id" bezeichnet. Die Namensgebung stammt aus der Überlegung heraus, dass sämtliche Transaktionen zusammen als Graph dargestellt werden könnten, wobei eine Kante eine Transaktion zwischen zwei Parteien darstellen würde. Die ID, welche eine Kante eindeutig referenziert, ist hier die "edge_id".

In dieser Version ist das Datenbank-Schema sehr einfach und intuitiv. Dies hat den Vorteil, dass beim Herauslesen der entsprechenden Daten aus dem internen Modell des RDF Input Files keine komplizierten Abfragen durchgeführt werden müssen, was einen verhältnismässig geringen Zeitaufwand für diesen Schritt zur Folge hat.

Bei Bedarf kann die Klasse *DatabaseManager* für komplexere Datenbankschemata durch wenig Programmieraufwand entsprechend angepasst werden.

5 Die Logik des Programms

In diesem Kapitel wird die Logik des Programms JRP erklärt. Zuerst wird auf die einzelnen Java-Klassen eingegangen, anschliessend auf das Refactoring des Codes nach Fowler (1999) und zum Schluss noch auf die Einschränkungen, die für das Programm gelten.

Im nachfolgenden Klassendiagramm sind sämtliche Klassen sowie deren Beziehungen aufgeführt. GUI-Elemente wie zum Beispiel Buttons oder Textfelder wurden in dieser Darstellung aus Gründen der Übersichtlichkeit weggelassen.

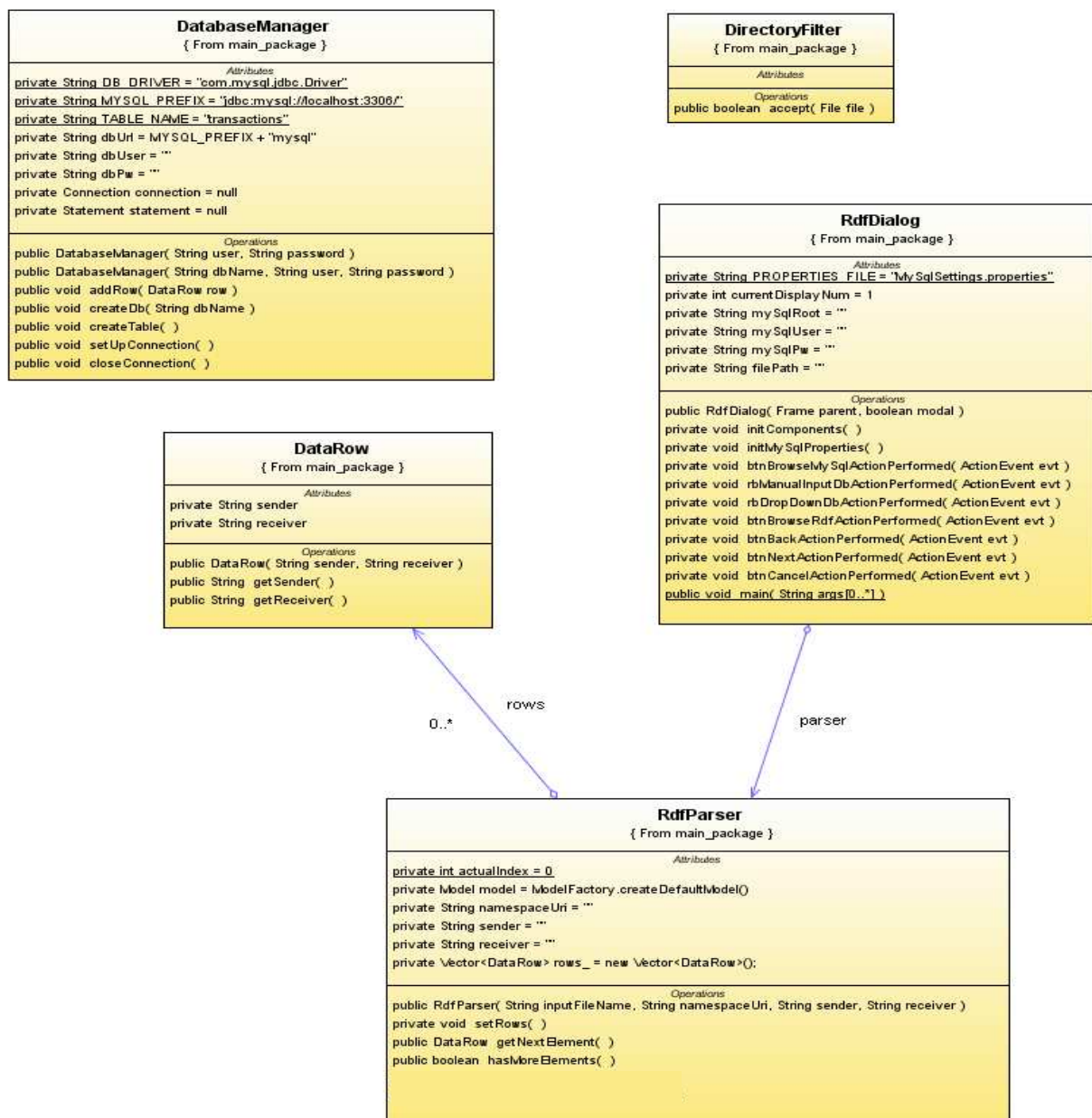


Abbildung 3: Das Klassendiagramm von JRP

5.1 Die Klasse *RdfParser*

Die Klasse *RdfParser* bietet Dienste an, um ein RDF File einlesen und ein internes Modell erzeugen zu können. Das Herzstück der Klasse ist die Instanzvariable *rows* vom Typ *Vector<DataRow⁴>*, in welcher die entsprechenden Zeilen enthalten sind, die anschliessend in der Datenbank gespeichert werden sollen. Über die Methode *setRows()* werden dem *Vector* die einzelnen *DataRow*'s hinzugefügt. Eine SPARQL-Abfrage auf dem Modell liefert hierfür die richtigen Daten zurück.

Im vierten Schritt des Dialogs wählt der Benutzer, wie in der Dokumentation im Appendix ausführlich beschrieben, die Tags für Sender und Empfänger sowie den entsprechenden Namespace-URI aus. Die Informationen werden dem *RdfParser*-Konstruktor übergeben, welcher diese den entsprechenden Instanzvariablen zuordnet. Somit können diese für die SPARQL-Abfrage in der Methode *setRows()* wiederverwendet werden.

5.2 Die Klasse *DatabaseManager*

Die Klasse *DatabaseManager* stellt sämtliche Methoden und Informationen zur Verfügung, welche im Zusammenhang mit der Datenbank benötigt werden. Wie bereits in Kapitel 4 erwähnt, bietet die Klasse zwei Konstruktoren an: Einer wird verwendet, falls die gewünschte Datenbank bereits existiert, der andere, falls zuerst eine neue Datenbank erstellt werden muss. Der zweite Konstruktor besitzt natürlich keinen Namen einer Datenbank als Argument. Dieser wird erst anschliessend der Methode *createDatabase()* bei deren Aufruf als Parameter übergeben.

Die anderen Methoden der Klasse erlauben es zudem, eine Verbindung mit der Datenbank aufzubauen, diese später wieder zu schliessen, eine Tabelle in einer Datenbank zu erstellen sowie Zeilen in eine Tabelle einzufügen. Benutzername, Passwort und Datenbank werden als Instanzvariablen gespeichert und zum Aufbau der Verbindung wieder benötigt. Das erstellen einer Datenbank oder Tabelle sowie das Einfügen von Zeilen erfolgt über den Aufruf von entsprechenden SQL-Statements.

⁴ DataRow ist eine eigene Klasse und wird im Kapitel 5.3 näher beschrieben

5.3 Die Klasse *DataRow*

Die Klasse *DataRow* ist sehr schlank gehalten und beinhaltet lediglich zwei Attribute (*sender* und *receiver*) vom Typ String, die dazugehörigen "getter-Methoden" und einen Konstruktor, der zwei Strings als Argumente erwartet, deren Werte den Instanzvariablen zugewiesen werden.

Diese Klasse hat folgende Existenzbegründung: Ziel war es, der Tabelle in der Datenbank eine beliebige Zeile über den Aufruf der Methode *addRow()* hinzufügen zu können. Der Methode sollte hierbei als Parameter ein Objekt übergeben werden, welches diese Zeile repräsentiert. Hierzu musste also zuerst eine neue Klasse erzeugt werden, um solche Objekte instanzieren zu können. Über die Accessor-beziehungsweise "getter-Methoden" kommt die *addRow()*-Methode schliesslich an die gewünschten Informationen des entsprechenden *DataRow*-Objektes heran. Müsste das Programm nun mehrere unterschiedliche Arten von Datenzeilen zur Verfügung stellen können, so könnte dies über ein Interface *DataRow* und mehrere Klassen gelöst werden, welche dieses Interface implementieren und somit eine bestimmte Art von Datenzeile repräsentieren.

5.4 Die Klasse *RdfDialog*

Der *RdfDialog* ist das eigentliche Graphical User Interface (GUI) der Applikation und beinhaltet somit sämtliche graphischen Elemente wie Frames, Buttons, Textfelder, und so weiter. Diese können mit dem visuellen GUI-Editor in Netbeans einfach und bequem via Drag and Drop eingefügt werden.

Der Dialog wurde in Anlehnung an das Tutorial "Creating a wizard console in NetBeans IDE 5.5" von Saleem (2007) erstellt. Das *mainPanel*, bei welchem das *CardLayout* gesetzt wurde, beinhaltet mehrere "cards", welche die einzelnen Frames des Dialogs repräsentieren. Diese wurden mit Zahlen von eins bis sechs nummeriert beziehungsweise benannt. So kann mithilfe der Methode *show()* der Klasse *CardLayout* und einem einfachen Zähler für die korrekte Abfolge der Frames im Dialog gesorgt werden.

Da gewisse Informationen in einem Frame eingegeben und zu einem späteren Zeitpunkt (sprich: bei einem späteren Aufruf der Methode *btnNextActionPerformed()*)

wieder abgerufen werden müssen, werden diese in den entsprechenden Instanzvariablen der Klasse *RdfDialog* (zwischen-)gespeichert. Ansonsten enthält die Klasse jedoch keinerlei Programmlogik.

5.5 Die Klasse *DirectoryFilter*

Wie in der Programmdokumentation erklärt, hat der Benutzer im fünften Schritt des Dialoges die Möglichkeit, eine der bestehenden Datenbanken via Dropdown-Menu auszuwählen. Um an die Namen der bestehenden Datenbanken zu gelangen, werden die Namen sämtlicher Unterverzeichnisse im Ordner *data* im MySQL-Hauptverzeichnis zuerst in einem String-Array gespeichert, womit anschliessend das Dropdown-Menu initialisiert werden kann. Das String-Array kann mit Hilfe der Methode *listFiles()* der Klasse *File* gefüllt werden. Als Parameter muss dieser Methode ein Objekt einer Klasse übergeben werden, welche das *FileFilter*-Interface implementiert. Dieses Interface deklariert die Methode *accept()*, welche den Wahrheitswert *true* zurückgeben sollte, falls der übergebene Parameter gewissen Bedingungen entspricht (andererseits *false*). Die Klasse *DirectoryFilter* implementiert diese Methode so, dass, wie es der Name bereits verrät, nur *true* zurückgegeben wird, falls es sich beim übergebenen File um ein Verzeichnis handelt. Somit gibt die Methode *listFiles()* bei Übergabe eines *DirectoryFilter*-Objekts die Namen sämtlicher Unterverzeichnisse in einem bestimmten Ordner zurück, was es schliesslich erlaubt, das Dropdown-Menu korrekt zu initialisieren.

5.6 Code Refactoring

Nach der Fertigstellung einer ersten Version von JRP wurde der Code mittels Refactoring nach Fowler (1999) so angepasst, dass allfällige spätere Änderungen daran auf einfachere Art und Weise vorgenommen werden können.

Da dieser Teil nicht zum Kernbereich dieser Arbeit zählt, ist dieser auch eher kurz und knapp gehalten. Folgende Refactorings nach Fowler (1999) sind während der Arbeit unter anderem zur Anwendung gekommen:

- **Move Method:** In der ersten Version des Programms war es teilweise der Fall, dass gewisse Klassen Dienste anboten, welche eigentlich andere Klassen hätten anbieten sollen. Bei solchen Fällen wurde die Methode in die entsprechende Klasse verschoben und natürlich die Aufrufe entsprechend angepasst.
- **Move Field:** Hier gelten die gleichen Argumente wie beim verschieben einer Methode in eine andere Klasse. Häufig war es der Fall, dass beim verschieben einer Methode auch gleich gewisse Variablen in die entsprechende Klasse verschoben werden mussten.
- **Rename Method:** Das Umbenennen von Methoden (oder auch von Variablen) kam während der Entwicklung von JRP sehr häufig vor. Gemäss Glinz und Gall (2006) ist die Wahl von Namen wesentlich für das Verständnis eines Programms, was sich während dieser Arbeit sehr oft bestätigt hat. Fowler (1999) führt weiter aus, dass der Name einer Methode deren Verwendungszweck offenlegen sollte. Dies ist vor allem dann sehr praktisch, wenn man selbst programmierte Methoden zu einem späteren Zeitpunkt wiederverwenden muss.
- **Add Parameter:** Auch dies kam sehr häufig vor. Einen Parameter musste einer Methode zum Beispiel dann hinzugefügt werden, wenn dieser entscheidende Informationen fehlten, die beim Aufruf übergeben werden können.
- **Remove Parameter:** Dieses Refactoring kam zum Beispiel zum Einsatz, nachdem die Datenbank dahingehend angepasst wurde, dass der Primary Key via Auto-Inkrement gesetzt wird, weshalb der *addRow()*-Methode kein Integer mehr als ID übergeben werden musste und der Parameter somit entfernt werden konnte.

Natürlich ist dies nur eine kleine Auswahl der am meisten verwendeten Refactorings. Eine vollständige Auflistung aller verwendeten Methoden würde an dieser Stelle zu weit greifen. Jedoch soll dieser Abschnitt trotzdem als kleiner Überblick über einen Teilschritt des gesamten Arbeitsprozesses dienen.

5.7 Einschränkungen

Neben der mehrfach erwähnten Funktionalität, die das Programm aufweist, sind natürlich auch Einschränkungen zu machen. In diesem Abschnitt wird kurz auf zwei zentrale Einschränkungen für JRP eingegangen:

1. **Speicherung von mehrwertigen Attributen:** RDF lässt, wie in der Einführung in Kapitel 2 erwähnt, mehrwertige Attribute in Form von Bags, Sequenzen oder Alternativen zu. In einer Datenbank können solche Attribute natürlich nicht auf die gleiche Art und Weise gespeichert werden. In der jetzigen Version kann JRP keine Bags behandeln. Dafür gäbe es jedoch unterschiedliche Arten von Möglichkeiten. Nehmen wir an, eine Email wird an mehrere Empfänger geschickt. In RDF werden diese in einem Bag zusammengefasst. Eine Möglichkeit, dies korrekt in einer Datenbank abzubilden, wäre, eine Zeile pro Empfänger einzufügen, wobei natürlich der Absender immer der gleiche bleibt. Dies hat jedoch den Nachteil, dass die Datenbank dadurch ziemlich stark "aufgeblasen" wird. Abbildung 4 zeigt dies schematisch auf.

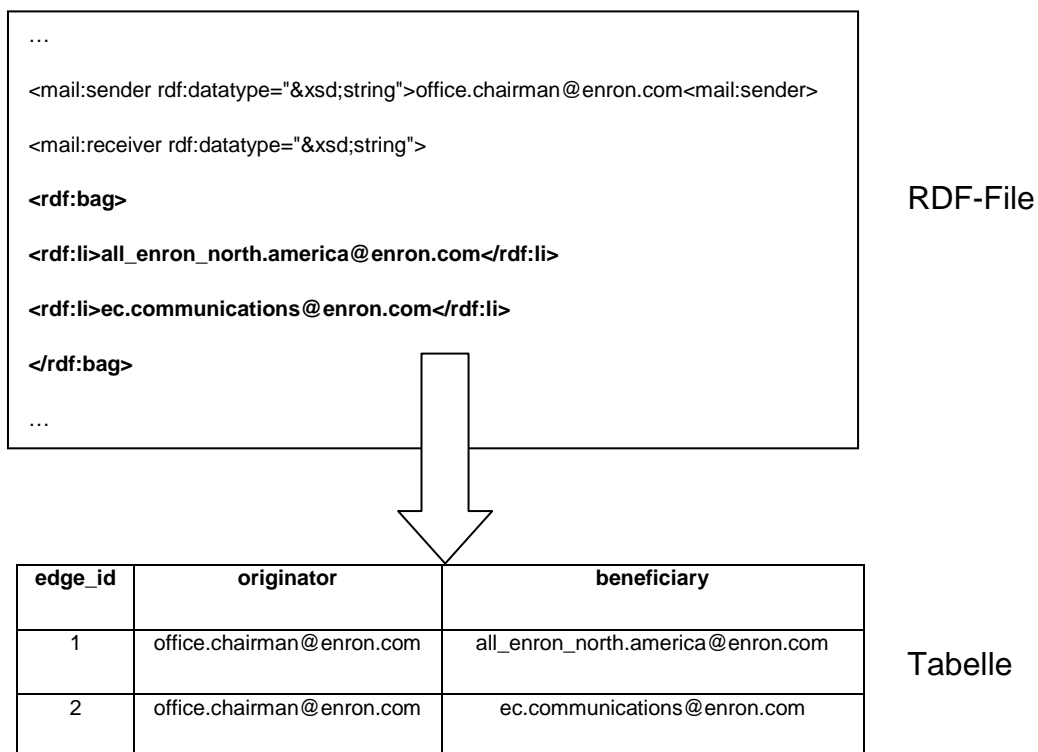


Abbildung 4: Lösung zur Speicherung mehrwertiger Attribute (Daten: Urs Wagner (2007), RDF Enron Email Datensatz)

2. **Fehlende Tests mit grossen Datenmengen:** Aufgrund der Tatsache, dass der vorliegende Enron Email Datensatz im RDF Format syntaktisch fehlerhaft⁵ war, konnte JRP leider nie mit grossen Datenmengen getestet werden. Somit lässt sich bis zum jetzigen Zeitpunkt leider noch nichts über die Skalierbarkeit des verwendeten RDF Parsers sagen.

Diese Einschränkung ist zwar nur temporär zu machen, würde jedoch ein späterer Test mit grossen Datenmengen keine zufriedenstellenden Ergebnisse liefern, käme das Skalierbarkeits-Problem als weitere Einschränkung hinzu.

⁵ Jena hat sich unter anderem an mehreren Stellen über nicht geschlossene Tags beklagt. Ein Beispiel dafür ist die folgende Fehlermeldung, welche beim Versuch, ein Modell mit Jena zu erstellen, aufgetreten ist:

The element type "mail:sender" must be terminated by the matching end-tag "</mail:sender>".

6 Testing

Dieser Abschnitt soll einen kurzen Überblick vermitteln, wie die Korrektheit des Programms überprüft wurde. Der erste Teil befasst sich mit den verwendeten Daten, der zweite Teil mit den Test-Methoden.

6.1 Testdaten

Als Testdatensatz lag anfänglich die RDF Form des Enron Email Datensatzes von Urs Wagner (2007) vor. Da diese Version jedoch syntaktisch fehlerhaft war, konnten bis jetzt keine Tests mit dem gesamten Datensatz durchgeführt werden. Zwecks Überprüfung der Korrektheit des Programmes wurden einzelne syntaktisch korrekte Abschnitte herauskopiert und in einem neuen File zusammengefügt. Dadurch konnte zumindest ein Eindruck vermittelt werden, dass das Programm die richtigen Daten ausliest und in der gewünschten Form in der Datenbank abspeichert.

Um die Funktionalität auch noch mit einem anderen Datensatz zu testen, wurde das File "vc-db-1.rdf" benutzt, welches auch für das SPARQL-Tutorial (<http://jena.sourceforge.net/ARQ/Tutorial/query1.html>) verwendet wird. Dieser Datensatz beinhaltet gemäss Hewlett-Packard (2005) eine Anzahl von vCard-Beschreibungen für verschiedene Leute. So sind in dem File einerseits der volle Name sowie der Name aufgeteilt in Vor- und Nachname gespeichert. Auf semantischer Ebene ist dieser Datensatz für die Zwecke von JRP natürlich nicht sehr brauchbar, da er keine Transaktionen zwischen verschiedenen Parteien beinhaltet. Zur Überprüfung der Auswahl der korrekten Werte und deren Speicherung in der Datenbank war er jedoch trotzdem hilfreich.

6.2 Testmethoden

Bevor natürlich das gesamte Programm getestet werden konnte, war es notwendig, die einzelnen Funktionen auf ihre Richtigkeit zu prüfen. Es wurde deshalb mit einer *TestDriver*-Klasse gearbeitet, mit der immer wieder gewisse neue Methoden getestet werden konnten. Um nachvollziehen zu können, was genau während dem Ablauf des Programmes (*TestDriver*) geschieht, wurde einerseits der Debugging-Modus mit Breakpoints verwendet, andererseits Bildschirmausgaben an bestimmten Orten im Code platziert. Bei kleineren Methoden konnte über die richtige Textausgabe auf die korrekte Arbeitsweise derselben geschlossen werden.

Eine andere und etwas formale Möglichkeit, die Korrektheit von Methoden zu testen, wären JUnit-Tests gewesen. Diese automatisierte Form des Testing wäre jedoch mit einem gewissen Aufwand verbunden gewesen, welcher vermutlich für ein solches, eher kleineres Projekt wohl eher unangebracht gewesen wäre.

7 Probleme / Lessons Learned

Während des gesamten Entwicklungsprozesses von JRP sind auch immer wieder Probleme aufgetaucht. Auf zwei davon wird an dieser Stelle noch kurz eingegangen.

1. **Probleme bei der Arbeit mit Sesame:** Anfänglich wurde versucht, RDF Files mit Sesame einzulesen und zu verarbeiten, was jedoch schlussendlich nicht gelang. Es ging dadurch ziemlich viel Zeit verloren, bevor schlussendlich der Entscheid fiel, mit Jena weiterzuarbeiten.
2. **Fehlerhafter Testdatensatz:** Aufgrund des syntaktisch fehlerhaften RDF Enron Email Datensatz war es nicht möglich, mit JRP Tests mit grossen Datenmengen durchführen zu können, weshalb im Bezug auf die Skalierbarkeit des Programmes bis jetzt noch keine Aussage gemacht werden kann. Dieses Problem wurde etwas entschärft, indem verschiedene syntaktisch korrekte Abschnitte des Datensatzes zu einem neuen Datensatz zusammengefügt wurden.

Im Grossen und Ganzen war diese Arbeit jedoch sehr lehrreich. Die Lehren, welche daraus für kommende Aufgaben gezogen werden können, sind unter anderem die folgenden:

- Für Tasks wie zum Beispiel die Auswahl einer Library zum Parsen von RDF Daten sollte man sich von Anfang an genügend Alternativen offen halten. Konzentriert man sich von Beginn weg auf eine einzige Lösung, so läuft man erstens Gefahr, bei einem Misserfolg ziemlich viel Zeit zu verlieren, und zweitens allenfalls nicht die beste Lösung ausgesucht zu haben.
- Für das Testing und die Überarbeitung des Codes sollte von Beginn weg genügend Zeit eingeplant werden. Geht man davon aus, dass das Programm bereits beim ersten Testdurchlauf ordnungsgemäss funktioniert, so läuft man Gefahr, keine Zeit für allfällige, doch noch nötige Korrekturen am Code einzuplanen.
- Tests sollten ausreichend sein und die korrekte Arbeitsweise des Programms garantieren können. Ein einziger korrekter Programm-Durchlauf ist hierfür zum Beispiel wohl kaum ausreichend.

8 Fazit

Der Java RDF Parser (JRP) ist ein Programm zum Einlesen von Files im RDF Format und zum Extrahieren von transaktionsbasierten Daten, die anschliessend in einer MySQL-Datenbank gespeichert werden können. Um die Daten einlesen und verarbeiten zu können, wird eine Java Library namens Jena verwendet. Wie in dieser Arbeit aufgezeigt, eignet sich diese für die Zwecke von JRP besser als die Alternativen VRP oder Sesame.

Die Datenbank weist ein sehr einfaches Schema auf und speichert in einer einzelnen Tabelle sowohl Sender als auch Empfänger einer Transaktion und weist dieser eine eindeutige Transaktionsnummer zu.

Das ganze Programm enthält Klassen für die Datenbank-Anbindung und Manipulation (*DatabaseManager*), das Einlesen und Verarbeiten von RDF Datensätzen (*RdfParser*) sowie die graphische Benutzeroberfläche (*RdfDialog*). Zwei weitere kleine Klassen dienen als Repräsentation einer Zeile in der Tabelle der Datenbank (*DataRow*) und zum Herauslesen von Unterverzeichnissen in einem bestimmten Ordner auf einem Rechner (*DirectoryFilter*).

Die Richtigkeit des Programms wurde mit realen Testdaten überprüft. Ein TestDriver diente zwischendurch immer wieder als Instrument, um die Funktionalität neuer Methoden zu testen. Leider kann bis zum jetzigen Zeitpunkt noch keine Aussage über die Skalierbarkeit des Parsers gemacht werden, da Tests mit grossen Datenmengen aufgrund des syntaktisch fehlerhaften RDF Enron Email Datensatzes nicht möglich waren.

Einschränkend muss zum Schluss gesagt werden, dass JRP keine mehrwertigen Attribute behandeln kann (zum Beispiel bei mehreren Empfängern). Eine mögliche Lösung dafür wäre, eine Zeile pro Empfänger in der Tabelle zu speichern, wodurch jedoch die Datenbank ziemlich stark aufgeblasen würde.

Literaturverzeichnis

- Aduna. (2007). *openRDF.org ...home of Sesame*. Abgerufen am 18. August 2007 von <http://www.openrdf.org/download.jsp>
- Axmark, D. (1999). MySQL Introduction. *Linux Journal, Volume 1999 , Issue 67es .*
- Baldwin, R. G. (2004). *Using JDBC with MySQL, Getting Started*. Abgerufen am 16. August 2007 von <http://www.developer.com/java/data/article.php/3417381>
- Dong, J. S., Lee, C. H., Lee, H. B., Li, Y. F., & Wang, H. (2004). A combined approach to checking web ontologies. *Proceedings of the 13th international conference on World Wide Web , 714 - 722.*
- Forschungsgruppe Requirements Engineering. (2005). Entwicklungsrichtlinien für Java-Software. Institut für Informatik, Universität Zürich.
- Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional.
- Glinz, M., & Gall, H. (2006). Software Engineering, Kapitel 6, Systematisches Programmieren: Lesbare und änderbare Programme schreiben (Skript zur Vorlesung). Zürich.
- Hewlett-Packard. (2005). *SPARQL Tutorial*. Abgerufen am 18. August 2007 von <http://jena.sourceforge.net/ARQ/Tutorial/index.html>
- Konchady, M. (1998). An Introduction to JDBC. *Linux Journal, Volume 1998 , Issue 55es .*
- Lüll, J. (2005). *Analytische Betrugserkennung - Erstellen einer Basis zur Anwendung von Data Mining-Methoden für die Bekämpfung internen Betrugs in Finanzinstituten (Diplomarbeit)*.
- McBride, B. (3. März 2007). *An Introduction to RDF and the Jena RDF API*. Abgerufen am 15. August 2007 von http://jena.sourceforge.net/tutorial/RDF_API/index.html
- Saleem, U. (2007). *Creating a wizard console in NetBeans IDE 5.5*. Abgerufen am 18. August 2007 von <http://www.netbeans.org/kb/55/wizard-tutorial-uzi.html>
- Selçuk Candan, K., Liu, H., & Suvarna, R. (July 2001). Resource description framework: metadata and its applications. *ACM SIGKDD Explorations Newsletter, Volume 3, Issue 1 , S. 6-19.*
- Wagner, U. (2007). RDF Enron Email Datensatz. Zürich.
- Xu, J. (2004). Die Evaluierung von Java RDF Parseern. *Diplomarbeit .* Frankfurt.

Appendix

A *Programmdokumentation*



Abbildung 5: JRP Logo

Das Programm JRP bietet dem Nutzer die Möglichkeit, ein RDF File seiner Wahl einzulesen und die gewünschten Informationen durch Auswahl der entsprechenden Tags in einer einfachen Datenbank zu speichern. Nachfolgend wird die Handhabung des Tools erklärt.

Starten des Programms

JRP benötigt keine Installation und kann direkt durch Ausführen der Datei `Java_RDF_Parser.jar` im Folder `dist` gestartet werden. Um das Programm ausführen zu können, sollte auf dem entsprechenden Rechner eine aktuelle Java Runtime Environment (JRE ab Version 1.5) installiert sein.

Allgemeine Bedienungshinweise

JRP ist sehr einfach zu bedienen. Mit den drei Buttons im unteren Bereich des Fensters kann man bequem vorwärts ("Next") und rückwärts ("Back") durch den Dialog navigieren oder diesen beenden ("Cancel"). Möchte man das Programm verlassen, so muss die Anfrage "Do you really want to quit?" mit "Yes" bestätigt werden.

Welcome to JRP

Nach dem Starten des Programms erscheint folgender Begrüßungsbildschirm, welcher weiter keine Funktionalität aufweist. Nach einem Klick auf den Button "Next" erscheint das nächste Fenster.

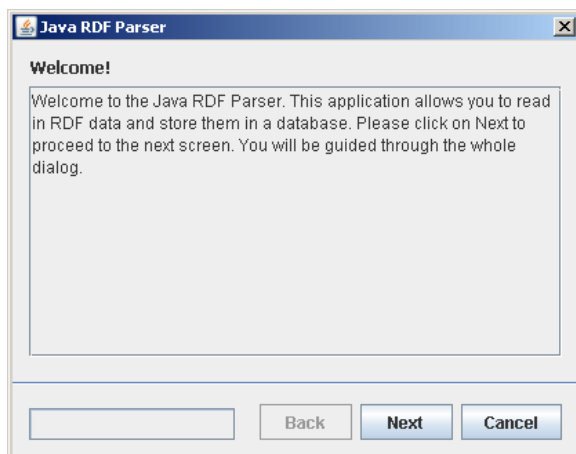


Abbildung 6: Welcome to JRP

Auswahl des RDF Input Files

In diesem Schritt wird das Eingabe-File ausgewählt, das von JRP geparkt werden soll.

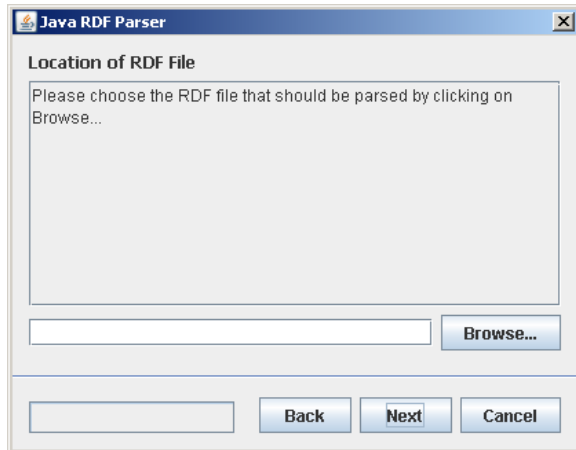


Abbildung 7: Auswahl des RDF Input Files

Nach einem Klick auf den Button "Browse" öffnet sich ein File-Browser, mit dem das gewünschte RDF File einfach und bequem gesucht und ausgewählt werden kann.

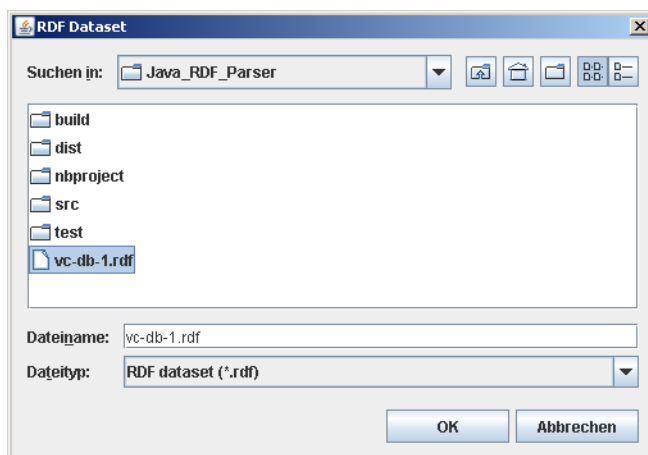


Abbildung 8: Der File-Browser

Durch einen Klick auf die Schaltfläche "OK" oder einen Doppelklick auf das File wird der entsprechende Pfad in das dafür vorgesehene Textfeld übernommen.

Selbstverständlich könnte dieser auch manuell hineingeschrieben werden. Mit "Next" gelangt man anschliessend einen Schritt weiter.

Eingabe der MySQL-Einstellungen

Nun müssen die MySQL-Einstellungen erfasst werden. Wurde bei der Installation von MySQL nichts verändert, so lautet der Username standardmässig *root* und Passwort muss keines eingegeben werden (in diesem Falle das Feld leer lassen).

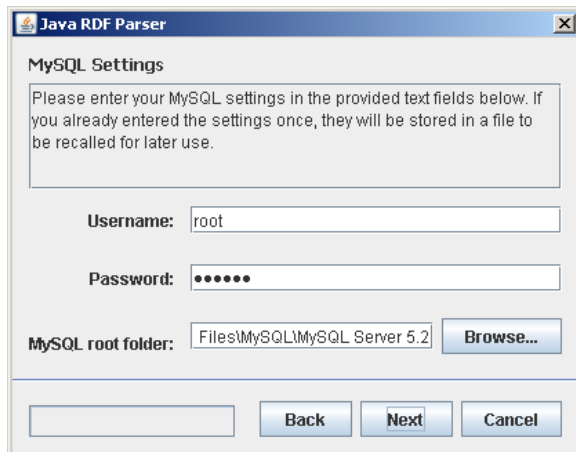


Abbildung 9: Eingabe der MySQL-Settings

Mit der Schaltfläche "Browse..." öffnet sich ein Folder Browser, mit dem das MySQL-Hauptverzeichnis auf dem Rechner ausgewählt werden kann. Dazu muss der Order angeklickt und die Auswahl mit "OK" bestätigt werden.

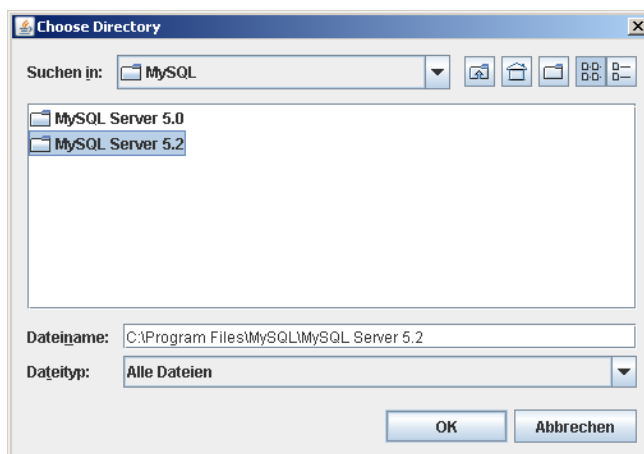


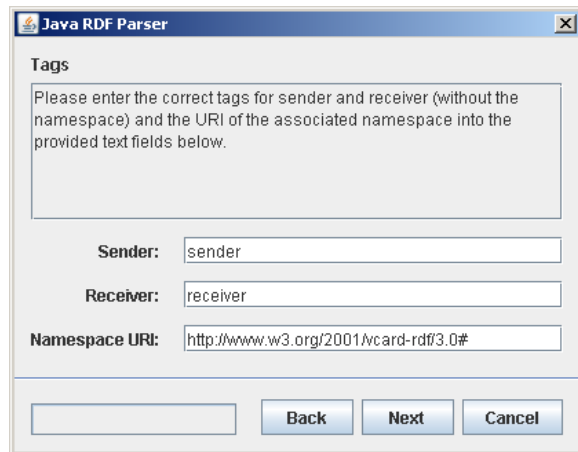
Abbildung 10: Der Folder Browser

Der Folder-Browser sieht optisch gleich aus wie der File-Browser, lässt im Unterschied dazu jedoch nur die Auswahl von Ordnern zu.

Die Auswahl eines Orderns wird nach der Bestätigung in das entsprechende Feld in der Maske übernommen, worauf mit "Next" fortgefahren werden kann.

Auswahl der Tags

Im nächsten Schritt müssen die Tags für Sender und Empfänger angegeben werden. Dazu werden die entsprechenden "local names" (also ohne namespace) in die dafür vorgesehenen Textfelder (*Sender* und *Receiver*) geschrieben. Im untersten Feld (*Namespace URI*) gibt man den URI des entsprechenden Namespaces ein. Dieser ist im RDF File unter `xmlns:<Name des Namespaces>` zu finden.



The screenshot shows a dialog box titled "Java RDF Parser" with a close button in the top right corner. The dialog has a section titled "Tags" with a text area containing the instruction: "Please enter the correct tags for sender and receiver (without the namespace) and the URI of the associated namespace into the provided text fields below." Below this are three text input fields: "Sender:" with the value "sender", "Receiver:" with the value "receiver", and "Namespace URI:" with the value "http://www.w3.org/2001/vcard-rdf/3.0#". At the bottom of the dialog, there is an empty text field followed by three buttons: "Back", "Next", and "Cancel".

Abbildung 11: Auswahl der Tags

Nach der Bestätigung mit der "Next"-Schaltfläche kann es je nach Grösse des Files einige Zeit dauern, bis der nächste Screen erscheint, da das File nun nach den eingegebenen Kriterien geparkt und der Output entsprechend vorbereitet wird. Dies kann unter Umständen viel Rechenaufwand in Anspruch nehmen.

Die Auswahl der Output-Datenbank

Als letzter Schritt muss noch die gewünschte Output-Datenbank ausgewählt werden. Hierbei bestehen zwei Möglichkeiten.

1. **Die gewünschte Datenbank existiert bereits?** – Existierende Datenbanken können via Dropdown Menu ausgewählt werden. In diesem Falle werden die entsprechenden Einträge an die bereits bestehenden angefügt, ohne dass diese dabei verloren gehen.

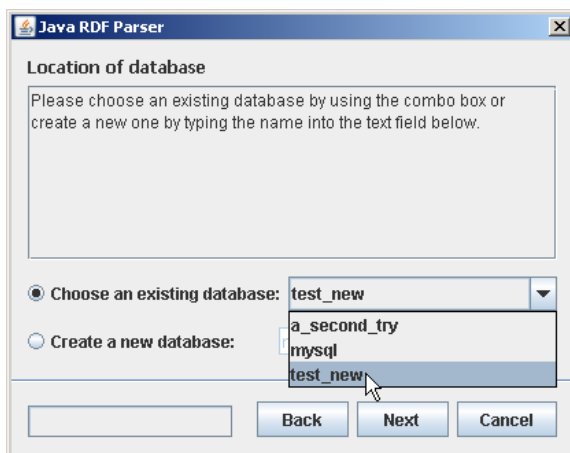


Abbildung 12: Auswahl einer bestehenden Datenbank

2. **Es muss eine neue Datenbank erstellt werden?** – Neue Datenbanken können erstellt werden, indem der gewünschte Name der Datenbank in das dafür vorgesehene Textfeld eingegeben wird.

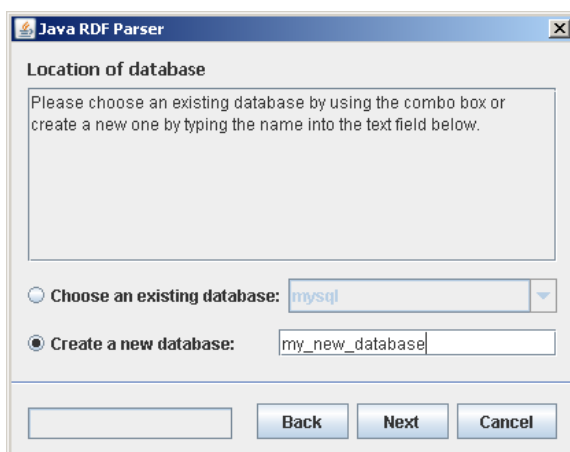


Abbildung 13: Erzeugung einer neuen Datenbank

Die gewünschte Option kann via Radio-Button ausgewählt werden. Somit ist immer nur eine Output-Variante pro Durchgang möglich.

Ende des Dialogs

Nach der erfolgreichen Ausgabe der ausgewählten Daten in die entsprechende Datenbank erscheint ein letzter Screen, der dies auch sogleich mit der entsprechenden Meldung bestätigt. Mit einem Klick auf die Schaltfläche "Finish", wird der Dialog beendet und damit das Programm geschlossen.

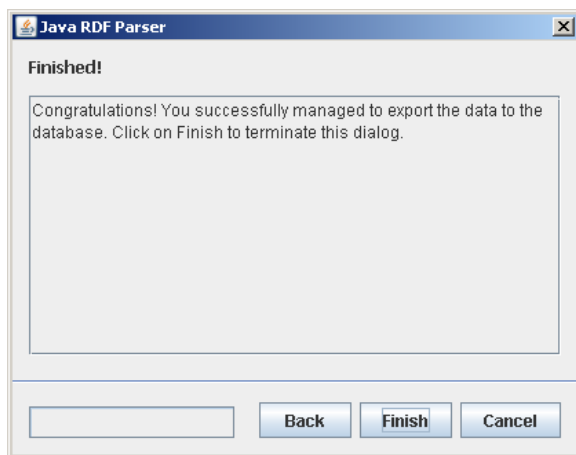


Abbildung 14: Ende des Dialogs

