Exposé einer Studienarbeit zur Implementation von DDDquery

Viktor Rosenfeld*

23. November 2007

1 Hintergrund

Im Rahmen des Projekts *DeutschDiachronDigital* [1] wurde von Thorsten Vitt die Anfragesprache DDD*query* [2] erstellt. DDD*query* basiert auf XPath und enthält weitere Sprachelemente, die sich aus der Verwendung als linguistische Anfragesprache ergeben:

- zusätzliche Achsen, um in linguistischen Anfragen oft vorkommende Navigationsschritte zu vereinfachen,
- Knotenmarkierungen, um die Ergebnismenge einzuschränken,
- Variablenzuweisungen, um Joins innerhalb von Pfadausdrücken zu ermöglichen.

*rosenfel@informatik.hu-berlin.de

2 Zielsetzung

Ziel dieser Studienarbeit ist die Implementierung eines Compilers von DDD*query* nach SQL, um DDD*query* erstmals praktisch nutzbar zu machen. Dabei soll eine DDD*query*-Anfrage in ein einzelnes SELECT-Statement umgewandelt werden. Neben einer Proof-of-Concept-Implementation soll die Performanz der erzeugten SQL-Anfragen überprüft und ggf. optimiert werden.

3 Umsetzung

3.1 Proof-of-Concept-Implementation

Thorsten Vitt hat im Rahmen der Entwicklung von DDD*query* die Implementation bereits angefangen. Neben einem mit JavaCC entwickelten Parser existieren außerdem die Infrastruktur zur Bearbeitung und Normalisierung der vom Parser gelieferten Syntaxbäume, sowie die Umwandlung eines SQL-Resultset in das DDD*query*-Ausgabeformat gXDF. Die Übersetzung von DDD*query*-Sprachelementen nach SQL ist ebenfalls angefangen; sie ist jedoch teilweise fehlerhaft und wird deswegen nicht benutzt werden.

Insbesondere müssen folgende Sprachelemente implementiert werden:

- die Achsen: `aligned`, `alignment`, `ancestor`, `ancestor-or-self`, `attribute`, `child`, `contained`, `contained-element`, `containing`, `containing-element`, `descendant`, `descendant-or-self`, `element-span`, `following`, `following-sibling`, `immediately-following`, `immediately-following-sibling`, `immediately-preceding`, `immediately-preceding-sibling`, `layer`, `matching-element`, `overlapping-following`, `overlapping-preceding`, `parent`, `preceding`, `preceding-sibling`, `prefix`, `sibling`, `suffix`, `whole-text`
- die Knotentests: `element()`, `attribute()`, `span()`,
- die Knotentests zur Textsuche: `exact-match()`, `re-match()`, `wild-match()`
- die Verwendung von Vergleichen innerhalb von Prädikaten,
- die Verwendung der Funktion `count`[1] innerhalb von Prädikaten,
- Markierungen und Variablenzuweisungen,
- Verknüpfungen von Pfadausdrücken mittels `und` und `oder`.

Aufgrund der von Thorsten Vitt geleisteten Vorarbeit beschränkt sich diese Studienarbeit auf die Imlementation eines Visitors, der den Syntaxbaum einer DDD*query*-Anfrage

[1] count wird für das Mapping von ANNIS-QL nach DDD*query* benötigt.

mittels Tiefensuche durchläuft und iterativ einen SELECT-Statement erstellt. Des Weiteren sind möglicherweise Anpassungen an dem Code von Thorsten Vitt erforderlich (Bugfixe).

Zur Implementation gehören auch eine ausführliche Dokumentation der Umwandlung von DDD*query*-Konstrukten in entsprechende SQL-Konstruktue, sowie Tests des neu entwickelten Codes mittels JUnit.

Nicht implementiert werden folgende Sprachelemente:

- Aggregationsfunktionen als Pfadschritte (siehe Abschnitt 7.4.8 in [2])[2],
- Reguläre Pfadausdrücke (siehe Abschnitt 7.4.15 in [2]).

3.2 Optimierung

Zur Optimierung der Bearbeitungsgeschwindigkeit existieren bereits einige Vorschläge. So können zum Beispiel die Tabellen `element` und `span` zusammengefasst werden, um die zwischen diesen Tabellen recht häufig vorkommenden Joins zu sparen. Da zwischen `span` und `element` eine $1 : N$-Beziehung herrscht, erhöht sich damit zwar die Redundanz in der Datenbank. Allerdings soll die Ausführgeschwindigkeit einer DDD*query*-Anfrage Vorrang gegenüber einer redundanzfreien Speicherung in der Datenbank haben.

Um die Suche im Text zu beschleunigen, können die Textinhalte der Spans in einer Tokentabelle materialisiert werden. Gegebenenfalls können auch umfassendere Satzelemente, wie Chunks oder Phrasen, zusätzlich materialisiert werden, um schnellere Joins über Textinhalte zu ermöglichen.

Zur weiteren Verbesserung der Bearbeitungsgeschwindigkeit sind Messungen notwendig. Kandidaten für Optimierungen sind die Textsuche, sowie komplexere Achsen (z.B. `contained-element`).

Literatur

[1] DeutschDiachronDigital. `http://www.deutschdiachrondigital.de/`, angeschaut am 20.11.2007.

[2] Thorsten Vitt. DDD*query* – Anfragen an komplexe Korpora. Master's thesis, Humboldt-Universität zu Berlin, 2005.

[2] Funktionen als Pfadschritt passen nicht auf das Ergebnisschema.