Ontologiespeicherung in Datenbanken im Kontext natürlichsprachlicher Dialogsysteme

Nils Weber
Christoph Eigenstetter
Markus Berg
Antje Düsterhöft

Hochschule Wismar
University of Applied Sciences
Technology, Business and Design

In vielen Anwendungen ist es notwendig komplexe Sachverhalte vorzuhalten um Eigenschaften, Zustände und Beziehungen zwischen den Objekten einer Domäne herzustellen und weiterführende Informationen abzuleiten. Hierfür eigenen sich besonders gut Ontologien und die Konzepte des Reasoning. Im Umfeld natürlichsprachlicher Dialogsysteme ist es von entscheidender Bedeutung möglichst umfassend Kontextinformationen und deren Zusammenwirken strukturiert zu verwalten, um daraus Informationen für die Dialoggenerierung zu gewinnen[9]. Der praktische Einsatz von Ontologien zeigte erhebliche Leistungsunterschiede zwischen den existierenden Knowlegde Base Systemen. Dieser Artikel stellt die Vorteile und Nachteile der wichtigsten Softwarelösungen gegenüber und gibt einen Überblick über die charakteristischen Eigenschaften der konkurierenden Produkte.

1 Einführung

Das „Natural Language Laboratory“[1] der Hochschule Wismar beschäftigt sich mit der computergestützten Verarbeitung von menschlicher Sprache. Dazu gehören u.a. Themengebiete wie Spracherkennung, Sprachsynthese, die semantische Interpretation von sprachlichen Äußerungen, das Führen von natürlichsprachlichen Dialogen sowie das Design von IVR-Systemen[2].
Der Schwerpunkt der aktuellen Forschungsprojekte liegt auf der Unterstützung des Anwenders mit Hilfe von flexiblen natürlichsprachlichen Dialogen in den verschiedensten Domänen. Um einen Sprachdialog zu führen, ist es notwendig eine Vielzahl unterschiedlicher Prozesse zu beherrschen, u.a. die semantische Interpretation von sprachlichen Äußerungen. Für diesen Zweck bieten Ontologien eine hervorragende Basis, da sie die Annotation von gespeicherten Daten mit Semantik unterstützen und durch logisches Folgern (Reasoning) die automatische Extraktion von implizit vorhandenen Daten ermöglichen.
Eines unserer aktuellen Forschungsprojekte, „TravelConsult im Dialog“[3], enthält Szenarien aus dem Bereich Tourismusmarketing. Ziel ist die Entwicklung einer zeitgemäßen Nutzerschnittstelle (natürlichsprachl. Dialog), die den Reisenden beim Auffinden und bei der Buchung von touristischen Angeboten komfortabel unterstützt. Da die Domäne „Tourismus“ im Vergleich zu anderen beschränkteren Szenarien vergleichsweise umfangreich ist und somit die zu verarbeitenden Ontologien ebenfalls eine hohe Komplexität erreichen, wurde es für uns notwendig, geeignete Speicherungsmöglichkeiten für große Ontologien zu finden, die den Projektanforderungen, besonders im Hinblick auf die Performance, genügen.

[1] http://www.et.hs-wismar.de/natlab
[2] IVR - **I**nteractive **V**oice **R**esponse Systems
[3] http://www.et.hs-wismar.de/natlab/travelconsult.html

Im Verlauf der Recherche wurden verschiedene Frameworks zur Speicherung von umfangreichen OWL-Ontologien getestet. Nahezu alle getesteten Lösungen benutzen im Hintergrund ein RDBMS für die eigentliche Speicherung und Indizierung der OWL-Daten. Zusätzlich müssen geeignete Anfragemechanismen, wie bswp. SPARQL[4], und adäquate Indizierungsmechanismen vorhanden sein. Zu den betrachteten Lösungen zählen: (a) Oracle 11g Semantic Technologies (b) OWLGres (c) Sesame (d) OWLIM (e) Jena.

2 Technologien

Technologien zur Speicherung und Verarbeitung von Semantik wurden in den letzten Jahren weiter entwickelt und werden immer ausgereifter. Dadurch werden sie in immer breiteren Anwendungsfeldern eingesetzt und die gespeicherten Ontologien und RDF-Graphen wachsen signifikant. Aus diesem Grund werden effiziente und skalierbare Technologien benötigt, die große Datenmengen verwalten können und nicht an die Größe des verfügbaren Arbeitsspeichers gebunden sind.
Eine Idee ist, bestehende relationale Datenbanksysteme zu nutzen, die ja bereits viele der oben genannten Anforderungen erfüllen, und diese um spezielle Funktionen für die Verarbeitung von Ontologien zu erweitern. Fast alle der in diesem Artikel vorgestellten Lösungen benutzen daher ein DBMS[5] entweder als reine Persistenzschicht, auf die über ein entsprechendes API[6] zugegriffen werden kann, oder die Funktionalität des API wird nativ in das DBMS integriert.
Ein weiterer Vorteil von Ontologien ist das automatische Folgern von implizitem Wissen mit Hilfe von Inferenzregeln. Das Reasoning wird bei vielen Lösungen direkt im RAM durchgeführt, so daß hier der Umfang des verwendeten Graphen durch die physikalische Größe des Arbeitsspeichers begrenzt ist. Gerade bei komplexen Ontologien, in der Größenordnung von mehreren Terrabytes, ist dies nicht akzeptabel. Zusätzlich gewinnt hier die Verarbeitungsgeschwindigkeit beim Laden, Anfragen und Schlussfolgern an Bedeutung.

2.1 Oracle 11g

Die Datenbank Oracle 11g stellt optional eine native RDF/RDFS/OWL-Unterstützung, die auf dem Oracle Spatial Network Data Model [2] basiert, zur Verfügung. Semantische Daten werden intern als RDF-Tripel auf das Network Data Model abgebildet. Dadurch wird das Laden, das Anfragen und das Inferencing von umfangreichen Graphen beschleunigt. Die Datenbank enthält eine native Inference-Engine, die OWL-DL, RDF, RDFS und benutzerdefinierte Regeln unterstützt [3]. Mit Hilfe von *Rulebases* können benutzerdefinierte Inferenzregeln definiert werden.
Semantische Daten werden mit Hilfe der *SEM_MATCH*-Funktion in einer SPARQL-ähnlichen Syntax abgefragt. SEM_MATCH kann mit regulären SQL-Statements kombiniert werden, so daß sich hier umfangreiche Möglichkeiten für die kombinierte Auswertung von semantischen und relationalen Daten ergeben (s. Abb. 1). Listing 1 findet bspw. Hotelnamen und Hotelort, die in Westmecklenburg liegen.

Listing 1: SPARQL-SQL Beispiel

```
SELECT x, h.ort
FROM TABLE(SEM_MATCH('(?x␣:in␣:Westmecklenburg)',
SEM_Models('owl'),
SEM_RULEBASES('OWLPRIME'),
SEM_ALIASES(SEM_ALIAS('','http://et.hs-wismar.de/natlab/owltest#')),null)),
Hotel h
WHERE h.name = x;
```

[4]SPARQL Protocol and RDF Query Language
[5]DBMS - **D**ata**b**ase **M**anagement **S**ystem
[6]API - **A**pplication **P**rograming **I**nterface

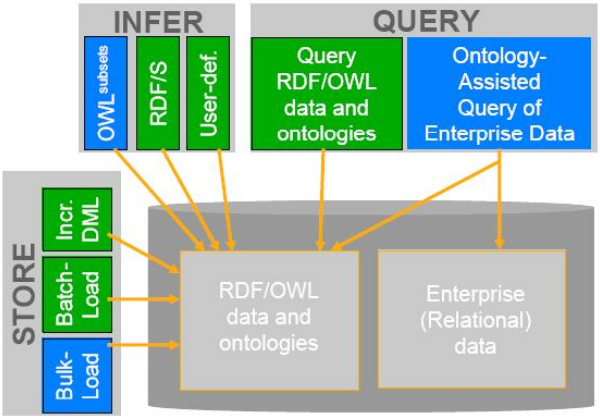

Abbildung 1: Oracle Database 11g RDF/OWL Semantic Data Store [1]

Es existieren Schnittstellen zu gängigen semantischen Design- und Analysewerkzeugen, wie TopBraid Composer[7] oder Protégé[8]. Einen weiteren Vorteil gerade beim Umgang mit umfangreichen Datenbeständen stellt die ausgereifte und skalierbare Oracle-Architektur dar, welche über das Oracle Network Data Model mitgenutzt wird.

2.2 OWLGres

OWLGres ist eine DL-Lite Reasoner Implementation, basierend auf dem DBMS PostgreSQL. Ziele von OWLGres sind das effiziente Abfragen einer skalierbaren, persistenten semantischen Datenbasis und das effiziente automatische Reasoning auf grossen RDF- und OWL-Datenbeständen. Derzeit werden Teile des OWL-DL Standards unterstützt. OWLGres ist als Java API implementiert und nutzt das DBMS PostgreSQL als Persistenzschicht.
Besonderen Wert wurde auf die Optimierung der Anfrage- und Reasoningperformance gelegt. Verbesserungen werden mit Hilfe von Query Simplification, Selectivity Optimization und der Zusammenfassung von mehreren SQL-SELECT-Statements mittels des UNION-Operators erreicht [4]. Für die Abfrage des Graphen wird SPARQL genutzt, wobei jedoch einige SPARQL-Konstrukte, wie bspw. OPTIONAL nicht unterstützt werden. Die Dokumentation ist noch lückenhaft, da sich das Projekt in einem frühen Stadium der Entwicklung befindet.

2.3 Sesame

Sesame[9] ist ein plattformunabhängiges Open Source Framework zum Speichern von RDF-Daten mit Unterstützung zur Anfrage und Inferencing von RDF Schema. Es kann im Zusammenspiel mit verschiedenen Speicherungskonzepten (Datenbanken, in-memory, Dateisystemen, Keyword Indexer) eingesetzt werden und stellt eine Vielzahl von Entwicklerwerkzeugen bereit. Verschiedene Anfragesprachen wie SeRQL und SPARQL sind über eine flexible API nutzbar.
Ein zentrales Konzept bei Sesame sind Repositories. Der Repostitory-Typ bestimmt die Art der Datenhaltung und hat Einfluss auf den Umfang des Inferencing. Über den sogenannten SAIL-Stack[10] werden alle relevanten Parameter gesetzt. Ohne Erweiterungen sind allerdings nur grundlegende Schlussfolgerungen möglich, die sich aus der Hierarchie ableiten lassen. Mit Hilfe des *Custom Inferencing* wird dem Nutzer eine Möglichkeit gegeben eigene transitive, symetrische oder inverse Properties zu definieren. Hierzu ist es notwendig den SAIL-Stack anzupassen und

[7] TopBraid Composer - `http://www.topquadrant.com`
[8] Protégé - `http://protege.stanford.edu/`
[9] Sesame - an Open Source RDF Framework, `http://www.openrdf.org/index.jsp`
[10] SAIL - **S**torage **A**nd **I**nference **L**ayer

ein gegeinetes Repository zu erstellen. Am Beispiel OWLIM soll der Zusammenhang zwischen Repositorytyp und Reasoning verdeutlicht werden.

2.4 OWLIM

OWLIM ist ein skalierbares semantisches Repository zur Speicherung, Integration und Analyse heterogener Daten. Dabei werden *Full RDFS*, eingeschränktes *OWL Lite* und *Horst* unterstützt. OWLIM ist als *Storage And Inference Layer* (SAIL) für Sesame implementiert und basiert auf der TREE-Engine[11]. Es werden die zwei Ausprägungen - *SwiftOWLIM* mit extrem schnellen Lade- und Inferenzzeiten und *BigOWLIM* für die Optimierung sehr großer Datenmengen unterschieden.
Bei der kommerziellen Version *BigOWLIM* ist die Skalierbarkeit das wesentlichen Kriterium. Hier werden die Daten nicht komplett im Speicher gehalten, sondern in Binärfiles abgelegt. Dieses Konzept bildet die Grundlage für das Handling von sehr großen Datenmengen. Anfrageoptimierung und spezielle Kriterien bei der Nutzung von *Equivalenzklassen* sollen verhindern, dass unnötige Inferenzen gebildet werden.
SwiftOWLIM dagegen setzt als in-memory-Version auf maximale Performanz. Basierend auf Hash-Table-Indizes werden Strategien zur Optimierung der Speicherung, Indizierung, Konsistenz und Integrität der Daten zur Verfügung gestellt. SwiftOWLIM gilt als schnellste RDF(S)- und OWL-Engine.

2.5 Jena

Jena[12] ist ein Open-Source Java Framework für die Entwicklung von semantischen Applikationen. Es stellt RDF und OWL APIs sowie eine SPARQL-Query-Engine zur Verfügung. Ursprünglich wurde Jena im Rahmen des „HP Labs Semantic Web Programme“[13] entwickelt. Es wird sowohl die in-memory-Speicherung von OWL Modellen als auch die persistente Speicherung in relationalen Datenbanken über das „Jena2 persistence subsystem“ (JPS) [7] unterstützt. JPS enthält einen Fastpath[14]-Algorithmus, welcher für Teile von SPARQL-Anfragen dynamisch SQL-Statements generiert, die dann direkt in der Datenbank bearbeitet werden. JPS unterstützt derzeit folgende DBMS: HSQLDB[15], MySQL, PostgreSQL, Apache Derby, Oracle und MS SQL-Server.
ARQ ist eine Implementierung der SPARQL-Sprache für das Jena Framework. Es werden die gängigen SPARQL-Funktionalitäten abgebildet und einige zusätzliche Features, wie „GROUP BY“-Klauseln, property paths, LET Variablen und Sub-Selects integriert [10].

3 Vergleich

Derzeit steht eine beachtliche Anzahl von Systemen zur Speicherung und zur Verarbeitung von semantischen Datenbeständen zur Verfügung. Die meisten unterscheiden sich in wesentlichen Punkten, so daß ein jedes System verschiedene Charakteristika aufweist und somit für unterschiedliche Anforderungen geeignet ist. Aufgrund der stetig wachsenden semantischen Datenmengen spielt die Performance und Stabilität bei der Verarbeitung von umfangreichen Graphen zunehmend eine besondere Rolle. Daher wurden Methoden entwickelt, um die Performance, Skalierbarkeit und Stabilität von semantikverarbeitenden Systemen standardisiert zu messen. Der bekannteste dieser sogenannten Benchmarks ist LUBM [8], welcher an der Lehigh University in Bethlehem, USA, entwickelt wurde.

[11] TREE - **T**riple **R**easoning and **R**ule **E**ntailment
[12] `http://jena.sourceforge.net`
[13] HP Labs Semantic Web Programme - `http://www.hpl.hp.com/semweb/`
[14] Jena Fastpath Query Processing - `http://jena.sourceforge.net/DB/fastpath.html`
[15] HSQLDB - **H**yper**SQL** **D**ata**b**ase

Tabelle 1: LUBM Loading (Time in hours)

Software	**LUBM 50**	**LUBM 500**	**LUBM 1000**	**LUBM 50k**
Oracle 11g	0:13:29	3:20:00	6:23:00	k.A.
OWLGres	k.A.	k.A.	k.A.	k.A.
OWLIM	0:02:00 (Swift)	03:00:00	k.A.	40:00:00 (BigOWLIM)
Sesame (Native Store)	0:11:15	3:22:00	k.A.	k.A.
Jena	k.A.	k.A.	k.A.	k.A.

LUBM stellt eine Ontologiestruktur, einen Datengenerator, 14 standardisierte SPARQL-Anfragen sowie verschiedene Messparameter, wie Ladezeit, Repositorygröße und Antwortzeiten, bereit.
Mit Hilfe des Datengenerators können vergleichbare Testdaten generiert werden. Dabei wird auf eine festgelegte OWL-Struktur zurückgegriffen, die eine Universitätsdomäne repräsentiert. Die Testdaten enthalten dann 1 .. n Universitäten (LUBM n), wobei jede Universität 15 - 25 departments mit den dazugehörigen Daten (Studenten, Vorlesungen, Mitarbeiter etc.) enthält. Somit ergibt sich bspw. für eine LUBM 50 Ontologie die Anzahl von 6,8 Mio. Tripeln.
Mit LUBM können die Zeit, die für das Laden einer Ontologie in das System benötigt wird sowie die physische Größe des belegten Speicherplatzes gemessen werden. Um die Anfrageeffizienz eines Knowledge Base Systems zu bestimmen, wurden 14 genormte SPARQL-Anfragen entwickelt, die sich hinsichtlich ihrer Selektivität, Komplexität und der Hierarchiekomplexität unterscheiden. Außerdem existiert für jede Query eine definierte Antwortmenge, anhand derer sich die Vollständigkeit einer Systemantwort und somit auch die Reasoningqualität ermitteln läßt.
Zum jetzigen Zeitpunkt existieren nur sporadische Angaben der Entwickler/Hersteller in Bezug auf LUBM-Ergebnisse ihrer Produkte. Die in der Tabelle 1 aufgeführten Werte sind somit nicht vollständig und nicht direkt vergleichbar und sollen lediglich als grober Anhaltspunkt für die Leistungsfähigkeit der Systeme dienen. Sie wurden im Rahmen unserer Recherche gewonnen. Tabelle 1 enthält die Ladezeiten für LUBM Ontologien verschiedener Grösse.

4 Zusammenfassung

Die in diesem Dokument beschriebenen Techniken geben einen Überblick über die wichtigsten Speicherungsmöglickeiten von Ontologien und über die Vorteile und Nachteile der jeweiligen Werkzeuge im Anforderungsksontext. Im Bereich der Datenhaltung unterscheidet man grundsätzlich Systeme, bei denen die Daten komplett im Hauptspeicher gehalten werden und Systeme mit zusätzlichen Datenbanken oder Binärdateien. Ein weiteres Schlüsselkriterium zur Bewertung von Knowlege Base Systemen ist der Umfang des Reasonings, also die Fähigkeit implizite Informationen aus den ursprünglichen Daten abzuleiten und nutzbar zu machen.
In Abhängigkeit von den Anforderungen der jeweiligen Anwendungen müssen sich die Entwickler bei der Wahl des Knowledge Base Systems zwischen möglichst guter Performanz oder der Fähigkeit, mit großen Datenmengen umzugehen, entscheiden. Gerade der Umgang mit großen Datenmengen kann unter Umständen dazu führen, dass einige Systeme bei denen die Daten im Speicher gehalten werden, abhängig von der zu Verfügung stehenden Hardware nicht nuztbar sind. Das Hochladen besonders umfangreicher Ontologien und Anfragen auf den Datenbestand dauern gegebenfalls extrem lange oder führen sogar zu Abstürzen der Systeme.
Auf der anderen Seite stehen Anforderungen an die Reaktionsgeschwindigkeit der Systeme, wobei hier in-memory Systeme wesentlich effektiver arbeiten. Einschränkungen in der Wahl der geeigneten Systeme sind im Umfang der Reasoningfähigkeiten zu suchen.
Über die Anforderungen an die Leistungsfähigkeit hinaus können auch die Implementierungssprachen der API's die Entscheidung für ein System beeinflussen. So sind die meisten der Sy-

steme als Javabibliotheken erhältlich. Mit *dotsesame*[16] existiert ein Versuch der Imlementierung einer C#-Schnittstelle, die jedoch zum gegenwärtigen Zeitpunkt nur über die Nutzung der Java-Klassen aus der C#-Umgebung heraus möglich ist.
Ein weiterer Aspekt bei der Integration von Knowledge Bases in komplexe Anwendungen ist die Stabilität unter realen Bedingungen. Hierbei sind beispielsweise Multiuserumgebungen mit Parallelzugriffen unter Umständen problematisch, da diese zu Inkonsistenzen innerhalb des Datenbestands führen können. Umfangreiche praktische Erfahrungen und Testergebnisse liegen diesbezüglich allerdings nur für das Jena-Framework vor und sind somit nicht nicht auf alle Systeme zu beziehen.
Abschließend bleibt noch der Kostenfaktor zu erwähnen, welcher bei kommerziellen Systemen wie Oracle nicht unerheblich ist. Das für den Umgang mit sehr großen Datenmengen optimierte BigOWLIM ist ebenfalls nur in einer Testversion kostenlos, stellt aber als einziges System Konzepte für das Reasoning von über 3.3 Milliarden Statements und den Upload von bis zu 6.7 Milliarden Triples zur Verfügung.

Literatur

[1] Semantic Data Integration for the Enterprise. Oracle Corporation, 2007

[2] Oracle Spatial Network Data Model. Oracle Corporation, May 2005

[3] Zhe Wu, George Eadon, Souripriya Das, Eugene Inseok Chong, Vladimir Kolovski, Melliyal Annamalai, Jagannathan Srinivasan: Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules in Oracle. In Proc. of 24th International Conference on Data Engineering (ICDE), April 7-12, 2008, Cancun, Mexico

[4] Markus Stocker, Mike Smith: Owlgres: A Scalable OWL Reasoner. In Proc. of OWLED2008. Fifth International Workshop, Karlsruhe, Germany, 2008

[5] Atanas Kiryakov, Damyan Ognyanov, Dimitar Manov: OWLIM a Pragmatic Semantic Repository for OWL. In Proc. of Int. Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2005), WISE 2005, 20 Nov, New York City, USA.

[6] Brian McBride: Jena Implementing the RDF Model and Syntax Specification. In Proc. of The Second International Workshop on the Semantic Web (SemWeb'2001), Hongkong, May 1, 2001

[7] Wilkinson, Kevin; Sayers, Craig; Kuno, Harumi; Reynolds, Dave: Efficient RDF Storage and Retrieval in Jena2. HPL-2003-266

[8] Y. Guo, Z. Pan, and J. Heflin. LUBM: A Benchmark for OWL Knowledge Base Systems. In Journal of Web Semantics, Vol 3, Issue 2, 2005

[9] N. Weber, C. Eigenstetter, M. Berg, A. Düsterhöft: Natural language dialogs in home automation systems (eingereicht NLDB 2009)

[10] ARQ A SPARQL Processor for Jena. `http://jena.sourceforge.net/ARQ/`

[16] dotsesame-Projekt `http://sourceforge.net/projects/dotsesame`