
Bedarfsgerechte Subskription und Fusion von Informationskanälen mit Metadatenannotationen

von
Thorben Ole Heins

betreut von
Prof. Dr. rer. nat. habil. Ralf Möller
M. Sc. Atila Kaya

Technische Universität Hamburg-Harburg
Institut für Software Systeme (STS)

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel oder Quellen dazu verwendet habe.

Hamburg, den 28. Februar 2007
Thorben Ole Heins

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die zum Gelingen dieser Arbeit beigetragen haben:

Zu aller erst möchte ich Herrn Prof. Dr. Möller für die überaus interessante und anspruchsvolle Aufgabenstellung und die Unterstützung bei der Bearbeitung danken.

Herrn M. Sc. Atila Kaya danke ich für die überaus engagierte Betreuung dieser Bachelorarbeit und die zahlreichen und produktiven Gespräche und Diskussionen sowie die Anregungen und Verbesserungsvorschläge, ohne die diese Arbeit nicht entstanden wäre.

Frau Dipl.-Ing. Sylvia Melzer möchte ich ebenfalls für die Unterstützung bei der Bearbeitung danken.

Der Racer Systems GmbH & Co. KG danke ich für die Bereitstellung des RacerPro und die Hilfe bei Fragen, die bei der Arbeit mit dem RacerPro auftraten.

Meiner Familie und meiner Freundin möchte ich an dieser Stelle für die Unterstützung während meines Studiums danken.

Inhaltsverzeichnis

1	Einleitung	6
1.1	Motivation	7
1.2	Ziel der Arbeit	8
1.3	Gliederung der Arbeit	9
2	Grundlagen	10
2.1	Metadatenannotationen	11
2.2	Beschreibungslogiken	11
2.2.1	TBox	11
2.2.2	ABox	12
2.3	Informationskanal	12
2.4	Fusion	12
2.4.1	Domänen ABox	13
2.5	Subskription	16
3	Architektur	17
3.1	Übersicht	18
3.2	Use-Cases	19
3.3	Komponenten	19
3.3.1	Datenbank	19
3.3.2	Reasoner	20
3.3.3	Extraktions Tools	20
3.3.4	Subskriptions- und Fusionssystem	20
3.3.4.1	FusionModule	20
3.3.4.2	SubscriptionManager	21
3.3.4.3	SubscriptionClient	21
4	Implementierung	22
4.1	Datenbank	23
4.2	Reasoner	23
4.2.1	Racer Schnittstelle	24
4.3	Extraktions Tools	24
4.4	Subskriptions- und Fusionssystem	26

4.4.1	FusionModule	26
4.4.2	SubscriptionManager	27
4.4.3	SubscriptionClient	29
4.4.4	SFSSetupModule	29
5	Zusammenfassung und Ausblick	31
5.1	Zusammenfassung	31
5.2	Ausblick	32
A	Anhang	33
A.1	Beispiel: TBox	33
A.2	Beispiel: Regel	34
A.3	Beispiel: Zwei Interpretation ABoxen die eine Inkonsistenz hervorrufen	35
A.4	CD	36

Abbildungsverzeichnis

2.1	Multimodales Dokument (HTML Seite)	14
2.2	Multimodales Dokument (Aufteilung)	14
3.1	Die Systemarchitektur	18
3.2	Use-Case: Klientanwendung	19
3.3	Use-Case: Extraktions Tools	19
3.4	Entity Relationship Modell des Datenbank Schemas . . .	19
4.1	ABoxSubmitter	25
4.2	FusionModule	27
4.3	SubscriptionManager	28
4.4	SubscriptionClient	30

Kapitel 1

Einleitung

In Zeiten der Informationsvielfalt und des Internet wird es immer schwieriger für Benutzer, die für sie relevanten Informationen zu finden. Die bedarfsgerechte Beantwortung von Benutzeranfragen stellt zur Zeit immer noch eine große Herausforderung für die Forschung dar.

Bereits seit längerer Zeit beschäftigt sich die Informatik mit dem Problem der bedarfsgerechten Beantwortung von Benutzeranfragen. Die Forschung in diesem Gebiet hat bereits interessante Ansätze geliefert [1]. Im Kontext des Semantic Webs wird in dieser Arbeit ein Teil eines Ansatzes vorgestellt, der dieses Problem mit Hilfe von Ontologien zu lösen versucht.

1.1 Motivation

Heute spielt die bedarfsgerechte Beantwortung von Benutzeranfragen eine immer größere Rolle und es ist anzunehmen, dass dieser Aspekt in der Zukunft noch an Bedeutung gewinnen wird. Es ist davon auszugehen, dass es in naher Zukunft viele Anwendungen geben wird, die sich mit der Verwaltung und dem Auffinden von multimedialen Inhalten beschäftigen werden, die eine intelligentere Form der Wissensgewinnung (Information Retrieval) erfordern. Der Begriff der Intelligenz bezieht sich hier auf das Auffinden eines Zusammenhangs von Daten und deren Semantik.

Das *BOEMIE*-Projekt (Bootstrapping Ontology Evolution with Multimedia Information Extraction) beschäftigt sich mit der Beantwortung von zentralen Problemen in diesem Zusammenhang. Im Rahmen des Projektes sollen unter anderem die Weichen für eine automatisierte Wissensextraktion aus multimedialen (multimodalen) Dokumenten gestellt werden. Aber auch das Problem der Evolution der *Ontologien* stellt eine wichtige Aufgabe in diesem Projekt dar. So soll ermöglicht werden, dass Ontologien dynamisch erweitert werden. Dies ist ein großer Vorteil, da es in den derzeitigen, auf Datenbanken basierenden Lösungen für die Wissensspeicherung, sehr aufwändig ist, die Wissensbasis dynamisch zu erweitern [2].

Unter einer Ontologie versteht man in der Informatik ein formal definiertes System, in dem Begriffe und Beziehungen definiert werden. Die formale Definition ermöglicht unter anderem, Schlussfolgerungen aus diesen Definitionen zu ziehen (Reasoning) [3, 4, 5].

Unter Zuhilfenahme von Ontologien kann ein so genannter Reasoner bestimmte Dienste anbieten. Diese Dienste beinhalten die Anfragebeantwortung, Konsistenzprüfung sowie die Schlussfolgerung (Deduktion), aber auch die Abduktion. Einige dieser Dienste sollen in dieser Arbeit genutzt werden, um die bedarfsgerechte Subskription gewährleisten zu können.

Das System, das im Rahmen dieser Arbeit entwickelt wird, hat im wesentlichen zwei Aufgaben:

- Die Daten, die aus Dokumenten im Idealfall vollkommen automatisch extrahiert werden, sollen in einer Datenbank verwaltet werden und in einem Prozess, der so genannten Fusion, unter bestimmten Voraussetzungen der bisher vorhandenen Wissensbasis hinzugefügt werden.

Diese Dokumente können mehrere Modalitäten aufweisen. Das heißt, für ein Dokument können verschiedene Verfahren zur Informationsextraktion genutzt werden. Dies ist beispielsweise bei Bilddokumenten der Fall, die auch Text enthalten.

Im Zuge der Fusion soll das Wissen der verschiedenen Modalitäten gebündelt werden. Hieraus kann sich ein Wissensgewinn ergeben.

- Es sollen Subskriptionen von Benutzern bedarfsgerecht beantwortet werden können. Hierzu gehört insbesondere die direkte Anzeige der Inhalte in der Benutzeroberfläche beim Eintreffen neuer relevanter Daten in der Wissensbasis.

Bis jetzt haben sich verschiedene Arbeiten mit der Aufgabe beschäftigt, die Subskription von Daten durchzuführen. Allerdings gibt es bis jetzt keinen Ansatz, der in diesem Zusammenhang die Daten, die für das Reasoning irrelevant sind, wie zum Beispiel binäre Bilddaten, in einer Datenbank verwaltet und die Ergebnisse der Subskription anzeigefertig an die Klientanwendung ausliefert. Die Verbindung zwischen Klientanwendung, Reasoner und Datenbank wird im Rahmen dieser Arbeit erstmals mit den für die Architektur gewählten Komponenten prototypisch entwickelt.

1.2 Ziel der Arbeit

Das Ziel dieser Bachelorarbeit ist die Entwicklung eines Systems, das auf Grund von Subskriptionen Daten an Benutzer in einer Form ausliefert, die eine Darstellung ermöglicht. Eine Subskription ist der eine Teil des asynchronen Publish/Subscribe Kommunikationsparadigmas aus der Informatik [6]. Der andere Teil dieses Paradigmas, die Veröffentlichung (Publish), wird im Zuge der Fusion neuer Daten mit den bereits vorhandenen Daten in der Wissensbasis durchgeführt. Unter einer Fusion versteht man im Rahmen dieser Arbeit zum einen den Wissensgewinn durch die Kombination von verschiedenen Informationskanälen, die ein und denselben Inhalt beschreiben und zum anderen das Hinzufügen von Daten zu einer bereits vorhandenen, konsistenten Menge von Daten mit der anschließenden Veröffentlichung. Hierbei wird dafür Sorge getragen, dass die Konsistenz der Daten weiterhin gewährleistet ist.

Das in dieser Arbeit zu entwerfende System soll verschiedene Komponenten miteinander verknüpfen. Diese Komponenten sind ein Reasoner, eine Datenbank, eine Klientanwendung und so genannte Informationskanäle. Die Klientanwendung soll Anfragen an das System stellen können. Er wird im Rahmen dieser Arbeit prototypisch entwickelt. Das System soll in der Lage sein, neue Daten aus diesen Informationskanälen zu den bereits vorhanden Daten in der Wissensbasis mittels Fusion hinzuzufügen.

Im Rahmen des BOEMIE-Projektes ist dieses Problem bis jetzt noch nicht gelöst. Diese Arbeit soll einen ersten Lösungsansatz hierfür entwickeln.

1.3 Gliederung der Arbeit

Das zweite Kapitel führt die notwendigen Grundlagen ein, die für diese Arbeit relevant sind. Hier werden grundlegende Begriffe definiert sowie die verwendeten Sprachen und Paradigmen erläutert. Auf diesen Grundlagen wird im dritten Kapitel der in dieser Arbeit entwickelte Entwurf der Systemarchitektur vorgestellt. Im vierten Kapitel wird das auf Grundlage dieses Entwurfs prototypisch implementierte System vorgestellt. Das fünfte Kapitel liefert eine Zusammenfassung und einen Ausblick auf Themen und Probleme, die in Zukunft genauer erforscht werden sollen.

Kapitel 2

Grundlagen

Für die Entwicklung des in 1.2 beschriebenen Systems sind verschiedene Voraussetzungen zu beachten und Zusammenhänge zu erläutern. In diesem Kapitel werden die grundlegenden Begriffe des Systems definiert und die zentrale Begriffe der Beschreibungslogik vorgestellt, die in dieser Arbeit genutzt werden.

2.1 Metadatenannotationen

Der Begriff „Metadatenannotation“ spielt im Kontext der Informatik und des semantischen Webs eine sehr große Rolle.

Unter Metadaten versteht man Daten, die andere Daten beschreiben. Ein einfaches Exempel hierfür ist ein Foto, das zum Beispiel im JPEG Format vorliegt. Dieses Format ermöglicht es Informationen über das Foto in derselben Datei zu speichern. Diese Informationen werden als Metadaten bezeichnet und könnten beispielsweise beinhalten, wann das Foto mit welchem Kamertyp und welcher Belichtungszeit und Blende aufgenommen wurde. Metadaten als solche sind für Maschinen nicht verwertbar, sondern im Allgemeinen nur für den Menschen interpretierbar [7].

Annotationen erfüllen in der Informatik und insbesondere im semantischen Web die Aufgabe, Metadaten maschinenlesbar zu machen [8, 9]. Zur Zeit stehen mehrere Alternativen für die Annotation zur Verfügung. Nennenswert ist in diesem Zusammenhang insbesondere das *Resource Description Framework* (RDF). Für Ontologien gibt es die *Web Ontology Language* (OWL), die auf Basis der RDF Syntax und unter Zuhilfenahme anderer Sprachkonstrukte eine sehr viel größere Mächtigkeit als RDF aufweist [10, 11]. Die in der prototypischen Architektur verwandte Reasoner Komponente kann auch mit dieser Art von Annotationsdaten umgehen.

Die Basis, auf die OWL als formale Sprache gegründet ist, sind die Beschreibungslogiken.

2.2 Beschreibungslogiken

Die Beschreibungslogiken (Description Logics (DL)) sind eine Familie der Wissensrepräsentationssprachen. Man kann sie benutzen, um das Wissen einer Anwendungsdomäne in klar verständlicher und strukturierter Weise formal darzustellen. Die Beschreibungslogiken ähneln der Prädikatenlogik, bieten allerdings die Möglichkeit, aus bereits vorhandenem explizitem Wissen auf neues implizites Wissen zu schließen. Dieser Vorgang wird als Reasoning bezeichnet [4, 12, 13]. Diese Eigenschaft ist im Kontext dieser Arbeit und dem BOEMIE-Projekt sehr wichtig.

Im folgenden werden die beiden für diese Arbeit benötigten Begriffe der Beschreibungslogiken definiert.

2.2.1 TBox

Der erste Schritt in der Modellierung einer Anwendungsdomäne ist die Definition der Konzepte, die in der Domäne vorhanden sind. Anschlie-

ßend kann man bestimmte Eigenschaften der Konzepte und Beziehungen zwischen den Konzepten festlegen. Es ist außerdem möglich, Bedingungen zu definieren, die Auswirkungen auf die Konzeptinstanzen haben. Nach Abschluss dieser Schritte hat man die so genannte *Ontologie* erstellt. Der Name TBox impliziert, dass in der TBox die Terminologie der Ontologie beschrieben wird.

2.2.2 ABox

Im zweiten Schritt der Wissensmodellierung werden so genannte ABoxen erstellt. Diese enthalten Behauptungen, oder im Englischen *Assertions*, über die Individuen, die beschrieben werden. Die Individuen werden als Instanzen eines in der TBox definierten Konzeptes bezeichnet. Instanzen können mit Hilfe von Rollen in der ABox in Beziehung gesetzt werden.

2.3 Informationskanal

Definition 2.1 *Informationskanäle sind alle Arten von Informationsquellen, aus denen neues Wissen bezogen werden kann.*

Man kann sich unter Informationskanälen die verschiedensten Quellen vorstellen. Beispielsweise können die heute weit verbreiteten RSS-Feeds als Informationskanäle angesehen werden. In ihrem Fall werden Daten in bestimmten Zeitintervallen an die Benutzer ausgeliefert [14, 15].

In dieser Arbeit werden allerdings andere Arten von Informationskanälen genutzt. Im Rahmen des BOEMIE-Projektes soll es ermöglicht werden, dass aus Quellen wie Zeitungsberichten, Internetseiten, Bildern, Audio- und auch Videodaten, automatisch Informationen extrahiert werden. Diese werden in den so genannten Metadatenannotationen abgespeichert und dem System zur Verfügung gestellt. Entsprechend den verschiedenen Medien gibt es verschiedene Informationskanäle, die das System mit besagten Metadatenannotationen beliefern.

2.4 Fusion

Der Begriff der Fusion ist für dieses Fachgebiet der Informatik noch nicht genau definiert. Das Wort hat seine Wurzeln im Lateinischen (*fusio*) und bedeutet das Gießen, Schmelzen. Im umgangssprachlichen oder auch wirtschaftlichen Sprachgebrauch bedeutet es *Verschmelzen* [16]. Diese Bedeutung soll es auch im Zusammenhang dieser Arbeit tragen. Das alleine kann natürlich nicht ausreichend beschreiben, was die Fusion im Detail ausmacht. Daher werden im Folgenden die beiden Schritte der Fusion vorgestellt.

2.4.1 Domänen ABox

Bevor die beiden Schritte für die Wissensfusion erläutert werden, wird eine weitere Neuerung vorgestellt. In der folgenden Beschreibung der Fusion wird das Wissen in einer so genannten Domänen ABox gehalten.

Sobald neues Wissen aus einzelnen Dokumenten zum bereits vorhandenen Wissen über die Domäne hinzugefügt werden soll, werden im Zuge der Fusion bestimmte Bedingungen geprüft. Nur wenn die Bedingungen erfüllt sind, wird das neue Wissen zu dem bereits vorhandenen Domänenwissen hinzugefügt.

Die ABox, die das gesamte geprüfte Wissen über die Domäne beinhaltet, wird ab jetzt Domänen ABox genannt. Sie wird unter anderem für die Beantwortung von Subskriptionen benutzt.

Dies hat den Vorteil, dass die Konsistenz der Wissensbasis durch einfache Abfragen geprüft werden kann.

1. Schritt

Definition 2.2 (Fusion I) *Dieser Schritt der Fusion beschreibt den Prozess, in dem neues Wissen aus verschiedenen Informationskanälen der Modalitäten eines Dokumentes in Zusammenhang gebracht wird, um dadurch einen Wissensgewinn zu erzielen.*

In diesem Schritt der Fusion kann neues Wissen erworben werden. Unter der Voraussetzung, dass das gesamte Wissen über ein Dokument mit mehreren Modalitäten zu einem Zeitpunkt zur Verfügung steht, kann das aus den Informationskanälen der verschiedenen Modalitäten eines Dokumentes zur Verfügung stehende Wissen gebündelt werden. Unter verschiedenen Modalitäten versteht man die verschiedenen Medien die ein Dokument enthält, also zum Beispiel Text, Audio, Bilder usw. .

Dies gelingt, da dieser Schritt Wissen aus multimodalen Inhalten verarbeiten kann. Es ist allerdings zu beachten, dass das gesamte Wissen eines Multimediadokuments zu selben Zeitpunkt für die Fusion zur Verfügung stehen muss. Stand bisher das Wissen eines jeden Teils eines Dokumentes für sich allein in einer so genannten Interpretation ABox, so kann jetzt das Wissen aus den verschiedenen Informationskanälen gebündelt werden. Das auf diese Weise fusionierte Wissen wird in einer so genannten Modalitäten ABox gehalten.

Ein schönes Beispiel für die Notwendigkeit dieses Schrittes ist in Abbildung 2.1 zu sehen. Es handelt sich hier um eine HTML Seite. Diese wurde von verschiedenen Extraktionsprogrammen nach Informationen durchsucht. Ein Extraktionsprogramm ist ein Programm, das aus einem Medium (Text, Bild etc.) Informationen über den Inhalt des Mediums



Abbildung 2.1: Beispiel für ein Dokument mit mehreren Modalitäten (HTML Seite)

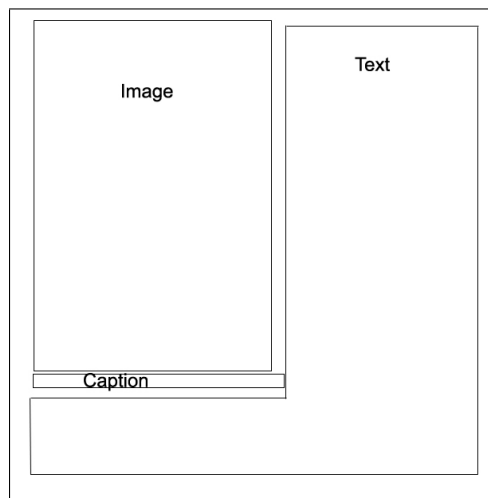


Abbildung 2.2: Beispiel für ein Dokument mit mehreren Modalitäten (Aufteilung)

extrahiert und als Metadatenannotation abgespeichert. Dies geschieht für jedes Medium bzw. jede Modalität einzeln.

Die Seite wird, wie in Abbildung 2.2 dargestellt, aufgeteilt. Interessant ist, dass mit Hilfe des OCR Verfahrens auch Informationen aus dem Text, der im Bild dargestellt ist, extrahiert werden können.

Im Zuge der Informationsextraktion sind drei verschiedene Dateien mit Metadatenannotationen entstanden. Für jede der Modalitäten (Text, Caption, Image) jeweils eine. Diese Metadatenannotationen enthalten unterschiedlich detaillierte Informationen über das auf der HTML Seite Beschriebene. Im Zuge der Fusion I werden diese Informationen in gewisser Weise zusammengetragen. Hieraus ergibt sich ein Wissensgewinn, der durch die alleinige Metadatenextraktion nicht möglich ist.

Anhand des Beispiels kann man sich dies auch klar machen. Die Bildinformationsextraktion liefert die Information, dass es sich bei dem Bild um ein Sportereignis handelt. Es kann allerdings leider nicht genau gesagt werden, ob es sich um Stabhochsprung oder Hochsprung handelt. Mittels Textextraktion kann aus der Bildunterschrift (Caption) der Name des Sportlers „Sergej Bubka“ ermittelt werden. Wiederum durch Textextraktion werden aus dem Bericht noch zahlreiche andere Informationen gewonnen. Unter anderem kann erkannt werden, dass der Sportler „Sergej Bubka“ beim Stabhochsprung die Goldmedaille gewonnen hat.

Durch die Fusion kann also nach der Wissensextraktion festgestellt werden, dass es sich bei dem Foto um ein Bild vom Stabhochsprung handelt.

Dieses Beispiel zeigt nur einen kleinen Teil dessen, was möglich ist.

Man kann sich auch Szenarien vorstellen, bei denen parallel zu dem Bericht auch noch ein Video zur Verfügung steht, aus dem der Ton mit Spracherkennung in Text umgewandelt wird und so weitere Informationen gebündelt und fusioniert werden können, beispielsweise die Sprunghöhe.

Dieser Wissensgewinn macht später Anfragen möglich, die etwa so aussehen könnten: Liefere mir alle Bilder, auf denen ein Stabhochsprung mit einer Mindesthöhe von 5,12 m zu sehen ist.

2. Schritt

Definition 2.3 (Fusion II) *Die Fusion II beschreibt den Prozess, in dem neues Wissen einer Wissensbasis, die so genannte Domänen ABox, hinzugefügt wird, wenn die Konsistenz der Wissensbasis sichergestellt werden kann.*

Dieser Schritt der Fusion ist gewissermaßen ein gesichertes Hinzufügen von neuen Informationen zu dem bisher vorhandenen Wissen in der Domänen ABox. Hierbei wird gesondert geprüft, ob Inkonsistenzen auftreten würden, wenn die neuen Daten zur Wissensbasis hinzugefügt werden sollen. Treten Inkonsistenzen auf, so werden die neuen Daten verworfen, anderenfalls werden die Daten im Reasoner publiziert.

In diesem Schritt der Fusion kann keinerlei neues Wissen gewonnen werden. In der prototypischen Implementierung, die in Kapitel 4 vorgestellt wird, wird nur dieser Schritt der Fusion durchgeführt.

Ein Beispiel für die Notwendigkeit dieser Fusion lässt sich ebenfalls leicht finden. Gegeben sei folgendes Szenario: Zwei Sportportale veröffentlichen jeweils einen Bericht über ein Sportereignis. In beiden Berichten wird erwähnt, dass ein Sportler A bei einem Wettkampf erfolgreich war. In einem Bericht wird festgestellt, dass Sportler A spanischer Nationalität ist. Wohingegen im anderen Bericht geschrieben wird, dass Sportler A mexikanischer Herkunft ist.

Nachdem die beiden Berichte die Textextraktion und den ersten Schritt der Fusion durchlaufen haben, wird nun zuerst der erste Text erfolgreich zur Domänen ABox hinzugefügt. Bei der Fusion II des zweiten Berichtes wird hingegen festgestellt, dass Sportler A zwei Nationalitäten haben soll, was laut der Definition der Ontologie, die in Anhang A.1 zu finden ist, nicht zulässig ist. Also wird der zweite Bericht nicht der Domänen ABox hinzugefügt, da diese sonst inkonsistent werden würde.

Die Identifizierung von identischen Instanzen in einer Domäne ist dem Domänen Experten überlassen. Dieser kann in der TBox Regeln für diese Identifizierung festlegen. In Anhang A.2 ist ein Beispiel für solch eine Regel zu finden.

2.5 Subskription

Die Subskription ist ein Teil des in der Informatik weit verbreiteten *Publish/Subscribe* Paradigmas. Es handelt sich hierbei um ein asynchrones Kommunikationsmodell. Benutzer können sich bei einem Anbieter für bestimmte Inhalte registrieren (subscribe) oder auch deregistrieren (unsubscribe). Macht der Anbieter Daten für die Benutzer öffentlich (publish), die zu der Subskription eines Benutzers passen, so erhält jeder Benutzer, der eine entsprechende Subskription getätigt hat, eine Nachricht mit den relevanten Daten [17]. Anstelle einer Subskription wird oft auch von einem Abonnement gesprochen. Diese Begriffe sind synonym zu verwenden.

In dieser Arbeit wird in Kapitel 4 eine prototypische Implementierung vorgestellt, die dieses Paradigma nutzt, um Benutzeranfragen zu beantworten. Hierbei werden vom Benutzer Themengebiete abonniert. Der Titel der Arbeit könnte vermuten lassen, dass der Benutzer direkt Informationskanäle abonniert, dies ist aber nicht der Fall. Die Verwaltung der Informationskanäle ist von Benutzereingaben entkoppelt.

Kapitel 3

Architektur

In diesem Kapitel wird die Architektur des Systems vorgestellt, das im Rahmen dieser Arbeit entworfen und prototypisch implementiert wurde.

Einen Überblick über die Architektur erhält man in Abbildung 3.1. Die Zusammenhänge zwischen den einzelnen Komponenten der Architektur sind in der der Abbildung dargestellt und erläutert.

In diesem Kapitel werden zuerst die zwei Use-Cases, die das System hat, vorgestellt. Danach werden alle Komponenten der Architektur und Ihre Aufgaben kurz erläutert.

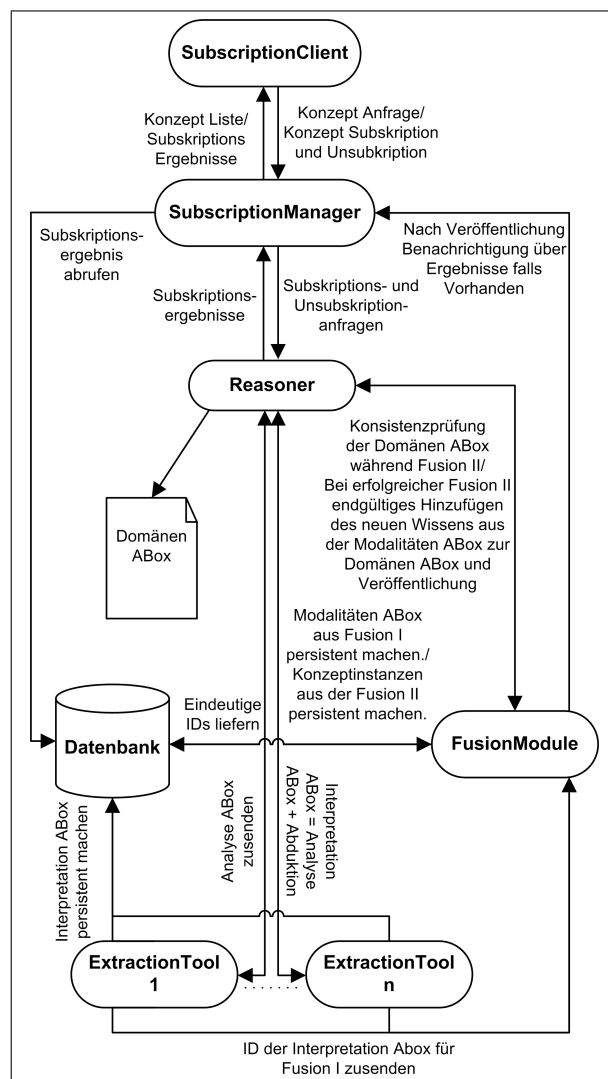


Abbildung 3.1: Die Systemarchitektur

3.1 Übersicht

Der in Abbildung 3.1 dargestellte Überblick über die Systemarchitektur ist während der ersten Phase der Entwicklung, der Analyse, entstanden. Es zeigt die Komponenten, aus denen sich das zu entwickelnde System zusammensetzt. Außerdem werden Abhängigkeiten der einzelnen Komponenten untereinander dargestellt und erläutert.

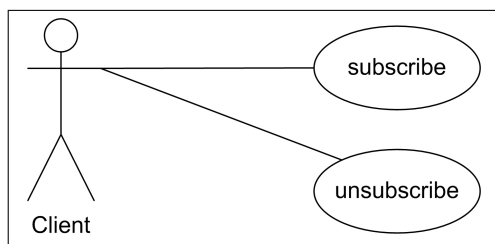


Abbildung 3.2: Use-Case: Klientenanwendung

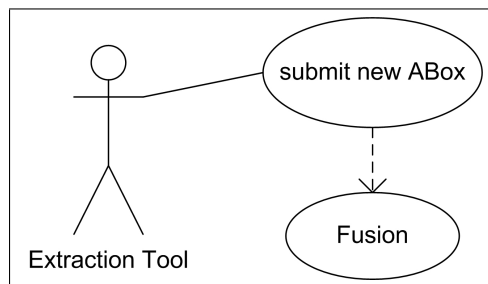


Abbildung 3.3: Use-Case: Extraktions Tools

3.2 Use-Cases

Das in dieser Arbeit entwickelte System soll die in den Abbildungen 3.2, 3.3 dargestellten Use-Cases unterstützen.

Der in Abbildung 3.2 dargestellte Use-Case stellt die typische Benutzerinteraktion mit dem System dar. Der Benutzer stellt mit einem SubscriptionClient die Verbindung zum System her und abonniert Daten (subscribe), die eine Instanz eines bestimmten Konzeptes enthalten. Er kann das Abonnement natürlich auch wieder beenden (unsubscribe).

Im zweiten Fall handelt es sich um das Hinzufügen von neuem Wissen zum System. Hier stellt kein Benutzer eine Anfrage an das System, sondern ein Extraktions Tool, das das System mit neuen Daten versorgt. Das Extraktions Tool teilt dem System eine ID einer neuen Interpretation ABox mit, unter der diese in der Datenbank abgerufen werden kann. Dieser Use-Case ist in Abbildung 3.3 dargestellt.

3.3 Komponenten

3.3.1 Datenbank

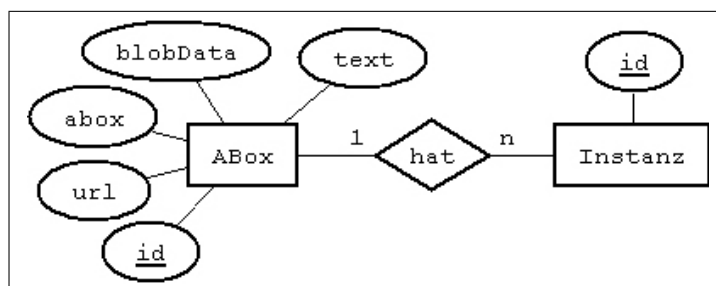


Abbildung 3.4: Entity Relationship Modell des Datenbank Schemas

Eines der zentralen Teile des Systems ist eine Datenbank. Sie spielt

in der Architektur die Rolle eines persistenten Speichers. In ihr werden insbesondere Daten gehalten, die für das Reasoning ohne Bedeutung sind.

Eine weitere wichtige Aufgabe der Datenbank ist die Vergabe von systemweit eindeutigen IDs für die verschiedenen Daten.

In Abbildung 3.4 ist das Datenbankschema abgebildet.

3.3.2 Reasoner

Der Reasoner ist ein zentrales Element der Architektur. Er hat im wesentlichen drei Aufgaben:

- Gewinnung von High-Level Konzepten aus Low-Level Konzepten und Beziehungen durch Abduktion
- Unterstützung der Publish/Subscribe Funktionalität
- Konsistenzprüfung der Domänen ABox

3.3.3 Extraktions Tools

Die Extraktions Tools liefern neues Wissen an das System, oder genauer gesagt an das `FusionModule`. Sie übernehmen in der Architektur die Rolle der Informationskanäle.

3.3.4 Subskriptions- und Fusionssystem

Die in Abbildung 3.1 abgebildete Architektur beinhaltet drei Komponenten, die im Rahmen dieser Arbeit entwickelt wurden. Dies sind das `FusionModule`, der `SubscriptionManager` und der `SubscriptionClient`. In den folgenden Abschnitten wird näher auf diese Komponenten eingegangen.

3.3.4.1 FusionModule

Das `FusionModule` hat in der Architektur folgende Aufgaben:

- Schnittstelle für die Benachrichtigung durch Extraktions Tools anbieten
- Durchführung der Fusion
- Verwaltung von Konzeptinstanzen in der Datenbank
- Aktualisierung der Instanz IDs in der Modalitäten ABox
- Publizierung der neuen Instanzen

3.3.4.2 SubscriptionManager

Der `SubscriptionManager` muss die folgenden Anforderungen erfüllen:

- Schnittstelle für Subskriptionen durch mehrere Klientanwendungen anbieten
- Verwaltung mehrerer Subskriptionen und Klientanwendungen (subscribe und unsubscribe)
- Auswertung von Antworten des Reasoners mit anschließendem Abrufen der entsprechenden Daten aus der Datenbank
- Benachrichtigung der Klientanwendungen beim Eintreffen neuer Ergebnisse

3.3.4.3 SubscriptionClient

An den `SubscriptionClient` sind folgenden Anforderungen zu stellen:

- Benutzeroberfläche für die Subskription und Unsubskription zur Verfügung stellen
- Schnittstelle für den Empfang der Ergebnisse der Subskriptionen anbieten
- Anzeige der Gesamtmenge der Ergebnisse als Liste mit der Möglichkeit, einzelne Elemente detailliert anzuzeigen

Kapitel 4

Implementierung

In diesem Kapitel wird die im Rahmen dieser Arbeit angefertigte prototypische Implementierung vorgestellt. Es werden die konkreten Komponenten der in Abbildung 3.1 dargestellten Architektur genannt und ihre Aufgaben und ihr Verhalten erläutert.

4.1 Datenbank

Die Datenbank spielt in der Architektur eine zentrale Rolle. Sie hält sämtliche Daten vor, die dem System bekannt sind und verwaltet zusätzlich konsistent die Identifikationsnummern (IDs) der ABoxen, Instanzen und Konzepte.

Mehrere Komponenten der Architektur greifen auf die Datenbank zu:

- Die Extraktions Tools speichern in ihr die Interpretation ABoxen mit neuen Assertions ab und schicken die IDs, die die Datenbank vergeben hat, an das **FusionModule**
- Das **FusionModule** ruft die neue Interpretation ABox aus der Datenbank ab und extrahiert alle Instanzen, die diese enthält. Danach fügt es alle Instanzen der Datenbank hinzu und aktualisiert die IDs der Instanzen in der Interpretation ABox. Dies ist nötig, um später von einer Instanz auf eine Interpretation ABox und somit auf die dazu gehörenden Daten schließen zu können.
- Der **SubscriptionManager** greift beim Eintreffen eines neuen Ergebnisses auf die Datenbank zu, um die benötigten Daten aus der Datenbank abzurufen und dann an den **SubscriptionClient** zu senden.

Bei der Wahl der Datenbank war in diesem Fall vor allem wichtig, dass eine JDBC Anbindung möglich ist. Fast jeder Datenbank-Hersteller bietet eine solche Anbindung. Da die Datenbank allerdings nur sehr einfache Anfragen bearbeiten muss und keine komplexen Strukturen abbilden soll (siehe Abbildung 3.4), reicht auch ein sehr schmales Produkt aus, um diese Anforderungen zu erfüllen. Aufgrund ihrer freien Verfügbarkeit und der einfachen und komfortablen Bedienung fiel die Wahl daher auf die frei verfügbare *MySQL* Datenbank. Es besteht natürlich die Möglichkeit, mit kleinen Anpassungen jede beliebige Datenbank zu verwenden.

4.2 Reasoner

Der Reasoner ist ein zentrales Element der Architektur. Er wird von zwei Komponenten kontaktiert. Er verwaltet die Domänen ABox, die das gesamte vorhandene Wissen der Domäne enthält.

Das **FusionModule** kopiert zuerst die Domänen ABox und fügt dieser Kopie dann die neuen Assertions zu, um danach zu prüfen, ob die Konsistenz der Domänen ABox noch gegeben ist. Sollte dies der Fall sein, werden die neuen Assertions der originalen Domänen ABox hinzugefügt und veröffentlicht (publish).

Der `SubscriptionManager` sendet bei jeder neuen Subskription oder Unsubskription einer Klientanwendung eine entsprechende Nachricht an den Reasoner.

Aufgrund der Tatsache, dass im BOEMIE-Projekt der RacerPro *RacerPro 1.9 (Renamed Abox and Concept Expression Reasoner)* der Racer System GmbH & Co. KG als Reasoner zum Einsatz kommt, wird er auch in dieser Arbeit verwendet. Die Racer System GmbH & Co. KG stellt außerdem für Forschungszwecke eine uneingeschränkte, wenn auch zeitlich begrenzte Lizenz zur Verfügung.

Leider ist die Publish/Subscribe Funktionalität des RacerPro nicht sehr ausgereift und weist einen entscheidenden Nachteil auf: Sobald neue Instanzen publiziert werden, wird nicht der Subskribierende benachrichtigt, sondern nur der Publizierende [18]. Dieses Problem wurde dadurch behoben, dass das `FusionModule` beim Publizieren neuer Instanzen den `SubscriptionManager` benachrichtigt.

4.2.1 Racer Schnittstelle

Um auf den RacerPro zugreifen zu können, benötigt man eine Schnittstelle. Der RacerPro bietet hierfür drei Alternativen an:

- DIG Interface
- Lisp API
- Java API

Das DIG Interface basiert auf einer standardisierten XML Schnittstelle, die durch die Description Logic Implementation Group (DIG) entwickelt wurde. Der Zugriff erfolgt hierbei über das **H**ypertext **T**ransfer **P**rotocol (HTTP). Die anderen beiden Schnittstellen basieren auf einfachen TCP Verbindungen.

In der Implementierung wird die TCP Schnittstelle genutzt.

Eine Wahl zwischen den beiden anderen Schnittstellen ist abhängig von der gewählten Implementationsprogrammiersprache. In diesem Fall wird Java genutzt und daher auch die Java API namens `JRacer`. Beide Schnittstellen basieren auf TCP Sockets [18].

4.3 Extraktions Tools

Im Rahmen des BOEMIE-Projektes werden Extraktionstools entwickelt, die automatisch den Inhalt aus Texten, Bildern und sogar Videos extrahieren können. Mit Hilfe anderer Techniken wird es außerdem möglich sein, den Inhalt von Audiodaten zu ermitteln. Hierfür wird im ersten

Schritt eine Spracherkennung durchgeführt und daran anschließend im zweiten Schritt eine normale Textextraktion vollzogen.

Eine ähnliche Vorgehensweise wird bei Text, der in Bilddaten auftaucht, benutzt. Hier wird im ersten Schritt allerdings das *OCR* Verfahren zur Texterkennung genutzt. OCR steht für **O**ptical **C**haracter **R**ecognition, was soviel bedeutet wie: Optische Buchstaben Erkennung. Darauf folgend wird wieder, wenn möglich, über die Textextraktion Wissen gewonnen.

Da diese Tools zur Zeit der Entwicklung des Systems im Rahmen dieser Arbeit noch nicht zur Verfügung standen, weil sie selber noch in der Entwicklung steckten, musste eine Alternative gefunden werden.

Der Einfachheit halber wurde ein Tool mit dem Namen **ABoxSubmitter** entworfen, das es ermöglicht, neue Interpretation ABoxen in das System einzupflegen.

ABoxSubmitter

Aufgrund der fehlenden Extraktions Tools wurde der einfache **ABoxSubmitter** als Tool entwickelt (siehe Abbildung 4.1), das die Extraktions Tools und ihren Einfluss auf das System simuliert.

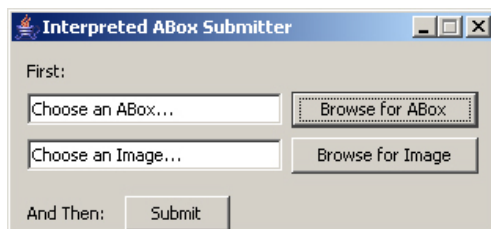


Abbildung 4.1: ABoxSubmitter der die Extraktions Tools simuliert.

Zur Zeit ist es nur möglich, Interpretation ABoxen, die Bilder beschreiben, einzupflegen. Die Funktionalität auf Text und Video zu erweitern, ist ohne viel Aufwand möglich, wird aber für diese Arbeit nicht weiterverfolgt.

Der **ABoxSubmitter** speichert die ABox mit dem Bild in der Datenbank ab und benachrichtigt anschließend das **FusionModule**. Dem Benutzer wird außerdem mitgeteilt, ob der Vorgang erfolgreich war oder nicht.

Das Tool wurde mit Hilfe des Java Swing Frameworks [19] entwickelt und in der Netbeans Entwicklungsumgebung entworfen.

4.4 Subskriptions- und Fusionssystem

Wie bereits in Abschnitt 3.3.4 erwähnt, enthält die Architektur drei Komponenten (**FusionModule**, **SubscriptionManager** und **SubscriptionClient**), die im Rahmen dieser Arbeit entwickelt und prototypisch implementiert wurden. In den folgenden Abschnitten wird näher auf die Funktionsweise dieser Komponenten eingegangen.

4.4.1 FusionModule

Das **FusionModule** hat in der entworfenen Architektur die Aufgabe, die Fusion durchzuführen. In der prototypischen Implementierung wurde aus Zeitgründen nur Fusion II umgesetzt. Deshalb verarbeitet Fusion II in der Implementierung keine Modalitäten ABoxen sondern Interpretation ABoxen.

Das ebenfalls im Rahmen dieser Arbeit entwickelte **ABoxSubmitter** Tool fügt neue Interpretation ABoxen in Textform der zentralen Datenbank hinzu. Danach benachrichtigt es das **FusionModule** über das Eintreffen einer neuen Interpretation ABox und teilt dem **FusionModule** dessen ID mit. Daraufhin ruft das **FusionModule** den neuen Interpretation ABox String aus der Datenbank ab. Dieser wird dann auf Instanzen von Konzepten durchsucht, die in der Taxonomie vorhanden sind. Die Konzepte der Taxonomie ruft das **FusionModule** bei der Initialisierung aus der Datenbank ab. Anschließend wird jede Instanz eines Konzeptes in der zentralen Datenbank abgespeichert, um eine eindeutige Instanz ID zu erhalten. Diese ID wird in der Interpretation ABox aktualisiert. Außerdem wird die Relation Instanz $\rightarrow$ ABox in der Datenbank abgespeichert. Als nächstes wird eine Kopie der Domänen ABox erstellt, in die daraufhin die neue ABox eingefügt wird. Anschließend können Regeln ausgeführt werden, wenn diese zuvor in einer Konfigurationsdatei festgelegt wurden (siehe Anhang A.2).

Das Erstellen einer Kopie der Domänen ABox ist nötig, da die Konsistenz der Domänen ABox durch das Hinzufügen neuer Assertions verloren gehen kann. Ein Beispiel für solch einen Fall ist in Abschnitt 2.4.1 zu finden.

Sollte eine Inkonsistenz auftreten, werden die neuen Daten verworfen. In jedem Fall wird die Kopie der Domänen ABox gelöscht. Erst nachdem sichergestellt ist, dass die Konsistenz beibehalten werden kann, wird die neue Interpretation ABox zu der originalen Domänen ABox hinzugefügt. Abschließend werden alle neuen Instanzen veröffentlicht (publish). Da der Racer nur dem Publizierenden über neue Subskriptionsergebnisse benachrichtigt, schickt das **FusionModule** die Antwort des Racers nach der Publikation einer Instanz an den **SubscriptionManager**.

Damit sind alle Anforderungen an das FusionModule erfüllt.

In Abbildung 4.2 ist das Konsolenfenster des FusionModule für einen erfolgreichen Fusionsvorgang für die erste Interpretation ABox aus Anhang A.3 abgebildet.

```

C:\WINDOWS\system32\cmd.exe
This is the FusionModule.
Connection to the database and Racer established.
Fetching available concepts from database.
35 concepts available.
Initializing the domainabox...
<PREPARE-ABOX-RULE (AND (?X <AN NAME>) (?Y <AN NAME>) (?X ?Y <CONSTRAINT <NAME>
<NAME> STRING=>)) <<SAME-AS ?X ?Y>>) --> <:RULE-1 :READY-TO-RUN>

Waiting for connection from an extraction tool.
Connection received from: localhost
Received ABox id from extraction tool: 1
Closed connection to the extraction tool.
Now processing the new ABox
Fetched ABox from database.
Parsing the ABox for concept instances.
Found instance of concept: MALEATHLETE
Storing instance and its relation(s) in database.
Updating old instance name <MALEAT1> with explicit one from database: I1
Found instance of concept: GERMAN
Storing instance and its relation(s) in database.
Updating old instance name <NAT1> with explicit one from database: I2
Found instance of concept: POLEVAULT
Storing instance and its relation(s) in database.
Updating old instance name <SPORT1> with explicit one from database: I3
Found instance of concept: POLEVAULTCOMPETITION
Storing instance and its relation(s) in database.
Updating old instance name <SPORTCOMPETITION1> with explicit one from database:
I4
Created ABox clone: DOMAINABOX0
Adding assertions to domainabox-clone.
<PREPARE-ABOX-RULE (AND (?X <AN NAME>) (?Y <AN NAME>) (?X ?Y <CONSTRAINT <NAME>
<NAME> STRING=>)) <<SAME-AS ?X ?Y>>) --> <:RULE-2 :READY-TO-RUN>

Executed rules: <NIL NIL>
Deleted the last rule.
Checked domainabox-clone for consistency: T
Adding assertions to domainabox.
Executed rules: <NIL>
Publishing new instances:
Publish resulted in: NIL
Instance published: I1
Publish resulted in: NIL
Instance published: I2
Publish resulted in: NIL
Instance published: I3
Publish resulted in: NIL
Instance published: I4
Deleted domainabox-clone: DOMAINABOX0
Waiting for connection from an extraction tool.

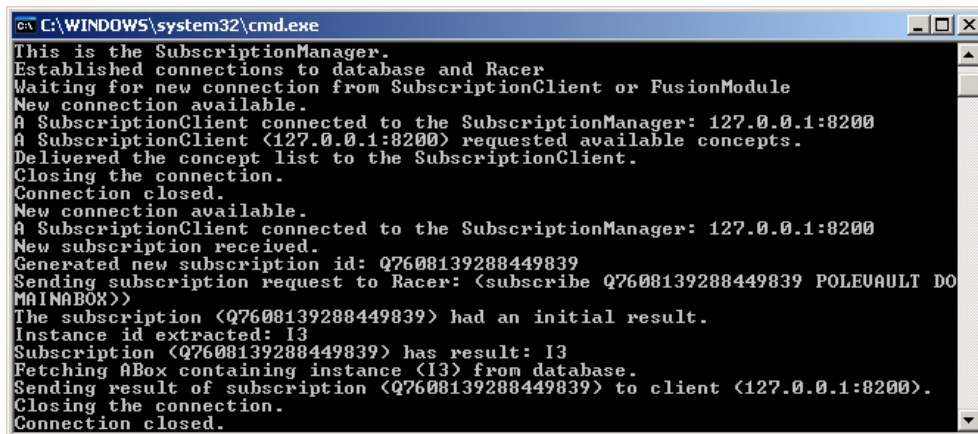
```

Abbildung 4.2: FusionModule nach dem Hinzufügen einer Interpretation ABox.

4.4.2 SubscriptionManager

Der SubscriptionManager übernimmt die Verwaltung der Subskriptionen, den Abruf von Ergebnisdaten aus der Datenbank und die Auslieferung der Ergebnisdaten an den SubscriptionClient. Er ist genau wie das FusionModule als einfache Konsolenanwendung implementiert (siehe Abbildung 4.3).

Sobald der SubscriptionManager gestartet wird, wartet er auf den Eingang einer neuen Verbindung. Eingehende Verbindungen können entweder von dem SubscriptionClient oder vom FusionModule kommen.



```

C:\WINDOWS\system32\cmd.exe
This is the SubscriptionManager.
Established connections to database and Racer
Waiting for new connection from SubscriptionClient or FusionModule
New connection available.
A SubscriptionClient connected to the SubscriptionManager: 127.0.0.1:8200
A SubscriptionClient <127.0.0.1:8200> requested available concepts.
Delivered the concept list to the SubscriptionClient.
Closing the connection.
Connection closed.
New connection available.
A SubscriptionClient connected to the SubscriptionManager: 127.0.0.1:8200
New subscription received.
Generated new subscription id: Q7608139288449839
Sending subscription request to Racer: <subscribe Q7608139288449839 POLEVAULT DO
MAINABOX>
The subscription <Q7608139288449839> had an initial result.
Instance id extracted: I3
Subscription <Q7608139288449839> has result: I3
Fetching ABox containing instance <I3> from database.
Sending result of subscription <Q7608139288449839> to client <127.0.0.1:8200>.
Closing the connection.
Connection closed.

```

Abbildung 4.3: SubscriptionManager nach der erfolgreichen Subskription eines Konzeptes

In jedem Fall wird für eine neue Verbindung ein neuer Thread [20] erzeugt, der diese Verbindung bearbeitet.

Verbindung vom SubscriptionClient

Kommt die Verbindung vom SubscriptionClient, so muss entschieden werden, was dieser vom SubscriptionManager möchte. Hierfür gibt es drei Möglichkeiten:

1. Eine Liste der Konzepte, die der Benutzer abonnieren kann
2. Ein Konzept, das abonniert werden soll (subscribe)
3. Die vorher getätigte Subskription kündigen (unsubscribe)

Im ersten Fall ruft der SubscriptionManager alle Konzepte der Ontologie aus der Datenbank ab und sendet sie in geeignetem Format an den SubscriptionClient.

Im zweiten Fall wird eine neue zufällige Subskriptions ID generiert, mit der dann eine neue Subskription an den Racer geschickt wird. Stehen bereits Ergebnisse zur Verfügung, wird vom RacerPro eine Liste mit Ergebnissen zurück gegeben. Ist dies der Fall, so werden die entsprechenden Daten aus der Datenbank abgerufen, in einem SubscriptionResult Objekt abgelegt, das an den SubscriptionClient weitergegeben wird. Gibt es noch keine Daten, so wird auch ein SubscriptionResult Objekt erzeugt, das allerdings ein Flag besitzt, das anzeigt, dass es leer ist. In jedem Fall wird die Subskriptions ID zusammen mit den Verbindungsdaten des SubscriptionClient in einem Verbindungs Pool gespeichert. Aus diesem können bei Bedarf die Verbindungsdaten abgerufen werden.

Im dritten Fall wird die Unsubskriptionsnachricht an den Racer weitergeleitet. Außerdem wird das (Verbindung, Subskriptions ID) Tupeln aus dem Verbindungs Pool gelöscht.

Verbindung vom FusionModule

Kommt die Verbindung vom **FusionModule**, empfängt der **SubscriptionManager** eine Liste mit neuen (Subskriptions ID, Instanz ID) Tupeln. Für jedes Element dieser Liste werden die Daten der Instanz aus der Datenbank abgerufen. Dann wird ein **SubscriptionResult** Objekt erstellt, in dem die Daten abgelegt werden, die zu der ABox der jeweiligen Instanz gehören. Dieses Objekt wird dann an den **SubscriptionClient** gesendet, damit dieser dem Benutzer das Ergebnis präsentieren kann.

Da das Unsubskriptions Kommando im Racer keine Wirkung hat, werden im **SubscriptionManager** ihm unbekannte Subskriptions IDs heraus gefiltert.

4.4.3 SubscriptionClient

Der **SubscriptionClient** implementiert eine graphische Oberfläche, die es ermöglicht, dass der Benutzer Daten abonnieren kann, die ein von ihm gewähltes Konzept als Instanz beinhalten.

Nachdem der **SubscriptionClient** gestartet wurde, kann der Benutzer ein Konzept auswählen, das er subscribieren möchte. Sobald er auf den Subscribe Button klickt, wird ein **SubscriptionResponseListenerThread** gestartet, der fortan mit einem TCP/IP **ServerSocket** auf neue Ergebnisse wartet.

Trifft ein neues Ergebnis ein, so wird es, wie in Abbildung 4.4 zu sehen, in einer Ergebnisliste angezeigt. Selektiert der Benutzer ein Ergebnis aus dieser Liste, so wird es auf der Ergebnisanzeigefläche in der Mitte des Fensters angezeigt. Zur Zeit werden nur Bilddaten als Ergebnisse angezeigt, da sich auch nur solche im System befinden.

Es ist möglich, bestimmte Einstellungen zu ändern. Das betreffende Menü ist unter der Lasche „Settings“ zu finden. Hier kann der Hostname und Port für den **SubscriptionManager** eingestellt werden sowie der Port, auf dem der **SubscriptionClient** auf neue Ergebnisse warten soll. Diese Einstellungen ermöglichen eine verteilte Nutzung des Systems.

4.4.4 SFSSetupModule

Damit die Voreinstellungen für die prototypische Implementierung einfach durchgeführt werden können, wurde das so genannte „**Subscription and Fusion-System**“-SetupModule entwickelt.

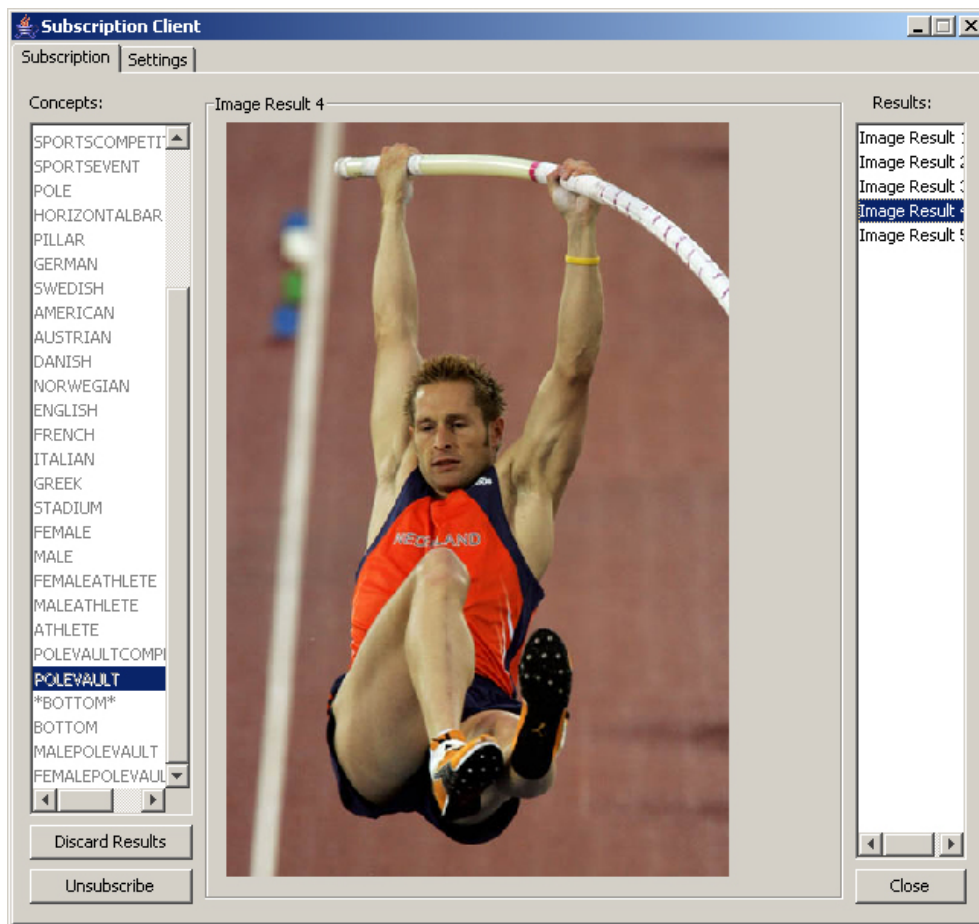


Abbildung 4.4: SubscriptionClient nach dem Empfang mehrerer Ergebnisse einer Subskription.

Dieses Modul initialisiert den Racer und die Datenbank. Es kann wie alle Komponenten über eine Konfigurationsdatei konfiguriert werden.

Kapitel 5

Zusammenfassung und Ausblick

5.1 Zusammenfassung

In dieser Arbeit wurden zwei verschiedene Probleme behandelt. Zum einen wurde die Fusion im Kontext der Wissensextraktion mit Beschreibungslogiken im Rahmen des BOEMIE-Projektes beleuchtet. Zum anderen wurde die bereits vorhandene Publish/Subscribe Funktionalität des RacerPro um das Abrufen von Ergebnisdaten der Subskriptionen aus einer Datenbank erweitert. Auch dies geschah im Hinblick auf das BOEMIE-Projekt, für das eine solche Funktionalität von Nutzen sein könnte.

Im Bereich der Wissensfusion mit Hilfe von Beschreibungslogiken wurde ein Ansatz entwickelt. Dieser Ansatz beinhaltet zwei voneinander unabhängige Schritte: Fusion I und Fusion II.

Die Fusion II wurde innerhalb dieser Arbeit in einer Komponente, dem `FusionModule`, der entwickelten Architektur prototypisch implementiert und hat sich in diesem Zusammenhang als durchaus nützlich erwiesen. Der andere Schritt (Fusion I) wurde in dieser Arbeit nur theoretisch erläutert und kann im Zuge der zukünftigen Forschung weiterverfolgt und implementiert werden.

Weiterhin wurde ein prototypisches System entwickelt, das mit Hilfe des Publish/Subscribe Paradigmas Benutzeranfragen bedarfsgerecht beantwortet. Außerdem wurde im Rahmen dieses Systems eine Komponente entwickelt, der `SubscriptionManager`, der die bisher fehlende Verbindung zwischen Reasoner und einer Datenbank herstellt. So können jetzt beliebige Klientanwendungen entwickelt werden, die Daten abonnieren und beliebig aufwändige Arten der Ergebnisanzeige verwirklichen können. In dieser Arbeit wurde eine einfache Klientanwendung entworfen,

der `SubscriptionClient`, die eine Möglichkeit für eine Klientenanwendung vorstellt.

Zusammenfassend kann man sagen, dass sich in dieser Arbeit einige interessante Ansätze ergeben haben, die es in Zukunft genauer zu erforschen gilt.

5.2 Ausblick

Die prototypische Implementierung, die im Rahmen dieser Arbeit entwickelt wurde, ist als Beweis für die Notwendigkeit der Fusion im Kontext der bedarfsgerechten Beantwortung von Subskriptionen zu verstehen. Aspekten wie Performanz, Sicherheit, Fehlerbehandlung und Skalierbarkeit des Systems wurde während der Implementierung keine ausgeprägte Aufmerksamkeit beigemessen. Probleme, die diese Aspekte betreffen, müssen in einer tatsächlichen Implementierung gelöst werden.

Der bisher nur theoretisch behandelte Schritt der Fusion (Fusion I) wird in Zukunft im Rahmen des BOEMIE-Projektes genauer untersucht, gegebenenfalls weiterentwickelt und implementiert werden. Hier wird es zukünftig sicherlich noch interessante Entwicklungen geben, da die Forschung auf diesem Gebiet noch am Anfang steht. Im Rahmen dieser zeitlich begrenzten Bachelorarbeit war eine weiterführende Evaluation und Implementierung nicht möglich. Die Implementierung der Fusion I sollte nach Möglichkeit so durchgeführt werden, dass der Domänen Experte Änderungen für die Bedingungen für Fusion I leicht durchführen kann. Im optimalen Fall kann der Domänen Experte diese Bedingungen festlegen, woraufhin sich das System dynamisch an die neuen Bedingungen anpasst.

Anhang A

Anhang

A.1 Beispiel: TBox

```
1
2 (full-reset)
3
4 (in-knowledge-base DOMAINTBOX)
5
6 ; This one enables the domain expert to identify instances
7 ; by their given concrete domain attribute, called name.
8
9 (define-concrete-domain-attribute name :type string)
10
11
12 (implies german Nationality)
13 (implies swedish Nationality)
14 (implies american Nationality)
15 (implies austrian Nationality)
16 (implies danish Nationality)
17 (implies norwegian Nationality)
18 (implies english Nationality)
19 (implies french Nationality)
20 (implies italian Nationality)
21 (implies greek Nationality)
22
23 (disjoint german swedish american austrian
24           danish norwegian english french italian greek)
25
26 (implies Stadium SportsPOI)
27
28 (implies Person (and
29                 (at-most 1 has-nationality Nationality)
30                 (some age *top*)
31                 (at-most 1 has-gender Gender)))
32
33 (implies Athlete (and
34                 Person
35                 (some has-part AthleteFace)
36                 (some has-part AthleteBody)))
37
38 (implies FemaleAthlete (and Athlete (some has-gender Female)))
39
40 (implies MaleAthlete (and Athlete (some has-gender Male)))
41
42 (implies Female Gender)
```

```

43 |
44 | (implies Male Gender)
45 |
46 | (disjoint Male Female)
47 |
48 | (implies SportsEvent (and
49 |     (some hast-part SportsCompetition)
50 |     (some has-event-name *top*)
51 |     (at-most 1 has-startdate)
52 |     (at-most 1 has-enddate)
53 |     (some has-city *top*)
54 |     (at-most 1 has-country *top*)))
55 |
56 | (implies SportsCompetition (and
57 |     (at-least 1 has-part SportsRound)
58 |     (some has-sport-name *top*)
59 |     (at-most 1 has-city *top*)
60 |     (some takes-places SportsPOI)))
61 |
62 | (implies SportsRound (and
63 |     (at-least 1 has-part SportsTrial)
64 |     (at-most 1 has-round-name *top*)
65 |     (at-most 1 has-startdate *top*)
66 |     (at-most 1 has-performance *top*)))
67 |
68 | (implies SportsTrial (and
69 |     (at-most 1 has-part Athlete)
70 |     (at-most 1 has-performance *top*)
71 |     (exactly 1 has-ranking *top*)))
72 |
73 | (implies PoleVaultCompetition (and
74 |     SportsCompetition
75 |     (at-least 1 has-part PoleVault)))
76 |
77 | (implies PoleVault (and
78 |     SportsTrial
79 |     (max has-upper-limit 530)
80 |     (some has-part Pole)
81 |     (some has-part HorizontalBar)
82 |     (some has-part Pillar)))
83 |
84 | (implies FemalePoleVault (and
85 |     PoleVault
86 |     (all has-part FemaleAthlete)
87 |     (max has-upper-limit 510)))
88 |
89 | (implies MalePoleVault (and
90 |     PoleVault
91 |     (all has-part MaleAthlete)))

```

A.2 Beispiel: Regel

```

1 | ; This rule is identifies instances that have the same name
2 | ; and states that they are the same with the same-as operation
3 |
4 | (prepare-abox-rule (and (?x (an name)) (?y (an name))
5 |     (?x ?y (constraint (name) (name) string=)))
6 |     ((same-as ?x ?y)))

```

A.3 Beispiel: Zwei Interpretation ABoxen die eine Inkonsistenz hervorrufen

```
1 (INSTANCE maleAt1 MaleAthlete)
2 (INSTANCE maleAt1 (STRING= name "John.Doe" ))
3 (INSTANCE nat1 german)
4 (RELATED maleAt1 nat1 has-nationality)
5 (INSTANCE sport1 PoleVault)
6 (INSTANCE sportCompetition1 PoleVaultCompetition)
```

```
1 (INSTANCE maleAt2 MaleAthlete)
2 (INSTANCE maleAt2 (STRING= name "John.Doe" ))
3 (INSTANCE nat2 swedish)
4 (RELATED maleAt2 nat2 has-nationality)
5 (INSTANCE sport2 PoleVault)
6 (INSTANCE sportCompetition2 PoleVaultCompetition)
```

A.4 CD

Der Arbeit ist eine CD mit der im Rahmen dieser Arbeit entstandenen Implementierung beigelegt. Im Ordner *SubscriptionAndFusionSystem* sind vier Unterordner zu finden:

- bin: Enthält das ausführbare Kompilat der im Ordner src vorliegenden Klassen. Ebenso sind Startskripte für die Windows Plattform vorhanden.
- doc: Enthält die Java Dokumentation der Klassen.
- lib: Beinhaltet die Klassenbibliotheken, die in der Implementierung verwendet wurden.
- src: Enthält die Klassendefinitionen der in dieser Arbeit entwickelten Implementierung

Für die Ausführung sind zwei weitere Komponenten nötig. Zum einen die MySQL Datenbank und zum anderen der RacerPro.

In dem Ordner Bachelorarbeit ist eine PDF Version dieser Arbeit enthalten.

Literaturverzeichnis

- [1] BOLSHAKOV, Igor A. ; GELBUKH, Alexander: *Computational Linguistics - Models, Resources, Applications*. Ciencia da la computación, 2004
- [2] BOEMIE.ORG: *Introduction — BOEMIE*. Version: 2007. <http://www.boemie.org>, Abruf: 28.1.2007
- [3] DAVIES, John ; STUDER, Rudi ; WARREN, Paul: *Semantic Web Technologies - Trends and research in ontology-based systems*. Wiley, 2006
- [4] BAADER, Franz: *The description logic handbook : theory, implementation and applications*. Cambridge, UK ; New York : Cambridge University Press, 2003
- [5] WIKIPEDIA: *Ontologie (Informatik) — Wikipedia, Die freie Enzyklopädie*. Version: 2007. [http://de.wikipedia.org/wiki/Ontologie_\(Informatik\)](http://de.wikipedia.org/wiki/Ontologie_(Informatik)), Abruf: 28.1.2007
- [6] RUH, William A. ; MAGINNIS, Francis X. ; ; BROWN, William J.: *Enterprise Application Integration*. John Wiley & Sons, INC., 2001
- [7] STUCKENSCHMIDT, Heiner ; HARMELEN, Frank van: *Information sharing on the semantic Web*. Springer, 2005
- [8] HANDSCHUH, Siegfried ; STAAB, Steffen: *Annotation for the Semantic Web*. IOS Press, 2003
- [9] WIKIPEDIA: *Annotation — Wikipedia, Die freie Enzyklopädie*. Version: 2006. <http://de.wikipedia.org/wiki/Annotation>, Abruf: 12.2.2007
- [10] W3.ORG: *Web Ontology Language OWL / W3C Semantic Web Activity*. Version: 2007. <http://www.w3.org/2004/OWL/>, Abruf: 20.2.2007

- [11] WIKIPEDIA: *Web Ontology Language* — *Wikipedia, Die freie Enzyklopädie*. Version: 2007. http://de.wikipedia.org/wiki/Web_Ontology_Language, Abruf: 12.2.2007
- [12] SUTANO, Yonny: *Practical Evaluation of Contraction and Abduction Algorithm in Description Logic*, TUHH, Master Thesis, 2006
- [13] WIKIPEDIA: *Beschreibungslogik* — *Wikipedia, Die freie Enzyklopädie*. Version: 2006. <http://de.wikipedia.org/wiki/Beschreibungslogik>, Abruf: 12.2.2007
- [14] JOHNSEN, Dave: *RSS and Atom in action : Web 2.0 building blocks*. Manning, 2006
- [15] WIKIPEDIA: *RSS* — *Wikipedia, Die freie Enzyklopädie*. Version: 2007. <http://de.wikipedia.org/wiki/Rss>, Abruf: 12.2.2007
- [16] LEXIKONREDAKTION, Meyers: Meyers großes Taschenlexikon. In: *Farn-Gap*. Institut & F.A. Brockhaus AG, 1995, S. 286
- [17] MELZER, Sylvia: *Ein inhaltsbasiertes Publish/Subscribe Architekturkonzept zur individuellen Sensordatenversorgung*, TUHH, Diplomarbeit, 2006
- [18] KG, Racer Systems GmbH & C.: *RacerPro User's Guide - Version 1.9*. 2005
- [19] FISCHER, Paul: *An Introduction to Graphical User Interfaces with Java Swing*. Addison Wesley, 2005
- [20] JECKLE, Mario: *Scriptum zur Vorlesung: Java Threads*. <http://www.jeckle.de/files/jtDeep.pdf>. Version: 2004, Abruf: 14.2.2007