UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

ArTS-F - Access restricted Triple Store Framework

Wintersemester 2010/11

Studienarbeit

vorgelegt von

Andreas Stahlhofen

Betreuer: Prof. Dr. Steffen Staab
Institute for Web Science and Technologies (WeST)
Felix Schwagereit
Institute for Web Science and Technologies (WeST)

Koblenz, im Januar 2011

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Die nächste Generation des World Wide Web, das Semantic Web, erlaubt Benutzern Unmengen an Informationen über die Grenzen von Webseiten und Anwendungen hinaus zu veröffentlichen und auszutauschen. Die Prinzipien von Linked Data beschreiben Konventionen, um diese Informationen maschinenlesbar zu veröffentlichen [BL09]. Obwohl es sich aktuell meist um Linked Open Data handelt, deren Verbreitung nicht beschränkt, sondern explizit erwünscht ist, existieren viele Anwendungsfälle, in denen der Zugriff auf Linked Data in Resource Description Framework (RDF) Repositories regelbar sein soll. Bisher existieren lediglich Ansätze für die Lösung dieser Problemstellung, weshalb die Veröffentlichung von vertraulichen Inhalten mittels Linked Data bisher nicht möglich war.
Aktuell können schützenswerte Informationen nur mit Hilfe eines externen Betreibers kontrolliert veröffentlicht werden. Dabei werden alle Daten auf dessen System abgelegt und verwaltet. Für einen wirksamen Schutz sind weitere Zugriffsrichtlinien, Authentifizierung von Nutzern sowie eine sichere Datenablage notwendig. Beispiele für ein solches Szenario finden sich bei den sozialen Netzwerken, wie Facebook oder StudiVZ. Die Authentifizierung aller Nutzer findet über eine zentrale Webseite statt. Anschließend kann beispielsweise über eine Administrationsseite der Zugriff auf Informationen für bestimmte Nutzergruppen definiert werden. Trotz der aufgezeigten Schutzmechanismen hat der Betreiber selbst immer Zugriff auf die Daten und Inhalte aller Nutzer. Dieser Zustand ist nicht zufriedenstellend.
Die Idee des Semantic Webs stellt einen alternativen Ansatz zur Verfügung. Der Nutzer legt seine Daten an einer von ihm kontrollierten Stelle ab, beispielsweise auf seinem privaten Server. Im Gegensatz zum zuvor vorgestellten Szenario ist somit jeder Nutzer selbst für Kontrollmechanismen wie Authentifizierung und Zugriffsrichtlinien verantwortlich.
Innerhalb der vorliegenden Arbeit wird ein Framework konzeptioniert und entworfen, welches es mit Hilfe von Regeln erlaubt, den Zugriff auf RDF-Repositories zu beschränken. In Kapitel 2 werden zunächst die bereits existierenden Ansätze für die Zugriffssteuerung vertraulicher Daten im Sematic Web vorgestellt. Des Weiteren werden in Kapitel 3 grundlegende Mechanismen und Techniken erläutert, welche in dieser Arbeit Verwendung finden. In Kapitel 4 wird die Problemstellung konkretisiert und anhand eines Beispielszenarios analysiert. Nachdem Anforderungen und Ansprüche erhoben sind, werden in Kapitel 6 verschiedene Lösungsansätze, eine erste Implementierung und ein Prototyp vorgestellt. Abschließend werden die Ergebnisse der Arbeit und die resultierenden Ausblicke in Kapitel 7 zusammengefasst.

2 Verwandte Arbeiten

De Coi et. al beschreiben ein Framework zum Aufstellen von Zugriffsregeln im Semantic Web [BDF+06] mit dem Namen „Protune“. Sie beschränken sich auf den Schutz von Web-Ressourcen, wie beispielsweise einem Dokument. Auf der Webseite des Projekts wird eine Live-Demo innerhalb einer Webanwendung angeboten. Der Versuch „Protune“ zu installieren und in eine eigene Anwendung zu integrieren, erwies sich als nicht trivial und ist aufgrund der unzureichend ausgefallenen Dokumentation gescheitert.
Tootoonchian et. al entwickelten speziell für soziale Netzwerke ein System namens „Lockr“ [TSGW09]. Es ermöglicht Zugriffsbeschränkungen auf der Basis sozialer Beziehungen zu erheben. Für Firefox existiert ein Plugin [1], mit welchem es möglich ist, den Zugang zu Fotos innerhalb von Flickr zu beschränken. Die Informationen über soziale Beziehungen werden vom sozialen Netzwerk „Facebook“ herangezogen. Eine Installation des Plugins innerhalb des aktuellen Firefoxs (3.6.8) ist nicht möglich, da es inzwischen veraltet ist.
Ein weiteres System ist PeLDS (Policy enabled Linked Data Server), ein von Hannes Mühleisen entwickelter Linked Data Server, welcher die Zugriffssteuerung auf die dort abgelegten RDF-Daten ermöglicht [Mü10]. Im Laufe seiner Arbeit entwickelt er eine auf SWRL (Semantic Web Rule Language) aufsetzende Sprache zur Formulierung von Zugriffsregeln. Im Gegensatz zur herkömmlichen SWRL-Syntax verwendet er eine der logischen Programmiersprache Prolog ähnliche Syntax. Eine evaluierte Implementierung von PeLDS existiert bereits. Es wird jedoch kein bereits existierender Triple Store verwendet, sondern eine eigene Implementierung basierend auf dem Jena Framework [jen10] und MySQL. Der generische Aspekt, die Zugriffsteuerung an weitere beliebige Triple Stores anzupassen, fehlt demnach.
In [LC08] und [ADCH+07] wird ein weiterer Lösungsansatz präsentiert, um den Zugriff auf RDF-Repositories zu regeln. Die vorgestellten Architekturen basieren beide auf dem Ansatz des Query Rewriting. Ein RDF-Query wird anhand von Zugriffsregeln ausgewertet und gegebenenfalls modifiziert, so dass ein Ergebnis erzeugt wird, für welches der Anfragende autorisiert ist. Da lediglich die Anfrage des Nutzers anhand der Regeln verändert und somit die geforderte Legitimation des Ergebnisses gewährleistet wird, ist dieser Lösungsansatz nicht von dem zu Grunde liegenden Triple Store abhängig. Die dort verwendeten Modelle zur Definition von Zugriffsregeln erlauben es den Zugriff feingranular auf der Basis von Triples zu steuern. Obwohl beide Arbeiten die gleiche Architektur verwenden, unterscheidet sich dennoch die Vorgehensweise der definierten Algorithmen.
In [LC08] zeigen Li et. al einen Algorithmus, welcher den Kopf (engl. head) der SPARQL-Anfrage und somit die dort defininierte Selektion betrachtet. Anhand der verschiedenen Triple Patterns innerhalb des Körpers (engl. body) der Anfrage werden Informationen über die Selektionsvariablen gesammelt, wie beispielsweise das Konzept. Zusätzlich kann hierfür auch die zu Grunde liegende Ontologie verwen-

[1] siehe http://addons.mozilla.org/de/firefox/addon/6815/

det werden. Anhand der gesammelten Informationen können entsprechende Zugriffsregeln ausfindig gemacht werden. Zu beachten ist dabei, dass jede Selektionsvariable von mindestens einer Regel betroffen sein muss. Ist dies nicht der Fall, wird sie aus der Selektion entfernt. Problematisch wird es beispielsweise bei einer Anfrage, welche alle verfügbaren Triples erfragt. Speziell dabei existiert lediglich ein Triple Pattern innerhalb des Körpers, welches aus drei Variablen besteht. Anhand dieser allgemeinen Formulierung können keine Informationen über die Selektionsvariablen gemacht werden. Folglich werden diese aus der Selektion entfernt und ein leeres Ergebnis wird zurückgegeben.
Im Gegensatz dazu überprüft der von Abel et. al vorgestellte Algorithmus in [ADCH+07] die Triple Patterns innerhalb des Körpers. Dabei werden zu jedem Pattern die entsprechenden Zugriffsregeln gesucht, dementsprechend weitere Patterns oder Filter hinzugefügt und gegebenenfalls der Kopf der Anfrage angepasst.
Einen ähnlichen Ansatz, welcher ebenfalls auf der Idee des Query Rewriting basiert, wird in [OCBCM10] vorgestellt. Durch das verwendete Modell zur Zugriffsteuerung kann jedoch nicht der anfragende Nutzer bei der Formulierung der Zugriffsregeln mit einbezogen werden. Dadurch lassen sich lediglich statische Regeln definieren, die beispielsweise unabhängig vom anfragenden Nutzer variieren.
Ebenso bietet Oracle die Möglichkeit den Zugriff auf RDF-Daten zu regeln [ora10]. Es werden zwei verschiedene Varianten vorgestellt; die Virtual Private Database (VPD) und die Oracle Label Security (OLS). Die Entscheidung für eine Variante ist erforderlich, da das Verwenden beider nicht möglich ist.
Die Alternative der VPD basiert auf der Verteilung von Nutzerrollen. Innerhalb der Datenbank können Zugriffsregeln für einzelne und Teile von Triples angelegt werden, welche bestimmten Nutzergruppen den Zugriff zu diesen gewährt. Stellt ein Nutzer eine SPARQL-Anfrage, wird sie mit Informationen zu dem anfragenden Nutzer modifiziert und anschließend ausgewertet.
Auch die zweite Möglichkeit der OLS basiert auf Nutzerrollen. Jedem Nutzer wird ein Label zugeordnet, ebenso wie den zu schützenden Triples. Wird eine SPARQL-Anfrage gestellt, wird ihr das Label des anfragenden Nutzers beigefügt. Bei der Auswertung werden die Labels des Nutzers mit denen der angefragten Triples verglichen. Falls das Label des Nutzers dominiert, wird der Zugriff gewährt. Auch diese Möglichkeit lässt es zu, einzelne und bestimmte Teile von Triples zu schützen. Beide vorgestellten Ansätze basieren jedoch auf Oracle Datenbanken, wodurch eine Weiterverwendung innerhalb von anderen Triple Stores ausgeschlossen ist.
Baader et. al beschreiben in [BKP09] eine Möglichkeit, verschiedenen Nutzern nur spezielle Views auf eine Ontologie zu gewähren. Anstatt Sub-Ontologien zu extrahieren, wird versucht die einzelnen Axiome mit einem Label zu versehen. Somit kann anhand bestimmter Kriterien überprüft werden, ob das Label eines Axioms zu der entsprechenden Sub-Ontologie eines Nutzers gehört. Wie eine Anbindung an ein bereits existierendes RDF-Repository aussehen könnte, wird in dieser Arbeit nicht berücksichtigt. Ebenso wird nur eine experimentelle Implementierung vorgestellt, welche öffentlich nicht verfügbar ist.

Einen weiteren Ansatz zur Beschreibung von Zugriffsregeln im Semantic Web beschreiben Hollenbach, Presbrey und Berners-Lee in [HPBL09]. Sie verwenden RDF Metadaten, um den Zugriff für die auf einem Server abgelegten RDF-Dateien zu steuern. Jede RDF-Datei erhält eine RDF-Metadaten-Datei mit einer Access Control List (ACL). Innerhalb dieser können anhand des Uniform Resource Identifier (URI) vom Foaf-Profil des anfragenden Nutzers Zugriffsrechte zugewiesen werden. Alternativ besteht die Möglichkeit für eine Gruppe von Nutzern Zugriffsrechte zu definieren, festgelegt durch eine OWL-Klasse wie beispielsweise eine Foaf-Person. Insgesamt werden drei Typen von Zugriffsregeln unterschieden: Lesen, Schreiben und Kontrolle. Unter Kontrolle wird ein besonderes Schreiben verstanden. Hat ein Nutzer die Kontrolle über eine RDF-Datei, ist er autorisiert die zugehörige ACL zu verändern. Bisher existiert eine Implementierung in Form eines Apache Tomcat Moduls. Da dieser Ansatz jedoch lediglich auf dem Schutz von RDF-Dateien basiert, kann der Zugriff für einzelne Triples nur durch Umwege gesteuert werden.
Die in [KFJ03] beschriebene Sprache „Rei“ bildet eine weitere Möglichkeit zur Formulierung von Regeln für die Zugriffssteuerung auf RDF-Daten. Ähnlich zur Semantic Web Rule Language (SWRL) werden die Regeln in OWL-Lite beschrieben. Der Fokus liegt dabei jedoch beim Erstellen von Ontologien um verschiedene Formen von Zugriffsregeln auszudrücken und nicht bei der Formulierung der Regeln speziell für bestimmte Anwendungsfälle.
Zhang et. al definieren in [ZAGC09] das Modell Relation Based Access Control (RelBAC) für Zugriffskontroll-Szenarien innerhalb des Web 2.0 mit Hilfe von Description Logics (DL). Dabei soll das typische Autorisierungsproblem bei der Zugriffskontrolle vollständig beschrieben werden. Bei der Formulierung der Regel fehlt jedoch, bezogen auf RDF-Triples, das Betrachten des Prädikats. Vielmehr wird innerhalb von RelBAC die Beziehung zwischen Subjekt und Objekt als erlaubte Zugriffsaktion angesehen. Ein erläuterndes Beispiel zeigt die Regel *Update(david; mb903ll/a)*, welche dem Nutzer *David* erlaubt die Ressource *mb90311/a* zu verändern. Eine solche Art der Formulierung reicht jedoch nicht aus, um den Zugriff auf RDF-Repositories feingranular zu beschränken.
Ein Konzept zur Authentifizierung von Nutzern eines Triple Stores wird von Story et. al in [SHJJ09] beschrieben. Dieser auf dem Secure Socket Layer (SSL) aufsetzende Ansatz wird als FOAF+SSL bezeichnet. Auf der Webseite des Projekts werden lauffähige Demos inklusive entsprechender Quelltexte vorgestellt. Der Vorgang bei der Authentifizierung mit FOAF+SSL verläuft passwortlos und basiert auf der Möglichkeit der Client-Authentifizierung. Ein Client möchte eine geschützte Ressource eines Foaf-Profils erfragen und initialisiert eine gesicherte Verbindung zum Server. Während des SSL-Handshakes fragt dieser nach dem Client-Zertifikat des Anfragenden. Im Gegensatz zur herkömmlichen Client-Authentifizierung überprüft der Server nicht die Signatur des Zertifikats, sondern liest lediglich den darin enthaltenen URI. Dieser verweist auf das Foaf-Profil des anfragenden Nutzers, welches den öffentlichen Schlüssel des Client-Zertifikats enthält. Der Web Server liest diesen und vergleicht ihn mit dem öffentlichen Schlüssel des vom Client übergebenen Zertifi-

kats. War der Vergleich erfolgreich, wird der Zugang zu der geschützten Ressource gewährt.

3 Grundlagen

Um vertrauliche Daten auf einem Computersystem ablegen zu können, werden an dieses bestimmte Anforderungen bezüglich der Sicherheit gestellt. Dazu gehören der Schutz der abgelegten Daten, die Authentifizierung eines Nutzers und die Regelung des Zugriffs durch Zugriffsrichtlinien. In Abschnitt 3.1 werden die zugehörigen Grundlagen erläutert.
Die im weiteren Verlauf verwendeten Technologien des Semantic Web werden in Abschnitt 3.2 vorgestellt. Hierzu gehört die Darstellung von Daten im RDF, die Verwaltung von Daten im RDF-Format mit Hilfe eines Triple Stores, sowie der Zugriff auf solche Daten mittels der Anfragesprache SPARQL.

3.1 Sicherheit

Um die Anforderungen an die Sicherheit des Computersystems besser aufzeigen zu können, wird in Definition 3.1 der Begriff *sicheres Computersystem* festgelegt. Hierbei werden die Forderungen Autorisierung von Nutzern und Zugriffskontrolle auf Informationen an das System gestellt.

Definition 3.1 *Ein sicheres Computersystem ist ein Computersystem, das mittels spezieller Funktionen den Zugriff auf Informationen in einer Art und Weise kontrolliert, so dass nur autorisierte Nutzer Zugriff auf diese Informationen haben [QZW+85].*

Die Voraussetzung für die Autorisierung ist die Authentifizierung. Detaillierter betrachtet muss man die Identität des Nutzers überprüfen und garantieren, bevor dieser für bestimmte Aktionen autorisiert wird. Dies geschieht während der Authentifizierung. Ist der Nutzer identifiziert, wird er anhand der Zugriffsrichtlinien autorisiert.
Definition 3.2 verdeutlicht die Anforderung an Zugriffsrichtlinien. Mit Hilfe der enthaltenen Regeln kann festgelet werden, ob ein Benutzer autorisiert ist bestimmte Informationen zu erfragen. Für die Zugriffsbeschränkung von vertraulichen Daten sind diese zwingend erforderlich.

Definition 3.2 *Eine Zugriffsrichtlinie (engl. Policy oder Access Control Policy) ist eine Menge von Regeln. Diese Regeln werden ausgewertet um festzustellen, ob einem Nutzer der Zugriff auf ein Datenobjekt erlaubt wird [QZW+85].*

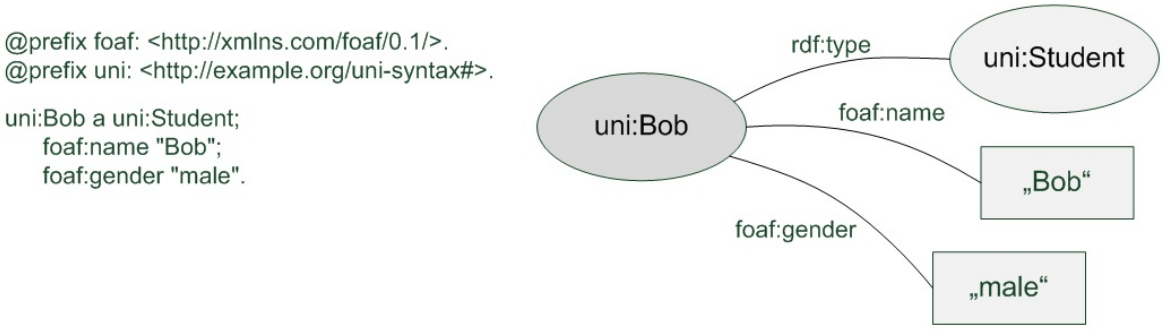
3.2 Semantic Web

Semantic Web Das klassische World Wide Web (WWW) ist ein „Web of documents“ [W3S10]. Im Gegensatz dazu versteht man unter dem Semantic Web ein „Web of data“[W3S10] als Erweiterung des WWW, welches als ein dezentralisiertes, dynamisches und standardisiertes semantisches Netzwerk (gerichteter Graph)

aufgefasst wird. Ein wichtiger Bestandteil des Semantic Web ist die Intention, Informationen im Web für den Computer verständlich und interpretierbar zu machen. Somit können bestimmte Daten, wie beispielsweise Orte und Personen miteinander in Beziehung gebracht werden [W3S10].

RDF Die Informationen im Semantic Web werden mit Hilfe des RDF repräsentiert. Dieses beschreibt einen Standard des World Wide Web Consortium (W3C), welcher zur Beschreibung von Web Ressourcen dient. Eine solche Beschreibung besteht aus einer Menge von Triples. Ein Triple selbst besteht aus einem Subjekt, einem Prädikat und einem Objekt. Zur Darstellung der Triples existieren bereits verschiedene Notationen. Abbildung 3.1 zeigt ein Beispieldokument im N3-Format inklusive dem entsprechenden RDF-Graphen.

Abbildung 3.1: Informationen in RDF-Darstellung inklusive Graph



Zu Beginn des Dokuments werden mit dem Stichwort „@prefix“ Abkürzungen für die verwendeten Namensräume definiert. So ist der Ausdruck „foaf:name“ äquivalent zu „http://xmlns.com/foaf/0.1/name“. Anschließend wird ein Knoten erzeugt, welcher vom Typ „uni:Student“ ist und als URI „uni:Bob“ erhält. Er repräsentiert somit einen Studenten, der zusätzliche Eigenschaften wie Name und Geschlecht erhält. Die Informationen von Name und Geschlecht werden als Literal bezeichnet, wobei Literal für den Datentyp „String“ steht. Somit besitzen sie keine URI, sondern bestehen lediglich aus einer beliebigen Zeichenkette. Weitere Datentypen wie „Float“ oder „Integer“ werden ebenfalls vom RDF unterstützt.

SPARQL SPARQL [SPA08] ist eine graph-basierte Anfragesprache für Informationen, die im RDF dargestellt sind. Sie erlaubt es in einer SQL-ähnlichen Syntax Anfragen an RDF-Graphen zu stellen. Insgesamt werden dabei vier verschiedene Anfragetypen unterschieden.

- **SELECT** - Ähnlich der relationalen Algebra wird bei einer Anfrage vom Typ SELECT eine Selektion und Projektion durchgeführt. Die zugehörige WHERE-

Klausel beschreibt, welche Informationen des RDF-Graphs in der Resultatmenge enthalten sein sollen. Im folgenden Listing ist eine solche Anfrage notiert:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX uni: <http://example.org/uni-syntax#>

SELECT ?name
WHERE {
        uni:s4081 foaf:name ?name.
}
```

Ähnlich wie in Abbildung 3.1 können auch innerhalb einer SPARQL-Anfrage mittels PREFIX Abkürzungen für die verwendeten Namensräume notiert werden. Nach SELECT folgen die Projektionsvariablen. Die angegebenen Triple Patterns innerhalb der WHERE-Klausel beschreiben die Selektion. Sie werden in einer N3-ähnlichen Syntax notiert. Das gezeigte Beispiel liefert in diesem Fall den Wert des Prädikates `foaf:name` der URI `uni:s4081`, welcher beispielsweise den Namen einer Person darstellt.

- **ASK** - Mit Hilfe einer ASK-Anfrage kann überprüft werden, ob eine bestimmte Selektion mindestens ein Ergebnis besitzt. Das Resultat besteht aus dem entsprechenden booleschen Wert (true, false). Folgendes Beispiel einer ASK-Anfrage liefert als Resultat `true`, wenn ein Triple mit dem Prädikat `foaf:name` und dem Objekt `"Alice"` existiert.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

ASK {
        ?x foaf:name "Alice".
}
```

- **CONSTRUCT** - Eine CONSTRUCT-Anfrage liefert als Resultat einen RDF-Graphen, welcher anhand eines Graph-Templates beschrieben wird. Für jedes Matching der Triple Patterns innerhalb der WHERE-Klausel werden die entsprechenden Variablen innerhalb des Graph-Templates ersetzt und schließlich zu einem Ergebnisgraph vereinigt. Folgendes Listing zeigt ein Beispiel für eine CONSTRUCT-Anfrage:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX ex: <http://example.org/ontology#>

CONSTRUCT {
        ?person ex:hasAunt ?aunt.
} WHERE {
        ?person ex:hasParent ?parent.
        ?parent ex:hasSister ?aunt.
}
```

Als Resultat würde diese Anfrage einen Graphen mit Triples der Form `<URI > ex:hasAunt ?aunt` ausgeben. Betrachtet man den semanitschen Aspekt der Anfrage, so ist die Tante einer Person die Schwester eines Elternteils dieser Person.

- **DESCRIBE** - DESCRIBE-Anfragen liefern als Resultat RDF-Graphen, die Informationen über bestimmte Ressourcen enthalten. Die zu beschreibenden Ressourcen werden nach dem Schlüsselwort DESCRIBE notiert. Folgendes Beispiel liefert einen Graph mit allen Triples, welche als Subjekt das Binding der Variable `?person` enthalten.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

DESCRIBE ?person {
        ?person foaf:name "Dave".
}
```

SPARQL 1.1 Update Die aktuelle Version von SPARQL bietet derzeit ausschließlich lesenden Zugriff auf Daten im RDF. Jedoch wird in der nächsten Version 1.1 von SPARQL [SPA10] die Möglichkeit für schreibenden Zugriff eingeführt. Zu den weiteren Neuerungen zählen Aggregatfunktionen wie SUM() und COUNT(), die Verwendung der aus SQL bekannten GROUP-BY-Klausel oder die Möglichkeit von Sub-Anfragen. Innerhalb von SPARQL 1.1 Update werden die im Folgenden erläuterten Anfragetypen untersützt.

- **INSERT DATA/DELETE DATA** - Mit Hilfe von INSERT DATA lassen sich konkrete Triples in einen RDF-Graph schreiben. Äquivalent dazu lassen sich mittels DELETE DATA konkrete Triples aus einem RDF-Graph löschen. Die Verwendung von Variablen ist in beiden Fällen nicht erlaubt.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX uni: <http://example.org/uni-syntax#>

DELETE DATA {
        uni:s4080 foaf:name "Carol".
        uni:s4081 foaf:name "Dave".
}
```

- **INSERT WHERE/DELETE WHERE** - Im Gegensatz zu INSERT beziehungsweise DELETE DATA bietet INSERT WHERE und DELETE WHERE die Möglichkeit Variablen zu verwenden und mittels der WHERE-Klausel bereits existierende Informationen aus einem RDF-Graph zu verwenden. Folgende Beispielanfrage fügt alle Mitglieder der Gruppe „alpha“ der Gruppe „gamma“ hinzu.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
INSERT {
        ?gamma foaf:member ?member.
} WHERE {
        ?gamma rdf:type foaf:Group;
                foaf:name "gamma".
        ?alpha rdf:type foaf:Group;
                foaf:name "alpha";
                foaf:member ?member.
}
```

- **DELETE/INSERT WHERE** - Mittels DELETE/INSERT WHERE können mit Hilfe einer Anfrage Updates innerhalb des RDF-Graphs realisiert werden. Folgende Beispielanfrage vertauscht die Gruppenmitglieder der Gruppe „alpha“ und „gamma“.

```
PREFIX uni: <http://example.org/uni-syntax#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

DELETE {
  ?alpha foaf:member ?alphaMember.
  ?delta foaf:member ?deltaMember.
} INSERT {
  ?alpha foaf:member ?deltaMember.
  ?delta foaf:member ?alphaMember.
}WHERE {
  ?alpha rdf:type foaf:Group;
    foaf:name "alpha";
    foaf:member ?alphaMember.
  ?delta rdf:type foaf:Group;
    foaf:name "delta";
    foaf:member ?deltaMember.
}
```

RDF Triple Store RDF Triple Stores [tri07] bilden die Datenbasis für das Semantic Web. Sie bieten einen persistenten Speicher für Informationen im RDF-Format. Um diese Informationen abzufragen, werden verschiedene Sprachen zu Verfügung gestellt wie zum Beispiel TQL, RQL, RDQL und SPARQL. Bekannte RDF Triple Stores sind beispielsweise Mulgara [mul10], Virtuoso [vir10] und Sesame [ses10].

4 Problembeschreibung und Analyse

Nachdem in Kapitel 3 die Grundlagen des Semantic Webs und der notwendigen Technologien zum Schutz von Informationen im RDF-Format innerhalb eines Triple Stores besprochen wurden, soll nachfolgend das Problem der Zugriffssteuerung konkretisiert und analysiert werden. Zunächst wird in Abschnitt 4.1 die Problemstellung anhand einer generischen Beschreibung weitgehend erläutert. Anschließend wird in Abschnitt 4.2 ein Szenario vorgestellt, welches die angesprochene Thematik in ihrer Anwendung verdeutlicht. Im Mittelpunkt stehen dabei Aufgaben aus dem Bereich der Universitätsverwaltung. Nach einer kurzen Beschreibung der Grundlagen werden fünf spezielle Anwendungsfälle vorgestellt, welche verschiedene Formen des Zugriffsschutzes verdeutlichen.

4.1 Generische Problembeschreibung

Um den Schutz von Informationen und Daten im RDF-Format gewährleisten zu können, müssen verschiedene Technologien verwendet werden. Diese werden in Form eines Frameworks vereint, welches im Folgenden als Access restricted Triple Store Framework (ArTS-F) bezeichnet wird.
Das Gesamtsystem bildet einen geschützten Triple Store. Hauptbestandteile sind der interne Triple Store, welcher im Backend als Datenspeicher fungiert und das ArTS-F, welches den Zugriff auf den Triple Store regelt. Für die im internen Triple Store enthaltenen Daten können Zugriffsbeschränkungen erhoben werden. Dies geschieht in Form von Zugriffsregeln, welche innerhalb des ArTS-F definiert werden. Daraus ergibt sich eine Menge von Zugriffsregeln, welche lediglich bestimmten Nutzern den Zugriff auf die RDF-Daten innerhalb des internen Triple Stores gewähren. Ein Nutzer kann folglich eine Anfrage an den geschützten Triple Store stellen, um für ihn zugängliche Daten zu erhalten.

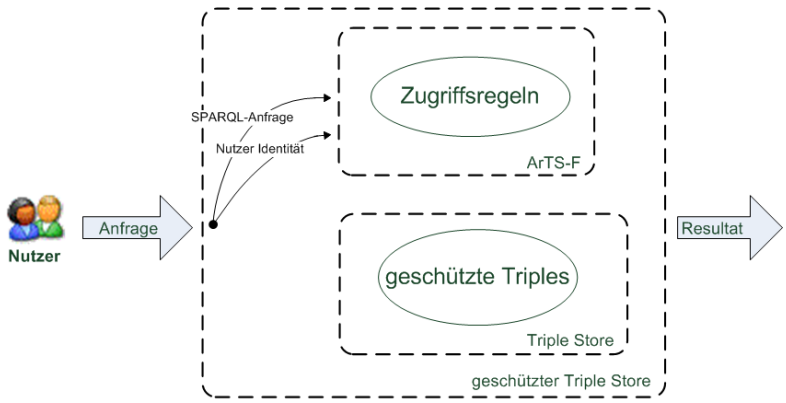


Abbildung 4.2: Generischer Architekturentwurf

Abbildung 4.2 gibt einen Überblick über die verschiedenen Bestandteile des Systems. Ein Nutzer stellt eine Anfrage an den geschützten Triple Store. Darin enthalten sind die Nutzeridentität in Form eines Zertifikats und die SPARQL-Anfrage. Diese Informationen werden zunächst an das ArTS-F übergeben und dort verarbeitet. Anhand der Zugriffsregeln und der Nutzeridentität enthält das Resultat der SPARQL-Anfrage nur Daten, für welche der anfragende Nutzer autorisiert ist.

4.2 Beispielszenario „Universitätsverwaltung“

Das folgende Beispielszenario beschreibt Verwaltungsaufgaben, die an einer Universität auftreten können. Daraus werden für das ArTS-F relevante Anwendungsfälle abgeleitet.

4.2.1 Einführung des Beispielszenarios

Die Universitätsverwaltung Koblenz verwaltet ihre Daten mit Hilfe eines Triple Stores. Ein wichtiger Aspekt bei Anfragen an die Daten ist die Zugriffsregelung. Manche Informationen dürfen aufgrund von Datenschutzrichtlinien nicht für jeden Nutzer zugänglich sein. Um entsprechende Zugriffsbeschränkungen definieren zu können wird das ArTS-F verwendet. Ein anfragender Nutzer ist entweder ein Angestellter der Universität oder ein Student. Er kann RDF-Daten erfragen und verändern, wenn er die entsprechenden Zugriffsrechte besitzt. Um einem Nutzer Zugriffsrechte zuordnen zu können, muss dessen Identität bekannt und authentifiziert sein. Abbildung 4.3 zeigt einen Ausschnitt der Ontologie der Universitätsverwaltung.

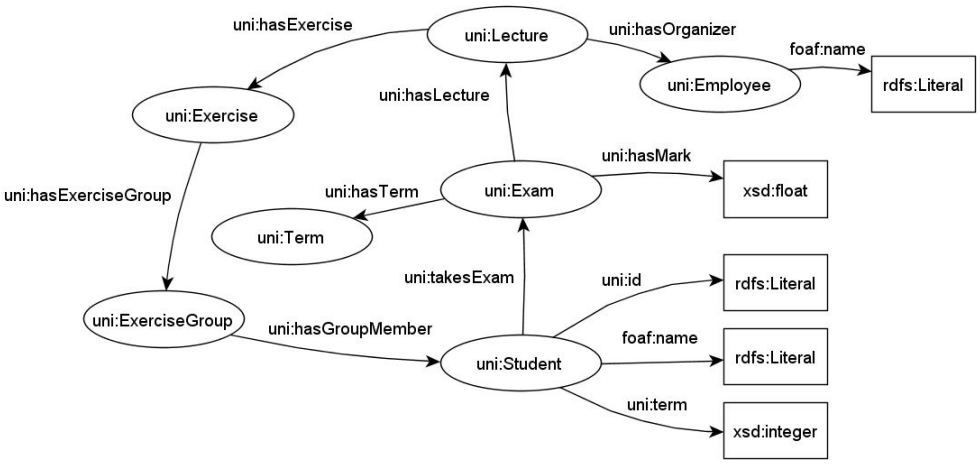


Abbildung 4.3: Ontologie der Universitätsverwaltung

Authentifizierung des Nutzers Bob arbeitet in einem wissenschaftlichen Institut an der Universität. Zu seinem Aufgabenbereich zählt die Leitung von Übungsgruppen zu bestimmten Vorlesungen und die Korrektur von Klausuren. Die zugehörigen Informationen und Daten werden im Triple Store der Universitätsverwaltung organisiert. Damit Bob Informationen erfragen kann, muss er sich mit Hilfe seines Zertifikats authentifizieren. Ist die Identität von Bob bestätigt, kann seine Anfrage zu bestimmten RDF-Daten berarbeitet werden. Ist die Authentifizierung fehlgeschlagen, wird seine Anfrage zurückgewiesen.

Klausurnoten Bob wurde erfolgreich authentifiziert und ist somit autorisiert, die Noten der letzten Klausuren einzusehen. Listing 4.1 zeigt einen Ausschnitt der entsprechenden Daten im RDF-Format in N3 Notation. In dieser Form können die Klausurnoten innerhalb des Triple Stores verwaltet und von Nutzern abgefragt werden.

Listing 4.1: Auszug aus der Datei uni_data.n3

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix uni: <http://example.org/uni-syntax#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
...
uni:e176 a uni:Employee;
        foaf:name "Bob".

uni:s4080 a uni:Student;
        foaf:name "Carol";
        uni:term "8"^^xsd:integer;
        uni:id "204110831";
        uni:takesExam uni:e138;
        uni:takesExam uni:e141;

uni:s4081 a uni:Student;
        foaf:name "Dave";
        uni:term "7"^^xsd:integer;
        uni:id "203220132";
        uni:takesExam uni:e139;
        uni:takesExam uni:e142.

uni:s3090 a uni:Student;
        foaf:name "John";
        uni:term "4"^^xsd:integer;
        uni:id "207201431";
        uni:takesExam uni:e140;
        uni:takesExam uni:e500;

uni:s2930 a uni:Student;
        foaf:name "Patrick".

uni:s312 a uni:Student;
```

```
        foaf:name "George".

uni:s5801 a uni:Student;
        foaf:name "Sarah".

uni:e138 a uni:Exam;
        uni:hasLecture uni:databases_ss10;
        uni:hasMark "2.3"^^xsd:float;
        uni:hasTerm uni:term_ss10.

uni:e139 a uni:Exam;
        uni:hasLecture uni:databases_ss10;
        uni:hasMark "4.0"^^xsd:float;
        uni:hasTerm uni:term_ss10.

uni:e140 a uni:Exam;
        uni:hasLecture uni:databases_ss10;
        uni:hasMark "3.3"^^xsd:float;
        uni:hasTerm uni:term_ss10.

uni:e141 a uni:Exam;
        uni:hasLecture uni:ai_ss10;
        uni:hasMark "1.0"^^xsd:float;
        uni:hasTerm uni:term_ss10.

uni:e142 a uni:Exam;
        uni:hasLecture uni:ai_ss10;
        uni:hasMark "2.0"^^xsd:float;
        uni:hasTerm uni:term_ss10.

uni:e500 a uni:Exam;
        uni:hasLecture uni:germ_ss09;
        uni:hasMark "3.0"^^xsd:float;
        uni:hasTerm uni:term_ss09.

uni:databases_ss10 a uni:Lecture;
        uni:hasOrganizer uni:e176;
        uni:hasExercise uni:databases_ex_ss10.

uni:ai_ss10 a uni:Lecture;
        uni:hasOrganizer uni:e176;
        uni:hasExercise uni:ai_ex_ss10.

uni:germ_ss09 a uni:Lecture.
        uni:hasExercise uni:germ_x_ss09.
...
```

Innerhalb einer Vorlesung existieren Übungsgruppen, welche in der Regel aus drei Studenten bestehen. Die Mitglieder einer Übungsgruppe müssen gemeinsam Aufgabenblätter bearbeiten und einreichen, um später die Zulassung zu der entsprechenden Klausur zu erhalten. Listing 4.2 zeigt, wie die Informationen der verschiedenen Übungsgruppen innerhalb des Triple Stores verwaltet werden.

Listing 4.2: Auszug aus der Datei uni_data.n3

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix uni: <http://example.org/uni-syntax#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
...
uni:databases_ss10 a uni:Lecture;
        ...
        uni:hasExercise uni:databases_ex_ss10.

uni:databases_ex_ss10 a uni:Exercise;
        uni:hasExerciseGroup uni:group_319;
        ...
        uni:hasExerciseGroup uni:group_389.

uni:group_319 a uni:ExerciseGroup;
        foaf:name "Delta";
        uni:hasGroupMember uni:s4080;
        uni:hasGroupMember uni:s4081;
        uni:hasGroupMember uni:s3090.

uni:group_320 a uni:ExerciseGroup;
        foaf:name "Alpha";
        uni:hasGroupMember uni:s2930;
        uni:hasGroupMember uni:s312;
        uni:hasGroupMember uni:s5801.
        ...
```

Zugriffsregeln Innerhalb des ArTS-F hat die Universitätsverwaltung Zugriffsbeschränkungen für vertrauliche RDF-Daten notiert, damit nur autorisierte Personen oder Organisationen diese vertraulichen Informationen einsehen können. In Tabelle 4.1 werden die verschiedenen Zugriffsrichtlinien dargestellt.

In der folgenden Aufzählung werden die einzelnen Zugriffsregeln natürlichsprachlich erläutert.

- *pol1* - Jedem Nutzer ist der lesende Zugriff zu Triples mit dem Prädikat `foaf:name` erlaubt.
- *pol2* - Ein Nutzer darf einsehen, ob ein Student an einer Klausur teilgenommen hat, wenn er Mitorganistor der entsprechenden Vorlesung ist.
- *pol3* - Ein Nutzer darf alle Eigenschaften einer Klausur einsehen, wenn er Mitorganisator der entsprechenden Vorlesung ist.
- *pol4* - Ein Nutzer darf alle Triples einsehen, bei welchen der URI seines Foaf-Profils das Subjekt ist.
- *pol5* - Ein Nutzer darf alle Eigenschaften einer Klausur einsehen, wenn er diese Klausur geschrieben hat.

```
pol1:
    allow read access to triple (?person, foaf:name, ?name).
```

```
pol2:
    allow read access to triple (?student, uni:takesExam, ?exam) if
        (?exam, uni:hasLecture, ?lecture) and
        (?lecture, uni:hasOrganizer, ?organizer) and
        ?requester == ?organizer.
```

```
pol3:
    allow read access to triple (?exam, ?pred, ?obj) if
        (?exam, uni:hasLecture, ?lecture) and
        (?lecture, uni:hasOrganizer, ?organizer) and
        ?requester == ?organizer.
```

```
pol4:
    allow read access to triple (?person, ?pred, ?obj) if
        ?requester == ?person.
```

```
pol5:
    allow read access to triple (?exam, ?pred, ?obj) if
        (?student, uni:takesExam, ?exam) and
        ?requester == ?student.
```

```
pol6:
    allow read access to triple (?group, uni:hasGroupMember, ?member) if
        ?requester == ?member.
```

```
pol7:
    allow insert access to triple (?exam, uni:hasMark, ?mark) if
        (?exam, uni:hasLecture, ?lecture) and
        (?lecture, uni:hasOrganizer, ?organizer) and
        ?requester == ?organizer.
```

```
pol8:
    allow delete access to triple (?exam, uni:hasMark, ?mark) if
        (?exam, uni:hasLecture, ?lecture) and
        (?lecture, uni:hasOrganizer, ?organizer) and
        ?requester == ?organizer.
```

Tabelle 4.1: Zugriffsregeln der Universitätsverwaltung innerhalb des ArTS-F

- *pol6* - Ein Nutzer darf die Mitglieder einer Übungsgruppe einsehen, wenn er selbst Mitglied dieser Übungsgruppe ist.
- *pol7* - Ein Nutzer darf die Note einer Klausur zum Triple Store hinzufügen, wenn er Mitorganisator der entsprechenden Vorlesung ist.
- *pol8* - Ein Nutzer darf die Note einer Klausur im Triple Store entfernen, wenn er Mitorganisator der entsprechenden Vorlesung ist.

4.2.2 Anwendungsfälle

Im Folgenden werden fünf Anwendungsfälle aus dem vorgestellten Beispielszenario abgeleitet. Diese sollen exemplarisch verdeutlichen, welche verschiedenen Mög-

lichkeiten von Anfragen existieren. Somit können im späteren Verlauf Probleme bei der Zugriffssteuerung konkretisiert und entsprechend analysiert werden.

1. Klausurnoten Bob möchte die Klausurnoten aller Studenten abfragen, zu welchen er Zugriff besitzt. Eine entsprechende SPARQL-Anfrage wird in Listing 4.3 beschrieben. Bob erhält als Ergebnis eine vollständige Liste mit den Namen aller Studenten inklusive der zugehörigen Klausurnoten der Vorlesungen „Datenbanken“ und „künstliche Intelligenz“ aus dem Sommersemester 2010.
Für die Ausführung der Anfrage sind die Zugriffsregeln *pol1*, *pol2* und *pol3* relevant. Da Bob als Organisator der Vorlesungen „Datenbanken“ und „künstliche Intelligenz“ im Sommersemester 2010 vermerkt ist, darf er alle Eigenschaften der zugehörigen Klausuren einsehen (*pol3*). Zusätzlich ist er autorisiert den Namen der Person einzusehen (*pol1*), die eine bestimmte Klausur geschrieben hat (*pol2*).

Listing 4.3: SPARQL-Anfrage für Klausurnoten aller Studenten

```
PREFIX uni: <http://example.org/uni-syntax#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?lecture ?name ?mark
WHERE {
        ?student uni:takesExam ?exam.
        ?student foaf:name ?name.
        ?exam uni:hasMark ?mark.
        ?exam uni:hasLecture ?lecture.
}
```

2. Durchschnittsnote eines Studenten Damit Studenten schnell einen Überblick über ihre Klausurnoten erlangen können, stellt das Prüfungsamt auf der Internetseite der Universität eine Webapplikation zur Verfügung. Um deren Funktionen nutzen zu können, muss sich jeder Student authentifizieren. Intern stellt die Applikation SPARQL-Anfragen an den geschützten Triple Store der Universitätsverwaltung inklusive der Identität des anfragenden Studenten.
Carol hat während ihres Studiums bereits mehrere benotete Klausuren geschrieben und möchte sich nun die Durchschnittsnote berechnen lassen. In Listing 4.4 wird die SPARQL-Anfrage dargestellt, welche die Webapplikation an den Triple Store stellt. Da Carol autorisiert ist ihre eigenen Daten (*pol4*) und zusätzlich auch ihre eigenen Noten einzusehen ((*pol5*), liefert die Anfrage die Durchschnittsnote aller von Carol geschriebenen Klausuren.

Listing 4.4: SPARQL-Anfrage für Klausurnote der Studentin Carol

```
PREFIX uni: <http://example.org/uni-syntax#>

SELECT (SUM(?mark)/COUNT(?mark) AS ?avg)
```

```
WHERE {
        ?student uni:takesExam ?exam.
        ?exam uni:hasMark ?mark.
}
```

3. Durchschnittsnote einer Klausur Carol möchte nun noch eine weitere Information erhalten. Die Webapplikation soll ihr nun die Durchschnittsnote all derjenigen Studenten ausgeben, welche die Datenbankklausur mitgeschrieben haben. Dieser Anwendungsfall unterscheidet sich deshalb von den bisher vorgestellten, weil Carol nicht autorisiert ist, die Klausurnoten der anderen Studenten zu erfragen, sondern lediglich ihre eigenen (*pol5*). Jedoch enthält der Ergebnisraum nicht die einzelnen personenbezogenen Noten, sondern nur die aus allen Klausurnoten berechnete Durchschnittsnote. Somit werden die maßgeblichen Datenschutzrichtlinien nicht verletzt und Carol ist es gestattet, die entsprechende SPARQL-Anfrage in Listing 4.5 zu stellen.

Listing 4.5: SPARQL-Anfrage für Durchschnittsnoten einer Klausur

```
PREFIX uni: <http://example.org/uni-syntax#>

SELECT (SUM(?mark)/COUNT(?mark) AS ?avg)
WHERE {
        ?exam uni:hasLecture uni:databases_ss10.
        ?exam uni:hasMark ?mark.
}
```

4. Teilnehmer der Übung einer Vorlesung Zum Abfragen von Informationen über die Zusammensetzung von Abgabegruppen sind Studenten nicht autorisiert, solange sie nicht selbst Mitglied dieser Gruppe sind (*pol6*). Carol möchte jedoch innerhalb einer Webapplikation erfahren, welche Personen an der Übung zur Vorlesung „Datenbanken“ teilgenommen haben. Listing 4.6 zeigt die entsprechende SPARQL-Anfrage, welche von der Webapplikation an den Triple Store gestellt wird. Obwohl die Teilnehmer der Übung anhand der Mitgliedschaft in einer Übungsgruppe ermittelt werden, wird die entsprechende Zugriffsregel nicht verletzt, da in der Ergebnismenge kein Zusammenhang zwischen den Studenten und ihrer Übungsgruppe erkennbar ist. Somit werden keine Datenschutzrichtlinien verletzt und die Anfrage wird korrekt ausgeführt.

Listing 4.6: SPARQL-Anfrage für Liste aller Teilnehmer der Datenbankübung

```
PREFIX uni: <http://example.org/uni-syntax#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?name
WHERE {
        uni:databases_ex_ss10 uni:hasExerciseGroup ?group.
```

```
        ?group uni:hasGroupMember ?student.
        ?student foaf:name ?name;
}
```

5. Klausurnoten verändern Carol hat während ihrer Klausureinsicht zur Vorlesung „Datenbanken“ aus dem Sommersemester 2010 festgestellt, dass Bob bei der Korrektur ein Fehler unterlaufen ist. Durch die neue Punktzahl verbessert sich ihre Note um „0.3“. Bob möchte nun die entsprechende Klausurnote innerhalb des Triple Stores der Universitätsverwalung aktualisieren. Hierzu verwendet er eine passende Webapplikation, bei welcher er die Noten verändern kann. Diese stellt eine entsprechende SPARQL-Anfrage an den Triple Store, welche in Listing 4.7 gezeigt wird.
Da Bob als Organisator der Vorlesung „Datenbanken“ im Sommersemester 2010 vermerkt ist, darf er zugehörige Klausurnoten eintragen (*pol7*) und entfernen (*pol8*). Zusätzlich darf er einsehen, dass Carol die Datenbankklausur mitgeschrieben hat (*pol2*) und ob diese auch der entsprechenden Vorlesung zugeordnet werden kann (*pol3*). Außerdem ist er dazu autorisiert die maßgebliche Note einzusehen (*pol3*).

Listing 4.7: SPARQL-Anfrage zum Aktualisieren der Klausurnote von Carol

```
PREFIX uni: <http://example.org/uni-syntax#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

DELETE {
        ?exam uni:hasMark ?mark.
} INSERT {
        ?exam uni:hasMark "2.0"^^xsd:float.
} WHERE {
        uni:s4080 uni:takesExam ?exam.
        ?exam uni:hasLecture uni:databases_ss10.
        ?exam uni:hasMark ?mark.
}
```

5 Anforderungen

Im Folgenden werden die Anforderungen für das ArTS-F definiert. Hierbei werden nichtfunktionale und funktionale Anforderungen unterschieden. Die funktionalen Anforderungen leiten sich aus den in Abschnitt 4.2.2 vorgestellten Anwendungsfällen ab.

5.1 Nichtfunktionale Anforderungen

Das ArTS-F soll bestimmte nichtfunktionale Anforderungen erfüllen. Eine der wichtigsten ist die Portierbarkeit und Übertragbarkeit des Systems, damit bereits existierende Triple Stores unterstützt werden. Dies verlangt eine generische Architektur, um die Unabhängigkeit des ArTS-F von einer konkreten Implementierung eines zu schützenden Triple Stores zu gewährleisten.
Ebenso wird besonders auf die Benutzbarkeit und Bedienung geachtet. Insbesondere die Formulierung der Regeln zum Schutz von Triples soll intuitiv und somit leicht erlernbar sein. Demnach sollte sich die verwendete Sprache als gut leserlich und flexibel präsentieren.
Die Leistung und Effizienz des ArTS-F sind ausschlaggebende Faktoren. Die Antwortzeit einer Anfrage sollte nicht zu hoch ausfallen. Diese ist stark von der Anzahl der Triples, welche im Triple Store verwaltet werden und der innerhalb des ArTS-F notierten Zugriffsregeln abhängig.
Folgende Liste fasst die soeben definierten nichtfunktionalen Anforderungen nochmals zusammen:

- **Portierbarkeit und Übertragbarkeit** - Leichte Integration innerhalb bereits existierender Triple Stores.
- **Flexibilität** - Flexible Sprache zur Formulierung von Zugriffsregeln.
- **Benutzbarkeit** - Intuitive, gut lesbare Regelsprache.
- **Leistung und Effizienz** - Antwortzeiten einer Anfrage möglichst gering halten.

5.2 Funktionale Anforderungen

Zunächst werden grundlegende funktionale Anforderungen für das ArTS-F definiert. Ein Nutzer kann Anfragen an den zu schützenden Triple Store stellen. Diese werden in der Anfragesprache SPARQL formuliert.
Bevor ein Nutzer eine Anfrage stellt, muss er authentifiziert werden. Somit sichert das ArTS-F mit Hilfe von FOAF+SSL die Identität eines anfragenden Nutzers.
Des Weiteren können Regeln formuliert werden, durch welche der Zugriff auf die

Triples innerhalb des Triple Stores beschränkt wird. Die Regeln werden in einer entsprechenden Regelsprache formuliert. Innerhalb einer Regel wird unterschieden, auf welche Art der Anfrage sich die Regel bezieht. Zu unterscheiden sind dabei das Abfragen, das Hinzufügen und das Löschen von Triples. Zusätzlich kann der Zugriff auf Triples erlaubt und verboten werden. Existiert für ein Triple keine Zugriffsregel, ist der Zugriff zu diesem nicht gewährt (deny by default).
Das ArTS-F kann die definierten Regeln anhand der Identität und der Inhalte des zu schützenden Triple Stores auswerten. Dabei wird bei einer Zugriffsverletzung der Zugriff auf alle erlaubten Triples gewährt, ohne diesen vollständig abzulehnen.
Die Verbindung zu dem internen Triple Store lässt sich konfigurieren, so dass die Portierbarkeit gewährleistet ist. Zusätzlich verhält sich das ArTS-F vollständig transparent, so dass kein Unterschied zwischen der Art und Weise besteht, wie Anfragen an den internen beziehungsweise geschützten Triple Store gestellt werden.
Speziellere funktionale Anforderungen, die sich vor allem auf die Flexibilität der Regeln beziehen, lassen sich aus den in Abschnitt 4.2.2 beschriebenen Anwendungsfällen ableiten.

Anforderung 1 Innerhalb des ersten Anwendungsfalls möchte Mitarbeiter Bob alle Klausnurnoten erfragen, zu welchen er Zugriff besitzt. Die zugehörige Regel besagt, dass er nur Zugriff zu Klausurnoten von Vorlesungen besitzt, bei welchen er Mitorganisator ist. Daraus ergibt sich die Anforderung, dass sich Regeln auf Teile eines Triples beziehen können, in diesem Fall auf das Prädikat und das Objekt. Außerdem können Variablen Teile von Triples ersetzen und innerhalb anderer Prädikate wiederverwendet werden.

Anforderung 2 Im zweiten Anwendungsfall möchte Carol die Durschnittsnote all ihrer bisher erhaltenen Klausurnoten abfragen. Die entsprechende Regel dazu lautet, dass ein Nutzer dazu autorisiert ist, seine eigenen Klausurnoten einzusehen. Demnach ergibt sich die Anforderung, dass der Zugriff auf Triples abhängig vom URI des Foaf-Profils des anfragenden Nutzers geregelt werden kann.

Anforderung 3 Beim dritten Anwendungsfall möchte Carol die Durchschnittsnote einer Klausur zu einer bestimmten Vorlesung erfahren. Obwohl die Zugriffsregel aus Anwendungsfall 2 dadurch verletzt werden würde, erhält Carol das korrekte Ergebnis, da dieses mit Hilfe einer Aggregatfunktion anonymisiert wird. Somit wird gefordert, dass das ArTS-F entweder anonymisierte Daten zur Verfügung stellen kann oder aber die Anfrage überprüft wird und das Verbot speziell für diese Anfrage aufgehoben wird.

Anforderung 4 Im vierten Anwendungsfall möchte Carol eine Liste aller Teilnehmer zu einer bestimmten Übung. Es exisitiert jedoch lediglich eine Regel, welche einem Nutzer erlaubt nur die Mitglieder einer Gruppe einzusehen, in welcher er

selbst Mitglied ist. Abbildung 5.4 verdeutlicht die Anfrage anhand einer Skizze. Die daraus folgende Anforderung legt fest, dass das ArTS-F entweder in der Lage ist, Anfragen zu überprüfen und somit die erforderlichen Rechte erteilt oder die Möglichkeit bietet das Triple zu umgehen und in diesem Fall einen direkten Pfad von *uni:Lecture* zu *uni:Student* herzustellen.

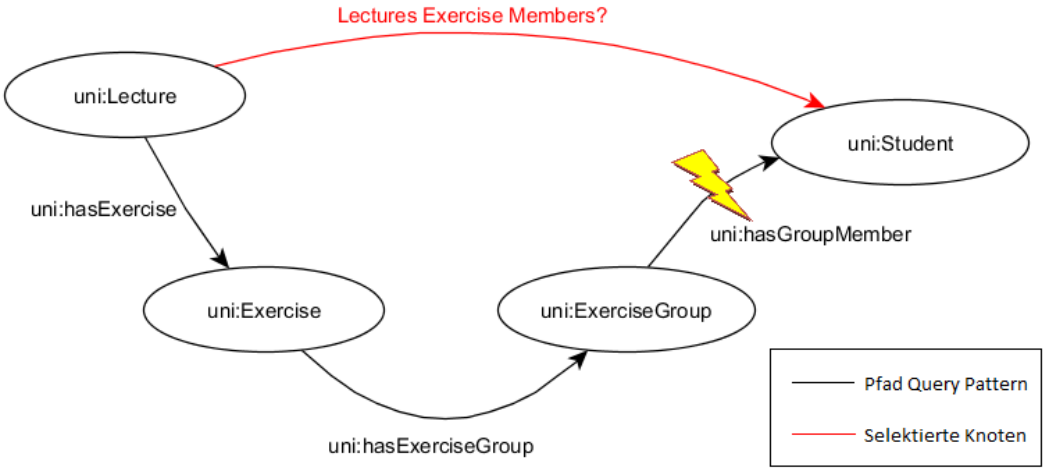


Abbildung 5.4: Anfrageskizze zu Anwendungsfall 4

Anforderung 5 Innerhalb von Anwendungsfall 5 möchte Bob eine Klausurnoten zu einer Vorlesung aktualisieren. Die entsprechende Zugriffsregel lautet, dass ein Mitarbeiter nur Klausurnoten von Vorlesungen verändern darf, bei welchen er Mitorganisator ist. Diese speziell für das Verändern von Daten definierte Regel ergibt die Anforderung, dass innerhalb einer Zugriffsregel die verschiedenen Arten von Anfragen unterschieden werden müssen (lesen, schreiben, löschen).

Folgende Liste fasst die aufgestellten funktionalen Anforderungen zusammen.

- **Anfragen** - Ein Nutzer kann SPARQL- und ARQ-Anfragen an den geschützten Triple Store stellen.
  - **Transparenz** - Das ArTS-F verhält sich transparent, so dass an den geschützten Triple Store in gleicher Art und Weise Anfragen gestellt werden können, wie an den internen Triple Store.
  - **Abfragen** - Ein Nutzer kann Triples abfragen.
  - **Verändern** - Ein Nutzer kann Triples verändern.
- **Konfiguration** - Die Verbindungsschnittstelle zum internen Triple Store ist konfigurierbar.

- **Authentifizierung** - Ein Nutzer wird, bevor er eine Anfrage stellt, mittels FOAF+SSL authentifiziert.
- **Regeln** - Es können Regeln formuliert werden, die den Zugriff auf die RDF-Daten regeln.
  - **Triple schützen** - Eine Regel kann sich auf Teile von Triples beziehen.
  - **Zugriff erlauben** - Eine Regel kann den Zugriff auf ein Triple erlauben.
  - **Zugriff verbieten** - Eine Regel kann den Zugriff auf ein Triple verbieten.
  - **Deny by Default** - Existiert keine Regel für ein Triple, so wird der Zugriff auf dieses nicht gewährt.
  - **Variablen** - Innerhalb von Regeln können Teile eines Triples durch Variablen ersetzt und weiterverwendet werden.
  - **Anfragender Nutzer** - Der Kontext einer Regel kann sich auf den anfragenden Nutzer beziehen (URI des Foaf-Profils).
- **Auswerten der Regeln** - Das ArTS-F kann definierte Regeln auswerten.
  - **Zugriffsverletzung** - Trotz Zugriffsverletzung soll der Zugriff auf erlaubte Triples gewährt werden, ohne diesen vollständig abzulehnen.

6 Entwurf des ArTS-F

Anhand der in Kapitel 4 definierten Anforderungen wird nun ein konkreter Entwurf des ArTS-F erarbeitet. Zu Beginn wird die verwendete Regelsprache zur Formulierung von Zugriffsregeln präsentiert. Anschließened werden drei verschiedene Architekturentwürfe vorgestellt und diskutiert. Mittels des ausgewählten Architekturentwurfs wird ein Algorithmus erarbeitet, welcher die Grundfunktionalität des ArTS-F darstellt. Im weiteren Verlauf wird die konkrete Implementierung inklusive einer Demo Applikation vorgestellt.

6.1 Regelsprache

Innerhalb des ArTS-F können mit Hilfe einer Regelsprache Zugriffsregeln definiert werden. Dadurch ist der Zugriff auf den Triple Store steuerbar. Die Anforderungen an die Regelsprache sind laut Kapitel 5.2 wie folgt definiert:

1. Eine Regel kann den Zugriff auf ein Triple erlauben.
2. Eine Regel kann den Zugriff auf ein Triple verbieten.
3. Eine Regel unterscheidet zwischen den Zugriffstypen „Lesen“, „Schreiben“ und „Löschen“.
4. Eine Regel kann sich auf Teile eines Triples beziehen.
5. Innerhalb einer Regel können Teile eines Triples durch Variablen ersetzt und weiterverwendet werden.
6. Der Kontext einer Regel kann sich auf den anfragenden Nutzer beziehen (URI des Foaf-Profils).

Mittels der in Listing 6.8 dargestellten „Erweiterte Backus-Naur-Form“ (EBNF) wird eine Regelsprache beschrieben, welche die erhobenen Anforderungen vollständig erfüllt. Die Übersichtlichkeit fördernd wurden folgende Ausdrücke nicht weiter beschrieben:

- <*String*> - Beliebige Zeichenkette, welche aus Zahlen und Buchstaben besteht.
- <*Intgeger*> - Ein Ganzzahliger Wert.
- <*Float*> - Eine Fließkommazahl.
- <*Var*> - Eine SPARQL-Variable.
- <*URI*> - Ein standardisierter URI [1].

[1] siehe http://tools.ietf.org/html/rfc3986

- *<Prefixed_URI>* - Eine URI, bei welcher der Namensraum mittels eines Prefixes ersetzt wurde.

Listing 6.8: EBNF zur Regelsprache des ArTS-F

```
<Policy> ::= <String> ":" ("allow" | "deny") <AccessType> "access to
    triple"
        <TriplePattern> [<Condition>] "."
<Condition> ::= "if" <Statement> ["and" <Statement>]*
<Statement> ::= <TriplePattern> | <Expr>

<AccessType> ::= "read"|"delete"|"insert"
<TriplePattern> ::= "(" <Node> "," <Node> "," <Node> ")"
<Expr> ::= <Arg> ("=="|"<="|"<"|">="|">"|"!=") <Arg>
<Arg> ::= <Var> | <Requester> | <Node> | <Const>
<Requester> ::= "?requester"
<Const> ::= <Integer> | <Float>
<Node> ::= <URI> | <Prefixed_URI> | <Var>
```

Jede Zugriffsregel besitzt ein Label, welches diese identifiziert. In der EBNF wird dieses durch „<String>“ beschrieben. Im Anschluss daran wird durch „allow“ oder „deny“ festgelegt, ob die Zugriffsregel den Zugriff auf ein Triple erlaubt oder verbietet (Anforderungen 1, 2). Durch „<AccessType>“ kann der Zugriffstyp der Regel definiert werden. Dabei steht Lesen (read), Schreiben (insert) und Löschen (delete) zur Auswahl (Anforderung 3). Daraufhin folgt ein Triple Pattern, beschrieben durch Subjekt, Prädikat und Objekt, auf welches der Zugriff geregelt werden soll. Dabei ist es möglich, Teile des Triples durch Variablen zu ersetzen (Anforderungen 4, 5). Zusätzlich kann eine Variable innerhalb eines Triple Pattern als „Wildcard“ verwendet werden, indem sie kein zweites mal innerhalb der Regel auftaucht. Optional können nun boolesche Ausdrücke und weitere Triple Patterns angegeben werden, welche mittels „and“ konjunktiv verknüpft werden. Innerhalb eines booleschen Ausdruckes kann anhand der vordefinierten Variable „?requester“ Bezug auf den anfragenden Nutzer genommen werden (Anforderung 6). In Listing 6.9 wird eine Zugriffsregel gezeigt, welche einem Nutzer erlaubt Gruppenmitglieder einer Gruppe zu löschen, wenn dieser Administrator dieser Gruppe ist.

Listing 6.9: Beispielregel

```
allowAdminDelete:
        allow delete access to triple (?group, foaf:member, ?
           member) if
                (?group, ex:hasAdmin, ?admin) and
                ?requester == ?admin.
```

6.2 Architektur

Insgesamt werden drei verschiedene Ansätze zur Realisierung des ArTS-F vorgestellt. Es folgt eine kurze Erläuterung der Entwürfe mit anschließender Prüfung der in Absatz 4.2.2 vorgestellten Anwendungsfälle.

6.2.1 Architekturentwurf - Ergebnisfilter

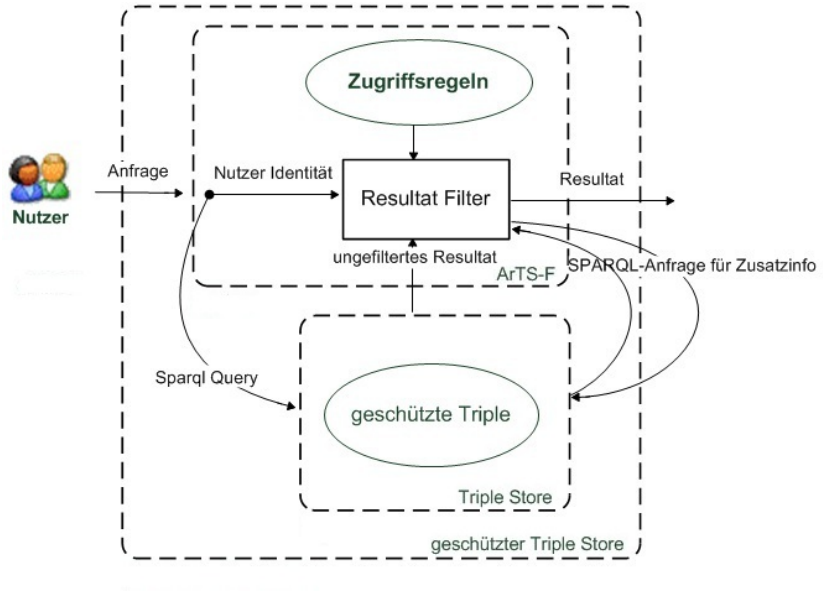


Abbildung 6.5: Architekturentwurf - Ergebnisfilter

Der in Abbildung 6.5 gezeigte Architekturentwurf wird als „Ergebnisfilter“ (EF) bezeichnet. Der Nutzer stellt eine Anfrage an den geschützten Triple Store. In der Anfrage enthalten ist die Identität des Nutzers und die SPARQL-Anfrage, um bestimmte RDF-Daten zu erhalten. Die SPARQL-Anfrage wird direkt an den internen Triple Store gestellt und liefert ein ungefiltertes Resultat. Dieses wird weitergeleitet an den ResultatFilter, welcher anhand der Zugriffsregeln die Ergebnisse entfernt, für welche der anfragende Nutzer nicht autorisiert ist. Dabei ist es gegebenenfalls nötig, mit Hilfe einer weiteren SPARQL-Anfrage Informationen über die Ergebnisse zu erfragen. Schließlich wird das gefilterte Resultat als Antwort an den Nutzer zurückgegeben. Ob der Ergebnisfilter als Architekturentwurf ausreichend geeignet ist, wird nun an der Möglichkeit zur Realisierung der in Abschnitt 5.2 vorgestellten Anforderungen beurteilt.

Realisierung von Anforderung 1 (EF) Die Anfrage aus Listing 4.3 liefert zunächst alle Klausurnoten, die innerhalb des Triple Stores existieren (siehe Tabelle 6.2.1). Darin sind jedoch Ergebnisse enthalten, für welche Bob nicht autorisiert ist. Diese zu filtern ist Aufgabe des Resultat Filters.

Innerhalb der Zugriffsregeln ist definiert, dass ein Triple der Form `?lecture uni:hasOrganizer uni:e176` zu jeder Vorlesung existieren muss, damit Bob Zugriff auf die entsprechenden Klausurnoten erhält. Demnach muss der Resultat-Filter eine weitere Anfrage (siehe Listing 6.10) an den internen Triple Store stellen, welche die passenden Informationen liefert. Als Ergebnis liefert diese die Vorlesungen `uni:databases_ss10` und `uni:ai_ss10`. Somit muss aus der ungefilterten

?lecture	?name	?mark
uni:databases_ss10	Carol	2.3
uni:ai_ss10	Carol	1.0
uni:databases_ss10	Dave	4.0
uni:ai_ss10	Dave	2.0
uni:databases_ss10	John	3.3
uni:germ_ss09	John	3.3

Tabelle 6.2: Ungefiltertes Resultat der SPARQL-Anfrage aus Anwendungsfall 1

Ergebnismenge die Zeile mit der Vorlesung `uni:germ_ss09` gelöscht werden und das gefilterte Ergebnis kann zurückgegeben werden.

Listing 6.10: SPARQL-Anfrage für zusätzliche Informationen zu den Ergebnissen der ursprünglichen Anfrage von Anwendungsfall 1

```
PREFIX uni: <http://example.org/uni-syntax#>

SELECT ?lecture
WHERE {
        ?lecture uni:hasOrganizer uni:e176.
}
```

Realisierung von Anforderung 2 (EF) Das Ergebnis der Anfrage aus Listing 4.4 lautet `2.6^^xsd:float` und soll die Durchschnittsnote aller Klausuren von Carol darstellen. Dieses Ergebnis ist jedoch falsch, weil die Anfrage die Noten aller Studenten erfragt und daraus die Durchschnittsnote berechnet. Um ein korrektes Ergebnis zu erzeugen, müsste der ResultatFilter die SPARQL-Anfrage kennen und diese anhand der Zugriffsregeln so modifzieren, dass lediglich die Klausurnoten von Carol zur Berechnung der Durchschnittsnote verwendet werden. Somit ist die Realisierung dieses Anwendungsfalls mit dem Ergebnisfilter nicht möglich.

Realisierung von Anforderung 3 (EF) Die Anfrage aus Listing 4.5 liefert als Ergebnis `3.2^^xsd:float`. Im Gegensatz zu Anwendungsfall 2 ist die gelieferte Durchschnittsnote korrekt, da hier nur die Klausurnoten der Datenbankklausur zur Berechnung verwendet wurden. Anhand der Zugriffsregeln wird nun überprüft, ob Carol autorisiert ist das Ergebnis zu erhalten. Da dieses lediglich aus einer Zahl besteht, müssen weitere Informationen eingeholt werden. Eine CONSTRUCT-Anfrage, wie in Listing 6.11 beschrieben, liefert gegebenenfalls benötigte Zusatzinformationen über die Herkunft. Da jedoch keine der in der CONSTRUCT-Anfrage definierten Triples existieren, bleibt die Ergebnismenge leer und keine Zugriffsregel wird verletzt. Das Ergebnis wird somit ungefiltert ausgegeben. Für diesen Anwendungsfall ist jedoch die Korrektheit des Ergebnis nicht garantiert. Wäre die Durchschnittsnote zufällig identisch mit der Note einer beliebigen Klausur eines anderen Studie-

renden, wird die für diesen Anwendungsfall definierte Zugriffsregel verletzt. Der ResultatFilter würde die Durchschnittsnote aus der Ergebnismenge löschen und diese leer zurückgeben. Somit ist die Realisierung dieses Anwendungsfalles nicht korrekt möglich.

Listing 6.11: SPARQL-Anfrage für zusätzliche Informationen zu den Ergebnissen der ursprünglichen Anfrage von Anwendungsfall 3

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

CONSTRUCT {
        ?o ?p "3.2"^^xsd:float.
}
WHERE {
        ?o ?p "3.2"^^xsd:float.
}
```

Realsierung von Anforderung 4 (EF) Wie zuvor wird auch beim vierten Anwendungsfall das Ergebnis der SPARQL-Anfrage aus Listing 4.6 betrachtet (siehe Tabelle 6.2.1), welches aus den Namen der Studenten besteht, die an der Übung zur Vorlesung „Datenbanken“ im Sommersemester 2010 teilgenommen haben.

?name
Carol
Dave
John
Sarah
Patrick
George

Tabelle 6.3: Resultat der SPARQL-Anfrage aus Anwendungsfall 4

Da der ResultatFilter anhand mangelnder Informationen nicht erkennen kann, ob eine Zugriffsregel verletzt wurde, müssen zusätzliche SPARQL-Anfragen an den internen Triple Store gestellt werden. Die in Listing 6.12 aufgeführte CONSTRUCT-Abfrage liefert alle Triples, bei welchen die gefundenen Literale das Objekt darstellen, in diesem Fall alle Triples mit dem Prädikat `foaf:name`. Diese lediglich anhand des Ergebnisses konstruierte Anfrage reicht jedoch nicht aus, damit der ResultatFilter erkennt, dass die innerhalb des ArTS-F definierte Regel *pol6* verletzt wurde. Hierfür muss eine komplexere Vorgehensweise entwickelt werden, bei welcher die Generierung der Anfragen auch die Zugriffsregeln mit einbezieht. Angenommen der ResultatFilter erkennt in diesem Fall die Verletzung der Regel *pol6*, werden die Literale „Sarah“, „Patrick“ und „George“ entfernt und somit nicht das erwartete Resultat geliefert. Demnach ist die Realisierung dieser Anforderung mit Hilfe des ResultatFilters nicht möglich.

Listing 6.12: SPARQL-Anfrage für zusätzliche Informationen zu den Ergebnissen der ursprünglichen Anfrage von Anwendungsfall 4

```
CONSTRUCT {
        ?s1 ?p1 "Carol".
        ?s2 ?p2 "Dave".
        ?s3 ?p3 "John".
        ?s4 ?p4 "Sarah".
        ?s5 ?p5 "Patrich".
        ?s6 ?p6 "George".
}
WHERE {
        ?s1 ?p1 "Carol".
        ?s2 ?p2 "Dave".
        ?s3 ?p3 "John".
        ?s4 ?p4 "Sarah".
        ?s5 ?p5 "Patrich".
        ?s6 ?p6 "George".}
```

Realisierung von Anforderung 5 (EF) Die in Listing 4.7 gezeigte Anfrage liefert kein Ergebnis zurück, welches gefiltert werden kann. Falls man jedoch die Veränderungen der Daten als Ergebnis ansieht, kann man dieses anhand der Zugriffsregeln überprüfen und gegebenenfalls filtern. Dabei muss der Zustand des Triple Stores vor und nach der Ausführung der Anfrage verglichen werden. Die Veränderungen werden anhand der Zugriffsregeln überprüft. Wird eine als nicht legitim erkannt, kann daraufhin eine INSERT- beziehungsweise DELETE-Anfrage an den internen Triple Store gestellt werden, um den entsprechenden Teil des Originalzustandes wiederherzustellen. Obwohl der Vergleich vom Zustand des Triple Stores vor und nach der Ausführung der Anfrage je nach Anzahl der Triples sehr rechenintensiv ausfallen kann, ist diese Anforderung mit Hilfe des Ergebnisfilters realisierbar.

6.2.2 Architekturentwurf - Query Transformation

Abbildung 6.6 zeigt einen weiteren Architekturentwurf, den der „Query Transformation“ (QT). Wie bereits bei dem vorherigen Entwurf beschrieben, stellt ein Nutzer eine Anfrage, bestehend aus SPARQL-Anfrage und Nutzeridentität, an den geschützten Triple Store. Anhand der Identität werden innerhalb des Query Transformators die nutzerspezifischen Zugriffsregeln aus der Menge aller Zugriffsregeln ausgewählt und auf die SPARQL-Anfrage angewendet. Diese wird so transformiert, dass sie lediglich Daten erfragt, für welche der Nutzer autorisiert ist. Anschließend wird die transformierte SPARQL-Anfrage an den internen Triple Store gestellt, welcher dann das Resultat liefert. Bezüglich der in Abschnitt 5.2 definierten Anforderungen wird nun die Realisierung des ArTS-F mit Hilfe des Architekturentwurfes „Query Transformation“ geprüft.

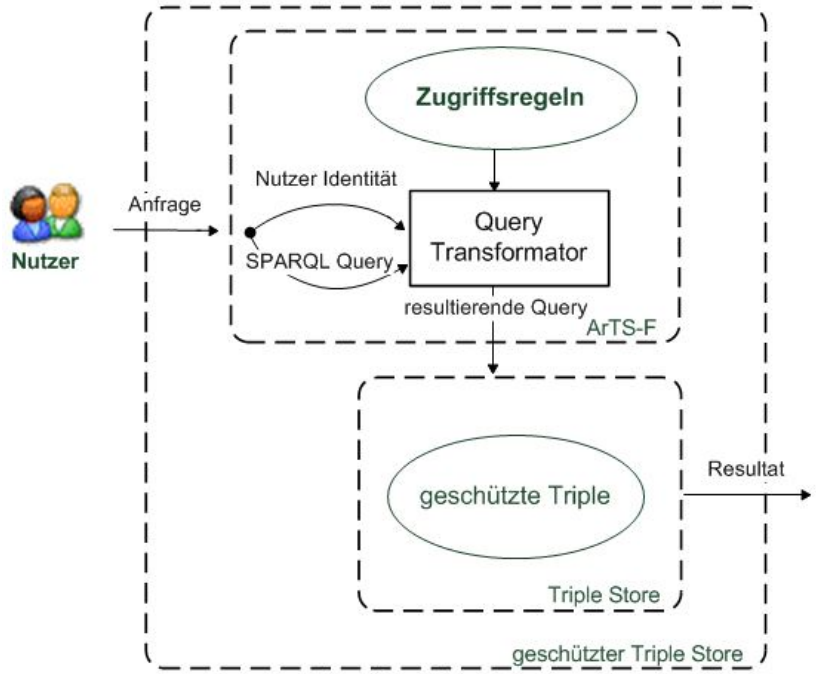


Abbildung 6.6: Architekturentwurf - Query Tranformation

Realisierung von Anforderung 1 (QT) Durch die allgemeine Form der SPARQL-Anfrage aus Listing 4.3 können in der Ergebnismenge Informationen enthalten sein, für welche Bob nicht autorisiert ist. Demnach muss der Query Transformator eine neue SPARQL-Anfrage generieren. Zunächst wird anhand der Zugriffsregeln nach einer Möglichkeit gesucht, um die usprüngliche SPARQL-Anfrage in eine legitime zu transformieren. In diesem Fall ist in der zurgehörigen Zugriffsregel die Bedingung notiert, dass Bob als Organisator zur Vorlesung der Klausuren vermerkt sein muss, welche er einsehen möchte. Somit genügt hier eine Erweiterung der Anfrage um das Triple `?lecture uni:hasOrganizer uni:e176`. Die neue Anfrage, welche in Listing 6.13 gezeigt wird, liefert lediglich Ergebnisse, für die Bob autorisiert ist.

Listing 6.13: Erweiterte SPARQL-Anfrage für alle Klausurnoten

```
PREFIX uni: <http://example.org/uni-syntax#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?lecture ?name ?mark
WHERE {
        ?student uni:takesExam ?exam.
        ?student foaf:name ?name.
        ?exam uni:hasMark ?mark.
        ?exam uni:hasLecture ?lecture.
  ?lecture uni:hasOrganizer uni:e176.
}
```

Realisierung von Anforderung 2 (QT) Ähnlich zu Anforderung 1 ist die Anfrage aus Listing 4.4 allgemein formuliert und liefert die Durchschnittsnote von allen innerhalb des Triple Stores notierten Klausurnoten. Die zutreffende Zugriffsregel fordert, dass ein Nutzer nur seine eigenen Klausurnoten einsehen darf. Die Erweiterung der Anfrage um das Triple Pattern `uni:s4080 uni:takesExam ?exam` bewirkt, dass lediglich die Durchschnittsnote der Klausuren von Carol berechnet wird. Somit lässt sich diese Anforderung mit Hilfe der Query Transformation realisieren.

Realisierung von Anforderung 3 (QT) Die in Listing 4.5 gezeigte SPARQL-Anfrage liefert die Durchschnittsnote der Datenbankklausur aus dem Sommersemester 2010. Da Carol lediglich dazu autorisiert ist ihr eigenes Klausurergebnis zu erfragen, muss eine zu der in Anforderung 2 äquivalente Query Transformation erfolgen. Als Ergebnis wird dabei lediglich die Note der Datenbankklausur von Carol zurückgegeben. Somit liefert die transformierte Anfrage zwar ein Ergebnis, jedoch nicht das gewünschte. Demnach ist diese Anforderung nicht mit Hilfe der Query Transformation realisierbar.

Realisierung von Anforderung 4 (QT) Die in Listing 4.6 notierte SPARQL-Anfrage liefert eine vollständige Liste aller Teilnehmer der Übung zur Datenbankvorlesung im Sommersemester 2010. Die relevante Zugriffsregel autorisiert Carol lediglich dazu, die Mitglieder ihrer eigenen Gruppe zu erfahren. Folglich wird die Anfrage um das Triple Pattern `?group uni:hasGroupMember uni:s4080` erweitert. Da als Ergebnis nur die Gruppenmitglieder von Carols Gruppe zurückgegeben werden, kann diese Anforderung mit Hilfe der Query Transformation nicht gelöst werden.

Realisierung von Anforderung 5 (QT) Durch die SPARQL-Anfrage in Listing 4.7 wird eine Veränderung an den Daten innerhalb des Triple Stores vorgenommen. Die relevanten Zugriffsregeln erlauben Bob die nötigen Veränderungen vorzunehmen, um eine Note zu aktualisieren, sowie die benötigten Informationen zu lesen. Somit befindet sich diese Anfrage bereits in einem legitimen Zustand und kann unverändert an den internen Triple Store gestellt werden.

6.2.3 Architekturentwurf - Graphview

Der Architekturentwurf „Graphview“ (GV) wird in Abbildung 6.7 gezeigt. Der Nutzer stellt eine Anfrage an den geschützten Triple Store. Die enthaltene Nutzeridentität wird vom TripleFilter verwendet, um mit Hilfe der Zugriffsregeln einen Teilgraph aus der Menge aller geschützten Triples zu extrahieren. Innerhalb des Teilgraphs existieren nur noch Triples, für welche der anfragende Nutzer autorisiert ist. Die in der Anfrage enthaltene SPARQL-Anfrage wird nun auf den Teilgraph angewendet. Das dadurch erzeugte Ergebnis wird als Resultat zurückgegeben. Wie bei

den bereits vorgestellten Architekturentwürfen wird nun die Eignung mittels der in Abschnitt 5.2 erarbeiteten Anforderungen beurteilt.

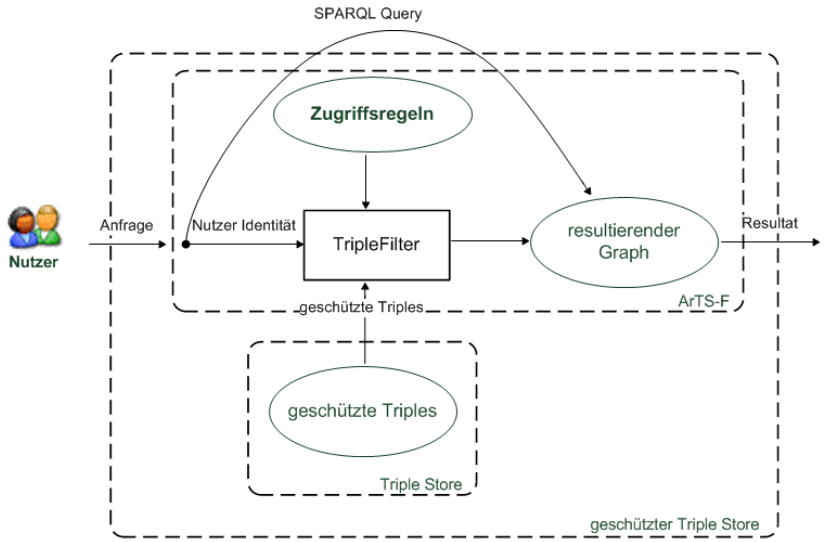


Abbildung 6.7: Architekturentwurf - Graphview

Realisierung von Anforderung 1 (GV) Anhand der Nutzerindentität von Bob und der im geschützten Triple Store enthaltenen Zugriffsregeln wird innerhalb des TripleFilters ein Teilgraph erzeugt, der nur Triples enthält, für welche Bob autorisiert ist. In diesem Fall ergibt sich dieser aus der Menge aller Triples, wobei Triples der Form `?exam uni:hasMark ?mark` nur dann enthalten sind, wenn Bob als Organisator der entsprechenden Vorlesung notiert ist. Ein solcher Teilgraph lässt sich mit Hilfe der in Listing 6.14 dargestellten CONSTRUCT-Anfrage erzeugen. Die SPARQL-Anfrage aus Listing 4.3 wird nun auf diesen Teilgraph angewendet und erzeugt eine legitime Ergebnismenge.

Listing 6.14: Teil der CONSTRUCT-Anfrage zur Erzeugung eines Teilgraphen für Nutzer Bob

```
PREFIX uni: <http://example.org/uni-syntax#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
        ?student uni:takesExam ?exam;
                 foaf:name ?name.
        ?exam uni:hasMark ?mark;
                 uni:hasLecture ?lecture.
} WHERE {
        ?student uni:takesExam ?exam;
                 foaf:name ?name.
        ?exam uni:hasMark ?mark;
```

```
                uni:hasLecture ?lecture.
        ?lecture uni:hasOrganizer uni:e176.
}
```

Realisierung von Anforderung 2 (GV) Mit Hilfe der Nutzeridentität von Carol und den entsprechenden Zugriffsregeln erzeugt der TripleFilter einen Teilgraph, welcher nur Noten von Klausuren enthält, die Carol geschrieben hat. Die CONSTRUCT-Anfrage in Listing 6.15 erzeugt den entsprechenden Teilgraph, an welchen nun die SPARQL-Anfrage aus Listing 4.4 gestellt werden kann. Diese berechnet somit die Durchschnittsnote lediglich aus den Klausurnoten von Carol.

Listing 6.15: Teil der CONSTRUCT-Anfrage zur Erzeugung eines Teilgraphen für Nutzer Carol

```
PREFIX uni: <http://example.org/uni-syntax#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
        ?student uni:takesExam ?exam.
        ?exam uni:hasMark ?mark.
} WHERE {
        ?student uni:takesExam ?exam.
        ?exam uni:hasMark ?mark.
        uni:s4080 uni:takesExam ?exam.
}
```

Realisierung von Anforderung 3 (GV) Innerhalb des dritten Anwendungsfalls fordert Studentin Carol Informationen an, für welche sie nicht autorisiert ist. Sie möchte die Durchschnittsnote der Klausur zur Vorlesung „Datenbanken“ aus dem Sommersemester 2010 erhalten. Innerhalb der SELECT-Klausel der in Listing 4.5 aufgeführten SPARQL-Anfrage sind jedoch Aggregatfunktionen enthalten, welche das Ergebnis anonymisieren. Damit die Korrektheit des Ergebnisses gewährleistet wird, darf der TripleFilter demnach beim Extrahieren keine Triples der Form `?exam uni:hasMark ?mark` entfernen, obwohl Carol für diese nicht autorisiert ist. Jedoch kennt der TripleFilter die SPARQL-Anfrage nicht mit der Folge, dass eine Überprüfung der SELECT-Klausel nicht möglich ist. Innerhalb des Teilgraphen existiert demnach nur die Klausurnote von Carol zur Vorlesung „Datenbanken“, weshalb sich die Durchschnittsnote lediglich aus dieser ergibt. Daher ist die Realisierung dieses Anwendungsfalles nicht möglich.

Realisierung von Anforderung 4 (GV) Der extrahierte Teilgraph innerhalb von Anforderung 4 enthält lediglich die Gruppenmitglieder der Übungsgruppe zur Datenbankvorlesung aus dem Sommersemester 2010, in welcher auch Carol Mitglied

ist. Ähnlich wie beim dritten Anwendungsfall kann hier anhand einer Untersuchung der SELECT-Klausel der SPARQL-Anfrage das Löschen dieser Triples verhindert werden. Jedoch ist dem TripleFilter die SPARQL-Anfrage nicht bekannt, weshalb diese Möglichkeit ausscheidet. Demnach scheitert auch die Realisierung des vierten Anwendungsfalles.

Realisierung von Anforderung 5 (GV) Die Realsierung von Anforderung 5 mit Hilfe der Graphview ist ähnlich der Vorgehensweise bei der Realisierung mit Hilfe des Ergebnisfilters. Hier wird jedoch die Veränderung des Zustandes der View von Bob vor und nach Ausführung der Anfrage anhand der Zugriffsregeln überprüft. Wird bei einer Veränderung eine Zugriffsregel verletzt, wird diese entsprechend mit DELETE oder INSERT rückgängig gemacht. Nachdem alle Zugriffsregeln geprüft wurden, werden die Änderungen ebenfalls am internen Triple Store vorgenommen.

6.2.4 Zusammenfassung

Zusammenfassend gibt Tabelle 6.2.4 einen Überblick darüber, welche Anforderungen mit welchem Architekturentwurf realisierbar sind.

	Ergebnisfilter	Query Transformation	Graphview
Anforderung 1	✓	✓	✓
Anforderung 2	X	✓	✓
Anforderung 3	X	X	X
Anforderung 4	X	X	X
Anforderung 5	✓	✓	✓

Tabelle 6.4: Überblick Architekturentwürfe

Mit Hilfe des Ergebnisfilters sind insgesamt nur zwei der fünf erhobenen Anforderungen erfüllt. Bei der Realisierung von Anforderung 3 wird entweder die Durchschnittsnote aller im Triple Store gehaltenen Klausurnoten zurückgegeben oder im ungünstigsten Fall ein leeres Ergebnis. Dadurch wird gegen die Anforderung verstoßen, dass bei einer Zugriffsverletzung der Zugriff auf erlaubte Triples gewährt werden soll, ohne diesen vollständig abzulehnen. Ein dieser Anforderung entsprechendes Resultat ist lediglich die Note der Datenbankklausur von Carol. Das Filtern des Ergebnisses reicht demnach nicht aus, um das geforderte ArTS-F zu realisieren und scheidet als grundlegende Architektur aus.
Der Entwurf der Graphview und der Query Transformation erfüllen jeweils drei der aufgestellten Anforderungen. Zusätzlich verstoßen sie gegen keine der weiteren erhobenen funktionalen Anforderungen. Unter dem Aspekt der Performanz und Leistung lassen sich jedoch Unterschiede erkennen. Die Antwortzeiten bei der Realisierung von Anforderung 5 mit Hilfe der Graphview können hoch ausfallen, da der Vergleich von zwei Graphen je nach Anzahl der Triples sehr rechenintensiv sein kann. Innerhalb der übrigen Anforderungen lassen sich Vorhersagen über

die Performanz nur schwierig abschätzen. Beim Entwurf der Query Transformation ist diese letztlich abhängig von der Länge der transformierten Anfrage, welche mit steigender Anzahl von Zugriffsregeln wächst. Im Gegensatz dazu ist die Performanz beim Entwurf der Graphview von der Anzahl der Triples innerhalb des Triple Stores abhängig. Die Berechnungszeit des extrahierten Graphs wächst mit steigender Anzahl von Triples. In [ADCH+07] wird diese Problematik kurz diskutiert und die Vorgehensweise mit Hilfe der Query Transformation als performanter betrachtet.
Eine bisher unbetrachtete Möglichkeit ist die Kombination von Query Transformation und Graphview. Die transformierte Anfrage wird dabei auf eine globale View angewendet, welche für jeden Nutzer gleich ist. Exemplarisch wird Anforderung 4 betrachtet, welche mittels der Erweiterung der View Funktionalität lösbar ist. Durch die in Listing 6.16 gezeigte CONSTRUCT-Anfrage wird ein direkter Pfad zwischen Übung und Übungsgruppenmitglieder hergestellt. Zusätzlich muss eine entsprechende Zugriffsregel definiert und die SPARQL-Anfrage aus Listing 4.6 angepasst werden. Auf ähnliche Art und Weise kann mit dieser Vorgehensweise auch Anforderung 3 erfüllt werden.

Listing 6.16: CONSTRUCT-Anfrage für globale View

```
PREFIX uni: <http://example.org/uni-syntax#>

CONSTRUCT {
        ?exercise uni:hasExerciseMember ?member.
} WHERE {
        ?exercise uni:hasExerciseGroup ?group.
        ?group uni:hasGroupMember ?member.
}
```

Eine Erweiterung der Query Transformation um die Funktionalität der Graphview erscheint somit am sinnvollsten. Im weiteren Verlauf der Arbeit wird jedoch lediglich die Hauptfunktionalität der Query Transformation erarbeitet. Die Erweiterung durch einen View Mechanismus wird nochmals in Abschnitt 7 als weiterer Ausblick aufgeführt.

6.3 Entwicklung des Algorithmus

Durch den in Abschnitt 6.2 erarbeiteten Architekturentwurf sind die Voraussetzungen für die Entwicklung eines passenden Algorithmus geschaffen. Entsprechend der Entscheidung für die Query Transformation muss dieser eine gegebene SPARQL-Anfrage so verändern, dass sie dem anfragenden Nutzer lediglich Triples als Ergebnis liefert, für welche dieser autorisiert ist. Als Grundlage wird dabei der in [ADCH+07] definierte Algorithmus verwendet. Im Folgenden werden zunächst notwendige Notationen eingeführt. Daraufhin wird die Vorgehensweise bei der Transformation zunächst ausschließlich für lesende Anfragen betrachtet und erläutert. Anschließend wird die Erweiterung des Algorithmus für die Transformation schrei-

bender Zugriffe vorgestellt.

6.3.1 SPARQL-Anfrage

Es wird vorausgesetzt, dass I, B und L Mengen von IRIs, Blank Nodes und Literalen sind. Des Weiteren repräsentieren die Mengen $Const$ und Var Konstanten und Variablen, so dass $Const = I \cup B \cup L$. Demnach kann ein RDF Graph als eine Menge von Triples $I \cup B \times I \times Const$ definiert werden [Pol07].
Eine SPARQL-Anfrage q wird notiert durch $q = (TF, TP, BE)$ mit:

- TF ist das Schablonen Format (engl. template format), welches entweder aus einer Menge von Variablen besteht (Projektion bei SELECT und DESCRIBE), einer Menge von Triples (CONSTRUCT, INSERT(DATA), DELETE(DATA)) oder aus der leeren Menge $\emptyset$ (ASK).
- TP sind Triple Patterns, welche in der WHERE-Klausel einer SPARQL-Anfrage notiert werden.
- BE ist ein boolescher Ausdruck, welcher innerhalb der SPARQL-Anfrage mittels FILTER notiert wird.

Ein Triple Pattern tp wird durch $tp = (s, p, o)$ ausgedrückt, wobei $s \in I \cup B \cup Var$, $p \in I \cup Var$ und $o \in Const \cup Var$. Zusätzlich liefert die Funktion $var(E)$ alle Variablen eines Ausdrucks E.
Mit Hilfe einer gegebenen Substitution θ lässt sich ein Triple Pattern tp zu einem neuen Triple Pattern tp' substituieren. Die Funktion $substituiere(tp, \theta)$ ist laut [ADCH+07] wie folgt definiert:

Definition 6.3 *Gegeben sei ein Triple Pattern $tp = (s, p, o)$ und eine Subsitution θ. Die Funktion $substituiere : tp, \theta \rightarrow (tp', BE)$ mit $tp' = (s', p, o')$ sei folgendermaßen definiert:*

$$- \quad \begin{cases} s' = v_s \; und \; be_s = (v_s = s), & wenn \; s \in I \cup B \\ s' = v_s \; und \; be_s = (v_s = Wert), & wenn \; s \in Var, [s = Wert] \in \theta \\ s' = s \; und \; be_s = \epsilon, & sonst \end{cases}$$

$$- \quad \begin{cases} o' = v_o \; und \; be_o = (v_o = o), & wenn \; o \in Const \\ o' = v_o \; und \; be_o = (v_o = Wert), & wenn \; o \in Var, [o = Wert] \in \theta \\ o' = o \; und \; be_o = \epsilon, & sonst \end{cases}$$

wobei v_s und v_o neue, zufällig generierte Variablen sind und $BE = \{be_s, be_o\}$.

Des Weiteren wird eine weitere Funktion für eine SPARQL-Anfrage q definiert, welche Triple Patterns mit Variablen beziehungsweise eine Menge von Variablen aus

den Triple Patterns der Anfrage kopiert.

Definition 6.4 *Gegeben sei eine SPARQL-Anfrage $q = (TF, TP, BE)$. Es existiert eine Funktion $kopiere : (TF, TP) \rightarrow TF'$:*
Für Anfragen, bei welchen TF aus Triple Patterns besteht, gilt:

$$TF' = \{tp_i | tp_i \in TP : tp_i \notin TF \wedge vars(tp_i) \neq \emptyset\}.$$

Für Anfragen, bei welchen TF aus einer Menge von Variablen besteht, gilt:

$$TF' = \{v_i | tp_i \in TP, v_i \in vars(tp_i) : v_i \notin TF \wedge v_i \notin TF'\}$$

6.3.2 Zugriffsregeln

Eine innerhalb des ArTS-F definierte Zugriffsregel pol bildet sich aus dem Prädikat $at \in \{allow, deny\}$, welches die Zugangsart definiert, dem Prädikat $gt \in \{read, insert, delete\}$, welches den Zugriffstyp festlegt, dem zu schützenden Triple Pattern $tp_s = (s, p, o)$, und optional einer Menge von Bedingungen, welche aus einer Anzahl von Triple Patterns $tp_i, i \in (1 \ldots n)$ und einer Anzahl von booleschen Ausdrücken $be_j, j \in (1 \ldots m)$ besteht. Somit kann eine Zugriffsregel folgendermaßen notiert werden:

$$pol = at(gt(tp_s)) \leftarrow tp_1, \ldots, tp_n, be_1, \ldots, be_m$$

Im Folgenden wird mit $H^T(pol)$ auf das zu schützende Triple Pattern $tp_s \in pol$ im Kopf und mit $B(pol)$ auf die booleschen Ausdrücke und Triple Patterns innerhalb des Körpers der Zugriffsregel pol verwiesen.
Eine Zugriffsregel pol kann für ein gegebenes Triple Pattern tp zutreffen, wenn für $H^T(pol)$ und tp ein allgemeinster Unifikator σ existiert (engl. most general unifier, mgu) [Bec06]. Der allgemeinste Unifikator für zwei Triple Patterns $tp_1 = (s_1, p_1, o_1)$ und $tp_2 = (s_2, p_2, o_2)$ wird folgendermaßen definiert:

Definition 6.5 *Gegeben seien zwei Triple Patterns $tp_1 = (s_1, p_1, o_1)$ und $tp_2 = (s_2, p_2, o_2)$. Man erhält ein unifiziertes Triple Pattern tp' mit Hilfe der Funktion $mgu : tp_1, tp_2 :\rightarrow \sigma$, welche wie folgt definiert ist:*

$$- \begin{cases} \sigma_s = \{s_1 = s_2\}, & wenn\ s_1 \in Var \wedge s_2 \in I \cup B \\ \sigma_s = \{s_2 = s_1\}, & wenn\ s_2 \in Var \wedge s_1 \in I \cup B \\ \sigma_s = \{s_1 = s_2\}, & wenn\ s_1 \neq s_2 \\ \sigma_s = \epsilon, & sonst \end{cases}$$

$$- \begin{cases} \sigma_p = \{p_1 = p_2\}, & wenn\ p_1 \in Var \wedge p_2 \in I \\ \sigma_p = \{p_2 = p_1\}, & wenn\ p_2 \in Var \wedge p_1 \in I \\ \sigma_p = \{p_1 = p_2\}, & wenn\ p_1 \neq p_2 \\ \sigma_p = \epsilon, & sonst \end{cases}$$

$$- \begin{cases} \sigma_o = \{o_1 = o_2\}, & wenn\ o_1 \in Var \wedge o_2 \in I \\ \sigma_o = \{o_2 = o_1\}, & wenn\ o_2 \in Var \wedge o_1 \in I \\ \sigma_o = \{o_1 = o_2\}, & wenn\ o_1 \neq o_2 \\ \sigma_o = \epsilon, & sonst \end{cases}$$

wobei $\sigma = \{\sigma_s, \sigma_p, \sigma_o\}$ *und* $tp' = tp_1\sigma$ *beziehungsweise* $tp' = tp_2\sigma$.

Ist eine Zugriffsregel pol für ein Triple Pattern tp zutreffend, ergeben sich aus den Triple Patterns $tp_i \in B(pol)$ und den booleschen Ausdrücken $be_j \in B(pol)$ (falls $B(pol) \neq \emptyset$) Bedingungen, welche den Zugriff auf das geschützte Triple $tp_s = H^T(pol)$ regeln.

Definition 6.6 *Gegeben sei ein Triple Pattern* tp *und eine Zugriffsregel* pol. *Des Weiteren* $\exists\sigma : \sigma = mgu(H^T(pol), tp)$. *Die Funktion* $extrahiere : tp, pol \rightarrow (TP, BE)$ *mit* $\forall tp_i \in B(pol), (tp'_i, BE'_i) = substituiere(\sigma, tp_i)$ *bildet sich laut [ADCH+07] wie folgt:*

- $PE = \{tp'_i | tp'_i \neq tp_i\}$
- $\widetilde{BE} = \{be_j\sigma | be_j \in B(pol) \wedge \exists tp_i : vars(be_j\sigma \cap (vars(tp_i\sigma) \cup vars(tp\sigma)) \neq \emptyset\}$
- $BE = BE'_i \cup \widetilde{BE} \cup \{\sigma_k | \sigma_k \in \sigma : \sigma_k = [X = Y] \wedge (X \in Const \vee Y \in Const)\}$

6.3.3 Algorithmus

Der Algorithmus in Anlehnung an [ADCH+07], welcher ausschließlich den Lesezugriff auf Triple Patterns innerhalb des Triple Stores regelt, sieht folgendermaßen aus:

Eingabe:
Eine SPARQL-Anfrage $q = (RF, TP, BE)$, eine Menge von Zugriffsregeln P und ein Zugangstyp at.

Ausgabe:
$TP_{neu} \equiv$ Menge von optionalen Feldern, welche sich aus P und at ergeben.
$BE_{neu} \equiv$ Konjunktion von booleschen Ausdrücken, welche sich aus P und at ergeben.

$regeln\_vorfiltern(q, P, at)$:
$BE_{oder} \equiv$ Disjunktion von booleschen Ausdrücken für ein Triple Pattern $tp \in TP$ bezüglich at.
$TP_{tp} \equiv$ Menge der Triple Patterns eines optionalen Feldes für ein Triple Pattern $tp \in TP$ bezüglich at.
$P_z \equiv$ Menge der zutreffenden Zugriffsregeln für ein Triple Pattern $tp \in TP$ bezüglich at.
$BE_{pol} \equiv$ Konjunktion von booleschen Ausdrücken einer für ein Triple Pattern $tp \in TP$ relevanten Zugriffsregel $pol \in P$ bezüglich at.

```
1  TP_neu = ∅;
2  BE_neu = ∅;
3  For each tp in TP do
4      TP_tp = ∅;
5      BE_oder = ∅;
6      P_z = {pol|pol ∈ P ∧ at(pol) == at ∧ gt(pol) == read ∧ ∃σ : σ = mgu(H^T(pol), tp)};
7      If(P_z == ∅ ∧ at == allow)
8          //Es existiert keine „allow“-Regel für das aktuelle Triple Pattern tp.
9          TP = {tp_i|tp_i ∈ TP : tp_i ≠ tp};
10         break;
11     If (∃pol ∈ P_z : extrahiere(e, pol) == (∅, ∅))
12         If(at == allow)
13             //Triple Pattern ist ohne Einschränkungen erlaubt.
14             BE_oder = TP_tp = ∅;
15             break;
16         If(at == deny)
17             //Zugriff zum aktuellen Triple Pattern tp ist nicht erlaubt.
18             TP = {tp_i|tp_i ∈ TP : tp_i ≠ tp};
19             break;
20     Else
21         For each pol ∈ P_z do
22             BE_pol = ∅;
23             (TP', BE') = extrahiere(e, pol);
24             If (TP' == ∅)
25                 BE_pol.fügeHinzu(BE', &&);
26             Else If (∃θ, tp_tp ∈ TP_tp, tp' ∈ TP' : θ = mgu(tp', tp_tp))
27                 For each tp' ∈ TP' do
28                     If (∃θ, tp_tp ∈ TP_tp : θ = mgu(tp', tp_tp))
29                         BE'θ;
30                 BE_pol.fügeHinzu(BE'θ, &&);
31             Else
32                 TP_tp.fügeHinzu(TP');
33                 BE_pol.fügeHinzu(BE', &&);
34             BE_oder.fügeHinzu(BE_pol, ||);
35         TP_neu.fügeHinzu(TP_tp);
36         BE_neu.fügeHinzu(BE_oder, &&);
```

Eingabe:
Eine SPARQL-Anfrage $q = (RF, TP, BE)$und eine Menge von Zugriffsregeln P.
Ausgabe:
Eine transformierte SPARQL-Anfrage $q' = (TF^+, TP^+, BE^+)MINUS(TF^-, TP^-, BE^-)$

$transformiere\_anfrage(q, TP^+_{neu}, BE^+_{neu}, TP^-_{neu}, BE^-_{neu})$
$TP^+_{neu} \equiv$ Menge von optionalen Feldern für „allow"-Regeln
$BE^+_{neu} \equiv$ Konjunktion von booleschen Ausdrücken für „allow"-Regeln.
$TP^-_{neu} \equiv$ Menge von optionalen Feldern für „deny"-Regeln
$BE^-_{neu} \equiv$ Konjunktion von booleschen Ausdrücken für „deny"-Regeln.

```
1   (TP+neu, BE+neu) = regeln_vorfiltern(q, P, allow);
2   (TP-neu, BE-neu) = regeln_vorfiltern(q, P, deny);
3   TF+ = TF- = kopiere(TP, TF);
4   TP+ = TP ∪ {OPTIONAL(tp+)|tp+ ∈ TP+neu};
5   BE+ = BE.fügeHinzu(BE+neu, &&);
6   TP- = TP ∪ {OPTIONAL(tp-)|tp- ∈ TP-neu};
7   BE- = BE.fügeHinzu(BE-neu, &&);
```

Zunächst wird die Methode $regeln\_vorfiltern$ betrachtet. Die äußere Schleife (3-36) durchläuft jedes einzelne Triple Pattern aus der WHERE-Klausel der SPARQL-Anfrage. Dabei werden, entsprechend dem angegebenen Zugangstyp, die zutreffenden Zugriffsregeln für das aktuelle Triple Pattern (6) und eine Menge optionaler Triple Patterns ermittelt, welche aus den zutreffenden Zugriffsregeln extrahiert werden (32). Falls keine Regel existiert, welche den Zugriff auf das aktuelle Triple Pattern erlaubt, wird dieses aus der WHERE-Klausel entfernt (7-10). Ist der Zugriff zum aktuellen Triple Pattern bedingungslos erlaubt, müssen für dieses keine Einschränkungen hinzugefügt werden (11-15). Zusätzlich wird überprüft, ob der Zugriff zum aktuellen Triple Pattern bedingungslos verboten ist und folglich das Triple Pattern aus der WHERE-Klausel entfernt (16-19).
Anschließend werden die zutreffenden Zugriffsregeln nacheinander abgearbeitet (21-34). Für jede Zugriffsregel wird eine Konjunktion der extrahierten booleschen Ausdrücke erzeugt (25, 30, 33). Besitzt ein Triple Pattern mehrere zutreffende Zugriffsregeln, wird eine Disjunktion aus den Konjunktionen der Regeln erstellt (34). Insgesamt wird schließlich eine Konjunktion erstellt, welche wiederum aus den Disjunktionen der einzelnen Triple Patterns entsteht (35) und eine Menge, welche für jedes Triple Pattern aus der WHERE-Klausel der SPARQL-Anfrage eine Menge von optionalen Triple Patterns enthält (36).
Die Methode $transformiere\_anfrage$ wendet zunächst die Methode $regeln\_vorfiltern$ an, um die entsprechenden Ergänzungen (OPTIONAL, FILTER, MINUS) für Zugriffsregeln des Zugangtyps „allow" (1) und „deny" (2) zu erhalten. In Zeile (3) wird die Funktion $kopiere$ aufgerufen. Dieser Schritt ist vor allem dann notwendig, wenn mittels entsprechendem TF alle möglichen Resultate für die Anfrage geliefert werden sollen (bei SELECT zum Beispiel „*"). Anschließend wird für jedes $tp_i \in TP^+_{neu}$ der WHERE-Klausel ein optionales Feld hinzugefügt (4). Dementsprechend wird dieser Schritt für TP^-_{neu} wiederholt (6). Falls die SPARQL-Anfrage bereits ein FILTER Feld besitzt, wird der konjunktive boolesche Ausdruck aus BE^+_{neu} mittels „&&"

angefügt, ansonsten wird ein neues FILTER Feld erzeugt (5). Äquivalent wird dieser Schritt für BE^{-}_{neu} wiederholt (7). Schließlich ergibt sich eine neue transformierte SPARQL-Anfrage mit $q' = (TF^{+}, TP^{+}, BE^{+})$, von welcher mittels eines MINUS-Feldes $MINUS(TF^{-}, TP^{-}, BE^{-})$ die nicht erlaubten Ergebnisse entfernt werden. Für den in Abschnitt 4.2.2 vorgestellten Anwendungsfall wird die transformierte SPARQL-Anfrage in Listing 6.17 dargestellt.

Listing 6.17: EBNF zur Regelsprache des ArTS-F

```
PREFIX  uni:  <http://example.org/uni-syntax#>

SELECT DISTINCT ?lecture ?student ?mark
WHERE
  { ?exam uni:hasMark ?mark .
    ?exam uni:hasLecture ?lecture .
    ?student uni:takesExam ?exam
    OPTIONAL
      { ?var_1  uni:hasOrganizer  ?var_2 .
        ?var_3  uni:takesExam  ?var_4 .
        ?var_5  uni:hasLecture  ?var_1 .}
    OPTIONAL
      { ?var_3  uni:takesExam  ?var_6 .
        ?var_1  uni:hasOrganizer  ?var_2 .
        ?var_7  uni:hasLecture  ?var_1 .}
    OPTIONAL
      { ?var_8  uni:hasLecture  ?var_1 .
        ?var_1  uni:hasOrganizer  ?var_2 .
        ?var_3  uni:takesExam  ?var_9 .}
    FILTER
    (
      (
        ((?exam = ?var_9) && (uni:e176 = ?var_3))
        || ((uni:e176 = ?var_2) && (?exam = ?var_8))
        || ( ?exam = uni:e176 )
      ) && (
        ((uni:e176 = ?var_2) && (?exam = ?var_7))
        || (?exam = uni:e176)
        || ((uni:e176 = ?var_3) && (?exam = ?var_6))
      ) && (
        ((?student = ?var_4) && (uni:e176 = ?var_3))
        || ((uni:e176 = ?var_2) && (?exam = ?var_5))
        || (?student = uni:e176)
        || (( uni:e176 = ?var_2 ) && (?student = ?var_5))
      )
    )
  }
```

Einen Spezialfall stellen jedoch DESBRIBE-Anfragen dar, welche zu dem vorgestellten Algorithmus nicht kompatibel sind. Eine Lösung bietet die Möglichkeit, diese zunächst in eine CONSTRUCT-Anfrage umzuwandeln und erst dann den Al-

gorithmus anzuwenden. Dabei wird folgende Vorgehensweise berücksichtigt:

Definition 6.7 *Gegeben sei eine DESCRIBE-Anfrage $q = (TF, TP, BE)$. Eine äquivalente CONSTRUCT-Anfrage $q' = (TF', TP, BE)$ bildet sich mit*

$$TF' = \{tp'_i | \forall v_i \in TF : tp'_i = (v_i, v_{p,i}, v_{o,i}\}$$

wobei $v_{p,i}$ und $v_{o,i}$ für jedes v_i neue, zufällig generierte Variablen sind.

6.3.4 Erweiterung des Algorithmus

Der zuvor vorgestellte Algorithmus ist ausschließlich zu Lese-Anfragen kompatibel. Entsprechend den Anforderungen an das ArTS-F muss dieser nun so erweitert werden, dass er ebenfalls auf Update-Anfragen angewendet werden kann. Dabei werden zunächst die verschiedenen Formen von Update-Anfragen innerhalb von SPARQL in Listing 6.18 betrachtet.

Listing 6.18: Verschiedene Formen einer Update-Anfrage innerhalb der Sprache SPARQL

```
1. (INSERT|DELETE) TF WHERE TP FILTER(BE)
2. (INSERT|DELETE) DATA TF
3. DELETE TF_d INSERT TF_i WHERE TP FILTER(BE), TF = {TF_d, TF_i}
```

Neben „read“ müssen mit „update“, „insert“ und „delete“ noch drei weitere Zugriffstypen unterschieden werden. Um den Zugriffstyp einer Anfrage q zu bestimmen, wird das Prädikat $gt(q)$ eingeführt.

Definition 6.8 *Gegeben sei eine SPARQL-Anfrage $q = (TF, TP, BE)$. Das Prädikat gt liefert den Zugriffstyp der Anfrage mit $gt(q) \in \{read, insert, delete, update\}$.*

Die Idee für die Erweiterung des Algorithmus liegt darin, den Zugriffstyp für eine Anfrage zu unterscheiden. Falls ein INSERT und/oder DELETE darin enthalten ist, müssen die zutreffenden Zugriffsregeln die Bedingung $gt(pol) = insert$ beziehungsweise $gt(pol) = delete$ erfüllen. Ähnlich wie bei einer Lese-Anfrage werden nun die Triple Patterns innerhalb von TF abgearbeitet und die der Zugriffsregeln entsprechenden Ergänzungen der Anfrage hinzugefügt. Für die Triple Patterns innerhalb von TP werden zutreffende Zugriffsregeln anhand der Bedingung $gt(pol) = read$ (vgl. Abschnitt 6.3.3) und ebenfalls die entsprechenden Ergänzungen der SPARQL-Anfrage hinzugefügt.
Mit Hilfe dieser Vorgehensweise können nun Update-Anfragen der Form (1) und (3) vor dem Ausführen anhand zutreffender Zugriffsregeln überprüft und gegebenenfalls transformiert werden. Einen Spezialfall bilden Anfragen der Form (2), welche keine WHERE-Klausel enthalten. Eine Lösung bietet jedoch die Möglichkeit, diese in eine Update-Anfrage der Form (1) umzuwandeln. Dabei wird zunächst ein leeres Gerüst einer Update-Anfrage der Form (1) erstellt und lediglich die Triples innerhalb von TF in diese eingefügt (vgl. Listing 6.19). Anschließend kann der erweiterte Algorithmus angewendet werden.

Listing 6.19: Umformung einer Update Anfrage der Form (2) in eine der Form (1)

```
// Update Anfrage Form (2)
(INSERT|DELETE) DATA TF

// äquivalente Update Anfrage der Form (1)
(INSERT|DELETE) TF WHERE TP, TP = ∅
```

In Listing 6.20 wird zunächst die erweiterte Methode $regeln\_vorfiltern$ betrachtet. Diese erhält zwei neue Eingabeparameter gt und TP'_{neu}, wobei gt den Zugriffstyp repräsentiert, anhand welchem die zutreffenden Regeln ausgewählt werden und TP'_{neu} die Menge der bereits hinzugefügten optionalen Triple Patterns. Dementsprechend ändert sich Zeile (6), in welcher die statische Bedingung $gt(pol) = read$ durch $gt(pol) = gt$ ersetzt wurde. Des Weiteren wird nun in den Zeilen (26-30) nach einem unifizierbaren Triple Pattern tp_{neu} für ein neues optionales Triple Pattern tp' nicht nur in TP_{neu} gesucht, sondern auch in TP'_{neu}.
Listing 6.21 zeigt die erweiterte Methode $transformiere\_anfrage$. Dabei wird zunächst überprüft, welchen Zugriffstyp die SPARQL-Anfrage besitzt. Demgemäß wird die Methode $regeln\_vorfiltern$ für die Triple Patterns innerhalb von TF aufgerufen und die entsprechenden Ergänzungen in TF^+_{neu}, TF^-_{neu}, BE^+_{neu} und BE^-_{neu} gesammelt. Standardmäßig werden für die Triple Patterns innerhalb von TP die Ergänzungen mit $gt = read$ berechnet und schließlich mit allen vorher berechneten Ergänzungen in TP^+, TP^-, BE^+ und BE^- vereint.

Listing 6.20: Veränderungen innerhalb der Methode $regeln\_vorfiltern$

Eingabe:
Eine Menge von Triple Patterns TP, eine Menge von Triple Patterns TP'_{neu},
eine Menge von Zugriffsregeln P, eine Zugangsart at und ein Zugriffstyp gt.

$regeln\_vorfiltern(TP, TP'_{neu}, P, at, gt)$:

6 $P_z = \{pol | pol \in P : at(pol) == at \wedge gt(pol) == gt \wedge$

26 Else If $(\exists \theta, tp_{neu} \in (TP'_{neu} \cup TP_{neu}), tp' \in TP' : \theta = mgu(tp', tp_{neu}))$
27 For each $tp' \in TP'$ do
28 If $(\exists \theta, tp_{neu} \in (TP'_{neu} \cup TP_{neu}) : \theta = mgu(tp', tp_{neu}))$
29 $BE'\theta$;
30 $BE_{pol}.fügeHinzu(BE'\theta, \&\&)$;

Listing 6.21: Erweiterte Methode $transformiere\_anfrage$

Eingabe:
Eine SPARQL-Anfrage $q = (RF, TP, BE)$ und eine Menge von Zugriffsregeln P.
Ausgabe:
Eine transformierte SPARQL-Anfrage $q' = (TF^+, TP^+, BE^+)MINUS(TF^-, TP^-, BE^-)$

$transformiere\_anfrage(q, TP^+_{neu}, BE^+_{neu}, TP^-_{neu}, BE^-_{neu})$
$TP^+_{neu} \equiv$ Menge von optionalen Feldern für „allow"-Regeln
$BE^+_{neu} \equiv$ Konjunktion von booleschen Ausdrücken für „allow"-Regeln.
$TP^-_{neu} \equiv$ Menge von optionalen Feldern für „deny"-Regeln
$BE^-_{neu} \equiv$ Konjunktion von booleschen Ausdrücken für „deny"-Regeln.

1 $TP^+_{neu} = TP^-_{neu} = \emptyset;$
2 $BE^+_{neu} = BE^-_{neu} = \emptyset;$
3 $If(gt(q) == insert)$
4 $\quad (TP^+_{neu}, BE^+_{neu}) = regeln\_vorfiltern(TF, P, allow, insert);$
5 $\quad (TP^-_{neu}, BE^-_{neu}) = regeln\_vorfiltern(TF, P, deny, insert);$
6 $If(gt(q) == delete)$
7 $\quad (TP^+_{neu}, BE^+_{neu}) = regeln\_vorfiltern(TF, P, allow, delete);$
8 $\quad (TP^-_{neu}, BE^-_{neu}) = regeln\_vorfiltern(TF, P, deny, delete);$
9 $If(gt(q) == update)$
10 $\quad (TP^+_{neu}, BE^+_{neu}) = regeln\_vorfiltern(TF_i, P, allow, insert);$
11 $\quad (TP'^+_{neu}, BE'^+_{neu}) = regeln\_vorfiltern(TF_d, P, allow, insert);$
12 $\quad TP^+_{neu} = TP^+_{neu} \cup TP'^+_{neu};$
13 $\quad BE^+_{neu}.fügeHinzu(BE'^+_{neu}), \&\&);$
14 $\quad (TP^-_{neu}, BE^-_{neu}) = regeln\_vorfiltern(TF_i, P, deny, insert);$
15 $\quad (TP'^-_{neu}, BE'^-_{neu}) = regeln\_vorfiltern(TF_d, P, deny, insert);$
16 $\quad TP^-_{neu} = TP^-_{neu} \cup TP'^-_{neu};$
17 $\quad BE^-_{neu}.fügeHinzu(BE'^-_{neu}), \&\&);$

19 $(TP'^+_{neu}, BE'^+_{neu}) = regeln\_vorfiltern(TP, P, allow, read);$
20 $TP^+_{neu} = TP^+_{neu} \cup TP'^+_{neu};$
21 $BE^+_{neu}.fügeHinzu(BE'^+_{neu}), \&\&);$
22 $(TP'^-_{neu}, BE'^-_{neu}) = regeln\_vorfiltern(TP, P, deny, read);$
23 $TP^-_{neu} = TP^-_{neu} \cup TP'^-_{neu};$
24 $BE^-_{neu}.fügeHinzu(BE'^-_{neu}), \&\&);$

26 $TF^+ = TF^- = kopiere(TP, TF);$
27 $TP^+ = TP \cup \{OPTIONAL(tp^+)|tp^+ \in TP^+_{neu}\};$
28 $BE^+ = BE.fügeHinzu(BE^+_{neu}, \&\&);$
29 $TP^- = TP \cup \{OPTIONAL(tp^-)|tp^- \in TP^-_{neu}\};$
30 $BE^- = BE.fügeHinzu(BE^-_{neu}, \&\&);$

6.4 Implementierung und Demo

Im Rahmen der Studienarbeit wurde das ArTS-F in der Programmiersprache Java implementiert. Das in Abbildung 6.8 vorgestellte Diagramm beschreibt die einzelnen Komponenten.

- **ArTS-F API** - Stellt die veschiedenen Funktionen des ArTS-F zur Verfügung.

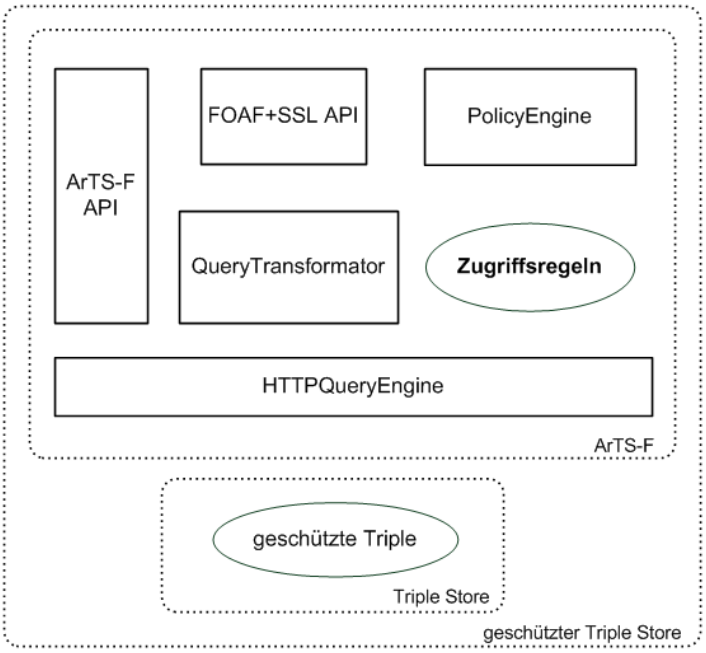


Abbildung 6.8: Komponentendiagramm des ArTS-F

- **FOAF+SSL API** - Authentifiziert den anfragenden Nutzer mittels FOAF+SSL und stellt dessen ID der PolicyEngine zur Verfügung.
- **PolicyEngine** - Wählt anhand der SPARQL-Anfrage und der ID des authentifizierten Nutzers die relevanten Zugriffsregeln aus und stellt diese dem QueryTransformator zur Verfügung. In dieser Komponente sind Teile der Methode *regeln_vorfiltern* des Algorithmus aus Abschnitt 6.3 zu finden.
- **QueryTransformator** - Erstellt die entsprechenden Ergänzungen anhand der Zugriffsregeln, welche von der PolicyEngine ermittelt wurden und transformiert die ursprüngliche SPARQL-Anfrage. In dieser Komponente sind ebenfalls Teile der Methode *regeln_vorfiltern* und *transformiere_anfrage* des Algorithmus aus Abschnitt 6.3 enthalten.
- **HTTPQueryEngine** - Verwaltet die Verbindung zum zu schützenden Triple Store über HTTP(S) und leitet die transformierte SPARQL-Anfrage an diesen weiter um letztendlich das Resultat zu erhalten und über die ArTS-F API zur Verfügung zu stellen.

Das Interface des ArTS-F bietet als Hauptfunktionen die Authentifizierung eines Nutzers anhand eines Zertifikats und das Ausführen der übergebenen SPARQL-Anfrage, welche intern mittels der definierten Zugriffsregeln transformiert wird. Da sich das Resultat für die verschiedenen Anfragetypen unterscheidet, wird für jeden eine Methode zur Verfügung gestellt. Als Resultat werden hier Objekte des Jena Frameworks verwendet. Alternativ wird die Möglichkeit geboten ein für jeden

Anfragetyp gleiches Resultat in Form eines XML formatierten Strings zu erhalten. In Listing 6.22 werden die Definitionen der Methoden des ArTS-F Interfaces aufgeführt.

Listing 6.22: Methoden des ArTS-F Interfaces

```
public interface IArTS_F {

        public boolean authenticateUser(X509Certificate
           certificate);

        public String getAuthenticatedUserId();

        public String getTransformedQuery();

        public String executeQuery();

        public Model executeConstructQuery();

        public Model executeDescribeQuery();

        public Boolean executeUpdateRequest();

        public Boolean executeAskQuery();

        public ResultSet executeSelectQuery();

        public HashSet<Policy> getMatchedPolicies();
}
```

Wird das ArTS-F als Webapplikation innerhalb eines Servlet Containers verwendet, so stellt es dem Anwender einen standardisierten SPARQL-Endpoint zur Verfügung (vgl. [W3C08]). Somit verhält es sich komplett Transparenz gegenüber dem anfragenden Nutzer, denn dieser muss lediglich seine SPARQL-Anfragen an die entsprechend neue Uniform Resource Locator (URL) senden. Das Sequenzdiagramm in Abbildung 6.9 stellt den Ablauf der Transformation und Ausführen einer SPARQL-Anfrage dar. Folgende Schritte werden dabei durchgeführt:

1. Das Zertifikat des Clients wird mit Hilfe der FOAF+SSL API überprüft.
2. Ist die Authentizität des anfragenden Nutzers bestätigt, kann die SPARQL-Anfrage ausgeführt werden.
3. Die authentifzierte Nutzeridentität (URI des Foaf-Profils) und die SPARQL-Anfrage werden an den Query Transformator weitergeleitet, welcher sie an die PolicyEngine durchreicht.
4. Die PolicyEngine wählt aus den Zugriffsregeln die für SPARQL-Anfrage und Nutzer Identität zutreffenden aus und liefert sie an den QueryTransformator.

5. Der QueryTransformator transformiert anhand der zutreffenden Zugriffsregeln die SPARQL-Anfrage und leitet sie an die ArTS-F API zurück.

6. Die transformierte SPARQL-Anfrage wird nun an die HTTPQueryEngine übergeben, welche diese per HTTP(S) an den zu schützenden Triple Store weiterleitet.

7. Das Resultat der Anfrage wird schließlich über die ArTS-F API an den Client übergeben.

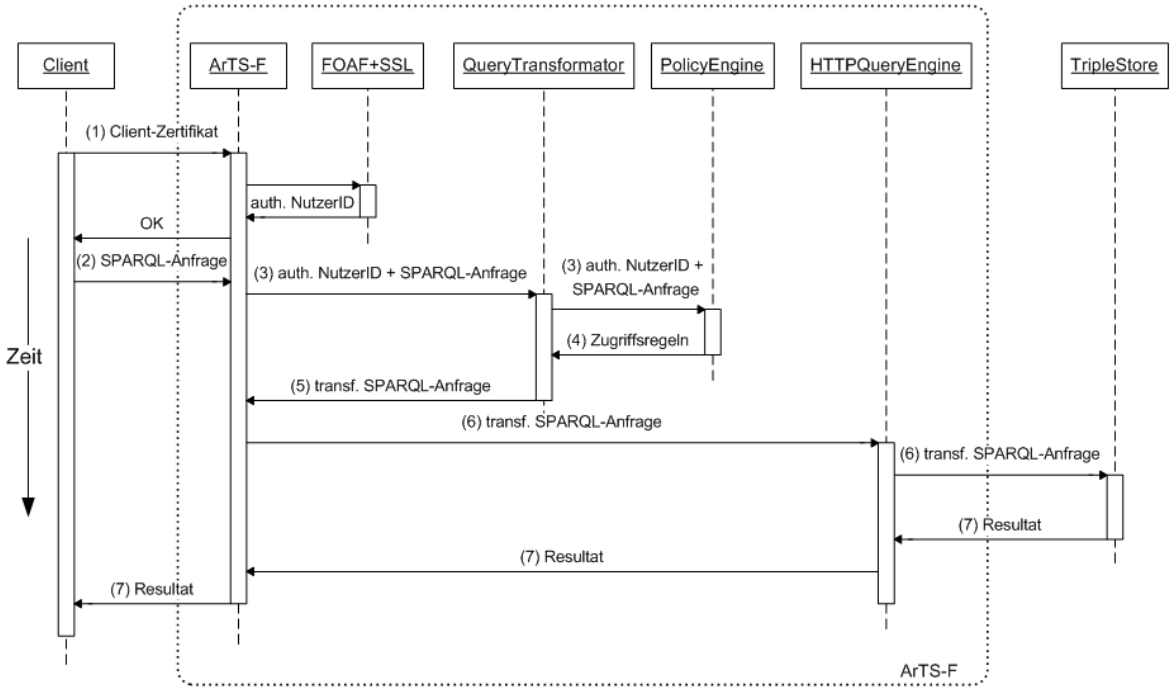


Abbildung 6.9: Ablauf einer Client Anfrage über das ArTS-F

Wird das ArTS-F zum ersten Mal verwendet, wird im Heimatverzeichnis des aktuellen Systembenutzers ein Ordner mit dem Name „ArTS-F“ angelegt, welcher die Konfigurationsdatei für das Framework enthält. Die Standardkonfiguration ist in Listing 6.23 aufgeführt. Insgesamt lässt sich diese in fünf Abschnitte einteilen:

(1) Spezifikation des SPARQL HTTP Endpunktes für Lese-Anfragen. Neben der URL wird noch der HTTP Parameternamen für die SPARQL-Anfrage sowie die HTTP-Methode (GET, POST) angegeben.

(2) Spezifikation des SPARQL HTTP Endpunktes für Schreib-Anfragen. Dabei werden URL, HTTP Parametername für SPARQL-Anfragen, sowie die HTTP-Methode (GET, POST) definiert.

(3) Relative oder absolute Pfadangaben zu Dateien, welche die Zugriffsregeln enthalten. Um die Möglichkeit zur Gliederung der Zugriffsregeln zu bieten, können mittels „;“ mehrere Dateien angegeben werden. Eine neue Zeile kann mit „\“ eingeleitet werden.

(4) Falls die HTTP-Endpunkte ebenfalls per FOAF+SSL geschützt sind, müssen der absolute beziehungsweise relative Pfad zu einem Client Zertifikat und das Export-Passwort angegeben werden. Standardmäßig ist dieser Abschnitt auskommentiert.

(5) Unter diesem Punkt können Abkürzungen für verwendete Namensräume definiert werden, welche innerhalb der Zugriffsregeln Verwendung finden. Wichtig dabei ist es, dass eine neue Zeile mittels „\“ gekennzeichnet wird und die Zuweisung innerhalb von Klammern steht. Standardmäßig sind häufig verwendete Namensräume, wie beispielsweise „rdf“, „rdfs“ und „foaf“, schon vordefiniert.

Listing 6.23: Standardkonfiguration des ArTS-F

```
#(1)
sparqlReadEndpoint=https://localhost/joseki/sparql
sparqlReadQueryParam=query
sparqlReadHTTPMethod=POST

#(2)
sparqlUpdateEndpoint=https://localhost/joseki/update/service
sparqlUpdateQueryParam=request
sparqlUpdateHTTPMethod=POST

#(3)
policies=policies.pol;policies2.pol;\
        C:\user\admin\policies\policies3.pol

#(4)
#trustedClientCert=cert.p12
#clientCertPassword=1337

#(5)
prefixMapping=\
        (rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#)\
        (rdfs=http://www.w3.org/2000/01/rdf-schema#)\
        (foaf=http://xmlns.com/foaf/0.1/)\
        ...
```

Zum Testen wurde eine Demo Web-Applikation entwickelt, mit deren Hilfe man über das ArTS-F SPARQL-Anfragen an einen über HTTP(S) erreichbaren Triple Store senden kann. Dieser muss lediglich in der Konfigurationsdatei eingetragen werden. Die Demo bietet bereits vordefinierte Anfragen und Zugriffsregeln, welche sich auf die in Abschnitt 4.2.2 vorgestellten Anwendungsfälle beziehen.

Abbildung 6.10 zeigt die Benutzeroberfläche. Als zusätzliche Option kann das Ausgabeformat der Anfrage festgelegt werden. Hier stehen „XML“, „JSON“ und „Text“ zur Auswahl. Mittels der Checkbox „Info“ werden nach der Ausführung der SPARQL-Anfrage weitere Informationen, wie die Berechnungszeit für die Transformation der Anfrage, die Antwortzeit des internen Triple Stores, die für die Anfrage relevanten Zugriffsregeln und die transformierte Anfrage angezeigt (vgl. Abbildung 6.11. Mit Hilfe des Buttons „Modify“ lassen sich die Zugriffsregeln modifzieren, welche in der Konfigurationsdatei angegeben wurden.

Abbildung 6.10: Benutzeroberfläche der Demo Web-Applikation

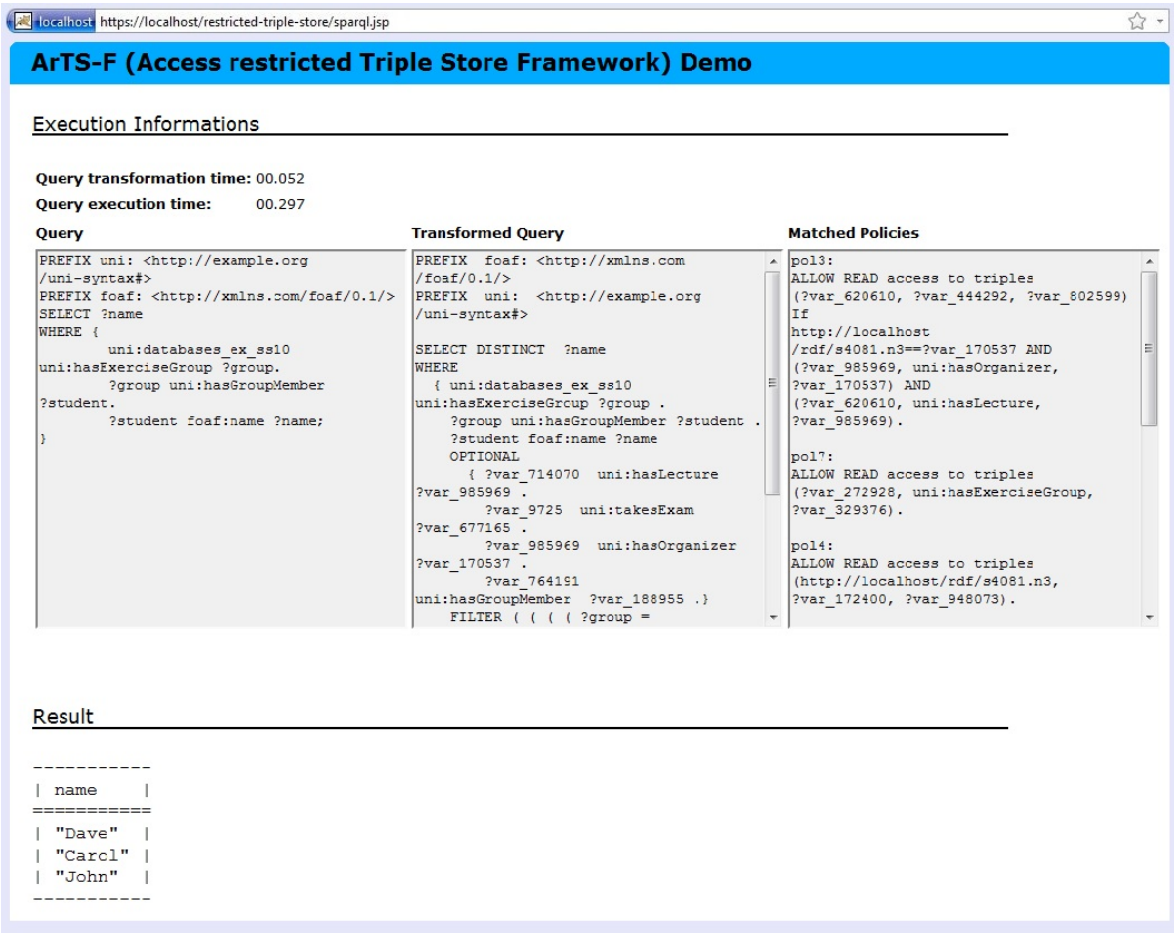


Abbildung 6.11: Informationsausgabe nach der Ausführung einer SPARQL-Anfrage

7 Fazit und Ausblicke

Das Ziel dieser Arbeit war es, ein transparentes und generisches Framework zu entwickeln, mit dessen Hilfe sich feingranular der Zugriff auf einen bereits existierenden Triple Store regeln lässt. Das ArTS-F erfüllt die erhobenen Anforderungen. Durch die konfigurierbare Anbindung über HTTP(S) wurde ein generisches Framework entwickelt, welches unabhängig von der Implementierung des zu schützenden Triple Stores ist. Zusätzlich wird ein standardisierter SPARQL HTTP-Endpoint angeboten, wodurch ein transparentes Verhalten des ArTS-F gewährleistet wird. Durch die Unterstützung von SPARQL 1.1 Update sind neben den üblichen Lese-Anfragen auch Update-Anfragen möglich. Als Bedingung für die Verwendung des ArTS-F ist zu nennen, dass auch der zu schützende Triple Store SPARQL 1.1 unterstützen muss.
Im Rahmen dieser Arbeit wurde dargelegt, dass einfache Anwendungsfälle mit der aktuellen Version des ArTS-F realisierbar sind. Mit steigender Komplexität erhöht sich jedoch die Wahrscheinlichkeit, dass das Ergebnis unkorrekt wird, wie beispielsweise in Abschnitt 6.2.2 mittes Anforderungen 3 und 4 gezeigt wurde. Diese Tatsache weist auf das Potenzial zur Erweiterbarkeit des ArTS-F hin. Anbieten würde sich in diesem Fall eine Erweiterung um einen Teil der Funktionalität des View-Mechanismus, damit weitere Anwendungsfälle abgedeckt werden können.
Ganz außer Acht wurde bisher das Prioritätsproblem der Zugriffsregeln gelassen. Als Beispiel dient folgendes Szenario: Einer größeren Nutzergruppe wird der Zugriff auf ein bestimmtes Triple verboten. Innerhalb der Gruppe existiert jedoch ein einziger Nutzer, welchem der Zugriff auf dieses Triple erlaubt werden soll. Mittels der derzeitigen Version des ArTS-F lässt sich eine solche Problemstellung nicht lösen, da die Priorität des Ablehnens (deny) höher als die des Erlaubens (allow) ist. Wie das Beispiel jedoch zeigt, ist es manchmal wünschenswert, diese Hierarchie für spezielle Zugriffsregeln zu modifzieren. Auch dadurch bieten sich neue Möglichkeiten zur Weiterentwicklung an.
Ähnlich wie das Framework ist auch die entwickelte Regelsprache als Prototy anzusehen. Denkbare Verbesserungen sind beispielweise die Verknüpfung der Bedingungen innerhalb des Körpers einer Regel zusätzlich mit Hilfe des OR-Operators oder die Angabe von mehreren zu schützenden Triples im Kopf einer Regel.
Zusammenfassend ist im Verlauf dieser Arbeit eine erste Version für ein Framework entstanden, mit welchem sich der Zugriff auf einen Triple Store für einfache Anwendungsfälle regeln lässt. Somit wurden die geforderten Ziele erreicht. Durch die Unterstützung von Update-Anfragen ist dieses Framework im Aspekt der Aktualität gegenüber den in anderen Arbeiten vorgestellten Ansätzen überlegen. Die gezeigten Problemstellungen verdeutlichen, dass das ArTS-F die Grundlagen der Zugriffsregelung auf Triple Stores abdeckt, jedoch ein breites Spektrum an Möglichkeiten zur Erweiterung besteht.

8 Abkürzungsverzeichnis

ACL Access Control List 4
ArTS-F Access restricted Triple Store Framework 11
DL Description Logics 4
EBNF „Erweiterte Backus-Naur-Form“ 24
OLS Oracle Label Security 3
PeLDS Policy enabled Linked Data Server
RDF Resource Description Framework 1
RelBAC Relation Based Access Control 4
SSL Secure Socket Layer 4
SWRL Semantic Web Rule Language 4
URI Uniform Resource Identifier 4
URL Uniform Resource Locator 46
VPD Virtual Private Database 3
WWW World Wide Web 6
W3C World Wide Web Consortium 7

Literaturverzeichnis

[ADCH+07] F. Abel, J.L. De Coi, N. Henze, A.W. Koesling, D. Krause, and D. Olmedilla. Enabling advanced and context-dependent access control in RDF stores. In *Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference*, pages 1–14. Springer-Verlag, 2007.

[BDF+06] PA Bonatti, C. Duma, N. Fuchs, W. Nejdl, D. Olmedilla, J. Peer, and N. Shahmehri. Semantic web policies–a discussion of requirements and research issues. *The Semantic Web: Research and Applications*, pages 712–724, 2006.

[Bec06] Prof. Dr. Bernhard Beckert. Unifikation. `http://www.uni-koblenz.de/~beckert/Lehre/Logik/10PraedikatenlogikSubstitutionenUnifikation_Teil2.pdf`, Juni 2006. Abgerufen am 10.09.2010.

[BKP09] F. Baader, M. Knechtel, and R. Peñaloza. A generic approach for large-scale ontological reasoning in the presence of access restrictions to the ontology's axioms. *The Semantic Web-ISWC 2009*, pages 49–64, 2009.

[BL09] Tim Berners-Lee. Linked data. `http://www.w3.org/DesignIssues/LinkedData.html`, Juni 2009.

[HPBL09] J. Hollenbach, J. Presbrey, and T. Berners-Lee. Using RDF metadata to enable access control on the social semantic web. In *Proceedings of the Workshop on Collaborative Construction, Management and Linking of Structured Knowledge (CK2009)*, 2009.

[jen10] Jena - A Semantic Web Framework for Java. `http://jena.sourceforge.net/index.html`, 2010. Abgerufen am 21.08.2010.

[KFJ03] Lalana Kagal, Tim Finin, and Anupam Joshi. A Policy Language for a Pervasive Computing Environment. *Policies for Distributed Systems and Networks, on IEEE International Workshop*, 2003.

[LC08] J. Li and W.K. Cheung. Query Rewriting for Access Control on Semantic Web. In *Proceedings of the 5th VLDB workshop on Secure Data Management*, pages 151–168. Springer-Verlag, 2008.

[Mü10] Hannes Mühleisen. Zugriffsrichtlinien und Authentifizierung für Linked-Data-Systeme. Master's thesis, Humboldt-Universität zu Berlin, http://muehleisen.org/da-hm-ld.pdf, 2010.

[mul10] Mulgara. http://mulgara.org/, Februar 2010. Abgerufen am 23.08.2010.

[OCBCM10] S. Oulmakhzoune, N. Cuppens-Boulahia, F. Cuppens, and S. Morucci. fQuery: SPARQL Query Rewriting to Enforce Data Confidentiality. *Data and Applications Security and Privacy XXIV*, pages 146–161, 2010.

[ora10] Fine-Grained Access Control for RDF Data. http://download.oracle.com/docs/cd/E11882_01/appdev.112/e11828/fine_grained_acc.htm, 2010. Abgerufen am 04.09.2010.

[Pol07] A. Polleres. From SPARQL to rules (and back). In *Proceedings of the 16th international conference on World Wide Web*, pages 787–796. ACM, 2007.

[QZW+85] L. Qiu, Y. Zhang, F. Wang, M. Kyung, and H.R. Mahajan. Trusted computer system evaluation criteria. *National Computer Security Center*, 1985.

[ses10] openRDF.org. http://www.openrdf.org/, Juli 2010. Abgerufen am 23.08.2010.

[SHJJ09] H. Story, B. Harbulot, I. Jacobi, and M. Jones. FOAF+ SSL: RESTful authentication for the social web. In *Proceedings of the First Workshop on Trust and Privacy on the Social and Semantic Web (SPOT2009)*. Citeseer, 2009.

[SPA08] SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/, Januar 2008. Abgerufen am 23.08.2010.

[SPA10] SPARQL 1.1 Update. http://www.w3.org/TR/sparql11-update/, Oktober 2010. Abgerufen am 20.10.2010.

[tri07] RDF Triple Stores. http://www.sembase.at/index.php/RDF_Triple_Stores, April 2007. Abgerufen am 23.08.2010.

[TSGW09] A. Tootoonchian, S. Saroiu, Y. Ganjali, and A. Wolman. Lockr: better privacy for social networks. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 169–180. ACM, 2009.

[vir10] Virtuoso universal server. http://virtuoso.openlinksw.com/, 2010. Abgerufen am 23.08.2010.

[W3C08] W3C. SPARQL Protocol for RDF. http://www.w3.org/TR/rdf-sparql-protocol/, Januar 2008. Abgerufen am 10.09.2010.

[W3S10] Semantic web. http://www.w3.org/standards/semanticweb, Februar 2010. Abgerufen am 01.04.2010.

[ZAGC09] R. Zhang, A. Artale, F. Giunchiglia, and B. Crispo. Using Description Logics in Relation Based Access Control. Technical report, Citeseer, 2009.