Humboldt-Universität zu Berlin

Institut für Informatik

Zugriffsrichtlinien und Authentifizierung für Linked-Data-Systeme

Diplomarbeit von Hannes Fabian Mühleisen

13. März 2010

1. Gutachter: Prof. Johann-Christoph Freytag, Ph.D.

2. Gutachter: Prof. Dr. Jens-Peter Redlich

Betreuer: Dipl. Inf. Martin Kost

Kontakt: hannes@muehleisen.org

Inhaltsverzeichnis

Kapitel 1

Einleitung

Das Semantic Web ist eine neue Generation des World Wide Web, das es seinen Nutzern erlaubt, jegliche Inhalte über die Grenzen von Anwendungen oder Webseiten hinaus auszutauschen. Zu diesem Zweck werden die Ressourcen des WWW mit standardisierten Metadaten versehen, die deren Semantik maschinenlesbar machen. Falls die Information „Horst Stein ist eine Person aus Berlin; Berlin liegt in Deutschland“ beispielsweise von einem Computer interpretiert werden kann, kann dieser auf die Frage nach Personen aus Deutschland mit „Horst Stein“ antworten. Wird diese Information veröffentlicht, können auch andere Nutzer die Antwort auf diese Frage ermitteln und mit ihren Informationen kombinieren. Um den Austausch und die Integration von Daten zu vereinfachen, beschreiben die Prinzipien von Linked Data eine Reihe von Konventionen, wie Informationen im Semantic Web veröffentlicht werden sollten [BL06]. Bei Linked Data ist es nicht vorgesehen, den Zugriff auf die veröffentlichten Informationen zu beschränken. Damit ist beispielsweise die Veröffentlichung persönlicher Daten mittels Linked Data derzeit noch nicht möglich.

Bisherige Systeme zur kontrollierten Veröffentlichung schützenswerter Informationen erfordern die Ablage dieser Daten auf Systemen der jeweiligen Betreiber. Für eine wirksame Kontrolle des Datenzugriffes sind eine geschützte Datenablage, entsprechende Zugriffsrichtlinien und die Authentifizierung von Nutzern notwendig. Als Beispiel für Systeme, die solche Elemente enthalten, können soziale Netzwerke wie Facebook oder XING genannt werden: Alle Nutzer melden sich auf einer zentralen Webseite an und hinterlegen dort Informationen. Meist können auf der Seite Einstellungen vorgenommen werden, wer welche Informationen sehen darf, beispielsweise nur „Freunde“. Trotzdem hat eine dritte Partei immer Zugriff auf alle Daten, nämlich der Betreiber eines solchen Systems. Dies stellt einen unbefriedigenden Zustand dar, da das Verhalten des Betreibers nicht vorhersehbar ist. Gelingt es einem Nutzer, sich unter Umgehung der Richtlinien Zugriff auf Daten zu verschaffen, ist die Privatsphäre aller Nutzer des Systems gefährdet.

Alternativ ist ein System denkbar, bei dem die Nutzer ihre Daten an einer von ihnen kontrollierten Stelle ablegen, zum Beispiel auf ihrem Bürocomputer oder einem privaten Server. Damit die – durchaus gewollte – Vernetzung mit Nutzern anderer Systeme zustande kommen kann und gleichzeitig die hinterlegten Informationen nicht für jeden beliebigen Internet-Teilnehmer sichtbar sind, kann eine Erweiterung von Technologien des Semantic

Web eingesetzt werden, die in der vorliegenden Arbeit beschrieben wird. Diese Erweiterung ermöglicht es ihren Nutzern, genau festlegen zu können, wer die von ihnen veröffentlichten Informationen abrufen kann – und wer nicht.

Die Inhalte des Semantic Web können im Format RDF dargestellt werden. Die bisherige Forschung im Bereich der Zugriffsrichtlinien auf Semantic-Web-Daten im RDF-Format wird in einem Artikel von Duma, Herzog und Shahmehri umfassend dargestellt [DHS07]. Reddivari, Finin und Joshi beschreiben in [RFJ07] eine Sprache für Zugriffsrichtlinien für RDF-Daten sowie die Implementierung eines Systems, dass diese Sprache auswertet. Auffallend hierbei ist jedoch, dass die Authentifizierung der anfragenden Nutzer nicht thematisiert wird. Durch die definierte Struktur der mittels RDF dargestellten Daten und anhand diese Daten beschreibenden Vokabularen und Ontologien können zudem oft neue Daten geschlussfolgert werden. Auch diese Möglichkeit wird in der verfügbaren Literatur im Bezug auf Zugriffskontrolle nicht ausreichend behandelt. Zudem werden die durch die Verwendung von Linked Data notwendigen Maßnahmen nicht thematisiert.

In der vorliegenden Arbeit wird die Authentifizierung der Nutzer mit Hilfe des existierenden Ansatzes unter der Verwendung von Sicherheitszertifikaten umgesetzt [SHJJ09]. Auf Basis der in [HPSB$^+$05] beschriebenen Semantic Web Rule Language (SWRL) wird eine Sprache zur Definition von Zugriffsrichtlinien entworfen, die eine feingranulare Beschreibung von Berechtigungen auch für abgeleitete Daten erlaubt. Das in der Arbeit beschriebene Konzept wird in einer prototypischen Implementierung einer RDF-Datenbank umgesetzt und mit Hilfe einer Beispielanwendung untersucht. Auf Basis einer RDF-Datenbank, die eine wirksame Zugriffskontrolle bietet, können Anwendungen entwickelt werden, die sich einerseits in das Linked-Data-Umfeld einfügen und andererseits dem Nutzer die Möglichkeit geben, Kontrolle über die Veröffentlichung ihrer Informationen auszuüben.

In Kapitel 2 werden die Begriffe Sicherheit und Privatsphäre in überprüfbare Kriterien gefasst, um ein System zur Verwaltung geschützter Daten bewerten zu können. Anschließend werden die Eigenschaften des Linked-Data-Konzepts sowie die ihm zugrunde liegenden Technologien des Semantic Web eingeführt: Das Datenmodell RDF, die Verwendung von Ontologien und Vokabularen, sowie passende Zugriffsmöglichkeiten werden beschrieben. Die Aspekte von der Vergabe von Zugriffsrichtlinien wie Arten von Richtlinien, Möglichkeiten zur Abgrenzung von Datenobjekten sowie die Definition von Zugriffsrichtlinien werden erläutert. Darauf folgend wird die Sicherung der Datenübertragung mittels SSL und die Möglichkeiten zur passwortlosen Authentifizierung beschrieben. Diese Konzepte und Technologien machen den Entwurf eines Systems zur Definition und Durchsetzung von Zugriffsrichtlinien im Kontext von Linked Data möglich.

Kapitel 3 beschreibt eine Anwendung zur Verwaltung von Kontakten in einem Adressbuch als Teil eines Beispielszenarios. In diesem Adressbuch werden die Daten dezentral auf Systemen unter der Kontrolle der Nutzer abgelegt. Trotzdem wird es den Nutzern ermöglicht, ausgewählten Kontakten Zugriffsrechte auf einzelne Datenelemente zu geben. Aus diesem Szenario werden Anforderungen an ein generisches Softwaresystem zur Veröffentlichung und zum Abruf von Daten unter dem Schutz von Zugriffsrichtlinien, verschlüsselter Datenübertragung sowie passwortloser Authentifizierung abstrahiert.

Die mit Hilfe des Beispielszenarios identifizierten Anforderungen werden in Kapitel 4 in Form von verschiedenen Spezifikationen konkretisiert. Unter dem Namen Policy enabled

Linked Data Server (PeLDS) wird die für die Umsetzung der Anforderungen erforderliche Funktionalität und das neue Format für Zugriffsrichtlinien spezifiziert. Die neue Komponente PeLDS ermöglicht die Verwaltung von Zugriffsrichtlinien sowie die Veröffentlichung und den Abruf von Daten. Um Implementierungen zu erleichtern, wird das Konzept in Funktionsgruppen in Form von Komponenten aufgeteilt. Die Aufteilung geschieht nach Behandlung von Anfragen, der Durchführung von Authentifizierung, der Regelauswertung sowie der Datenspeicherung.

Im Rahmen dieser Arbeit ist eine Implementierung des beschriebenen Konzepts entstanden. Diese Implementierung wird in Kapitel 5 näher erläutert und daraufhin Konzept und Implementierung einer Evaluation unterzogen. Diese Evaluation umfasst die Umsetzung der in der Analyse erarbeiteten Anforderungen, die Ausdrucksmächtigkeit der Sprache zur Formulierung von Zugriffsrichtlinien, die Methoden zum Schutz der Privatsphäre der Nutzer, die Sicherheit des Systems sowie die Integration in das Linked-Data-Konzept. Anschließend werden die Leistungsdaten des Prototypen erfasst und mit der bestehenden Lösung Joseki verglichen, um die Auswirkung der Verarbeitung der Zugriffsrichtlinien auf die Leistung des Systems einzuschätzen. Das in der Analysephase erarbeitete Beispielszenario wurde ebenfalls implementiert, um die Funktionen von PeLDS zu demonstrieren. Diese Implementierung eines verteilten Adressbuchs wird beschrieben und zusätzlich entsprechend der genannten Kriterien evaluiert.

In Kapitel 6 werden die Ergebnisse dieser Arbeit zusammenfassend dargestellt. Mögliche Erweiterungen des PeLDS-Konzepts sowie Optimierungsmöglichkeiten für Implementierungen von PeLDS werden genannt. Die neuen Möglichkeiten für den Entwurf und die Entwicklung von Anwendungen, die auf der Funktionalität von PeLDS aufbauen, werden beschrieben.

Kapitel 2

Grundlagen

Unterstützt ein Computersystem Zugriffsrichtlinien für die darin veröffentlichten Daten, bietet es sich für die Ablage sensibler oder zu schützender Informationen an. Um den Zugriff auf diese Informationen zuverlässig zu kontrollieren, kommt der Sicherheit des Computersystems zentrale Bedeutung zu. Die veröffentlichten Informationen können zudem die Privatsphäre der Nutzer gefährden, sollte die Durchsetzung der Zugriffsrichtlinien fehlschlagen. Aus diesen Gründen werden in Abschnitt 2.1 Anforderungen an sichere Computersysteme sowie Anforderungen für den Schutz der Privatsphäre der Nutzer in einem Computersystem eingeführt.

Für die Arbeit relevante Technologien und Begriffe des Semantic Web werden in Abschnitt 2.2 erläutert: Aufbauend auf dem Linked-Data-Prinzip wird die Datendarstellung im Resource Description Framework (RDF), die Erstellung und Verwendung von Ontologien mittels OWL sowie der Zugriff auf diese Daten mittels SPARQL erklärt. RDF definiert ein abstraktes Datenmodell sowie dessen Darstellung in Textform. Dieses Datenmodell benutzt Tripel zur Formalisierung von Aussagen. Diese Aussagen bestehen aus Subjekt, Prädikat und Objekt – vergleichbar mit einfachen Sätzen in menschlicher Sprache. Eine Menge von Tripeln wird in einem RDF-Graphen zusammengefasst. Die verwendbaren Prädikate können von Vokabularen festgelegt werden, soll zusätzlich zu einer syntaktischen Auswertung des RDF-Graphen auch eine semantische Interpretation der abgelegten Inhalte durch Computersysteme möglich sein. Für den Zugriff auf in RDF-Graphen gespeicherte Daten steht die an SQL angelehnte Anfragesprache SPARQL zur Verfügung.

RDF-Graphen können mit Hilfe von standardisierten Austauschformaten und Protokollen veröffentlicht werden. Um jedoch die Auslieferung der RDF-Graphen und dessen Elementen an anfragende Nutzer kontrollieren zu können, werden mit Abschnitt 2.3 verschiedene Ansätze zur Darstellung von Zugriffsrichtlinien untersucht. Diese Richtlinien werden vom Nutzer formuliert, durch sie hat der Nutzer die Kontrolle über den Zugriff auf die von ihm veröffentlichten Daten. Eine Zugriffsrichtlinie kann beispielsweise sein, nur Familienmitgliedern den Zugriff auf persönliche Daten zu gewähren. Für die allgemeine Formulierung von Richtlinien existieren verschiedene Ansätze, für den Bereich des Semantic Web wurde die Sprache SWRL entwickelt. Sie kann verwendet werden, um eine Zugriffskontrolle zu realisieren.

Damit die Auslieferung von RDF-Graphen sinnvoll kontrolliert werden kann, muss der Nutzer eindeutig identifiziert werden, der diese Daten anfordert. Zusätzlich muss sichergestellt werden, dass auf die ausgelieferten Daten während ihrer Übertragung nicht unberechtigt zugegriffen wird. Abschnitt 2.4 stellt HTTPS als bestehendes System zur Absicherung von Kommunikation vor. Zusätzlich wird mit FOAF+SSL (Friend of a Friend + Secure Socket Layer) auf ein existierendes Protokoll zur Authentifizierung von Kommunikationspartnern ohne vorhergehende Anmeldung eingegangen. Hierzu werden mit Linked Data veröffentlichte Informationen sowie kryptographische Verfahren benutzt.

2.1 Sicherheit und Privatsphäre

Wenn ein Computersystem entwickelt werden soll, das es seinen Nutzern ermöglicht, selbst zu entscheiden, wem Informationen über sich bekannt gegeben werden sollen, muss besonderes Augenmerk auf die Sicherheit des Computersystems gelegt werden. Die veröffentlichten Daten können die Privatsphäre der Nutzer beeinträchtigen, sollten diese Dritten gegen den Willen des Nutzers zugänglich gemacht werden. Um entsprechende Vorgaben für den Entwurf sowie nachprüfbare Kriterien für die Evaluation solchen Systems zu identifizieren, werden allgemeine Anforderungen an sichere Computersysteme in Unterabschnitt 2.1.1 aufgeführt. Darauf folgen in Unterabschnitt 2.1.2 wichtige Forderungen an Computersysteme, die die Privatsphäre der Nutzer schützen sollen sowie eine Adaption dieser Forderungen auf das Semantic-Web-Umfeld. Der Begriff *Nutzer* bezeichnet hier Personen, die sich eines Computersystems bedienen, er wird in Unterabschnitt 2.2.5 genauer beschrieben.

2.1.1 Sicherheit

Um den Begriff eines sicheren Computersystems im Sinne der Arbeit festzulegen, wird dieser in Definition 2.1 erläutert. Daran anschließend werden sechs in der Literatur identifizierte Forderungen genannt, die für die Bewertung von Systemen im Bezug auf Sicherheit verwendet werden können. Darin werden Begriffe wie System und Daten nur abstrakt verwendet, um diese Forderungen möglichst universell anwendbar zu machen.

Definition 2.1 *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 [Lat85].*

Die Anforderungen an ein sicheres Computersystem lauten nach [Lat85] wie folgt:

(a) Richtlinien - Eine explizite und formal definierte Zugriffsrichtlinie (siehe Definition 2.6) wird vom Computersystem durchgesetzt. Für jegliche eindeutig identifizierte Nutzer und abgrenzbare Mengen von Daten existieren Regeln, aus denen das Computersystem ermitteln kann, ob eine Aktion zulässig ist.

(b) Klassifizierung - Es ist möglich, jede abgrenzbare Menge von Daten zuverlässig hinsichtlich ihres Sicherheitsstatus’ zu klassifizieren.

(c) Authentifizierung - Die Nutzer des Computersystems werden identifiziert. Jeder Zugriff auf Information wird hinsichtlich der Rechte der Nutzer geprüft. Die Information, die zur Identifizierung notwendig ist, wird sicher verwaltet und von jeder Systemkomponente, die sicherheitsrelevante Aktionen durchführt, geprüft.

(d) Verantwortlichkeit - Status- und Zugriffsinformationen werden gespeichert, um jede sicherheitsrelevante Aktion dem verantwortlichen Nutzer zuordnen zu können. Es ist für den Betreiber des Systems zu jedem Zeitpunkt nachvollziehbar, welcher Nutzer wann welchen Zugriff durchgeführt hat.

(e) Garantie - Das Computersystem enthält einen Mechanismus, mit dem die korrekte Umsetzung der bisher genannten Punkte geprüft wird. Das Ergebnis dieser Prüfung ist vom Betreiber des Systems einsehbar. Bei einem positiven Ergebnis dieser Prüfung wird davon ausgegangen, dass das System die Anforderungen an ein sicheres Computersystem erfüllt.

(f) Fortlaufender Schutz - Der genannte Mechanismus zur Prüfung der Anforderungen ist gegen Manipulationen geschützt.

Ein Zugriffskontrollsystem - wie in dieser Arbeit vorgestellt - ordnet sich in den Bereich der in Definition 2.1 genannten „speziellen Funktionen“ ein: Die darin enthaltene eindeutige Identifizierung der Benutzer und die Durchsetzung von Zugriffsrichtlinien werden verwendet, um die Forderungen nach Durchsetzung von *Richtlinien* und *Authentifizierung* der Nutzer umzusetzen.

2.1.2 Privatsphäre

Als verhältnismäßig neues Recht wurde die Privatsphäre (engl. privacy) erstmals 1890 in einem juristischen Kontext diskutiert [WB90]. Die Auffassung der Autoren Warren und Brandeis lässt sich auf die Forderung nach einem „Recht, alleine gelassen zu werden“ zusammenfassen. Die Notwendigkeit für ein solches eigenes Recht leiteten sie aus den Fortschritten der Vervielfältigungstechnologie ab: Zeitungs- und Fotodruck hatten dazu geführt, dass private Texte, Zeichnungen und Fotografien recht ungehemmt veröffentlicht worden waren. Als Beispiel geben sie den Tagebucheintrag an, in dem ein Mann zugibt, dass er seine Frau betrogen hat. Ohne ein Recht auf Privatsphäre könnte dieser Text beliebig veröffentlicht werden. Wichtig ist außerdem, dass in dem Moment, in dem eine Person freiwillig Informationen über sich selbst veröffentlicht, die Privatsphäre nicht mehr geschützt ist.

Eine für den Bereich der Datenverarbeitung passende Definition stammt von 1967, Westin definiert Privatsphäre wie in Definition 2.2 dargelegt. Hier ist zu bemerken, das der Begriff der „Information“ sehr weit gefasst ist und etwa genauso Fotografien als auch den Aufenthaltsort umfassen kann.

Definition 2.2 *Privatsphäre ist das Recht jedes Einzelnen, für sich selbst zu bestimmen, wann, wie und in welchem Ausmaß Informationen über sich Dritten zugänglich gemacht werden [Wes67].*

Das Verständnis von Privatsphäre hat sich vom einfachen „Recht, alleine gelassen zu werden“ zum heutigen „Recht auf informationelle Selbstbestimmung“ weiterentwickelt. Es ist mittlerweile nicht mehr zulässig, Daten einer Person selbst ohne Veröffentlichung zu verwenden, ohne dass diese Person dem zugestimmt hat. Natürlich ist das hier oft verwendete Beispiel mit der Telefonnummer oder dem aktuellen Aufenthaltsort im Bezug auf die möglichen Probleme bei versehentlicher Veröffentlichung etwas marginal. Trotzdem können in manchen Fällen aus verschiedenen an sich nicht kritischen Informationen sensible Informationen erzeugt werden, wie in einem Beispiel in [Fre09] beschrieben wird. Ein Beispiel in [DHS07] zeigt, welchen Stellenwert der Schutz der Privatsphäre haben sollte, da die Weitergabe persönlicher Daten – wie zum Beispiel eine Erkrankung mit HIV – weitreichende soziale Folgen haben kann.

Für den Bereich der Datenverarbeitung in Computersystemen wurden in [AKSX02] unter Anderem auf der Basis rechtlicher Einschränkungen zehn Prinzipien für ein Computersystem, das die Privatsphäre der Nutzer schützt, aufgestellt:

(a) Verwendungszweck - Für persönliche Informationen, die in einem Computersystem abgelegt werden, soll der Zweck, zu dem diese Information gespeichert wurden, mit diesen Informationen verknüpft sein.

(b) Zustimmung - Der Verwendungszweck soll die Zustimmung der Person haben, die die persönlichen Informationen abgegeben hat.

(c) Beschränkte Datenmenge - Die Menge der abgelegten persönlichen Informationen soll auf ein für den Zweck notwendiges Minimum beschränkt werden.

(d) Beschränkte Nutzung - Die Nutzung der Daten soll sich auf die festgelegten Verwendungszwecke beschränken.

(e) Beschränkte Veröffentlichung - Die persönlichen Informationen sollen ausschließlich im Rahmen des Verwendungszwecks an Dritte weitergegeben werden.

(f) Beschränkte Speicherdauer - Persönliche Informationen sollen nur so lange gespeichert werden, wie es für den Verwendungszweck notwendig ist.

(g) Richtigkeit - Die persönlichen Informationen sollen zutreffend und aktuell sein.

(h) Sicherheit - Das Computersystem, dass die persönlichen Daten enthält, soll gegen Datendiebstahl und Datenmissbrauch abgesichert sein.

(i) Offenheit - Die Personen, deren persönliche Daten abgelegt sind, sollen in der Lage sein, alle abgelegten Informationen über sich selbst abzurufen.

(j) Einhaltung der Prinzipien - Die Personen, deren persönliche Daten abgelegt sind, sollen die Einhaltung der genannten Prinzipien überprüfen können.

Privatsphäre und Semantic Web

Wenn man ein Computersystem im Umfeld des Semantic Web betrachtet, in dem die Nutzer persönliche Informationen, selbst ablegen, sollten nach [DHS07] eine Reihe von Forderungen an diese Systeme gestellt werden. Diese Forderungen enthalten die beschriebenen Anforderungen an ein sicheres Computersystem, hierdurch wird der Punkt *Sicherheit* umgesetzt [MBK08]. Die genannten Prinzipien für Systeme, die die Privatsphäre ihrer Nutzer schützen, kommen nur in Teilen zur Anwendung, wenn der Nutzer selbst seine Daten dort ablegen und verwalten kann: Es kann von seiner Zustimmung ausgegangen werden, er selbst beschränkt die abgelegte Datenmenge und deren Speicherdauer. Die Richtigkeit der Daten kann angenommen werden, ebenfalls die Offenheit. Die Punkte *Sicherheit* und *Einhaltung der Prinzipien* wurden im vorhergehenden Kapitel beschrieben.

Aus denen in [DHS07] und [AKSX02] wurde daher die folgende Liste zusammengestellt, die die Anforderungen an ein System im Umfeld des Semantic Web in Bezug auf Schutz der Privatsphäre der Nutzer beschreibt:

(a) Klassifizierung - Nutzer müssen in der Lage sein, ihre Informationen bei deren Ablage in einem Computersystem nach ihrer Relevanz bezüglich des Schutzes ihrer Privatsphäre zu klassifizieren. In Unterabschnitt 2.3.2 wird dargestellt, wie dies mit RDF-Daten möglich ist.

(b) Zugriffskontrolle - Die Nutzer kontrollieren den Zugriff auf von ihnen im Computersystem gespeicherten Daten. Hierzu definieren sie, welchem Nutzer welche Daten zur Verfügung gestellt werden sollen. Dies ermöglicht den Schutz der Informationen, die der Nutzer über sich selbst veröffentlicht und trägt damit zum Schutz seiner Privatsphäre bei. Diese Definition kann mit Hilfe von Datenklassifizierung und entsprechenden Zugriffsrichtlinien erfolgen. Hierdurch wird die *Beschränkte Veröffentlichung* erreicht.

(c) Richtlinienkontrolle - Der Nutzer hat die Kontrolle über die für seine Daten gültigen Zugriffsrichtlinien (siehe Definition 2.6). Der Abruf der Zugriffsrichtlinien ist nicht möglich, da durch die Analyse der Richtlinien Informationen ungewollt übermittelt werden können. Um im genannten Beispiel zu bleiben, wenn eine Richtlinie das Ausliefern des HIV-Status verhindern soll und diese zugänglich ist, kann vermutet werden, dass genau diese Information existiert.

(d) Verwendung von Daten durch Dritte - Schließlich soll der Nutzer festlegen können, in welcher Art und Weise seine Informationen von Dritten verwendet werden können. Dies stellt sich üblicherweise als sehr schwierig dar, da die Kontrolle über Systeme von Dritten im Allgemeinen unmöglich ist. Solche Vorhaben können nur mit speziellen technischen und organisatorischen Methoden wie beispielsweise Verschlüsselungshardware und Zertifizierungen von Systemen durchgeführt werden.

Diese in diesem Abschnitt genannten Punkte werden verwendet, um in Analyse und Entwurf eine für den Schutz der Privatsphäre der Nutzer befriedigende Lösung zu finden.

2.2 Semantic Web

Im „herkömmlichen“ World Wide Web (WWW) werden Dokumente zwischen verschiedenen am Internet angeschlossenen Computern ausgetauscht und referenziert. Über den Inhalt dieser Dokumente ist bekannt, dass sie üblicherweise im Format der Hypertext Markup Language (HTML) erstellt werden und außer Text noch eine Vielzahl anderer Inhaltselemente wie zum Beispiel audiovisuelle Medien enthalten können. Der Text, der für einen Computer leicht aus diesem Dokument extrahierbar ist, ist von Formatierungsbefehlen abgesehen unstrukturiert. Daher ist es sehr schwierig, aus diesem unstrukturierten Text automatisiert eine Semantik zu schließen.

Definition 2.3 *Das Semantic Web ist eine Erweiterung des WWW durch Konzepte, die eine inhaltliche Interpretation der verfügbaren Informationen zulassen [BLHL01].*

Hier setzt das in Definition 2.3 von Berners-Lee, Hendler und Lassila so abgegrenzte *Semantic Web* an [BLHL01]. Unter Verwendung von Standards und Technologien des WWW werden nun keine unstrukturierten Dokumente mehr ausgetauscht, sondern strukturierte Daten. Diese Daten sind mit einer ebenfalls maschinenlesbaren Semantik versehen, die durch ein so genanntes Vokabular definiert wird. Mit Hilfe dieser Metadaten werden die veröffentlichten Daten interpretierbar. Somit ist es für ein Programm möglich, aus der definierten Logik Schlussfolgerungen zu ziehen und diese Daten mit anderen Daten in Beziehung zu setzen.

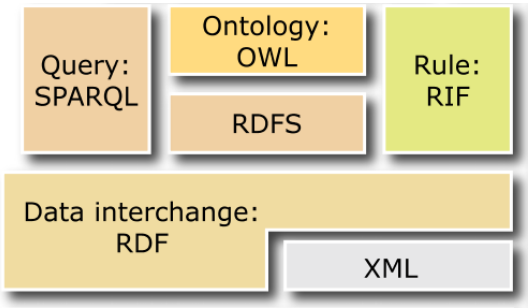

Abbildung 2.1: Semantic Web: Technologie-Stack [Wor09]

In Abbildung 2.1 wird ein hierarchischer Aufbau der für das Semantic Web verwendeten Technologien gegeben: Das Semantic Web basiert sowohl auf der Fähigkeit von XML, beliebige Schemata zur Datenstrukturierung anzulegen, als auch auf dem flexiblen Datenmodell *Resource Description Framework (RDF)* (siehe Unterabschnitt 2.2.1) für den Datenaustausch. Auf der nächsten Ebene ist über RDF eine Ontologiesprache definiert, um das in Dokumenten verwendete Vokabular zu beschreiben. Wenn Computersysteme in der Lage sein sollen, die in RDF dargestellten Informationen zu interpretieren und logische Schlüsse daraus abzuleiten, sind komplexere Ontologien notwendig, als mit den begrenzten RDF-Schemata (RDFS) möglich wären. OWL in seinen Varianten besitzt die notwendige Ausdrucksmächtigkeit und logische Grundlage [MH04]. Anfragen an RDF-Daten können

mit der Sprache SPARQL durchgeführt werden. Das Rule Interchange Format (RIF) beschreibt eine Sprache zur Formulierung für Regeln, die zwischen Computerprogrammen ausgetauscht werden, die allerdings noch nicht als Standard verabschiedet ist.

Das Datenmodell RDF für die Darstellung von Daten wird im Folgenden in Unterabschnitt 2.2.1 eingeführt, danach beschreibt Unterabschnitt 2.2.2 eine Möglichkeit, Vokabulare für RDF-Daten in RDFS oder OWL zu definieren. Unterabschnitt 2.2.3 geht dann auf die „Linked-Data“ Erweiterung des Semantic Web ein. Zuletzt erläutert Unterabschnitt 2.2.4 mit SPARQL eine Zugriffsmöglichkeit auf RDF-Daten.

Beispiel 2.1

Angenommen ein Internet-Nutzer mit dem Namen „Stein“ hat seine Telefonnummer in einem WWW-Dokument veröffentlicht. Dieses wird von einer herkömmlichen Suchmaschine indiziert. Da diese Dokumente keine Semantik enthalten, nimmt die Suchmaschine zwei Wörter in ihren Datenbestand auf: „Stein“ und „030123456“. Sucht ein anderer Nutzer nun nach der Telefonnummer von Herrn Stein, in dem er „Stein“ in seine Suchmaschine eingibt, ist klar, dass hier kein sinnvolles Resultat erzeugt werden kann. Die Suchmaschine weiß nicht, dass „Stein“ eine Person ist, und der Nutzer hat keine Möglichkeit, speziell nach dessen Telefonnummer zu suchen. Da Herr Stein zudem das Pech hat, einen Namen zu haben, der nicht sofort erkennen lässt, dass es sich um den Nachnamen einer Person handelt, schwinden die Erfolgsaussichten auf einen Treffer weiter.

Hätte Herr Stein nun seine Telefonnummer nach den Prinzipien des Semantic Web veröffentlicht, könnte die Suchmaschine folgende Information indizieren: Eine **Person** mit dem **Nachnamen** „Stein“ hat eine **Telefonnummer**, die „030-123456“ lautet. Eine Suchanfrage nach der Telefonnummer der Person „Stein“ hätte nun sehr viel höhere Erfolgsaussichten, da ein intelligenterer Zugriff möglich ist.

2.2.1 Resource Description Framework

Um wie in Abschnitt 2.2 beschrieben maschinenlesbare Informationen im WWW veröffentlichen zu können, bedarf es eines Datenmodells. Dieses Datenmodell wird durch das Resource Description Framework (RDF) nach Definition 2.4 bereitgestellt. Wichtig ist, dass zwischen Modell und textueller Repräsentation desselben unterschieden wird. RDF beschreibt so genannte Graphen, keine Syntax, davon abgeleitet sind mögliche textuelle Repräsentationen etwa das verbreitete RDF/XML [BM04] oder – in kompakterer und lesbarerer Form – Notation 3 (N3) [BL98a].

Definition 2.4 *Das „Resource Description Framework“ ist ein universelles Datenmodell, um Information im WWW darzustellen [MMM04].*

RDF basiert auf dem Prinzip von Aussagen über Ressourcen. Ressourcen sind die kleinste Einheit in RDF, sie werden durch einen Uniform Resource Identifier (URI) repräsentiert. URIs sind eindeutige Bezeichner von im WWW verfügbaren Daten, meistens beschreiben sie zudem die Art und Weise, wie auf diese Daten zugegriffen werden kann. [MD09]. Jede Aussage über eine Ressource wird in der Form Subjekt - Prädikat - Objekt formalisiert. Diese Formalisierung wird als *Tripel* bezeichnet. Eine Menge dieser Tripel

kann in der Form eines RDF-Graphen dargestellt werden. Im RDF-Graphen sind die Subjekte und Objekte der Tripel die Knoten. Knoten mit identischem Bezeichner gelten als identisch. Die im RDF-Graphen enthaltenen Kanten werden durch die Tripel beschrieben und nach dem verwendeten Prädikat benannt. Die verwendeten Bezeichner für Ressourcen und Prädikate können beliebig gewählt werden. Soll der RDF-Graph Informationen jedoch in einer maschinenlesbaren Semantik enthalten, werden die verwendeten Prädikate einer wie in Unterabschnitt 2.2.2 beschriebenen Ontologie entnommen. Die Objekte können entweder Referenzen auf andere Ressourcen, Literale (Zeichenketten) oder anonyme Knoten enthalten. Anonyme Knoten sind Subjekte im RDF-Graph, die keine URI haben. Zwei RDF-Graphen gelten als identisch, wenn sie identische Tripel enthalten. Zwei Tripel gelten als identisch, wenn sie in Subjekt, Prädikat und Objekt übereinstimmen.

Beispiel 2.2

In unserem Beispiel nehmen wir an, dass der bereits vorgestellte Herr Stein eine Person ist, die eine Telefonnummer hat. Dies wird in Abbildung 2.2 in Form eines RDF-Graphen dargestellt. Es wird sichtbar, dass Horst Stein durch eine URI eindeutig identifiziert wird. Die Prädikate sind aus einem eigenen Vokabular (siehe Unterabschnitt 2.2.2) entnommen, sie bezeichnen hier Name und Telefonnummer. Die beiden Objekte sind Literale, also Zeichenketten. Hier sind somit zwei Tripel dargestellt:

- http://example.com/horststein http://example.com/ont#name 'Horst Stein'
- http://example.com/horststein http://example.com/ont#phone '030123456'

Dieser Graph kann nun mittels RDF/XML serialisiert werden. In Listing 2.1 ist eine mögliche Serialisierung dargestellt. Eine weitere mögliche Serialisierung mittels N3 ist in Listing 2.2 gegeben. Diese Serialisierungen werden beispielsweise verwendet, um den RDF-Graphen zwischen verschiedenen Computersystemen auszutauschen.

```
1 <rdf:Description rdf:about="http://example.com/horststein">
2   <ont:name>Horst Stein</ont:name>
3   <ont:phone>030123456</ont:phone>
4 </rdf:Description>
```

Listing 2.1: RDF/XML-Serialisierung - Ausschnitt

```
1 @prefix ont: <http://www.example.com/ont#> .
2 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3 <http://example.com/horststein> ont:name "Horst Stein" ;
4   ont:phone "030123456" .
```

Listing 2.2: RDF/N3-Serialisierung

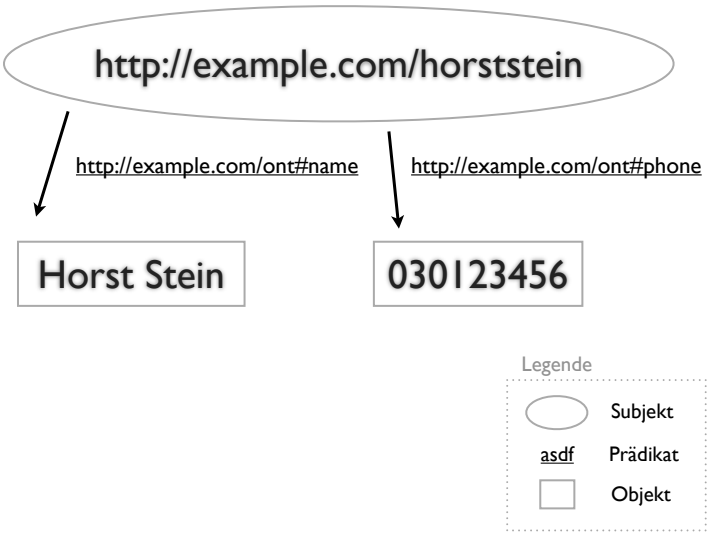

Abbildung 2.2: RDF-Graph

2.2.2 Ontologien und Vokabulare

Ontologien und Vokabulare sind ein wichtiger Bestandteil des Semantic Web, da sie einerseits Konventionen zum Aufbau von RDF-Graphen beschreiben und andererseits die Semantik beschreiben, in denen die Daten interpretiert werden. Die Verwendung von Vokabularen und Ontologien ist nicht zwingend notwendig, RDF kann auch auf rein syntaktischer Ebene interpretiert werden. Interpretationen der Daten durch Computersysteme auf semantischer Ebene sowie die Bestimmung abgeleiteter Daten machen jedoch die Verwendung von Ontologien notwendig. Häufig werden Ontologien und Vokabulare von gleichartigen Datensätzen gemeinsam verwendet. Damit können die so ausgezeichneten Daten einfach integriert oder zusammengeführt werden. Ein Vokabular (auch als Schema bezeichnet) kann mit Hilfe von RDF Schema (RDFS) [BGM04] definiert werden. Sollen komplexere Beziehungen zwischen Konzepten oder Konsistenzbedingungen ausgedrückt werden, kann die für diese Zwecke entwickelte Web Ontology Language (OWL) als Erweiterung von RDFS verwendet werden. Definition 2.5 beschreibt den Begriff *Ontologie* im Kontext dieser Arbeit. Ontologien können im Sinne der Definition ein gemeinsames Vokabular beschreiben, nach dem Daten strukturiert werden. Dies ermöglicht den Austausch von Daten zwischen den Nutzern einer Ontologie. Im Bereich des Semantic Web existieren maschinenlesbare Sprachen wie RDFS oder OWL zur Definition von Ontologien. Spezielle Softwarepakete können die in diesen Sprachen enthaltenen Definitionen auswerten.

Definition 2.5 *Eine Ontologie definiert die Begriffe, die einen Wissensbereich beschreiben [Hef04].*

RDF Schema

RDF Schema (RDFS) bietet eine Möglichkeit, ein Vokabular aus *Konzepten* und deren *Eigenschaften* für die Beschreibung von Ressourcen anzulegen. Die so definierten Konzepte und Eigenschaften können dann innerhalb von RDF-Daten verwendet werden. Damit definiert ein RDF-Vokabular ein Typsystem für RDF-Daten. RDFS selbst wird mit Hilfe von RDF dargestellt. Die Vokabulare selbst sind dadurch ebenfalls RDF-Graphen und können genau wie diese verarbeitet werden. Ein RDFS-Vokabular ist mit einem Schema für Relationale Datenbanken vergleichbar, und kann ebenfalls in einem Entity-Relationship-Diagramm dargestellt werden [BL98b].

RDFS unterscheidet zwischen Konzepten und Eigenschaften (engl. property). Mit Konzepten kann – ähnlich wie mit Klassen in der objektorientierten Programmierung – eine „ist-ein“ - Hierarchie (engl. is-a hierarchy) beschrieben werden, in die Ressourcen eingeordnet werden können, um ihren Typ festzulegen. Außerdem können in RDFS Eigenschaften für Ressourcen für die jeweiligen Konzepte definiert werden. Eigenschaften beschreiben Beziehungen zwischen Subjekten und Objekten. Die Objekte dieser Eigenschaften können zudem in ihrem Wertebereich beschränkt werden. Eine Ressource in einem RDF-Graphen wird als *Instanz* eines Konzepts bezeichnet, wenn sie anhand einer speziellen Eigenschaft zur Festlegung ihres Typs auf dieses Konzept verweist und die im Vokabular für dieses Konzept definierten Eigenschaften besitzt.

Wichtig ist allerdings, dass es jederzeit möglich ist, beliebige Instanzen mit im Vokabular nicht genannten Eigenschaften zu beschreiben. Dies muss keine Verletzung des Vokabulars bedeuten und die Instanzen sind weiterhin auf Konsistenz prüfbar. Eine solche Erweiterung ist in einem relationalen Schema nicht ohne Weiteres möglich [BL98b].

Beispiel 2.3

> Das Konzept *Mensch* wird definiert, das Konzept *Mann* leitet sich von dem Konzept *Mensch* ab. Das Konzept *Mensch* besitzt die Eigenschaften *Name* sowie *Telefonnummer*. Wenn man nun in wie in Beispiel 2.2 für die Beispielressource von Herrn Stein festlegt, dass sie den Typ *Mann* besitzt, ist klar, dass diese Instanz die genannten Eigenschaften haben kann. Angenommen, die Eigenschaft *Telefonnummer* wäre auf ganzzahlige Werte beschränkt, dann würde das RDF-Tripel „Herr Stein hat die Telefonnummer abcdef“ gegen dieses Vokabular verstoßen. Gleichzeitig kann aber aus der Aussage, dass Herr Stein eine Instanz des Konzeptes *Mann* ist geschlossen werden, dass Herr Stein ein *Mensch* ist. In Listing 2.3 wird die RDF/XML-Serialisierung des genannten Vokabulars angegeben. Dort wird die in diesem Beispiel beschriebene Struktur maschinenlesbar dargestellt.

Web Ontology Language

OWL stellt ebenfalls ein Vokabular für die Beschreibung von Konzepten und deren Eigenschaften bereit. Im Unterschied zu RDFS können diese Beschreibungen differenzierte Beziehungen zwischen Konzepten, die Kardinalitäten dieser Beziehungen, Eigenschaften und Einschränkungen von Eigenschaften sowie Aussagen über Gleichheit enthalten. Zudem basiert OWL auf einer definierten Logik auf Basis der Prädikatenlogik. OWL ist in

```
1 <rdfs:Class rdf:ID="Mann">
2   <rdfs:subClassOf rdf:resource="#Mensch"/>
3 </rdfs:Class>
4
5 <rdf:Property rdf:ID="name">
6   <rdfs:domain rdf:resource="#Mensch"/>
7   <rdfs:range rdf:resource="&xsd;string"/>
8 </rdf:Property>
9
10 <rdf:Property rdf:ID="phone">
11   <rdfs:domain rdf:resource="#Mensch"/>
12   <rdfs:range rdf:resource="&xsd;integer"/>
13 </rdf:Property>
```

Listing 2.3: RDF-Schema als RDF/XML - Ausschnitt

drei verschiedenen Varianten spezifiziert [MH04]. Jede dieser Varianten basiert auf RDFS, jedoch sind nicht immer alle möglichen RDF(S)-Konstrukte zulässig. In der folgenden Liste wird eine Übersicht über die verschiedenen Varianten gegeben:

- „OWL Lite“ besitzt eine eingeschränkte Ausdrucksmächtigkeit. Daraus ergibt sich eine geringere formale Komplexität als OWL DL, die OWL Lite entscheidbar macht. Durch diese Entscheidbarkeit ist OWL Lite effizient auswertbar.
- „OWL DL“ besitzt die volle Mächtigkeit der OWL-Sprachspezifikation. Einige Konstrukte werden jedoch durch Regeln eingeschränkt, um das Problem, ob ein Dokument einem OWL-Vokabular entspricht, für ein Programm entscheidbar zu machen.
- „OWL Full“ unterliegt keinerlei Einschränkungen. Dadurch ist es allerdings nicht möglich, die Entscheidbarkeit zu gewährleisten. Programme können Probleme haben, OWL Full-basierte Ontologien zu verarbeiten. Beispielsweise kann bei OWL Full nicht garantiert werden, dass die Auswertung einer OWL-Ontologie immer terminiert.

OWL-Anweisungen werden normalerweise von einem so genannten *Reasoner* ausgewertet. Hierbei handelt es sich um ein Programm, dass eine OWL-Ontologie auf einen RDF-Graph anwendet und das zunächst die Daten auf Konsistenz zur Ontologie überprüft und zudem aus Daten und Ontologie ableitbaren Aussagen bestimmt.

Beispiel 2.4

Wenn aufbauend auf Beispiel 2.3 sichergestellt werden soll, dass alle Instanzen des Konzepts „Mann“ für die Eigenschaft „Geschlecht“ den Wert „männlich“ besitzen, dann ist es – anders als in RDFS – mit OWL möglich, dies auszudrücken. In Listing 2.4 wird die RDF/XML-Serialisierung einer OWL-Ontologie in serialisierter Form angegeben, die die genannte Einschränkung beinhaltet: Das Konzept „Mann“ und alle als äquivalent markier-

ten Konzepte besitzen eine Einschränkung auf dem Wert der Eigenschaft „Geschlecht“. Der Wert dieser Eigenschaft muss „männlich“ lauten. Damit ist es beispielsweise auch möglich, einer Instanz des Konzepts „Mensch“, die diesen Wert in der Eigenschaft „Geschlecht“ hat, automatisch dem Konzept „Mann“ zuzuordnen. Beispielsweise wäre es nun zwar möglich, eine Instanz des Konzepts „Mann“ mit „weiblich“ als Wert für die Eigenschaft „Geschlecht“ anzulegen, deren Verifikation nach dem gegeben OWL-Schema würde jedoch einen Widerspruch erzeugen und damit fehlschlagen.

```
1 <owl:Class rdf:ID="Mensch"/>
2 <owl:Class rdf:ID="Mann"/>
3   <rdfs:subClassOf rdf:resource="#Mensch"/>
4   <owl:equivalentClass>
5     <owl:Restriction>
6       <owl:onProperty rdf:resource="#Geschlecht"/>
7       <owl:hasValue rdf:resource="#maennlich" rdf:type="#
            Geschlecht"/>
8     </owl:Restriction>
9   </owl:equivalentClass>
10 </owl:Class>
```

Listing 2.4: OWL als RDF/XML - Ausschnitt

2.2.3 Linked Data

Damit die Integration von RDF-Daten über die Grenzen von Systemen und Organisationen hinaus erfolgen kann, wurden eine Reihe von Konventionen über die verwendeten RDF-Graphen und die darin verwendeten Bezeichner festgelegt. Sind diese Konventionen erfüllt, spricht man von *Linked Data* [BHIBL08]. Linked Data verwendet RDF als Datenmodell und wird häufig synonym mit dem Begriff des Semantic Web verwendet [BHIBL08].

Die Eigenschaften von Linked Data umfassen die folgenden Punkte: Alle verwendeten Bezeichner beschreiben Verbindungen über das Hypertext Transfer Protocol (HTTP) [Int99] ($LD1$). Praktisch bedeutet das, dass die URIs mit „http“ beginnen. So ist es möglich, von der URI des Bezeichners Informationen abzurufen. Weiterhin wird gefordert, dass der unter dieser URI erreichbare Computer bei einer solchen Anfrage auch RDF-Daten, die genau diese Ressource näher beschreiben, an den anfragenden Computer sendet ($LD2$). Damit handelt es sich bei einem in Linked Data verwendeten Bezeichner für Ressourcen um einen Uniform Resource Locator (URL). Eine URL unterscheidet sich von einer URI dadurch, dass sie nicht nur eine Ressource eindeutig bezeichnet, sondern gleichzeitig beschreibt, wie auf diese Ressource zugegriffen werden kann [BLMM94]. Zusätzlich sollten RDF-Informationen Verknüpfungen zu anderen Ressource enthalten, damit weitere RDF-Daten über diese Ressource gefunden werden können ($LD3$).

Wird ein RDF-Graph nach den Prinzipien von Linked Data veröffentlicht, kann dieser Graph von anderen Computersystemen nur anhand einer im Graphen enthaltenen URI

abgerufen werden. Dieser Vorgang wird als *Dereferenzierung* bezeichnet und macht die beschriebene Integration von RDF-Daten aus verschiedenen Quellen möglich.

Diese Eigenschaften werden in Beispiel 2.5 erläutert. Ein weiteres Beispiel für eine Anwendung, die mittels Linked Data verschiedene Datenquellen kombiniert, ist „Researchers Map“. Die Anwendung visualisiert Publikationen von Wissenschaftlern in einer Karte. Researchers Map wurde vom Autor dieser Arbeit mitentwickelt [HMF09].

Beispiel 2.5

Wenn man das in Beispiel 2.2 genannte Szenario um Linked-Data-Eigenschaften erweitert, ergibt sich ein veränderter RDF-Graph, dessen Serialisierung in Listing 2.5 angegeben ist: Die URI der Ressource, die Herrn Stein beschreibt (Zeile 1), ist über HTTP dereferenzierbar ($LD1$). Wird diese URI dereferenziert, erhält das anfragende Programm genau die in dem Listing genannten Tripel ($LD2$). Diese Tripel enthalten mit *rdfs:sameAs* (Zeile 4) einen Verweis auf die andere Ressource, im Beispiel etwa Herr Steins Webseite bei seinem Arbeitgeber ($LD3$). Enthält die Webseite seines Arbeitgebers beispielsweise ein weiteres *ont:phone* -Tripel, kann eine Anfrage nach allen Tripeln, die eine Telefonnummer für Steins Ressource enthalten, sowohl die Telefonnummer der privaten Webseite als auch die der Webseite des Arbeitgebers ergeben.

```
1 <rdf:Description rdf:about="http://example.com/horststein">
2   <ont:name>Horst Stein</ont:name>
3   <ont:phone>030123456</ont:phone>
4   <rdfs:sameAs
5     rdf:resource="http://www.example.org/mitarbeiter/stein"/>
6 </rdf:Description>
```

Listing 2.5: Linked Data - Ausschnitt

2.2.4 Anfragesprache SPARQL

Sind Daten als RDF-Graph verfügbar, ist es nützlich, auf diesem RDF-Graphen Abfragen durchführen zu können. Es nicht effizient, für jedes Tripel den gesamten Graphen von einem möglicherweise entfernten Server abzurufen, um dann das eine interessante Tripel daraus zu extrahieren. Daher wurde die Sprache SPARQL Protocol and RDF Query Language (SPARQL) entwickelt, die es erlaubt, in einer SQL-ähnlichen Syntax *Anfragen* an einen RDF-Graphen zu stellen [PS08]. Ähnlich wie in der relationalen Algebra wird innerhalb einer Abfrage vom Typ *SELECT* eine Selektion und Projektion durchgeführt: In einer *WHERE*-Klausel werden die Elemente des RDF-Graphen ausgewählt, die als *Resultat* der Anfrage gewünscht sind. Hierzu wird spezifiziert, welche Eigenschaften die Ressourcen, die als Resultat erwünscht sind, aufweisen sollen. Dies stellt eine Selektion dar. Durch die Angabe der der für die Ausgabe relevanten Elemente der Selektion in der *WHERE*-Klausel wird eine Projektion durchgeführt. Weiterhin ist eine Sortierung der Ergebnisse sowie eine Begrenzung der Ergebnismenge analog zu SQL in SPARQL vorgesehen.

SPARQL spezifiziert sowohl die Abfragesprache als auch das Zugriffsprotokoll. Das Ergebnis einer SPARQL-Abfrage wird je nach Art der Anfrage meist in einem von SPARQL definierten XML-Format [BB08] geliefert. Hier werden alle von der Abfrage erfassten Daten dargestellt. Anders als SQL spezifiziert SPARQL an sich keine Möglichkeit, die RDF-Daten zu verändern, hierfür existiert jedoch SPARQL/Update, mit der SPARQL in die Lage versetzt wird, einen RDF-Graphen zu verändern [SMB+08]. SPARQL/Update wurde allerdings bis jetzt noch nicht als Standard verabschiedet.

Obwohl RDF ein anderes Datenmodell als relationale Datenbanken verwendet, lehnt sich SPARQL stark an SQL an. Einschränkungen werden jedoch auf Tripel-Basis getroffen. Der Nutzer ist dadurch in der Lage, Filterbedingungen auf Tripelebene anstelle von Attributen in Relationen anzuwenden. Als Resultat werden so genannte „Lösungen“ ausgegeben, das sind Elemente aus dem RDF-Graphen, die den Bedingungen der Anfrage entsprechen. Es kann für eine Anfrage durchaus mehrere Lösungen geben. In Beispiel 2.6 wird eine SPARQL-Anfrage und deren Resultat in Tabelle 2.1 dargestellt.

Beispiel 2.6

Herr Stein veranstaltet einen Männerabend. Er weist daher sein Adressbuch an, ihm Namen und Telefonnummern von drei beliebigen Freunden auszugeben, die männlich sind. Die Liste soll nach Namen sortiert sein. Es wird angenommen, dass dem in Beispiel 2.3 gezeigten Vokabular entsprechende Daten im RDF-Graphen verfügbar sind. Einige Instanzen des im Vokabular definierten Konzepts sind im Anhang in Listing A.5 dargestellt.

Steins Adressbuchprogramm übersetzt diese Bedingung nun in die in Listing 2.6 gezeigte SPARQL-Anfrage: Zunächst wird mit PREFIX eine Abkürzungskonvention für die URIs der Tripel-Elemente festgelegt, um die Lesbarkeit zu verbessern. Präfixe definieren eine Ersetzung, hier werden alle Vorkommen von *ont:* in URIs durch <`http://www.example.com/ont#`> ersetzt. Das SELECT-Schlüsselwort legt den Typ der Anfrage fest, darauf folgt eine Auflistung der Variablen, die im Resultat erscheinen. Innerhalb der WHERE-Klausel werden nun die Bedingungen für die Tripel angegeben, die dieser Anfrage entsprechen (Selektion). Die verwendete Schreibweise ähnelt dem erwähnten N3-Format. ?person ist eine temporäre Variable, die das Subjekt bezeichnet. Für dieses Subjekt gilt nun, dass der Wert der Eigenschaft ont:Geschlecht ont:maennlich sein soll, und dass an die Ausgabevariablen (Projektion) ?name und ?phone die jeweiligen Werte der verwendeten Beispielontologie gebunden werden sollen. Schließlich wird das Resultat noch analog zu SQL mit ORDER BY sortiert und mit LIMIT in seiner Länge begrenzt. In Tabelle 2.1 wird das Resultat dieser Anfrage beispielhaft dargestellt.

?name	?phone
Andreas Ast	030433344
Frank Schmidt	030286622
Richard Müller	030344554

Tabelle 2.1: Resultat der SPARQL-Anfrage aus Listing 2.6

```
1 PREFIX ont: <http://www.example.com/ont#>
2 SELECT ?name ?phone
3 WHERE {
4    ?person ont:Geschlecht ont:maennlich.
5    ?person ont:Name ?name.
6    ?person ont:Phone ?phone.
7 }
8 ORDER BY ?name
9 LIMIT 3
```

Listing 2.6: SPARQL-Anfrage

2.2.5 Topologie des Semantic Web

Die allgemeine Systemarchitektur des Semantic Web entspricht dem auch im WWW verbreiteten Client-Server-Prinzip: Ein *Server* ist ein Computersystem, das Dienste für andere *Computersysteme* bereitstellt. Die menschlichen *Nutzer* bedienen sich wiederum eines als *Client* bezeichneten Computersystems, um die vom Server angebotene *Funktionalität* in Anspruch zu nehmen. Als Funktionalität wird die Menge durch einen Server angebotenen Dienste bezeichnet. Ein *Dienst* stellt eine spezifische Dienstleistung zur Verfügung. Der Zugriff auf einen Dienst kann durch verschiedene *Operationen* erfolgen.

Diese Beschreibung verallgemeinert die in [Int08] genannte Definition, nach der ein Server ein Programm ist, dass Dienste bereitstellt, da ein Computersystem sowohl Hard- als auch Software umfasst. Die Bündelung der von einem Server angebotenen Dienste in seiner Funktionalität entkoppelt die angebotenen Dienste von einer konkreten Implementierung eines Servers.

In Beispiel 2.7 wird ein Beispiel für die Verwendung der genannten Begriffe angegeben.

Beispiel 2.7

> Ein Server zur Verwaltung von Bildern stellt die Funktionalität zur Verwaltung von Bildern bereit. Diese besteht aus verschiedenen Diensten, Erstellung von Bildern, Veränderung von Bildern, Entfernung von Bildern und Abruf von Bildern. Der Dienst zur Veränderung von Bildern unterstützt Operationen zur Skalierung und zur Rotation einzelner Bilder. Ein menschlicher Nutzer benutzt nun ein Computersystem, um diese Funktionalität in Anspruch zu nehmen, in dem Bilder zwischen Client und Server übertragen werden, und der Dienst zur Veränderung aufgerufen wird.

Im Umfeld des Semantic Web werden die Programme zum Abruf und zur Auswertung von dort veröffentlichten Daten häufig als Agenten bezeichnet, um ihre Fähigkeit zum eigenständigen „Handeln“ innerhalb des Semantic Web zu zeigen [Hen01].

[VOO95] beschreibt Agenten als Programme, die das für die Abarbeitung von speziellen Aufgaben notwendige Wissen enthalten. Diese Aufgaben können dem Agenten übertragen werden, zusätzlich ist es möglich, dass der Agent anhand seiner Programmierung selbst entscheidet, eine Aufgabe abzuarbeiten.

Nach [Lew08] ist ein Agent ein Computersystem, dass innerhalb einer Umgebung in der Lage ist, im Sinne seiner Programmierung diese Umgebung zu erfassen und mit ihr zu interagieren. Reaktive Agenten handeln aufgrund ihrer Wahrnehmung der Umgebung und möglicherweise vorherigen Wahrnehmungen oder allgemeinen Zielen. Alternativ werden nicht-reaktive Agenten beschrieben, die auf Anforderungen beispielsweise Schnittstellen oder Informationen bereitstellen.

Im Allgemeinen kann jedoch nicht davon ausgegangen werden, dass jeder Client, der mittels der Methoden des Semantic Web auf dort abgelegte Daten zugreift, sich durch die Definitionen eines Agenten beschreiben lässt. Einer großen Menge von Clients für Semantic-Web-Anwendungen fehlt die Möglichkeit zur eigenständigen Interaktion mit dem Semantic Web. Diese Clients können in der Begriffswelt des bekannten WWW am ehesten mit Webbrowsern verglichen werden. Daher wird im Folgenden lediglich der Begriff *Client* verwendet, ohne jedoch auszuschließen, dass die von Server angebotene Funktionalität auch durch intelligente Agenten genutzt werden kann.

Wie in Abbildung 2.3 dargestellt wird die Struktur des Semantic Web als eine Topologie von Servern, Clients und Nutzern beschrieben: Das Semantic Web als Weiterentwicklung des WWW besteht wie dieses aus einer Vielzahl eigenständiger Server, die von verschiedenen Organisationen oder Privatpersonen betrieben werden. Die menschlichen Nutzer des Semantic Web bedienen sich eines Clients, um auf die von den Servern angebotenen Dienste zuzugreifen [Lew08]. Die Clients stellen somit die Schnittstelle zwischen Benutzer und System dar und werden benutzt, um Informationen im Semantic Web zu veröffentlichen, zu verarbeiten oder abzurufen.

Die in der Grafik dargestellten Pfeile symbolisieren den Austausch von RDF-Daten. Nutzer benutzten einen Client, um Daten auf Servern zu veröffentlichen (I). Ein Client kann auch benutzt werden, um Anfragen an diese Server zu stellen (I). Server können sich zur Beantwortung von Anfragen anderer Server bedienen (IIa), und der Client ist in der Lage, die von zwei Servern gelieferten Ergebnisse zu zu integrieren (IIb/c). Innerhalb des Semantic Web sind keine weiteren Strukturen und Hierarchien aus dem Aufbau des Netzes erkennbar. Prinzipiell sind alle Server gleichberechtigt, ihre Zugehörigkeit zu Organisationen oder Privatpersonen wird durch andere Systeme festgelegt. Eine solche Zugehörigkeit kann zum Beispiel über ein Zertifikatsystem oder Namenskonventionen festgelegt werden [RSA78].

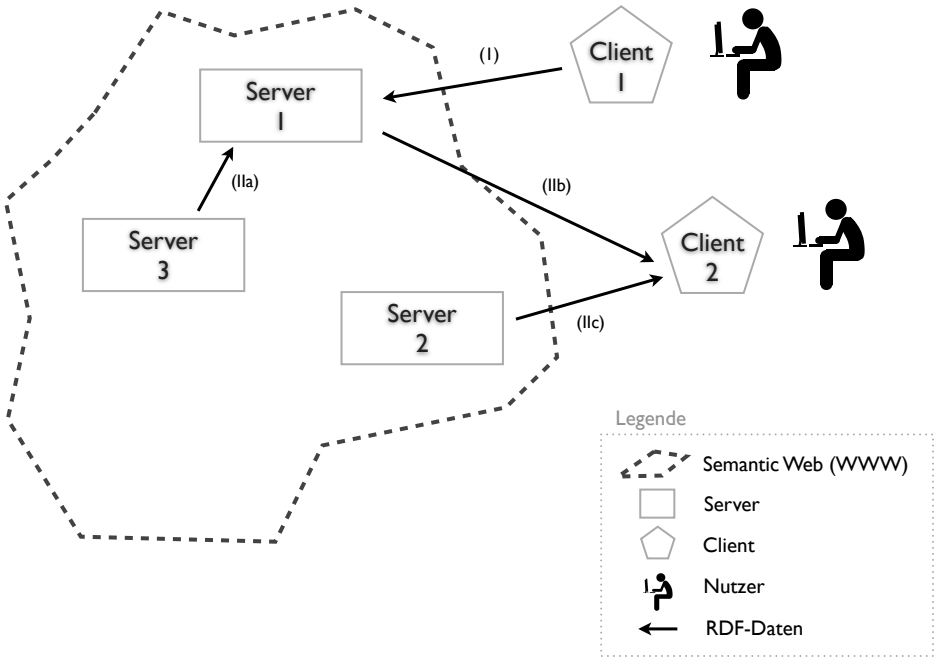

Abbildung 2.3: Topologie des Semantic Web

2.3 Zugriffsrichtlinien

Die in Abschnitt 2.2 beschriebenen Techniken gehen davon aus, dass jeder WWW-Nutzer alle von anderen Nutzern veröffentlichten Informationen abrufen kann. Es gibt jedoch Fälle, in denen es wünschenswert ist, bestimmte Informationen nur einem beschränkten Personenkreis zugänglich zu machen. Wie bereits beschrieben, stellt die Veröffentlichung personenbezogener Daten im Internet einen solchen Fall dar.

Im WWW wird dieses Problem bisher meist entweder über eine Anmeldung mit Benutzername und Passwort an einer bestimmten Webseite oder über eine Beschränkung der Computer, von denen aus bestimmte Informationen abgerufen werden können gelöst. Gerade im Bereich der Webseiten, deren Inhalte durch die Nutzer erstellt werden, wie zum Beispiel das eingangs erwähnte Facebook, ist es notwendig, dass der jeweilige Nutzer vom Server eindeutig identifiziert werden kann. Der Server erzeugt daraufhin anhand der Identifikation eine angepasste Webseite. Welche konkreten Informationen ein Nutzer angezeigt bekommt hängt von der entsprechenden Webseite ab. Die Betreiber legen über die Programmierung der Webseite fest, welche Informationen welchem Nutzer unter welchen Bedingungen angezeigt werden. Damit erstellen sie für die gesamte Website gültige Zugriffsrichtlinien. Der Nutzer hat jedoch oft nur eingeschränkte Kontrolle über diese Richtlinien und deren Durchsetzung baut darauf auf, dass alle Informationen zentral verwaltet werden.

Im Linked-Data-Umfeld ist dies jedoch schwieriger: Daten sind möglicherweise über viele Server verteilt und nur über entsprechende Referenzen verknüpft. Antworten auf Anfragen können Tripel, die von verschiedenen Servern geliefert werden enthalten. Damit lässt sich das eben beschriebene Szenario nicht mehr durchsetzen. Der Nutzer müsste einen Weg haben, sich gleichzeitig bei allen Servern, die für eine Beantwortung der Anfrage in Frage kommen, anzumelden. In einem zweiten Schritt müssten die Daten mit expliziten Richtlinien versehen werden, da deren implizite Definition über die Programmierung der Webseite – wie eben beschrieben – in einem generischen Umfeld nicht mehr durchführbar ist.

In diesem Abschnitt werden Systeme zur expliziten Formulierung von Zugriffsrichtlinien vorgestellt, die der Definition 2.6 entsprechen. Diese Richtlinien sollen es dem veröffentlichenden Nutzer erlauben, die Nutzer oder Programme festzulegen, die Zugriff auf die veröffentlichten Daten haben sollen. Damit kann der Nutzer kontrollieren, unter welchen Voraussetzungen seine Informationen ausgewählten Dritten zugänglich gemacht werden. Richtlinien enthalten zusätzlich noch eine so genannte *Klassifikation*, die die Datenobjekte beschreibt, die von den jeweiligen Regeln referenziert werden.

Definition 2.6 *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 [Lat85].*

In Verbindung mit den in Abschnitt 2.1 beschriebenen Methoden zur eindeutigen und sicheren Identifizierung von Nutzern oder Computern ist es möglich, den Nutzern des Semantic Web die Kontrolle über die von ihnen veröffentlichten Daten zu geben und gleichzeitig die Funktion und wünschenswerten Eigenschaften des Semantic Web zu erhalten.

Zugriffsrichtlinien können in natürlicher Sprache formuliert werden. Damit diese Richtlinien jedoch von einem Programm interpretiert und durchgesetzt werden können, ist eine maschinenlesbare Formulierung notwendig. Für diese Formulierung existieren verschiedene Ansätze und Implementierungen, die im Folgenden vorgestellt werden:

Unterabschnitt 2.3.1 stellt verschiedene Arten der Definition von Zugriffsrichtlinien vor. Unterabschnitt 2.3.2 beschreibt die Notwendigkeit und Möglichkeiten, schützenswerte Daten auf verschiedenen Abstraktionsebenen als solche zu markieren, Unterabschnitt 2.3.3 zeigt exemplarisch den Aufbau und eine Darstellung von Richtlinien und Regeln.

2.3.1 Richtlinientypen

Es existieren verschiedene Ansätze, wie Richtlinien definiert und durchgesetzt werden. Im Folgenden werden drei wichtige Muster für den Entwurf von Zugriffsrichtlinien vorgestellt.

Discretionary Access Control

Ein System, das Discretionary Access Control (DAC) unterstützt, kann zwischen benannten Nutzern und benannten Objekten (zum Beispiel Dateien) unterscheiden. Es existiert ein Mechanismus, der den Nutzern erlaubt (aber nicht vorschreibt), die Zugriffsrechte auf diese Objekte anderen Nutzern oder Nutzergruppen zu geben. Außerdem muss eine Möglichkeit gegeben sein, das Propagieren (Automatische Weitergabe auf Unterobjekte) von Rechten einzuschränken. Nutzer, die keine Zugriffsrechte auf Objekte besitzen, können nicht auf diese zugreifen. Die Granularität der Zugriffsrechte kann bis auf die Ebene der einzelnen Nutzer verfeinert werden. Nur Nutzer, die Zugriffsrechte auf ein Objekt besitzen, können Zugriffsrechte auf dieses Objekt vergeben [Lat85].

Ein bekanntes – wenn auch nicht vollständiges – Beispiel für ein solches System ist das UNIX-Dateisystem. Ein UNIX-System besitzt für jede Datei (ein Datenobjekt) einen Verweis auf den Besitzer dieser Datei. Der Besitzer kann nun Lese-, Schreib- und Ausführungsrechte für jede Datei oder in Ordnern gruppierte Dateien getrennt vergeben. Zusätzlich können diese Rechte auch noch für andere Nutzer festgelegt werden.

Mandatory Access Control

Bei Mandatory Access Control (MAC) muss jedes Objekt und jeder Nutzer durch eine Zugriffsrichtlinie beschrieben oder berechtigt sein. Jedes Objekt wird durch eine Sicherheitsklassifikation weiter beschrieben. Diese Sicherheitskategorien können hierarchisch angeordnet sein und sie stellen die Basis für die Entscheidung über den Zugriff von Nutzern auf Objekte dar. Ein System mit MAC unterstützt mehrere Klassifikationsebenen. Nur Nutzer, die die gleiche oder eine höhere Sicherheitsklassifikation wie ein Objekt besitzen, können dieses lesen oder bearbeiten. Die Nutzer werden vom System eindeutig identifiziert, um Eingaben von außerhalb des Systems sicher einem Nutzer und dessen Zugriffsrichtlinien und -privilegien zuordnen zu können. Gleichzeitig können die Nutzer nicht wie bei DAC Rechte auf Objekte anderen Nutzern weitergeben, dies muss über eine Veränderung der Sicherheitsklassifikation des Objekts oder des Nutzers erreicht werden. [Lat85].

Ein solches System könnte beispielsweise in einer Behörde mit Sicherheitsaufgaben eingesetzt werden. Solche Behörden klassifizieren ihre Informationen meist über verschiedene Gruppen der notwendigen Geheimhaltung wie etwa in Deutschland „nur für den Dienstgebrauch“, „vertraulich“, „geheim“ und „streng geheim“. Wird nun jedem Mitarbeiter dieser Behörde eine Sicherheitsklassifikation zugewiesen, kann ein EDV-System diese Geheimhaltungsstufen überprüfen und damit durchsetzen.

Role-based Access Control

Role-Based Access Control (RBAC) unterscheidet anders als die beiden genannten Verfahren nicht zwischen einzelnen Nutzern. Dies kann sinnvoll sein, da in vielen Anwendungen anders als in DAC es keine Rolle spielt, welcher spezifische Nutzer eine Datei bearbeitet. Eine Rolle (Role) beschreibt einen Aufgabenbereich und damit verbunden Zugriffsrechte, den ein Nutzer im Wirkungsbereich des Systems hat (zum Beispiel „Sachbearbeiter“). Die Zugriffsrechte werden nun ausschließlich diesen Rollen zugeordnet. Die Nutzer wiederum werden entsprechend ihres Aufgabenbereichs ebenfalls (durchaus auch mehreren) Rollen zugeordnet. Mit RBAC ist einfach möglich, die Rechte eines Nutzers zu verändern, wenn dieser einem anderen Tätigkeitsbereich zugeordnet wird: Hier werden einfach die entsprechenden Rollen aktiviert beziehungsweise deaktiviert. RBAC ist insgesamt eine vereinfachte Form der MAC ohne die komplexe Hierarchisierung von Rechten, jedoch sind Komposition und Ableitung von Rollen durchaus möglich [FK92].

Ein Beispiel für ein RBAC-basiertes System könnte ein System zur Verwaltung von Kursen an einer Universität sein. Hier haben Nutzer, die die Rolle „Lehrbeauftragte“ besitzen, beispielsweise dadurch das Recht, Kurse anzulegen. Nutzer mit der Rolle „Student"haben dadurch lediglich das Recht, sich in Kurse eintragen.

2.3.2 Datenklassifikation

Um Richtlinien für den Zugriff auf RDF-Daten (Unterabschnitt 2.2.1) erstellen, muss klar definiert sein, welche Elemente des RDF-Graphen von einer Richtlinie betroffen sind. Diese Abgrenzung wird hier als Klassifikation bezeichnet. Soll auf beliebige Daten zugegriffen werden, muss überprüft werden, ob diese in den Wirkungsbereich einer Richtlinie fallen. Ist das der Fall, werden die dazugehörigen Richtlinien ausgewertet. Es kann sinnvoll sein, die Klassifikation der Daten nicht direkt in der eigentlichen Richtlinie vorzunehmen, sondern mittels eines Bezeichners auf eine getrennte Klassifikation zu verweisen. Dann kann innerhalb von Richtlinien einfach auf diese Daten verwiesen werden. Zudem wird Definition von Richtlinien und Definition von Klassifikationen getrennt, was flexiblere Richtlinien erlaubt. Es stellt sich die Frage, auf welcher Ebene des in Abschnitt 2.2 beschriebenen Modells Daten klassifiziert werden sollen. Grundsätzlich wird – mit größer werdender Abstraktion – zwischen der syntaktischen Ebene, der Modellebene und der semantischen Ebene unterschieden. Als syntaktische Ebene kann zum Beispiel eine RDF/XML-Serialisierung eines RDF-Graphen bezeichnet werden. Die Elemente eines RDF-Graphen wie Subjekt, Prädikat und Objekt sowie deren Beziehungen untereinander befinden sich auf der Modellebene. Zuletzt liegt die aus der Gesamtheit der RDF-Graphen gewonnene Information

auf der semantischen Ebene. Es ist meist möglich, Klassifikationen für höhere Ebenen in Klassifikationen auf niedrigen Ebenen zu übersetzen. So kann eine Klassifikation über ein RDF-Konzept leicht in eine Klassifikation der darin definierten Tripel überführt werden.

Im Folgenden werden die verschiedenen Ebenen für Klassifikationen erläutert. Abschließend findet sich in Beispiel 2.8 ein Beispiel für Klassifikationen auf verschiedenen Ebenen. Tabelle 2.2 zeigt die identifizierten Klassifikationsebenen als Übersicht.

Ebene	*erfasste Elemente*	*sinnvoll?*
Syntaktische Ebene	z.B. XML-Tags	✗
Modellebene	RDF-Tripel	✓
Semantische Ebene	Instanzen	✓
Ressourcenebene	URIs	✓

Tabelle 2.2: Klassifikationsebenen

Syntaktische Ebene

Es ist in RDF **nicht** möglich, Richtlinien etwa wie in [KH00] auf der syntaktischen Ebene der RDF/XML-Serialisierung zu erstellen, da die XML-Darstellung wie in Unterabschnitt 2.2.1 beschrieben nur eine von vielen möglichen Serialisierungen darstellt, in denen RDF dargestellt werden kann. Zusätzlich sind innerhalb des RDF/XML-Formates verschiedene semantisch äquivalente jedoch syntaktisch unterschiedliche Darstellungen möglich. Genauso problematisch würde es sich mit Richtlinien auf N3-Ebene verhalten. Um eine Richtlinie auf syntaktischer Ebene zu formulieren, müsste die Serialisierung der Elemente des RDF-Graphen fest vorgegeben werden. Um syntaktische Unterschiede in der Darstellung eines Graphen zu verdeutlichen, beschreiben die in Listing 2.7 sowie in Listing 2.8 beispielhaft angegebenen RDF/XML-Serialisierungen identische RDF-Graphen. Beide Serialisierungen enthalten eine Beschreibung der Ressource <`http://example.com/horststein/pos`>. Dieser Ressource ist eine Instanz des Konzepts *Position*, diese Festlegung ist in unterschiedlichen Schreibweisen möglich. Weitere Eigenschaften der Ressource sind *longitude* und *latitude*, die entweder als XML-Knoten oder als XML-Attribute definiert werden können.

```
1 <rdf:Description rdf:about="http://example.com/horststein/pos"
      rdf:type="&ont;Position">
2   <ont:latitude>48.755723</ont:latitude>
3   <ont:longitude>9.190233</ont:longitude>
4 </rdf:Description>
```

Listing 2.7: Positionsdaten als RDF/XML - Ausschnitt

```
1 <Position rdf:about="http://example.com/horststein/pos"
2   latitude="48.755723" longitude="9.190233" />
```

Listing 2.8: Positionsdaten als RDF/XML - Alternative Darstellung - Ausschnitt

Modellebene

Eine Klassifikation der RDF-Aussagen (Tripel) auf der Modellebene sind die kleinste Einheit, die sinnvoll von Richtlinien beschrieben werden kann. Tripel sind von der textuellen Darstellung unabhängig und sind die kleinste Einheit im Datenmodell von RDF. Damit sind sie der erste Ansatzpunkt für Klassifikationen. Tripel können am Einfachsten über so genannte Muster erfasst werden. Bei Tripelmustern (engl. triple patterns) werden von den drei Elementen Subjekt, Prädikat und Objekt entweder Werte vorgegeben oder ein Platzhalter eingesetzt, der für beliebige Werte stehen kann. Ähnlich der in Unterabschnitt 2.2.4 eingeführten Abfragesprache SPARQL sind damit alle Tripel, die von diesem Muster betroffen sind über einen zusätzlichen Bezeichner identifizierbar. Ein Ansatz zur Klassifikation von Tripeln wird in [JF06] vorgeschlagen. Hier besteht eine Klassifikation SP aus einer Menge von Tupeln sp_n, die als erstes Element ein Muster für Tripel und als zweites Element einen Bezeichner enthalten. Das Tripelmuster beschreibt, welche Tripel von der Klassifikation betroffen sind. Jeweils für Subjekt, Prädikat und Objekt können Platzhalter eingesetzt werden, die einen beliebigen Wert bezeichnen. Diese Muster können nun verwendet werden, um bei der Beantwortung von Anfragen zu entscheiden, ob ein Tripel von einer Richtlinie betroffen ist. Ist das der Fall, wird die Richtlinie ausgewertet und die Auslieferung des Tripels entweder erlaubt oder verweigert. Die eigentlichen Richtlinien verweisen dort auf die vergebenen Bezeichner zur Klassifikation.

Semantische Ebene

Es ist sinnvoll, die gewünschten Richtlinien auf Basis der via RDFS oder OWL deklarierten Konzepte oder Instanzen zu definieren. Damit können mehrere Tripel auf einmal geschützt werden. Der Schutz kann auch neu hinzugefügte Eigenschaften des geschützten Konzepts betreffen. Zusätzlich können durch Schlüsse (Propagation) aus den Ontologie-Daten auch aus bestehenden Konzepten abgeleitete RDF-Konzepte automatisch als geschützt markiert werden. Dies ist sinnvoll, da es leicht möglich ist, die gleiche Aussage durch unterschiedliche Tripelkombinationen auszudrücken. Dadurch steigt die Ausdrucksstärke der Richtlinien mit dem Grad der Abstraktion von den Daten. Ein Ansatz, der RDF-Schemata bei der Definition und Auswertung von Richtlinien berücksichtigt findet sich in [RFJ07].

Ressourcenebene

Im Linked-Data-Bereich ist es möglich, über die Bezeichner von Ressourcen anhand ihrer URI-Form Inhalte zu klassifizieren. Hierbei kann eine gesamte Ressource als geschützt markiert werden. Damit die Clients diese Ressource in Betracht zu ziehen können, wird in eine öffentliche Ressource ein Verweis auf die geschützte Ressource eingefügt. Dieses Konzept

wird ausführlich in [SHJJ09] beschrieben. Problematisch hierbei ist, dass Ressourcen, die sowohl öffentlich verfügbare sowie geschützte Tripel enthalten mit diesem Konzept nicht geschützt werden können. Für ein solches Szenario müssten zwei Ressourcen angelegt werden, die dann jeweils aufeinander verweisen. Gleichzeitig muss für jede unterschiedliche Art des Zugriffs eine neue Ressource angelegt werden. Beispielsweise unterscheiden sich Informationen, die für die Freunde einer Person zugänglich sind, von denen, die für die Eltern zugänglich sind und von denen, die für die Geschwister zugänglich sind. In diesem Fall müssten vier verschiedene Ressourcen angelegt werden.

Beispiel 2.8

Herr Stein möchte seine GPS-Position im Internet veröffentlichen. Allerdings ist er nicht damit einverstanden, dass jeder beliebige Nutzer herausfinden kann, wo er sich gerade aufhält. Daher markiert er die aus Länge und Breite bestehende GPS-Position als schützenswert. Diese Kombination erhält den willkürlichen Bezeichner *gpspos*.

Eine Klassifikation auf Modellebene ist möglich, im Beispielfall wird Subjekt <`http://example.com/horststein/pos`> und Prädikate *ont:latitude* bzw. *ont:longitude* unter Verwendung eines Standardnamensraums festgelegt, damit sind die Tripel, das die Position von Herrn Stein enthalten eindeutig beschrieben. Nun kann der Zugriff auf die Position von einer Richtlinie eingeschränkt werden. Eine Klassifikation von Elementen des RDF-Graphen mit dem Bezeichner *gpspos* nach [JF06] für diesen Fall lautet beispielsweise:
$sp_1 := ([< http://example.com/horststein/pos >, ont : latitude, ?x],' gpspos')$
$sp_2 := ([< http://example.com/horststein/pos >, ont : longitude, ?x],' gpspos')$
$SP := \{sp_1, sp_2\}$
Der Platzhalter $?x$ bezeichnet hier einen beliebigen Wert.

Weiterhin könnte das verwendete Konzept *Position* auf der semantischen Ebene als zu *gpspos* gehörig klassifiziert werden. Schließlich wäre es möglich, auf der Ressourcenebene die Daten über ihre URI <`http://example.com/horststein/pos`> als geschütztes Objekt zu klassifizieren.

2.3.3 Richtliniendefinition

Aufbauend auf der Klassifikation der Daten können nun die eigentlichen Richtlinien erstellt werden, die festlegen, wer in welchem Umfang auf die Daten zugreifen kann. Diese Richtlinien können sich auf verschiedenste Informationen beziehen. Im Bezug auf Zugriffskontrolle werden die Identität und die verfügbaren Informationen über den anfragenden Nutzer zentrale Bedeutung haben. Gleichzeitig kann es notwendig sein, die Richtlinien selbst als zu schützende Information zu markieren. Das diese Richtlinien verarbeitende System stellt daraufhin sicher, dass nur Nutzer, die sich in geeigneter Weise authentifiziert haben, diese Daten abrufen können.

Regeln werden üblicherweise in einem Prolog-ähnlichen Format angegeben, das sich auf Hornformeln zurückführen lässt, um die Auswertung der Regeln einfach zu halten. In Beispiel 2.9 ist ein Beispiel für eine Regel nach dem in [RFJ07] verwendeten Format angegeben. Die verwendeten Prädikate sind fest in diesem System vorgegeben.

Beispiel 2.9

Angenommen Herr Stein will seine GPS-Position nur den Mitgliedern seiner Familie zugänglich machen, könnte er folgende Regel erstellen:

$permit(see(U, (\texttt{http://example.com/horststein/pos}, ?, ?))) :-$
$existTriple(U, \texttt{ont:familyMember}, \texttt{http://example.com/horststein})$

In natürlicher Sprache wird hier ausgedrückt: Der Nutzer U hat die Erlaubnis (*permit*) die Tripel, die die Position von Herr Stein <`http://example.com/horststein/pos`> beschreiben, zu sehen (*see*), wenn U ein Familienmitglied (*ont:familyMember*) von Herr Stein <`http://example.com/horststein`> ist.

Semantic Web Rule Language

Für die Datenstrukturen, die durch RDFS und OWL beschrieben werden, ist die Formulierung von Regeln anspruchsvoll. Zu diesem Zweck wurde daher die Semantic Web Rule Language (SWRL) entwickelt, die auf der Basis von OWL und der als RuleML [LS03] bezeichneten Sprache zur Formulierung von Regeln aufbaut. Auf SWRL aufbauend wird derzeit RIF entwickelt, das auf der Basis der Prädikatenlogik verschiedene Dialekte für den Ausdruck von Regeln [BHK+07, BK07] definiert. Im Folgenden werden jedoch SWRL-Regeln benutzt, da RIF noch nicht (Stand: August 2009) in einer finalen Version bereitsteht und außerdem noch keine Implementierungen existieren. SWRL-Regeln können als RDF-Graph dargestellt werden. Damit lassen sich SWRL-Regeln leicht in Semantic-Web-Anwendungen verwenden.

Regeln in SWRL beschreiben Implikationen und bestehen aus zwei Listen von Prädikaten, dem Bedingungsteil (engl. antecedent) und dem Aktionsteil (engl. consequent). Die Semantik entspricht der der Implikation, wenn alle Prädikate des Bedingungsteils den booleschen Wert *true* annehmen, werden alle Prädikate des Aktionsteils ausgewertet. Die verwendbaren Prädikate sind von der Sprachspezifikation vorgegeben [HPSB+05].

Die folgende Liste zeigt die verwendbaren Prädikate in SWRL-Regeln und gibt die Bedingung an, nach der diese Prädikate den booleschen Wert *true* annehmen:

- $C(x)$ - Ein Objekt x ist eine Instanz des Konzepts C.
- $D(z)$ - Der Wert z ist vom Datentyp D.
- $P(x, y)$ - Das Objekt x hat eine Eigenschaft P mit Verweis auf das Objekt y.
- $Q(x, z)$ - Das Objekt x hat eine Eigenschaft Q mit dem Wert z.
- $sameAs(x, y)$ - Die Objekt x und y sind äquivalent.
- $differentFrom(x, y)$ - Die Objekt x und y sind nicht äquivalent.
- $builtin(r, z_1, \ldots, z_n)$ - Die Funktion r mit den Parametern z_* liefert *true*. Als Funktion stehen eine Reihe von Standardfunktionen wie etwa Stringvergleiche vordefiniert bereit. Es ist zusätzlich vorgesehen, dass Nutzer weitere Funktionen definieren.

Da die XML-basierte Darstellung von SWRL für Menschen nicht gut lesbar ist, wird auch sie üblicherweise in einer Prolog-ähnlichen Notation vorgestellt [HPSB+05, MSS05]. Diese Notation wird in Beispiel 2.10 gezeigt. Auffällig ist, dass der Nutzer in SWRL beliebige Prädikate durch die Angabe einer Ontologie definieren und auswerten kann. Problematisch hierbei ist, dass der Standard keine direkte Übersetzung dieser kompakten Darstellung in die SWRL-Syntax vorsieht.

Beispiel 2.10

$Person(U) \wedge isFamilyMember(U, \texttt{http://example.com/horststein}) \rightarrow$
$allowAccessTo(U, \texttt{http://example.com/horststein/pos})$

Die Semantik dieser Regel ist mit der in Beispiel 2.9 beschriebenen Regel vergleichbar. Die verwendeten Prädikate sind $C(x)$ für $Person(U)$ sowie $P(x, y)$ für $isFamilyMember$. Mit $allowAccessTo$ ist ein Prädikat nach $Q(x, z)$. Die Regel wird wie folgt ausgewertet: Falls eine mit der Variable U bezeichnete Ressource eine Instanz des Konzepts $Person$ bezeichnet, sowie die Beziehung $isFamilyMember$ zwischen U und der Ressource mit dem Bezeichner <`http://example.com/horststein`> besteht, so wird der Ressource U der Wert <`http://example.com/horststein/pos`> für die Eigenschaft $allowAccessTo$ zugewiesen. Ein Programm kann nun diese Eigenschaft von U auswerten, um den Zugriff auf die Ressource <`http://example.com/horststein/pos`> zu schützen. Damit ist es möglich, das genannte Beispiel – der Schutz der Ressource, die die Position von Herrn Stein beschreibt – umzusetzen.

Versuchsweise Umsetzungen von SWRL-Regeln im Rahmen dieser Arbeit haben gezeigt, dass die manuelle Formulierung dieser Regeln entweder in der eigenen XML-Syntax oder in der alternativen RDF-Darstellung nur schwierig möglich ist. Aus diesem Grund sind Werkzeuge notwendig, um SWRL-basierte Richtlinien erstellen und verwalten zu können.

2.4 HTTP-Verschlüsselung und Authentifizierung

Die Datenübertragung zwischen Systemen, die Linked-Data-Information wie in Unterabschnitt 2.2.3 beschrieben entweder bereitstellen oder abrufen, findet über das HTTP-Protokoll statt. Informationen, die über HTTP übertragen werden, können jedoch leicht während ihrer Übertragung abgehört werden, da keine Verschlüsselung vorgesehen ist. Wenn ein System schützenswerte Informationen übertragen soll, ist die Verwendung einer Verschlüsselung notwendig und sinnvoll. Hier setzt das Hypertext Transfer Protocol Secure (HTTPS) Protokoll an. Hierbei kommunizieren zwei Systeme über einen verschlüsselten Kanal miteinander. Dadurch sind die übertragenen Daten vor Abhörversuchen Dritter geschützt. Damit ein verschlüsselter Kanal ohne den vorherigen Austausch eines Passworts möglich ist, wird asymmetrische Verschlüsselung verwendet [Int00, RSA78]. Innerhalb des verschlüsselten Kanals werden die Daten unverändert im HTTP-Format übertragen. Die Umstellung eines Systems von HTTP auf HTTPS ist daher meist leicht möglich. Üblicherweise wird HTTPS benutzt, um einen Server – etwa für Online-Banking – dem Client gegenüber sicher zu identifizieren und die Kommunikation zu verschlüsseln. Der Benutzer

kann dann verhältnismäßig sicher sein, auch wirklich mit dem Server seiner Bank verbunden zu sein. Im Folgenden wird gezeigt, wie HTTPS zudem benutzt werden kann, um ein System, dass eine Anfrage stellt, sicher zu authentifizieren. Nach einer Einführung in HTTPS und die genannte Authentifizierung in Unterabschnitt 2.4.1 wird in Unterabschnitt 2.4.2 gezeigt, wie die Prinzipien von Linked Data eine Vereinfachung dieser Authentifizierung möglich machen.

2.4.1 HTTPS und Client-Authentifizierung

Um eine HTTPS-Verbindung aufzubauen, werden die folgenden Schritte vom anfragenden System (Client) und dem antwortenden System (Server) ausgeführt. Im Vorfeld wurde ein asymmetrischer Schlüssel für den Server erstellt und auf diesem hinterlegt. Der öffentliche Schlüssel wurde mit dem privaten Schlüssel einer vertrauenswürdigen Zertifizierungsstelle (CA) signiert. Diese Kombination aus öffentlichem Schlüssel und Signatur wird als Zertifikat bezeichnet. Der Client besitzt eine Liste der öffentlichen Schlüssel vertrauenswürdiger Zertifizierungsstellen [Int00, CSF+08, DA99].

1 Der Client (etwa ein Webbrowser) sendet eine Anfrage an den standardisierten TCP-Port 443 auf dem der Adresse entsprechenden System. Beispielsweise führt der Aufruf der Adresse <`https://example.https`> zum Aufbau einer HTTPS-Verbindung mit dem System mit dem DNS-Namen example.https.

2 Der Server sendet sein Zertifikat an den Client. Der Client kann anhand seiner Liste vertrauenswürdiger CAs überprüfen, ob der entsprechende Server von einer dieser CAs zertifiziert wurde. Ist diese Überprüfung erfolgreich, ist der Server gegenüber dem Client sicher authentifiziert.

3 Der Client sendet optional sein Zertifikat an den Server. Der Server kann nun genauso wie im vorherigen Schritt überprüfen, ob der Client von einer vertrauenswürdigen CA zertifiziert wurde. Ist dieser Schritt erfolgreich, ist der Client gegenüber dem Server authentifiziert. Dieser Schritt wird bei der gängigen Anwendung von HTTPS zur Absicherung von Website-Besuchen nicht verwendet.

4 Der Client erzeugt einen zufälligen Sitzungsschlüssel und verschlüsselt ihn anhand des im Zertifikat des Servers enthalten öffentlichen Schlüssels. Nur der Besitzer des privaten Schlüssels zu diesem öffentlichen Schlüssel kann diesen entschlüsseln. Der zufällige Sitzungsschlüssel wird an den Server gesendet und dort entschlüsselt. Damit sind ausschließlich Client und Server im Besitz des Sitzungsschlüssels.

5 Mit Hilfe des Sitzungsschlüssels wird eine symmetrische Verschlüsselung sowie Entschlüsselung der Nutzdaten durchgeführt. Die Nutzdaten werden innerhalb dieses so genannten SSL-Tunnels im HTTP-Format übertragen.

Sollte nur eine Verschlüsselung des Kommunikationskanals und keine Authentifizierung durchgeführt werden, kann diese für Client oder Server abgeschaltet werden [HTT08, DA99].

2.4.2 Authentifizierung mit FOAF+SSL

Die häufig verwendeten Authentifizierungsmodelle mittels Benutzername und Passwort behindern die Funktionen des Semantic Web: Ist für die Dereferenzierung einer URI eine Benutzername-Passwort-Kombination erforderlich, kann dies nicht mehr automatisiert von einem Client erledigt werden, es wäre Benutzerinteraktion notwendig. Um dies zu vermeiden, kann die FOAF+SSL-Authentifizierung verwendet werden. Hierbei kann ein Client seine Identität nachweisen, muss dem Server aber im Vorfeld nicht bekannt sein. FOAF+SSL verwendet die Friend of a Friend (FOAF)-Ontologie zur Beschreibung von Personen innerhalb von RDF-Graphen sowie das HTTPS-Protokoll, bei dem HTTP-Anfragen über einen verschlüsselten Kanal übertragen werden [SHJJ09, Dum02, Int00].

Die hier beschriebene Methode zur passwortlosen Authentifizierung für HTTP kann mit dem in [HTT08, DA99] beschriebenen HTTPS statt.FOAF+SSL macht sich die selten genutzte Möglichkeit der Client-Authentifizierung zu Nutze: Der Client sendet mit seiner kryptographisch signierten Anfrage auch einen öffentlichen Schlüssel mit, anhand dessen der Server feststellen kann, ob diese Anfrage mit dem passenden privaten Schlüssel signiert wurde [RSA78]. Der Server kann daraufhin anhand seiner Konfiguration entscheiden, ob der Benutzer oder der Herausgeber des Zertifikats, mit dem der Schlüssel signiert wurde, vertrauenswürdig sind. Diese Konfiguration muss jedoch im Vorfeld vorgenommen werden, und ist daher für die von Linked Data geförderte spontane Vernetzung und Integration von einander prinzipiell unbekannten Servern hinderlich. Hier geht FOAF+SSL wiederum einen anderen Weg. Der Client überprüft das Zertifikat des Servers, um sicher zu sein, mit einem bestimmten Server verbunden zu sein. Der Server jedoch fordert zwar ein Zertifikat an, muss dessen Signatur jedoch nicht überprüfen.

Mittels RDF kann für jeden Nutzer eine URI vergeben werden. Unter dieser URI wird die Serialisierung eines RDF-Graphen hinterlegt, die Informationen aus dem FOAF-Vokabular enthält [Dum02]. Dazu muss die Person, die dieses FOAF-Dokument erstellt, Kontrolle über den Server haben, der für die darin verwendete URI zur Bezeichnung der Person Daten ausliefert. Kann für eine Anfrage nun gezeigt werden, dass die Person, die die Anfrage stellt, mit der Person die auch die FOAF-Information unter dem Bezeichner dieser Person erstellt hat, identisch ist, ist klar, dass diese Anfrage von der so bezeichneten Person ausgeht. Hierzu fügt der Nutzer seinen FOAF-Informationen die Metadaten eines kryptographischen Schlüssels hinzu, im Fall von RSA etwa den Modulus und den Exponenten des privaten Schlüssels [RSA78]. Gleichzeitig wird dem Zertifikat, das bei der HTTPS-Anfrage an einen Server geschickt wird, die URI des Nutzers in einem dort für allgemeine Metadaten vorgesehenen Feld hinzugefügt. Einem Zertifikat können problemlos auch mehrere Bezeichner hinzugefügt werden.

Der Server empfängt nun bei der Anfrage das Zertifikat und extrahiert eine oder mehrere URIs, die den Benutzer bezeichnen, daraus. Für jede dieser URIs wird nun versucht, RDF-Information unter dieser URI abzurufen. Wie eben beschrieben, werden die FOAF-Informationen (nach dem Linked-Data-Prinzip) unter genau der URI veröffentlicht, die sie beschreiben. Damit empfängt der Server bei seiner Anfrage nun einen RDF-Graphen mit FOAF-Informationen. Da der Nutzer diesen zuvor die Schlüssel-Metadaten hinzugefügt hat, kann der Server schließlich die Metadaten aus FOAF mit den Metadaten aus dem

Anfragezertifikat vergleichen. Stimmen sie überein, ist klar, dass die entsprechende Anfrage von der Person kommt, die diese FOAF-Information veröffentlicht hat. Ein Nutzer, der den privaten Schlüssel nicht besitzt und keine Kontrolle über den veröffentlichenden Server hat, kann sich nicht auf diese Weise authentifizieren.

Werden in den für die Bezeichnung von Nutzern verwendeten URLs DNS-Bezeichner anstelle von IP-Adressen verwendet, ist FOAF+SSL auf das im WWW vorhandene zentral verwaltete Domain Name System (DNS) zur Umsetzung von DNS-Bezeichnern wie *www.example.com* in IP-Adressen angewiesen. Gelingt es einem böswilligen WWW-Nutzer, die in DNS verwalteten Informationen zu manipulieren, kann kann die Authentifizierung mittels FOAF+SSL gefälscht werden. Jedoch befindet sich mit DNSSEC ein System bereits teilweise im Einsatz, dass Manipulationen des DNS stark erschwert [Int05].

Damit ist der Nutzer eindeutig identifiziert, und kann mittels seiner URI von Zugriffsrichtlinien erfasst werden. Beispiel 2.11 zeigt einen Ablauf einer Authentifizierung.

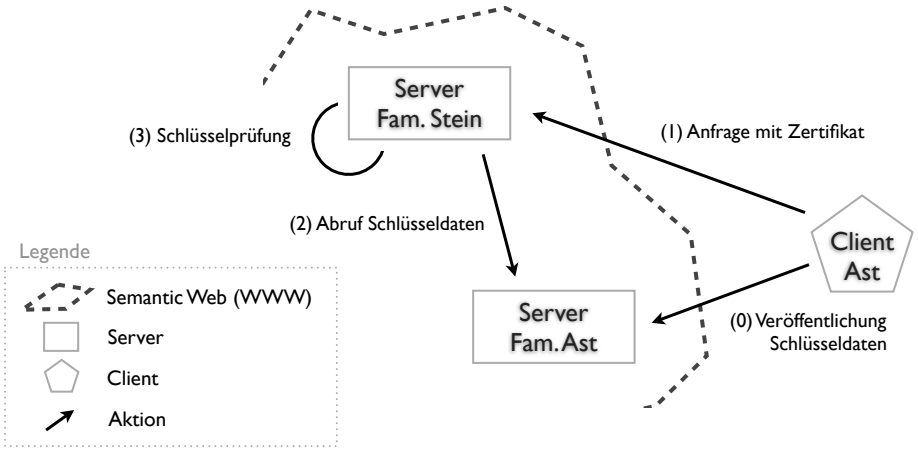

Abbildung 2.4: Authentifizierung mit FOAF+SSL

Beispiel 2.11

In Abbildung 2.4 ist der beispielhafte Ablauf einer Authentifizierung mittels FOAF+SSL grafisch dargestellt: Als Vorbedingung hat Herr Ast in Schritt (0) einen RDF-Graphen auf dem Server der Familie Ast ablegt. Dieser Graph enthält neben FOAF-Informationen auch seine Schlüssel-Metadaten für seinen RSA-Schlüssel in Form von Modulus und Exponent.

Herr Ast möchte sich nun am Server der Familie Stein unter Benutzung seines Bezeichners <`http://example.ast/herrast`> anmelden. Hierzu schickt sein Client eine Anfrage (1) an den Server der Familie Stein. Die Anfrage enthält den öffentlichen Schlüssel von Herrn Ast. Der Server der Familie Stein prüft nun, ob die Anfrage von dem dazugehörigen privaten Schlüssel signiert wurde. Daraufhin wird mittels des Bezeichners

von Herrn Ast der dazugehörige RDF-Graph vom Server der Familie Ast abgerufen (2). Der Server der Familie Stein prüft nun die dort abgerufenen Schlüssel-Metadaten gegen die Informationen aus dem in der Anfrage gesendeten öffentlichen Schlüssel. Stimmen die Angaben überein, ist klar, dass die Anfrage von Herr Ast gekommen ist. Damit ist Herr Ast am Server der Familie Stein eindeutig authentifiziert. In Listing 2.10 ist als Beispiel eine RDF-Serialisierung angegeben, die Metainformationen über den öffentlichen Teil eines RSA-Schlüssels von Herrn Stein enthält. In Listing 2.9 ist das Zertifikat angegeben, das diesen RSA-Schlüssel enthält. In Zeile 7 und 8 sind die in der FOAF-Datei ebenfalls enthaltenen Informationen über Modulus und Exponent (in gekürzter Form) sichtbar. Diese stimmen mit den in Listing 2.10 enthaltenen Informationen überein. In Zeile 11 des Zertifikats ist die URI sichtbar, über die das Dokument abgerufen werden kann. Hierzu wird ein spezielles Feld („Subject Alternative Name“) im Zertifikat genutzt.

```
1  Certificate:
2    Data:
3      Subject: CN=Herr Ast
4      Subject Public Key Info:
5        Public Key Algorithm: rsaEncryption
6          RSA Public Key: (2048 bit)
7            Modulus (2048 bit): 00:cb:50:[...]:53:39
8            Exponent: 65537 (0x10001)
9      X509v3 extensions:
10       X509v3 Subject Alternative Name:
11         URI:http://example.ast/herrast
```

Listing 2.9: Zertifikat mit Referenz auf FOAF-URI (gekürzt)

```
1  <rsa:RSAPublicKey rdf:about="#foafssl_cert1">
2    <cert:identity rdf:resource="http://example.ast/herrast" />
3    <rsa:modulus>
4      <rdf:Description rdf:about="#foafssl_mod1">
5        <cert:hex>00:cb:50:[...]:53:39</cert:hex>
6      </rdf:Description>
7    </rsa:modulus>
8    <rsa:public_exponent>
9      <rdf:Description rdf:about="#foafssl_exp1">
10       <cert:decimal>65537</cert:decimal>
11     </rdf:Description>
12   </rsa:public_exponent>
13 </rsa:RSAPublicKey>
```

Listing 2.10: FOAF-Dokument mit Schlüssel-Metainformationen (gekürzt)

Kapitel 3

Analyse und Anforderungen

Nach der Einführung in die für diese Arbeit wichtigen Grundbegriffe des Semantic Web, der Beschreibung von Zugriffsrichtlinien und dem Überblick über Sicherheit und Privatsphäre wird nun in Abschnitt 3.1 ein Szenario entworfen, für das eine Kombination dieser Technologien sinnvoll ist. Das Szenario beschreibt ein Online-Adressbuch, bei dem die dort enthaltenen Informationen über die Nutzer dieses Adressbuchs auf verschiedenen Systemen verteilt gespeichert sind. Gleichzeitig können die Nutzer kontrollieren, welche anderen Nutzer auf die von ihnen veröffentlichten Daten zugreifen können. Die verschiedenen Anwendungsfälle für diese Anwendung werden aufgezeigt.

Da die Komponenten, die ein im Szenario beschriebenes System möglich machen, auch für andere Problemstellungen verwendbar sind, wird in Abschnitt 3.2 von den konkreten Anforderungen des Szenarios abstrahiert und allgemeine Anforderungen aufgestellt. Diese Anforderungen beschreiben ein generisches System, das verteilte Anwendungen mit Zugriffskontrolle für die darin verwalteten Daten im Kontext von Linked Data möglich macht.

Anschließend werden mit Abschnitt 3.3 die Anforderungen an die zentrale Komponente dieses generischen Systems gezeigt. Bei dieser Komponente handelt es sich um einen Server, der Dienste zur Speicherung und zur Abfrage von RDF-Daten unterstützt. Hierbei soll es möglich sein, die Nutzung dieser Methoden mittels vom Nutzer definierten Richtlinien einzuschränken. Die Verwaltung dieser Richtlinie stellt einen weiteren vom Server im Rahmen seiner Funktionalität angebotenen Dienst dar.

3.1 Beispielszenario „Verteiltes Adressbuch“

Das hier betrachtete Beispielszenario beschreibt die Erstellung eines Systems zur Verwaltung eines verteilten und internetbasierten Adressbuchs. Nutzer können innerhalb dieses Adressbuchs eine Liste ihrer Bekannten verwalten. Die eigentlichen Informationen im Adressbuch werden direkt von Servern abgerufen, die von den jeweiligen Kontakten kontrolliert werden. Dies hat den Vorteil, dass Daten automatisch aktualisiert werden, sowie der jeweilige Kontakt diese ändert.

Gleichzeitig soll es in dem System für die Nutzer möglich sein, Teile ihrer darin ver-

öffentlichten Daten (wie zum Beispiel ihre Telefonnummer oder ihre GPS-Position) so zu markieren, dass diese nur für ausgewählte Kontakte oder Kontaktgruppen sichtbar sind. Die Informationen in diesem Adressbuch sollen maschinenlesbar und durch ein standardisiertes RDF-Vokabular strukturiert sein, somit können die Nutzer unterschiedliche Clients zum Zugriff auf diese Daten nutzen. Als Datenmodell soll RDF zum Einsatz kommen, dies ermöglicht die Informationsintegration zwischen den einzelnen Kontakten mit Hilfe von Linked Data.

Mit bestehenden Onlinediensten ist ein solches Szenario nur teilweise möglich. Der Onlinedienst Plaxo bietet beispielsweise die Erstellung eines Adressbuchs und die Weitergabe ausgewählter Informationen an andere Nutzer. Problematisch jedoch bleibt, dass jegliche Daten auf Servern der Firma gespeichert werden und der Nutzer jenseits sich möglicherweise ändernden allgemeinen Geschäftsbedingungen keine Möglichkeit hat, die Verwendung seiner persönlichen Daten im Sinne von Unterabschnitt 2.1.2 zu kontrollieren.

Das im Folgenden beschriebene Szenario löst diese Probleme: Nutzer kontrollieren den Zugriff auf ihre Daten, und diese werden nicht auf Systemen von Dritten gespeichert. Damit befinden sich die Kontaktdaten zwar jederzeit im Einflussbereich der Nutzer, können jedoch von diesen gleichzeitig für Dritte freigegeben werden. Wichtig ist, dass die Daten nicht mehr kontrolliert werden (können), sobald sie einem berechtigten Kontakt einmal angezeigt wurden.

Unterabschnitt 3.1.1 zeigt nun den grundsätzlichen Aufbau des Systems und anschließend führt Unterabschnitt 3.1.2 die verschiedenen Anwendungsfälle des verteilten Adressbuchs auf.

3.1.1 Systemmodell

Das Systemmodell des verteilten Adressbuchs entspricht dem in Abbildung 2.3 gezeigten Modell: Die Nutzer bedienen sich eines Adressbuch-Clients, um ihre Kontaktdaten zu verwalten und Daten von anderen Kontakten abzurufen. Die Clients veröffentlichen Daten auf einem ihnen zugeordneten Adressbuch-Server und rufen die Daten ihrer Kontakte von mehreren Adressbuch-Servern ab.

Eine Grafische Benutzerschnittstelle (GUI) wird vom Client bereitgestellt. Die die Daten verwaltenden Server befinden sich unter der Kontrolle der Nutzer. Beispielsweise ist es – wie im Bild gezeigt – möglich, dass ein solcher Server für mehrere Mitglieder einer Familie installiert wird, oder dass eine Firma diesen für alle Mitarbeiter der Firma einrichtet. Die für die Funktion entsprechend des Topologiemodells notwendigen Server befinden sich ebenfalls auf Rechnern, die der Kontrolle der Nutzer unterliegen. Im einfachsten Fall sind Client und Server auf einem einzigen Computersystem verfügbar. Die Auftrennung in Client und Server ist deshalb sinnvoll, weil viele Computersysteme keine permanente Internetverbindung haben. Die Integration der Kontaktdaten ist jedoch nur möglich, wenn die Daten jedes Kontakts auch über das WWW erreichbar sind. Es wird daher davon ausgegangen, dass der Server eine permanente Verbindung zum Internet besitzt, für den Client ist dies nicht notwendig.

Zwischen den einzelnen Servern soll keine weitere Verbindung in Form einer Bekanntmachung oder ähnlichen technischen Maßnahmen existieren. Dies unterstützt die Grund-

architektur des WWW. Gleichzeitig macht dieses Konzept es für Nutzer sehr einfach, ihre Systeme dem Gesamtkonzept „Verteiltes Adressbuch“ hinzuzufügen. Alles was hierfür notwendig ist müssen, ist die entsprechenden Softwarepakete für Client und Server auf ihrem System zu installieren.

Für ein solches Systemmodell existieren verschiedene Beispiele, die populäre Blogsoftware *Wordpress* ist etwa ähnlich aufgebaut: Nutzer bedienen sich eines Clients, um mit Hilfe einer vom Server angebotenen Funktionalität Beiträge für das jeweilige Blog auf dem dazugehörigen Server zu veröffentlichen. Blogeinträge können mit Einträgen, die andere Nutzer auf anderen Servern veröffentlicht haben, verknüpft werden, ohne dass zwischen den beteiligten Servern Vorarbeiten notwendig werden. Zugriffsbeschränkungen sind hierbei jedoch nicht vorgesehen.

3.1.2 Anwendungsfälle

Um eine Übersicht darüber zu geben, welche Aktionen die Nutzer des verteilten Adressbuchs in diesem System ausführen können, werden nun die verschiedenen Anwendungsfälle gezeigt. Als allgemeine Vorbedingung wird bei allen Anwendungsfällen vorausgesetzt, dass die Software, die für die Funktion des Adressbuchs notwendig ist, korrekt auf einem Server installiert wurde. Eine beliebige Installation des Adressbuchs wird hier als *Instanz* bezeichnet. In Abbildung 3.1 wird eine Übersicht über die Anwendungsfälle des Beispielszenarios *Verteiltes Adressbuch* als UML-Anwendungsfalldiagramm angegeben. Der Nutzer hat die Möglichkeit zur Veröffentlichung, zum Schutz und zum Abruf von Kontaktinformationen. Als Voraussetzung für die Ausführung der Aktionen ist eine Authentifizierung am System notwendig, um die Identität des Nutzers festzustellen. Damit diese Authentifizierung möglich ist, muss der Nutzer sich zuvor am System angemeldet (registriert) haben.

Anmeldung

Damit das Adressbuch einen Nutzer identifizieren kann, ist eine Anmeldung an der Instanz notwendig. Hierzu werden verschiedene Informationen wie Name, gewünschter Benutzername und Passwort abgefragt, verifiziert und gespeichert. Mittels seines Benutzernamens und seines Passworts kann der Nutzer sich daraufhin an der Instanz authentifizieren. Es ist möglich, die beschriebene Anmeldung durch getrennte Systeme wie etwa einen unternehmensinternen Verzeichnisdienst zu ersetzen, so dass eine dedizierte Nutzerdatenbank in der Instanz nicht notwendig sein muss.

Authentifizierung

Ist ein Nutzer an der Instanz angemeldet, kann eine Authentifizierung durchgeführt werden. Hierzu meldet sich der Nutzer bei der ersten Interaktion mit der Instanz über seinen in der Anmeldung vergebenen Nutzernamen und Passwort an. Ist die Prüfung der angegebenen Daten gegen die Nutzerdatenbank erfolgreich, kann der Nutzer die folgenden Anwendungsfälle aufrufen. Ist diese Prüfung nicht erfolgreich, wird dem Nutzer eine entsprechende Fehlermeldung angezeigt.

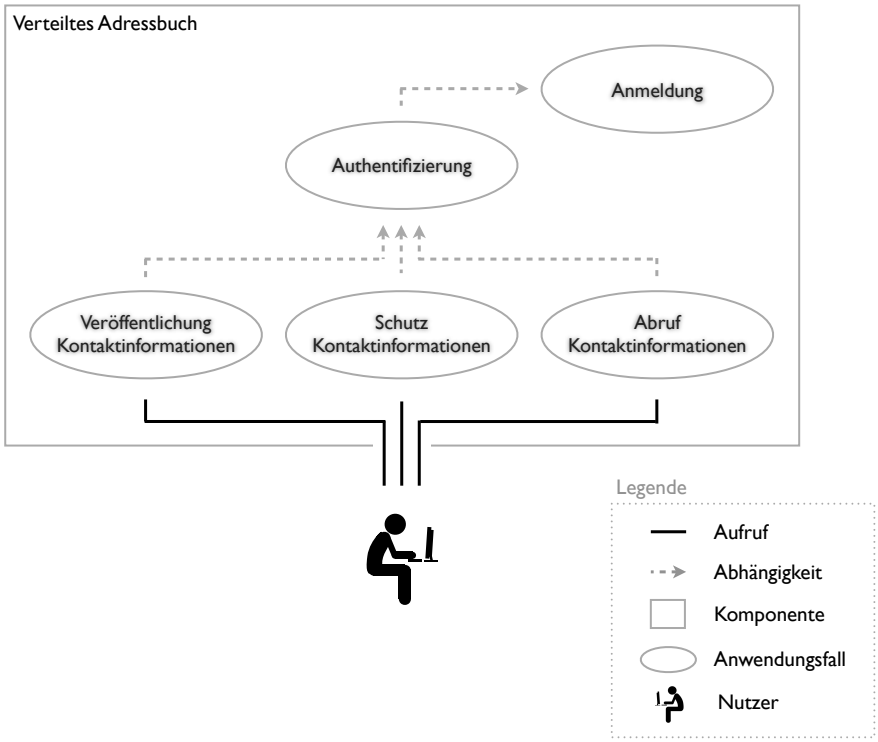

Abbildung 3.1: Anwendungsfälle für das verteilte Adressbuch

Kontakte verwalten

Da ein Adressbuch ohne Kontakte wenig sinnvoll ist, kann der Nutzer seine Kontakte über verschiedene Aktionen verwalten. Kontakte können hinzugefügt und gelöscht werden. Damit der Grundsatz des verteilten Systems gewahrt werden kann, werden Kontakte immer zusammen mit einem Bezeichner für die Instanz, an der sie sich angemeldet haben, identifiziert. URIs können hierfür benutzt werden, beispielsweise könnte die URI <`http://famstein.example/personen/horststein`> den Nutzer *horststein* beschreiben, der an der Instanz, die auf dem Rechner mit dem DNS-Namen *famstein.example* installiert ist, angemeldet ist. Ist ein Kontakt einmal hinzugefügt, versucht die Instanz, einen besser von Menschen lesbaren Kontaktnamen zu erstellen. Hierzu kann die Instanz des hinzugefügten Kontakts um einen solchen Bezeichner gebeten werden. Die Instanz verwaltet für jeden Nutzer eine Liste seiner Kontakte, die ausschließlich deren Bezeichner enthält, um den Grundsatz der verteilten Datenhaltung durchzusetzen.

Kontaktgruppen verwalten

Nutzer können ihre Kontakte verschiedenen Gruppen zuweisen. Ein Kontakt kann sich in mehreren Gruppen befinden. Es ist nicht notwendig, dass ein Kontakt sich in einer Gruppe befindet. Für die Verwaltung der Gruppen können Gruppen angelegt werden. Ist eine Gruppe angelegt, können dieser Kontakte hinzugefügt und aus dieser entfernt werden. Gruppen können ebenfalls gelöscht werden, dabei wird jedoch keiner der darin enthaltenen Kontakte gelöscht.

Veröffentlichung von Kontaktinformationen

Ein Nutzer kann in der Instanz beliebige Kontaktdaten über sich veröffentlichen. Beispiele für Kontaktdaten sind Verweise auf Kommunikationsmittel, Adressen, Standort, Interessen, Fotos, aktuelle Projekte und so weiter. Das verteilte Adressbuch schlägt verschiedene Informationstypen vor, diese können jedoch vom Nutzer erweitert werden. Die eingegebenen Kontaktinformationen werden von der Instanz dauerhaft gespeichert. Der Nutzer kann diese jederzeit verändern. Eine Löschung von einzelnen Einträgen in die Gesamtheit der Kontaktinformationen ist ebenfalls möglich.

Schutz von Kontaktinformationen

Damit nicht jede Kontaktinformation jedem beliebigen Nutzer des WWW zur Verfügung steht, kann der Nutzer für jedes einzelne Informationselement (wie z.B. eine Telefonnummer) festlegen, für welche seiner Kontakte oder welche Kontaktgruppen diese Information sichtbar sein soll. Es ist zudem möglich, den eigenen Adressbucheintrag in seiner Gesamtheit zu schützen. Hierzu bietet das verteilte Adressbuch eine Eingabemöglichkeit für die Zugriffsrichtlinien auf die Nutzerdaten. Diese Richtlinien können jederzeit bearbeitet werden. Es wird dem Nutzer auch erlaubt, keinerlei Einschränkungen zu treffen. In so einem Fall werden alle der Instanz bekannten Informationen an anfragende Clients ausgeliefert.

Abruf von Kontaktinformationen

Ein Nutzer kann nach dem Hinzufügen von Nutzern zu seiner Kontaktliste dessen Kontaktinformationen abrufen. Die angezeigten Informationen über den Kontakt hängen davon ab, welche Richtlinien zum Schutze seiner Informationen der angezeigte Kontakt angegeben hat.

3.2 Verallgemeinerung der Anforderungen

Das in Abschnitt 3.1 beschrieben System könnte als solches eigenständig entworfen und umgesetzt werden. Es ist jedoch auffallend, dass die an dieses System gestellten Anforderungen und die daraus resultierende Funktionalität in einem Aspekt generisch ist: Das Problem der Veröffentlichung von RDF-Daten auf einem Linked-Data-Server sowie die Einschränkung des Abrufs dieser Daten durch Dritte ist einer Vielzahl von Programmen gemein, beispielsweise würde ein verteiltes soziales Netzwerk ähnliche Funktionen erfordern. Daher werden

nun aus den im Szenario genannten Anforderungen und Komponenten generische Anforderungen entwickelt. Stehen nach diesen generischen Anforderungen entwickelte Systeme zur Verfügung, kann die im Szenario entworfene Anwendung sehr viel leichter implementiert werden. Gleichzeitig können diese Systeme für beliebige durchaus unterschiedliche Anwendungen genutzt werden. Im Sinne der Wiederverwendbarkeit von Komponenten und der Auslagerung von generischer Funktionalität in Bibliotheken ist dieser Ansatz sinnvoll.

Die folgenden Abschnitte beschreiben in Unterabschnitt 3.2.1 die allgemeinen Anforderungen an den Client sowie in Unterabschnitt 3.2.2 die Anforderungen an einen Server, der einen großen Teil der für das verteilte Adressbuch notwendigen Funktionalität bietet. Anzumerken bleibt, dass Komponenten wie etwa die GUI des verteilten Adressbuchs sich nicht generisch implementieren lassen, so dass diese hier nicht weiter vertieft werden. Für die abstrakte Funktionalität werden daher nur die der im vorherigen Abschnitt beschriebenen Anwendungsfälle betrachtet, die Kommunikation zwischen Client und Server erfordern:

- Veröffentlichung von Kontaktinformationen (Abschnitt 3.1.2)
- Schutz von Kontaktinformationen (Abschnitt 3.1.2)
- Abruf von Kontaktinformationen (Abschnitt 3.1.2)

Da das Szenario die Möglichkeiten des Semantic Web nutzen und sich möglich nahtlos in das in den Grundlagenkapiteln eingeführte Linked-Data-Umfeld [BHIBL08, Wor09] einfügen soll, kommt RDF als Datenmodell zum Einsatz. Hierbei ist besonders die Nutzung von dereferenzierbaren URIs als Bezeichner für Elemente im RDF-Graphen wie in Unterabschnitt 2.2.3 beschrieben wichtig. Das abstrahierte Konzept soll die Veröffentlichung von beliebigen Daten, Authentifizierung sowie die Vergabe und Durchsetzung von Richtlinien innerhalb eines Linked-Data-Servers möglich machen.

Benötigt wird ein System aus einem Server, die die Durchsetzung von Richtlinien und die darauf folgende Auslieferung von RDF-Daten geeignet übernimmt, sowie aus einem Client, die den Nutzer bei der Verwaltung seiner Daten und der Zugriffsrichtlinien für diese Daten unterstützt.

3.2.1 Client

Der Client benötigt eine Möglichkeit, Daten an Server zu senden und Daten von einem Server zu empfangen: Für die Veröffentlichung sowie den Schutz von Kontaktinformationen werden die entsprechenden Daten sowie Zugriffsrichtlinien an einen Server gesendet. Beim Abruf von Kontaktdaten muss der Client eine Möglichkeit haben, sich bei dem Server, bei dem die Daten gespeichert sind, im Namen seines Nutzers sicher zu identifizieren, da sonst der angefragte Server keine Möglichkeit hat, die vergebenen Richtlinien zu prüfen und über eine Herausgabe von Daten zu entscheiden. Speziell bei der Veröffentlichung von Daten muss der Client sicherstellen können, dass der Server die angegebenen Zugriffsrichtlinien auch korrekt umsetzt, da es andernfalls vorkommen kann, dass Daten gegen den in der Zugriffsrichtlinie erklärten Willen des Nutzers zugänglich gemacht werden. Implementiert eine generische Bibliothek die genannten Funktionen, kann der Teil des Clients,

der die Kommunikation mit dem Server übernimmt, durch sie bereitgestellt werden. Die Funktionen des Clients im verteilten Adressbuchs beschränkt sich daraufhin auf Darstellung einer grafischen Oberfläche sowie der Verarbeitung der Nutzereingaben und einer Nutzerverwaltung. Dies reduziert die Komplexität der Komponente stark. Die genannten Anforderungen lassen sich durch bestehende Komponenten wie zum Beispiel ARC [Now09] umsetzen. Daher wird der Client im folgenden Teil nicht weiter betrachtet.

3.2.2 Server

Nach den identifizierten Anwendungsfällen benötigt ein generischer Server Funktionen für die Datenspeicherung der an ihn gesendeten Daten. Gleichzeitig werden die Zugriffsrichtlinien in geeigneter Form gespeichert. Zusätzlich ist es möglich, die angegebenen Richtlinien auf Korrektheit zu prüfen und das Ergebnis der Prüfung dem Client mitzuteilen. Die Herausgabe von Daten an Dritte erfordert eine Komponente, die den anfragenden Nutzer oder den vom Nutzer verwendeten Client eindeutig identifiziert. Ist diese Identifizierung erfolgreich, wertet der Server die für die angefragten Daten relevanten Zugriffsrichtlinien aus und liefert die angefragten Daten bei einem positiven Ergebnis der Auswertung an den anfragenden Client aus. Für jeden Client, der Daten auf dem Server veröffentlicht, wird eine eigene Datenbasis geschaffen, um die veröffentlichten Daten eindeutig einem Regelsatz zuweisen zu können. Steht ein solcher Server zur Verfügung, erübrigt sich die Entwicklung einer dedizierten Serverkomponente für das verteilte Adressbuch. Dies reduziert die Komplexität der gesamten Anwendung „verteiltes Adressbuch“ stark.

3.2.3 Linked Data - Integration

Die für Client und Server genannten generischen Anforderungen können zu einem Teil durch Technologien des Semantic Web abgedeckt werden: Mit Hilfe der Prinzipien von Linked Data (Siehe dazu Unterabschnitt 2.2.3) ist die Veröffentlichung und Integration von beliebigen Daten über Organisationsgrenzen leicht möglich. Der Zugriff auf die Daten durch verschiedene Programme ist durch standardisierte Protokolle und Formate ebenfalls durchführbar. Aus diesen Gründen ist es sinnvoll, für die genannten Komponenten auf Basistechnologien des Semantic Web aufzubauen. Jedoch ist die Forderung nach der Kontrolle der Auslieferung von veröffentlichte Daten an ausgewählte Kontakte zum Zeitpunkt der Erstellung dieser Arbeit im Kontext von Linked Data nicht möglich. Hierfür wird eine Komponente benötigt, die nach der Identifikation der Nutzer die Formulierung und Durchsetzung von Zugriffsrichtlinien auf Linked-Data-Informationen ermöglicht. Die Verwendung von bestehenden Programmbibliotheken vereinfacht die Entwicklung der Datenhaltung und der Kommunikation stark. Für die bisher nicht vorgesehene Durchsetzung von Zugriffsrichtlinien sowie für die Authentifizierung der Nutzer ist jedoch eine Erweiterung dieser Technologien notwendig.

3.3 Anforderungen an die Serverkomponente

Die Notwendigkeit einer Serverkomponente wurde im Systemmodell des verteilten Adressbuchs beschrieben. Nachdem die generische Anforderungen der Komponente identifiziert wurden wird nun die gewünschte Funktionalität beschrieben. Die hier beschriebene Komponente stellt ein System zur Speicherung und für den Abruf von RDF-Daten dar, in dem der Zugriff auf die Daten von vom Nutzer zu vergebenden Zugriffsrichtlinien abhängig gemacht werden kann. Im Hinblick auf die allgemeine Verwendbarkeit für ähnliche Szenarien werden weitere Anforderungen an den generischen Server beschrieben. Unterabschnitt 3.3.1 beschreibt die funktionalen Anforderungen. Daraufhin zeigt Unterabschnitt 3.3.2 die Anforderungen an die Zugriffsrichtlinien. Anschließend werden in Unterabschnitt 3.3.3 die Anforderungen an Formate und Protokolle und in Unterabschnitt 3.3.4 sonstige Anforderungen genannt.

3.3.1 Funktionale Anforderungen

Hier werden die im Einzelnen notwendigen Dienste beschrieben, die ein Server bereitstellen muss, um ein dem Szenario entsprechendes Konzept unterstützen zu können. Nach dem allgemeinen Fall der Authentifizierung werden in den anschließenden Unterabschnitten die Verwaltung der Zugriffsrichtlinien sowie verändernde Operationen auf dem Datenbestand des Servers beschrieben. Diese verändernden Operationen umfassen Veröffentlichung, Veränderung und Löschung von RDF-Daten. Abschließend wird in Abschnitt 3.3.1 der Ablauf einer Abfrage auf durch mit Zugriffsrichtlinien geschützte Daten erklärt. Eine Übersicht der Anwendungsfälle ist in Abbildung 3.2 als UML-Anwendungsfalldiagramm dargestellt. Auf diesen einfachen Operationen aufbauend können komplexe Zugriffsmuster realisiert werden. Grundsätzlich soll der Server mehrbenutzerfähig sein, das bedeutet, dass mehrere Nutzer gleichzeitig und voneinander unabhängig die Funktionalität des Servers in Anspruch nehmen können. Jeder Nutzer kann mehrere RDF-Graphen auf einem Server veröffentlichen. Der Nutzer, der eine Menge von Daten (auch als *Datensatz* bezeichnet) auf einem Server veröffentlicht, gilt als dessen *Besitzer*. Nur dieser Besitzer ist berechtigt, für diese Daten gültige Richtlinien und Daten zu verändern.

Authentifizierung

Entsprechend der identifizierten Anforderungen an ein sicheres Computersystem ist eine Authentifizierung der die Funktionalität des Servers über einen Client in Anspruch nehmenden Nutzer notwendig. Dies soll ohne Eingabe von Benutzername und Passwort vonstatten, da sonst ein manueller Eingriff des Nutzers notwendig wäre. Die Authentifizierung soll mit Hilfe von in Abschnitt 2.1 beschriebenen elektronischen Zertifikaten umgesetzt werden: Bei einem Verbindungsaufbau wird sowohl von Server als auch vom Client ein digitales Zertifikat der Anfrage beigefügt, anhand dessen es beiden Seiten möglich ist, über die Identität der Gegenstelle Gewissheit zu erlangen. Alle daran anschließende Kommunikation soll dann über einen verschlüsselten Kanal ab. Die Authentifizierung ist optional, es sind kann vom Nutzer durchaus gewünscht werden, Daten auch unbeschränkt zu veröffentlichen. In solch

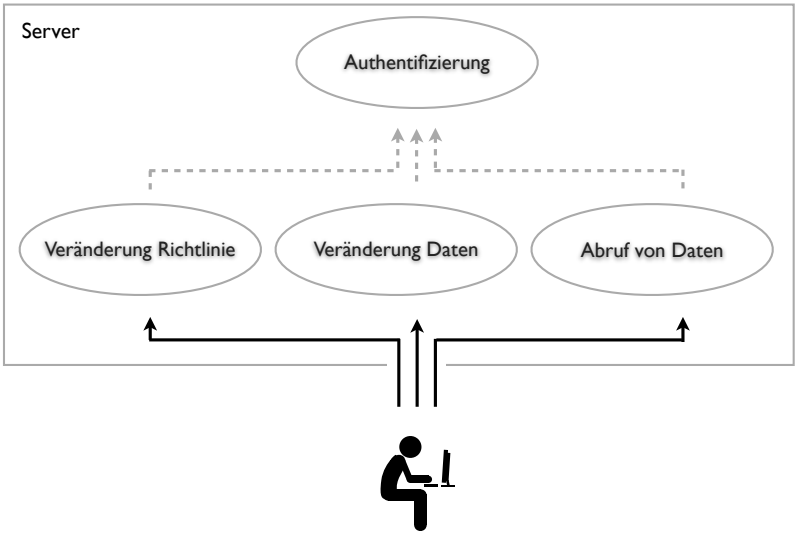

Abbildung 3.2: Anwendungsfälle: Server

einem Fall ist die Authentifizierung nicht notwendig. Gleichzeitig ist so eine Kompatibilität zu bestehenden Linked-Data-Systemen möglich.

Veränderung von Richtlinien

Ein Nutzer kann mittels eines Clients eine Zugriffsrichtlinie für die Datensätze, als deren Besitzer er eingetragen ist, auf einem Server festlegen. Dieser Ablauf ist in Abbildung 3.3 als UML-Kommunikationsdiagramm angegeben:

Der Nutzer drückt mit Hilfe des Clients eine Zugriffsrichtlinie für seine Daten aus(1). Der Client stellt eine Verbindung zum Server her, und führt zunächst im Namen des Nutzers eine Authentifizierung wie in Abschnitt 3.3.1 durch (2). Daraufhin sendet der Client über den nun verschlüsselten Kanal die Zugriffsrichtlinie in einem maschinenlesbaren Format an den Server (3). Der Server führt eine Prüfung Zugriffsrichtlinie durch (4), speichert die Richtlinie bei erfolgreicher Prüfung, und sendet eine Meldung an den Client (5). Der Client kann nun gegebenenfalls die Speicherung der Zugriffsrichtlinie dem Nutzer in der grafischen Oberfläche bestätigen (6). Die getrennte Übertragung von Richtlinien und Daten ermöglicht dem Client einen Abbruch der Interaktion mit dem Server, sollte dieser die angegebene Richtlinie nicht erfolgreich ablegen können. Beispiel 3.1 zeigt eine Veröffentlichung nach dem hier beschriebenen Prinzip.

Beispiel 3.1

Der Nutzer „Stein“ benutzt sein Adressbuch-Programm (den Client), um seine Telefonnummer auf einem Server zu veröffentlichen. Er möchte diese jedoch nur dem Nutzer „Ast“ zur Verfügung stellen. Sein Client drückt diese Zugriffsrichtlinie in einer für seinen

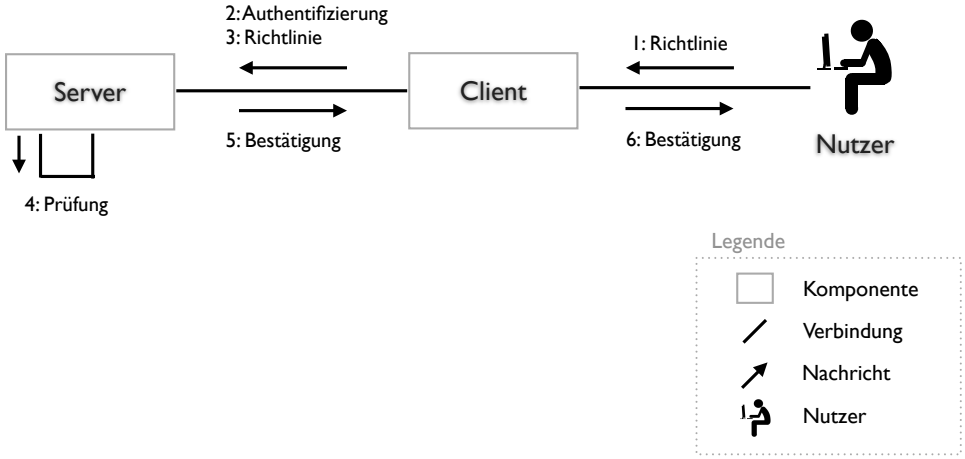

Abbildung 3.3: Anwendungsfall: Veränderung von Richtlinien

Linked-Data-Server interpretierbaren Sprache zur Formulierung von Zugriffsrichtlinien aus. Der Client sendet die Zugriffsrichtlinie für den Datensatz, für den „Stein“ als Besitzer eingetragen ist, an den Server, der Server prüft und speichert die Richtlinie. Der Server sendet eine Bestätigung für die erfolgreiche Speicherung der Richtlinie an den Client. Der Client kann den diese Bestätigung der Operation grafisch darstellen.

Veränderung von Daten

Daten können auf einem Server veröffentlicht, verändert und gelöscht werden. Wird ein Datensatz erstmals auf einem Server veröffentlicht, wird der Nutzer, in dessen Auftrag diese Daten veröffentlicht werden, als Besitzer des entsprechenden Datensatzes eingetragen. Veränderung und Löschung sind daraufhin nur für den Besitzer des jeweiligen Datensatzes zulässig. Der entsprechende Ablauf wird in Abbildung 3.4 illustriert: Der Nutzer formuliert mit Hilfe seines Clients die zu veröffentlichenden Daten in einem maschinenlesbaren Format (1). Der Client baut eine Verbindung zum Server auf, daraufhin wird eine Authentifizierung wie in Abschnitt 3.3.1 durchgeführt (2). Der Client sendet nun die Daten an den Server, dieser prüft und speichert diese Daten(4). Der Server sendet eine Bestätigung an den Client (5), der Client kann diese dem Nutzer anzeigen (6).

Beispiel 3.2

Der Nutzer „Stein“ benutzt sein Adressbuch-Programm, um seine Telefonnummer zu veröffentlichen. Er sendet hierzu die entsprechenden Daten mittels seines Adressbuch-Programmes an seinen Server. Dieser speichert die Telefonnummer ab und sendet eine Erfolgsmeldung an den Client.

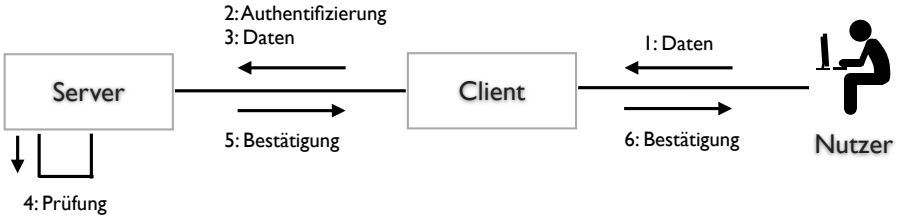

Abbildung 3.4: Anwendungsfall: Veränderung von Daten

Abruf von Daten

Ein Nutzer kann seinen Client verwenden, um Daten von einem Server abfragen. Dies wird in Abbildung 3.5 gezeigt. Der Nutzer formuliert seine Anfrage mit Hilfe des Clients (1). Der Client führt daraufhin im Namen des Nutzers eine wie in Abschnitt 3.3.1 beschriebene Authentifizierung durch (2). Daraufhin stellt der Client die Anfrage des Nutzers in einer geeigneten Anfragesprache eine Anfrage für den Abruf von Daten an den Server (3). Der Server prüft diese Anfrage anhand der für diesen Datensatz gültigen Zugriffsrichtlinie und entscheidet, ob und welche Daten dem authentifizierten Nutzer zur Verfügung gestellt werden (4). Der Server übermittelt diese autorisierten Daten daraufhin an den Client (5), dieser stellt sie gegebenenfalls dem Nutzer in lesbarer Form dar (6). Dem Besitzer eines Datensatzes ist der Abruf der von ihm veröffentlichten Daten ohne Beachtung der Zugriffsrichtlinie möglich.

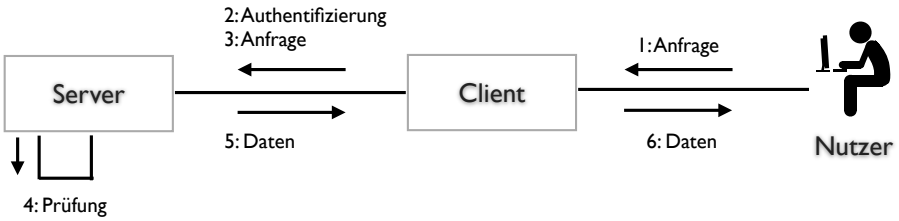

Abbildung 3.5: Anwendungsfall: Abruf von Daten

Beispiel 3.3

Der Nutzer „Ast“ benutzt sein Adressbuch-Programm, um die Telefonnummer des Nutzers „Stein“ herauszufinden. Da Nutzer „Stein“ den Nutzer „Ast“ autorisiert hat, diese abzurufen, kann der Client von Nutzer „Ast“ diese abrufen und dem Nutzer anzeigen.

3.3.2 Regelformat und -auswertung

Für jeden auf dem Server verfügbaren Graphen kann eine eigene Zugriffsrichtlinie definiert werden. Zugriffsrichtlinien bestehen aus einer oder mehreren Regeln. Diese Regeln bestehen aus zwei Teilen, Voraussetzung und Konsequenz. Die Voraussetzung ist eine prädikatenlogische Aussage über den angefragten RDF-Graphen sowie über den anfragenden Nutzer. Die Konsequenz einer Regel beschreibt mittels der im folgenden Abschnitt beschriebenen Datenklassifikation, welche Teile des auf dem Server abgelegten RDF-Graphen dem authentifizierten Nutzer zugänglich gemacht werden. Alternativ ist es möglich, die Elemente auszuwählen, die dem Nutzer *nicht* zugänglich gemacht werden sollen.

Der Server wertet bei jeder Anfrage die für den entsprechenden Graphen passende Zugriffsrichtlinie aus, stellt fest, für welche Regeln die Voraussetzungen erfüllt sind und gibt die in den entsprechenden Konsequenzen genannten Elemente des Graphen frei. Richtlinien können außer der Beschränkung des Zugriffs von Daten genauso die Veränderung von Daten betreffen. Regeln können über ein unabhängiges Protokoll für Graphen festgelegt und verändert werden. Diese Veränderung oder Festlegung ist nur dem Nutzer möglich, der einen Graphen angelegt hat. Werden Regeln definiert, bevor der von ihnen betroffene Graph angelegt wurde, wird ein leerer Graph angelegt. Ist die Zugriffsrichtlinie syntaktisch oder semantisch fehlerhaft, wird ein Fehler ausgegeben und die fehlerhafte Richtlinie nicht gespeichert.

Datenklassifikation

Den in Unterabschnitt 2.3.2 beschriebenen Klassifikationsebenen folgend werden die dort genannten Möglichkeiten für Klassifikationen innerhalb des Datenmodells umgesetzt. Informationen können im Falle des Datenmodells RDF auf der Modellebene, auf der semantischen Ebene sowie auf der Ressourcenebene klassifiziert werden. Hierfür stellt das Regelformat Möglichkeiten bereit, Informationen auf der Modellebene über Tripelmuster zu klassifizieren. Auf der semantischen Ebene können Elemente nach ihrer Instantiierung eines RDF- oder OWL-Konzepts identifiziert werden. Schließlich können Elemente auch über ihre URI auf der Ressourcenebene benannt werden. Die verschiedenen Klassifikationsmöglichkeiten können innerhalb von Richtlinien kombiniert werden.

Prioritäten

Werden mehrere Regeln innerhalb einer Richtlinie definiert, können bei der Auswertung dieser Regeln durchaus Konflikte auftreten, beispielsweise könnte eine Regel den Zugriff auf ein Datenobjekt erlauben, während eine andere Regel den Zugriff auf das selbe Datenobjekt verbietet. Es ist möglich, solche Fälle über eine Vergabe von Prioritäten für Regeln zu behandeln. Dies ist jedoch nicht wünschenswert, da dadurch die Komplexität der Regeln stark steigt. Sinnvoller ist es, eine systemweite Regelung festzulegen, die etwa bei solchen Konflikten den Zugriff verbietet. Dadurch sinkt die Komplexität der Zugriffsrichtlinien, eine Eigenschaft, die im Sinne einer möglichst einfachen Nutzbarkeit des Systems erwünscht ist.

Negation

Innerhalb der Prädikatenlogik ist eine Negation möglich, die die Aussage der negierten Prädikate oder -Gruppen umkehrt. Eine Unterstützung von Negation durch das Regelformat ist möglich, jedoch aus Gründen der Systemleistung nicht unbedingt wünschenswert: Wählt ein Prädikat eine Menge von Datenobjekten aus, und wird dieses verneint, betrifft das verneinte Prädikat alle übrigen verfügbaren Datenobjekte. Aus diesem Grund ist eine Negation beispielsweise in der in Abschnitt 2.3.3 vorgestellten Regelsprache SWRL nicht vorgesehen.

3.3.3 Datenformat und -zugriff

Unabhängig von der internen Darstellung von Daten werden nun die Protokolle und Datenformate beschrieben, die der Server in seiner externen Kommunikation einsetzt. Die Verwendung eines einfachen und verbreiteten Protokolls sowie verbreiteter Datenformate ist wünschenswert, da so die Nutzung des Dienstes durch Programme sowie seine Integration mit fremder Software möglich ist. Ein einfaches Protokoll zeichnet sich beispielsweise dadurch aus, dass nur einfache und zeitlich begrenzte Operationen unterstützt werden. Ein Beispiel für ein einfaches Protokoll stellt HTTP dar, das darauf aufbauende Entwurfsmuster des Representational State Transfer (REST)[Fie00] gibt noch weitere Vereinfachungen vor. Hierbei müssen keinerlei Statusinformationen über die Kommunikation zwischen den Systemen vorgehalten werden. Dies hat den Vorteil, dass eine Nutzung des Server-Dienstes durch Programme einfach möglich ist. Zusätzlich wird ein Vorteil in der Systemleistung erreicht.

Datenmodell und -format

Als Datenmodell wird RDF wie in Unterabschnitt 2.2.1 beschrieben verwendet, dass in einer dafür geeigneten Form persistent auf einem Sekundärspeicher des Servers abgelegt und geladen werden kann. Für jeden Nutzer, der Daten auf dem Server im Sinne von Abschnitt 3.3.1 veröffentlicht, können beliebig viele RDF-Graphen erzeugt werden. Diesen Graphen wird ein Bezeichner zugewiesen (engl. „Named Graph“), der diesen eindeutig identifiziert. Bei jeglicher Aktion, die auf dem Server ausgeführt wird, muss dieser Bezeichner dem Server bekannt sein. Als Bezeichner für Graphen bietet sich das URI-Format an. RDF- Graphen werden über eine spezielle Anfragesprache erzeugt, manipuliert und angelegt. Eine weitere Anfragesprache wird verwendet, um Anfragen an den Server zu stellen. Die Antworten auf diese Anfragen enthalten dann die ausgewählten Graphenelemente in einer passenden Serialisierung, wird beispielsweise ein gesamter Graph ausgegeben, kann das RDF/XML-Format verwendet werden.

Datenveröffentlichung und -veränderung

Der Server bietet eine Schnittstelle für die Erstellung, Manipulation und Entfernung von Graphen. Hierbei wird für einen definierten Graphen in einer Anfrage festgelegt, welche Tripel hinzugefügt, verändert oder gelöscht werden. Das Protokoll hierfür ist einfach: Der

Client sendet eine Anfrage an den Server, die einen Verweis auf den betreffenden Graphen und die Veränderungsanforderung in der Anfragesprachen enthält. Existiert ein Graph dieses Namens noch nicht, wird er angelegt. Existiert ein Graph dieses Namens, wird geprüft, ob der anfragende Client berechtigt ist, diesen Graphen zu bearbeiten. Hierzu werden - falls vorhanden - entsprechende Regeln für die Datenveränderung ausgewertet. Sind keine Regeln für Datenveränderung definiert, wird diese Veränderung nur dem Nutzer erlaubt, der den entsprechenden Graphen erstmalig erstellt hat. Ergibt die Prüfung der Berechtigungen ein positives Ergebnis und ist die angegebene Veränderungsanfrage syntaktisch korrekt, wird sie ausgeführt und der veränderte Graph wieder persistent gespeichert. Üblicherweise sollte vor der ersten Veröffentlichung von Daten in einem Graph eine Zugriffsrichtlinie für diesen Graph definiert werden. Die Veröffentlichung der Daten kann so von der erfolgreichen Definition der Zugriffsrichtlinie auf dem Server abhängig gemacht werden.

Ein Beispiel für die hier beschriebene Sprache zur Beschreibung von Änderungen eines RDF-Graphen stellt die Sprache SPARQL/Update [SMB+08] dar.

Anfragesprache und -Protokoll

Der Server stellt eine Schnittstelle für die Abfrage für Teilmengen der veröffentlichten RDF-Graphen bereit. Das hierfür verwendete zweistufige Protokoll läuft wie folgt ab: Der Client stellt eine Anfrage an den Server. Diese enthält einen Verweis auf den angefragten RDF-Graphen sowie eine Abgrenzung des gewünschten Teilgraphen. Existiert ein Graph mit dem angegebenen Namen, wird die entsprechende Zugriffsrichtlinie – so vorhanden – ausgewertet. Hierdurch entsteht ein temporärer Graph, der die Elemente enthält, für die der aktuelle Client eine Berechtigung besitzt. Auf diesem temporären Graphen wird die Anfrage des Clients nun ausgeführt und das Ergebnis an den Client gesendet.

Ein Beispiel für die genannte Sprache zur Abgrenzung von Teilen eines RDF-Graphen ist die Sprache SPARQL, die in Unterabschnitt 2.2.4 beschreiben wird.

3.3.4 Sonstige Anforderungen

Im Folgenden werden weitere, von konkreter Funktionalität unabhängige Anforderungen formuliert.

Sicherheit

Der Zugriff auf die von den Nutzern veröffentlichten Daten ist nicht möglich, ohne dass die entsprechenden Zugriffsrichtlinien ausgewertet und durchgesetzt werden. Zugriffsrichtlinien werden nicht veröffentlicht.

Externe Schnittstellen

Die Serverkomponente bietet drei Schnittstellen an: Eine Schnittstelle für die Abfrage von Daten, eine Schnittstelle für die Veränderung von Daten und schließlich eine Schnittstelle zur Definition und Veränderung von Zugriffsrichtlinien. Davon abgesehen sind keine weiteren Schnittstellen notwendig. Die Administration des Systems findet nicht während des

laufenden Betriebs statt. Konfigurationsparameter können vor dem Start des Servers in einer Konfigurationsdatei festgelegt werden. Statusinformationen werden in eine Logdatei geschrieben.

Leistungsverhalten

Die Zeit, die der Server für die Bearbeitung einer Anfrage an einen der angebotenen Dienste benötigt, von der Anzahl der im angefragten Graphen abgelegten Tripel und von der Anzahl der in der dazugehörigen Zugriffsrichtlinie enthaltenen Regeln ab. Bei einem Zugriff müssen alle Regeln ausgewertet und für jedes Tripel entschieden werden, ob es dem anfragenden Nutzer verfügbar gemacht wird. Wünschenswert ist eine angemessene Skalierung des Aufwands für eine Anfrage nach der Menge der von ihr betroffenen Tripel und ausgewerteter Richtlinien. Gleichzeitig soll es möglich sein, je nach verfügbaren Systemressourcen Anfragen mehrerer Nutzer gleichzeitig zu beantworten, ohne dass es dadurch zu Leistungseinbrüchen kommt. Der Server soll für die Beantwortung interaktiver Anfragen aus einer grafischen Oberfläche heraus eingesetzt werden können. Damit sollte seine Antwortzeit in den meisten Fällen weniger als eine Sekunde [DWT82] betragen.

Kapitel 4

Entwurf „Policy enabled Linked Data Server“

In Kapitel 3 wurden allgemeine Anforderungen an System zur Veröffentlichung und zum Abruf von RDF-Daten unter dem Schutz von Zugriffsrichtlinien mit passwortloser Authentifizierung angegeben. Der im Folgenden angegebene Entwurf beschreibt unter dem Namen Policy enabled Linked Data Server (PeLDS) eine mögliche Umsetzung dieser Anforderungen. Für diese Umsetzung werden in diesem Kapitel Datenformate, Zugriffsprotokolle und Anfrageformate spezifiziert.

Bei PeLDS handelt es sich um einen Linked-Data-Server. Im Unterschied zu bestehenden Lösungen wie etwa dem verbreiteten Serverprogramm Joseki unterstützt PeLDS die Vergabe von Zugriffsrichtlinien, die den Datenzugriff einschränken. Hierzu enthält PeLDS zusätzlich zu RDF-Speicher und entsprechenden Zugriffsmethoden auch Komponenten, die Nutzer identifizieren und die Richtlinien auswerten und durchsetzen. Ein System mit dieser Funktionalität wurde bislang in dieser Form nicht veröffentlicht, obwohl eine Reihe von Szenarien denkbar sind, in denen ein solches System erforderlich ist. Nutzer können mittels eines PeLDS-Systems RDF-Graphen veröffentlichen und mit Zugriffsrichtlinien versehen. Auf einer Installation können beliebig viele Nutzer beliebig viele Graphen veröffentlichen. Das Softwareprodukt PeLDS umfasst einen eigenständig lauffähigen Linked-Data-Server als ausführbares Programm. Ebenfalls mitgeliefert werden der Programmcode mit seiner integrierten Dokumentation sowie ein Installationshandbuch.

Die folgenden Abschnitte gliedern sich wie folgt: Die Systemübersicht in Abschnitt 4.1 untersucht nach einer Darstellung der Komponenten, welche Einschränkungen für diesen Entwurf gelten. Danach beschreibt Abschnitt 4.3 die von PeLDS angebotenen Dienste aus Nutzersicht. Diese Dienste übertragen die in Abschnitt 3.3 erhobenen notwendigen Funktionen in konkretere Formate und Protokolle. Darauf folgend beschreibt Abschnitt 4.4 die identifizierten notwendigen Komponenten aus Entwicklersicht und die Funktionalität, die die einzelnen Module anbieten. Schließlich führt Abschnitt 4.2 in das neu entworfene Richtlinienformat in einer Prolog-ähnlichen Syntax unter der Bezeichnung PsSF ein, dessen Neuentwicklung notwendig wurde.

4.1 Systemübersicht

In Abbildung 4.1 ist eine Übersicht über das Gesamtsystem und dessen Komponenten abgebildet. Das System unterstützt zwei grundlegende Prozesse: Daten können auf einem Server veröffentlicht und von diesem auch wieder abgerufen werden.

Für die Veröffentlichung von Daten wird der Nutzer durch die grafische Benutzeroberfläche *GUI* unterstützt. Mittels der Komponente *Semantic Web Publish* können diese Daten an einen vertrauenswürdigen Server gesendet werden. Über die Zugriffsschnittstelle *API* wird der Benutzer mittels einer weiteren Komponente *Auth* authentifiziert und die entsprechende Richtlinien mit *Rules* ausgewertet. Schließlich werden die Daten in der Datenbank *Store* abgelegt. Die angesprochene Zugriffsrichtlinie wurde in einem untergeordneten Prozess veröffentlicht.

Werden Daten von einem oder mehreren Servern abgerufen, veranlasst der Nutzer dies wiederum über die grafische Benutzeroberfläche, daraufhin wird die Komponente zur Abfrage von Daten des Semantic Web *Semantic Web Client* benutzt, um von einem oder mehreren Servern Daten abzurufen. Im Server wird der eben beschriebene Ablauf mit Zugriffsschnittstelle, Authentifizierung und Richtlinienprüfung in ähnlicher Form durchgeführt. Ergibt die Auswertung der in der Richtlinie enthaltenen Regeln ein positives Resultat, werden dem anfragenden Nutzer die entsprechenden Informationen aus der Datenbank übermittelt.

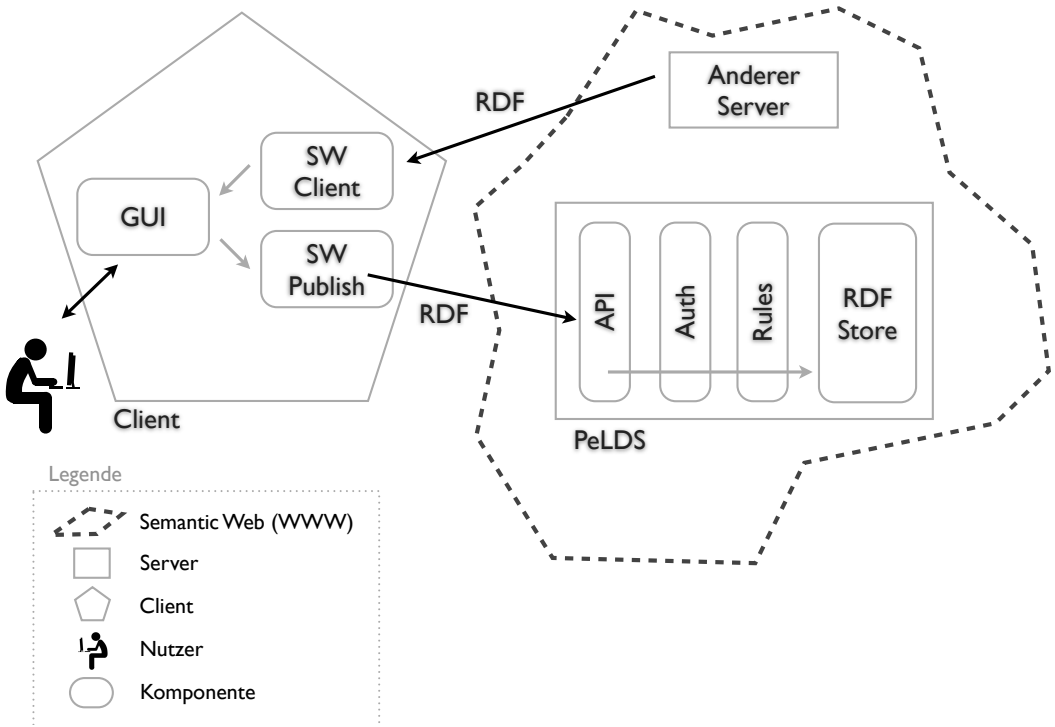

Abbildung 4.1: Strukturübersicht Systemkomponenten

Im weiteren Verlauf dieses Abschnitts wird auf die für diesen Entwurf relevanten Vorgaben der Umgebung in Unterabschnitt 4.1.1 eingegangen. Anschließend beschreibt Unterabschnitt 4.1.2 eine Modellierung von Zugriffsrichtlinien, die hier verwendet wird. Unterabschnitt 4.1.3 beschreibt die Komponenten zur Datenspeicherung näher, und schließlich werden in Unterabschnitt 4.1.4 die Eigenschaften der Laufzeitumgebung erläutert.

4.1.1 Vorgaben der Umgebung

Für die Umsetzung der Anforderungen werden diverse Formate und Protokolle benötigt. Zunächst ist ein Datenmodell notwendig, welches einer definierten Semantik und Repräsentation bedarf. Eine verbreitete Repräsentation stellt beispielsweise die Textform dar. Für die Ausführung der verschiedenen Operationen auf diesem Datenmodell ist die Definition eines Kommunikationsprotokolls erforderlich. Die mittels dieses Protokolls übertragenen Nachrichten werden bezüglich ihres Inhaltes und ihrer Reihenfolge innerhalb des Protokolls festgelegt. Der Ablauf der Sitzungssteuerung im Bezug auf die Authentifizierung kann ebenfalls nach verschiedenen Paradigmen erfolgen. Im Folgenden werden unterschiedliche Varianten diskutiert und eine die erfolgende Auswahl motiviert.

Eine der in der Analysephase identifizierte Anforderung (siehe Unterabschnitt 3.2.3) ist es, die entwickelten Dienste im Umfeld von Linked Data zu erbringen. Die im Grundlagenkapitel (Unterabschnitt 2.2.3) beschriebenen Anforderungen an Linked Data *LD1*, *LD2* und *LD3* werden nun auf Ihre Auswirkungen auf diesen Entwurf untersucht.

Linked Data beschreibt Anforderungen an Semantic-Web-Daten. Damit ist klar, dass als Datenmodell RDF mit seinem Tripelmodell zum Einsatz kommt. *LD1* beschreibt die Namenskonvention für innerhalb von RDF-Graphen verwendete Bezeichner, nach der URLs benutzt werden sollen. Genauso wie *LD3*, das Verknüpfungen innerhalb der Daten fordert, hat dies keine direkten Auswirkungen auf diesen Entwurf. *LD2* legt fest, dass beim Versuch eines Verbindungsaufbaus über das Protokoll HTTP mit den verwendeten Bezeichnern als Ziel-URL der angegebene Server als Antwort RDF-Daten liefert. Die ausgelieferten Daten sollen die mit der URL bezeichnete Ressource – falls möglich – näher beschreiben. Um diese Anforderung umzusetzen, benötigt der Server eine spezialisierte Operation für diesen als Dereferenzierung bezeichneten Vorgang. Da für diese Interaktion das HTTP-Protokoll als Kommunikationsprotokoll festgelegt ist, liegt es nahe, dieses für alle Anfragen an das System zu verwenden.

Da der Server sich in das Linked-Data-Umfeld wie in Unterabschnitt 2.2.3 einfügen soll, sind Teile des Entwurfes damit bereits festgelegt. Datenmodell und Datenaustauschformat sind wie beschrieben festgelegt (siehe Unterabschnitt 2.2.1). Für Anfrageformat und Anfrageresultate (siehe Unterabschnitt 2.2.4) sowie die dafür verwendete Schnittstelle [CFT08] sind de-facto Standards vorgegeben. Diese werden eingehalten, jedoch möglicherweise um weitere Funktionen erweitert, ohne die Kompatibilität zum jeweiligen Standard zu gefährden. Gleichzeitig vereinfachen diese zahlreichen Vorgaben viele Entwurfsentscheidungen, und es kann häufig auf existierende Komponenten zurückgegriffen werden. Es ist sinnvoll, die genannten Standards einzuhalten, um die Integration der entworfenen Software mit bestehender Software zu ermöglichen. In Tabelle 4.1 werden die bestehenden Vorgaben zusammenfassend dargestellt.

Bereich	*Vorgabe*	*Referenz*
Datenmodell	RDF / Linked Data	[MMM04, BHIBL08]
RDF-Serialisierung	RDF/XML	[BM04]
Anfrageprotokoll	SPARQL-Protokoll	[CFT08]
Anfragesprache	SPARQL	[PS08]
Anfrageresultat	SPARQL-Resultat / RDF/XML	[BB08, BM04]
Bezeichnerformat	URI/URL-Format	[MD09, BLMM94]
Transportprotokoll	HTTP(S)	[Int99, Int00]

Tabelle 4.1: Übersicht über die Vorgaben für den Entwurf

4.1.2 Modellierung der Zugriffsrichtlinien

Um mittels RDF dargestellte SWRL-Regeln für die Formulierung von Zugriffsrichtlinien nutzen zu können, muss wie in Unterabschnitt 2.2.2 eingeführt ein entsprechendes Vokabular erstellt werden. Die UML-Klassendarstellung des im Rahmen dieser Arbeit erstellten Vokabulars ist in Abbildung 4.2 abgebildet. Die Hauptelemente sind die Klassen für die Darstellung von Regeln (*Rule*) und zur Abbildung von Aktionen auf RDF-Daten (*Action*). Eine Aktion enthält einen Verweis auf eine Menge vom Benutzer angegebener Regeln. Die Regeln enthalten daraufhin Verweise auf eine Menge von Datenklassifikatoren *TriplePattern*, mit denen wie in Abschnitt 4.2 beschrieben Teile eines RDF-Graphen zur Veröffentlichung freigegeben werden können. Damit die Instanzen der Regelklasse vom Reasoner als SWRL-Regeln erkannt werden können, sind sie von der SWRL-Regelklasse *Imp* abgeleitet. Schließlich sind die einzelnen Aktionen *QueryAction* für eine Anfrage sowie *UpdateAction* für verändernde Operationen spezifiziert, um Regeln spezifisch für einzelne Zugriffsarten formulieren zu können. Für jede Anfrage werden entsprechende Instanzen der Klassen dieser Ontologie erzeugt und von einem Reasoner bearbeitet. Bei einem *Reasoner* handelt es sich um ein Programm, das anhand von OWL-Aussagen und SWRL-Regeln korrekten Konsequenzen ermitteln kann. Der konkrete Ablauf der Regelauswertung wird bei den entsprechenden Komponenten erläutert. Zusätzlich wird in Beispiel 4.1 ein Beispiel für die Verwendung des Vokabulars angegeben. Eine vollständige Repräsentation des Vokabulars ist in Listing A.6 angegeben.

Beispiel 4.1

Der in Abbildung 4.3 grafisch dargestellte RDF-Graph enthält eine Instanz des in Unterabschnitt 4.1.2 zur Darstellung von Zugriffen und Richtlinien entworfenen Vokabulars. Der Graph beschreibt einen einen Zugriff auf in PeLDS verwaltete Daten. Die Ressource *action_1* repräsentiert diesen Zugriff. Über ihr Typattribut *rdf:type* ist festgelegt, dass es sich um eine Instanz von *pelds:QueryAction* handelt, also einen lesenden Zugriff. Die Eigenschaft *pelds:actor* mit Verweis auf die Ressource <`http://example.com/annastein`> beschreibt, dass die Person, die von dieser URI beschrieben wird, erfolgreich authentifiziert wurde. Durch die Eigenschaft *pelds:matchedRule* wird auf die Regeln aus der entsprechenden Zugriffsrichtlinie verwiesen, deren Voraussetzungen erfüllt waren. Im Beispielfall ist dies die in der Ressource *rule_f1* repräsentierte Regel. Sie verweist wiederum

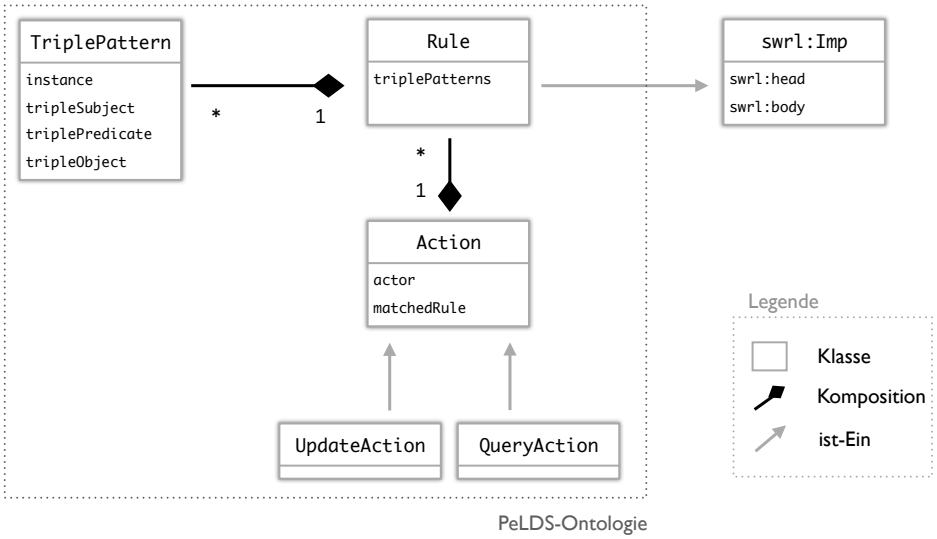

Abbildung 4.2: Klassendiagramm: Darstellung der PeLDS-Zugriffe

mittels *pelds:triplePatterns* auf die zu dieser Regel gehörende Datenklassifikation, die aus einem Tripelmuster *tp_1* besteht. Dieses stellt eine Spezifikation von Tripeln dar, die die Ressource *http://example.com/horststein* beschreiben, was durch die Eigenschaft *pelds:tripleSubject* mit diesem Wert definiert wird.

4.1.3 Datenhaltung

Die von den Nutzern veröffentlichten Daten werden von PeLDS verwaltet. Um diese Verwaltung über größere Zeiträume durchführen zu können, müssen diese dauerhaft abgespeichert werden. Hierzu sind eine Datenbank und spezielle Zugriffsmethoden erforderlich. Außerdem soll es möglich sein, die angebotenen Dienste mehreren Nutzern gleichzeitig zur Verfügung zu stellen, wobei Beeinflussungen zwischen den Nutzern ausgeschlossen werden müssen. Für die Umsetzung dieser Anforderungen kann die folgende Kombination von Softwarekomponenten zum Einsatz kommen: Die Programmbibliothek Jena kann RDF-Daten verwalten und außerdem SPARQL-Anfragen auf diesen ausführen [Hew09a], Named Graphs for Jena (NG4J) stellt eine Erweiterung von Jena dar, mit der beliebig viele unterschiedliche RDF-Graphen, die über einen Bezeichner identifiziert werden, verwaltet und persistent gespeichert werden können [BCH09]. Dies ist nützlich, da ein PeLDS-Server beliebig viele Nutzer und beliebig viele RDF-Graphen verwalten kann. Jena ist in der Programmiersprache Java entwickelt, damit ist es für die weiteren Komponenten sinnvoll, ebenfalls Java zu benutzen, falls Jena/NG4J verwendet wird.

Zusätzlich wird ein Datenbanksystem benötigt, in dem PeLDS die RDF-Graphen, Zu-

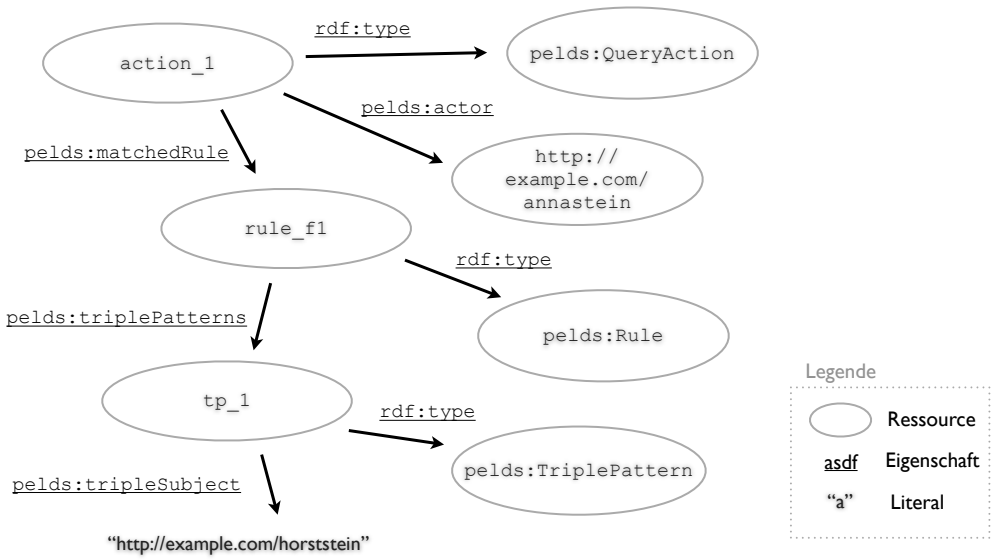

Abbildung 4.3: Beispiel: Instanzen des PeLDS-Schemas

griffsrichtlinien und Metadaten über Graphen wie etwa den Ersteller eines Graphen abspeichern kann. Dieses Datenbanksystem wird über eine abstrahierte Datenbankschnittstelle angebunden. Hierdurch ist PeLDS nicht an ein konkretes Datenbanksystem gebunden. Für die Implementierung des Prototypen von PeLDS wurde die freie Datenbank MySQL [Sun09b] mit Hilfe der Java-Datenbank-Abstraktionsschicht JDBC verwendet, da nur die SQL-Basisoperationen $SELECT$, $INSERT$, $UPDATE$ und $DELETE$ verwendet werden.

4.1.4 Laufzeitumgebung

Alle Zugriffe auf PeLDS erfolgen mittels des Protokolls HTTP. Damit dies möglich ist, muss PeLDS dieses Protokoll implementieren. Dies stellt eine häufige Aufgabe dar, daher existieren spezialisierte Laufzeitumgebungen für solche Programme, um deren Implementierung zu vereinfachen. Für die Implementierung des Prototyps wird daher der freie Java-Servlet-Container Tomcat verwendet [Apa09b]. Tomcat hat eine Reihe von Vorteilen für die Implementierung dieser Anwendung, beispielsweise steht für diesen Server zusätzlich zur Unterstützung von HTTP-Anfragen eine Komponente zur benötigten Authentifizierung mittels FOAF+SSL wie in Unterabschnitt 2.4.2 beschrieben zur Verfügung.

4.2 Richtlinienformat PsSF/SWRL

Zugriffsrichtlinien bestehen aus einer Menge von Regeln im SWRL-Format. Dieses wird in Abschnitt 2.3.3 eingeführt. Der Server garantiert, dass die zu einem Datensatz gehörenden Regeln bei jeder Operation auf diesem Datensatz ausgewertet werden. Da die direkte Darstellung von SWRL-Regeln nur sehr schwer von Menschen lesbar ist, wird eine Kurzform für die Darstellung von SWRL-Regeln eingeführt und verwendet. Diese Kurzform wird hier als Prolog-style SWRL Format (PsSF) bezeichnet und lässt sich in die RDF-Darstellung der SWRL-Regeln [HPSB+05] übersetzen. Zusätzlich werden weitere Prädikate für die Verwendung innerhalb von Regeln definiert, da SWRL nicht für die Formulierung von Zugriffsrichtlinien entworfen wurde. An dieser Stelle wird das Regelformat erläutert, die Übersetzung in SWRL ist in Unterabschnitt 4.4.3 beschrieben.

Regeln werden abhängig von der Art des Zugriffs definiert: Regeln können für Veröffentlichung, Veränderung, Löschung und Anfrage von Daten definiert werden. Alle von SWRL definierten Konstrukte können innerhalb von Regeln verwendet werden. Die angegebenen Regeln enthalten eine Datenklassifikation (siehe unten), die beschreibt, für welche Elemente, Konzeptinstanzen oder Ressourcen des entsprechenden RDF-Graphen diese Regeln gelten. Regeln werden aus Gründen der Entscheidbarkeit von Regeln nach [MSS05] immer positiv formuliert. Mit der gleichen Semantik wie in Beispiel 2.10 wird in Listing 4.1 ein Beispiel für das verwendete Regelformat gegeben. Die im Beispiel angegebene Regel beschreibt, welche Daten bei einer Anfrage (QueryAction) an Familienmitglieder von Herrn Stein ausgeliefert werden. Dies wird über das isRelatedTo()-Prädikat festgelegt. Ist der Teil der Regel vor dem Pfeil erfüllt, wird das permit_resource-Prädikat ausgeführt. Dies führt dazu, dass die im Parameter angegebene Ressource dem anfragenden Nutzer freigegeben wird.

```
1 /*
2 Permit family members to see horst's position
3 */
4 FamilyPosRule:
5 QueryAction(?action) &&
6         actor(?action,?actor) &&
7         foaf:isRelatedTo(http://example.com/horststein,?actor)
8 =>
9 permit_resource(http://example.com/horststein/pos);
```

Listing 4.1: Regeldefinition in PsSF - Beispiel

4.2.1 Regelformat

Eine Richtlinine besteht aus einer Menge von Regeln. Regeln bestehen aus einem Bezeichner und einer Liste von Voraussetzungen sowie einer Liste von Konsequenzen. Beide Listen bestehen aus einer Menge von Prädikaten. Prädikate bestehen aus Präfix, Name und einer

Liste von Argumenten. Argumente können Variablen oder feste Werte sein. Variablen müssen nicht definiert werden, dies geschieht bei deren erstmaliger Verwendung automatisch. In Tabelle A.3 im Anhang wird eine EBNF-ähnliche Spezifikation des Formats angegeben.

4.2.2 Zusätzliche Prädikate

Von den durch SWRL vorgesehenen Prädikaten abgesehen werden noch drei weitere Prädikate zur Auswahl von Teilen eines RDF-Graphen unterstützt. Diese Prädikate setzen die in Unterabschnitt 2.3.2 beschriebene Klassifikation von Daten um. Die zusätzlichen Prädikate sind notwendig, da SWRL an sich keine Sprache zur Formulierung von Zugriffsrichtlinien ist, und daher nicht die notwendigen Möglichkeiten zur Datenklassifikation besitzt.

- **permit_resource(resourceUri)** Die Resource mit dem in *resourceUri* gegebenen Bezeichner wird durch dieses Prädikat erfasst. Der Parameter kann ein $*$ - Zeichen enthalten. Das $*$-Zeichen passt auf beliebige Zeichenketten (Wildcard). Der Parameter kann durch die Verwendung eines Präfixes abgekürzt werden.

- **permit_instance(conceptUri)** Alle Instanzen des OWL- oder RDFS-Konzepts mit dem Bezeichner *conceptUri* werden durch dieses Prädikat erfasst.

- **permit_triple(subject,predicate,object)** Das Tripel, das die angegebenen Parameter *subject*, *predicate* und *object* hat wird durch dieses Prädikat erfasst. Parameter können durch ein $*$ - Zeichen ersetzt werden. Damit akzeptiert das Prädikat alle Werte für diesen Parameter. Parameter können durch die Verwendung eines Präfixes abgekürzt werden.

In Tabelle 4.2 werden einige Beispiele für den Einsatz der beschriebenen erweiterten Prädikate gegeben. Diese Prädikate können benutzt werden, um innerhalb von SWRL-Regeln das RDF-Modell zu beschreiben, das dem berechtigten Benutzer ausgeliefert wird.

permit_resource(http://example.com/horst)	Normalfall
permit_resource(ex:horst)	Verwendung eines Präfixes
permit_instance(http://example.com/ont#Ding)	Normalfall
permit_instance(exOnt:Person)	Verwendung eines Präfixes
permit_triple(ex:horst,exOnt:pos,ex:horst/pos)	Verwendung von Präfixen
permit_triple(ex:horst,*,*)	Verwendung von Wildcards und Präfixen

Tabelle 4.2: EBNF-Darstellung des Regelformats

4.2.3 Verwendung

In Listing 4.2 sind weitere Beispiele für Regeln im PsSF-Format angegeben. Die Bewertung der einzelnen Regeln wird in der folgenden Auflistung erläutert.

Die Regelvoraussetzung wird üblicherweise mit einer Bindung der entsprechenden Aktion eingeleitet, derzeit möglich sind *Action* für alle Aktionen sowie *QueryAction* für Leseoperationen sowie *UpdateAction* für verändernde Operationen. Sollen Aussagen über den anfragenden Nutzer getroffen werden, wird dieser mit Hilfe des *actor()*-Prädikates an die Aktion gebunden. Der Bezeichner der Aktion (etwa *?action*) muss in beiden Prädikaten übereinstimmen.

Der Konsequenzteil enthält bei zur Zugriffsbeschränkung eingesetzten Regeln meist ausschließlich eines oder mehrere der in Unterabschnitt 4.2.2 aufgeführten Prädikate zur Klassifikation der von dieser Regel betroffenen Elemente des RDF-Graphen.

Die Formulierung herkömmlicher SWRL-Regeln mit Hilfe des PsSF-Formats ist weiterhin möglich, dies kann beispielsweise verwendet werden, um eine Reihe von Ressourcen als Instanz des Konzepts „geschütztes Objekt“ zuzuweisen, und Instanzen dieses Konzepts daraufhin mit einer Regel zu schützen. Ein Beispiel für eine solche Richtlinie ist in Abschnitt A.1 angegeben.

- *AnnaResRule* - Regel für die Rechtevergabe an die individuelle Nutzerin <`http://example.com/annastein`>. Die Ressource <`http://example.com/horststein/pos`> wird zum lesenden Zugriff (*QueryAction*) freigegeben.
- *PersonInstanceRule* - Für Nutzer, für die im Datensatz als Typ *exOnt:Person* eingetragen wird, werden alle Instanzen dieses Typs im Datensatz freigegeben.
- *PosUpdaterRule* - Der Anwendung mit der URL <`http://example.org/posupdater`> wird schreibender Zugriff auf alle Tripel mit den Prädikaten *exOnt:longitude* sowie *exOnt:latitude* gewährt. Eine Anwendung zur Aktualisierung der Position des Nutzers auf einem mobilen Gerät könnte mit dieser Regel diese Position verändern.
- *WorldSeeAlsoRule* - Durch die fehlende Bindung des Nutzers über das Prädikat *actor()* wird die Authentifizierung des Nutzers nicht überprüft. Damit wird auch nicht via FOAF+SSL authentifizierten Nutzern der Zugriff auf die Tripel mit dem Subjekt <`http://example.com/horststein`> und dem Prädikat *rdfs:seeAlso* erlaubt.
- *GroupTripleRule* - Nutzern, für die im Datensatz Tripel mit dem Subjekt <`http://example.com/horststein`>, dem Prädikat *exOnt:knows* und der URI des Nutzers im Prädikat sowie ein Tripel mit dem Subjekt <`http://example.com/horststein/group/familie`>, dem Prädikat *exOnt:member* sowie der URI des Nutzers im Prädikat existieren erhalten Zugriff auf alle Tripel mit dem Prädikat *exOnt:phone*.

```
1 AnnaResRule:
2 QueryAction(?action) &&
3    actor(?action, http://example.com/annastein)
4 =>
5 permit_resource(http://example.com/horststein/pos);
6
7 PersonInstanceRule:
```

```
8  QueryAction(?action) &&
9     actor(?action,?actor) &&
10    exOnt:Person(?actor)
11 =>
12 permit_instance(exOnt:Person);
13
14 PosUpdaterRule:
15 UpdateAction(?action) &&
16    actor(?action,http://example.org/posupdater)
17 =>
18 permit_triple(*,exOnt:longitude,*) &&
19 permit_triple(*,exOnt:latitude,*);
20
21 WorldSeeAlsoRule:
22 pelds:QueryAction(?action)
23 =>
24 permit_triple(http://example.com/horststein,rdfs:seeAlso,*);
25
26 GroupTripleRule:
27 QueryAction(?action) &&
28    actor(?action,?actor) &&
29    exOnt:knows(http://example.com/horststein,?actor) &&
30    exOnt:member(http://example.com/horststein/group/familie,?
         actor)
31 =>
32 permit_triple(*,exOnt:phone,*);
```

Listing 4.2: Weitere PsSF-Beispiele

4.3 Funktionalität

Die Funktionalität von PeLDS wird nach dem in Unterabschnitt 2.2.5 genannten Schema gegliedert und beschrieben. In diesem Abschnitt wird vor allem die Nutzersicht auf das System beschrieben. Verbindungsaufbau, Eingabewerte, Ablauf, Datenformate und Rückgabewerte werden für jede Operation gezeigt und erläutert.

Alle Operationen verwenden das Basisprotokoll HTTP oder dessen verschlüsselte Variante HTTPS, mittels dieses Protokolls können die einzelnen Dienste angesprochen werden. Innerhalb einer HTTP-Anfrage können beliebige Anfrageparameter übertragen werden. Diese Parameter besitzen einen Bezeichner und werden verwendet, um entsprechende Parameter für die einzelnen Operationen festzulegen. In den folgenden Abschnitten werden die jeweils gültigen Parameter und ihre erwarteten Werte genannt.

Für die Beschreibung der einzelnen Operationen wird vorausgesetzt, dass eine PeLDS-Installation über die URI <`https://example.pelds/`> über HTTP erreicht werden kann.

Im Folgenden wird zunächst die Veröffentlichung von Zugriffsrichtlinien beschrieben. Diese Richtlinien werden verwendet, um die Herausgabe der von den Nutzern veröffentlichten Daten zu kontrollieren. Danach werden die Möglichkeiten zur Veröffentlichung und Veränderung von RDF-Daten mit Hilfe der Sprache SPARQL/Update zur Formulierung von gewünschten Veränderungen gezeigt. Schließlich werden die Operationen zum Abruf von Daten beschrieben. Zusätzlich bietet der Server auch einen Dienst zur Erstellung von kryptographischen Zertifikaten an wie in Unterabschnitt 4.3.5 gezeigt.

Eine Übersicht über die Funktionalität wird in Tabelle 4.3 gegeben.

Dienst	*Operation*	*Verweis*
Richtlinienverwaltung	Veröffentlichung Veränderung Anzeige	4.3.1
Datenverwaltung	Veröffentlichung Veränderung Löschung	4.3.2
Datenabruf	Abfragesprache Dereferenzierung	4.3.3 4.3.4
Zertifikate	Zertifizierung	4.3.5

Tabelle 4.3: Übersicht über die Funktionalität von PeLDS

4.3.1 Veränderung und Abfrage der Zugriffsrichtlinie

Der Dienst zur Verwaltung von Zugriffsrichtlinien enthält zwei Operationen: Eine Zugriffsrichtlinie kann verändert oder abgerufen werden. Die hier zu verwendenden Zugriffsrichtlinien werden wie in der Analyse festgestellt von den Nutzern des Systems festgelegt. Daher benötigen die Nutzer eine Möglichkeit, dem Server ihre Richtlinien mitzuteilen. Hierzu dient diese Operation: Der Nutzer gibt den Bezeichner eines Graphen sowie die Zugriffsrichtlinie für diesen Graphen an. Der Server prüft, ob der Nutzer berechtigt ist, für den

bezeichneten Graphen die Zugriffsrichtlinie zu verändern. Ist das der Fall, wird die vom Nutzer angegebene Richtlinie mit einem Verweis auf den betreffenden Graph gespeichert. Die Zugriffsrichtlinie wird stets für den gesamten Graphen festgelegt. Eine möglicherweise bestehende Richtlinie wird überschrieben, Richtlinien werden somit ausschließlich atomar verändert. Eine Authentifizierung ist für diese Operation zwingend notwendig.

Anfrageparameter

Die Anfrageparameter für eine Anfrage für den Zugriff auf die Zugriffsrichtlinien lauten:

- **dataset** Graph-URI (notwendig)
- **rules** PsSF-Zugriffsrichtlinie (optional)

Ablauf

In Abbildung 4.4 wird der Ablauf einer Regelveränderung zwischen den in Abbildung 4.1 beschriebenen Komponenten dargestellt.

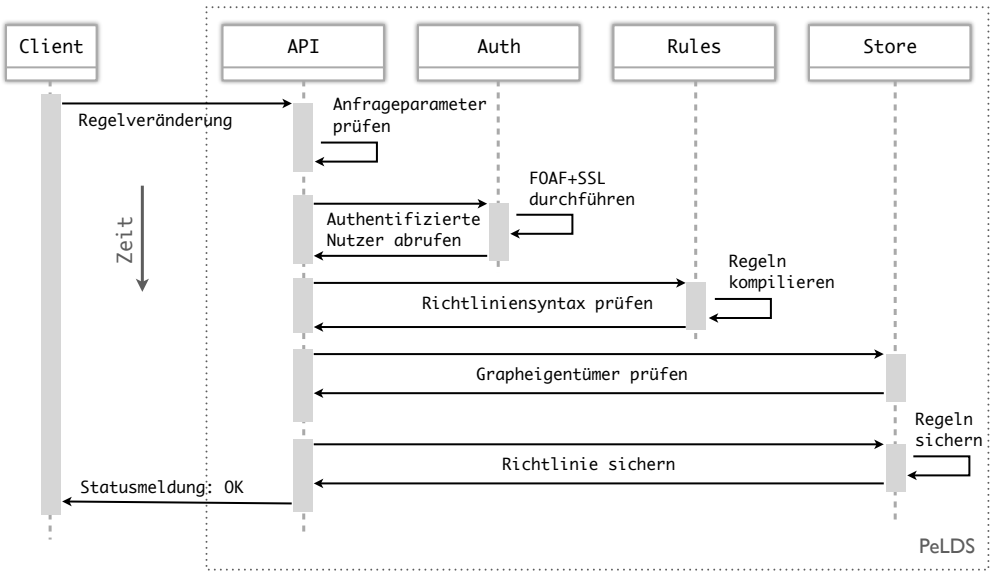

Abbildung 4.4: Regelveränderung: Sequenzdiagramm

1 Der Client stellt eine HTTP/HTTPS-Anfrage an die API, die einen Graphbezeichner in URI-Form sowie eine Zugriffsrichtlinie im PsSF-Format enthält. Dieses Format ist in Abschnitt 4.2 beschrieben.

2 Die API überprüft das Vorhandensein der beschriebenen Anfrageparameter.

3 Die Authentifizierung via FOAF+SSL wird durch *Auth* durchgeführt. Resultat ist eine Liste von URI-Bezeichnern, die erfolgreich überprüft wurden.

4 Die Richtliniensyntax wird in *Rules* überprüft. Hierzu werden die in der Richtlinie enthaltenen Regeln testhalber kompiliert.

5 Der Eigentümer des im Parameter **dataset** bezeichneten Graphen wird überprüft, nur dieser ist berechtigt, die Richtlinie zu verändern. Existiert der angegebene Graph noch nicht in der Datenbank, wird er angelegt.

6 Die neue Zugriffsrichtlinie wird für den bezeichneten Graph gespeichert und dem Client der HTTP-Statuscode „200 - OK“ übermittelt.

Bei der Anzeigeoperation wird eine möglicherweise bestehende Richtlinie aus der Datenbank geladen. Besteht keine Richtlinie für den angegebenen Graphen in der Datenbank, wird ein Fehler ausgegeben. Soll die Anzeigeoperation verwendet werden, ist der Parameter **rules** wegzulassen. In Beispiel 4.2 wird beispielhaft der Ablauf einer Regelveränderung beschrieben.

Beispiel 4.2

Herr Stein legt eine Zugriffsrichtlinie für seine in einem späteren Schritt veröffentlichten Daten fest. Hierzu erstellt er die in Listing 4.3 gezeigte Richtlinie. Diese besagt, dass nur Menschen, mit denen er verwandt ist, alle Informationen über ihn unter seiner URI <`http://example.com/horststein`> abrufen können (FamilyRule). Für alle anderen Nutzer soll nur der Verweis auf andere URIs sichtbar sein (PublicRule).

1 Herr Stein stellt eine Anfrage über HTTPS an die Schnittstelle zur Regelveränderung unter <`http://example.pelds/rule`>. Seine Anfrage enthält die beschriebene Richtlinie sowie den Bezeichner des Graphen, für den diese Richtlinie gelten soll (<`http://example.com/horststein`>) als Parameter.

2 Herr Stein wird via FOAF+SSL authentifiziert. Es wird festgestellt, dass noch kein Graph mit dem angegebenen Bezeichner <`http://example.com/horststein`> existiert, daher wird dieser angelegt und Herr Stein als erstellender Nutzer mit allen Rechten eingetragen.

3 Die von Herr Stein angegebenen Regeln werden syntaktisch geprüft, akzeptiert und für den neuen Graph abgespeichert. Der Server antwortet mit dem Statuscode „200 - OK“, um die erfolgreiche Abspeicherung der Richtlinie anzuzeigen.

Nun ist der Regelsatz von Herr Stein gespeichert und in seinem RDF-Graphen veröffentlichte Informationen können nur nach Auswertung dieser Regeln abgerufen werden. Dieses Beispiel wird in Beispiel 4.3 fortgesetzt.

```
1 PublicRule:
2 pelds:QueryAction(?action)
3  =>
4 pelds:permit_triple(http://example.com/horststein,rdfs:seeAlso
      ,*) ;
5
6 FamilyRule:
7 pelds:QueryAction(?action) &&
8 pelds:actor(?action,?actor) &&
9 foaf:isRelatedTo(http://example.com/horststein,?actor)
10  =>
11 pelds:permit_triple(http://example.com/horststein,*,*) ;
```

Listing 4.3: Zugriffsrichtlinie für Herr Stein

4.3.2 Datenveröffentlichung und -veränderung

Nutzer können Daten auf dem Server veröffentlichen. Nach dieser Veröffentlichung ist eine Möglichkeit zur Veränderung dieser Daten sowie zur Entfernung notwendig. Diese Befehle werden allesamt von der einzigen Operation zur Datenveränderung abgedeckt. Für die Veröffentlichung und Veränderung wäre es möglich, dem Server bei jeder Datenveränderung einen serialisierten RDF-Graphen zu senden. Dies hat jedoch den Nachteil, dass selbst für die Änderung eines einzigen Tripels der gesamte Graph übertragen werden muss. Zudem erhöht die Übertragung von Teilen die Sicherheit des Systems. Sollte die Kommunikation trotz verschlüsselter Übertragung abgefangen werden, kann nicht der gesamte Graph eingesehen werden. Es ist daher sinnvoller, nur die gewünschten Änderungen am Graphen an den Server zu übertragen. Hierzu wurde die Sprache SPARQL/Update entworfen [SMB+08]. Diese ähnelt der in Unterabschnitt 2.2.4 beschriebenen Sprache zur Formulierung von Anfragen, die verfügbaren Befehle sind wie bei SPARQL an SQL angelehnt.

Die Sprache SPARQL/Update erlaubt es, Anfragen für die Erzeugung sowie Veränderung von RDF-Graphen zu formulieren. PeLDS unterstützt ausschließlich diese Sprache für den schreibenden Zugriff auf RDF-Daten und bietet eine Schnittstelle dafür an. Hierzu wird eine Anfrage an die vorgesehene URI <`https://example.pelds/update`> gestellt.

Anfrageparameter

Die Anfrageparameter für eine SPARQL/Update-Anfrage lauten:

- **query** Eine SPARQL/Update-Veränderungsanfrage (notwendig)
- **default-graph-uri** Graph-Bezeichner in URI-Form (notwendig)

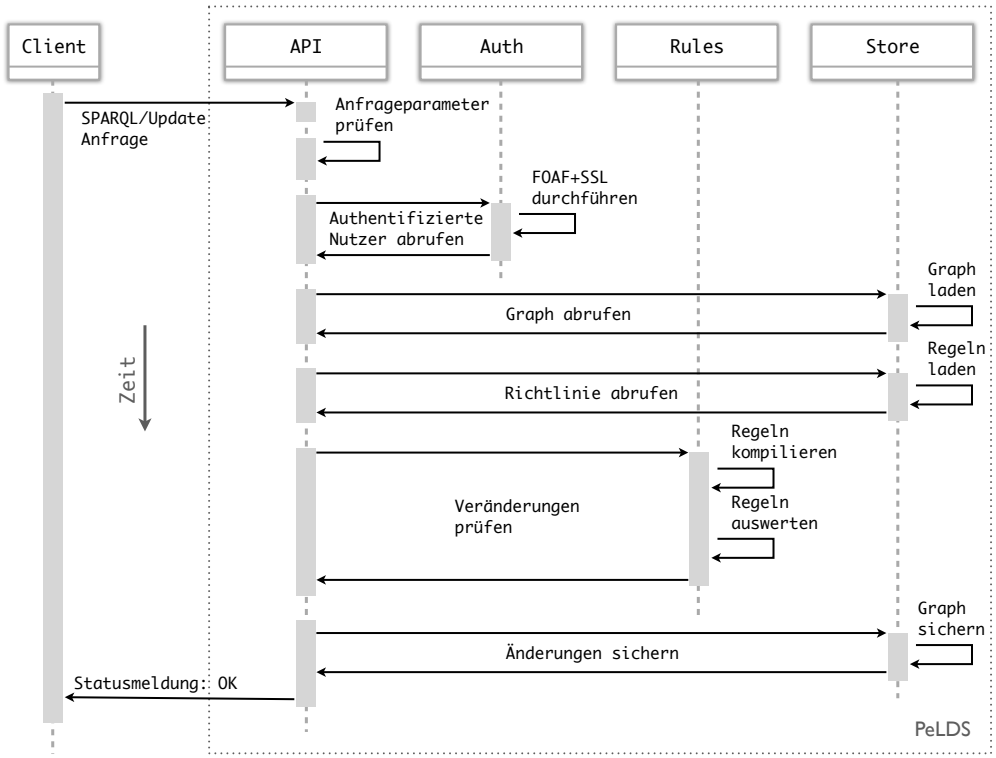

Abbildung 4.5: Datenveränderung: Sequenzdiagramm

Ablauf

Das Protokoll für Veränderungen ähnelt dem in Unterabschnitt 4.3.3 verwendeten Ablauf für Anfragen stark. Im Unterschied dazu ist jedoch bei der Veränderungs-Anfrage eine Authentifizierung sowie verschlüsselte Übertragung zwingend vorgeschrieben. Der generelle Ablauf stellt sich wie folgt dar und ist in Abbildung 4.5 als Sequenzdiagramm abgebildet:

1 Der Client stellt eine HTTPS-Anfrage an das Servlet.

2 Das Servlet überprüft die beschriebenen Anfrageparameter auf syntaktische Korrektheit.

3 Die Authentifizierung via FOAF+SSL wird durchgeführt. Resultat ist eine Liste von URI-Bezeichnern, die erfolgreich überprüft wurden.

4 Der in der Anfrage benannte RDF-Graph wird aus der Datenbank („Store“) geladen.

5 Die für diesen Graphen angegeben Regeln werden aus der Datenbank geladen und kompiliert. Das Resultat wird in das SWRL-Format übertragen.

6 Die angeforderte Änderung des Graphen wird durchgeführt. Die veränderten Tripel werden beobachtet. Wird ein Tripel verändert, für das die Regeln einen solchen Zugriff verbieten, werden alle Änderungen rückgängig gemacht. Falls keine Konflikte auftreten, werden die Änderungen gespeichert. Siehe hierzu auch Unterabschnitt 4.4.3.

Wurde die Änderung erfolgreich durchgeführt, wird dies dem anfragenden Client durch die Übertragung des HTTP-Statuscodes „200 - OK“ kommuniziert. Sollte ein Fehler auftreten, wird dies mittels des von HTTP vorgesehenen Fehlercode „500 - Internal Server Error“ dem Client mitgeteilt [Int99]. Falls möglich, enthält diese Fehlerantwort eine Präzisierung der Gründe für den Fehlerfall in Textform. Jedoch bleibt zu beachten, dass die Präzisierung dieser Gründe keine Hilfestellung für eine Umgehung der Regeln geben darf, um die Systemsicherheit nicht zu gefährden. Beispielsweise wird dem Client im Fehlerfall nicht kommuniziert, welche Regel (oder deren Inhalt) zur Ablehnung der Anfrage geführt hat. Die gewählte Art und Weise, in der Fehler kommuniziert werden, stellt damit einen Kompromiss dar, der die Bedienbarkeit so gut als möglich unterstützt, jedoch gleichzeitig die Systemsicherheit nicht gefährdet. Beispiel 4.3 beschreibt exemplarisch den Ablauf und Resultate einer Datenveränderung.

Beispiel 4.3

Herr Stein veröffentlicht seine Daten. Die in Beispiel 4.2 gezeigte Zugriffsrichtlinie ist bereits auf dem Server installiert. Nun stellt er die in Listing 4.4 angegebene Anfrage im SPARQL/Update-Format an den Server. Diese Anfrage enthält fünf Tripel: Bei der Ressource <`http://example.com/horststein`> handelt es sich um eine Person (Zeile 6), diese Person heißt „Horst Stein“ (Zeile 7), ihre Telefonnummer lautet „030123456“ (Zeile 8), sie ist mit der mit <`http://example.com/annastein`> bezeichneten Person verheiratet (Zeile 9) und unter der Adresse <`http://example.org/staff/horststein`> sind weitere Informationen zu dieser Person zu finden (Zeile 10).

1 Herr Stein stellt eine Anfrage über HTTPS an die Schnittstelle zur Datenveränderung unter <`http://example.pelds/update`>. Seine Anfrage enthält die genannte Anfrage im SPARQL/Update-Format sowie den Bezeichner des Graphen, der verändert werden soll (<`http://example.com/horststein`>, Zeile 3).

2 Herr Stein wird via FOAF+SSL authentifiziert. Es wird festgestellt, dass schon ein Graph mit dem angegebenen Bezeichner <`http://example.com/horststein`> existiert. Da Herr Stein unter seiner authentifizierten URI als Ersteller dieses RDF-Graphen eingetragen ist, wird die Veränderungsoperation zugelassen.

3 Die von Herr Stein angegebene Anfrage wird durchgeführt, der Datensatz in Form eines RDF-Graphen geändert und gespeichert. Der Server antwortet mit dem Statuscode „200 - OK“, um auf die erfolgreiche Veröffentlichung der Daten hinzuweisen.

Die Daten von Herr Stein wurden nun veröffentlicht und können unter Beachtung seiner Richtlinie abgerufen werden. In Beispiel 4.4 wird dieses Beispiel fortgesetzt.

```
1 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3 INSERT INTO <http://example.com/horststein> {
4   <http://example.com/horststein>
5     a foaf:Person ;
6     foaf:name "Horst Stein" ;
7     foaf:phone "030123456" ;
8     foaf:isMarriedTo <http://example.com/annastein> ;
9     rdfs:seeAlso <http://example.org/staff/horststein> .
10 }
```

Listing 4.4: SPARQL/Update für Herr Stein

4.3.3 Datenabruf mittels Anfragesprache

In vielen Fällen werden nur wenige Elemente eines RDF-Graphen für die Beantwortung einer Anfrage benötigt. Dadurch ist es nicht notwendig, dem Client alle Elemente zu senden, für die er berechtigt ist. Es wäre sinnvoller, dem Client eine Möglichkeit zu geben, nur die Elemente abzurufen, die benötigt werden. In solchen Fällen kann eine Anfragesprache benutzt werden, um nur die Tripel des Graphen auszuwählen, die im konkreten Fall für die Beantwortung einer Anfrage benötigt werden. Hierfür wurde die Sprache SPARQL entworfen und standardisiert. Die meisten existierenden RDF-Server bieten eine Schnittstelle für Anfragen mittels SPARQL an.

Mittels der in Unterabschnitt 2.2.4 beschrieben Anfragesprache SPARQL können Anfragen für die Auswahl von Elementen eines RDF-Graphen an einen RDF-Server gestellt werden, der SPARQL unterstützt. PeLDS bietet eine solche Schnittstelle für den lesenden Zugriff auf die gespeicherten Daten an. Hierzu muss eine Anfrage an die entsprechende URI <`https://example.pelds/query`> gestellt werden.

Anfrageparameter

Die Anfrageparameter für eine SPARQL-Anfrage lauten:

- **query** Eine SPARQL-Anfrage (notwendig)
- **output** Ausgabeformat. Alternativ kann das Ausgabeformat auch über einen HTTP-Header ausgewählt werden (Content Negotiation). Mögliche Werte sind „xml“ für XML-Format [BB08], „json“ für das entsprechende JSON-Format [Int06] oder „txt“ für eine Textausgabe.
- **default-graph-uri** Graph-Uri, kann weggelassen werden, wenn die SPARQL-Anfrage diese explizit enthält. Dies kann durch die Verwendung des Schlüsselwortes „FROM“ innerhalb der SPARQL-Anfrage erreicht werden.

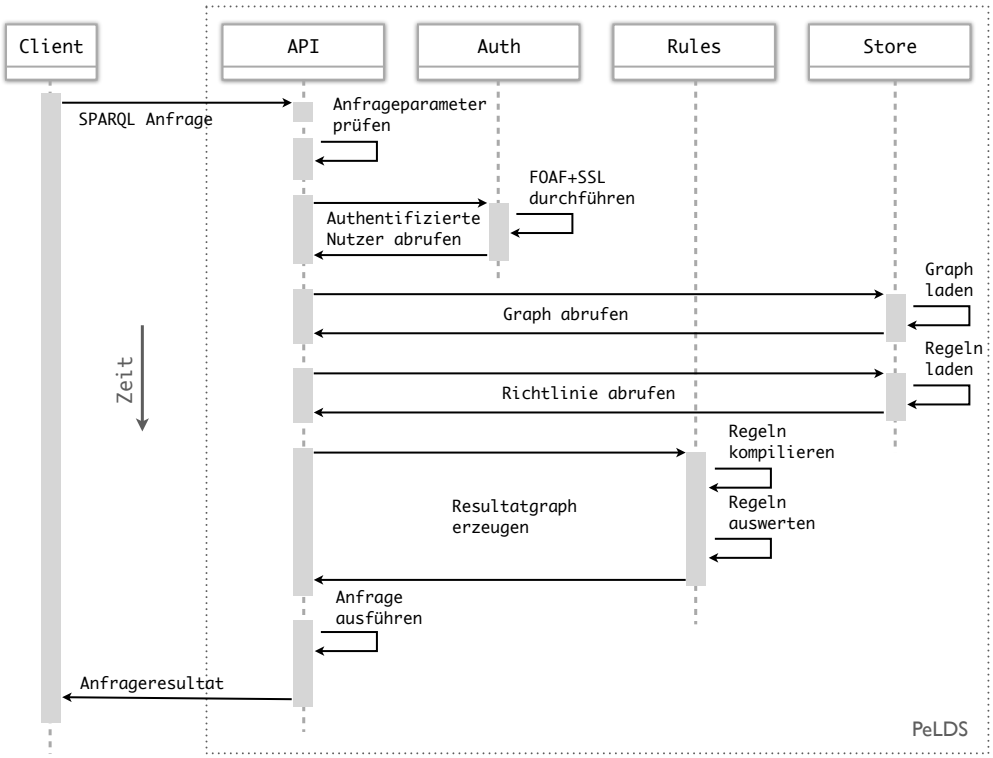

Abbildung 4.6: Anfrage: Sequenzdiagramm

Ablauf

Die genannte Schnittstelle implementiert das standardisierte SPARQL-Protokoll [CFT08]. Dieses wurde durch die Forderung nach der expliziten Angabe des Graphen geringfügig eingeschränkt. Die Angabe des Graphen ist notwendig, um die für diesen Graphen angegeben Regeln auffinden zu können. Zusätzlich ist eine Authentifizierung mittels FOAF+SSL (siehe Unterabschnitt 2.4.2) möglich, jedoch nicht vorgeschrieben, um Kompatibilität mit bestehender Software zu gewährleisten. Wird keine Authentifizierung verwendet, wird der anfragende Client mittels Metadaten im Anfrageresultat darauf hingewiesen, dass der Server diese Authentifizierung bietet. Weiterhin können bei nicht durchgeführter Authentifizierung keine Regeln zur Anwendung kommen, die die Identität des anfragenden Nutzers beinhalten.

In Abbildung 4.6 wird der Ablauf einer Anfrage zwischen den in Abbildung 4.1 beschriebenen Komponenten gezeigt.

1 Der Client stellt eine HTTP/HTTPS-Anfrage an die API, die eine SPARQL-Anfrage enthält

2 Die API überprüft die beschriebenen Anfrageparameter auf syntaktische Korrektheit.

3 Die Authentifizierung via FOAF+SSL wird durch *Auth* durchgeführt. Resultat ist eine Liste von URI-Bezeichnern, die erfolgreich überprüft wurden.

4 Der in der Anfrage benannte RDF-Graph wird aus der Datenbank (*Store*) geladen.

5 Die für diesen Graphen angegebene Richtlinie wird durch *Rules* kompiliert. Die Regeln werden anhand der Daten im RDF-Graphen ausgewertet. Aus dem Resultat dieser Auswertung wird ein Resultatgraph erzeugt, der nur noch autorisierte Tripel enthält.

6 Auf dem Resultatgraph wird die angegebenen SPARQL-Anfrage ausgeführt und das Ergebnis zurück an den Client geschickt. Siehe hierzu auch Unterabschnitt 4.4.3.

Resultat

Das Format der Antwort des Servers hängt von der Art der Anfrage ab. Eine SELECT-Anfrage führt zu einer Antwort im SPARQL-Resultatformat [BB08], eine DESCRIBE-Anfrage erzeugt eine Antwort im RDF-Format, die einen Graphen enthält. Falls der Graph Ontologien enthält, sind die daraus anhand der gespeicherten Daten ableitbaren Informationen ebenfalls abrufbar. Die Serialisierung der verschiedenen Antworttypen hängt von den vom Benutzer angegeben Formatpräferenzen ab. Tritt ein Fehler auf, wird dies mittels des von HTTP vorgesehenen Fehlercode „500 - Internal Server Error“ dem Client mitgeteilt [Int99]. Falls möglich, enthält diese Fehlerantwort eine Präzisierung der Gründe für den Fehlerfall in Textform. In Beispiel 4.4 wird ein Beispiel für eine Anfrage gegeben.

Beispiel 4.4

Nachdem Herr Stein in Beispiel 4.2 eine Richtlinie für den RDF-Graph mit dem Bezeichner <`http://example.com/horststein`> vergeben hat und in Beispiel 4.3 in diesem Graph Daten über sich veröffentlicht hat, ruft nun seine Frau, Anna Stein, seine aktuelle Telefonnummer ab. In Listing 4.5 wird die von ihr abgesetzte SPARQL-Anfrage angegeben. In dieser Anfrage wird zunächst festgelegt, dass Name und Telefonnummer in der Ausgabe erscheinen sollen (Zeile 2). In der gleichen Zeile wird mittels des Schlüsselwortes $FROM$ festgelegt, welcher Graph verwendet werden soll. Das Schlüsselwort $WHERE$ legt fest, dass nur Tripel mit dem Subjekt <`http://example.com/horststein`> für die Antwort in Frage kommen (Zeile 5), und dass die Objekte der Tripel mit diesem Subjekt und den Prädikaten $foaf : phone$ sowie $foaf : name$ an die jeweiligen Variablen gebunden werden sollen (Zeilen 6,7).

1 Frau Stein stellt mittels ihres Clients eine SPARQL-Anfrage über HTTPS an die Schnittstelle zur Datenabfrage unter <`https://example.pelds/query`>. Ihre Anfrage enthält die beschreibende SPARQL-Anfrage. Diese Anfrage enthält auch den Bezeichner des Graphen, der angefragt werden soll (<`http://example.com/horststein`>). Gleichzeitig wird über Content Negotation festgelegt, dass das Resultat im Textformat zurückgegeben werden soll.

2 Frau Stein wird via FOAF+SSL authentifiziert. Details dieses Vorgangs sind im Bei-

spiel 2.11 zu finden. Danach ist sie mit ihrer URI <http://example.com/annastein> sicher identifiziert.

3 Die von Herr Stein vergebene Zugriffsrichtlinie wird ausgewertet. Frau Stein wird von der Regel „FamilyRule“ erfasst, da ein Tripel existiert, dass die dort angegebene Voraussetzung erfüllt. Damit hat Frau Stein Zugriff auf alle Informationen über die Ressource von Herr Stein.

4 Die von Frau Stein angegebene Anfrage wird durchgeführt, ihr Resultat in dem von ihr gewünschten Format zurückgegeben. Der Server antwortet mit dem Statuscode „200 - OK“, um auf die erfolgreiche Veröffentlichung der Daten hinzuweisen.

Das Ergebnis der Anfrage ist in Tabelle 4.4 dargestellt. Die für die Beantwortung verwendeten Daten sind diejenigen, die in Listing 4.4 angegeben sind. Sichtbar wird, dass die Telefonnummer von Herrn Stein darin enthalten ist. Frau Stein kann diese sehen, da Herr Stein diese Information mittels seiner Richtlinie für sie freigegeben hat.

```
1 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2 SELECT ?name ?phone FROM <http://example.com/horststein>
3 WHERE {
4   <http://example.com/horststein>
5     foaf:phone ?phone ;
6     foaf:name ?name .
7 }
```

Listing 4.5: SPARQL-Anfrage von Frau Stein

name	phone
Horst Stein	030123456

Tabelle 4.4: Ergebnis der SPARQL-Anfrage aus Listing 4.5

4.3.4 Datenabruf mittels Dereferenzierung

Linked Data (Unterabschnitt 2.2.3) fordert bei der so genannten Dereferenzierung einer innerhalb eines RDF-Graphen verwendeten URI die Verfügbarkeit von RDF-Daten unter dieser URL. Da die hier entworfene Software sich in das Linked-Data-Umfeld integrieren soll, ist es notwendig, diese Dereferenzierung zu unterstützen.

Um diese Anforderung umzusetzen, wird die nun beschriebene Operation verwendet: Sie erfasst Anfragen unter einer vom Nutzer festzulegenden URL und leitet diese an die Schnittstelle zum Datenabruf mittels Anfragesprache (Unterabschnitt 4.3.3) weiter. Diese stellt dann – falls möglich – einen RDF-Graphen zu Verfügung, der die mit dem Bezeichner,

der der URL der Anfrage entspricht näher beschreibt. Da diese Operation nur durch die Auflösung einer URL aufgerufen wird, hat sie lediglich die URL der Anfrage als Parameter.

Notwendige Einstellungen für diese Operation werden im Umfeld der Installation des Systems getroffen. Hierbei wird konfiguriert, unter welcher URI der SPARQL-Dienst erreichbar ist und zusätzlich, welches Präfix für die URLs der Weiterleitung gelten soll. Dies kann verwendet werden, um eine Abbildung von erreichbaren URLs auf die im Graphen verwendeten URIs zu realisieren.

In Beispiel 4.5 wird der Ablauf einer Dereferenzierung im Kontext des in diesem Kapitel verwendeten Beispiels dargestellt.

Beispiel 4.5

Analog zu Beispiel 4.4 hat Herr Stein Daten und eine dazu passende Richtlinie auf einer PeLDS-Instanz veröffentlicht. Dem Client von Frau Stein wurde lediglich die URI <`http://example.com/horststein`>, die Herrn Stein bezeichnet, bekannt gemacht. Daher stellt Frau Steins Cleint eine HTTP-Anfrage an diese URI. Die Instanz erhält diese Anfrage und ermittelt anhand ihrer Konfiguration die – in diesem Fall identische – URI des Graphen <`http://example.com/horststein`>, der Informationen über die erwähnte Ressource enthält, die Herr Stein beschreibt.

Eine HTTP-Weiterleitung an eine andere URI wird erzeugt und an den Client von Frau Stein übertragen. Diese Anfrage enthält eine SPARQL-Anfrage vom Typ *DESCRIBE* sowie den Verweis auf den entsprechenden Graph mittels des Parameters *default-graph-uri*. Die beschriebene Anfrage wird in Listing 4.6 dargestellt.

Stellt der Client von Frau Stein nun eine Anfrage an diese URI, erhält sie als Antwort die Serialisierung eines RDF-Graphen, der die von Herr Stein veröffentlichten und für sie verfügbaren Informationen enthält. Eine mögliche Serialisierung des Resultatgraphs ist in Listing 4.7 angegeben.

```
1 DESCRIBE <http://example.com/horststein>
```

Listing 4.6: SPARQL-DESCRIBE-Anfrage

```
1 <rdf:Description rdf:about="http://example.com/horststein">
2   <foaf:name>Horst Stein</foaf:name>
3   <foaf:phone>030123456</foaf:phone>
4   <foaf:isRelatedTo rdf:resource="http://example.com/annastein"
        />
5   <rdfs:seeAlso rdf:resource="http://example.org/staff/
        horststein"/>
6    [...]
7 </rdf:Description>
```

Listing 4.7: SPARQL-Resultat nach DESCRIBE-Anfrage - gekürzt

4.3.5 Zertifikaterstellung

Um die für Nutzer nicht triviale Erstellung von digitalen Zertifikaten zu vereinfachen, bietet der PeLDS-Server zusätzlich eine Funktion, mit dem unter Angabe der URI des Nutzers ein wie in Unterabschnitt 2.4.2 beschriebenes digitales Zertifikat erstellt wird. Ein solches digitales Zertifikat besteht aus dem öffentlichen Schlüssel und einer kryptographischen Signatur für das gesamte Zertifikat. Um diese Signierung für den Client durchführen zu können, benötigt der PeLDS-Server einen eigenen kryptographischen Schlüssel.

Hierzu wird eine geeignete Anfrage an die URL <`https://example.pelds/cert`> gestellt.

Anfrageparameter

Die Anfrageparameter für eine Zertifikaterstellung (alle notwendig) lauten:

- **spkac** Zertifizierungsanfrage in kodierter Form, siehe unten
- **url** Web ID (URL) zur Bezeichnung des Nutzers oder Computersystems
- **lifetime** Gültigkeitszeitraum des Zertifikates ab dem aktuellen Zeitpunkt in Tagen

Ablauf

In Abbildung 4.7 ist der Ablauf einer Anfrage zwischen den in Abbildung 4.1 beschriebenen Komponenten als Sequenzdiagramm angegeben. Als einen Eingabewert erhält diese Operation eine Signed Public Key And Challenge (SPKAC), diese enthält unter anderem den öffentlichen Teil eines asymmetrischen Schlüsselpaares. Hierzu wird von dem Client, der die Anfrage stellt, ein Schlüsselpaar aus öffentlichem und privaten Schlüssel erstellt. Der öffentliche Teil sowie ein zufälliger Eingabewert (*Challenge*) wird mittels des privaten Schlüssels signiert und diese signierten Daten werden an den Server übertragen. Durch den zufällige Eingabewert kann eine einfache Prüfung durchgeführt werden, ob der Client im Besitz des passenden privaten Schlüssels ist. Dies wird durch die Überprüfung der im SPKAC enthaltenen Signatur erreicht. In Listing 4.8 ist der Inhalt eines SPKAC exemplarisch dargestellt. Dieser umfasst den öffentlichen Schlüssel (Zeilen 2-6), die zufällige Challenge (Zeile 7) sowie die Signatur (Zeilen 8,9).

Der öffentliche Schlüssel des Clients wird nun mittels des Server-Schlüssels in einem neuen Zertifikat abgelegt. Die angegebene Web ID des Clients wird ebenfalls im Zertifikat gespeichert. Das gesamte Zertifikat wird mittels des Schlüssels des Servers signiert und im in Listing 2.9 exemplarisch angegebenen Format in an den Client gesendet. Eine Speicherung des Zertifikates findet nicht statt.

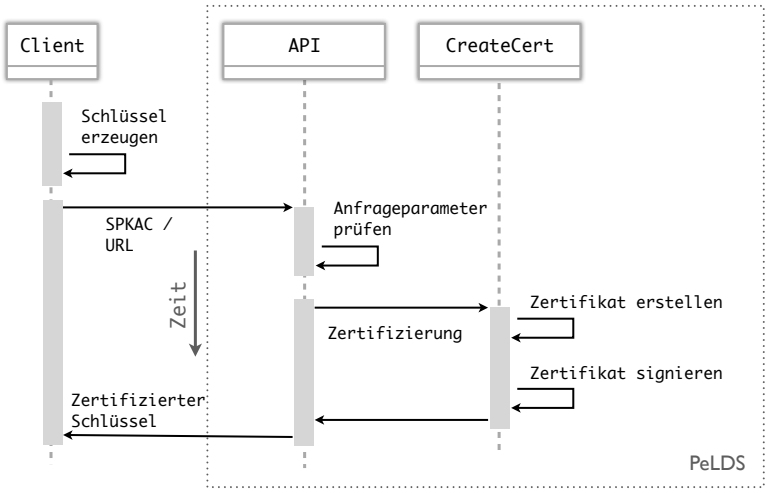

Abbildung 4.7: Zertifikaterstellung: Sequenzdiagramm

```
Netscape SPKI:
  Public Key Algorithm: rsaEncryption
  RSA Public Key: (2048 bit)
  Modulus (2048 bit):
      00:c6:[...]:ec:11
  Exponent: 65537 (0x10001)
  Challenge String: ch_1255621971379_
  Signature Algorithm: md5WithRSAEncryption
      69:59:[...]:43:67
```

Listing 4.8: Inhalt SPKAC-Anfrage (gekürzt)

4.4 Komponenten

Die Funktionalität des PeLDS-Systems wird hier in verschiedene, nach ihrer Funktion gruppierten Komponenten aufgeteilt. Dies geschieht, um eine optimale Systemstruktur mit Hinblick auf Erweiterbarkeit und Wiederverwendbarkeit einzelner Komponenten zu erhalten.

In Abbildung 4.8 ist eine solche Aufteilung als Klassendiagramm gegeben. PeLDS gliedert sich in vier Hauptbereiche, die im Folgenden weiter erläutert werden: Die Komponente für die Behandlung von Anfragen über HTTP(S) *Servlet* (Unterabschnitt 4.4.1), die Komponente für die FOAF+SSL-Authentifizierung *Auth* (Unterabschnitt 4.4.2), die Komponente für die Auswertung von Regeln *Rules* (Unterabschnitt 4.4.3) sowie die Komponente *Store*, die Daten persistent speichert (Unterabschnitt 4.4.4). Auf wichtige Module wird im Hinblick auf ihre benötigte Funktion eingegangen, im Interesse einer besseren Übersicht wurden in der Grafik weniger wichtige Module weggelassen.

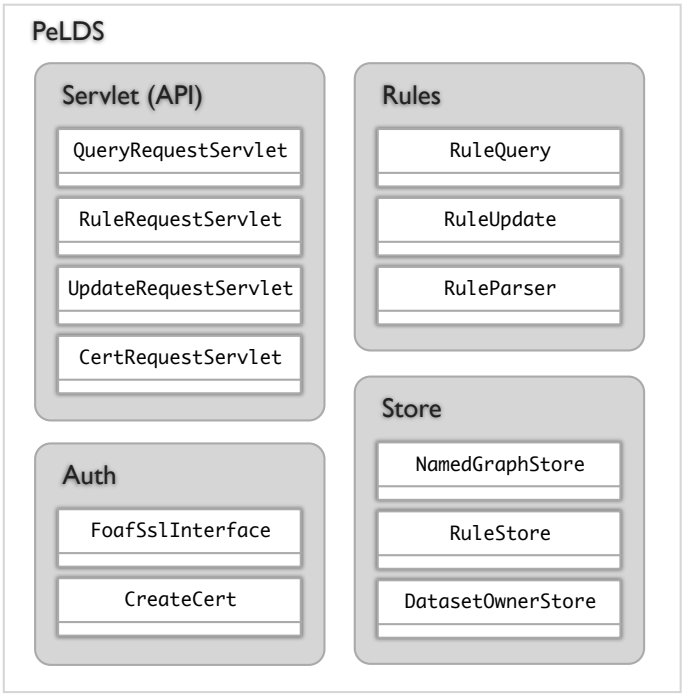

Abbildung 4.8: Klassenstruktur PeLDS - Ausschnitt

4.4.1 HTTP-Anfragen

Wie in Unterabschnitt 4.1.1 erläutert wird PeLDS Anfragen über das HTTP-Protokoll erhalten und beantworten. Innerhalb einer Laufzeitumgebung existieren drei Module, die verschiedene Anfragetypen dediziert behandeln. Die Anfragekomponente *QueryRequestServlet* implementiert den in Unterabschnitt 4.3.3 erläuterten Ablauf. Durch die Komponente zur Veränderung *UpdateRequestServlet* wird die in Unterabschnitt 4.3.2 gezeigte Funktionalität realisiert. Die Zertifizierungs-Komponente *CertRequestServlet* führt den in Unterabschnitt 4.3.5 beschriebenen Vorgang durch. Die Servlet-Klassen erhalten eine HTTP-Anfrage und geben eine HTTP-Antwort, die Nutzdaten enthält, oder einen Fehlercode zurück. Die Funktion der genannten Module ist in den referenzierten Kapiteln genauer beschrieben.

4.4.2 Authentifizierung

Die im Analysekapitel festgestellte Notwendigkeit für passwortfreie Authentifizierung kann wie dort beschrieben mit Hilfe von SSL-Zertifikaten und asymmetrischer Verschlüsselung implementiert werden. Für die Verifikation der Identität der Kommunikationspartner ist es allerdings nach dem SSL-Standard notwendig, dass die verwendeten Zertifikate in einem vorherigen Schritt von einer jeweils vom anderen Kommunikationspartner als vertrauenswürdig eingestuften Instanz digital signiert wurden. Dies stellt jedoch für die für die Eigenschaft der beliebigen und spontanen Vernetzung, wie bei Linked Data und im WWW allgemein üblich, ein Problem dar: Spontane Vernetzung ist nicht möglich, wenn ein vorgelagerter Schritt für den Aufbau der Vertrauenswürdigkeit notwendig ist. Mit dem in Unterabschnitt 2.4.2 beschriebenen Verfahren FOAF+SSL ist jedoch eine Prüfung der Identität des anfragenden Systems ohne vorherigen Aufbau der Vertrauenswürdigkeit über Verifikation der Zertifikate und ohne Passwörter möglich. FOAF-SSL stellt damit eine alternative Möglichkeit zur Herstellung von Vertrauenswürdigkeit innerhalb von SSL dar. Diese Authentifizierung kann damit verwendet werden, um eine spontane Vernetzung im Sinne der Prinzipien von Linked Data durchzuführen, und zusätzlich noch eine Möglichkeit zur Authentifizierung einführen zu können. Zu diesem Zweck enthält PeLDS diese Komponente. Für die Authentifizierung mittels FOAF+SSL stehen in der Komponente *auth* zwei Klassen bereit: Die Schnittstelle zur bestehenden Komponente zur Authentifizierung bietet *FoafSslInterface*. *CreateCert* wird verwendet, um neue Zertifikate zu erstellen, dieser Ablauf ist detailliert in Unterabschnitt 4.3.5 beschrieben und wird hier daher nicht wiederholt.

FoafSslInterface

Diese Klasse führt den Authentifizierungsmechanismus FOAF+SSL durch. Hierzu wird das im angegebenen Kapitel angegebene Verfahren durchgeführt. Hierzu wird auf die vom anfragenden System angegebenen X509-Clientzertifikate zugegriffen. Jedes angegebene Zertifikat wird darauf überprüft, ob es den von FOAF+SSL spezifizierten Anforderungen genügt. Als Rückgabewert wird eine Liste von verifizierten Web IDs an die anfragende Komponente übertragen.

Jede Anfrage kann mehrere Zertifikate enthalten und jedes Zertifikat kann Verweise auf mehrere URIs enthalten. Um eine einfache Verarbeitung der aus den Zertifikaten extrahierten und verifizierten URIs zu ermöglichen, überführt FoafSslInterface die hierarchische Struktur aus Zertifikaten und URIs in eine eindimensionale Liste authentifizierter URIs. Im weiteren Verlauf ist es nicht mehr relevant, von welchem Zertifikat eine URI verifiziert wurde, daher wird diese Information entfernt.

Wurde eine URI erfolgreich authentifiziert, wird diese Tatsache zusammen mit dem verwendeten Zertifikat in einem Zwischenspeicher vorgehalten. Dies beschleunigt später folgende Anfragen stark, bei denen das identische Zertifikat verwendet wird.

In der prototypischen Implementierung wird die bestehende Java-Klasse *DereferencingFoafSslVerifier* für die Verifikation von FOAF+SSL-Identifikatoren verwendet [Sto09]. Damit stellt FoafSslInterface lediglich eine Schnittstelle und Datenkonverter zu dieser bestehenden Klasse dar. In Beispiel 2.11 ist ein Beispiel des genannten Authentifizierungsvorganges angegeben.

4.4.3 Regelauswertung

Die Komponente *Rules* stellt wichtige Funktionalität bereit: Hier werden die vom Nutzer vergebenen Zugriffsrichtlinien ausgewertet und entschieden, ob ein Tripel dem anfragenden Nutzer zur Verfügung gestellt wird. Die einzelnen Module in dieser Komponente umfassen *RuleQuery* zur Auswertung der Zugriffsrichtlinie für die Anfrage an einen RDF-Graphen sowie *RuleParser* für die Übersetzung der Zugriffsrichtlinie in ein Format, das maschinell ausgewertet werden kann.

RuleParser

Das Modul RuleParser stellt Methoden zur Überführung von Regeln vom in Abschnitt 4.2 entworfenen kompakten Format zur Formulierung von Zugriffsrichtlinien bereit. Hierzu wird die textuelle Repräsentation der Regeln im Format PsSF zunächst in eine hierarchische Struktur von Zugriffsrichtlinie, Regel und Prädikaten überführt. In einem zweiten Schritt kann diese abstrakte Struktur in SWRL-Regeln übersetzt werden. Diese indirekte Verarbeitung ist sinnvoll, um die Übersetzung nach SWRL vom der textuellen Repräsentation der Regeln unabhängig zu gestalten.

Diese Übersetzung stellt sich wie folgt dar: Für jede Regel wird eine Instanz des Konzepts Rule aus der in Unterabschnitt 4.1.2 beschriebenen Ontologie in RDF erstellt. Alle in der Regel möglicherweise enthaltenen Variablen werden als SWRL-Variablen angelegt. Dabei werden die Variablen so umbenannt, dass ihre Bezeichner für die aktuelle Regel einzigartig sind, um ungewollte Effekte mit anderen Regeln in der aktuellen Richtlinie zu verhindern. Alle Prädikate (Atome) werden ebenfalls nach SWRL übersetzt und der Rule-Instanz in eine der beiden Prädikatlisten hinzugefügt. Die ebenfalls in der Regel enthaltenen Tripelmuster werden außerdem dem RDF-Graphen hinzugefügt und von der Regel referenziert.

Damit wie erwähnt nach der Auswertung der Regeln die Tripelmuster der Regeln ermittelt werden können, die zur Ausführung gekommen sind, erhält jede Regel noch eine

zusätzliche Konsequenz, die die Eigenschaft *machedRule* in der *Action*-Instanz verändert, sollte die Regel ausgeführt werden. Eine Regel wird in SWRL ausgeführt, wenn alle Prädikate des Voraussetzungsteil den Booleschen Wert *true* ergeben. SWRL ist in Abschnitt 2.3.3 detailliert beschrieben.

RuleQuery

Das Modul *RuleQuery* wertet die vergebene Zugriffsrichtlinie für einen RDF-Graphen aus. Hierzu wird dem Modul der Graph, eine Liste der Bezeichner der erfolgreich authentifizierten Nutzer sowie eine Darstellung der Zugriffsrichtlinie in dem in Abschnitt 4.2 definierten Format angegeben. Als Ausgabe liefert das Modul einen Graphen, der nur noch Tripel enthält, auf die der durch die Liste seiner Bezeichner ausgewiesene Nutzer Zugriffsrechte im Sinne der Zugriffsrichtlinie besitzt.

Die weitere Verarbeitung geschieht in drei Schritten:

(a) Das PeLDS-Vokabular, die Nutzerdaten und die Zugriffsrichtlinie wird in einen RDF-Graphen geladen.

(b) Die in der Richtlinie enthaltenen Regeln und die anhand der vom Nutzer angegebenen Ontologien aus den Daten ableitbaren Schlüsse werden ausgewertet.

(c) Die von den Richtlinien für den anfragenden Nutzer freigegebenen Elemente der Nutzerdaten werden dem Resultatgraphen hinzugefügt.

Bei der Vorbereitung des Modells wird im ersten Schritt unter anderem die Komponente zur Ableitung von Schlüssen anhand der vom Nutzer angegeben Ontologien aus den RDFS/OWL-Beschreibungen konfiguriert. Diese Komponente wird als Reasoner bezeichnet. Ein Reasoner wird ebenfalls benutzt, um SWRL-Regeln auszuwerten. Damit handelt es sich um eine zentrale Komponente. In der prototypischen Implementierung von PeLDS wird Pellet [Cla09] als Reasoner für RDFS, OWL und SWRL verwendet.

Im zweiten Schritt wird die Zugriffsrichtlinie ausgewertet. Dem PeLDS-Vokabular (Unterabschnitt 4.1.2) entsprechend wird eine Instanz des Konzepts Action erstellt, die Verweise auf die in der Richtlinie enthaltenen Regeln enthält. Die Richtlinie ist für diesen Zweck von RuleParser von ihrer kompakten Darstellung in PsSF in die SWRL-Darstellung überführt worden. Action-Instanz sowie Regeln im SWRL-Format werden nun zusammen mit dem Graphen, der die Nutzerdaten enthält, vom Reasoner ausgewertet. Als Ergebnis dieser Auswertung wird klar, welche Regeln für diesen Datensatz und diesen anfragenden Nutzer zutreffen.

Der dritte Schritt legt nun fest, welche Tripel dem Ausgabegraphen hinzugefügt werden können. Hierzu wird zunächst geprüft, ob überhaupt eine Regel zur Ausführung gekommen ist. Ist das nicht der Fall, wird die Bearbeitung abgebrochen. Daraufhin werden alle in den ausgeführten Regeln referenzierten TriplePattern-Instanzen ermittelt. Wie in der Formatdefinition beschrieben, können diese Tripelmuster entweder Tripel, Ressourcen oder Instanzen beschreiben. Je nach Typ werden die Tripel ermittelt, die von den Mustern erfasst werden und dem temporären Ausgabegraphen hinzugefügt.

Algorithmus Anfrage

Der für die Bearbeitung von Anfragen verwendete Algorithmus ist in Algorithmus 1 als Pseudocode dargestellt. Die verwendeten Funktionen werden in Tabelle 4.5 erläutert. Aufbauend auf einem leeren Resultatgraphen (Zeile 1) wird jeder in der Anfrage angegebenen Datensatz und die dazugehörige Richtlinie geladen (Zeilen 3,4). Jede in der Richtlinie enthaltene Regel wird ausgewertet und anhand der im Graph enthaltenen Daten sowie der Nutzeridentität geprüft, ob sie für die aktuelle Anfrage zutrifft (Zeilen 5,6). Nun werden die in der Regel enthaltenen Tripelmuster ermittelt, die auf jedes Tripelmuster zutreffenden Elemente des Graphen ermittelt und diese Tripel dem Resultatgraphen hinzugefügt (Zeilen 7-10). Ähnlich wird mit den in der Regel enthaltenen Konzeptbezeichnern verfahren: Alle im Graph enthaltenen Instanzen des in der Regel bezeichneten Konzepts werden ermittelt und alle Eigenschaften dieser Instanzen dem Resultatgraphen hinzugefügt (Zeilen 11-15). Schließlich wird die in der Anfrage enthaltenen SPARQL-Anfrage auf dem Resultatgraphen ausgeführt und das Resultat dieser Ausführung dem Nutzer bereitgestellt (Zeile 19).

Algorithmus 1 Regelauswertung - Anfrage

Require: A SPARLQ query *sparqlQuery*

```
1:  resultGraph ← {}
2:  for all graphUri ∈ mentionedGraphs(sparqlQuery) do
3:      graph ← loadGraph(graphUri)
4:      for all rule ∈ loadRules(graphUri) do
5:          evaluate(rule, graph, userIdentity)
6:          if hasMatched(rule) then
7:              for all triplePattern ∈ getTriplePatterns(rule) do
8:                  matchedTriples ← findTriples(graph, triplePattern)
9:                  resultGraph ← resultGraph + matchedTriples
10:             for all class ∈ getInstancePatterns(rule) do
11:                 for all instance ∈ findInstances(graph, class) do
12:                     resultGraph ← resultGraph + listProperties(instance)
13: return executeQuery(resultGraph, sparqlQuery)
```

Datenfluss Anfrage

Um die Verarbeitung der Daten und die Arbeitsweise des von Algorithmus 1 beschriebenen Algorithmus zu verdeutlichen, ist in Abbildung 4.9 der Datenfluss einer Anfrage dargestellt. Der geschützte RDF-Graph enthält Ressourcen R, die durch Eigenschaften P mit anderen Ressourcen oder Literalen verbunden sein können. Die Daten sind als RDF-Tripel gespeichert.

Die Richtlinie enthält eine Regel *Regel1*. Diese Regel enthält eine unbekannte Voraussetzung, von der ausgegangen wird, dass sie erfüllt ist. *Regel1* klassifiziert mit Hilfe der beiden Tripelmuster *R1 - P1 - ** sowie ** - P4 - ** einen Teil des geschützten Graphen. Mit *** werden beliebige Werte bezeichnet. Da die Voraussetzung dieser Regel erfüllt ist,

erzeugt die Bearbeitung der Anfrage im ersten Schritt einen temporären Resultatgraphen. Dieser enthält nur noch die Tripel, die durch die Richtlinie zur Herausgabe an den anfragenden Nutzer freigegeben wurden. Beispielsweise ist das Tripel *R1 - P4 - „StringA“* im Resultatgraphen enthalten, da das Tripelmuster ** - P4 - ** auf dieses zutrifft.

Die angegebene Anfrage enthält ebenfalls ein Tripelmuster. *R1 - P4 - ** sucht demnach für ein Tripel, dass mittels der Eigenschaft *P4* die Ressource *R1* näher beschreibt. Im zweiten Schritt wird dieses Muster auf den Resultatgraphen angewendet. Da ein passendes Tripel darin enthalten ist, ergibt die Auswertung der Anfrage das Resultat *R1 - P4 - „StringA“*.

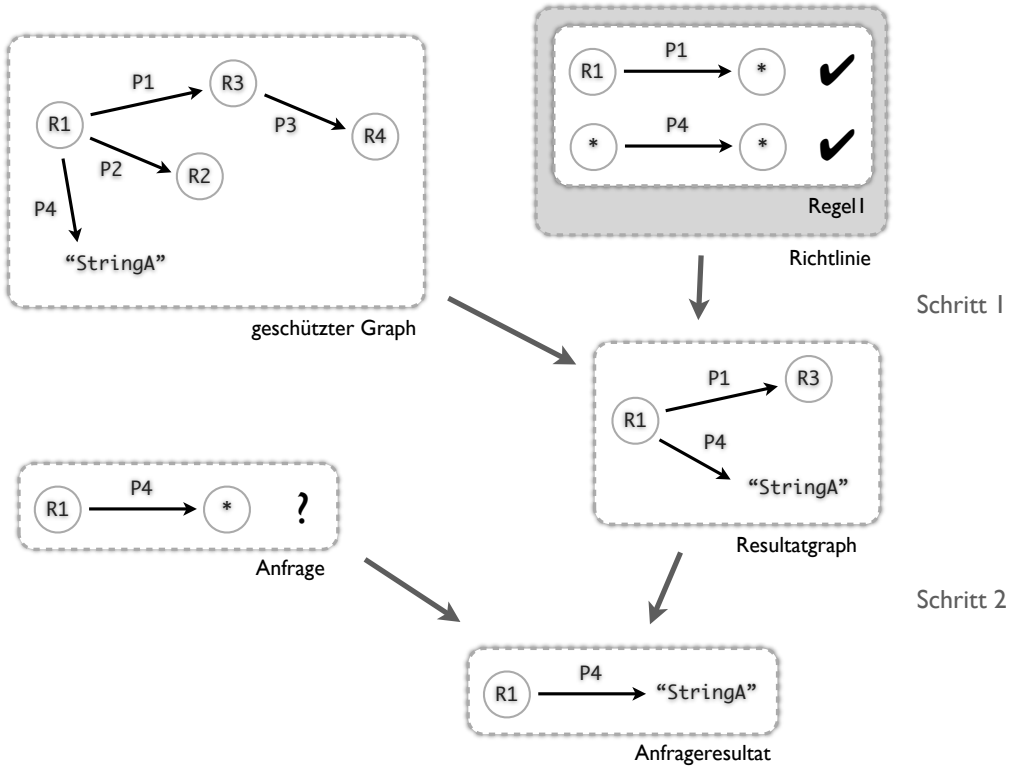

Abbildung 4.9: Anfrage: Datenfluss

Algorithmus Datenveränderung

Für die Autorisierung von verändernden Operationen auf einem RDF-Graph wird der in Algorithmus 2 als Pseudocode angegebene Algorithmus verwendet. Die im Pseudocode verwendeten Funktionen werden in Tabelle 4.5 beschrieben. Der in der Anfrage bezeichnete RDF-Graph wird geladen, und die ebenfalls in der Anfrage angegebene Veränderungsoperation im Format SPARQL/Update wird testhalber ausgeführt. Die von der Verände-

rungsoperation betroffenen Elemente des Graph werden ermittelt (Zeilen 1-4). Analog zu Algorithmus 1 wird die entsprechende Richtlinie geladen und die enthaltenen Regeln evaluiert (Zeilen 5-7). Alle Tripel, auf die enthalten Tripelmuster oder deren Instanzen von einem Instanzenmuster bezeichnet wurden zutreffen, werden aus der Liste der betroffenen Elemente entfernt (Zeilen 8-16). Der Graph wird gespeichert, wenn schließlich kein Element in der Liste der betroffenen Tripel mehr enthalten ist (Zeilen 19-21). Dies ist der Fall, wenn alle Veränderungen durch die Richtlinie autorisiert wurden.

Algorithmus 2 Regelauswertung - Datenveränderung

Require: A SPARQL/Update request $sparqlUpdateQuery$

```
 1: graphUri ← mentionedGraphs(sparqlUpdateQuery)
 2: graph ← loadGraph(graphUri)
 3: executeUpdate(graph, sparqlUpdateQuery)
 4: changedTriples ← getChangedTriples(graph)
 5: for all rule ∈ loadRules(graphUri) do
 6:     evaluate(rule, graph, userIdentity)
 7:     if hasMatched(rule) then
 8:         for all triplePattern ∈ getTriplePatterns(rule) do
 9:             matchedTriples ← findTriples(changedTriples, triplePattern)
10:             changedTriples ← changedTriples − matchedTriples
11:         for all class ∈ getInstancePatterns(class) do
12:             for all instance ∈ findInstances(graph, class) do
13:                 changedTriples ← changedTriples − listProperties(instance)
14: if size(changedTriples) = 0 then
15:     saveGraph(graph)
```

SWRL-Darstellung

Um die Komplexität der Darstellung der vom Nutzer angegebenen Regeln im SWRL-Format zu illustrieren, wird in Listing 4.9 die RDF/XML-Darstellung eines einzigen Prädikates aus der in Listing 4.1 angegebenen Regel gezeigt. Dieses Prädikat

$$foaf : isRelatedTo(http://example.com/horststein, ?actor)$$

stellt ein Eigenschaftsprädikat vom Typ $P(x, y)$ wie in Abschnitt 2.3.3 beschrieben dar. Hierfür wird ein RDF-Objekt vom Typ $swrl : IndividualPropertyAtom$ erzeugt (Zeilen 1,2). Dies enthält nun einen Verweis auf die Eigenschaft $isRelatedTo$ (Zeile 3) und die Argumente für dieses Prädikat (Zeilen 4,5). Dieses Prädikat beschreibt die Existenz eines Tripels mit Subjekt <`http://example.com/horststein`>, Prädikat $foaf : isRelatedTo$ und mit einer Variablen als Objekt.

4.4.4 Datenspeicher

Diese Gruppe von Komponenten (*Store*) kapselt Zugriffe auf die von PeLDS verwendeten Datenbanken für die Verwaltung der gespeicherten RDF-Graphen, der dafür vergebenen

```
<rdf:Description rdf:nodeID="A18">
  <rdf:type rdf:resource="http://www.w3.org/2003/11/swrl#
      IndividualPropertyAtom"/>
  <swrl:propertyPredicate rdf:resource="http://xmlns.com/foaf
      /0.1/isRelatedTo"/>
  <swrl:argument1 rdf:resource="http://example.com/horststein"/
      >
  <swrl:argument2 rdf:resource="http://www.pelds.org/ont#
      variable_actor_851103400"/>
</rdf:Description>
```

Listing 4.9: Regelkompilierung - Prädikat aus Beispiel als SWRL in RDF/XML

Zugriffsrichtlinien sowie für die Speicherung der Bezeichner der Nutzer, die einen speziellen Datensatz erstellt haben. Diese Kapselung ist sinnvoll, um die Art und Weise, wie diese Informationen gespeichert werden, von ihrem Zugriff unabhängig zu machen. In der prototypischen Implementierung wird mittels den hier beschriebenen Komponenten durch die Anfragesprache SQL auf eine relationale Datenbank zugegriffen.

NamedGraphStore

PeLDS verwaltet für beliebig viele Nutzer beliebig viele RDF-Graphen. Die hier beschriebene Komponente bietet eine Möglichkeit zur Verwaltung einer Vielzahl von RDF-Graphen. Graphen können angelegt, geladen und gespeichert werden. Zusätzlich existiert eine Möglichkeit, in allen Graphen nach einer mit einer URI bezeichneten Ressource zu suchen. Als Rückgabewert liefert diese Ressource die Bezeichner aller Graphen, die ein Tripel enthalten, das diese Ressource beschreibt. Dies wird für die Unterstützung der Dereferenzierung, bei der kein Graph-Bezeichner angegeben werden kann, verwendet. Eine explizite Löschung von Graphen ist nicht vorgesehen, Graphen können allerdings geleert werden, so dass sie keine Tripel mehr enthalten.

RuleStore

Nutzer können Zugriffsrichtlinien für die von ihnen erzeugten Graphen vergeben. Diese Richtlinien werden in einer Datenbank gespeichert. RuleStore bietet eine Schnittstelle für die Festlegung und die Veränderung von Richtlinien an. Für die Veränderung von Richtlinien ist es unerheblich, ob bereits eine Richtlinie besteht, da diese nur atomar verändert werden können. Für die Auswertung der Richtlinie kann diese mittels dieser Klasse auch ausgelesen werden.

DatasetOwnerStore

Da die Vergabe von Richtlinien nur dem Nutzer gestattet ist, der den entsprechenden Datensatz bzw. Graphen erstellt hat, bietet diese Komponente eine Schnittstelle für den

Zugriff auf die Datenbank, in der gespeichert wird, welcher Nutzer welchen Datensatz erstellt hat. Hierzu existieren Methoden für die Erzeugung und Entfernung eines Eintrags in dieser Datenbank, sowie für den Abruf der für einen angegebenen Nutzer bestehenden Datensätze und den Abruf des erstellenden Nutzers für einen angegebenen Datensatz.

Prädikat	*Semantik*
mentionedGraphs(q)	In Anfrage q angegebene Datensätze als Liste
loadGraph(u)	Graph u aus der Datenbank
loadRules(u)	Zugriffsrichtlinie für den Graph u als Liste
evaluate(r,g,u)	Richtlinie r für Nutzer u anhand Datensatz g auswerten
hasMatched(r)	*true*, falls Voraussetzungen für Regel r erfüllt
getTriplePatterns(r)	Tripelmuster der Regel r als Liste
findTriples(g,p)	Elemente aus Graph g entsprechend Tripelmuster p
getInstancePatterns(r)	Instanzenmuster der Regel r als Liste
findInstances(g,c)	In g enthaltenen Instanzen der Klasse c
listProperties(o)	Tripel, die Eigenschaften des Objekts o sind
executeQuery(g,q)	Abfrageoperation q auf Graph g ausführen
executeUpdate(g,q)	Veränderungsanfrage q auf Graph g ausführen
getChangedTriples(g)	Bei letzter Veränderung von Graph g geänderte Tripel
size(g)	Anzahl der Tripel im Graph g

Tabelle 4.5: Prädikate der PeLDS-Algorithmen

Kapitel 5

Umsetzung und Evaluation

In Kapitel 4 wurden die notwendigen Eigenschaften einer Umsetzung des Entwurfs „Policy enabled Linked Data Server“ (PeLDS) beschrieben. Die Vorteile einer Implementierung, die diesem Konzept folgt, sind zuvor in Kapitel 3 gezeigt worden. Um die Realisierbarkeit des beschriebenen Konzeptes zu prüfen, wurde eine Software, die das PeLDS-Konzept implementiert, im Rahmen dieser Arbeit als Prototyp entwickelt.

Die Eigenschaften des PeLDS-Prototypen werden zunächst in Abschnitt 5.1 beschrieben: Die Architektur des Prototypen als Java-Webanwendung mit Unterstützung einer relationalen Datenbank als Speichersystem sowie der konkreten Protokolle wird in Unterabschnitt 5.1.1 grafisch dargestellt und erläutert. Anschließend folgt in Unterabschnitt 5.1.2 eine Evaluation des Prototypen im Hinblick auf die Umsetzung der in den Analysekapiteln erhobenen Anforderungen wie etwa funktionaler Eigenschaften, Typ der unterstützten Zugriffsrichtlinien, Datenklassifikation, Datenformat und Datenzugriff. Es zeigt sich, dass diese Anforderungen in vollem Umfang umgesetzt wurden. Die in den Grundlagenkapiteln beschriebenen Kriterien zu Schutz von Datensicherheit und Privatsphäre der Nutzer sowie die Umsetzung des Linked-Data-Prinzips werden ebenfalls für den Prototypen überprüft. Diese Kriterien gelten ebenfalls als erfüllt. Unterabschnitt 5.1.3 beschreibt die für die Bewertung des Prototypen durchgeführten Messungen der Systemleistung einerseits für die Veröffentlichung von Daten und andererseits für die Veränderung von Daten. Dabei wird auch ein zum Teil mit PeLDS vergleichbares System aus dem SPARQL-Server Joseki und dem RDF-Speichersystem TDB in die Bewertung mit einbezogen und dessen Leistungsdaten mit dem Prototypen verglichen. Der Vergleich ergibt einen akzeptablen Mehraufwand durch die Auswertung der Zugriffsrichtlinien. Zusätzlich zeigt Unterabschnitt 5.1.4 die Umsetzung eines Zugriffs auf PeLDS mit Hilfe der RDF-Bibliothek ARC und der Skriptsprache PHP. In Abschnitt A.5 ist eine Installations- und Konfigurationsanleitung für PeLDS enthalten.

Um einen Teil der Leistungsfähigkeit von PeLDS einfacher demonstrieren zu können, wurde zusätzlich eine Anwendung, die dem im Analysekapitel beschriebenen Konzept des verteilten Adressbuchs folgt, im Rahmen dieser Arbeit auf der Basis des PeLDS-Prototypen entwickelt. Diese Anwendung wird in Abschnitt 5.2 erläutert. Hierfür wird zunächst ein Überblick über die Gesamtarchitektur der Anwendung in Unterabschnitt 5.2.1 angegeben.

Anhand der in Unterabschnitt 3.1.2 beschriebenen Anwendungsfälle wird deren Umsetzung mit Hilfe von Bildschirmfotos der Implementierung in Unterabschnitt 5.2.2 gezeigt. Abschließend wird die Anwendung wie auch der PeLDS-Prototyp in Unterabschnitt 5.2.3 evaluiert, eine Installations- und Konfigurationsanleitung findet sich in Abschnitt A.6.

5.1 Prototyp „Policy enabled Linked Data Server“

Der im Rahmen dieser Arbeit entstandene Prototyp von PeLDS implementiert die in Abschnitt 4.4 beschriebenen Komponenten und bietet alle in Abschnitt 4.3 gezeigten Operationen für den Zugriff und die Veränderung von RDF-Daten. Im Folgenden wird zunächst eine Übersicht über die Architektur von PeLDS gegeben. Daraufhin wird der Prototyp im Bezug auf Sicherheit und Privatsphäre, der Umsetzung der Kriterien des Semantic Web und Linked Data, den möglichen Richtlinientypen und Möglichkeiten zur Datenklassifikation entsprechend den in den Grundlagenkapiteln angegeben Kriterien bewertet. Es folgt eine Einschätzung der Systemleistung mit der Auswertung verschiedener Testergebnisse sowie die Implementierung einer Interaktion mit PeLDS in der Skriptsprache PHP.

Der genannte Prototyp steht auf der PeLDS-Website unter `http://www.pelds.org` zusammen mit einer Installationsanleitung als Softwarepaket zur Verfügung. Auf eine Installation des Prototypen kann unter `https://store.pelds.org` zugegriffen werden. Im Folgenden wird der Begriff „PeLDS“ auch dafür benutzt, um die prototypische Implementierung zu bezeichnen.

5.1.1 Architektur

Die Einbettung von PeLDS in existierende Systeme ist in Abbildung 5.1 grafisch dargestellt. PeLDS in kompilierter Form als *PeLDS Servlet* wird innerhalb des *Tomcat Servlet Container* ausgeführt, der wiederum eine Anwendung ist, die innerhalb des *Java Runtime Environment* läuft. Für die persistente Speicherung von Daten kommuniziert PeLDS über das JDBC-Protokoll mit einem relationalen Datenbanksystem *Relational DB Server*. PeLDS kann via HTTP oder HTTPS RDF-Informationen von anderen Systemen laden. Der Zugriff auf PeLDS erfolgt von Anwendungen aus unter Verwendung einer *RDF / SPARQL Library*, also einer Sammlung von Funktionen zur Behandlung von RDF-Daten und für die Anfrage mittels der Anfragesprache SPARQL an einen PeLDS-Server. Der Client, der Anfragen stellt, wird via FOAF+SSL authentifiziert. Alle Komponenten mit Ausnahme von *PeLDS Servlet* existieren bereits und sind frei verfügbar. Die Darstellung der Systemgrenzen (*System delimiter*) stellt die übliche Abgrenzung der Computersysteme dar, PeLDS und alle Komponenten sind auf einem System installiert, die Anfragebibliothek auf einem anderen System.

5.1.2 Evaluation

PeLDS bietet eine Komponente zur Verwaltung von Daten, die einerseits die Techniken des Semantic Web wie etwa RDF-Datenmodell und Datenzugriff unterstützt und andererseits

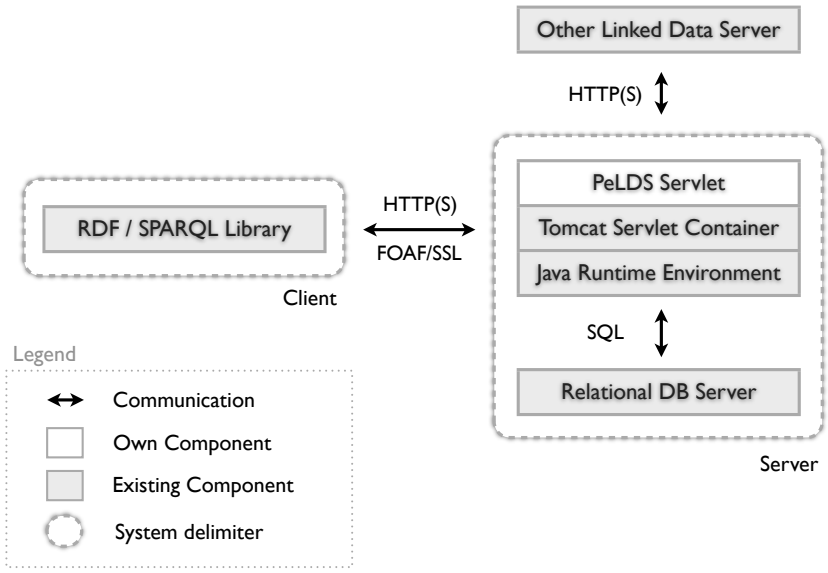

Abbildung 5.1: PeLDS - Architektur

den Schutz von RDF-Daten mittels Authentifikation und der Vergabe von Zugriffsrichtlinien erlaubt. In der Analyse der Anforderungen an diese Komponente wurden verschiedene Aspekte einer Lösung identifiziert und beschrieben. Die prototypische Implementierung des entworfenen Konzepts, welches aus diesen in Abschnitt 3.3 beschriebenen Anforderungen entstandenen ist, wird im Folgenden auf die Umsetzung der Anforderungen überprüft. Diese Prüfung betrifft a) die funktionalen Anforderungen, b) die Möglichkeiten der Vergabe von Zugriffsrichtlinien, c) die Umsetzung der Datenklassifikation, d) des verwendeten Datenformats und der dafür möglichen Zugriffsmethoden, e) die vom System angebotene Unterstützung der Datensicherheit sowie der Privatsphäre der Nutzer und f) die Integration der Umsetzung in das Linked-Data-Umfeld. Die Ergebnisse der Bewertung sind in Tabelle 5.1 zusammengefasst.

Funktionale Anforderungen

Während der Problemanalyse wurden die in Unterabschnitt 3.3.1 aufgeführten Anwendungsfälle für die Serverkomponente erkannt. Es wird nun gezeigt, wie diese im Entwurf berücksichtigten Anwendungsfälle in der Implementierung von PeLDS umgesetzt werden.

- Authentifizierung - Nutzer können sich am Server mit Hilfe von digitalen Zertifikaten anmelden. Diese Anmeldung wird mittels FOAF+SSL verifiziert. Eine Eingabe von Benutzername oder Passwort oder vorherige manuelle Konfiguration ist nicht notwendig. Siehe dazu auch 4.4.2.

- Veröffentlichung - Der Nutzer kann in einem ersten Schritt seine Richtlinie an den Server senden. Der Server prüft diese Richtlinie, erstellt bei erfolgreicher Prüfung einen neuen Graph mit dieser Richtlinie (4.3.1). In einem folgenden Schritt werden nun die Daten an den Server geschickt und dort unter dem Schutz der Richtlinie gespeichert (4.3.2).
- Veränderung - Daten können mittels der Veränderungsoperation (4.3.2) geändert werden. Nur der Nutzer, der den Datensatz erstellt hat, kann die dafür gültige Richtlinie verändern (4.3.1). Der Nutzer kann innerhalb seiner Richtlinie Dritten die Rechte zur Veränderung von Daten einräumen.
- Entfernung - Daten können mittels der Veränderungsoperation (4.3.2) auch gelöscht werden. Richtlinien können mit der Operation zur Verwaltung von Richtlinien gelöscht werden (4.3.1).
- Anfrage - Daten können mit der Abrufoperation angefragt werden (4.3.3). Als Anfragesprache wird SPARQL verwendet. Mit Hilfe der im Datensatz vorhandenen Ontologien ableitbare Informationen werden durch den Reasoner erzeugt und können als Teil der Basisdaten ebenfalls abgerufen werden.

Alle in der Analyse genannten Anwendungsfälle für die Serverkomponente wurden umgesetzt und stehen in der prototypischen Implementierung zur Verfügung.

Zugriffsrichtlinien

Die im Rahmen dieser Arbeit entworfene Richtliniensprache PsSF (siehe Abschnitt 4.2) basiert auf der Prädikatenlogik. Dadurch besitzt sie eine sehr hohe Ausdrucksmächtigkeit und kann für eine Vielzahl von Szenarien eingesetzt werden. Die in Unterabschnitt 2.3.1 genannten Richtlinientypen können sämtlich in PsSF dargestellt werden. Die folgende Auflistung gibt Hinweise darauf, wie die verschiedenen Richtlinientypen mit PsSF und PeLDS umgesetzt werden können.

- *Discretionary Access Control (DAC)* - Die geforderte Unterscheidung zwischen Nutzern und Datenobjekten fällt in PeLDS leicht. Nutzer werden anhand ihrer URI identifiziert. Datenobjekte sind die Elemente der jeweiligen RDF-Graphen in der gewünschten Granularität. Nutzer können durch Veränderung der Zugriffsrichtlinien anderen Nutzern und Gruppen Zugriff auf die Datenobjekte geben. Automatische Propagation von Rechten ist in PeLDS zwar nicht vorgesehen, kann aber über eine Zugriffsrichtlinie realisiert werden. In Abschnitt A.1 ist ein Beispiel für die Umsetzung von DAC in PeLDS gegeben. Damit wird die Umsetzung von DAC in PeLDS als möglich betrachtet.
- *Mandatory Access Control (MAC)* - PeLDS ermöglicht die Vergabe einer Zugriffsrichtlinie für jeden gespeicherten RDF-Graphen, falls Nutzer, die nicht der Ersteller dieses Graphen sind, auf diesen zugreifen wollen. Eine für jedes Objekt notwendige Sicherheitsklassifikation lässt sich in RDF leicht darstellen. Beispielsweise kann

jeder Instanz des Konzepts Dokument eine Eigenschaft *securityClassification* mit einem Verweis auf ein Klassifikationsobjekt hinzugefügt werden. Das von PeLDS unterstützte RDFS ermöglicht eine hierarchische Klassifikation sowie den Zugriff auf dieser Hierarchie durch Regeln. Zugriffsrechte auf Elemente der Hierarchie können auch Nutzern zugeordnet und entsprechend ausgewertet werden. Die Umsetzung eines System nach dem Konzept der MAC auf Basis von PeLDS ist damit möglich.

- *Role-based Access Control (RBAC)* - Analog zur MAC können Nutzern auch Rollenbezeichnungen innerhalb des RDF-Graphen zugeordnet und von PeLDS-Richtlinien ausgewertet werden. Die einer Rolle zugeordneten Rechte lassen sich ebenfalls in einer Zugriffsrichtlinie beschreiben. Damit ist die Umsetzung eines RBAC-basierten System in PeLDS möglich.

Die Untersuchung der verschiedenen Richtlinientypen hat ergeben, dass die in PeLDS verwendete Richtliniensprache PsSF die Umsetzung der verbreitetsten Richtlinientypen möglich macht. Die Ausdrucksmächtigkeit von PsSF wird damit für die Beschreibung der genannten Richtlinientypen als ausreichend angesehen.

Datenklassifikation

Die Elemente der in PeLDS abgelegten RDF-Graphen können mit Hilfe der in Unterabschnitt 4.2.2 definierten Prädikate auf verschiedene Art und Weise klassifiziert werden. Elemente können dafür auf den drei in Unterabschnitt 2.3.2 genannten Ebenen ausgewählt werden.

(a) Modellebene - Einzelne Tripel der RDF-Graphen können durch die Angabe von konkreten Werten für Subjekt, Prädikat und Objekt der Tripel klassifiziert werden. Um eine umfassendere Auswahl von Tripeln zu erlauben, können die Werte auch durch Wildcards ersetzt werden.

(b) Semantische Ebene - Die im RDF-Graphen enthaltenen Instanzen der mittels RDFS oder OWL definierten Konzepte können über ihre Zugehörigkeit zu diesen Konzepten von Richtlinien selektiert werden. Hierzu ermittelt der in PeLDS enthaltene Reasoner vor der Auswertung diese Zugehörigkeit. Über das Prädikat zur Auswahl von Instanzen werden alle Tripel erfasst, die einer dieser Instanzen als Eigenschaft zugeordnet sind.

(c) Ressourcenebene - Elemente können anhand ihrer Bezeichner in URL-Form klassifiziert werden. Dies stellt einen Spezialfall der Klassifikation über die Modellebene dar, bei der nur die Subjekte der Tripel, die den Ressourcenbezeichner enthalten, durch einen Wert vorgegeben sind. Dies stellt damit einen Spezialfall des ersten Punktes dar, bei dem die Werte von Prädikat und Objekt durch Wildcards offen gelassen werden.

Elemente können damit nach allen für RDF-Daten identifizierten Klassifikationsmethoden in PeLDS klassifiziert und damit geschützt werden.

Datenformat und Zugriff

Als Datenmodell verwendet PeLDS RDF. Die RDF-Graphen werden in einer von NG4J verwalteten relationalen Datenbank persistent auf einem Sekundärspeicher des Servers abgelegt. Jeder Nutzer kann beliebig viele Graphen auf dem Server ablegen, und PeLDS kann prinzipiell beliebig viele Nutzer verwalten. Als Bezeichner für Graphen werden URIs verwendet. Der Server bietet eine HTTP-Schnittstelle für die Veränderung von Daten wie in Unterabschnitt 4.3.2 beschrieben an. Diese Schnittstelle unterstützt die Sprache SPARQL/Update für die Beschreibung von Änderungen an RDF-Graphen. Der Server stellt eine Schnittstelle für die Abfrage für Teilmengen der veröffentlichten RDF-Graphen bereit. Für den Abruf von Elementen der gespeicherten RDF-Graphen verwendet PeLDS die Sprache SPARQL. Die angebotenen Schnittstellen für den Abruf von Daten wurden in Unterabschnitt 4.3.3 detailliert beschrieben. Die in Unterabschnitt 3.3.3 identifizierten Anforderungen an Datenformat und Datenzugriff gelten damit als umgesetzt.

Sicherheit und Privatsphäre

Die in Unterabschnitt 2.1.1 aufgeführten Kriterien für die Sicherheit eines Computersystems werden in der folgenden Auflistung bezüglich ihres Zutreffens auf PeLDS geprüft.

(a) Richtlinien - PeLDS setzt die vom Nutzer angegebene explizite und formal definierte Sicherheitsrichtline im entworfenen Format PsSF durch. Das System kann für jeden Nutzer und jedes atomare Datenelement entscheiden, ob eine Aktion zulässig ist. Details dieser Richtlinien wurden in diesem Abschnitt bereits erläutert.

(b) Klassifizierung - PeLDS unterstützt die Klassifikation von Daten auf den im Datenmodell RDF relevanten Ebenen. Diese Ebenen umfassen Klassifikation von Tripeln, Ressourcen und Instanzen. Die weiterführende Bewertung der Klassifizierung wurde in diesem Abschnitt durchgeführt.

(c) Authentifizierung - Mittels der in Unterabschnitt 2.4.2 und Unterabschnitt 4.4.2 beschriebenen Authentifizierung werden die anfragenden Nutzer in PeLDS mit den dort beschriebenen Einschränkungen authentifiziert.

(d) Verantwortlichkeit - PeLDS führt für jeden Zugriff ein Logbuch, mit dem jede Aktion auf einem RDF-Graphen einem Nutzer zugeordnet werden kann. Der Schutz dieses Logbuchs liegt außerhalb des Einflussbereichs von PeLDS und muss durch nachgelagerte Systeme sichergestellt werden. Üblicherweise werden die Zugriffsrechte auf die Datei, die das Logbuch enthält so gewählt, dass nur ein Systemadministrator Zugriff auf diese Datei hat.

(e) Garantie - Die prototypische Implementierung von PeLDS enthält *keine* Methoden für die Prüfung der Erfüllung der genannten Punkte. Die im Rahmen dieser Arbeit durchgeführten Tests des Prototypen legen es jedoch nahe, dass eine solche Methode leicht implementierbar wäre. Der Zugriff auf das nachgelagerte Datenbanksystem, in dem die von PeLDS verwalteten RDF-Graphen gespeichert werden, liegt außerhalb der Kontrolle von PeLDS.

(f) Fortlaufender Schutz - Durch die Möglichkeit der Installation von PeLDS auf vom Nutzer ausgewählten Computersystemen gibt es keine Möglichkeit, die Integrität der PeLDS-Installation zu gewährleisten. Die Integrität der Installation könnte jedoch durch spezielle Zertifizierungen von Dritten geprüft werden.

Trotz der Einschränkungen der fehlenden Garantie sowie des Nichtvorhandenseins von Möglichkeiten zum fortlaufenden Schutz der Garantie kann die prototypische Implementierung von PeLDS als sicheres System bezeichnet werden. Eine Weiterentwicklung dieser Implementierung könnte die fehlenden Punkte umsetzen.

PeLDS als anwendungsunabhängiges System zur durch Zugriffsrichtlinien geschützten Ablage von RDF-Graphen kann eingesetzt werden, um Anwendungen zu entwickeln, die die Privatsphäre der Nutzer schützen. Die Bewertung von PeLDS nach denen in Unterabschnitt 2.1.2 genannten Kriterien im Bezug auf die Möglichkeiten zum Schutz der Privatsphäre der Nutzer wird durchgeführt, da PeLDS wichtige Grundlagen für Anwendungen für die Verwaltung sensibler Daten bieten kann.

(a) Klassifizierung - siehe gleichnamigen Punkt in Abschnitt 5.1.2 sowie Abschnitt 5.1.2. PeLDS unterstützt die Klassifikation von Daten auf den im Datenmodell RDF relevanten Ebenen. Der Punkt wird als erfüllt betrachtet.

(b) Zugriffskontrolle - siehe Punkt „Richtlinien“ Abschnitt 5.1.2 sowie Abschnitt 5.1.2. PeLDS setzt die vom Nutzer angegebene explizite und formal definierte Sicherheitsrichtline im entworfenen Format PsSF durch. Der Punkt wird ebenfalls als erfüllt betrachtet.

(c) Richtlinienkontrolle - Richtlinien für den Zugriff auf RDF-Graphen werden innerhalb von PeLDS verwaltet und von diesem nur an den Nutzer ausgeliefert, der den Datensatz angelegt hat, für den die Richtlinie gilt. Die Richtlinie kann nur von dem Nutzer verändert und angelegt werden, der den entsprechenden Graphen angelegt hat. Dritte haben keinerlei Zugriff auf die Richtlinie.

(d) Verwendung von Daten durch Dritte - Wurden Elemente eines RDF-Graphen an einen berechtigten anfragenden Nutzer ausgeliefert, ist keine Kontrolle darüber mehr möglich. Die Verwendung von entsprechenden Verschlüsselungen und Autorisierungssystemen würde die für die Einbindung von PeLDS in das Linked-Data-Umfeld notwendige Kompatibilität mit anderen Systemen verhindern und ist daher nicht implementiert.

Die für den Schutz der Privatsphäre der Nutzer notwendigen Maßnahmen können durch ein auf PeLDS basiertes System einfacher realisiert werden, als dies mit anderen Anwendungen möglich wäre. In PeLDS kann ein differenziertes Zugriffsschema definiert und durchgesetzt werden.

Linked Data-Integration

PeLDS kann die Prinzipien von Linked Data wie in Unterabschnitt 2.2.3 beschrieben umsetzen. Dies erfordert jedoch die Kooperation der Nutzer. Die von den Nutzern vergebenen

Bezeichner für die Graphen sowie für die Ressourcen in diesen Graphen sind frei wählbar. Sinnvollerweise werden die Bezeichner jedoch in Form von dereferenzierbaren HTTP-URLs vergeben. Ist die von PeLDS unterstützte Operation für den Datenabruf mittels Dereferenzierung wie in Unterabschnitt 4.3.4 beschrieben korrekt konfiguriert, können andere Systeme nur anhand der URL der Ressource die für diese Systeme freigegebenen Daten, die diese Ressource beschreiben, abrufen. Damit erfüllen in PeLDS veröffentliche RDF-Daten die Linked-Data-Eigenschaften. Die in Linked Data mögliche Kombination von verschiedenen Datenquellen wird durch die verwendete Authentifizierung weiter unterstützt, da FOAF+SSL keine Konfiguration für den Datenaustausch zwischen zwei beliebigen Systemen erfordert.

Bereich	*Anforderung*	*Ergebnis*	
Funktionale Anforderungen	Authentifizierung	✓	FOAF+SSL
	Veröffentlichung	✓	PeLDS
	Veränderung	✓	PeLDS
	Löschung	✓	PeLDS
	Anfrage	✓	PeLDS
Richtlinientyp	Discretionary	✓	Richtlinie möglich
	Mandantory	✓	Richtlinie möglich
	Role-based	✓	Richtlinie möglich
Datenklassifikation	Modellebene	✓	PsSF
	Semantische Ebene	✓	PsSF
	Ressourcenebene	✓	PsSF
Datenformat und -zugriff	Datenmodell	✓	RDF
	Zugriffssprache	✓	SPARQL
	Zugriffsprotokoll	✓	HTTP
Sicherheit	Richtlinien	✓	PsSF
	Klassifizierung	✓	PsSF
	Authentifizierung	✓	FOAF+SSL
	Verantwortlichkeit	✓	Logdatei
	Garantie	×	
	Fortlaufender Schutz	×	
Privatsphäre	Klassifizierung	✓	PsSF
	Zugriffskontrolle	✓	PeLDS
	Richtlinienkontrolle	✓	PeLDS
	Datenverwendung	×	
Linked Data	HTTP-URLs	✓	Vergabe möglich
	Dereferenzierung	✓	PeLDS
	Verweise	✓	Vergabe möglich

Tabelle 5.1: PeLDS - Übersicht über die Bewertungsergebnisse

5.1.3 Systemleistung

Der Vergleich von PeLDS mit anderen Systemen ist nicht ohne Weiteres möglich, die einzige in der Literatur [RFJ07] beschriebene vergleichbare Implementierung ist nicht verfügbar. PeLDS implementiert eine RDF-Datenbank, auf die mittels SPARQL zugegriffen werden kann. Außerdem bestimmt PeLDS die aus den abgelegten Daten ableitbaren Informationen mittels eines Reasoners und stellt diese abgeleiteten Informationen für Anfragen und Veränderungen bereit. Schließlich wertet PeLDS die angegebene Zugriffsrichtlinie aus und authentifiziert die anfragenden Nutzer. Ein System, dass zumindest Teile dieser Anforderungen realisiert, kann mittels des SPARQL-Servers Joseki [Hew09b] konstruiert werden. Joseki unterstützt jedoch die Vergabe von Zugriffsrichtlinien nicht. Für den Test wurde eine Installation von Joseki mit der Jena-Komponente TDB [Hew09a] verbunden, die einen RDF-Datenspeicher ähnlich des in PeLDS verwendeten NG4J [BCH09] realisiert. Für die Bestimmung der ableitbaren Informationen wurde Joseki zusätzlich mit dem Reasoner Pellet [Cla09] verbunden. Außer Konkurrenz wurden die ausgeführten Tests auch noch auf einer Joseki-Installation ohne Reasoner ausgeführt. Ein Vergleich der getesteten Systeme ist in Tabelle 5.2 dargestellt. Die durchgeführten Tests sollen den durch die Auswertung von Zugriffsrichtlinien entstehenden Mehraufwand bewerten.

Funktion	PeLDS	Joseki & Pellet	Joseki
RDF-Speicherung	✓	✓	✓
SPARQL-Anfragen	✓	✓	✓
SPARQL/Update	✓	✓	✓
HTTP-Schnittstelle	✓	✓	✓
OWL/RDFS Reasoning	✓	✓	×
Authentifizierung	✓	×	×
Zugriffsrichtlinien	✓	×	×
Dereferenzierung	✓	×	×

Tabelle 5.2: Testsysteme - Vergleich der angebotenen Funktionen

Testsystem

Alle Tests wurden auf einem MacBook unter Mac OS X ausgeführt. Dieser Computer enthält einen 2,2 GHz Intel Core 2 Duo 64-Bit-Prozessor und 2 GB Arbeitsspeicher. Sowohl Joseki als auch PeLDS werden von der Java-Laufzeitumgebung ausgeführt. Als Datenbank für PeLDS kommt MySQL zum Einsatz, PeLDS selbst wird innerhalb des Servlet-Containers Apache Tomcat ausgeführt. Die Java-Laufzeitumgebung ist für eine maximale Ausnutzung des verfügbaren Arbeitsspeichers konfiguriert. In Tabelle 5.3 sind die Versionsnummern der beteiligten Pakete aufgeführt. Um die Funktion von Joseki in PeLDS möglichst weit nachzubilden, wurde in PeLDS die in Listing 5.1 angegebene Zugriffsrichtlinie installiert, die auch unangemeldeten Nutzern lesenden und schreibenden Zugriff auf den gesamten Datensatz erlaubt. Dies führt dazu, dass sich PeLDS genauso wie Joseki verhält.

Softwarepaket	*Version*
Mac OS X	10.6.1
Java JRE	1.6.0
Joseki	3.4.0
TDB	0.8.2
Pellet	2.0
MySQL	5.4.3
Tomcat	6.0

Tabelle 5.3: Testsystem - Softwarepakete

```
1 AllQueryRule:   QueryAction(?action)  => permit_triple(*,*,*);
2 AllUpdateRule:  UpdateAction(?action) => permit_triple(*,*,*);
```

Listing 5.1: Leistungstest - PeLDS-Richtlinie

Testdaten

Als Testdaten wurden vom Testdatengenerator des Berlin SPARQL Benchmark (BSBM) [BS09] erzeugte Tripel verwendet. Vorteilhaft hierbei ist, dass anhand eines gewählten Skalierungsfaktors praktisch beliebig viele Tripel erzeugt werden können. Für die Tests wurden Testdaten mit den Skalenfaktoren 1 bis 8 erzeugt. Für die Skalenfaktoren 1/4 und 1/2 wurde der Testdatensatz mit dem Skalierfaktor 1 jeweils halbiert, da der Generator keine kleineren Datensätze erzeugen kann.

Testresultate Anfrageoperation

Bei der Prüfung der Anfrageleistung wurde für jeden der erzeugten Datensätze ein Testlauf mit allen drei Systemen (PeLDS, Joseki mit Pellet sowie Joseki) durchgeführt. Hierzu wurde der entsprechende Datensatz in die jeweiligen Server importiert und die in Listing 5.2 dargestellte sehr einfache SPARQL-Anfrage mindestens drei Mal ausgeführt und die dafür benötigte Zeit gemessen. Die Anfrage ergibt als Resultat alle im jeweiligen Datensatz vorhandenen Tripel und führt dazu, dass der eventuell vorhandene Reasoner alle aus den Daten ableitbaren Tripel errechnen muss (engl. „full closure“), um diesen Effekt mit einzubeziehen. Als Resultat wurde die geringste Ausführungszeit aus allen Testläufen für diesen Datensatz verwendet, um Cachingvorteile mit in das Resultat einzubeziehen. Die Resultate der Testläufe sind in in Abbildung 5.2 als Scatterplot zwischen Anzahl der Tripel auf der x-Achse sowie der Verarbeitungszeit in Sekunden auf der y-Achse dargestellt. In diesem Plot ist auch eine Approximation der Werte und der jeweilige R^2-Wert des Anpassungstests der Messwerte an die Approximationsfunktion mit der Methode der kleinsten Fehlerquadrate dargestellt. Bei der Approximationsfunktion handelt es sich um ein Polynom zweiten Grades. Der verwendete Anpassungstest wird in Abschnitt A.2 näher beschrieben. Die einzelnen Messwerte sind in Tabelle A.1 in Sekunden angegeben.

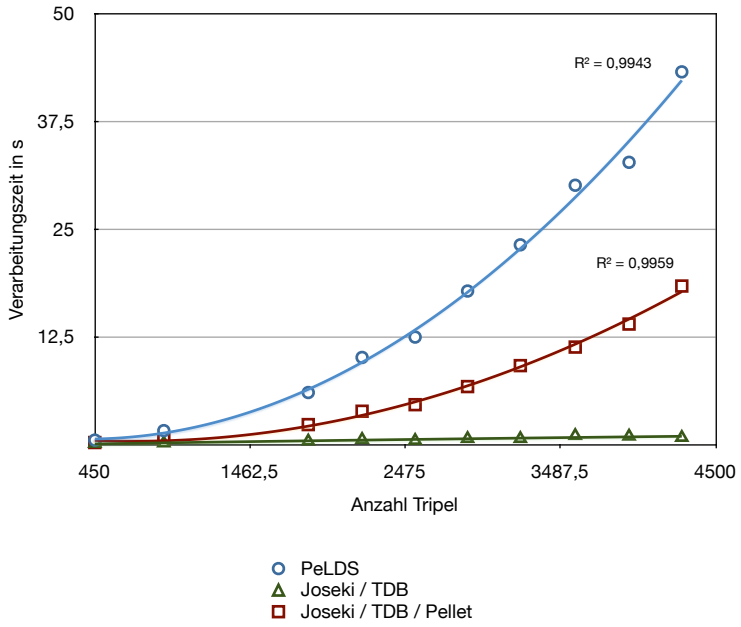

Abbildung 5.2: Anfrageleistungstest - Grafik

```
1 SELECT ?s ?p ?o
2 WHERE { ?s ?p ?o . }
```

Listing 5.2: Anfrageleistungstest - SPARQL-Anfrage

Testresultate Veränderungsoperation

Um die Leistung der Datenveränderung zu prüfen, wurde ebenfalls für alle erzeugten Datensätze ein Testlauf mit PeLDS und Joseki durchgeführt. Hierzu wurde der entsprechende Datensatz in die jeweiligen Server importiert und die dafür benötigte Zeit gemessen. Der Importvorgang stellt eine Veränderungsoperation dar und wurde je Datensatz drei Mal ausgeführt. Resultat ist die geringste Ausführungszeit aus allen Testläufen für diesen Datensatz. Die Resultate der Testläufe sind in in Abbildung 5.3 als Scatterplot analog der in Abschnitt 5.1.3 beschriebenen Methode dargestellt. Die einzelnen Messwerte sind in Tabelle A.2 in Sekunden angegeben.

Interpretation

Bei dem durchgeführten Test sollte die für die Auswertung von Zugriffsrichtlinien benötigte Zeit und ihre Auswirkung auf die Antwortzeit des Gesamtsystems bestimmt werden. Daher wurde die SPARQL-Anfrage einerseits so einfach wie möglich gehalten, andererseits führt diese Anfrage auch dazu, dass der Reasoner wie beschrieben alle ableitbaren Tripel erzeugt.

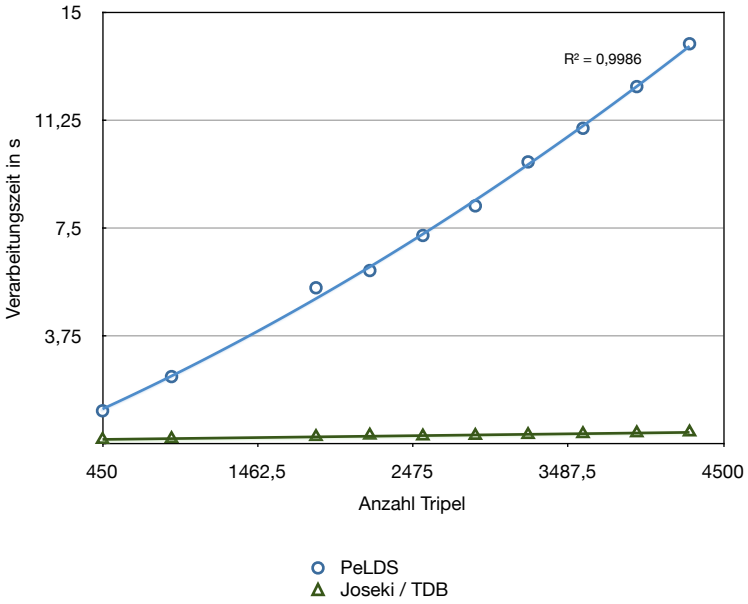

Abbildung 5.3: Veränderungsleistungstest - Grafik

Dies erklärt die große Differenz zwischen der Joseki-Installation ohne Reasoner und den beiden anderen getesteten Systemen.

Der relevante Vergleich zwischen der Joseki-Instanz mit Pellet-Unterstützung und PeLDS ergab bei der Anfrageoperation einen Mehraufwand in Höhe des linearen Faktors 2,5. Für eine prototypische Implementierung im Vergleich zu einer optimierten Lösung ist diese Differenz durchaus akzeptabel. Die für den Test verwendete Zugriffsrichtlinie führt dazu, dass alle im Datensatz enthaltenen Tripel in den Resultatgraphen kopiert werden müssen, was den maximalen Aufwand für die Regelauswertung darstellt.

Bei der Veränderung wird klar, dass die Auswertung der Zugriffsrichtlinie für die Datenveränderung im Vergleich zu Joseki, bei dem keinerlei Prüfung der eingefügten Daten erfolgt, durchaus substanziell ist. Jedes veränderte Tripel wird einer Prüfung unterzogen, dies erklärt den angefallenen Mehraufwand. Eine Verarbeitungszeit von beispielsweise 30s für ca 4000 Tripel wird jedoch für die prototypische Implementierung als akzeptabel angesehen. Für die in vielen Anwendungen auftretenden Datensätze mit weit weniger als 500 Tripeln wird eine Antwortzeit kleiner als eine Sekunde erreicht, dies ermöglicht interaktive Anwendungen [DWT82].

Wichtig ist, dass der Anpassungstest auf eine Approximation mit einem Polynom zweiter Ordnung für die Messergebnisse von PeLDS einen sehr guten R^2-Wert von 0,99 ergibt. Es kann vermutet werden, dass der Zusammenhang zwischen Anzahl der verarbeiteten Tripel und der Verarbeitungszeit einer quadratischen Funktion folgt. Optimierungen der Regelauswertung in PeLDS können den genannten Mehraufwandsfaktor im Vergleich zu Joseki/Pellet möglicherweise reduzieren.

Die gemessenen Leistungsdaten sind für die Implementierung der Beispielanwendung ausreichend, die dort je Graph gespeicherte Anzahl von Tripeln liegt weit unter 1000, damit ist eine interaktive Bedienung der Anwendung mit direkten Zugriffen auf PeLDS möglich.

5.1.4 RDF / SPARQL - Bibliothek

Anwendungen auf PeLDS-Basis, die in der Skriptsprache PHP implementiert werden, können über die Bibliothek ARC [Now09] auf PeLDS zugreifen. Damit die von PeLDS unterstützte Authentifizierung über FOAF+SSL durchgeführt werden kann, wurde ein Patch für ARC erstellt, bei den ARC-Entwicklern eingereicht und von diesen akzeptiert. In aktuellen ARC-Versionen ist die Unterstützung für die bei FOAF+SSL notwendige Verwendung von SSL-Clientzertifikaten bereits enthalten. In Listing 5.3 ist ein vollständiges einfaches Beispiel für die Verwendung von ARC für den Zugriff auf Daten, die von einer PeLDS-Installation verwaltet werden, dargestellt: Aus dem RDF-Graph <`http://example.com/graph`> werden alle Tripel ausgelesen und ausgegeben. Hierzu wird ARC lediglich die URL der SPARQL-Schnittstelle von PeLDS angegeben (Zeile 4) und ein im PEM-Format kodiertes SSL-Clientzertifikat (Zeile 5) übergeben. Daraufhin kann eine SPARQL-Anfrage unter Verwendung der Authentifizierung an PeLDS gesendet werden (Zeile 9). ARC liefert das Ergebnis in Form eines multidimensionalen Feldes zurück, über dieses wird iteriert und die Variablenbindungen ausgegeben (Zeilen 10-12). Schreibender Zugriff wird ähnlich ausgeführt, hier wird lediglich auf eine andere PeLDS-Schnittstelle zugegriffen und eine Anfrage im SPARQL/Update-Format verwendet.

```
1 <?php
2 require_once("arc/ARC2.php");
3 $config = array(
4   'remote_store_endpoint' => "https://example.com/pelds/query",
5   'arc_reader_ssl_local_cert' => "/etc/certificates/user.pem",
6 );
7 $store = ARC2::getRemoteStore($config);
8 $q = "SELECT ?s ?p ?o FROM <http://example.com/graph> WHERE { ?
    s ?p ?o . }";
9 $data = $store->query($q,'rows');
10 foreach ($data as $row) {
11   print $row['s']." ".$row['s']. " ". $row['s']."<br />\n";
12 }
13 ?>
```

Listing 5.3: Lesender Zugriff auf PeLDS mit PHP und ARC

5.2 Beispielanwendung „Verteiltes Adressbuch“

Das in Abschnitt 3.1 beschriebene Beispielszenario wurde im Rahmen dieser Diplomarbeit umgesetzt, um die Funktionen von PeLDS zu demonstrieren. Eine interaktive Version ist unter `http://contacts.pelds.org` verfügbar. Die in der Beschreibung des Beispielszenarios genannten Ziele konnten erreicht werden: Eine Integration von Kontaktdaten ist zwischen den von unterschiedlichen Einheiten betriebenen Installationen des Adressbuchs ohne vorherige Installationsarbeiten möglich. Nutzern anderer Installationen können in der selben Art und Weise wie lokalen Nutzern sicher Zugriffsrechte gegeben werden. Durch die Verwendung von Linked Data für den Austausch der Daten können weitere Anwendungen leicht auf die freigegebenen Daten im RDF-Format zugreifen. Der Nutzer kann damit mit Hilfe dieser Anwendung und der Dienste, die von PeLDS angeboten werden, die Kontrolle über die Weitergabe seiner Daten an Dritte behalten. Eine einfache Benutzerschnittstelle unterstützt den Nutzer bei der Formulierung der entsprechenden Zugriffsrichtlinie.

5.2.1 Architektur

Grundsätzlich könnte das verteilte Adressbuch als eigenständige Desktop-Anwendung implementiert werden. Leider kann derzeit nicht vorausgesetzt werden, dass jeder PC zu jeder Zeit mit dem Internet verbunden und auch aus diesem erreichbar ist. Da im verteilten Adressbuch jedoch die Daten der Kontakte zum Zeitpunkt der Anfrage erst von den Systemen der Kontakte abgerufen werden, ist es nicht sinnvoll, die Applikation auf dem jeweiligen Desktop-System zu installieren.

Wie in Abbildung 5.4 dargestellt besteht die Anwendung aus zwei Teilen: *Addressbook Client* ist eine Javascript-Anwendung, die in einem Webbrowser ausgeführt wird. Zur Darstellung wird die übliche Kombination aus HTTP und CSS benutzt. Der Client stellt nun für die verschiedenen Aktionen wie etwa den Abruf einer Kontaktliste Anfragen via HTTP an den dazugehörigen *Addressbook Server*. Dieser Server ist als PHP-Webanwendung [The09] unter der Verwendung der Bibliothek ARC für den Zugriff auf RDF-Daten mittels SPARQL [Now09] implementiert. Die Serverkomponente ist für ihre Funktion auf einen PHP-Interpreter angewiesen. Dieser Interpreter wird meist in einem Apache HTTP Server [Apa09a] ausgeführt. Sämtliche persistente Daten werden von einer PeLDS-Instanz verwaltet. Der Zugriff auf die Daten geschieht über SPARQL bzw. SPARQL/Update.

Da es möglich ist, dass Daten von Nutzern abgerufen werden, die nicht an der selben Instanz registriert sind, kann der *Addressbook Server* diese von entfernten Instanzen abrufen. Hierzu wird FOAF+SSL im Namen des anfragenden Nutzers durchgeführt, und mittels der von Linked Data vorgesehenen Dereferenzierung auf diese Daten zugegriffen.

Um den Zugriff auf die Daten nur autorisierten Nutzern zu gestatten, wird auf jeder der Kommunikationsschichten eine Authentifizierung ausgeführt. Zwischen *Addressbook Client* und *Addressbook Server* wird nach einer Anmeldung mit Benutzername und Passwort über eine HTTP-Sitzung die Identität des Nutzers sichergestellt. Zwischen *Addressbook Server* und *PeLDS* wird eine Authentifizierung mittels FOAF+SSL und verschlüsselte Übertragung über SSL durchgeführt.

Der *Addressbook Client* kann von beliebigen Rechnern aus ausgeführt werden, eine

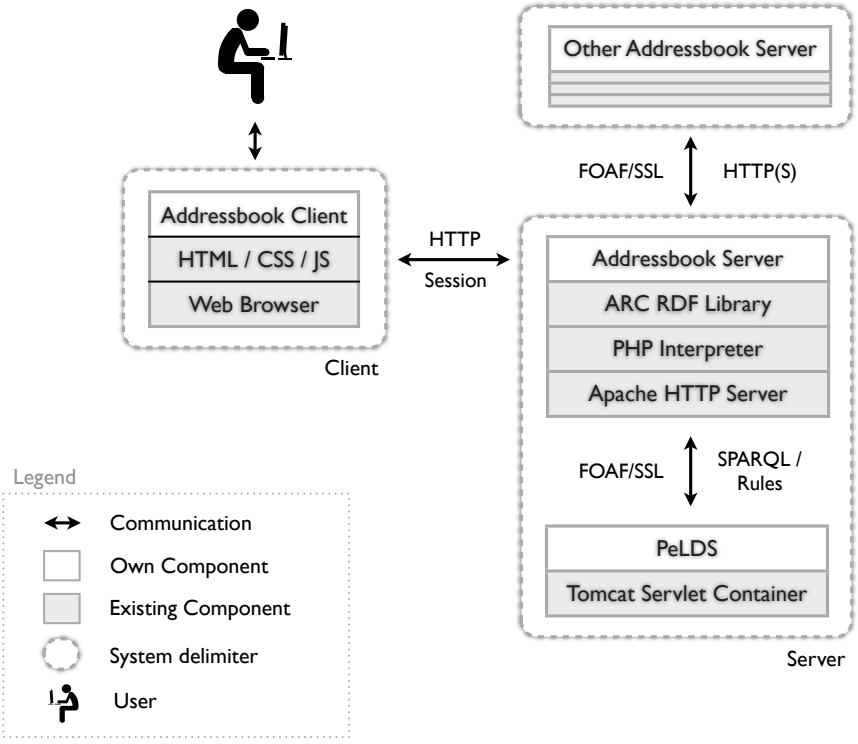

Abbildung 5.4: Verteiltes Adressbuch - Architektur

Installation ist nicht notwendig. Die benötigten Programm- und Layoutdateien werden dem Browser vom Server aus bei der ersten Interaktion bereitgestellt.

Die Schlüssel und Zertifikate werden auf dem jeweiligen *Addressbook Server* verwaltet, da einerseits manche Browser die Verarbeitung dieser Dateien nicht unterstützen und so andererseits der Nutzer von einem beliebigen System aus auf seine Kontakte zugreifen kann.

5.2.2 Funktionen

In Unterabschnitt 3.1.2 wurden die funktionalen Anforderungen in Form von Anwendungsfällen für die Anwendung „Verteiltes Adressbuch“ identifiziert. Die Anwendungsfälle wurden innerhalb der gewählten Architektur mit Hilfe verschiedener Techniken für die Gestaltung der Benutzeroberfläche umgesetzt. Grundsätzlich gliedert sich die grafische Oberfläche des Adressbuchs in die Masken zur Registrierung und Anmeldung, die Kontaktansicht sowie die Maske zur Angabe und Konfiguration der Kontaktdaten des Nutzers.

Registrierung

Nutzer müssen sich am System registrieren, bevor eine Anmeldung möglich ist. Der Nutzer gibt in der entsprechenden Eingabemaske den gewünschten Nutzername und ein Passwort an. Um Tippfehler zu vermeiden, wird das gewünschte Passwort doppelt abgefragt. Bei Betätigung der Schaltfläche *Registrieren* werden folgende Schritte durchgeführt:

(a) Die Nutzereingaben werden geprüft. Ist der Nutzername bereits im System vorhanden oder zu kurz, wird eine entsprechende Meldung ausgegeben. Weiterhin werden die angegebenen Passwörter auf Übereinstimmung und ausreichende Länge geprüft.

(b) Eine den Nutzer im Folgenden bezeichnenden URL wird erzeugt. Hierzu wird der Nutzername an die in der Konfiguration angegebene System-URL angehängt. Ist die System-URL beispielsweise <`http://example.com/addressbook`> und der Nutzername *horststein*, ergibt das <`http://example.com/addressbook/horststein`> als neuen globalen Bezeichner dieses Nutzers.

(c) Ein kryptographischer Schlüssel wird erstellt und mit Hilfe von PeLDS zertifiziert (siehe Unterabschnitt 4.3.5). Die Schlüsselmetadaten werden in PeLDS so abgelegt, dass mit dem erzeugten Zertifikat eine Anfrage mit FOAF+SSL möglich ist. Schlüssel und Zertifikat werden verwendet, um im Namen des Nutzers Inhalte in PeLDS abzuspeichern und abzufragen.

(d) Die Nutzerdaten werden in der Datenbank angelegt, so dass eine Überprüfung der Anmeldeversuche möglich ist.

(e) Ein neuer RDF-Graph wird für den Nutzer erzeugt. In diesem Graphen werden die persönlichen Daten des Nutzers abgelegt, und der Nutzer kann hierfür die entsprechende Zugriffsrichtlinie innerhalb der Anwendung festlegen.

Waren alle Schritte erfolgreich, wird dem Nutzer eine Erfolgsmeldung angezeigt.

An- und Abmeldung

Bevor ein Nutzer Zugriff auf seine Daten erhält, muss er sich am System anmelden. Hierbei wird geprüft, ob ein Nutzer mit dem angegebenen Nutzernamen in der Datenbank existiert und ob das angegebene Passwort mit dem Passwort, dass bei der Registrierung festgelegt wurde, übereinstimmt. Ist die Prüfung erfolgreich, erhält der Nutzer Zugriff, die Ansicht *Kontakte* wird angezeigt. Über die Auswahl der Schaltfläche *Abmeldung* kann die Anmeldung am System widerrufen werden.

Kontaktübersicht

Die Hauptansicht des Adressbuchs ist in Abbildung 5.5 dargestellt und gliedert sich vertikal in drei Hauptbereiche:

(a) Menüzeile mit Titel der Ansicht und Auswahlfeldern für die möglichen Ansichten

(b) Hauptbereich für Inhalte. Die dargestellte Hauptansicht *Kontakte* ist horizontal in drei Spalten aufgeteilt.

(i) Auflistung der verfügbaren Kontaktgruppen (*Gruppe*)

(ii) Auflistung der Kontakte in der ausgewählten Kontaktgruppe (*Name*)

(iii) Detailansicht des ausgewählten Kontakts (rechte Spalte)

(c) Fußzeile mit möglichen Operationen

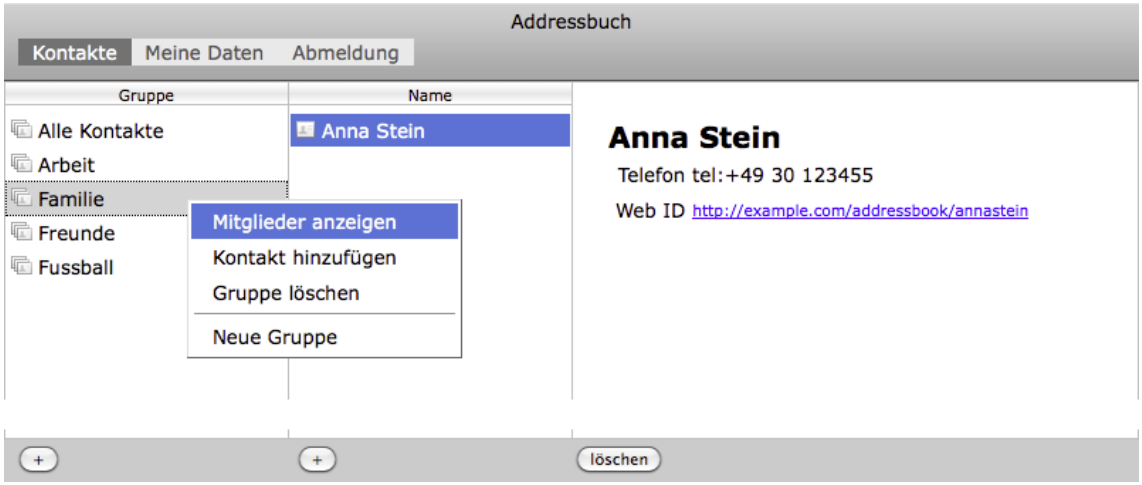

Abbildung 5.5: Verteiltes Adressbuch - Übersicht

In den Auflistungen kann wie dargestellt ein Kontextmenü zur Bearbeitung der Einträge aufgerufen werden. Die Anwendungsfälle *Kontakte verwalten* sowie *Kontaktgruppen verwalten* werden durch das Kontextmenü abgedeckt. In der Fußzeile sind häufig benötigte Operationen redundant verfügbar.

Nutzerdaten und Sichtbarkeit verwalten

Ein Nutzer kann seine Daten wie in Abbildung 5.6 dargestellt in der Ansicht *Mein Account* bearbeiten und die Zugriffsrichtlinie für seine Daten festlegen. Die Daten werden in dedizierten Eingabefeldern angezeigt und bearbeitet. Die möglichen Datenelemente können bei der Installation des Systems angepasst werden. In der rechten Spalte *Sichtbarkeit* können die Berechtigungen für die jeweiligen Datenelemente bearbeitet werden. Das Adressbuch unterstützt verschiedene Sichtbarkeitsstufen:

(a) *öffentlich* - Datenelement ist öffentlich verfügbar

(b) *Alle Kontakte* - Datenelement ist nur für die Nutzer sichtbar, die der Nutzer als Kontakt hinzugefügt hat

(c) *Gruppe* - Datenelement ist nur für Nutzer sichtbar, die der Nutzer der ausgewählten Kontaktgruppe hinzugefügt hat. In der Abbildung ist beispielsweise die Telefonnummer nur für die Gruppe *Familie* sichtbar.

(d) *Kontakt* - Datenelement ist nur für einen ausgewählten Kontakt sichtbar.

(e) *nicht sichtbar* - Datenelement ist nur für den erstellenden Nutzer sichtbar.

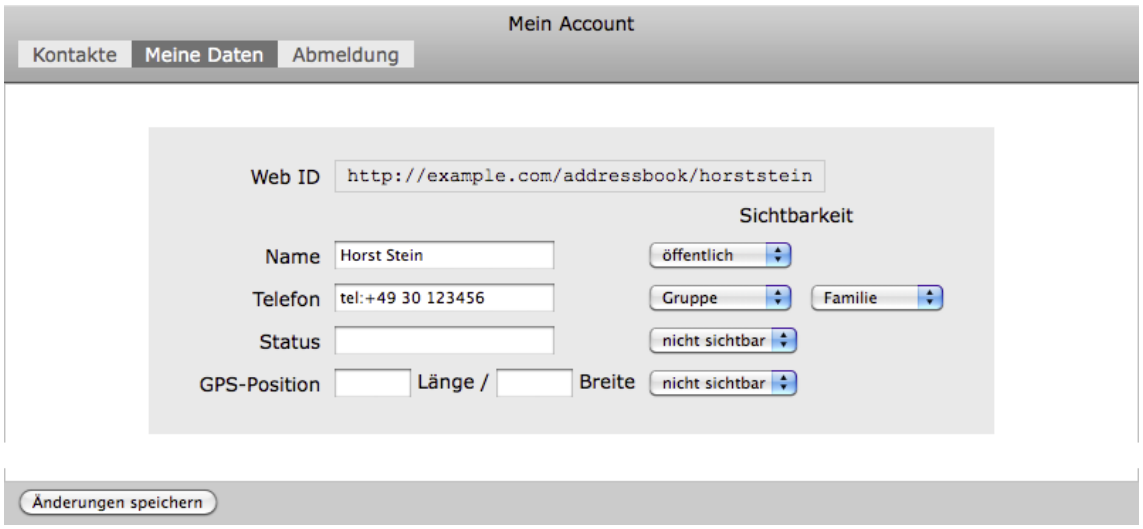

Abbildung 5.6: Verteiltes Adressbuch - Meine Daten

Die festgelegten Sichtbarkeitsstufen werden bei Betätigung von *Änderungen speichern* als PeLDS-Zugriffsrichtlinie für die Nutzerdaten ausgedrückt und an PeLDS übertragen. Die Durchsetzung der Richtlinie wird von PeLDS übernommen. Ist Richtlinienaktualisierung erfolgreich, werden die aktualisierten Daten daraufhin ebenfalls an PeLDS übertragen.

Beispiel

Zwei Nutzer sind am Adressbuch unter <`http://example.com/addressbook`> angemeldet: Horst und Anna Stein. Die in Abbildung 5.5 dargestellte Ansicht zeigt die Ansicht, die Horst Stein hat. Horst Stein hat die Gruppe *Familie* angelegt, und seine Frau zunächst anhand ihrer URI <`http://example.com/addressbook/annastein`> als neuen Kontakt in das Adressbuch eingetragen und sie daraufhin der Gruppe *Familie* hinzugefügt.

Frau Stein bekommt in ihrer in Abbildung 5.7 abgebildeten Kontaktansicht nun eine im oberen Bereich unter *Kontaktanfragen* eingeblendete Mitteilung, dass sie von Herrn Stein als Kontakt hinzugefügt worden ist. Sie wählt dort nun die Operation *hinzufügen* aus, damit wird Horst Stein ihrem Adressbuch als neuer Kontakt hinzugefügt. Die nun verfügbare Detailansicht von Horst Stein ist in Abbildung 5.8 dargestellt. Anna Stein sieht die Telefonnummer von Horst Stein, da dieser – wie in Abbildung 5.6 dargestellt – diese

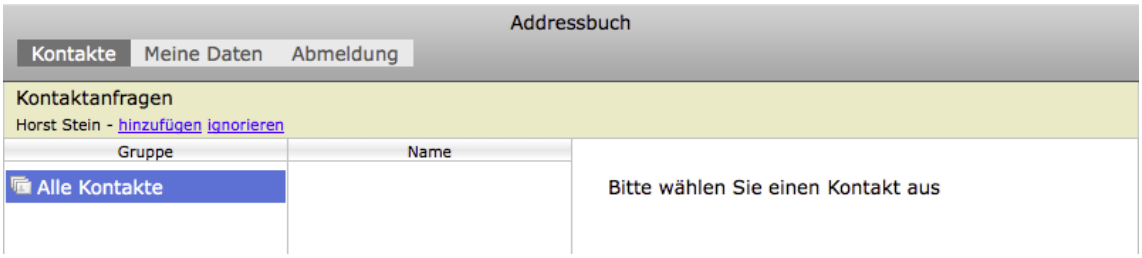

Abbildung 5.7: Verteiltes Adressbuch - Anfrage

Information für die Mitglieder seiner Gruppe *Familie* freigegeben hat. In Listing 5.4 wird ein Teil der für Horst Stein erzeugten Zugriffsrichtlinie angegeben. Die erste Regel (Zeilen 1-4) legt fest, dass Horst Steins Name öffentlich verfügbar ist. Die zweite Regel (Zeilen 6-12) legt fest, dass nur Mitglieder der Gruppe *Familie* die Telefonnummer sehen können.

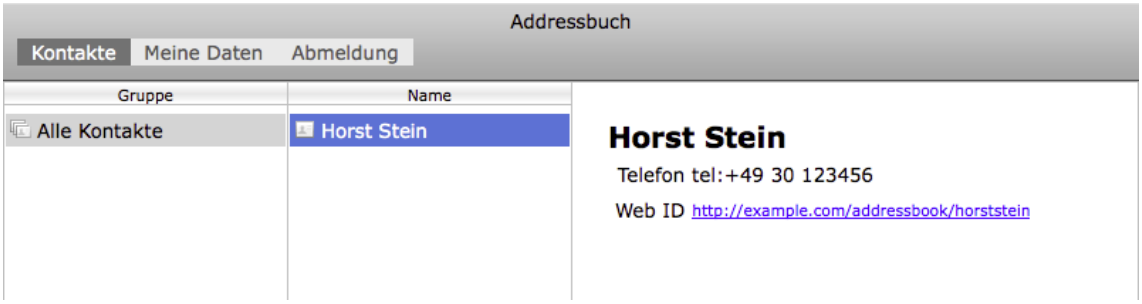

Abbildung 5.8: Verteiltes Adressbuch - Detailansicht Anna Stein

5.2.3 Evaluation

Nachdem in den Grundlagenkapiteln Kriterien etwa für die Bewertung eines sicheren Systems angegeben wurden, werden diese Kriterien nun angewendet, um das Programm „verteiltes Adressbuch“ hinsichtlich der Umsetzung der verschiedenen nicht-funktionalen Anforderungen zu bewerten. Die Ergebnisse der Bewertung des Adressbuchs sind in Tabelle 5.4 kurz zusammengefasst.

Sicherheit

Nach der in Definition 2.1 angegebenen Definition von Sicherheit ist das Adressbuch ein sicheres System: Es wird mit Hilfe von speziellen Funktionen kontrolliert, dass Daten nur an autorisierte Nutzer ausgeliefert werden: Das Adressbuch selbst hält keine Daten, diese Aufgabe wurde an PeLDS delegiert. Das Adressbuch kontrolliert allerdings den Zugang zu den kryptographischen Schlüsseln. Mit diesen Schlüsseln können alle von dieser Instanz

```
1 AddressbookDocument_RdfInputText_RdfVisibilityWorld_name:
2   pelds:QueryAction(?action)
3 =>
4 permit_triple(http://example.com/addressbook/horststein,foaf:
    name,*);
5
6 AddressbookDocument_RdfInputText_RdfVisibilityGroup_phone:
7   pelds:QueryAction(?action) &&
8   pelds:actor(?action,?actor) &&
9   foaf:knows(http://example.com/addressbook/horststein,?actor)
      &&
10  foaf:member(http://example.com/addressbook/horststein/group/
      familie,?actor)
11 =>
12 permit_triple(http://example.com/addressbook/horststein,contact
    :phone,*);
```

Listing 5.4: Richtlinie Adressbuch Horst Stein (Ausschnitt)

in der PeLDS-Instanz abgelegten Daten abgerufen oder verändert werden. Aus diesem Grund ist kein entfernter Zugriff auf die Zertifikate möglich. Nur derjenige Nutzer, für den der jeweilige Schlüssel angelegt und zertifiziert wurde, kann diesen nach der erfolgreichen Anmeldung am Adressbuch für den Zugriff auf in einer PeLDS-Installation veröffentlichte Daten nutzen.

Die in Unterabschnitt 2.1.1 genannten Kriterien werden nun stichpunktartig behandelt.

(a) Richtlinien - Nutzer können Zugriffsrichtlinien für die in ihrem Profil angegebenen Kontaktinformationen vergeben. Das Adressbuch unterstützt verschiedene Sichtbarkeitsstufen, die von Richtlinien durchgesetzt werden.

(b) Klassifizierung - Nutzer können alle Datenelemente, die sie in ihrem Profil veröffentlichen, durch Angabe der jeweiligen Sichtbarkeit als geschützt markieren.

(c) Authentifizierung - Vor dem Zugriff auf geschützte Daten müssen sich Nutzer am Adressbuch anmelden.

(d) Verantwortlichkeit - Sämtliche Zugriffe auf das Adressbuch werden in der Logdatei des Apache HTTP Server, auf dem der Adressbuch Server ausgeführt wird, aufgezeichnet. Damit kann nachvollzogen werden, welcher Nutzer des Adressbuchs welche Operation durchgeführt hat.

Die Umsetzung der Forderungen nach Garantie und fortlaufendem Schutz beschränkt sich auf die Verifikation der korrekten Funktion der Anmeldeprozedur sowie der korrekten Durchsetzung von Richtlinien. Siehe dazu auch Unterabschnitt 5.1.2. Zusammenfassend

kann festgestellt werden, dass das Adressbuch den genannten Kriterien für ein sicheres System genügt.

Privatsphäre

Das Adressbuch kann nach den in Unterabschnitt 2.1.2 aufgeführten Kriterien für die technische Umsetzung eines Systems, welches die Privatsphäre der Nutzer schützen soll, bewertet werden.

(a) Klassifizierung - Das Adressbuch bietet dem Nutzer die Möglichkeit, selbst zu entscheiden, welche seiner Kontaktdaten welchen Nutzern zur Verfügung gestellt werden. Die Klassifizierung wird über die im RDF-Format dargestellten Daten in der Richtliniensprache PsSF durchgeführt.

(b) Zugriffskontrolle - Die Verwendung der innerhalb des Adressbuchs gespeicherten und als geschützt markierten Daten ist nur nach einer Anmeldung der Nutzer mit deren Benutzername und Kennwort möglich.

(c) Richtlinienkontrolle - Die Zugriffsrichtlinie, die für die vom Nutzer mittels des Adressbuchs veröffentlichten Daten gilt, kann von diesem entsprechend seiner Vorstellungen eingestellt werden.

(d) Verwendung von Daten durch Dritte - Ist eine Information einmal an einen berechtigten Nutzer ausgeliefert, kann sie nicht mehr kontrolliert werden. Da das Adressbuch die Möglichkeit einer Anzeige der Informationen wie etwa einer Telefonnummer auf einem Anzeigegerät unterstützt, kann die Information im einfachsten Fall vom Bildschirm abgeschrieben werden und kann dann nicht weiter von technischen Methoden geschützt werden.

Von der Kontrolle der Daten, die an Dritte ausgeliefert werden abgesehen, unterstützt das Adressbuch den Nutzer beim Schutz seiner Privatsphäre. Weiterhin ist festzustellen, dass die Ablage der Daten auf einem System unter der Kontrolle des Nutzers einen relevanten Sicherheitsvorteil im Vergleich zu den in Abschnitt 3.1 erwähnten verbreiteten zentralisierten Systemen darstellt.

Richtlinientyp

Anhand der in Unterabschnitt 2.3.1 angegebenen Klassifikation verschiedener Richtlinientypen wird deutlich, dass das Adressbuch eine Richtlinie vom Typ *Discretionary Access Control* erzeugt. Der Nutzer kann die Zugriffsrechte auf ein Objekt anderen Nutzern oder Nutzergruppen vergeben. Propagation kann eingeschränkt werden, wenn das entsprechende Datenfeld – falls möglich – in mehrere Datenfelder aufgeteilt wird. Beispielsweise kann die Angabe einer Adresse in die verschiedenen Felder für Straße, Hausnummer usw. aufgeteilt werden.

Nutzer, die keine Zugriffsrechte auf ein Datenobjekt haben, können auf diese nicht zugreifen, das Adressbuch lässt als kleinste Granularität die Vergabe von Rechten an einzelne Nutzer zu.

Datenklassifikation

Der Nutzer kann die Datenklassifikation, also die Zusammenfassung von Tripeln zu von einer Richtlinie betroffenen Gruppe im Adressbuch nicht direkt beeinflussen. Im Interesse einer einfachen Bedienung wird diese Entscheidung während der Installation des Adressbuchs getroffen. Hier werden beispielsweise für die Darstellung einer GPS-Position zwei Tripel für Länge und Breite zur Gruppe „Position“ zusammengefasst und die Art und Weise, wie diese Information bearbeitet werden soll, festgelegt. In Abbildung 5.6 wird eine Möglichkeit zur Bearbeitung einer Position abgebildet. In dieser Art werden alle Informationen, die der Nutzer über sich angeben kann, in Tripelgruppen abgebildet, für die vom Nutzer jeweils die Zugriffsrechte vergeben werden können. In der Implementierung ist dies so gelöst, dass die jeweiligen Gruppen den Konsequenzteil der dazugehörigen PsSF-Regel definiert. Die Eingaben des Nutzers ergeben dann die Bedingung der Regel.

Den in Unterabschnitt 2.3.2 angegebenen Typen von Klassifikationen können die definierten Gruppen sowohl auf Basis einzelner Tripel (Modellebene), auf der Grundlage der definierten Konzepte und Eigenschaften (Semantische Ebene) sowie auf der Ebene der verwendeten URIs definiert werden (Ressourcenebene). Damit bietet das Adressbuch die gewünschte Flexibilität bei der Definition der Datengruppen und damit der Klassifikation.

Semantic Web / Linked Data

Die Verwendung von Technologien des Semantic Web oder Linked Data (siehe Unterabschnitt 2.2.3) war eine der Voraussetzungen für den Entwurf, daher wurde bei der Umsetzung darauf geachtet, diese einzuhalten.

Alle Daten sind innerhalb von RDF-Graphen als Tripel abgelegt. Die Nutzer werden anhand einer URL bezeichnet, die eine Verbindung über das Protokoll HTTP beschreibt (*LD1*). Wird diese URL angefragt, liefert das Adressbuch je nach Art des anfragenden Clients einen Verweis auf eine RDF-Darstellung der verfügbaren Informationen über den Nutzer, der von dieser URL beschrieben ist (*LD2*). Der Nutzer kann diese URL, die auch als Web ID bezeichnet wird, innerhalb der grafischen Oberfläche abrufen, da sie auch benutzt wird, um neue Kontakte dem Adressbuch hinzuzufügen.

Die weiterhin geforderte Linked-Data-Eigenschaft, Verknüpfungen zu weiteren Informationen anzubieten (*LD2*), kann leicht erfüllt werden, indem die für den Nutzer sichtbaren Felder beispielsweise um ein Feld für Verweise auf weitere Daten erweitert werden. Insgesamt wird deutlich, dass das Adressbuch die Linked-Data-Eigenschaften erfüllt.

Systemleistung

Das Adressbuch selbst speichert keine Daten, sämtliche Inhalte werden innerhalb der konfigurierten PeLDS-Instanz gespeichert. Die Geschwindigkeit, in der diese Inhalte angefragt werden können, ist damit der einzige limitierende Faktor der Systemleistung. Das Adressbuch selbst hängt in seiner Leistung so direkt von der Leistung des PeLDS-Systems ab, die in Unterabschnitt 5.1.2 beschrieben wird. Sollte dennoch ein Engpass in der Ausführung der PHP-Skripte der Serverkomponente des Adressbuchs auftreten, können leicht mehrere

Installationen des Adressbuchs mit der selben PeLDS-Instanz als Datenspeicher konfiguriert werden.

Bereich	*Anforderung*	*Ergebnis*
Funktionale Anforderungen	Registrierung	✓
	Anmeldung	✓
	Kontakte verwalten	✓
	Gruppen verwalten	✓
	Informationen ablegen	✓
	Informationen schützen	✓
	Informationen abrufen	✓
	Verteilte Installation	✓
	Lokale Datenhaltung	✓
Sicherheit	Richtlinien	✓
	Klassifizierung	✓
	Authentifizierung	✓
	Verantwortlichkeit	✓
	Garantie	×
	Fortlaufender Schutz	×
Privatsphäre	Klassifizierung	✓
	Zugriffskontrolle	✓
	Richtlinienkontrolle	✓
	Datenverwendung	×
Linked Data	HTTP-URLs	✓
	Dereferenzierung	✓
	Verweise	✓

Tabelle 5.4: Verteiltes Adressbuch - Übersicht über die Bewertungsergebnisse

Kapitel 6

Zusammenfassung und Ausblick

6.1 Zusammenfassung

Das Ziel der vorliegenden Arbeit war es, Möglichkeiten zur Authentifizierung von Nutzern und die Vergabe und Durchsetzung von Zugriffsrichtlinien im Umfeld von Linked Data zu untersuchen. Ein weiteres Ziel bestand darin, ein System zu entwickeln, das diese Funktionalität umsetzt, da ein solches System noch nicht zur Verfügung steht.

Die im Beispielszenario des verteilten Adressbuchs erhobenen Anforderungen haben zur Spezifikation einer Serverkomponente zur sicheren Ablage von RDF-Daten geführt. Die Verwendung des Linked-Data-Konzepts löst zentrale Probleme dieser Anforderungen, die Datenintegration zwischen beliebigen Systemen wird damit einfach möglich. Für die Definition von Zugriffsrichtlinien sowie zur Beschreibung der Elemente eines RDF-Graphen, die von einer Richtlinie betroffen sind, wurde die auf SWRL basierte Sprache PsSF entwickelt. Der Vergleich von bestehenden Lösungen zur Definition von Zugriffsrichtlinien hat ergeben, dass nur eine für das Datenmodell RDF entworfene Richtliniensprache die erforderliche präzise Kontrolle erlaubt. Mit PsSF können Nutzer eines RDF-Servers beschreiben, welche Tripel welchen Nutzern übermittelt werden sollen. Der diese Sprache unterstützende Linked-Data-Server PeLDS ist damit der erste verfügbare Linked-Data-Server, der die Vergabe von Zugriffsrichtlinien unterstützt. PeLDS ist als Prototyp im Rahmen dieser Arbeit entstanden. Für die notwendige Authentifizierung der Nutzer wird in PeLDS das Verfahren FOAF+SSL eingesetzt, bei dem die Identität des anfragenden Nutzers ohne Vorarbeiten zwischen den einzelnen Systemen geprüft werden kann.

Der Vergleich der Leistungsdaten des PeLDS-Prototyps mit der Software Joseki macht deutlich, dass Authentifizierung und die Auswertung von Zugriffsrichtlinien die Systemleistung eines RDF-Servers nicht unverhältnismäßig beeinträchtigt. Für die Datenabfrage mit gleichzeitiger Berechnung der aus RDF-Daten und Ontologien ableitbarer Informationen wurde ein linearer Faktor für den genannten Mehraufwand ermittelt. Der Prototyp steht auf der im Rahmen der Arbeit entstandenen Website `http://www.pelds.org` zum Download zur Verfügung, um eine Weiterentwicklung von PeLDS über diese Arbeit hinaus zu ermöglichen.

Da es sich bei PeLDS um eine Komponente handelt, deren Funktion für Nutzer nur

schwer erkennbar ist, wurde ein Programm entwickelt, das eine Möglichkeit für den Einsatz von PeLDS anschaulich darstellt. Die Bewertung dieser Implementierung eines verteilten Adressbuchs im Bezug auf Sicherheit und Privatsphäre ergab, dass diese Anwendung mit Hilfe von PeLDS die Privatsphäre ihrer Nutzer deutlich besser schützen kann als bestehende Systeme. Eine Installation dieser Anwendung findet sich unter `http://contacts.pelds.org`, die notwendigen Programmdateien sind ebenfalls verfügbar.

6.2 Ausblick

Mögliche Weiterentwicklungen des Konzepts PeLDS sowie der Richtliniensprache PsSF sind in vielen Bereichen denkbar. Die Herausforderung der Ablage von zu RDF-Daten gehörigen Binärdateien wie etwa digitales Bildmaterial in ebenfalls geschützter Weise ist hier ein wichtiger Punkt. Im Bereich der Richtliniensprache PsSF sind weitere Prädikate denkbar, insbesondere könnte ein Negations-Operator in einigen Anwendungsfällen nützlich sein. Im Bereich des von PeLDS verwendeten Protokolls sind weitere Operationen auf den Zugriffsrichtlinien möglich, beispielsweise die selektive Auswahl und Veränderung einzelner Regeln. Sollen umfangreiche Datensätze verwaltet werden, sind Optimierungen der verschiedenen Operationen des Prototypen notwendig. Zwischenspeicher können an verschiedenen Stellen implementiert werden, um die Zugriffsgeschwindigkeit weiter zu erhöhen. Die Auswertung der Zugriffsrichtlinien kann zudem teilweise durch das nachgelagerte Datenbanksystem übernommen werden, um die Systemleistung weiter zu verbessern.

Abschließend bleibt festzustellen, dass die für diese Arbeit gesteckten Ziele erreicht wurden. Mit PeLDS steht ein System zur Durchsetzung von Zugriffsrichtlinien für RDF-Daten zur Verfügung. Die Implementierung des verteilten Adressbuchs hat gezeigt, dass es möglich ist, mit Hilfe von PeLDS Anwendungen zum sicheren Austausch persönlicher Daten über Systemgrenzen hinweg zu entwickeln. Durch die Speicherung der Daten auf einem vom Nutzer kontrollierten System ist der Schutz der Privatsphäre unabhängig von Vertrauen in eine zentrale Instanz. Dies kann einen Paradigmenwechsel in Entwurf, Umsetzung und Nutzung vieler Anwendungen einleiten. Es stellt sich die Frage, warum Millionen Internetnutzer weiterhin ihre persönlichen Daten oft zweifelhaften Unternehmen und Systemen anvertrauen sollten. Es ist zu hoffen, dass sich Anwendungen auf Basis dieses oder ähnlicher Systeme in der Zukunft gegen den zu erwartenden Widerstand durch bestehende Anbieter etablieren können. In jedem Falle ist es wünschenswert, die viel versprechende Vision des Semantic Web durch neue Anwendungen mit Leben zu erfüllen.

Anhang A

Ergänzende Inhalte

A.1 Umsetzung von Discretionary Access Control mit PsSF

Um zu zeigen, wie ein System, das den Richtlinientyp Discretionary Access Control (DAC) umsetzt mit Hilfe von PeLDS umgesetzt werden kann, wurde das folgende Beispiel entwickelt. Bei DAC ist es dem Nutzer ermöglicht, anderen Nutzern Zugriffsrechte auf Datenobjekte zu geben (*DAC1*). Diese Zugriffsrechte werden automatisch auf Unterobjekte weitergegeben, der Vorgang wird als Propagation bezeichnet (*DAC2*). Es ist möglich, Zugriffsrechte nur für einzelne Nutzer zu vergeben (*DAC3*), und nur der Nutzer, der ein Objekt besitzt, kann Zugriffsrechte auf dieses vergeben (*DAC4*).

Der in Listing A.1 serialisiert dargestellte RDF-Graph enthält ein primitives Vokabular zu Beschreibung von Baumstrukturen sowie einige Instanzen von Knoten. Das Vokabular (Zeilen 4-6) definiert die Konzepte *TreeNode* und *PublicNode* sowie die Eigenschaft *parentNode*. Baumknoten sind Instanzen von *TreeNode*, sie enthalten möglicherweise einen Verweis auf einen Elternknoten mit *parentNode*. Die Instanzen *root*, *node1*, *node2* und *node3* sind als Baum miteinander verbunden, *root* ist der Wurzelknoten mit den Kindknoten *node1* und *node3*, *node2* ist wiederum ein Kindknoten von *node1*.

```
1 @prefix owl: <http://www.w3.org/2002/07/owl#> .
2 @prefix ex:  <p://ont#> .
3
4 ex:TreeNode     a  owl:Class .
5 ex:PublicNode   a  owl:Class .
6 ex:parentNode   a  owl:ObjectProperty .
7
8 <p://root>    a  ex:TreeNode .
9
10 <p://node1>  a  ex:TreeNode ;
11   ex:parentNode  <p://root> .
12
13 <p://node2>  a  ex:TreeNode ;
14   ex:parentNode  <p://node1> .
```

```
15
16 <p://node3>  a  ex:TreeNode  ;
17   ex:parentNode  <p://root> .
```

Listing A.1: DAC in PeLDS - Daten

Die zu diesem RDF-Graphen gehörige Zugriffsrichtlinie, die DAC umsetzt, ist in Listing A.2 im Format PsSF angegeben. Die Richtlinie enthält drei Regeln: *RootAccessRule*, *RecursiveAccessRule* sowie *PermitRule*. *RootAccessRule* und *PermitRule* setzen *DAC1* und *DAC3* um, der Nutzer vergibt Zugriffsrechte auf die Instanz *root*, in dem er sie in *RootAccessRule* als Instanz des Konzepts *PublicNode* deklariert und die Instanzen dieses Konzepts in *PermitRule* für einen ausgewählten Nutzer (*p://user/horst*) zugänglich macht. *DAC2* wird von *RecursiveAccessRule* umgesetzt, diese Regel definiert Kindknoten eines Knoten, der Instanz des Konzepts *PublicNode* ist, als ebenfalls diesem Konzept zugehörig. Die DAC-Eigenschaft *DAC4* kann analog umgesetzt werden.

```
1 RootAccessRule:
2 ex:TreeNode(?node) &&
3 swrl:sameAs(?node,p://root)
4  =>
5 ex:PublicNode(?node);
6
7 RecursiveAccessRule:
8 ex:PublicNode(?parent) &&
9 ex:TreeNode(?child) &&
10 ex:parentNode(?child,?parent)
11  =>
12 ex:PublicNode(?child);
13
14 PermitRule:
15 pelds:QueryAction(?action) &&
16 pelds:actor(?action,p://user/horst)
17  =>
18 pelds:permit_instance(ex:PublicNode) ;
```

Listing A.2: DAC in PeLDS - Regeln

Wird nun vom Nutzer *p://user/horst* die in Listing A.3 dargestellte SPARQL-Anfrage an PeLDS gestellt, kann er auf *node3* zugreifen, da die Vererbung seiner Zugriffsrechte auf *root* dies ermöglicht. Das Resultat dieser Anfrage ist in Listing A.4 angegeben.

```
1 DESCRIBE <p://node3> FROM <p://graph>
```

Listing A.3: SPARQL-Anfrage für DAC

```
1 <p://node3>  a  ex:TreeNode, ex:PublicNode  ;
2   ex:parentNode  <p://root> .
```

Listing A.4: DAC in PeLDS - Resultat

A.2 Methode der kleinsten Quadrate

Bei der Methode der kleinsten Quadrate [Wei09] handelt es sich um einen statistischen Anpassungstest. Mit dieser Methode kann festgestellt werden, wie gut eine Serie von Messwerten einer vorgegebenen Funktion folgt. Die Methode ergibt einen als R^2 bezeichneten Wert im Intervall [0,1], je höher der Wert ist, desto besser beschreibt die Funktion den Verlauf der Messwerte. Die Messwerte sind Elemente der Menge $y := \{y_1, y_2, \ldots, y_n\}$, wobei y_i den Messwert mit dem in i bezeichneten Index bezeichnet. $\overline{y}$ beschreibt das arithmetische Mittel aller Messwerte. $f(x)$ ist der Funktionswert der vorgegebenen Approximationsfunktion an der Stelle x. Damit berechnet sich R^2 wie folgt:

$$R^2 := 1 - \frac{\sum_i (y_i - f(i))^2}{\sum_i (y_i - \overline{y})^2}$$

Je näher die Messwerte y_i an den entsprechenden Funktionswerten $f(i)$ sind, umso geringer fällt der Wert des Zählers aus, und umso höher ist der Ergebniswert von R^2 im Intervall [0,1]. Bei hohen Werten von R^2 kann darauf geschlossen werden, dass die Messwerte von der angegebenen Funktion gut approximiert werden.

A.3 Messwerte Leistungstests

In Tabelle A.1 sind die Messwerte des Anfrageleistungstests für jede der verwendeten Datensatzgrößen in Sekunden (s) angegeben. In Tabelle A.2 sind diese Werte in gleicher Form für den Test der Datenveränderung enthalten.

BSBM-Skala	*Tripel (ca.)*	*Joseki*	*Joseki / Pellet*	*PeLDS*
1/4	450	0,11	0,23	0,51
1/2	900	0,17	0,59	1,61
1	1842	0,39	2,31	6,03
2	2193	0,58	3,86	10,10
3	2539	0,46	4,62	12,46
4	2881	0,68	6,72	17,80
5	3225	0,68	9,15	23,16
6	3583	1,03	11,30	30,07
7	3934	0,96	13,97	32,73
8	4278	0,81	18,38	43,22

Tabelle A.1: Anfrageleistungstest - Messwerte

A.4 EBNF-Spezifikation der PsSF-Regelsyntax

In Tabelle A.3 ist eine Spezifikation des von PeLDS unterstützten Richtlinienformats PsSF in der Erweitereten Backus-Naur-Form (EBNF) angegeben [Int96].

BSBM-Skala	*Tripel (ca.)*	*Joseki*	*PeLDS*
1/4	450	0,15	1,15
1/2	900	0,18	2,34
1	1842	0,25	5,43
2	2193	0,30	6,03
3	2539	0,27	7,25
4	2881	0,29	8,28
5	3225	0,32	9,81
6	3583	0,35	10,98
7	3934	0,38	12,43
8	4278	0,41	13,92

Tabelle A.2: Veränderungsleistungstest - Messwerte

```
Ruleset              ::=  ( Rule )*
Rule                 ::=  Label ':' AntecedentList '=>' ConsequentList ';'
AntecedentList       ::=  Atom ( '&&' Atom )*
ConsequentList       ::=  Atom ( '&&' Atom )*
Atom                 ::=  ( Prefix ':' )? Name '(' ArgumentList ')'
ArgumentList         ::=  Argument ( ',' Argument )*
Argument             ::=  ( Variable | Literal | Url | Wildcard)
Variable             ::=  '?' STR
Literal              ::=  '"' [^"]+ '"'
Url                  ::=  Definiert in RFC 1738 [BLMM94]
Wildcard             ::=  '*'
Label, Prefix, Name  ::=  STR
STR                  ::=  [a-zA-Z]+
```

Tabelle A.3: EBNF-Darstellung des Regelformats

A.5 Installationsanleitung PeLDS

Diese Anleitung beschreibt die Installation auf Basis des auf CD und unter `http://www.pelds.org` verfügbaren Installationspakets von PeLDS.

A.5.1 Voraussetzungen

PeLDS kann auf den Unix-Systemen Linux und Mac OS X sowie auf Windows installiert werden. Auf allen Systemen muss eine Java-Laufzeitumgebung [Sun09a] (mindestens Version 1.6) verfügbar sein. Zusätzlich wird eines der unterstützten Datenbanksysteme HSQLDB, MySQL [Sun09b], Oracle, PostgreSQL, oder Apache Derby vorausgesetzt. Diese Anleitung beschreibt die Installation mit MySQL.

Die Umgebungsvariable *JAVA_HOME* muss auf das Verzeichnis verweisen, in dem die Java-Laufzeitumgebung installiert wurde.

A.5.2 Entpacken

Das Installationspaket ist in einer Tar/Gzip-Datei gepackt. Dieses muss auf die Festplatte des Systems, auf das PeLDS installiert werden soll, entpackt werden. Hierzu kann unter Unix-Systemen der Befehl

```
tar xvzf pelds-package-latest.tar.gz
```

verwendet werden. Der angegebene Dateiname muss möglicherweise angepasst werden. Der Entpackvorgang erzeugt ein Verzeichnis *pelds-tomcat*, das PeLDS enthält. Dieses kann nun an eine beliebige Stelle im Dateisystem verschoben werden, zum Beispiel */var*, so dass die PeLDS-Installation im Verzeichnis */var/pelds-tomcat* verfügbar ist.

A.5.3 Datenbank

Im Datenbanksystem muss eine neue Datenbank erzeugt werden. Wird MySQL eingesetzt wird, kann das mitgelieferte SQL-Skript *tables.sql* im Unterverzeichnis *conf* verwendet werden. Dieses Skript erstellt eine Datenbank mit dem Namen *pelds* und einen gleichnamigen Nutzer. Das im SQL-Skript angegebene Passwort *changeme* sollte durch Bearbeitung der Datei geändert werden. Unter Unix kann das folgende Kommando für die Ausführung des Skripts benutzt werden. Das Passwort des MySQL-Nutzers *root* muss im weiteren Verlauf eingegeben werden.

```
mysql -u root -p < tables.sql
```

A.5.4 Konfiguration

PeLDS wird in der Datei *pelds.properties* im Verzeichnis *conf* konfiguriert. In dieser Datei müssen die Zugangsdaten für die verwendete Datenbank eingetragen werden. Wurde die MySQL-Setupdatei verwendet, muss lediglich der Wert *pelds.db.pass* auf das in Unterabschnitt A.5.3 vergebene Passwort für den Datenbank-Nutzer *pelds* geändert werden. Die

weiteren Konfigurationsvariablen sind in **??** beschrieben, müssen jedoch zunächst nicht verändert werden.

A.5.5 Start

Für den Start des Systems wird unter Unix-Systemen die Datei *startup.sh* und unter Windows die Datei *startup.bat* ausgeführt. Diese befinden sich im Ordner *bin* in der PeLDS-Installation. Bei einem erfolgreichen Start wird eine den folgenden Zeilen ähnliche Ausgabe erzeugt.

```
Using CATALINA_BASE:  /var/pelds-tomcat
Using CATALINA_HOME:  /var/pelds-tomcat
Using CATALINA_TMPDIR:  /var/pelds-tomcat/temp
Using JRE_HOME:  /System/Library/Frameworks/JavaVM.framework/
    Home
```

Der PeLDS kann nun verwendet werden. PeLDS reagiert auf HTTP-Anfragen an den TCP-Port 7080 sowie auf HTTPS-Anfragen auf dem TCP-Port 7443. Wird bei Anfragen nichts oder nur eine Fehlermeldung angezeigt, können Hilfestellungen zur Problembehebung den Logdateien im Verzeichnis *logs* entnommen werden. Weitere Hilfe bei der Installation kann auf der PeLDS-Webseite `http://www.pelds.org` abgerufen werden.

A.5.6 Konfiguration

PeLDS kann vom Nutzer konfiguriert werden, um das System an lokale Gegebenheiten anzupassen. Die in der Konfigurationsdatei *pelds.properties* vom Nutzer zu verändernden Variablen werden in der folgenden Auflistung beschrieben. Der Ort, an dem diese Datei gesucht wird, wird durch die Eigenschaft *ConfigFileLocation* in der Datei *web.xml* festgelegt.

- *pelds.db.url* - JDBC-URL für den Datenbankzugriff
- *pelds.db.user* / *pelds.db.pass* - Nutzername und Passwort für den Datenbankzugriff
- *pelds.db.driver* - Datenbanktreiber für das verwendete DBMS
- *pelds.cert.keystore.location* - Ort eines Java-Keystores, in dem das Serverzertifikat gespeichert ist
- *pelds.cert.keystore.pass* - Passwort für den Zugriff auf den Keystore
- *pelds.cert.server.name* / *pelds.cert.server.pass* - Name und Passwort des Serverzertifikats
- *pelds.dereferencing.prefix* - URL-Präfix für die Dereferenzierung
- *pelds.dereferencing.endpoint* - SPARQL-Endpoint für Dereferenzierung
- *pelds.prefixes.map* - Datei, die die Namensraum-Präfixe für Regelauswertung enthält

A.6 Installationsanleitung Verteiltes Adressbuch

Diese Anleitung beschreibt die Installation auf Basis des auf CD und unter `http://www.pelds.org` verfügbaren Installationspakets der Beispielapplikation „Verteiltes Adressbuch“.

A.6.1 Voraussetzungen

Das verteilte Adressbuch kann auf den Unix-Systemen Linux und Mac OS X sowie auf Windows installiert werden. Auf allen Systemen muss ein Apache-Webserver [Apa09a] installiert sein. Die PHP-Laufzeitumgebung [The09] (mindestens Version 5) installiert sein. Zusätzlich wird das Datenbanksystem MySQL [Sun09b] benötigt.

A.6.2 Entpacken

Das Installationspaket ist in einer Tar/Gzip-Datei gepackt. Dieses muss auf die Festplatte des Systems, auf das PeLDS installiert werden soll, entpackt werden. Hierzu kann unter Unix-Systemen der Befehl

```
tar xvzf addressbook-latest.tar.gz
```

verwendet werden. Der angegebene Dateiname muss gegebenenfalls angepasst werden. Der Entpackvorgang erzeugt ein Verzeichnis *addressbook*, das die Anwendung enthält. Dieses kann nun an eine beliebige Stelle im Dateisystem verschoben werden, zum Beispiel */var*, so dass die Installation des Adressbuchs im Verzeichnis */var/addressbook* verfügbar ist.

A.6.3 Konfiguration

Damit ein Zugriff über HTTP auf das Adressbuch möglich ist, muss ein entsprechender Verweis in die Apache-Konfiguration eingetragen werden. Ein solcher Verweis ist im Folgenden angegeben. *ServerName* muss auf einen für den Server gültigen Namen gesetzt werden.

```
<VirtualHost *:80>
    ServerName addressbook.example.com
    DocumentRoot /var/addressbook
    <Directory /var/addressbook>
        AllowOverride All
   </Directory>
</VirtualHost>
```

Das Adressbuch wird in der Datei *config.php* im Verzeichnis *lib* konfiguriert. Dazu wird die Datei *config.dist.php* nach *config.php* kopiert und angepasst. In der Konfigurationsdatei müssen die Zugangsdaten für die verwendete Datenbank eingetragen werden. Hierzu werden die Werte der Variablen *TEMP_STORE_** verändert. Die Variable *PELDS_INSTANCE* wird so gesetzt, dass die vergebene URL auf eine PeLDS-Installation verweist. Ist PeLDS

auf dem selben System wie das Adressbuch installiert, kann der vorgegebene Wert beibehalten werden. Der Wert *INSTANCE_ URL* verweist auf die URI, unter der diese Installation erreichbar ist.

Weitere Konfigurationsvariablen sind in Unterabschnitt A.6.5 beschrieben, müssen jedoch zunächst nicht verändert werden.

Schließlich wird die Variable *IS_ SET_ UP* auf den Wert *true* gesetzt, um die Konfiguration abzuschließen.

A.6.4 Start

Der Apache-Server muss neu gestartet werden, damit die geänderten Einstellungen wirksam werden. War die Konfiguration erfolgreich, sollte eine der in **??** ähnliche Maske beim Aufruf der URL des Adressbuchs mit einem Browser angezeigt werden.

A.6.5 Konfiguration

Die Konfiguration des Adressbuchs ist recht übersichtlich. Im Folgenden werden die in der Konfigurationsdatei *lib/config.php* vom Nutzer zu verändernden Variablen beschrieben.

- *INSTANCE_ URL* - URL, unter der diese Installation des Adressbuchs erreichbar ist.
- *PELDS_ INSTANCE* - URL, unter der ein PeLDS-System erreichbar ist.
- *TEMP_ STORE_ ** - Zugriffsdaten auf eine MySQL-Datenbank, die für die Zwischenspeicherung von RDF-Daten verwendet wird.
- *IS_ SET_ UP* - wird auf *true* gesetzt, wenn die anderen Variablen gesetzt wurden. Dies erlaubt den Zugriff auf das Adressbuch.

A.7 Instanzen für SPARQL-Anfrage

```
@prefix ont:  <http://www.example.com/ont#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http://maxpage.example/card#me>
  ont:geschlecht <http://example.com/ont#maennlich> ;
  ont:name "Max Muenchhausen" ;
  ont:phone "0711766192" .

<http://ast.example/andreas>
  ont:geschlecht <http://example.com/ont#maennlich> ;
  ont:name "Andreas Ast" ;
  ont:phone "030433344" .

<http://bmfsfj.example/people#ulla>
  ont:geschlecht <http://example.com/ont#weiblich> ;
  ont:name "Ulla von der Schleien" ;
  ont:phone "030185550" .

<http://firmaschmidt.example/chef>
  ont:geschlecht <http://example.com/ont#maennlich> ;
  ont:name "Frank Schmidt" ;
  ont:phone "030286622" .

<http://mueller.example/people/richard>
  ont:geschlecht <http://example.com/ont#maennlich> ;
  ont:name "Richard Mueller" ;
  ont:phone "030344554" .
```

Listing A.5: Instanzen eines RDF-Schemas im N3-Format

A.8 PELDS-OWL-Vokabular

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY ont "http://www.pelds.org/ont">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY pelds "http://www.pelds.org/ont#">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&ont;"
```

```
        xmlns:owl="&owl;"
        xmlns:rdf="&rdf;"
        xmlns:rdfs="&rdfs;"
        xmlns:pelds="&pelds;">

  <owl:Ontology rdf:about="" rdfs:label="PeLDS Ontology"/>
  <owl:Class rdf:about="http://www.w3.org/2003/11/swrl#Imp"/>
  <owl:Class rdf:about="http://www.w3.org/2003/11/swrl#Variable
      "/>

<!-- Action class -->
  <owl:Class rdf:about="#Action"/>

<!-- Action properties -->
  <owl:ObjectProperty rdf:about="#actor">
    <rdfs:domain rdf:resource="#Action"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#matchedRule">
    <rdfs:domain rdf:resource="#Action"/>
    <rdfs:range rdf:resource="#Rule"/>
  </owl:ObjectProperty>

<!-- Action subclasses -->
  <owl:Class rdf:about="#QueryAction">
    <rdfs:subClassOf rdf:resource="#Action"/>
  </owl:Class>

  <owl:Class rdf:about="#UpdateAction">
    <rdfs:subClassOf rdf:resource="#Action"/>
  </owl:Class>


<!-- Rule class -->
  <owl:Class rdf:about="#Rule">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2003/11/
        swrl#Imp"/>
  </owl:Class>

<!-- Rule properties -->
  <owl:ObjectProperty rdf:about="#triplePatterns">
    <rdfs:domain rdf:resource="#Rule"/>
  </owl:ObjectProperty>
```

```
<!-- TriplePattern class -->
  <owl:Class rdf:about="#TriplePattern"/>

<!-- TriplePattern properties -->
  <owl:DatatypeProperty rdf:about="#tripleSubject" rdf:type="
      http://www.w3.org/2002/07/owl#FunctionalProperty">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#
        string"/>
    <rdfs:domain rdf:resource="#TriplePattern"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#triplePredicate" rdf:type="
      http://www.w3.org/2002/07/owl#FunctionalProperty">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#
        string"/>
    <rdfs:domain rdf:resource="#TriplePattern"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#tripleObject" rdf:type="
      http://www.w3.org/2002/07/owl#FunctionalProperty">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#
        string"/>
    <rdfs:domain rdf:resource="#TriplePattern"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#instance" rdf:type="http://
      www.w3.org/2002/07/owl#FunctionalProperty">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#
        string"/>
    <rdfs:domain rdf:resource="#TriplePattern"/>
  </owl:DatatypeProperty>

</rdf:RDF>
```

Listing A.6: Modellierung von Zugriffsrichtlinien in OWL im RDF/XML-Format

Literaturverzeichnis

[AKSX02] Agrawal, Rakesh ; Kiernan, Jerry ; Srikant, Ramakrishnan ; Xu, Yirong: Hippocratic Databases. In: *28th International Conference on Very Large Data Bases*, 2002

[Apa09a] Apache Software Foundation: *Apache HTTP Server*. `http://httpd.apache.org/`. Version: 11 2009

[Apa09b] Apache Software Foundation: *Apache Tomcat*. `http://tomcat.apache.org/`. Version: 10 2009

[BB08] Beckett, Dave ; Broekstra, Jeen: *SPARQL Query Results XML Format*. `http://www.w3.org/TR/rdf-sparql-XMLres/`. Version: 01 2008

[BCH09] Bizer, Chris ; Cyganiak, Richard ; Hartig, Olaf: *NG4J - Named Graphs API for Jena*. `http://www4.wiwiss.fu-berlin.de/bizer/ng4j/`. Version: 10 2009

[BGM04] Brickley, Dan ; Guha, R.V. ; McBride, Brian: *RDF Vocabulary Description Language*. `http://www.w3.org/TR/rdf-schema/`. Version: 02 2004

[BHIBL08] Bizer, Christian ; Heath, Tom ; Idehen, Kingsley ; Berners-Lee, Tim: Linked data on the web (LDOW2008). In: *WWW '08: Proceeding of the 17th international conference on World Wide Web*. New York, NY, USA : ACM, 2008. – ISBN 978–1–60558–085–2, S. 1265–1266

[BHK+07] Boley, Harold ; Hallmark, Gary ; Kifer, Michael ; Paschke, Adrian ; Polleres, Axel ; Reynolds, Dave: *RIF Core Dialect*. `http://www.w3.org/TR/rif-core/`. Version: 2009 07

[BK07] Boley, Harold ; Kifer, Michael: *RIF Basic Logic Dialect*. `http://www.w3.org/TR/rif-bld/`. Version: 2009 07

[BL98a] Berners-Lee, Tim: *Notation 3*. `http://www.w3.org/DesignIssues/Notation3.html`. Version: 1998

[BL98b] Berners-Lee, Tim: *Relational Databases on the Semantic Web*. `http://www.w3.org/DesignIssues/RDB-RDF.html`. Version: 09 1998

[BL06] Berners-Lee, Tim: *Linked Data*. http://www.w3.org/DesignIssues/LinkedData.html. Version: 07 2006

[BLHL01] Berners-Lee, Tim ; Hendler, James ; Lassila, Ora: *The Semantic Web*. http://www.scientificamerican.com/article.cfm?id=the-semantic-web. Version: 05 2001

[BLMM94] Berners-Lee, T. ; Masinter, L. ; McCahill, M.: *RFC 1738 - Uniform Resource Locators (URL)*. http://www.ietf.org/rfc/rfc1738.txt. Version: 12 1994

[BM04] Beckett, Dave ; McBride, Brian: *RDF/XML Syntax Specification*. http://www.w3.org/TR/rdf-syntax-grammar/. Version: 02 2004

[BS09] Bizer, Christian ; Schultz, Andreas: The Berlin SPARQL Benchmark. In: *International Journal On Semantic Web and Information Systems - Special Issue on Scalability and Performance of Semantic Web Systems* (2009)

[CFT08] Clark, Kendall G. ; Feigenbaum, Lee ; Torres, Elias: *SPARQL Protocol for RDF*. http://www.w3.org/TR/rdf-sparql-protocol/. Version: 01 2008

[Cla09] Clark and Parsia: *Pellet: The Open Source OWL Reasoner*. http://clarkparsia.com/pellet/. Version: 10 2009

[CSF+08] Cooper, D. ; Santesson, S. ; Farrell, S. ; Boeyen, S. ; Housley, R. ; Polk, W.: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. http://tools.ietf.org/html/rfc5280. Version: 05 2008

[DA99] Dierks, T. ; Allen, C.: *RFC 2246 - The TLS Protocol*. http://tools.ietf.org/html/rfc2246. Version: 01 1999

[DHS07] Duma, Claudiu ; Herzog, Almut ; Shahmehri, Nahid: Privacy in the Semantic Web: What Policy Languages Have to Offer. In: *Policies for Distributed Systems and Networks, IEEE International Workshop on* 0 (2007), S. 109–118. ISBN 0–7695–2767–1

[Dum02] Dumbill, Edd: *XML Watch: Finding friends with XML and RDF*. http://www.ibm.com/developerworks/xml/library/x-foaf.html. Version: 01 2002

[DWT82] Doherty, Walter ; Watson, Thomas ; Thadani, Ahrvind: *The Economic Value of Rapid Response Time*. http://www.vm.ibm.com/devpages/jelliott/evrrt.html. Version: 1982

[Fie00] Fielding, Roy T.: *Architectural Styles and the Design of Network-based Software Architectures*, University of California, Irvine, Diss., 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

[FK92] Ferraiolo, David ; Kuhn, Richard: Role-Based Access Control. In: *Proceedings of 15th National Computer Security Conference*, 1992

[Fre09] Freytag, Johann-Christoph: *Context Quality and Privacy - Friends or Rivals?* 2009. – QuaCon 2009 submission

[Hef04] Heflin, Jeff: *OWL Web Ontology Language - Use Cases and Requirements*. `http://www.w3.org/TR/2004/REC-webont-req-20040210/#onto-def`. Version: 02 2004

[Hen01] Hendler, James: Agents and the Semantic Web. In: *IEEE Intelligent Systems* 16 (2001), Nr. 2, S. 30–37. `http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/5254.920597`. – DOI http://doi.ieeecomputersociety.org/10.1109/5254.920597. – ISSN 1541–1672

[Hew09a] Hewlett-Packard Development Company: *Jena – A Semantic Web Framework for Java*. `http://jena.sourceforge.net/`. Version: 10 2009

[Hew09b] Hewlett-Packard Development Company: *Joseki - A SPARQL Server for Jena*. `http://www.joseki.org/`. Version: 10 2009

[HMF09] Hartig, Olaf ; Mühleisen, Hannes ; Freytag, Johann-Christoph: *Linked Data for Building a Map of Researchers*. 2009. – Wettbewerbsbeitrag bei der SFSW2009

[HPSB+05] Horrocks, Ian ; Patel-Schneider, Peter F. ; Boley, Harold ; Tabet, Said ; Grosof, Benjamin ; Dean, Mike: *SWRL: A Semantic Web Rule Language*. `http://www.w3.org/Submission/SWRL/`. Version: 2004 05

[HTT08] *Kapitel* HTTPS Client Authentication. In: *The Java EE 5 Tutorial*. Sun Microsystems, 2008, 865-868

[Int96] International Organization for Standardization: *ISO/IEC 14977 - Extended Backus-Naur Form*. 1996

[Int99] Internet Engineering Task Force: *RFC 2616 - Hyptertext Transfer Protocol*. `http://www.ietf.org/rfc/rfc2616.txt`. Version: 1999

[Int00] Internet Engineering Task Force: *RFC 2818 - HTTP Over TLS*. `http://tools.ietf.org/html/rfc2818`. Version: 2000

[Int05] Internet Engineering Task Force: *RFC 4033 - DNS Security Introduction and Requirements*. `http://tools.ietf.org/html/rfc4033`. Version: 2005

[Int06] Internet Engineering Task Force: *The application/json Media Type for JavaScript Object Notation (JSON)*. `http://www.ietf.org/rfc/rfc4627.txt`. Version: 07 2006

[Int08] International Organization for Standardization: *ISO/IEC 2382 - Part 36: Learning, education and training.* 2008

[JF06] Jain, Amit ; Farkas, Csilla: Secure Resource Description Framework: an Access Control Model. In: *SACMAT'06*, 2006

[KH00] Kudo, M. ; Hada, S.: XML document security based on provisional authorization. In: *Proceedings of the 7th ACM Conference on Computer and Communications Security*, 2000

[Lat85] Latham, Donald C.: *Trusted Computer System Evaluation Criteria (Orange Book).* Department of Denfense, 1985

[Lew08] Lewis, Daniel J.: Intelligent agents and the Semantic Web. In: *IBM developerWorks - Web development - Technical library* (2008)

[LS03] Lee, Jae K. ; Sohn, Mye M.: The eXtensible Rule Markup Language. In: *Commun. ACM* 46 (2003), Nr. 5

[MBK08] Marsh, Steve ; Brown, Ian ; Khaki, Fayaz: *Privacy Engineering Whitepaper.* `http://www.ktn.qinetiq-tim.net/content/files/groups/privacy/CSKTN_Privacy_SIG_White_Paper_08.pdf`. Version: 2008. – A Report from a Special Interest Group of the Cyber Security KTN

[MD09] Mealling, M. ; Denenberg, R.: *RFC 3305 - Uniform Resource Identifiers.* `http://www.rfc-editor.org/rfc/rfc3305.txt`. Version: 06 2009

[MH04] McGuinness, Deborah L. ; Harmelen, Frank van: *OWL Web Ontology Language - Overview.* `http://www.w3.org/TR/owl-features/`. Version: 02 2004

[MMM04] Manola, Frank ; Miller, Eric ; McBride, Brian: *RDF Primer.* `http://www.w3.org/TR/rdf-primer/`. Version: 02 2004

[MSS05] Motik, Boris ; Sattler, Ulrike ; Studer, Rudi: Query Answering for OWL-DL with rules. In: *Web Semantics: Science, Services and Agents on the World Wide Web* (2005), 07

[Now09] Nowack, Benjamin: *ARC RDF Classes for PHP.* `http://arc.semsol.org`. Version: 11 2009

[PS08] Prud'hommeaux, Eric ; Seaborne, Andy: *SPARQL Query Language for RDF.* `http://www.w3.org/TR/rdf-sparql-query/`. Version: 01 2008

[RFJ07] Reddivari, Pavan ; Finin, Tim ; Joshi, Anupam: Policy-Based Access Control for an RDF Store. In: *Proceedings of the IJCAI-07 Workshop on Semantic Web for Collaborative Knowledge Acquisition*, 2007

[RSA78] Rivest, R. L. ; Shamir, A. ; Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. In: *Commun. ACM* 21 (1978), Nr. 2, S. 120–126. http://dx.doi.org/http://doi.acm.org/10.1145/359340.359342. – DOI http://doi.acm.org/10.1145/359340.359342. – ISSN 0001–0782

[SHJJ09] Story, Henry ; Harbulot, Bruno ; Jacobi, Ian ; Jones, Mike: *FOAF+SSL: RESTful Authentication for the Social Web*. 2009. – eingereicht bei Semantic Web Conference 2009

[SMB+08] Seaborne, Andy ; Manjunath, Geetha ; Bizer, Chris ; Breslin, John ; Das, Souripriya ; Davis, Ian ; Harris, Steve ; Idehen, Kingsley ; Corby, Olivier ; Kjernsmo, Kjetil ; Nowack, Benjamin: *SPARQL Update*. http://www.w3.org/Submission/2008/SUBM-SPARQL-Update-20080715/. Version: 07 2008

[Sto09] Story, Henry: *Sommer Project*. https://sommer.dev.java.net/. Version: 10 2009

[Sun09a] Sun Microsystems: *Developer Resources for Java Technology*. http://java.sun.com/. Version: 11 2009

[Sun09b] Sun Microsystems: *MySQL Enterprise Server*. http://mysql.com/products/enterprise/server.html. Version: 10 2009

[The09] The PHP Group: *PHP: Hypertext Preprocessor*. http://www.php.net. Version: 11 2009

[VOO95] Virdhagriswaran, Sankar ; Osisek, Damian ; O'Connor, Pat: Standardizing agent technology. In: *StandardView* 3 (1995), Nr. 3, S. 96–101. http://dx.doi.org/http://doi.acm.org/10.1145/226191.226200. – DOI http://doi.acm.org/10.1145/226191.226200. – ISSN 1067–9936

[WB90] Warren ; Brandeis: The Right to Privacy. In: *Harvard Law Review* (1890)

[Wei09] Weisstein, Eric W.: *Least Squares Fitting*. http://mathworld.wolfram.com/LeastSquaresFitting.html. Version: 11 2009

[Wes67] Westin, A. F.: *Privacy and Freedom*. Atheneum, 1967

[Wor09] World Wide Web Consortium: *Semantic Web Project Page*. http://www.w3.org/2001/sw/. Version: 06 2009

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

BSBM	Berlin SPARQL Benchmark
DAC	Discretionary Access Control
DNS	Domain Name System
EBNF	Erweitereten Backus-Naur-Form
FOAF	Friend of a Friend
GUI	Grafische Benutzerschnittstelle
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
MAC	Mandatory Access Control
N3	Notation 3
NG4J	Named Graphs for Jena
OWL	Web Ontology Language
PeLDS	Policy enabled Linked Data Server
PsSF	Prolog-style SWRL Format
RBAC	Role-Based Access Control
RDF	Resource Description Framework
RDFS	RDF Schema
REST	Representational State Transfer
RIF	Rule Interchange Format
SPARQL	SPARQL Protocol and RDF Query Language
SPKAC	Signed Public Key And Challenge
SWRL	Semantic Web Rule Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WWW	World Wide Web

Anhang B

Dank

Dank gilt allen, die mich soweit gebracht haben, dass ich diese Arbeit schreiben konnte, und allen, die in irgendeiner Form an dieser Diplomarbeit mitgewirkt haben:

- Den Gutachtern Prof. Freytag und Prof. Redlich für ihre Zeit
- Martin Kost für seine geduldige Betreuung
- Olaf Hartig für die ersten Literaturhinweise
- Tim Reiner für die Korrektur des Entwurfs und die darin enthaltenen Illustrationen
- Tabea Freutel für die Aufmunterung, als diese nötig war
- Kornelia und Rolf für die vielfältige Unterstützung in den letzten 25 Jahren

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit mit dem Titel *"Zugriffsrichtlinien und Authentifizierung für Linked-Data-Systeme"* selbständig und ohne unerlaubte Hilfe verfasst und nur die angegebenen Quellen verwendet habe.

Berlin, den 13. März 2010

Hannes Mühleisen

Einverständniserklärung

Ich bin damit einverstanden, dass die vorliegende Arbeit mit dem Titel *"Zugriffsrichtlinien und Authentifizierung für Linked-Data-Systeme"* in der Bibliothek ausgelegt wird.

Berlin, den 13. März 2010

Hannes Mühleisen