Reaktive Terminorganisation im Semantic Web

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Mag. rer. soc. oec.

im Diplomstudium Wirtschaftsinformatik

Angefertigt am Institut für Wirtschaftsinformatik – Data & Knowledge Engineering

Betreuung / Begutachter:

o. Univ.-Prof. Dipl.-Ing. Dr. Michael Schrefl

Mitbetreuung:

Mag. Dr. Michael Huemer

Eingereicht von:

Bernhard Leonhardsberger

Linz, im Juli 2015

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.
Die vorliegende Diplomarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, Juli 2015

_____

Bernhard Leonhardsberger

Danksagung

An dieser Stelle möchte ich mich bei all jenen bedanken, die mich bei der Entstehung dieser Diplomarbeit unterstützt haben.

Ich bedanke mich bei Herrn o. Univ.-Prof. Dipl.-Ing. Dr. Michael Schrefl für seine eingebrachten Ideen und konstruktiven Anmerkungen in den Diplomandenseminaren. Ein besonderer Dank gilt meinem Betreuer Mag. Dr. Michael Huemer, der mich erheblich beim Aufbau und Inhalt dieser Diplomarbeit unterstützt hat. Er trug durch seine fachliche und engagierte Betreuung sowie seinen Ratschlägen wesentlich zu dieser Arbeit bei.

Weiters möchte ich mich bei meiner Familie bedanken, die mich moralisch unterstützte und durch ihre motivierenden Worte die Entstehung dieser Arbeit förderte. Hierbei möchte ich besonders meine Eltern hervorheben, die mir mit viel Geduld und Verständnis während des Studiums entgegengekommen sind.

Kurzfassung

Das World Wide Web besteht aus einer enormen Menge an Daten und Informationen über Dinge und Personen aus der realen Welt, welche durch Hyperlinks miteinander verbunden sind. Diese Informationen sind vorwiegend für den Menschen aufbereitet, nicht jedoch für die maschinelle Verarbeitung. Maschinen haben Probleme dabei, Daten und Informationen aus einem bestimmten Kontext in Verbindung zu bringen. Das Semantic Web soll sie dabei unterstützen, Verknüpfungen und Zusammenhänge selbst zu erkennen und dadurch eine Automatisierung von bestimmten Aufgaben ermöglichen. Diese Automatisierung wird in relationalen Datenbanken mittels Trigger umgesetzt, welche im Web eine analoge Funktionalität in Regeln wiederfinden. Mit Hilfe dieser Regeln werden nun reaktive und proaktive Handlungen seitens einer Maschine ermöglicht und sie unterstützen den Menschen bei der Handhabung, Wartung und der Überwachung von Abläufen, Prozessen und Anwendungen. Eine besondere Herausforderung im World Wide Web ist die Arbeit mit verteilten Daten, denn dabei können Probleme in Verbindung mit Verfügbarkeit, Datenkonsistenz und Zugriffsrechten entstehen.

In dieser Arbeit wird mit Hilfe eines speziell für den Einsatz im Semantic Web entworfenen Frameworks, dem Resource Description Framework, eine Webanwendung entwickelt, welche den Benutzer beim Planen von Aktivitäten und Terminen unterstützen soll. Diese Anwendung soll intuitiv erwartete Aufgaben automatisch erledigen und auf vom Benutzer vordefinierte und gewünschte Ereignisse eine bestimmte Handlung automatisiert durchführen. Aus den vorgegebenen Rahmenbedingungen und Anforderungen werden ein Lösungskonzept und ein Lösungsvorschlag erarbeitet und anschließend umgesetzt. Der Fokus liegt dabei auf der Umsetzung der reaktiven Verhaltensweise der Anwendung in Verbindung mit einer verteilten Datenhaltung. Dabei auftretende Grenzen und Einschränkungen werden aufgezeigt und Lösungsvorschläge dafür erarbeitet.

Abstract

The World Wide Web is filled with a vast amount of data and information describing people and things that exist in the real world. This information is connected via hyperlinks and is rather designed for people to understand than machines to handle. Machines have difficulties merging distributed data together, knowing nothing else about this data other than the address. The semantic web supports machines in creating links between data which is distributed around the world and allows automation through this existing links in various situations. This automation in the web is still a big challenge, while in relational databases this concept of automation is already implemented via triggers. To use triggers in the World Wide Web, several concepts have been published. One concept in particular is using event-condition-action rules, which allows a reactive behaviour from a machine, being then able to handle tasks by themselves which otherwise would have to be solved by a human. A specific challenge in the World Wide Web is operating with distributed data, because problems concerning data availability, data consistency and access rights are likely to occur.

In this thesis, a web application will be presented, which allows a user to handle appointments and personal activities. This web application uses the Resource Description Framework (RDF) and will combine RDF and event-condition-action rules, resulting in an intuitive behaviour supporting the user. The user himself can set requirements for the application when to act.

After discussing the requirements of the application, an approach to reactive calendar management together with a proof-of-concept prototype – with the focus on the development of the reactive behaviour using distributed data – will be presented. Challenges and problems will be identified and possible solutions will be presented.

Inhaltsverzeichnis

Abkürzungen

bzw.	beziehungsweise
d.h.	das heißt
engl.	englisch
FOAF	Friend of a Friend
HTTP	Hypertext Transfer Protocol
JSP	Java Server Page
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
SPARQL	SPARQL Protocol and RDF Query Language
UML	Unified Modelling Language
URI	Unified Resource Identifier
W3C	World Wide Web Consortium
WWW	World Wide Web
XFN	XHTML Friends Network
XML	Extensible Markup Language
z.B.	zum Beispiel

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Das World Wide Web, kurz Web, wurde mit dem Gedanken einer vernetzten Informationsquelle ins Leben gerufen [Bern98]. Die Informationsdokumente werden durch Hyperlinks verbunden und bilden die Grundlage der vernetzten Struktur. Das Web diente in erster Linie zur Präsentation von Informationen durch Maschinen für den Menschen. Durch das unaufhaltbare Hinzufügen unzähliger Daten, wird es jedoch sowohl für den Menschen als auch für die Maschine zunehmend schwieriger, die gesuchte Information zu finden. Dadurch, dass die Präsentation der Daten hauptsächlich für den Menschen gedacht ist, erkennt der Mensch Zusammenhänge zwischen verteilten Datenquellen im Web anhand des Inhalts. Maschinen hingegen können zwar den Hyperlinks folgen, erkennen jedoch keinen Zusammenhang anhand des Inhalts von Dokumenten.

Dieses Problem führte zur Weiterentwicklung des Webs aus der ursprünglichen Form in das Semantic Web. Das Semantic Web ist eine von Tim Berners-Lee publizierte Idee, um für Menschen lesbare Dokumente im Web auch für Maschinen interpretierbar zu machen. [BeHL01] [SnBH06] Um den Inhalt der Dokumente näher zu beschreiben, werden Metadaten eingesetzt. Da im Web viele verteilte heterogene Daten entstanden sind, musste diese Beschreibung der Dokumente vereinheitlicht werden. Die W3C hat dazu das Resource Description Framework, kurz RDF, spezifiziert. [Beck04] [HaPa14] Mit RDF ist es unter anderem möglich, Beziehungen zwischen verteilten Daten auszudrücken. Sichergestellt wird dies mit Hilfe von Unified Resource Identifiers (URIs). URIs liefern für die Maschine eine Möglichkeit, semantisch zusammenhängende aber physisch getrennte Daten weiterzuverfolgen. Hierbei werden im Gegensatz zum Web keine Dokumente sondern Daten verknüpft.

Das Web untersteht einem regen Wandel an (neuer) Information und kann deshalb nicht als statische Informationsdatenbank angesehen werden. Es ist daher, insbesondere bei Anwendungen im Bereich E-Learning, E-Science, E-Government, etc. welche reaktiv[1] sein sollen, notwendig, dass Änderungen erkannt werden und darauf reagiert wird. In relationalen Datenbanken wird auf Zustandsänderungen mittels Trigger reagiert. Im Semantic Web, im Besonderen in Verbindung mit RDF, kann man dafür aktive Regeln einsetzen. Event-Condition-Action (ECA) Regeln sind den Triggern ähnlich und erlauben es der Maschine, beim Eintreten eines bestimmten Events(E) eine Bedingung(C) zu überprüfen und anhand der Überprüfung der Bedingung eine Aktion(A) durchzuführen [PaPW06]. Dieses automatische Handeln war zentraler

[1] Auf das Eintreten von Änderungen oder bestimmter Events automatisch reagieren und die daraus richtigen Folgeentscheidungen treffen.

Gedanke in der Beschreibung der Semantic Web Idee [BeHL01] und soll das Web in die nächste Ära der Mensch Maschine Kommunikation heben.

Im Zuge dieser Arbeit soll die Vereinbarkeit einer verteilten semantischen Datenhaltung mit einer regelbasierten Anwendungslogik, welche automatisch auf Änderungen reagiert, anhand eines reaktiven Terminkalenders gezeigt werden.

1.1. Aufgabenstellung und Zielsetzung

Aufgabenstellung dieser Arbeit ist die Entwicklung einer reaktiven Terminorganisation im Semantic Web. Ausgangslage sind dabei verteilte Ressourcen[2] (Personen, Gegenstände oder Räume), welche mithilfe dieses Dienstes sich gegenseitig zu Terminen einladen können. Durch den Einsatz von (durch die Ressourcen vordefinierten) Regeln wird der Dienst gezwungen, reaktiv und halb-automatisch zu reagieren. Auf eintretende Events durch Ressourcen agiert die Terminorganisation proaktiv und unterstützt alle bei diesem Termin beteiligten Ressourcen. Dabei werden individuelle (durch die Ressourcen vordefinierte) Prioritäten berücksichtigt und in den Entscheidungsprozess eingebunden. Jeder Benutzer soll selbst über seine Daten verfügen können. Folglich sind die zu verwendenden Daten über das Internet verstreut und als verteilt zu betrachten.

Folgende funktionale und nicht funktionale Anforderungen werden festgelegt um die Aufgabenstellung abbilden zu können:

- Personen können Termine zu ihrem Kalender hinzufügen
- bei der Erstellung eines Termins, können mehrere mögliche Abhaltungen für diesen Termin vorgeschlagen werden
- es können nicht nur Personen, sondern auch andere Ressourcen wie Gegenstände oder Räume an Terminen teilnehmen
- jede Person kann Freundesgruppen mit zugehörigen Prioritäten anlegen
- Ressourcen können zu einem Termin zusagen, absagen oder einen fixierten Termin abbrechen
- das System reagiert reaktiv und (halb-)automatisch auf eintretende Ereignisse
- das System reagiert proaktiv auf Veränderungen
- die Anwendung soll Web-basiert sein
- es werden verteilte, semantische Daten verwendet

[2] In dieser Arbeit wird mit Ressource immer eine ‚non-information resource', definiert in [BiCH07], bezeichnet.

- es wird das Resource Description Framework (RDF) verwendet
- einzelne Teilbereiche werden so fern möglich generisch gehalten um eine Erweiterbarkeit zu gewährleisten
- die Sichtbarkeit von privaten Daten wird eingeschränkt; es wird zwischen Informationen für Freunde, Nicht-Freunde und Informationen, welche nur für den Benutzer sichtbar sind, unterschieden

Ziel dieser Arbeit ist es, eine Webanwendung zu schaffen, um semantischen Dateninhalt in verteilten Ressourcen mit aktiven Regeln zu kombinieren und deren Einsatz anhand des umgesetzten Dienstes zu zeigen. Eine verteilte Verarbeitung von Ereignissen und Regeln ist nicht Teil dieser Arbeit.

1.2. Anwendungsfall

Die drei Teilnehmer[3] eines Termins, John Grady, Matthew Sobol und Peter Sebeck, werden im folgenden Beispiel eine Terminvereinbarung simulieren. Es befinden sich bereits Termine, Regeln und Prioritäten in den persönlichen Kalendern dieser drei Teilnehmer. Jeder Teilnehmer ist bei den jeweiligen anderen Personen in mindestens einer Freundesgruppe, welche eine gewisse Priorität besitzt, und der Teilnehmer kann in bestimmten Regeln eingebunden sein. Die Kalender der Teilnehmer sehen folgendermaßen aus:

John Grady's Kalender:

John Grady besitzt die Freundesgruppen „Arbeitskollegen" mit der Priorität „7" und „Schachfreunde" mit der Priorität „4". Matthew Sobol befindet sich sowohl in der Gruppe Arbeitskollegen als auch in der Gruppe Schachfreunde. Peter Sebeck befindet sich in der Gruppe Schachfreunde. John Grady hat folgende Termine fixiert:

Termin 1		Termin 2	
Bezeichnung	Jour fixe – Projekt X	**Bezeichnung**	Nachhilfe Spanisch
Start	05.05.2014, 13:30 Uhr	**Start**	05.05.2014, 17:00 Uhr
Ende	05.05.2014, 14:30 Uhr	**Ende**	05.05.2014, 18:00 Uhr
Kategorie	geschäftlich	**Kategorie**	privat
Status	fixiert	**Status**	fixiert
Ort	Besprechungszimmer 2. Stock	**Ort**	zuhause
Ersteller	John Grady	**Ersteller**	John Grady
Teilnehmer	John Grady	**Teilnehmer**	John Grady

[3] Da in der deutschen Sprache durch den generischen Maskulin beide Geschlechter gleichermaßen miteinbezogen werden, wird in dieser Arbeit auf ein angehängtes „Innen" und dergleichen verzichtet.

	Matthew Sobol		Nora Hayden
	Sekretärin Margit		
	Mitarbeiter Hajime		
	Mitarbeiterin Shimamoto		

Tabelle 1 Terminkalender von John Grady

Matthew Sobol's Kalender:

Matthew Sobol besitzt eine Freundesgruppe „Kollegen" mit der Priorität „5". In dieser Gruppe befindet sich John Grady. Matthew Sobol hat zurzeit folgenden Termin fixiert:

Termin 3	
Bezeichnung	Seminar
Start	05.05.2014, 15:00 Uhr
Ende	05.05.2014, 18:30 Uhr
Kategorie	geschäftlich
Status	fixiert
Ort	Seminarraum 4711
Ersteller	Sekretärin Margit
Teilnehmer	Matthew Sobol
	Sekretärin Margit
	Mitarbeiterin Shimamoto

Tabelle 2 Terminkalender von Matthew Sobol

Matthew Sobol hat eine Regel erstellt, dass bei Terminkollisionen der neue Termin automatisch abgesagt werden soll.

Peter Sebeck's Kalender:

Peter Sebeck besitzt die Freundesgruppen „Schachclub" mit der Priorität „2" und die Freundesgruppe „Familie" mit der Priorität „9". John Grady befindet sich in der Freundesgruppe Schachclub, während sich Peter Sebecks' Frau und seine Kinder in der Freundesgruppe Familie befinden. Peter Sebeck hat folgenden fixierten Termin.

Termin 4	
Bezeichnung	Familienausflug
Start	03.05.2014, 08:00 Uhr
Ende	05.05.2014, 16:00 Uhr
Kategorie	privat
Status	fixiert
Ort	München
Ersteller	Frau Anna
Teilnehmer	Frau Anna
	Tochter Yasmin
	Sohn Raphael

Tabelle 3 Terminkalender von Peter Sebeck

Peter Sebeck hat eine Regel erstellt, welche Terminvorschläge automatisch ablehnen soll, wenn er zum eingeladenen Zeitpunkt bereits einen Termin hat, und die Priorität derjenigen Person die ihn einlädt niedriger ist, als jene derjenigen Person die den bestehenden Termin erstellt hat.

John Grady möchte sich mit seinen Freunden Matthew und Peter zu einem Essen treffen und schlägt zwei mögliche Termine vor (vgl. Abbildung 1).

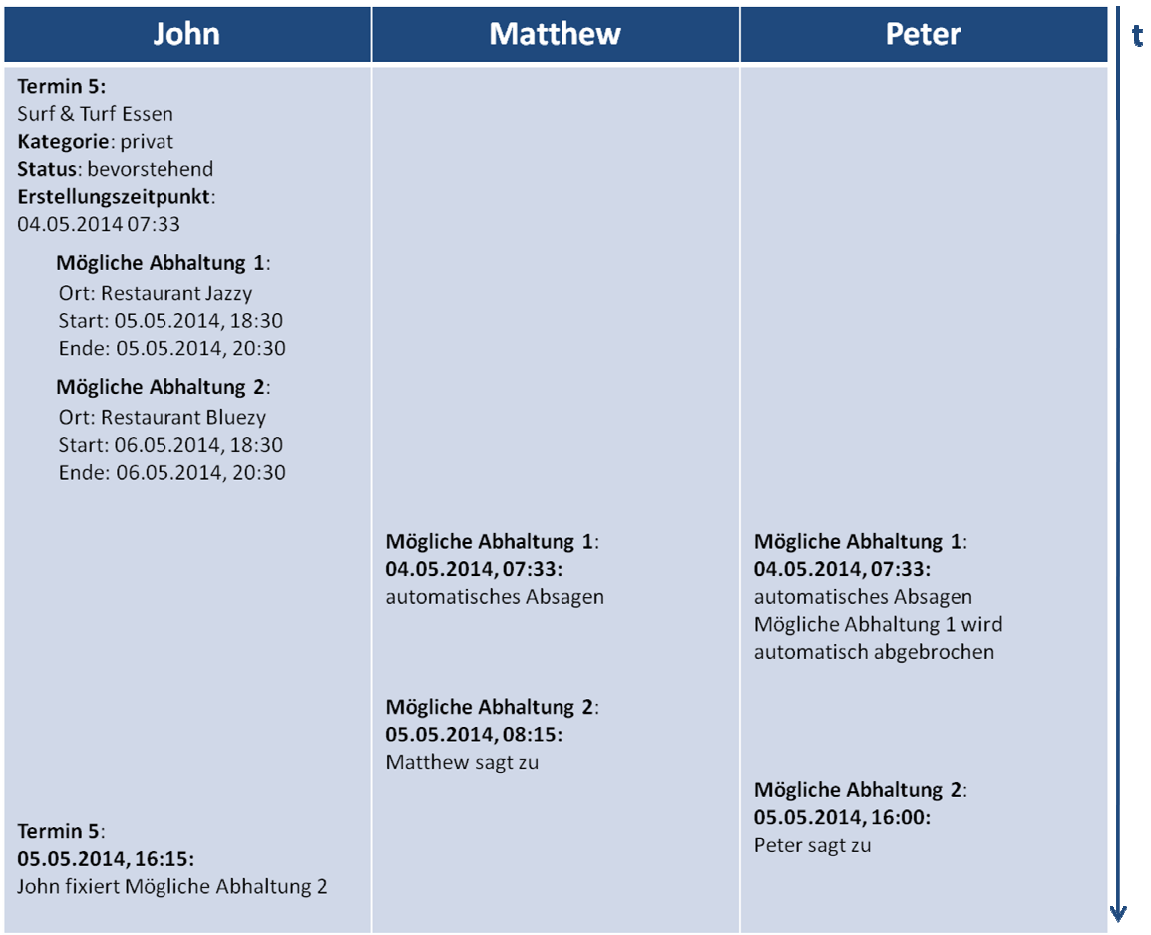


Abbildung 1 Beispiel einer möglichen Terminvereinbarung

Nach der Erstellung der Vorschläge am 04.05.2014 um 07:33 Uhr erkennt das System automatisch, dass Matthew zu diesem Zeitpunkt bereits ein Seminar hat. Die von Matthew angelegte Regel besagt, dass der neue Termin bei einer Kollision automatisch abgelehnt werden soll. Somit wird die Mögliche Abhaltung 1 nach der Erstellung automatisch abgesagt. Peter befindet sich zum Zeitpunkt der ersten möglichen Abhaltung noch auf einem Familienausflug, welcher von seiner Frau Anna erstellt wurde. Seine Frau Anna hat bei Peter eine Priorität von „9“, während John bei Peter eine Priorität von „2“ hat. Die von Peter angelegte Regel besagt, dass bei Terminkollisionen der neue Termin automatisch abgesagt werden soll, wenn die Priorität nicht höher als die des Erstellers des bestehenden Termins ist. In diesem Fall wird somit die Mögliche Abhaltung 1 automatisch abgesagt.

Da mit Peter auch der letzte verbleibende Teilnehmer dieser möglichen Abhaltung abgesagt hat, wird dieser mögliche Termin automatisch abgebrochen. Am folgenden Tag sagen sowohl Matthew als auch Peter zum zweiten Terminvorschlag zu und John fixiert den Termin 5 mit der zweiten möglichen Abhaltung im Restaurant Bluezy für den 06.05.2014 um 18:30 Uhr, am 05.05.2014 um 16:15 Uhr.

1.3. Aufbau der Arbeit

Zu Beginn der Arbeit wird in Kapitel 2 auf grundlegende Konzepte und Technologien des Semantic Webs näher eingegangen. Nach einem Überblick über Grundlagen des Hypertext Transfer Protocols und des Semantic Webs werden für die Arbeit relevante Bereiche wie das Jena Framework und Event-Condition-Action Regeln näher erläutert.

In Kapitel 3 wird der Lösungsansatz für die Aufgabenstellung näher beschrieben. Es wird auf die konzeptuelle Ebene zur Entwicklung der Webanwendung näher eingegangen. Es werden Themenbereiche abgegrenzt und zu erwartende Probleme beschrieben.

In Kapitel 4 wird der vorgestellte Lösungsansatz aus logischer Sicht näher betrachtet. Dabei wird auf die Implementierung in RDF eingegangen und das erstellte Vokabular näher erläutert.

Kapitel 5 befasst sich mit dem Design und den in der Umsetzung zu verwendenden architektonischen Komponenten. Es wird die Aufgabe jeder Komponente beschrieben und auf deren Funktion näher eingegangen.

Kapitel 6 bezieht sich auf die Implementierung selbst. Nach einem kurzen Überblick über verwendete Werkzeuge wird die entwickelte Webanwendung beschrieben und detailliert behandelt.

In Kapitel 7 zeigt eine Zusammenfassung noch einmal kurz die Ziele und inwieweit diese realisiert werden konnten auf. Anschließend folgt ein Ausblick auf mögliche Erweiterungen.

Im Anhang werden Testdaten, Codedetails und genaue Programmversionen angeführt und beschrieben.

2. Zugrundeliegende Konzepte und Technologien

Im folgenden Kapitel wird auf die in der Arbeit verwendeten Technologien näher eingegangen. Beginnend mit den unterschiedlichen Methoden der zustandslosen Übertragung von Daten mittels des Hypertext Transfer Protocols wird anschließend auf das Semantic Web generell und im Detail auf die Datenrepräsentation in RDF und die Abfragesprache SPARQL eingegangen. Für die Erweiterung um Regeln werden Event-Condition-Action Regeln im Grundansatz und im Speziellen näher erläutert. Abschließend wird das Java-Framework JENA kurz erklärt und dessen Arbeitsweise betrachtet.

2.1. Hypertext Transfer Protocol

Um die Kommunikationsmöglichkeiten zwischen einem Client und einem Server im World Wide Web zu vereinheitlichen, hat Tim Berners-Lee 1991/92 das Hypertext Transfer Protocol, kurz HTTP, vorgestellt. HTTP ist ein generisches, zustandsloses und objektorientiertes Protokoll, welches Websystemen erlaubt, unterschiedliche Repräsentationen von Dateiinhalten darauf aufbauend zu verwenden [Bern92]. Später (1999) wurde es vom World Wide Web Consortium im Standard RFC 2616 [FUGC+99] näher definiert. Als Kommunikationseinheit zwischen dem Client und dem Server wird eine Nachricht (eine sogenannte ‚`HTTP-message`') verwendet, wie im folgenden Beispiel angeführt wird. Eine Nachricht setzt sich aus dem Nachrichtenkopf (`Message-Header`) und dem Nachrichtenkörper (`Message-Body`) zusammen [FUGC+99]. Eine Nachricht ist entweder die Anfrage eines Clients an einen Server (`Request`) oder die Antwort des Servers (`Response`) auf die Anfrage des Clients.

```
…
HTTP-message    = Request | Response      ; HTTP/1.1 messages
…
```

Die Kommunikationseinheit ist Teil einer Transaktion, welche sich aus den folgenden Bestandteilen zusammensetzt:

1. Verbindungsaufbau: Der Client führt den Verbindungsaufbau zum Server durch. Üblicherweise versucht er das unter dem Port 80. Das HTTP Protokoll unterstützt aber auch alle anderen nicht reservierten Ports. Dieser kann in der URL näher spezifiziert werden [FUGC+99].
2. Anfrage (`Request`): Der Client sendet die Request - Nachricht an den Server. Zum Beispiel könnte der Client die Anfrage `http://www.example.org/murakami.html` an

den Server senden. Die Übertragung erfolgt in diesem Fall auf den Port 80 des Servers. Es wird folgende HTTP-GET-Anforderung (vgl. Kapitel 2.1.2) an den Server gerichtet [FUGC+99].

```
…
GET /murakami.html HTTP/1.1
Host: www.example.org
…
```

3. Antwort (Response): Der Server sendet die Response – Nachricht an den Client. Diese könnte zum Beispiel folgendermaßen aussehen [FUGC+99]:

```
…
HTTP/1.x 200 OK
Date: Wed, 04 Dec 2013 14:20:11 GMT
Server: Apache/2.0.40 (Red Hat Linux)
…
Content-Length: 4539
Content-Language: de
Connection: close
Content-Type: text/html
…
<html>
<head>
…
```

4. Verbindungsabbruch: Kann durch beide Teilnehmer durchgeführt werden. Sollte der Server wider Erwarten die Verbindung zu früh beenden, versucht der Client die Verbindung neu aufzubauen [FUGC+99].

2.1.1. Uniform Ressource Identifier

Im Standard RFC 3986 der Internet Engineering Task Force wurde der Uniform Ressource Identifier, kurz URI, als eine bestimmte kompakte Zeichenfolge, welche eine abstrakte oder physische Ressource identifiziert, definiert [BFDW+05]. URIs werden im World Wide Web hauptsächlich zur Bezeichnung bestimmter Ressourcen eingesetzt. Die generische URI Syntax lässt die Bezeichnung unterschiedlichster Adressen zu und setzt sich aus den Teilen `scheme`, `authority`, `path`, `query` und `fragment` zusammen [BFDW+05].

```
URI = scheme "://" authority ["/" path] ["?" query] ["#" fragment]
```

Am Beispiel `foo://example.com:8042/over/there?name=ferret#nose` kann man folgende Teile ablesen [BFDW+05]:

scheme	authority	path	query	fragment
foo	example.com:8042	over/there	name=ferret	nose

Tabelle 4 Bestandteile einer URI

Das `scheme` repräsentiert den Typ von URI, welcher den Aufbau der restlichen URI festlegt. Übliche Schemata im WWW sind `http`, `ftp`, `mailto`, `file` *und* geo. Viele URI Schemata (wie http oder ftp) haben eine zentrale Instanz, welche die Namensverteilung autorisiert und verwaltet. Die generische Syntax lässt es zu, individuelle (Namens-)Verteilungen korrekt anzusprechen, sei es mittels des Namens und eines bestimmten Ports, oder zusätzlich mit Anmeldeinformationen zu dieser `authority`. Die Bezeichnung der `authority` beginnt mit einem ‚//' und endet mit einem ‚/'. Ein mögliches Beispiel für eine *authority* mit Anmeldung wäre [BFDW+05]:

```
//nobody:password@example.org:8080/
```

Der `path` enthält (üblicherweise hierarchisch organisierte) Daten um eine bestimmte Ressource (welche durch `scheme` und `authority` bereits näher eingegrenzt wurden) zu identifizieren. Der `path` endet entweder mit einem ‚?' oder einem ‚#' oder durch das Ende der URI [BFDW+05]. Die `query` Komponente beinhaltet (nicht hierarchisch organisierte) Daten zur weiteren Identifizierung der Ressource. Sie beginnt mit einem ‚?' und endet mit einem ‚#' oder mit dem Ende der URI. Ein Beispiel für eine gültige `query` wäre:

```
?uid=20140326-08&newapp=multievent
```

Die Komponente `fragment` erlaubt eine indirekte Identifikation einer sekundären Ressource (vgl. Kapitel 2.2.1.2), indem es auf eine primäre Ressource verweist. Die identifizierte sekundäre Ressource kann ein Teil oder ein Auszug einer primären Ressource sein [BFDW+05].

2.1.2. HTTP-Request-Methoden

Wie oben (vgl. Kapitel 2.1) bereits erwähnt, ist ein möglicher Nachrichteninhalt einer HTTP Transaktion eine Anfrage des Clients an den Server (Request). Dieser Request beinhaltet die Methode die der Server durchführen soll. Es gibt mehrere dieser Request – Methoden, welche aber nicht alle auch zwingend vom Server unterstützt werden müssen. Auf jeden Fall unterstützt werden die Methoden GET und HEAD, da sie als „sicher" gelten und nur zur Datenabfrage und

nicht zur Datenänderung gedacht sind. Folgende Request – Methoden wurden im Standard RFC 2616 des World Wide Web Consortiums spezifiziert [FUGC+99]:

1. GET: Mit der Methode GET fordert der Client eine bestimmte, durch die mitgelieferte URI identifizierte, Ressource an.
2. POST: POST wurde so konzipiert, dass man damit bei bereits vorhandenen Dateien am Server Daten anfügt, oder eine neue Datei zu einem bestimmten Verzeichnis hinzufügen kann.
3. PUT: PUT dient dazu, eine bestimmte Ressource auf den Server an den der Request gesendet wird, hochzuladen.
4. HEAD: Die Methode HEAD fordert wie GET eine bestimmte Ressource vom Server an, erhält als Antwort lediglich den Message-Header (ohne den Message-Body, vgl. Kapitel 2.1).
5. DELETE: Die Methode DELETE erlaubt es dem Client, eine bestimmte Ressource am Server zu löschen. Die Ressource ist danach nicht mehr erreichbar.
6. OPTIONS: Eine Anfrage des Clients mit OPTIONS liefert eine Liste mit den vom Server unterstützten Request-Methoden.
7. TRACE: Die Antwort auf einen Request mit TRACE ist ident mit der Anfrage. So kann der Client überprüfen, ob die Anfrage am Weg zum Server abgeändert wurde.
8. CONNECT: Diese Methode ist für den Einsatz eines Proxyservers gedacht, der in der Lage ist, als SSL Tunnel zu fungieren.

Für die Verwendung der Request-Methoden PUT, DELETE und POST muss der Webserver so konfiguriert werden, dass er die Anfragen zulässt und sie zu verarbeiten weiß.

2.1.3. WebDAV

WebDAV ist ein offener Standard zur Bereitstellung beliebiger Dateien im World Wide Web [Duss07]. Es stellt eine Erweiterung des HTTP/1.1 Protokolls dar, welches zusätzliche Methoden zur Manipulation von Dateien durch einen Client zur Verfügung stellt. Es erweitert das HTTP um folgende Request-Methoden (vgl. Kapitel 2.1.2) [Duss07]:

1. PROPFIND: Mit der Methode PROPFIND kann man Eigenschaften einer Datei oder einer bestimmten Ressource auf einem entfernten Webserver ermitteln. Mit PROPFIND kann man ebenfalls Auskunft über die Verzeichnisstruktur eines entfernten Systems erhalten.
2. PROPPATCH: Die Methode PROPPATCH ermöglicht das ändern oder löschen bestimmter Eigenschaften einer Ressource auf einem entfernten Webserver.
3. MKCOL: Mit MKCOL kann man ein Verzeichnis auf einem entfernten Webserver erstellen.

4. COPY: Mit der Methode COPY lässt sich eine Ressource von einer URI auf eine andere URI kopieren.
5. MOVE: Mit MOVE kann man eine Ressource von einer URI auf eine andere URI verschieben.
6. LOCK: Mit der Methode Lock lässt sich eine bestimmte Ressource sperren um den Zugriff anderer Clients währen der eigenen Anfrage zu verhindern.
7. UNLOCK: UNLOCK entfernt die mit LOCK angelegte Sperre wieder von einer bestimmten Ressource.

WebDAV wird von den gängigsten Webservern und Frameworks mittels Plugin oder über Schnittstellen unterstützt.

2.2. Semantic Web

Das Semantic Web ist eine Erweiterung des vorhandenen World Wide Webs in der Hinsicht, dass vorhandene Daten, oder in Zukunft hinzukommende Daten, mit Metadaten versehen werden, um die Interpretation des Inhalts nicht nur für den Menschen, sondern auch für die Maschine zu gewährleisten [HKRS08]. Die Semantik einer bestimmten Information ist für einen Menschen oft schnell erkennbar, nicht jedoch für eine Maschine. Gibt man den Suchbegriff „Queen“ als einzige Informationsquelle einer Suchmaschine an, gibt es semantisch gesehen Unterschiede in mehreren Richtungen. Es kann damit die Musikband Queen, ein Lexikoneintrag zu Queen, ein Film namens Queen, eine Zeitschrift namens Queen, ein Schiff mit diesem Namen oder die Königin Elisabeth gemeint sein. [AnHa08] führen in diesem Zusammenhang das Beispiel einer Verneinung in einem Satz an, welche für einen Menschen, nicht jedoch für eine Maschine intuitiv erkannt wird. Zum Beispiel die Unterscheidung der Aussagen: „ich bin ein Universitätsprofessor für Computerwissenschaften“ und „ich bin ein Universitätsprofessor für Computerwissenschaften, sollte man glauben ...“, stellt für eine Maschine ein immer noch sehr ehrgeizig zu erreichendes Ziel dar [AnHa08].

Tim Berners Lee et al. haben in [BeHL01] den Wirkungsbereich des Semantic Webs über die Verwendung in direktem Bezug auf den Computer hinaus noch weiter ausgedehnt. Es wird das visionäre Beispiel verlinkter Gegenstände im Alltag angeführt. Ein läutendes Mobiltelefon veranlasst die Soundanlage im Wohnzimmer die Lautstärke zu reduzieren [BeHL01], oder die mit dem Web verbundene Mikrowelle sucht auf der Webseite des Herstellers des gefrorenen Mikrowellenessens nach der optimalen Zubereitungskonfiguration.

Um diese Hürde zu überwinden, ist die Verwendung von Vereinheitlichungen und Standards durch die involvierten Geräte essentiell. 2006 wurde erneut von einem Team rund um Tim Berners Lee die Notwendigkeit dieser Standards aufgezeigt und versucht, das Bewusstsein für RDF, RDFS, XML, URI, OWL und andere bereits etablierte Standards zu schärfen [SnBH06]. In [Bern07] und [BeHL01] wurden auch Standards für andere mögliche, an das Web angebundene Geräte, aufgezeigt um die Interoperabilität der inkludierten Objekte zu unterstützen (vgl. Abbildung 2).

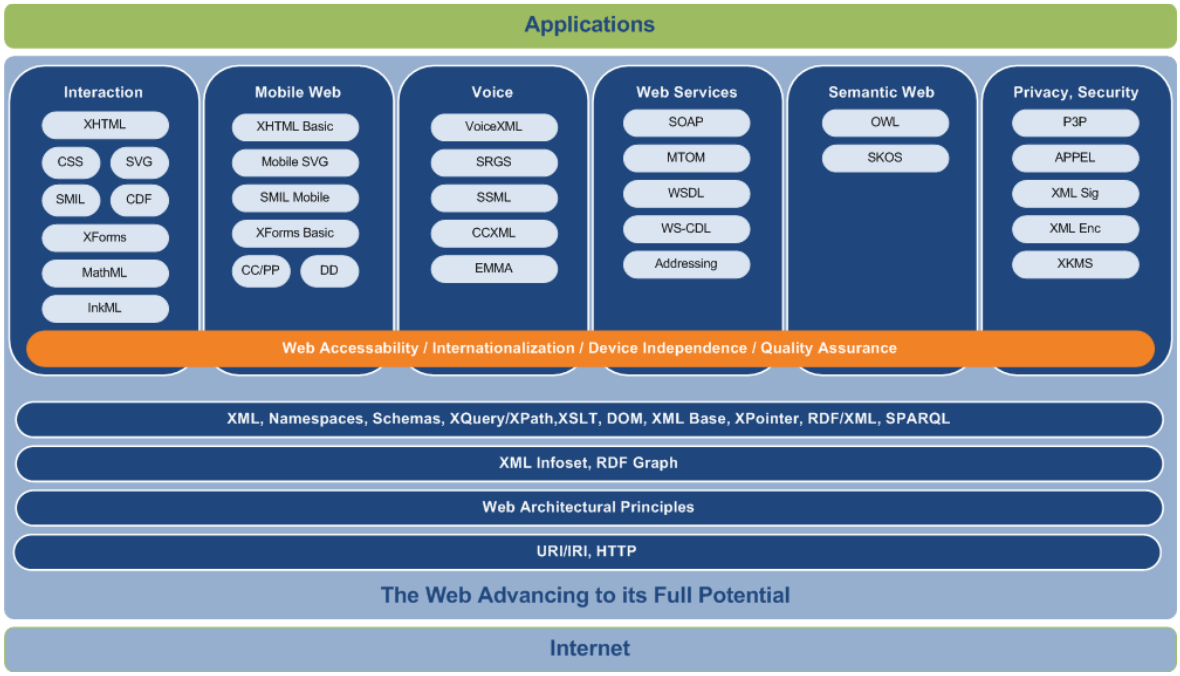


Abbildung 2 Standards zur Unterstützung der Interoperabilität im Web (nach [Bern07])

Wie bereits erwähnt, war die Grundidee des Semantic Web von Tim Berners Lee sehr visionär. Er hat bei der ersten World Wide Web Conference 1994 bereits erste Ideen dazu geäußert [SnBH06]. [HKRS08] formulieren die Ziele des Semantic Webs etwas weniger weit vorgreifend: „Finde Wege und Methoden, Informationen so zu repräsentieren, dass Maschinen damit in einer Art und Weise umgehen können, die aus menschlicher Sicht nützlich und sinnvoll erscheint.“.

Wie die oben genannten Standards miteinander in Beziehung stehen und zusammenarbeiten sollen, hat Tim Berners Lee erstmals 2000 mit dem Semantic Web Stack visualisiert [Bern00]. Die damit entstandene Architektur des Semantic Webs soll die Hierarchie der verwendeten Sprachen verdeutlichen und vermitteln, dass die jeweiligen Schichten aufeinander aufbauen bzw. miteinander in Verbindung stehen [AnHa08]. Der Semantic Web Stack wurde in den letzten 10 Jahren immer weiterentwickelt und um Komponenten erweitert (vgl. Abbildung 3). Aufbauend auf eindeutige URI Adressen werden XML Dokumente erstellt. XML ist besonders gut zum Senden von Dokumenten im Web geeignet und erlaubt es, strukturierte Web Dokumente mit einem

benutzerdefinierten Vokabular zu erstellen [AnHa08]. Zu Beginn waren XML und RDF im Semantic Web Stack noch getrennt. Wie in Abbildung 3 erkennbar, wird RDF nun teilweise aufbauend neben XML angeführt. RDF ist ein Datenmodell zum Spezifizieren von einfachen Aussagen über Objekte im Web. RDF hat unter anderen eine XML basierte Syntax und wird deshalb damit in Verbindung gesetzt und im Semantic Web Stack auch teilweise aufbauend auf XML dargestellt. RDFS (RDF Schema) erlaubt es, Klassen und Eigenschaften, Subklassen und Subeigenschaften sowie Einschränkungen bezüglich Zugehörigkeit und Wertebereich zu bestimmten Webobjekten zu definieren. RDFS basiert auf RDF und ist deshalb darauf liegend dargestellt [AnHa08]. RDFS kann als primitive Ontologiesprache angesehen werden, hat jedoch beschränkte Möglichkeiten hinsichtlich Kardinalitäten und Constraints bzw. Wertebereichseinschränkungen. Um diese Probleme zu lösen, wird die Web Ontology Language (OWL) eingesetzt. Sie erweitert RDFS um die Möglichkeit zusätzliche Constraints und Aussagen über Kardinalitäten zu machen. OWL basiert größtenteils auf einer Beschreibungslogik und nützt bereits vorhandene Reasoner. Diese Beschreibungslogik ist ein Teilbereich der Prädikatenlogik, bei der ein effizientes Reasoning unterstützt wird. SPARQL ist eine Abfragesprache speziell für RDF basierte Datenmodelle. Es ermöglicht die Informationsbeschaffung für Semantic Web Anwendungen. RIF ist ein W3C Standard um es unterschiedlichen Regelsprachen zu ermöglichen, Regeln auszutauschen. Regeln werden eingesetzt, um die Überprüfung von Bedinungen, welche in RDFS und OWL zum Teil nicht abgebildet werden können, zu gewährleisten [AnHa08].

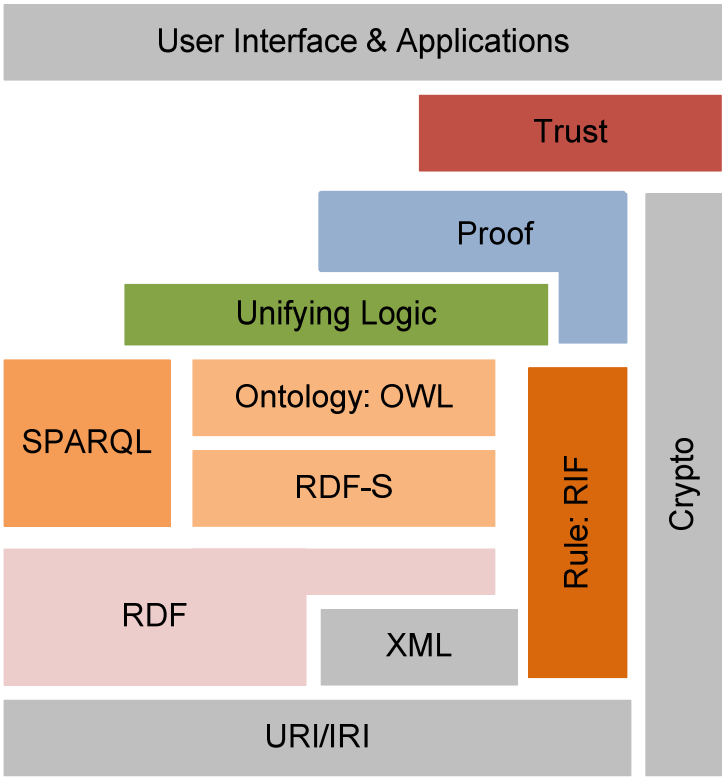


Abbildung 3 Der Semantic Web Stack (nach [Bern07])

Für die weiteren Ebenen gibt es bisher noch keine Standards. Die Unifying Logic soll Unterschiede der darunterliegenden Ebenen aufheben. Die Ebene Kryptographie beschreibt die Notwendigkeit der Überprüfung einer Quelle. Es ist wichtig, den Daten einer bestimmten Quelle vertrauen zu können. Realisiert wird dies mittels digitaler Signaturen und Information basierend auf Empfehlungen bereits verifizierter Quellen [AnHa08]. Die Ebenen Proof und Trust verwenden diese Informationen und überprüfen die Signaturen von RDF Statements um weitere Aussagen über die Vertrauenswürdigkeit der Quelle zu tätigen [HPPH05].

Eine mögliche Alternative zur Umsetzung des Semantic Webs mittels Metadaten wäre der Einsatz von künstlicher Intelligenz, um die Semantik zusammenhängender (und vielleicht sogar verteilter) Information, die ein Mensch durch Erfahrung und Wissensaneignung erkennt, auch für die Maschine zu gewährleisten [HKRS08]. Dieser ist es jedoch (noch) nicht möglich, die erforderlichen Daten in der geeigneten Qualität aufzubereiten [AnHa08]. Mittels statistischer Analysen, natürlicher Sprachverarbeitung und maschinellem Lernen hat die Anwendung IBM Watson das semantische Arbeiten von Maschinen jedoch bereits in eine neue Ära versetzt. Ihr ist es möglich, durch das Analysieren einer riesigen, zuvor gesammelten, Menge an Daten, menschenähnliche Schlussfolgerungen und Formulierungen zu erstellen [Ferr12]. Anders als der im Jahr 2012 vorgestellte Google Knowledge Graph, benötigt IBM Watson keine konstante Verbindung zum Internet sondern verwendet die vorhandene Datenmenge. Der Knowledge Graph hingegen nutzt

mehrere unterschiedliche Quellen (Ontologien, semantische Faktenlisten und Linked Data) um nach Beziehungen und Schlüsselwörtern zu suchen und Verbindungen zu erkennen und herzustellen. In 2014 wurde der Knowledge Vault Ansatz vorgestellt, welcher den Knowledge Graph um maschinelles Lernen erweitert. Er verbindet nun die Abfrage von unterschiedlichen Quellen mit dem Lernen, welches aus IBM Watson bereits bekannt ist, um menschenähnliche Schlussfolgerungen zu erzeugen [DGHH+14].

2.2.1. Linked Data

Linked Data beschreibt mehrere Ansätze, strukturierte Daten im Web untereinander so zu verbinden und zu veröffentlichen, wie es jetzt bereits mit Dokumenten möglich ist. [HeBi11] [Bern06] Tim Berners Lee hat dazu vier Prinzipien vorgestellt, welche eingehalten werden sollten, wenn man Linked Data verwendet.

Das erste Prinzip besagt, dass Objekte (sowohl abstrakte Konzepte als auch Objekte und Gegenstände aus der realen Welt, welche man beschreiben möchte) mit URIs versehen werden um diese zu identifizieren.

Das zweite Prinzip erläutert die Notwendigkeit zur Verwendung von dereferenzierbaren HTTP URIs, damit es möglich ist, diese URIs aufzusuchen und eine Beschreibung des Objekts oder des Konzepts zu erhalten.

Um dem dritten Prinzip gerecht zu werden, muss diese Beschreibung von Objekten oder Konzepten in einem standardisierten Verfahren passieren. Tim Berners Lee nennt hier zur Beschreibung der Daten RDF und zur Abfrage dieser Daten SPARQL. Nur Standards garantieren eine einheitliche Basis für unterschiedlichste Konzepte in den unterschiedlichsten Branchen.

Das vierte Prinzip besagt, dass Objekte und Konzepte miteinander verbunden werden sollen, um das Entdecken von Zusammenhängen zu ermöglichen. [HeBi11] grenzen dieses Prinzip noch etwas ein, in dem sie Hyperlinks aus Web Dokumenten für das Semantic Web als RDF Links bezeichnen um sie davon zu unterscheiden. [HeBi11] erklären weiter, dass genau diese (externen) RDF Links fundamental für das Web of Data (vgl. Kapitel 2.2.1.1) sind und erst das Verfolgen von Daten durch Applikationen und die damit verbundene Interkonnektivität von Objekten ermöglichen.

2.2.1.1. Web of Data

Das Web of Data wird laut Tim Berners Lee wie auch das normale Web durch Dokumente repräsentiert, jedoch anstatt diese Dokumente mit Hypertext zu verbinden (in HTML), werden diese Dokumente durch Daten über Objekte (in RDF) verbunden [Bern98]. [Heat09] erklärt das

Web of Data auch als Resultat des Verbindens von Daten durch Linked Data. Er vergleicht das Verbinden von HTML Dokumenten mit dem Verbinden von Daten durch RDF Triple, grenzt es jedoch noch genauer auf die von Tim Berners Lee im Jahr 2006 erwähnten Technologien URI, HTTP und RDF ein. [HeBi11] stellen sich unter dem Begriff Web of Data einen riesigen Graphen vor, in dem sämtliche Informationen über unterschiedlichste Themenbereiche, geografische Orte, Menschen, Firmen, Filme, Musik, etc. in Form von RDF Statements (vgl. Kapitel 2.2.2) hinterlegt sind. Diese Informationen können Eigenschaften oder Beziehungen über bzw. zwischen den Dingen sein. Diese Dinge werden im Web of Data als ‚*resources*' bzw. Ressourcen bezeichnet [BiCH07] [JaWa04]. Im Semantic Web werden alle Informationen als RDF Statements, welche Informationen über Ressourcen enthalten, ausgedrückt [SaCy08]. Diese Ressourcen werden durch URIs (vgl. Kapitel 2.1.1), ebenso wie Dokumente im WWW, identifiziert. Dies erlaubt es nun Informationen über ein Dokument als RDF Statement darzustellen, hat aber auch den Nachteil, dass Informationen über eine Ressource und Informationen über ein Dokument vermischt werden können [SaCy08].

Um diese Informationen zu trennen, unterscheiden [BiCH07], [JaWa04] und [SaCy08] zwischen ‚*information resources*' (Informationsressourcen), welche Dokumente beschreiben, und ‚*non-information resources*' oder ‚*other resources*' (Nicht-Informationsressourcen), welche reale Objekte oder Dinge beschreiben. Da die Adressierung beider Ressourcen über URIs stattfindet, ergab sich das Problem der Trennung der semantischen Information über die jeweiligen Ressourcen. Zum Beispiel wird die Person `John Grady` auf seiner Homepage näher beschrieben. `Nora Hayden`, `John Grady's` Freundin, mag vielleicht nicht die Homepage von `John Grady`, mag aber die Person `John Grady`. Somit werden eigentlich zwei URIs benötigt. Eine für die Homepage oder ein RDF Dokument, welches `John Grady` beschreibt, und eine URI für `John Grady` selbst. [SaCy08] schlagen für dieses Problem zwei mögliche Lösungen vor, Hash URIs und 303 URIs.

2.2.1.2. Hash URI

Bei der Verwendung von Hash URIs wird an die URI ein `fragment` (vgl. Kapitel 2.1.1) angehängt um eine Nicht-Informationsressource zu identifizieren, beispielsweise `http://www.example.com/about.rdf#JohnGrady`. Beim Aufruf dieser URI schneidet der Server bei der Anfrage das `fragment` ab und sendet die Information, die sich auf dieser URI befindet, als Repräsentation zurück. Nachteile dieser Variante sind, dass man mit einer Hash URI keine Informationsressourcen identifizieren kann und bei einem Aufruf der Hash URI der gesamte Inhalt der URI repräsentiert wird, nicht bloß die gewünschte Information über die Nicht-Informationsressource [SaCy08].

2.2.1.3. 303 URI

Bei der Verwendung von 303 URIs bedient man sich des HTTP Status Codes 303 See Other. Beim Aufruf einer URI wird vom Server nicht direkt mit der Information geantwortet, sondern er antwortet mit einer URI, auf der die Informationen zu finden sind [SaCy08]. Beispielsweise bei der Anfrage an den Server mit der URI `http://www.example.com/noninformationresource/JohnGrady` antwortet der Server nicht mit der Repräsentation der auf der URI befindlichen Information, sondern erneut mit einer URI, z.B. `http://www.example.com/informationresource/JohnGrady`. Eine erneute Anfrage an den Server mit der als Antwort erhaltenen URI liefert erst die Repräsentation. Diese Variante hat zum Vorteil, dass nun getrennte URIs für Informationsressourcen und Nicht-Informationsressourcen unterstützt werden. Der Nachteil davon ist, dass mehrere Anfragen an den Server erforderlich sind [SaCy08].

2.2.2. RDF

Das Resource Description Framework, kurz RDF, ist ein Datenmodell zur Repräsentation von Informationen über Ressourcen (vgl. Kapitel 2.2.1.1). [ScRa14] [BiCH07] [AnHa08] RDF war ursprünglich eher für die Verarbeitung von Information durch Anwendungen gedacht als zur Präsentation für den Menschen [ScRa14]. Es bietet deshalb ein Grundgerüst zur Beschreibung von Information, welche den Austausch zwischen Maschinen ohne den Verlust der Bedeutung dieser Information, ermöglicht [ScRa14] [HKRS08]. Diese Beschreibung von Information wird in RDF durch mehrere Triple repräsentiert. Ein Triple besteht aus drei Teilen: einem Subjekt, einem Prädikat und einem Objekt. Ein Triple spiegelt die Struktur eines normalen Satzes wieder [BiCH07]:

John	hat die **E-Mail Adresse**	**john@grady.com**.
(**Subjekt**)	(**Prädikat**)	(**Objekt**)

Jedes Triple repräsentiert eine Aussage (Statement) über die Beziehung zwischen Subjekt und Objekt [KlCa04]. Das Subjekt eines Triples ist eine URI, welche die Ressource identifizieren soll. Das Objekt kann entweder ein einfaches Wort, ein Datum oder eine Zahl (als Bezeichnung) oder eine URI einer anderen Ressource sein. Das Prädikat weist darauf hin, welche Art von Beziehung zwischen dem Subjekt und dem Objekt besteht [HeBi11]; z.B. dass eine Person eine andere Person kennt, oder dass eine Person an einem bestimmten Ort geboren wurde. Das Prädikat wird auch über eine URI identifiziert, welche jedoch aus einem Vokabular stammt. Vokabulare sind eine Sammlung von URIs, welche zur Beschreibung von Beziehungen zwischen Subjekten und Objekten verwendet werden können [ScRa14]. Es existieren bereits Vokabulare zur Beschreibung

von Personen (FOAF), Lizenzbedingungen (CC), Projekte (DOAP), etc. welche, sofern möglich, verwendet werden sollen, anstatt ein neues Vokabular zu kreieren.

[ScRa14] und [HeBi11] erwähnen im Zusammenhang mit RDF Triple auch den Begriff eines RDF Graphen, der eine Menge an RDF Triple beschreibt. Dabei werden Subjekt und Objekt als Knoten (‚*nodes*') dargestellt und mittels eines gerichteten Pfeils verbunden, wobei der Pfeil die Beziehung zwischen Subjekt und Objekt darstellt [HeBi11]. Abbildung 4 zeigt ein Beispiel eines RDF Graphen, welches Ausdrücken soll, dass Matthew John kennt.

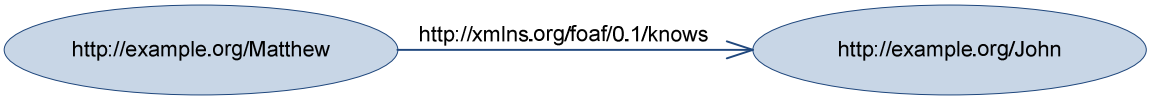


Abbildung 4 Ein einfacher RDF Graph

Bei diesem Beispiel ist das Objekt eine URI auf der man Informationen finden kann und kein Literal (wie z.B. der Name von John oder seine E-Mail Adresse). Diese Art von RDF Triple, bei der drei URI Referenzen verwendet werden (für Subjekt, Prädikat und Objekt), nennt man RDF Link [BiCH07]. RDF Links sind die Basis für das in Kapitel 2.2.1.1 vorgestellte Web of Data und halten wie ein „Kleber" das Web of Data zusammen [HeBi11].

2.2.2.1. Leere Knoten

Wie oben erwähnt, kann ein Objekt eines RDF Graphen sowohl Literale, also z.B. ein einfaches Wort, ein Datum oder eine Zahl, oder aber eine URI, welche eine Ressource identifiziert, sein und wird durch einen Knoten dargestellt. Es gibt jedoch noch eine weitere Modellierungsmöglichkeit, einen sogenannten leeren Knoten (engl. ‚*blank node*'). Ein leerer Knoten ist eine Art Hilfsknoten der eine Hilfsressource beschreibt, jedoch nicht global durch eine URI adressiert wird [HKRS08]. [ScRa14] vergleichen leere Knoten mit Variablen in der Algebra, welche etwas repräsentieren, ohne deren Wert festzulegen. Abbildung 5 zeigt die Verwendung eines leeren Knotens, wobei der leere Knoten die Informationen einer bestimmten Adresse von Matthew darstellen soll. [HeBi11] zufolge ist die Verwendung von leeren Knoten zwar syntaktisch richtiges RDF, aber es ist kein sauberer Weg, Linked Data darzustellen. Der Grund dafür ist, dass leere Knoten das Zusammenführen von Informationen aus unterschiedlichen Quellen erschweren. Als Lösung schlagen [HeBi11] vor, sofern möglich, diese Knoten durch URIs eindeutig identifizierbar zu machen.

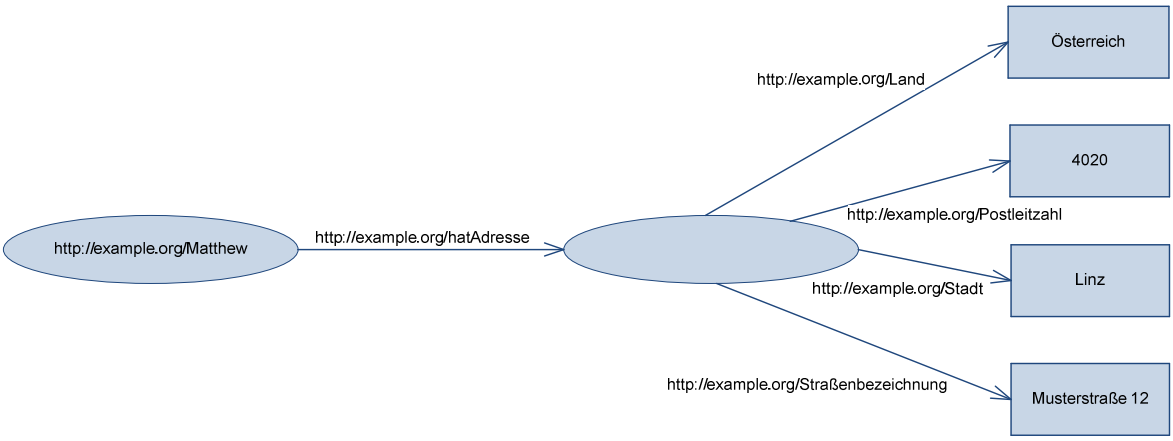


Abbildung 5 Darstellung eines leeren Knotens in RDF

2.2.2.2. Serialisierungen und Namensräume

Es gibt mehrere Möglichkeiten RDF Triple in textueller Form darzustellen. [HeBi11] unterscheiden zwischen den vom W3C standardisierten Serialisierungsformaten RDF/XML, RDFa, Turtle, N-Triples und JSON for LD.

Das am weitesten verbreitetste Format ist RDF/XML, welches jedoch für den Menschen im Vergleich zu den anderen Formaten schwer lesbar ist [HeBi11]. Ein Beispiel für ein in RDF/XML dargestelltes RDF Triple wäre:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:vcard="http://www.w3.org/2006/vcard/ns#">

<rdf:Description rdf:about="http://example.org/Matthew">
      <vcard:fn>Matthew Sobol</vcard:fn>
</rdf:Description>
</rdf:RDF>
```

Dieses RDF Triple besagt, dass die Ressource `http://example.org/Matthew` den Namen ‚Matthew Sobol' hat. Repräsentiert wird die Information ‚Matthew Sobol' in diesem Beispiel durch die Bezeichnung `<vcard:fn>` (wobei `fn` für ‚*formatted name*' steht), welches eine verkürzte Schreibweise für `http://www.w3.org/2006/vcard/ns#fn ist.` Diese verkürzte Schreibweise wird durch sogenannte Namensräume (engl. ‚*namespaces*') ermöglicht [HKRS08]. In diesem Beispiel wird ein Namensraum `vcard` definiert, welcher auf das vCard Vokabular verweist und die Verwendung dessen Elemente in kürzerer Schreibweise erlaubt.

RDFa ist ein Serialisierungsformat, welches RDF Triple in HTML Dokumenten einbetten lässt. Dabei wird der bestehende HTML Inhalt einer Webseite mittels RDFa markiert und der dort befindliche HTML Code modifiziert [HeBi11]. Turtle ist ein Format, welches das Beschreiben eines RDF Graphen in kompakter textueller Form erlaubt [BeBe11] und gleichzeitig eine Erweiterung von N-Triples. N-Triples ist ein Serialisierungsformat, welches nur vollständige URIs in jedem

einzelnen RDF-Triple erlaubt. Es besitzt daher viele redundante Angaben, hat aber den Vorteil, dass jede einzelne Zeile die notwendigen Informationen zu diesem RDF Triple besitzt [HeBi11], und so z.B. jede einzelne Zeile von Programmen eigenständig seriell eingelesen werden kann. Gängige RDF Frameworks wie Apache Jena, SemWeb-DotNet oder EasyRDF unterstützen zumeist mindestens eine, teilweise aber sogar alle genannten Serialisierungsformate.

2.2.2.3. Vokabulare

Um eine einheitliche Abbildung von Informationen über Nichtinformationsressourcen aus der realen Welt zu gewährleisten, wird, wie oben bereits erwähnt, von [ScRa14] und [HeBi11] empfohlen, vorhandene Vokabulare zu verwenden. Neben Vokabularen zur Beschreibung von Freundesbeziehungen wie z.B. FOAF[4] (*‚friend of a friend'*) gibt es sehr spezielle Vokabulare z.B. um Webseiten für Suchmaschinen semantisch interpretierbar zu machen (schema.org[5]) [ScRa14].

Für die Beschreibung von Menschen und Organisationen bzw. die Beschreibung von Kalenderdaten wurden, angelehnt an die Standards iCalendar[6] und vCard[7], RDF - Vokabulare entwickelt. Diese beiden Vokabulare sollen im folgenden Exkurs näher erläutert werden.

Exkurs: Kalendervokabular RDF Calendar

In RDF Calendar wurden die Kalender- und Terminelemente und dessen Eigenschaften aus iCalendar konvertiert und für RDF aufbereitet [CoMi05]. Das folgende Terminbeispiel in iCalendar

```
BEGIN:EVENT
UID:20140515T094445Z-3895-69-1-7@jammer
DTSTART;VALUE=DATE:20140515
DTEND;VALUE=DATE:20140518
SUMMARY:RDF Conference
LOCATION:Linz
END:VEVENT
```

wird in RDF Calendar mit der RDF/XML Serialisierung folgendermaßen dargestellt [CoMi05]:

```
01 <rdf:RDF xmlns="http://www.w3.org/2002/12/cal#">
02 ...
03 <Vevent rdf:about="event_20140515T094445">
04    <uid>20140515T094445Z-3895-69-1-7@jammer</uid>
05    <dtstart rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
06      2014-05-15</dtstart>
07    <dtend rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
08      2014-05-18</dtend>
09    <summary>RDF Conference></summary>
10    <location>Linz</location>
```

[4] http://www.foaf-project.org/.
[5] http://schema.org/.
[6] http://www.ietf.org/rfc/rfc2445.
[7] http://tools.ietf.org/html/rfc6350.

```
11 </Vevent>
12 ...
13 </rdf:RDF>
```

In diesem Beispiel wird ein Termin dargestellt, der einen eindeutigen Bezeichner, einen Startzeitpunkt (`2014-05-15`), einen Endzeitpunkt (`2015-05-18`) und eine Bezeichnung (RDF `Conference`) hat und an einem bestimmten Ort (`Linz`) stattfindet.

In RDF Calendar werden sowohl Termine als auch Benachrichtigungen (z.B. ein Alarm) oder andere Elemente (z.B. Zeitzonen) als Kalenderkomponenten betrachtet, welche sich in einem Kalenderobjekt befinden [CoMi05]. Es existieren neben der Komponente Vevent noch die Komponenten Vtodo, Vjournal, Vtimezone, Valarm und Vfreebusy [CoMi05]. Vevent stellt dabei einen Termin mit mehreren Teilnehmern dar. Ein Vtodo ist eine Aufgabe die eine Person für sich selbst in ihren Kalender einträgt (z.B. an einer Seminararbeit schreiben oder den Müll raustragen). Vjournal ist eine Art Notiz oder ein Tagebucheintrag, welcher in textueller Form dargestellt wird und nur für den Kalenderinhaber gedacht ist. Vtimezone ermöglicht es, Termine um unterschiedliche Zeitzonen zu erweitern. Valarm stellt eine Benachrichtigung bzw. einen Alarm dar und Vfreebusy ist eine Terminkategorie, welche einen Zeitraum der Nichtverfügbarkeit wie einen Urlaub oder Ferien darstellt [CoMi05].

RDF Calendar stellt eine Reihe von Eigenschaften (‚*properties*') zur Verfügung, welche von den Kalenderkomponenten verwendet werden können. In Tabelle 5 werden (nur) die für diese Arbeit relevanten Eigenschaften aus RDF Calendar aufgeführt und kurz beschrieben [CoMi05].

Property	Bedeutung
dtstamp	Erstellungszeitpunkt der jeweiligen Komponente
dtstart	Beginnzeitpunkt der jeweiligen Komponente
dtend	Endzeitpunkt einer Vevent oder Vfreebusy Komponente
due	Endzeitpunkt einer Vtodo Komponente
duration	Dauer der jeweiligen Komponente
status	Status der Bestätigung einer Komponente
attendee	Teilnehmer einer Komponente
attach	Ein angehängtes Dokument an die jeweilige Komponente
categories	Kategoriebezeichnung der Komponente
comment	Kommentare oder Hinweise zur Komponente
summary	Kurze Beschreibung der jeweiligen Komponente
geo	Geografische Informationen zur Abhaltung der jeweiligen Komponente
organizer	Ersteller der jeweiligen Kompontente
related-to	Ermöglicht das Vernetzen von Komponenten
uid	Eindeutige Bezeichnung der jeweiligen Komponente
trigger	Gibt an, wann ein Alarm ausgelöst werden soll
repeat	Gibt an, ob ein Alarm wiederholt ausgelöst werden soll
last-modified	Zeitpunkt wann die jeweilige Komponente das letzte Mal geändert wurde

description	Eine detailliertere Beschreibung der jeweiligen Komponente

Tabelle 5 Properties aus RDF Calendar und deren Bedeutung

Alle Zeitangaben werden in RDF Calendar nicht als Textzeichenfolge dargestellt sondern als DATE-TIME Objekt, welche eine aus iCalendar spezifizierte Zeitangabe ist [CoMi05]. Die Zeitangabe wird folgendermaßen dargestellt:

```
DTSTART:20140518T110200
```

Dabei gelten die ersten 4 Zeichen als Jahresangabe, `2014`, und die anschließenden Zeichen als Monat und Tag, `18. Mai`. Durch ein ‚T' gekennzeichnetes Trennzeichen wird die Zeitangabe in Stunden, Minuten und Sekunden angegeben, `11:02:00 Uhr`.

Das konkrete RDF Vokabular, welches im Zuge dieser Arbeit verwendet wurde, ist unter http://www.w3.org/2002/12/cal/icaltzd abrufbar.

Exkurs: Vokabular zur Beschreibung von Menschen und Organisationen, RDF vCard

In RDF vCard wurden die Elemente zur Beschreibung von Menschen und Organisationen aus vCard konvertiert und für RDF aufbereitet [Iann10]. Es gibt mehrere Elemente in RDF vCard, welche zur Beschreibung verwendet werden. Diese sind VCard, Name, Address, Organisation, Location, Tel, Label und Email. Ein RDF vCard Objekt, dargestellt in RDF/XML, könnte z.B. folgendermaßen aussehen:

```
<rdf:RDF xmlns:v="http://www.w3.org/2006/vcard/ns#">
...
<v:VCard rdf:about="http://example.com/me/John">
      <v:fn>John Grady</v:fn>
      <v:email>john@grady.com</v:email>
      <v:tel>
            <rdf:Description>
            <rdf:value>+43 4711 0815</rdf:value>
            <rdf:type
rdf:resource="http://www.w3.org/2006/vcard/ns#Work"/>
            <rdf:type
rdf:resource="http://www.w3.org/2006/vcard/ns#Fax"/>
            </rdf:Description>
      </v:tel>
</v:VCard>
...
</rdf:RDF>
```

In diesem Beispiel wird die vCard einer Person namens `John Grady` mit der E-Mail Adresse `john@grady.com` und der Faxnummer `+43 4711 0815` dargestellt.

In RDF vCard werden mehrere Eigenschaften zur Beschreibung einer vCard zur Verfügung gestellt. In Tabelle 6 werden die für diese Arbeit relevanten Eigenschaften aufgeführt und beschrieben [Iann10].

Property	Bedeutung
adr	Adressobjekt einer vCard
email	E-Mail Adresse einer vCard
tel	Telefonnummernobjekt einer vCard
country-name	Ländername eines Adressobjekts
street-address	Straßenbezeichnung und Hausnummer eines Adressobjekts
locality	Stadtbezeichnung eines Adressobjekts
fax	Faxnummer eines Telefonnummernobjekts
title	Titel einer Person oder Organisation, welche durch eine vCard beschrieben wird
nickname	Kurzbezeichnung einer Person, welche durch eine vCard beschrieben wird
fn	Name einer Person oder Organisation, welche durch eine vCard beschrieben wird

Tabelle 6 Properties aus RDF vCard und deren Bedeutung

Das konkrete RDF Vokabular, welches im Zuge dieser Arbeit verwendet wurde, ist unter http://www.w3.org/wiki/images/2/2b/Vcard.rdf abrufbar.

2.2.3. RDFS

RDF Schema (kurz RDFS) ist eine Wissensrepräsentations- oder Ontologiesprache, welche es ermöglicht, semantische Abhängigkeiten zu beschreiben, und ist Bestandteil der RDF Recommendation des W3C [HKRS08]. Diese semantischen Abhängigkeiten geben Literalen oder Ressourcen, welche in RDF deklariert werden, eine semantische Bedeutung und erlauben es einer Maschine dadurch Beziehungen und Zusammenhänge zu erkennen. [HKRS08] [BrGu14]

RDFS ist selbst ein spezielles RDF Vokabular und sein Namensraum (üblicherweise mit `rdfs:` abgekürzt) lautet `http://www.w3.org/2000/01/rdf-schema#`. Es besitzt keine thematischen Aussagen wie die in Kapitel 2.2.2.3 erwähnten Vokabulare FOAF oder schema.org, sondern es stellt universelle Ausdrucksmittel bereit. Diese Ausdrucksmittel ermöglichen es, innerhalb eines RDFS Dokuments Aussagen über semantische Beziehungen über vom Benutzer selbst festgelegte thematische Ressourcen zu machen [HKRS08].

Es gibt jedoch Grenzen in der Modellierungsfähigkeit von RDFS. Eine gravierende Einschränkung ist, dass negative Aussagen wie z.B. dass eine bestimmte Person keine Bücher liest (oder nicht lesen kann), in RDFS nicht ausgedrückt werden kann [HKRS08]. Man könnte ein Prädikat ex:Analphabet einführen, jedoch ist das nicht die Intention einer beschreibenden Semantik [HKRS08], da dadurch ein Problem bei folgendem Beispiel entsteht:

```
ex:John          rdf:type          ex:Analphabet .
ex:John          rdf:type          ex:Leseratte .
```

Diese beiden Aussagen führen in RDFS nicht zu einem Widerspruch (sollten sie in einer vernünftigen Logik jedoch tun) und es gibt in RDFS auch keine Möglichkeit zu spezifizieren, dass `ex:Analphabet` und `ex:Leseratte` keine gemeinsamen Elemente enthalten dürfen [HKRS08]. [HKRS08] nach gibt es ausdrucksstärkere Sprachen wie etwa OWL, auf die in solchen Fällen zurückgegriffen werden kann.

2.2.3.1. Klassen und Instanzen

Ressourcen (vgl. Kapitel 2.2.1.1) können in Gruppen aufgeteilt werden. Diese Gruppen werden als Klassen bezeichnet [BrGu14]. Jedes Mitglied einer Klasse wird als Instanz dieser Klasse bezeichnet. Klassen selbst sind Ressourcen und werden mittels einer URI eindeutig identifiziert und können mehrere Eigenschaften besitzen. Um eine Ressource als Instanz einer Klasse zu definieren, wird die URI `rdf:type` verwendet [BrGu14] [HKRS08]. Im folgenden Beispiel wird mittels `rdf:type` dargestellt, dass die Ressource `ex:John` eine Instanz der Klasse `ex:Person` ist.

```
ex:John          rdf:type          ex:Person .
```

Um festzulegen, welche Ressourcen Klassen sein sollen und welche nicht, wird in RDFS die vordefinierte URI `rdfs:Class` bereitgestellt [HKRS08]. Mit Hilfe dieser URI, kann der Benutzer selbst definieren, welche Ressourcen als Klassen gelten sollen.

2.2.3.2. Klassenhierarchien

In RDFS gibt es die Möglichkeit Klassenhierarchien zu bilden, welche es erlauben, transitive Aussagen über Klassenbeziehungen zu tätigen. Sichergestellt wird dies mittels der URI `rdfs:subClassOf` [HKRS08]. Im folgenden Beispiel wird durch die `rdfs:subClassOf`–Beziehung ermöglicht, dass nicht nur die Aussage, dass `ex:John` zur Klasse `ex:Person` gehört, sondern auch, dass `ex:John` zur Klasse `ex:Lebewesen` gehört.

```
ex:Person        rdfs:subClassOf        ex:Lebewesen .
ex:John          rdf:type               ex:Person .
```

[HKRS08] bezeichnen hierbei `ex:Person` als Unterklasse und `ex:Lebewesen` als Oberklasse und die Beziehung zwischen beiden Klassen als Unterklassenbeziehung. Mittels Unterklassenbeziehungen ist es möglich, nicht bloß vereinzelt Zusammenhänge anzugeben, sondern ganze Klassenhierarchien zu modellieren [HKRS08].

2.2.3.3. Eigenschaftshierarchien

Wie die eben erklärten Unterklassenbeziehungen Beziehungen zwischen Klassen ermöglichen, gibt es in RDFS auch die Möglichkeit Beziehungen zwischen Eigenschaften zu spezifizieren

[HKRS08]. Das folgende Beispiel beschreibt, dass zwischen ex:John und ex:Lisa die Beziehung ex:istVerheiratetMit besteht.

```
ex:istVerheiratetMit    rdf:type                    rdf:Property .
ex:John                 ex:istVerheiratetMit        ex:Lisa .
```

Subproperties erlauben nun eine Eigenschaft, welche mit ex:istVerheiratetMit eine Beziehung eingeht. In folgendem Beispiel kann durch die Subpropertybeziehung abgeleitet werden, dass nicht nur die Aussage, dass ex:John glücklich mit ex:Lisa verheiratet ist, sondern es kann auch schlussgefolgert werden, dass ex:John mit ex:Lisa verheiratet ist [HKRS08].

```
ex:istGlücklichVerheiratetMit  rdf:subPropertyOf  ex:istVerheiratetMit .
ex:John          ex:istGlücklichVerheiratetMit      ex:Lisa .
```

Für einen Menschen ist auf den ersten Blick klar, dass ex:John und ex:Lisa zur Klasse ex:Person gehören, da nur Personen miteinander verheiratet sein können. Für eine Maschine ist dies nicht automatisch erkennbar. Um den Definitionsbereich und den Wertebereich einer Aussage festlegen zu können, stellt RDFS die URIs rdfs:domain und rdfs:range bereit [HKRS08]. Mit rdfs:domain lässt sich der Typ eines Subjekts festlegen und mit rdfs:range lässt sich ein Objekt typisieren, welches mit einem bestimmten Prädikat zu diesem Subjekt vorkommt. Im Falle des Prädikats ex:istVerheiratetMit wären folgende Aussagen notwendig um sicherzustellen, dass nur zwei Instanzen der Klasse ex:Person miteinander die Beziehung ex:istVerheiratetMit eingehen können [HKRS08].

```
ex:istVerheiratetMit        rdfs:domain      ex:Person .
ex:istVerheiratetMit        rdfs:range       ex:Person .
```

2.2.4. SPARQL

SPARQL steht für SPARQL Protocol and RDF Query Language und ist eine von der W3C empfohlene Abfragesprache für die Abfrage und Darstellung von in RDF spezifizierten Informationen [HKRS08]. Mit Hilfe einer Turtle–ähnlichen Syntax und Anfragevariablen können Anfragen über RDF Graphen gesendet formuliert werden. Das folgende Beispiel zeigt eine Abfrage in SPARQL und besteht aus drei wesentlichen Bestandteilen [HKRS08]:

```
PREFIX ical: <http://www.w3.org/2002/12/cal/icaltzd#>
SELECT ?status ?dtstart ?organizer
WHERE {
?events ical:dtstart ?dtstart .
?events ical:organizer ?organizer .
?events ical:status ?status. }
```

Das Schlüsselwort PREFIX legt einen Namensraum (vgl. Kapitel 2.2.2.2) fest. Das Schlüsselwort SELECT ist eines von vier möglichen Ausgabemöglichkeiten von Abfrageresultaten (sogenannte

‚*query forms*'). [HKRS08][HaSe13] Der dritte Bestandteilt ist das Schlüsselwort WHERE, mit dem die eigentliche Anfrage eingeleitet wird. Ihm folgt ein Graphenmuster, welches aus drei Triple mit Variablen besteht. Als Ausgabe liefert dieses Beispiel alle Startzeitpunkte, Organisatoren und Status von Events [HKRS08].

Die vier möglichen ‚*query forms*' sind SELECT, CONSTRUCT, ASK und DESCRIBE und stellen wie oben bereits erwähnt unterschiedliche Ausgabemöglichkeiten von Abfragen dar [HaSe13]. SELECT retourniert entweder die gesamte oder eine bestimmte Teilmenge der von den angeführten Variablen (Graphenmuster) gebundenen Werten. CONSTRUCT retourniert einen RDF Graphen, der durch das Abgleichen von Werten anhand eines bestimmten Graphenmusters entstand. ASK liefert, abhängig davon ob das abgefragte Graphenmuster zutrifft oder nicht, einen Wert ‚wahr' oder ‚falsch' zurück. DESCRIBE liefert als Ergebnis einen RDF Graphen, der Informationen über die angefragten Ressourcen enthält [HaSe13].

Weitere wichtige Schüsselwörter bei SPARQL Anfragen sind FILTER und OPTIONAL. Mit FILTER lassen sich Einschränkungen der Gültigkeitsbereiche von Graphenmustern festlegen [HKRS08]. Im folgenden Beispiel wird mit dem Schlüsselwort FILTER die Menge an Ergebnissen insofern eingeschränkt, dass nur Events die den Status `confirmed` haben in die Ergebnismenge hinzugefügt werden sollen:

```
PREFIX ical: <http://www.w3.org/2002/12/cal/icaltzd#>
SELECT ?status ?dtstart ?organizer
WHERE {
?events ical:dtstart ?dtstart .
?events ical:organizer ?organizer .
?events ical:status ?status.
FILTER ((str(?status)=“confirmed“)}
```

Das Schlüsselwort OPTIONAL erlaubt es, Ergebniszeilen trotz fehlendem Ergebnis bei einer Variable, in die Menge von Ergebnissen hinzuzufügen [HaSe13]. In folgendem Beispiel würde ein Event, welches bei `?organizer` keinen Wert eingetragen hat normalerweise nicht in das Ergebnis hinzugefügt, da es nicht beide Kriterien erfüllt (d.h. sowohl einen Startzeitpunkt als auch einen Organisator zu haben).

```
PREFIX ical: <http://www.w3.org/2002/12/cal/icaltzd#>
SELECT ?status ?dtstart ?organizer
WHERE {
?events ical:dtstart ?dtstart .
 OPTIONAL {?events ical:organizer ?organizer }}
```

Durch das Schlüsselwort OPTIONAL wird es trotz fehlendem Wert in das Ergebnis hinzugefügt [HaSe13].

2.3. ECA - Regeln

Aktive Datenbanksysteme unterstützen die Möglichkeit, auf eintretende Events in und um die Datenbank, zu reagieren. [DiGa96], [WiCe96] und [Pato99] stellen mehrere Ansätze zur Umsetzung vor. Eine Möglichkeit ist es, Regeln einzusetzen, die beim Eintreten bestimmter Ereignisse eine bestimmte Aktion durchführen, sogenannte Event-Condition-Action Regeln. Der grundsätzliche Aufbau dieser Regeln ist:

```
on event if condition do action
```

D.h. zum Eintritt eines bestimmten Ereignisses (`event`), wird eine Bedingung (`condition`) überprüft und anhand des Ergebnisses dieser Überprüfung eine Aktion (`action`) durchgeführt. Dieses reaktive Verhalten bereits etablierter, aktiver relationaler Datenbanksysteme soll auch im Web ausgenützt und verwendet werden [Patr05]. [BrEc06] zeigen auf, dass die Standards HTTP und SOAP bereits für eine Kommunikationsinfrastruktur zur Entwicklung reaktiver Systeme im Web sorgen, jedoch eine Sprache auf abstrakterer Ebene notwendig ist. Damit ist gemeint, dass man eine Art Regelsprache benötigt, die es dem Menschen ermöglicht, komplexe reaktive Abläufe einfach zu modellieren und sich um Systemprobleme und Netzwerkkommunikation im Zuge der Modellierung kümmert und den Anwender nicht damit belastet.

Wie in Kapitel 2.2.2.2 bereits erläutert, ist die Serialisierung RDF/XML weit verbreitet und es wurden daher bereits Überlegungen angestellt, wie man ECA – Regeln sowohl in RDF als auch in XML umsetzen kann. [ScBe01] haben bereits gezeigt, wie man XML – Schemas mit ECA-Regeln erweitern und manipulieren kann. [PoPW06] stellen eine ECA – Regelsprache speziell für XML und RDF vor. Dabei werden die vom W3C entwickelten Abfragesprachen XQuery und XPath eingesetzt. Der Aufbau ihrer Regeln sieht folgendermaßen aus:

```
ON INSERT XPath-Ausruck
IF XQuery-Ausdruck oder Konstante TRUE
DO DELETE/INSERT/UPDATE XPath-Ausdruck
```

[PoPW06] haben diese Regeln speziell für RDF dahingehend noch weiter ausgebaut, dass Regeln direkt auf RDF-Graphen- bzw. RDF-Triple-Ebene eingreifen können. D.h. mit dem XPath – Ausdruck wird bis zum Objekt eines Triple navigiert und dessen Literal überprüft. In weiterer Folge wurde die ECA-Regelsprache RDFTL entwickelt, welche den Einsatz in verteilten Peer 2 Peer Netzwerken erlaubt. [PaPW06] [BrEP06] RDFTL ist ein möglicher Ansatz um ECA – Regeln in RDF mit einer RDF/XML Serialisierung umzusetzen.

2.4. JENA

JENA ist ein Open Source Framework für das Semantic Web in Java [Apac11]. Es stellt eine Programmierschnittstelle zu Verfügung mit der man RDF Graphen manipulieren kann. JENA ermöglicht die Erstellung und Erweiterung eines Datenmodells basierend auf RDF Graphen. Weiters unterstützt es die Abfragesprache SPARQL sowie RDF/XML, Turtle und Notation 3 (vgl. Kapitel 2.2.2.2) zur Serialisierung von RDF Graphen.

In Jena werden alle Informationen, welche durch eine Menge von RDF Triple beschrieben werden, in einer Datenstruktur, dem `Model`, hinterlegt [Apac11]. Dieses `Model` beschreibt einen RDF-Graphen (vgl. Kapitel 2.2.2). `Graph` ist eine einfachere Java API als `Model`, die dafür gedacht ist, die Funktionalität von JENA zu erweitern. `Model` ist eine Java API, welche alle notwendigen Methoden zur Manipulation von RDF Daten in JENA zur Verfügung stellt [Apac11].

3. Spezifikation des reaktiven Terminkalenders

Im folgenden Kapitel soll ein möglicher Lösungsansatz erarbeitet und detailliert erläutert werden. Dabei sollen die funktionalen und nicht funktionalen Anforderungen aus Kapitel 1.1 in Betracht gezogen werden. Der Lösungsweg wird von konzeptueller Sicht aus untersucht, ausgearbeitet und anschließend von für die Arbeit nicht relevanten Bereichen abgegrenzt.

3.1. Funktionale Anforderungen

Es wird nun aus konzeptueller Sicht der gewählte Lösungsansatz beschrieben und näher erläutert. Dabei wird nach einer detaillierten Beschreibung des Entwurfs näher auf das Konzept der Regeln, Terminklassen, Prioritäten, Terminwichtigkeiten und Zustandsänderungen von Terminen eingegangen.

3.1.1. Entwurf

Aus den in Kapitel 1.1 vorgegebenen funktionalen und nicht funktionalen Anforderungen wurden die in Abbildung 6 modellierten Klassen und deren Beziehungen herausgearbeitet. Dabei wurde in Betracht gezogen, dass das Wort „Termin“ unterschiedliche Arten der Auslegung ermöglicht. Ein Termin kann ein abstraktes Konzept, zu dem es mehrere mögliche Abhaltungsmöglichkeiten gibt sein, oder beispielsweise ‚nur‘ eine Aufgabe für nur eine Person (z.B. seine Tochter vom Kindergarten abzuholen). Bei der Auswahl von möglichen Terminkategorien wird das bereits bestehende RDF Vokabular RDF Calendar [CoMi05] (vgl. 2.2.2.3) herangezogen. Im Folgenden werden nun die Klassen und deren Beziehungen näher betrachtet:

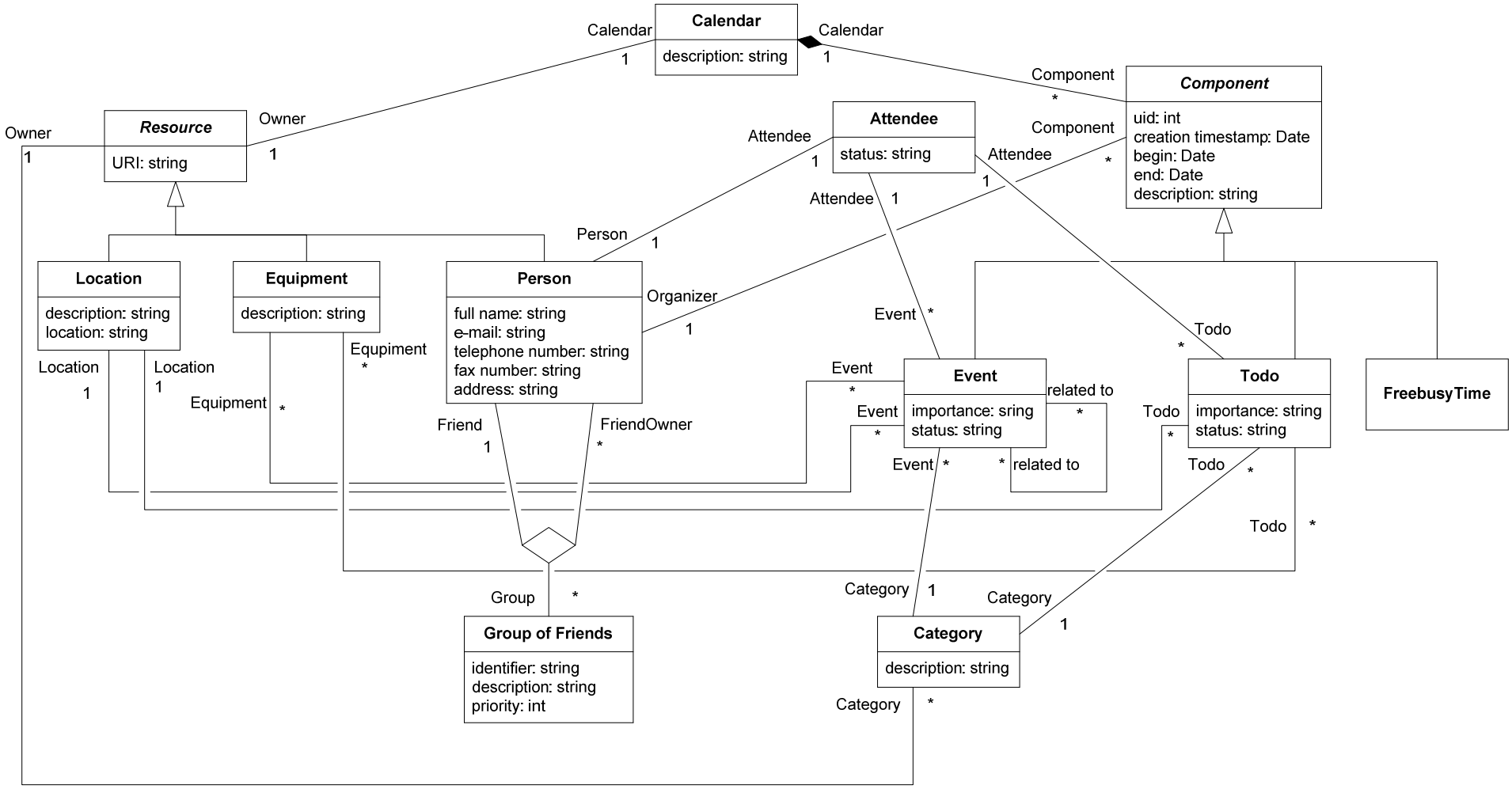


Abbildung 6 Klassendiagramm zur Umsetzung der Anforderungen

Die Klasse `Calendar` stellt das zentrale Terminkalenderobjekt jeder einzelnen Ressource dar. Jede Ressource, die Teil der Anwendung ist, besitzt genau einen Kalender der sämtliche Termine dieser Ressource beinhaltet. Der Kalender ist das Grundgerüst zur Repräsentation von Terminen einer bestimmten Ressource. Einem Terminkalender sind mehrere Terminkomponenten zugeordnet.

Die Klasse `Component` ist eine generelle Klasse für alle möglichen Inhalte eines Kalenders. Angelehnt an das RDF Calendar Vokabular (vgl. Kapitel 2.2.2.2), spezialisiert sich `Component` in die drei möglichen Terminvarianten `Todo`, `Event`[8] und `FreeBusyTime`. Jede `Component` besitzt, ebenfalls angelehnt an das RDF Calendar Vokabular, genau einen Organisator, welcher eine `Person` sein muss. Weiters hat jede `Component` eine eindeutige ID, einen Zeitpunkt der Erstellung, einen Start- und Endzeitpunkt und eine Beschreibung. `Component` ist neben `Resource` eine der zwei abstrakten Klassen welche als generelles Konzept dienen sollen.

Die Klasse `Todo` ist eine mögliche Terminvariante aus dem Vokabular RDF Calendar, welche einen Termin darstellt, der nur einen Teilnehmer (`Attendee`), nämlich dessen Organisator hat. Ein Beispiel für einen solchen Termin wäre eine sportliche Aktivität zu betreiben oder an einer Seminararbeit zu arbeiten. Zusätzlich zu den Eigenschaften einer `Component`, besitzt `Todo` die Attribute Kategorie, Wichtigkeit, Status und findet genau an einem Ort (`Location`) statt.

Die Klasse `Event` ist eine weitere mögliche Terminvariante aus dem Vokabular RDF Calendar, welche einen Termin im herkömmlichen Sinne mit mehreren Teilnehmern darstellt. Der Termin hat neben den Eigenschaften einer `Component` noch einen Status. Dieser Status ist nicht zu verwechseln mit dem Status eines Teilnehmers. Weiters hat `Event` eine Kategorie und eine Wichtigkeit. Ein `Event` kann auch mehrere mögliche Abhaltungen haben, wobei davon genau eine mögliche Abhaltung fixiert wird. Diese Abhaltungen wissen voneinander mittels der Beziehung `related-to` und haben die gleichen Eigenschaften wie ein (normales) `Event`.

Die Klasse `FreeBusyTime` ist die dritte aus dem Vokabular RDF Calendar abgeleitete Terminvariante, welche einen Zeitraum darstellt, in dem eine `Resource` nicht für Termine verfügbar ist, beispielsweise Semesterferien oder Betriebsurlaub. `FreeBusyTime` hat keinen Ort, keinen Status, keine Teilnehmer und auch keine Wichtigkeit. Sie besitzt lediglich die Eigenschaften von `Component`.

Die Klasse `Attendee` stellt den Teilnehmer eines Termins dar und hat als Eigenschaft einen Status, der für diesen Termin eindeutig ist.

[8] Nicht zu verwechseln mit einem Event im Sinne eines eintretenden Ereignisses. Mit Event wird an dieser Stelle eine bestimmte Art von Termin bezeichnet.

Die abstrakte Klasse Resource bezeichnet den Besitzer eines Kalenders. Resource spezialisiert sich in entweder einen Raum, einen Gegenstand oder eine Person. Eine Resource besitzt genau einen Terminkalender und ist durch eine URI eindeutig identifizierbar.

Die Klasse Person ist eine mögliche Spezialisierung von Resource und stellt eine menschliche Person dar. Die Attribute dieser Resource sind an das Vokabular vCard angelehnt [Iann10]. Neben der von Resource erhaltenen eindeutigen URI, besitzt Person einen Namen, eine E-Mail Adresse, mehrere mögliche Telefonnummern und mehrere mögliche Adressen. Eine Person kann Termine organisieren, Kategorien erstellen, an Terminen teilnehmen und Freundesgruppen und dessen Prioritäten erstellen.

Die Klasse Equipment ist eine weitere mögliche Spezialisierung von Resource und stellt einen Gegenstand dar, der bei einem Termin benötigt wird, beispielsweise ein Videoprojektor oder ein Buch. Neben einer eindeutigen URI besitzt der Gegenstand eine Bezeichnung.

Die Klasse Location ist die dritte mögliche Spezialisierung von Resource und bezeichnet einen Ort, an dem Termine stattfinden können, beispielsweise Hörsaal 16. Neben einer eindeutigen URI zur Identifizierung besitzt Location eine Ortsangabe.

Die Klasse Group of Friends stellt eine Freundesgruppe dar. Jede Person kann mehrere Freundesgruppen mit Prioritäten erstellen und diesen Freundesgruppen Teilnehmer zuweisen. Ein Teilnehmer kann sich in mehreren Freundesgruppen befinden.

Die Klasse Category stellt die Terminkategorie dar. Sowohl Todo als auch Event haben jeweils genau eine Terminkategorie, welcher sie angehören. Eine Terminkategorie hat eine Bezeichnung und genau einen Besitzer.

Zusätzlich zu den Anforderungen wird folgende Annahme getroffen und vorausgesetzt:

Hat ein Teilnehmer zu einem Termin zugesagt, gilt er für diesen Zeitraum als beschäftigt; unabhängig davon ob der Termin zu dem er zugesagt hat bereits confirmed ist (dieser Termin könnte noch pending sein)

3.1.2. Terminklassen

Aus den eben beschriebenen Klassen ergeben sich somit die Terminarten Event, Todo und FreeBusyTime. Aus der funktionalen Anforderung, dass ein Termin auch mehrere mögliche Abhaltungen haben kann ergibt sich jedoch noch eine weitere Terminart, welche als Multievent

bezeichnet werden soll, um eine Unterscheidung zu einem Termin mit nur einer Abhaltungsmöglichkeit zu gewährleisten.

Ein `Multievent` soll dabei nur einen Terminkopf haben, welcher durch die Beschreibung des Termins, der Wichtigkeit, den Teilnehmern und einer Kategorie bezeichnet werden soll. Jede einzelne Abhaltungsmöglichkeit des `Multievents` soll einen bestimmten Start- und Endzeitpunkt, einen bestimmten Ort und einen Status haben und über die anderen Abhaltungsmöglichkeiten Bescheid wissen.

3.1.3. Zustände und Zustandswechsel von Terminen und Teilnehmern

Aus den beschriebenen Klassen lassen sich folgende Termin- und Teilnehmerzustände ableiten:

Ein Termin kann die Zustände abwartend (`pending`), fixiert (`confirmed`) und abgebrochen (`cancelled`) haben. Ein Teilnehmer kann die Zustände abwartend (`pending`), zugesagt (`confirmed`), abgesagt (`declined`) und abgebrochen (`cancelled`) haben.

Ein Termin der Klasse `Todo` hat nach der Erstellung den Status `confirmed`. Auch der Organisator hat den Teilnehmerstatus `confirmed`. Da kein anderer Teilnehmer teilnehmen wird, ist somit kein abwartender Zustand erforderlich. `Todo` kann auch den Status `cancelled` einnehmen, sofern der Organisator seinen Status auf `declined` oder den Terminstatus auf `cancelled` setzt.

Ein Termin der Klasse `Event` hat nach der Erstellung den Status `pending`. Der Organisator hat den Teilnehmerstatus `confirmed`. Jeder weitere Teilnehmer hat nach der Erstellung des Termins den Teilnehmerstatus `pending`. Gleiches gilt für die Abhaltungsmöglichkeiten eines `Multievents`. Zum Erstellungszeitpunkt haben sie den Status `pending` und der Organisator den Teilnehmerstatus `confirmed`, während jeder weitere Teilnehmer den Teilnehmerstatus `pending` hat. Ändert sich ein Terminstatus bei einer möglichen Abhaltung eines `Multievents`, sollen die anderen möglichen Abhaltungen abgebrochen werden, da nur eine mögliche Abhaltung eines `Multievents` fixiert sein kann.

Ein Termin der Klasse `FreeBusyTime` hat weder einen Termin- noch einen Teilnehmerstatus. Das bedeutet, dass Termine dieser Klasse auch nicht abgesagt werden können, d.h. einmal erstellt, gilt der Organisator in diesem Zeitraum als beschäftigt.

Hat ein `Todo` oder ein `Event` einmal den Status `cancelled`, kann es diesen Status nicht mehr verlassen und hat es den Status `pending` einmal verlassen, kann es nicht mehr dahin zurückkehren. Gleiches gilt für die Teilnehmerstatus `pending` und `declined`. Hat ein Teilnehmer

einmal `declined` oder den Status `pending` verlassen, kann er den Status nicht mehr ändern bzw. dahin zurückkehren.

Zustandsänderungen von Teilnehmerstatus können entweder durch die Teilnehmer selbst oder aber durch Regeln ausgelöst werden. Diese Statusänderungen sollen, sofern möglich, unmittelbar nachdem der Änderungswunsch geäußert wurde in das Datenmodell geschrieben werden. Ist der Teilnehmer des Termins jedoch nicht gleichzeitig auch der Ersteller, kann der Änderungswunsch nur in zwei Schritten durchgeführt werden. Die Statusänderung von Terminen wird normalerweise durch den Organisator durchgeführt und kann somit automatisch erfolgen. Eine Ausnahme stellt hier jedoch das Absagen eines bereits fixierten Termins durch einen Teilnehmer dar. Diese Absage muss wieder in zwei Schritten durchgeführt werden.

Zum Schreiben auf RDF Dateien über HTTP mittels HTTP PUT (vgl. Kapitel 1) wird eine Authentifizierung benötigt. Da nicht davon auszugehen ist, dass ein Teilnehmer die Zugangsdaten von Webservern anderer Teilnehmer besitzt, muss der Änderungswunsch deshalb halb-automatisch in zwei Schritten vorgenommen werden. Nach Bekanntwerden des Änderungswunsches soll deshalb im ersten Schritt in der RDF Datei des Teilnehmers eine Notiz für den Organisator des Termins, dessen Teilnehmerstatus geändert werden soll, hinterlegt werden. Beim nächstmöglichen Schreibzugriff auf die RDF Datei des Organisators des besagten Termins, soll die Antwort im zweiten Schritt automatisch umgesetzt und geschrieben werden.

3.1.4. Priorität, Wichtigkeit und Freundesgruppen

Es gilt die Freundespriorität und die Terminwichtigkeit zu unterschieden. Ein Termin kann die Wichtigkeitskategorien normal (`standard`), wichtig (`important`) und sehr wichtig (`very important`) einnehmen. Jeder Teilnehmer befindet sich in einer oder in mehreren Freundesgruppe(n) des Terminorganisators. Der Terminorganisator hat dieser Freundesgruppe eine Priorität zugewiesen. Abhängig von der Priorität der Freundesgruppe in der sich der Teilnehmer befindet, hat dieser eine bestimmte Priorität von 1-10 (bei diesem Terminorganisator).

Es wird die Funktionalität abgebildet, dass man Freundesgruppen erstellen und neue Freunde hinzufügen kann. Weiters ist es möglich, diese Personen in die Freundesgruppen mit einer bestimmten Priorität hinzuzufügen.

3.1.5. Regeln

Das reaktive und proaktive Verhalten soll, sofern möglich, generisch und erweiterbar gehalten werden.

Um das reaktive Agieren der Anwendung zu ermöglichen, werden ECA-Regeln (vgl. Kapitel 2.3) eingesetzt, während hingegen das proaktive Handeln der Anwendung sowohl durch ECA-Regeln als auch durch vom System vorgegebene Regeln umgesetzt wird. Dabei wird zwischen systemweiten Regeln und privaten Regeln unterschieden. Eine systemweite Regel ist z.B. dass der Organisator eines Termins bei diesem auch dabei sein muss. Ein Beispiel für eine private Regel wäre das Ablehnen von Terminen einer bestimmten Person. Eine systemweite Regel ist jedoch mächtiger als eine private Regel und kann sie somit außer Kraft setzen. Wird in einer privaten Regel eine Bedingung einer systemweiten Regel missachtet, wird sie von der systemweiten Regel überstimmt.

Jeder Ressource, bzw. dem Besitzer dieser Ressource, soll es selbst möglich sein, private Regeln zu definieren. Kann eine Aktion einer Regel nicht automatisch vom System ausgeführt werden, so soll der Benutzer darüber informiert und auf seine Entscheidung gewartet werden.

3.1.5.1. Events (E) einer ECA - Regel

Die für den Benutzer bei den Regeln zu definierenden Ereignisse sollen erweiterbar gehalten werden. Folgende mögliche eintretende Ereignisse (E) sollen auf jeden Fall unterstützt werden:

- Der Benutzer wird zu einem Termin eingeladen
- Ein Freund antwortet auf die Einladung des Benutzers
- Der Benutzer sagt eine Einladung ab / Der Benutzer antwortet auf eine Einladung
- Der Benutzer ladet jemanden zu einem Termin ein
- Ein Freund antwortet auf eine Termineinladung bei dessen Termin der Benutzer auch teilnimmt, der Benutzer jedoch nicht der Organisator des Termins ist

3.1.5.2. Bedingungen (C) einer ECA - Regel

Die Bedingungen einer Regel sollen vom Benutzer selbst festgelegt werden können und von ganzen Freundesgruppen bis zur einzelnen Termineigenschaft beliebig verschachtelbar sein. Ein Beispiel für eine Bedingung (C) wäre z.B. falls ich zu einem Termin einer bestimmten Terminkategorie eingeladen werde, muss die Priorität des Erstellers größer als 5 sein. Der Benutzer soll diese Regel mit vordefinierten Bausteinen verschachteln und beliebig strukturieren können.

Die Bausteine selbst sollen durch logische Operatoren in Verbindung mit runden Klammern als Begrenzung verbunden werden. In Tabelle 7 werden die Schlüsselwörter angeführt und kurz beschrieben.

Schlüsselwort	Beschreibung
ON	legt das eintretende Ereignis fest, bei welchem die Regel ausgelöst werden soll
IF	legt die Bedingung fest, welche überprüft werden soll
DO	legt die Aktion fest, welche durchgeführt werden soll
AND	verknüpft Bausteine mit einer logischen Und-Verknüpfung
OR	verknüpft Bausteine mit einer logischen Oder-Verknüpfung
>	Vergleichsoperator „größer als“ für Bausteine
<	Vergleichsoperator „kleiner als“ für Bausteine
=	Vergleichsoperator für Bausteine die Zeitpunkte betreffen
IS	Vergleichsoperator für Bausteine die Ressourcen, Kategorien oder Freundesgruppen betreffen
TO	drückt eine Zustandsänderung zu einem bestimmen Zustand aus
OF	Verknüpft einen Baustein mit einer Resource
!=	Vergleichsoperator für Bausteine die Zeitpunkte betreffen
ISNOT	Vergleichsoperator für Bausteine die Ressourcen, Kategorien oder Freundesgruppen betreffen

Tabelle 7 Schlüsselwörter einer Regel

In Tabelle 8 werden Bausteine angeführt und kurz beschrieben, welche mögliche eintretende Ereignisse dieses speziellen Anwendungsfalles darstellen.

Baustein	Beschreibung
NEW EVENT	Ein eintretendes Ereignis bei der Erstellung eines neuen Events
NEW TODO	Ein eintretendes Ereignis bei der Erstellung eines neuen Todos
NEW FREEBUSY	Ein eintretendes Ereignis bei der Erstellung einer neuen FreeBusyTime
UPDATE EVENT	Ein eintretendes Ereignis bei der Aktualisierung eines Events
UPDATE TODO	Ein eintretendes Ereignis bei der Aktualisierung eines Todos
UPDATE FREEBUSY	Ein eintretendes Ereignis bei der Aktualisierung einer FreeBusyTime
CANCEL EVENT	Ein eintretendes Ereignis beim Abbruch eines Events
CANCEL TODO	in eintretendes Ereignis beim Abbruch eines Todos
CANCEL FREEBUSY	in eintretendes Ereignis beim Abbruch einer FreeBusyTime

Tabelle 8 Bausteine für eintretende Ereignisse einer Regel

In Tabelle 9 werden all jene Bausteine, welche zur Erstellung von Bedingungen einer Regel für diesen speziellen Anwendungsfall benötigt werden angeführt und kurz beschrieben.

Baustein	Beschreibung
ORGANIZER	Baustein für einen Organisator eines Termins
ORGANIZER.PRIORITY	Baustein für die Priorität eines Organisators
BUSY	Baustein der festlegt, ob ich zu dieser Zeit beschäftigt bin
BUSY.PRIORITY	Baustein für die Priorität des existierenden Termins
FREE	Baustein der festlegt, ob ich zu dieser Zeit für Termine offen bin
DTSTART	Startzeitpunkt eines neuen Termins
DTEND	Endzeitpunkt eines neuen Termins

IMPORTANCE	Wichtigkeit eines neuen Termins
STANDARD	eine mögliche Wichtigkeitskategorie
IMPORTANT	eine mögliche Wichtigkeitskategorie
VERY IMPORTANT	eine mögliche Wichtigkeitskategorie
CATEGORY	Baustein für eine Terminkategorie
ATTSTATUS	Baustein für den Status eines Teilnehmers
STATUS	Baustein für den Status eines Termins
PENDING	ein möglicher Termin- oder Teilnehmerstatus
CONFIRMED	ein möglicher Termin- oder Teilnehmerstatus
DECLINED	ein möglicher Termin- oder Teilnehmerstatus
CANCELLED	ein möglicher Terminstatus
FRIENDGROUP	Baustein für eine Freundesgruppe
FRIENDGROUP.PRIORITY	Baustein für die Priorität einer Freundesgruppe
LOCATION	Baustein für einen bestimmten Ort
EQUIPMENT	Baustein für einen bestimmten Gegenstand

Tabelle 9 Bausteiner zur Festlegung der Bedingung einer Regel

In Tabelle 10 werden jene Bausteine näher beschrieben und kurz erläutert, welche zur Festlegung einer bestimmten Aktion einer Regel benötigt werden.

Baustein	Beschreibung
DECLINE APP	Baustein für die durchzuführende Aktion zum Absagen eines Termins
NOTIFY APP	Baustein für die durchzuführende Aktion zum Benachrichtigen eines Termins
CONFIRM APP	Baustein für die durchzuführende Aktion zum Zusagen eines Termins
CANCEL APP	Baustein für die durchzuführende Aktion zum Abbrechen eines Termins

Tabelle 10 Bausteine zur Festlegung einer bestimmten Aktion einer Regel

Jede existierende Ressource (sowohl Personen, Orte als auch Gegenstände), Kategorie und Freundesgruppe soll ebenfalls als Baustein fungieren können.

Als Alternative zur Bausteinmethode soll die Möglichkeit geschaffen werden, dem Benutzer selbst eine SPARQL ASK Abfrage als Regelbedingung eingeben zu lassen.

3.1.5.3. Aktionen (A) einer ECA - Regel

Nach der Festlegung vom Ereigniszeitpunkt und der zu überprüfenden Bedingung muss noch die auszuführende Aktion festgelegt werden. Die für den Benutzer auszuwählenden Aktionen (A) sollen folgende sein:

- Der Benutzer möchte seinen Teilnehmerstatus für diesen Termin auf zusagen ändern
- Der Benutzer möchte seinen Teilnehmerstatus für diesen Termin auf absagen ändern
- Der Organisator möchte den Terminstatus für diesen Termin auf abbrechen ändern
- Der Organisator möchte den Terminstatus für diesen Termin auf bestätigt ändern

– Der Benutzer möchte über diesen Termin benachrichtigt werden

3.1.5.4. Beispiele für mögliche Regeln

Es sollen zur Demonstration unterschiedlicher Mechanismen einige bestimmte ECA-Regeln abgebildet werden. Es soll dabei das Verhalten sowohl beim Einfügen, als auch beim Aktualisieren eines Termins abgebildet werden. Ebenfalls abgedeckt soll die Überprüfung von Zeitüberschneidungen und das Vergleichen von bestimmten Prioritäten werden. Dazu wurden folgende vier Regeln ausgearbeitet:

1. Regel: Der Benutzer möchte über alle neu erstellten Termine die die höchste Wichtigkeitskategorie besitzen informiert werden, egal ob er bereits einen Termin zu diesem Zeitpunkt hat oder nicht.
2. Regel: Bei einer Terminkollision bei der Erstellung eines neuen Termins bei dem der Benutzer Teilnehmer ist, soll überprüft werden, ob die Priorität des Erstellers höher ist als die Priorität des Erstellers des bereits fixierten Termins. Sollte die Priorität nicht höher sein, soll der neue Termin automatisch abgelehnt werden.
3. Regel: Wird ein Termin einer bestimmten Terminkategorie aktualisiert, soll überprüft werden, ob eine bestimmte Person zu diesem Termin zugesagt hat. Hat sie zugesagt, soll mein eigener Teilnehmerstatus bei diesem Termin auf abgesagt geändert werden.
4. Regel: Wird ein neuer Termin einer bestimmten Terminkategorie bei dem der Benutzer als Teilnehmer eingeladen wird erstellt, soll überprüft werden, ob der Ersteller eine gewisse Priorität überschreitet und ob der Benutzer keinen weiteren Termin in dieser Zeit hat. Trifft dies zu, soll der Termin automatisch zugesagt werden.

Von einer Anwendung die Termine verwaltet, werden intuitive Handlungen erwartet, die für einen Menschen als sinnvoll erachtet werden. Es folgt eine Liste von Systemregeln, welche für alle Ressourcen gelten:

– Wenn vier von fünf Teilnehmern eines Termins absagen und dann auch noch der fünfte Teilnehmer seinen Terminstatus auf absagen ändert, soll der Termin automatisch abgebrochen werden.

– Wenn eine Ressource einen bereits fixierten Termin abbricht, sollen alle anderen Teilnehmer darüber informiert werden, da man nicht davon ausgehen kann, dass jeder Teilnehmer regelmäßig seine bereits fixierten Termine auf Teilnehmerstatusänderungen überprüft.

– Der Organisator eines Termins muss zwingend am Termin teilnehmen.

- Ein Teilnehmer der zu einem Termin zugesagt hat, gilt für diesen Zeitraum als beschäftigt, d.h. unabhängig davon ob der Terminstatus bereits fixiert ist oder noch abwartend.
- Eine Ressource kann zu einem Zeitpunkt nur einen Termin wahrnehmen.
- Es müssen keine bestimmten Zeitfenster zwischen zwei Terminen frei bleiben, d.h. wenn um 15 Uhr ein Termin endet, kann ab 15:01 Uhr ein neuer Termin wahrgenommen werden.

3.1.5.5. (Halb-)automatisches Verhalten

Die verteilte Datenhaltung auf verschiedenen Webservern erschwert das reaktive Verhalten, insbesondere wenn man Regeln miteinbezieht. Nehmen wir das Beispiel eines zu erstellenden Termins, bei dem zum Zeitpunkt der Erstellung alle Regeln der Teilnehmer überprüft, und darauf entsprechend reagiert werden soll. Zur Verdeutlichung soll das Beispiel aus Kapitel 1.2 um einen weiteren Teilnehmer, Marcus Yallow erweitert werden, welcher eine für Freunde sichtbare Regel in seinem Kalender definiert hat, die besagt, dass er bei Terminkollisionen über den neuen Termin informiert werden möchte. Weiters hat er zum Zeitpunkt der ersten möglichen Abhaltung bereits einen vorhandenen fixierten Termin („Musikstunde“, am 05.05.2014 von 18:00 – 19:00 Uhr).

Erstellt nun John den Termin am 04.05.2014 um 07:33 Uhr, erkennt das System folgende Kollisionen, welche von Regeln abgedeckt werden:

- Matthews Regel führt dazu, dass Matthew Mögliche Abhaltung 1 automatisch ablehnt
- Peters Regel führt dazu, dass auch Peter die Mögliche Abhaltung 1 automatisch ablehnt
- Marcus‘ Regel führt dazu, dass sein Terminstatus weiterhin auf abwartend bleiben soll, er jedoch über diese mögliche Abhaltung informiert werden möchte.

Die Erweiterung mit Marcus führt dazu, dass die mögliche Abhaltung 1 nicht automatisch abgebrochen wird, wie in Kapitel 1.2 angeführt. Weiters führt die Erweiterung dazu, dass nicht nur der Termin im Kalender von John eingetragen werden soll, sondern auch eine Benachrichtigung für Marcus‘ Kalender, dass die Anwendung ihm bei der nächsten Möglichkeit (beim nächsten Login von Marcus) diesen Termin mit besonderem Status anzeigt. Diese besondere Statusanzeige soll solange angezeigt werden, bis Marcus auf die mögliche Abhaltung 1 antwortet. Sobald geantwortet wurde, soll die Benachrichtigung entfernt werden. Die Antwort und die Entfernung der Benachrichtigung soll in einem Schritt erfolgen. Bei der nächsten

Möglichkeit (d.h. beim nächsten Login von John), soll die Anwendung die Benachrichtigung für Marcus aus seinem Kalender entfernen und den Teilnehmerstatus von Marcus bei diesem Termin ändern. Somit ergibt sich ein halb-automatisches Verhalten des Systems nicht nur bei Zustandsänderungen (vgl. Kapitel 3.1.3), sondern auch bei Regeln.

3.2. Abgrenzung

Im Zuge der Implementierung werden mehrere Aspekte außer Acht gelassen, da deren nähere Behandlung für diese Arbeit nicht relevant ist. Zur Demonstration der Funktionalität ist die Erstellung eines Prototyps ausreichend. Deshalb wird bei der Umsetzung kein besonderes Auge auf folgende Bereiche gelegt:

Verteilung: Eine verteile, dezentrale Logik beim Client soll außer Acht gelassen und dessen Umsetzung nicht verfolgt werden.

Performanz: Die Menge an Terminen in einem Kalender um eine für den Benutzer erträgliche Antwortzeit zu erhalten ist für diese Arbeit nicht wichtig, es soll lediglich die Funktionalität getestet werden und dazu reichen wenige (<100) Termindaten.

Archivierung: Um Speichergröße und Performanz im Rahmen zu halten, wäre eine Archivierung alter Termine (die den Status abgebrochen haben oder bereits in der Vergangenheit liegen) sinnvoll. Die Archivierung ist jedoch nicht Teil dieser Arbeit und soll deshalb ignoriert werden

Zugriffsrechte: Abgesehen von der Sichtbarkeit von privaten Daten, wird auf Zugriffsrechte auf bestimmte RDF Dateien keine Rücksicht genommen. D.h. sobald jemand die URIs der Dateien mit dem Protected-Graph kennt, kann er auch dessen Inhalt lesen. Auch beim Zugriff auf bestimmte Kalender (Login) wird kein bestimmtes Passwort abgefragt oder eine Session erstellt. Dies ist für die Überprüfung der Funktionalität irrelevant.

Verfügbarkeit von Daten: Das System geht davon aus, dass der Zugriff auf die RDF Dateien immer gegeben ist. D.h. die Pfade der Dateien dürfen sich nach dem Hinzufügen als Freund nicht mehr ändern. Sind diese Dateien nicht mehr erreichbar (weil sich z.B. der Pfad geändert hat), werden sie auch nicht mehr gefunden.

Usability der UI: Die Benutzeroberfläche soll als Prototyp dargestellt werden. D.h. das Abfangen von sämtlichen Falscheingaben ist nicht von Relevanz.

4. Implementierungsansatz mit RDF

Im folgenden Kapitel wird nun die Realisierung des gewählten Lösungsansatzes aus Kapitel 3 in RDF beschrieben und näher erläutert. Dabei wird nach der Beschreibung des speziell für die Terminorganisation erstellten Vokabulars näher auf die Sichtbarkeit von Benutzerdaten, die logische Sicht der Terminklassen, Prioritäten, Terminwichtigkeiten und Regeln eingegangen.

4.1. Das Vokabular sem_cal

Für den Anwendungsfall einer Terminorganisation wird ein spezielles Vokabular entworfen, welches alle verlangten Anforderungen abbilden kann. Dabei werden Klassen für die Ressourcen, die Teilnehmer, die Regeln, die Terminkategorien, die Freundesgruppen, die Termin- und Teilnehmerzustände sowie einige Klassen für das reaktive Verhalten der Anwendung eingeführt. Folgende Auszüge sollen stellvertretend für alle Änderungen die notwendigen Erweiterungen verdeutlichen:

```
01 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:sem="http://localhost/rdf/sem_cal.rdf#"
02 ...
03 <!-- classes -->
04 <rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#Resource"/>
05 <rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#Person">
06 <rdfs:subClassOf
rdf:resource="http://localhost/rdf/sem_cal.rdf#Resource"/>
07 </rdfs:Class>
08 <rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#Room">
09 <rdfs:subClassOf
rdf:resource="http://localhost/rdf/sem_cal.rdf#Resource"/>
10 </rdfs:Class>
11 <rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#Equipment">
12 <rdfs:subClassOf
rdf:resource="http://localhost/rdf/sem_cal.rdf#Resource"/>
13 </rdfs:Class>
14 <rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#Attendee"/>
15 <rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#Rule"/>
16 <rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#IDUpdate"/>
17 ...
```

`Person`, `Room` und `Equipment` sind Subklassen der Klasse `Resource`, welche alle Nichtinformationsressourcen darstellen soll. `Attendee` stellt den Teilnehmer dar, `Rule` die vom Benutzer angelegten ECA – Regeln und `IDUpdate` dient zur Unterstützung des reaktiven Verhaltens der Anwendung.

Weiters wurden für diese Klassen die erforderlichen Eigenschaften festgelegt. Als Beispiel sollen hier die Eigenschaften eines Teilnehmers angeführt werden. `attendeePerson` stellt dabei die Teilnehmerresource dar, während `attendeeStatus` den Teilnehmerstatus widerspiegelt.

```
01 ...
02 <!-- properties -->
03 <rdf:Property
rdf:about="http://localhost/rdf/sem_cal.rdf#attendeePerson">
04 <rdfs:domain
rdf:resource="http://localhost/rdf/sem_cal.rdf#Attendee"/>
05 <rdfs:range rdf:resource="http://localhost/rdf/sem_cal.rdf#Person"/>
06 </rdf:Property>
07 <rdf:Property
rdf:about="http://localhost/rdf/sem_cal.rdf#attendeeStatus">
08 <rdfs:domain
rdf:resource="http://localhost/rdf/sem_cal.rdf#Attendee"/>
09 <rdfs:range
rdf:resource="http://localhost/rdf/sem_cal.rdf#Attendeestatus"/>
10 </rdf:Property>
11 ...
```

Neben den Klassen und Eigenschaften wurden noch die Terminprioritäten, die Terminstatus und die Teilnehmehrstatus dieser Anwendung in das Vokabular aufgenommen. In folgendem Beispiel werden stellvertretend die Teilnehmerstatus dargestellt. Das vollständige Vokabular ist dem Anhang zu entnehmen.

```
01 ...
02 <sem:Attendeestatus rdf:about="aStatus_confirmed">
03    <sem:attendeeStatusValue>confirmed</sem:attendeeStatusValue>
04 </sem:Attendeestatus>
05 <sem:Attendeestatus rdf:about="aStatus_pending">
06    <sem:attendeeStatusValue>pending</sem:attendeeStatusValue>
07 </sem:Attendeestatus>
08 <sem:Attendeestatus rdf:about="aStatus_declined">
09    <sem:attendeeStatusValue>declined</sem:attendeeStatusValue>
10 </sem:Attendeestatus>
11 <sem:Attendeestatus rdf:about="aStatus_cancelled">
12    <sem:attendeeStatusValue>cancelled</sem:attendeeStatusValue>
13 </sem:Attendeestatus>
14 </rdf:RDF>
```

Klassen- und Eigenschaftsnamen wurden an die von [HKRS08] vorgeschlagene Notation angelehnt, bei der kleingeschriebene Eigenschaftsnamen und großgeschriebene Klassennamen empfohlen werden.

4.2. Sichtbarkeit von Benutzerdaten

Um der Vorgabe einer verteilten Datenhaltung gerecht zu werden, die Sichtbarkeit von Daten einzuschränken und jeder Benutzer selbst über seine Daten verfügen können soll, werden die Daten verteilt auf einer vom Benutzer gewählten URI abgelegt. Da jedoch nicht alle Inhalte dieses

Benutzers publik gemacht werden sollen, müssen Einschränkungen der Sichtbarkeit definiert werden.

Um eine Transparenz innerhalb der Datenstruktur zu schaffen, wird eine Trennung der Daten in drei separate Graphen vorgenommen. Die Aufteilung wird durch folgende Graphen repräsentiert:

Der Public-Graph*:* Im Public-Graph stehen jene Informationen, die für sämtliche Ressourcen sichtbar sein sollen. Das sind Daten zur Kontaktaufnahme und Termine.

Der Protected-Graph*:* Im Protected-Graph stehen jene Informationen, die nur für bestimme Ressourcen (z.B. Freunde) sichtbar sein sollen. Das sind nähere Details zur Person, Freundesgruppen, Terminkategorien und Regeln.

Der Private-Graph*:* Im Private-Graph stehen jene Informationen, die nur für die Ressource selbst sichtbar sind. Das sind Prioritäten und private Regeln, sowie die URIs der hinzugefügten Freunde, Räume und Gegenstände.

Durch das Aufteilen der Daten entsteht das Problem, dass die Daten nun nicht mehr zusammenhängend in einer einzigen Quelle zu finden sind, sondern jeder Graph jeweils in einer RDF Datei abgelegt ist. In RDFS ist es jedoch möglich, Relationen zwischen einzelnen Dateien herzustellen. In diesem speziellen Fall wird mittels der Property `rdfs:seeAlso` diese Verbindung zwischen den getrennten Daten hergestellt. D.h. man teilt der Datei die den Private-Graph enthält mit, wo sich der Protected-Graph befindet und diesem wiederum sagt man damit, wo er den Public-Graph finden kann. Die Reihenfolge hierbei ist wichtig, da man sonst die Sichtbarkeit von privaten Informationen riskiert, wenn sie nicht eingehalten werden sollte. Im folgenden Beispiel wird kurz die Relation auf den Public-Graph in der Datei, die den Protected-Graph beinhaltet, gezeigt:

```
…
<rdfs:seeAlso rdf:resource="http://localhost/rdf/john_public.rdf"/>
…
```

Die Private-Graph-Datei:

Sie beinhaltet neben den notwendigen Namensräumen die Adressen der Protected-Graph-Dateien der Freunde, Orte und Gegenstände, und sowohl die Namen als auch die zugeteilten Prioritäten der Freundesgruppen. Folgender Auszug aus der Private-Graph-Datei von John Grady soll einen kurzen Einblick liefern:

```
01 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
02 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
03 xmlns:sem="http://localhost/rdf/sem_cal.rdf#"
04 xmlns:v="http://www.w3.org/2006/vcard/ns#">

05 <sem:Person rdf:about="http://localhost/rdf/John">
06 <rdfs:seeAlso rdf:resource="http://localhost/rdf/john_protected.rdf"/>
07 <rdfs:seeAlso
rdf:resource="http://localhost/rdf/matthew_protected.rdf"/>
08 <rdfs:seeAlso
rdf:resource="http://localhost/rdf/peter_protected.rdf"/>
09 <rdfs:seeAlso rdf:resource="http://localhost/rdf/s30118.rdf"/>
10 ...
11 </sem:Person>

12 <rdf:Description rdf:about="Friendsgroup_1">
13 <sem:hasDescription>working colleagues</sem:hasDescription>
14 <sem:fpriority>3</sem:fpriority>
15 </rdf:Description>

16 <rdf:Description rdf:about="Friendsgroup_2">
17 <sem:hasDescription>soccer mates</sem:hasDescription>
18 <sem:fpriority>8</sem:fpriority>
19 </rdf:Description>
20 ...
21 </rdf:RDF>
```

In den Zeilen 06 – 09 werden die dem Benutzer bekannten Adressen der Protected-Graph-Dateien seiner Freunde angeführt. Die Zeilen 12 – 19 zeigen die Prioritäten der vom Benutzer festgelegten Freundesgruppen und deren Namen.

Die Protected-Graph-Datei:

Sie beinhaltet zusätzliche Informationen über die Person, die nur für Freunde gedacht sind (Telefonnummern und Adressen), die durch den Benutzer angelegten Regeln, sowie Informationen für die Anwendung. Dies sind Informationen zu Freundesgruppen, welche Freunde sich gemeinsam in welchen Freundesgruppen befinden, Kategorien des Besitzers und Update-Anweisungen für die Anwendung. Folgender Auszug aus John's Protected-Graph-Datei soll dabei diese Informationen verdeutlichen:

```
01 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
02 ...
03 <rdf:Description rdf:about="http://localhost/rdf/John">
04 <v:fax>+43 732 456 789</v:fax>
05 <v:tel>
06    <rdf:Description>
07    <rdf:value>+43 723 654 789</rdf:value>
08    <rdf:type rdf:resource="http://www.w3.org/2006/vcard/ns#Home"/>
09    </rdf:Description>
10 </v:tel>
11 <v:adr>
```

```
12    <rdf:Description>
13    <v:street-adress>Daemonstreet 47</v:street-adress>
14    <v:locality>Linz</v:locality>
15    <v:postal-code>4020</v:postal-code>
16    <v:country-name>Austria</v:country-name>
17    <rdf:type rdf:resource="http://www.w3.org/2006/vcard/ns#Home"/>
18    </rdf:Description>
19 </v:adr>
20 <rdfs:seeAlso rdf:resource="http://localhost/rdf/john_public.rdf"/>
21 </rdf:Description>

22 <sem:Friendgroup rdf:about="Friendsgroup_1">
23 <sem:hasFriendgroupOwner rdf:resource="http://localhost/rdf/John"/>
24 <sem:hasFriends>
25 <rdf:Bag rdf:about="bag_1">
26 <rdf:li>
27    <rdf:Description rdf:about="http://localhost/rdf/Peter"/>
28 </rdf:li>
29 <rdf:li>
30    <rdf:Description rdf:about="http://localhost/rdf/Matthew"/>
31 </rdf:li>
32 </rdf:Bag>
33 </sem:hasFriends>
34 </sem:Friendgroup>

35 <sem:rule rdf:about="http://localhost/rdf/john_public.rdf#rule12345">
36 <sem:ruleID>12345</sem:ruleID>
37 <sem:ruleDescription>CancelDueToLowerPriority</sem:ruleDescription>
38 <sem:ruleEvent>(ON) NEW EVENT</sem:ruleEvent>
39 <sem:ruleCondition>(IF) DTSTART (=) BUSY (AND) ORGANIZER.PRIORITY
(&lt;) BUSY.PRIORITY</sem:ruleCondition>
40 <sem:ruleAction>(DO) DECLINE APP</sem:ruleAction>
41 <sem:ruleOwner rdf:resource="http://localhost/rdf/John"/>
42 <sem:ruleStatus>active</sem:ruleStatus>
43 </sem:rule>

44 <sem:Category rdf:about="category_1">
45 <sem:hasCategoryOwner rdf:resource="http://localhost/rdf/John"/>
46 <sem:hasCategoryValue>private</sem:hasCategoryValue>
47 </sem:Category>
48 <sem:Category rdf:about="category_2">
49 <sem:hasCategoryOwner rdf:resource="http://localhost/rdf/John"/>
50 <sem:hasCategoryValue>business</sem:hasCategoryValue>
51 </sem:Category>

52 <sem:IDUpdateAnswer>
53 <sem:hasIDAnswerId>update_20140521-31_2</sem:hasIDAnswerId>
54 </sem:IDUpdateAnswer>
55 ...
56 </rdf:RDF>
```

In den Zeilen 03 – 21 werden die näheren Informationen zur Person dargestellt. Darrauffolgend wird in den Zeilen 22 – 34 eine Freundesgruppe dargestellt, welche den Besitzer John hat und die Freunde Peter und Matthew besitzt. Die Zeilen 35 – 43 repräsentieren eine von John festgelegte Regel und in den Zeilen 44 – 47 wird eine von ihm erstellte Kategorie dargestellt. Abschließend wird noch eine für die Anwendung relevante Nachricht dargestellt, die besagt, dass ein Update eines bestimmten Events durchgeführt wurde.

Für das Framework selbst spielt es keine Rolle ob eine Information in der Public-Graph-Datei oder in der Protected-Graph-Datei steht. D.h. sollte bei einer Erweiterung gewünscht sein, dass die Regeln in der Public-Graph-Datei stehen sollen, so ist das ohne Aufwand änderbar.

Die Public-Graph-Datei:

Sie beinhaltet die Informationen, die die Person zur Kontaktaufnahme veröffentlichen möchte (Name und E-Mail Adresse). Weiters beinhaltet sie den Kalender und alle Termine dieser Person. Zusätzlich werden noch anwendungsrelevante Anweisungen in der Public-Graph-Datei abgelegt, welche für das halb-automatische Verhalten bei Regelverletzungen verwendet werden. Folgendes Beispiel zeigt einen Ausschnitt aus John's Public-Graph-Datei:

```
01 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
02 ...
03 <rdf:Description rdf:about="http://localhost/rdf/John">
04 <v:fn>John Grady</v:fn>
05 <v:email>john@grady.exa</v:email>
06 </rdf:Description>

07<Vcalendar rdf:about="http://localhost/rdf/john_public.rdf#Calendar1">
08 <component>
09 <Vevent rdf:about="20140320-01">
10 <uid>20140320-01</uid>
11 <dtstamp
rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-03-
21T21:14:50</dtstamp>
12 <dtstart
rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-03-
21T09:00:00</dtstart>
13 <dtend rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-
03-21T17:00:00</dtend>
14 <location rdf:resource="http://localhost/rdf/s30118.rdf#room"/>
15 <description>Meeting with Matthew</description>
16 <categories rdf:resource="http://localhost/rdf/category_2"/>
17 <organizer rdf:resource="http://localhost/rdf/John"/>
18 <sem:Attendee>
19 <rdf:Description rdf:about="20140320-01_01">
20 <sem:attendeePerson rdf:resource="http://localhost/rdf/John"/>
21 <sem:attendeeStatus
rdf:resource="http://localhost/rdf/aStatus_cancelled"/>
22 </rdf:Description>
23 </sem:Attendee>
24 <sem:Attendee>
25 <rdf:Description rdf:about="20140320-01_02">
26 <sem:attendeePerson rdf:resource="http://localhost/rdf/Matthew"/>
27 <sem:attendeeStatus
rdf:resource="http://localhost/rdf/aStatus_declined"/>
28 </rdf:Description>
29 </sem:Attendee>
30 <rdf:resource="http://localhost/rdf/eStatus_cancelled"
31 <priority rdf:resource="http://localhost/rdf/ePriority_standard"/>
32 </Vevent>
33 </component>
34 </Vcalendar>
```

```
35 <sem:IDCheckAnswer>
36 <sem:hasIDAnswerCreator rdf:resource="http://localhost/rdf/John"/>
37 <sem:hasIDAnswerOwner rdf:resource="http://localhost/rdf/Peter"/>
38 <sem:hasIDAnswerUid>20140522-23</sem:hasIDAnswerUid>
39 <sem:hasIDAnswer>confirmed</sem:hasIDAnswer>
40 </sem:IDCheckAnswer>
41 ...
42 </rdf:RDF>
```

In den Zeilen 03 – 06 werden öffentliche Informationen des Benutzers dargestellt. Die Zeilen 09 – 32 stellen einen abgebrochenen Termin von John dar. Abschließend wird in den Zeilen 35 – 40 eine für die Anwendung relevante Überprüfung gezeigt, welche aussagt, dass beim nächsten Login von Peter ein bestimmter Teilnehmerstatus bei einem Termin geändert werden soll.

4.3. Terminwichtigkeit, Terminstatus und Teilnehmerstatus

Da die Terminwichtigkeiten und die möglichen Status von Terminen und Teilnehmern für alle Benutzer und Termine gleich sind, werden diese im selbst erstellten Vokabular fixiert. Die Terminwichtigkeiten sollen direkt bei jedem Termin hinterlegt werden, mit Ausnahme von `FreeBusyTime` – Terminen, welche, per Definition, keine Wichtigkeit besitzen (vgl. Kapitel 3.1.1). Die Terminwichtigkeit wird mit der Eigenschaft `priority` aus dem RDF Calendar Vokabular dargestellt und wird in RDF bei den Terminen als Resourcen (`standard`, `important` und `very important`) folgendermaßen verwendet:

```
01 ...
02 <Vevent rdf:about=”...”>
03  ...
04  <location rdf:resource="...">
05  ...
06  <priority rdf:resource="http://localhost/rdf/ePriority_standard"/>
07 </Vevent>
08 ...
```

Ebenfalls über Resourcen angeführt werden Termin- und Teilnehmerstatus, wie folgendes Beispiel mit den Zeilten 02 und 06 verdeutlichen soll:

```
01 ...
02 <status rdf:resource="http://localhost/rdf/eStatus_confirmed"/>
03 <sem:Attendee>
04    <rdf:Description rdf:about="20131104-02_01">
05    <sem:attendeePerson rdf:resource="http://localhost/rdf/Matthew"/>
06    <sem:attendeeStatus
rdf:resource="http://localhost/rdf/aStatus_confirmed"/>
07    </rdf:Description>
08 </sem:Attendee>
09 ...
```

4.4. Terminklassen

Die vorgestellten Terminklassen sollen als Kalenderkomponenten in RDF dargestellt werden. Die Eigenschaften der Komponenten wurden bereits in Kapitel 2.2.2.3 erläutert. Mit Ausnahme der Eigenschaft `Attendee` können alle Eigenschaften übernommen und eingesetzt werden. `Attendee` setzt in RDF Calendar jedoch nicht alle für diese Arbeit notwendigen Eigenschaften voraus. Deshalb wird `Attendee` um einen Status erweitert.

4.4.1. Vevent

Wie in Kapitel 3.1.1 bereits erläutert, stellt Vevent einen Termin im herkömmlichen Sinne dar, mit mehreren Teilnehmern, bestimmten Teilnehmerstatus und einer bestimmten Priorität. Die Umsetzung der Terminkomponente `Event` mit der abgewandelten Eigenschaft `Attendee` wird in folgendem Beispiel verdeutlicht:

```
01 ...
02 <component>
03 <Vevent rdf:about="...">
04 <uid>UID</uid>
05 <dtstamp rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
YYYY-MM-DDThh:mm:ss</dtstamp>
06 <dtstart rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
YYYY-MM-DDThh:mm:ss</dtstart>
07 <dtend rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
YYYY-MM-DDThh:mm:ss</dtend>
08 <location rdf:resource="..."/>
09 <description>VeventExample</description>
10 <categories rdf:resource ="..."/>
11 <organizer rdf:resource="..."/>
12 <sem:Attendee>
13 <rdf:Description rdf:about="...">
14 <sem:attendeePerson rdf:resource="..."/>
15 <sem:attendeeStatus rdf:about="..."/>
16 </rdf:Description>
17 </sem:Attendee>
18 <sem:Attendee>
19 <rdf:Description rdf:about="...">
20 <sem:attendeePerson rdf:resource="..."/>
21 <sem:attendeeStatus rdf:resource ="..."/>
22 </rdf:Description>
23 </sem:Attendee>
24 <status rdf:resource ="..."/>
25 <priority rdf:resource="..."/>
26 </Vevent>
27 </component>
28 ...
```

`Vevent` legt die Art der Terminklasse fest. UID dient zur eindeutigen Identifikation des Termins. `dtstamp`, `dtstart` und `dtend` sind Zeitangaben für den Erstellungszeitpunkt, den Startzeitpunkt und den Endzeitpunkt des Termins. `location` gibt an, wo der Termin stattfindet und soll wiederum eine eigene Resource und nicht bloß ein Literal sein. `description` in Zeile 09 gibt die

Beschreibung des Termins an, während categories die Kategorie festlegt. organizer legt den Ersteller des Termins fest und mit sem:Attendee werden von Zeile 12 bis 23 die um den Status erweiterten Teilnehmer dargestellt. sem:attendeePerson gibt dabei die Teilnehmerresource und sem:attendeeStatus deren Teilnehmerstatus an. Mit status wird der Terminstatus beschrieben und priority legt die Terminwichtigkeit fest. Die Terminkomponente Multievent ist einem mit Vevent ident, wird jedoch um die Eigenschaft related-to erweitert, welche angibt, mit welchen anderen möglichen Abhaltungen (UIDs) das jeweilige Vevent in Verbindung steht:

```
01 ...
02 <component>
03 <Vevent rdf:about="...">
04 ...
05 <related-to>UIDs</related-to>
06 </Vevent>
07 </component>
08 ...
```

4.4.2. Vtodo

Die Umsetzung der Terminkomponente Todo ist dem Event ähnlich, nur dass Todo nur einen Teilnehmer, nämlich den Organisator hat und der Endzeitpunkt nicht durch das Attribut dtend, sondern durch due ausgedrückt wird, wie in folgendem Beispiel zu sehen ist:

```
01 ...
02 <component>
03 <Vtodo rdf:about="...">
04 <uid>UID</uid>
05 <dtstamp rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
YYYY-YYYMM-DDThh:mm:ss</dtstamp>
06 <dtstart rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
YYYY-YYYMM-DDThh:mm:ss</dtstart>
07 <due rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">YYYY-MM-
08 DDThh:mm:ss</due>
09 <location rdf:resource="..."/>
10 <description>VtodoExample</description>
11 <categories rdf:resource="..."/>
12 <organizer rdf:resource="..."/>
13 <sem:Attendee>
14 <rdf:Description rdf:about="...">
15 <sem:attendeePerson rdf:resource="..."/>
16 <sem:attendeeStatus rdf:about="..."/>
17 </rdf:Description>
18 </sem:Attendee>
19 <status rdf:about="..."/>
20 <priority rdf:about="..."/>
21 </Vtodo>
22 </component>
23 ...
```

4.4.3. FreeBusyTime

Die Umsetzung der Komponente `FreeBusyTime` unterscheidet sich zu den eben erwähnten Terminklassen in der Hinsicht, dass sie keine Prioritäten, keinen Status und keine Kategorie besitzt und zusätzlich noch an keinem bestimmten Ort stattfindet. `FreeBusyTime` besitzt in RDF Calendar die Eigenschaft Teilnehmer. Da der Teilnehmer für diese Anwendung jedoch um einen Status erweitert werden soll, und `FreeBusyTime` keinen Status besitzt (vgl. Kapitel 3.1.3), wird der Teilnehmer weggelassen, wie in folgendem Beispiel gezeigt wird:

```
01 ...
02 <component>
03 <Vfreebusy rdf:about="...">
04 <uid>UID</uid>
05 <dtstamp rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
YYYY-MM-DDThh:mm:ss</dtstamp>
06 <dtstart rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
YYYY-MM-DDThh:mm:ss</dtstart>
07 <dtend rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
YYYY-MM-DDThh:mm:ss</dtend>
08 <description>VfreebusyExample</description>
09 <organizer rdf:resource="..."/>
10 </Vfreebusy>
11 </component>
12 ...
```

4.5. Prioritäten und Freundesgruppen

Die Prioritäten werden von der jeweiligen Person selbst an dessen Freundesgruppen vergeben und stellen private Information dar. Sie werden somit getrennt von den öffentlichen Daten im Private-Graph hinterlegt. In RDF Calendar gibt es keine vordefinierte Klasse für Freundesgruppen und auch keine Eigenschaft für deren Priorität. Es soll daher um genau diese Klasse und Priorität erweitert werden. Folgendes Beispiel soll die Erweiterung des Vokabulars demonstrieren, wobei die Information über die Priorität im Private-Graph abgebildet werden soll:

```
01 ...
02 <Friendgroup rdf:about="...">
03 <sem:hasFriendgroupOwner rdf:resource="..."/>
04 <sem:hasFriends>
05 <rdf:Bag rdf:about="...">
06 <rdf:li>
07   <rdf:Description rdf:resource="...Freundesresource..."/>
08 </rdf:li>
09 <rdf:li>
10   <rdf:Description rdf:resource="...Freundesresource..."/>
11 </rdf:li>
12 </rdf:Bag>
13 </sem:hasFriends>
14 <sem:hasDescription>Bezeichnung der
Freundesgruppe</sem:hasDescription>
15 <sem:fpriority>4</sem:fpriority>
```

```
16 </sem:Friendgroup>
17 ...
```

Die Freundesgruppe hat dabei eine eindeutige URI, einen Namen, einen Ersteller, die Freunde, welche sich in der Gruppe befinden und eine bestimmte, vom Benutzer selbst festgelegte Priorität von 1 bis 10.

4.6. Reaktives Verhalten

Wie in Kapitel 3.1.3 vorgegeben, werden Zustandsänderungen, wenn sie nicht direkt nach Bekanntwerden des Änderungswunsches umgesetzt werden können, (halb-)automatisch durchgeführt. Um dieses (halb-)automatische Handeln in RDF umzusetzen, werden zwei Mechanismen benötigt, die Mitteilung, dass ein bestimmter Termin überprüft werden soll, und die (halb-)automatisch darauf erstellte Antwort.

4.6.1. Reaktives Verhalten mit Regeln

Öffentliche und für Freunde sichtbare Regeln können bereits zum Zeitpunkt der Erstellung eines Termins überprüft werden und fließen somit in die Entscheidung, welche Teilnehmerstatus bei der Erstellung eingetragen werden sollen, mit ein. Kann jedoch ein Teil der Bedingung (z.B. Priorität) nicht sofort ermittelt werden, oder wünscht der jeweilige Teilnehmer eine Benachrichtigung des Termins, muss diese Benachrichtigung, zusätzlich zum Termin, auch in den Kalender eingetragen werden. Diese Benachrichtigung soll in RDF folgendermaßen umgesetzt werden:

```
01 <IDCheck rdf:about="...">
02   <hasIDCreator rdf:resource="..."/>
03   <hasIDOwner rdf:resource="..."/>
04   <hasIDToCheck>UID</hasIDToCheck>
05   <hasIDRuleId>RuleID</hasIDRuleId>
06 </IDCheck>
```

`hasIDCreator` bezeichnet dabei den Ersteller des Termins, während `hasIDOwner` jenen Teilnehmer bezeichnet, dessen Regel diese Benachrichtigung hervorgerufen hat. `UID` bezeichnet den zu überprüfenden Termin und `RuleID` jene Regel, welche die Benachrichtigung ausgelöst hat.

Wie in Kapitel 3.1.5.5 festgelegt, soll auf diese Benachrichtigung in einem zweiten Schritt, dem nächsten Login des zu benachrichtigenden Teilnehmers, geantwortet werden. Diese Antwort soll in RDF folgendermaßen umgesetzt werden:

```
01 <IDCheckAnswer rdf:about="...">
02   <hasIDAnswerCreator rdf:resource="..."/>
```

```
03   <hasIDAnswerOwner rdf:resource="..."/>
04   <hasIDAnswerUid>UID</hasIDAnswerUid>
05   <hasIDAnswer rdf:resource="..."/>
06 </IDCheckAnswer>
```

hasIDAnswerCreator bezeichnet dabei den Ersteller der Antwort, während mit hasIDAnswerOwner der Ersteller des Termins angeführt wird. UID ist der eindeutige Bezeichner des Termins und in hasIDAnswer wird die Antwort des Teilnehmers (confirmed oder declined als Resource) angeführt.

4.6.2. Reaktives Verhalten bei Statusänderungen von Teilnehmern

Ändert ein Teilnehmer den Teilnehmerstatus eines Termins den er selbst erstellt hat, kann das System diese Änderung sofort durchführen (vgl. Kapitel 3.1.3). Ist der Teilnehmer jedoch nicht gleichzeitig auch der Ersteller, erfolgt die Aktualisierung in zwei Schritten, der Benachrichtigung über die Änderung beim nächstmöglichen Zeitpunkt (Login des Terminorganisators) und der Aktualisierung des Termins selbst.

Die Benachrichtigung einer Zustandsänderung soll in RDF folgendermaßen abgebildet werden:

```
01 <IDUpdate rdf:about="...">
02   <hasIDUpdateID>updateID</hasIDUpdateID>
03   <hasIDUpdateUid>UID</hasIDUpdateUid>
04   <hasIDUpdateCreator rdf:resource="..."/>
05   <hasIDUpdateOwner rdf:resource="..."/>
06   <hasIDUpdateStatus rdf:resource="..."/>
07 </IDUpdate>
```

hasIDUpdateID ist ein eindeutiger Bezeichner und soll zur Kommunikation zwischen den Teilnehmern dienen. UID ist der Bezeichner des zu ändernden Termins. hasIDUpdateCreator bezeichnet den Teilnehmer, der die Änderung vornehmen lassen möchte und hasIDUpdateOwner ist der Ersteller des Termins. hasIDUpdateStatus beschreibt abschließend den Wert auf den der Terminstatus geändert werden soll. Beim nächsten Login des Terminorganisators wird die Benachrichtigung eingelesen und darauf reagiert.

4.7. Regelaufbau in RDF

Um eine generische Basis für die Regeln zu schaffen, dürfen diese nicht, für diesen speziellen Fall (einer Terminorganisation), in der Geschäftslogik ‚versteckt' sein, sondern sollen erweiterbar bzw. änderbar getrennt davon deklariert werden. Eine mögliche Umsetzung wäre es, die Regeln wie Trigger in einem aktiven Datenbanksystem direkt an das Datenmodell anzubinden. Diese Trigger werden somit unabhängig von der Repräsentation und Geschäftslogik ausgeführt.

Da aus den Anforderungen hervorgeht, dass das Resource Description Framework zu verwenden ist, müssen diese Regeln somit direkt in RDF abgebildet werden, anstatt sie in einem eigenen Format oder in einer eigenen Datei abzulegen. Die in Kapitel 3.1.5 genannten möglichen Ereignisse sollen wie Trigger in aktiven Datenbanksystemen bei insert-, update- oder delete-Ereignissen eingreifen und überprüft werden. Angelehnt an den von [PaPW06] vorgeschlagenen Ansatz RDFTL (vgl. Kapitel 2.3) wurde folgender Aufbau für eine ECA - Regel in RDF gewählt:

```
ON Terminevent
IF ASK SPARQL Query
DO Aktion
```

Das `Terminevent` kann dabei eines von mehreren möglichen Ereignissen (vgl. Kapitel 4.4) sein. Die `IF` Bedingung soll mittels einer speziellen SPARQL Query (vgl. Kapitel 2.2.4) feststellen, ob die `Aktion`, welche unter `DO` angeführt wird, ausgeführt werden soll. Die Aktion kann dabei entweder eine Statusänderung einer bestimmten Person bei einem Termin sein, oder eine Benachrichtigung an den Regelersteller (vgl. Kapitel 3.1.5.3).

Die vordefinierten Bausteine (vgl. Kapitel 3.1.5.2) der Regeln, welche vom Benutzer zusammengesetzt werden, müssen in gültige SPARQL Abfragen umgewandelt werden. Für dieses Umwandeln werden Details zum Datenmodell benötigt und soll daher vom Controller (vgl. Kapitel 5) durchgeführt werden.

Der eben vorgeschlagene Aufbau soll in RDF folgendermaßen umgesetzt werden:

```
01 <rule rdf:about="http://example.org/John#rule12350">
02    <ruleID>12345</ruleID>
03    <ruleDescription>ruleName</ruleDescription>
04    <ruleEvent> (ON)... </ruleEvent>
05    <ruleCondition> (IF) ... (AND) ... </ruleCondition>
06    <ruleAction> (DO) ... </ruleAction>
07    <ruleOwner rdf:resource="http://example.org/John"/>
08    <ruleStatus>active</ruleStatus>
09 </rule>
```

Die Regel hat dabei eine eindeutige URI und einen eindeutigen Bezeichner `ruleID`. Neben einer Beschreibung, `ruleDescription`, werden die drei Teile einer ECA-Regel, das Ereignis, `ruleEvent`, die Bedingung, `ruleCondition`, und die Aktion, `ruleAction`, angeführt. Abschließend wird noch ein Besitzer dieser Regel, `ruleOwner`, und ein aktueller Status der Regel, `ruleStatus`, ob sie aktiv oder inaktiv ist, angeführt.

4.8. Umzusetzende Regeln

Die in Kapitel 3.1.5.4 vorgestellten Regeln sollen nun mit den vorgestellten Bausteinen modelliert werden. Dabei soll die Darstellung der Regeln auf die in Kapitel 4.7 vorgestellten Bereiche `ruleEvent`, `ruleCondition` und `ruleAction` reduziert werden.

1. Regel: Ich möchte über alle neu erstellten Termine, die die höchste Wichtigkeitskategorie besitzen, informiert werden, egal ob ich bereits einen Termin zu diesem Zeitpunkt habe oder nicht.

Das `ruleEvent` bezieht sich bei dieser Regel auf alle neu erstellten Termine (zu `FreeBusyTime` Zeiträumen/Terminen kann ich nicht eingeladen werden):

```
<ruleEvent>(ON) NEW EVENT (OR) NEW TODO</ruleEvent>
```

Die Bedingung wird mit folgenden Bausteinen überprüft, wobei erwähnt werden soll, dass die Notation eines Vergleichsoperators in RDF nicht mit „<“ oder „>“ angegeben werden können, sondern diese an die HTML Darstellung angepasst werden müssen:

```
<ruleCondition>(IF) IMPORTANCE (&gt;) IMPORTANT</ruleCondition>
```

Die Aktion soll eine Benachrichtigung des Teilnehmers über das Event sein:

```
<ruleAction>(DO) NOTIFY APP</ruleAction>
```

2. Regel: Bei einer Terminkollision bei der Erstellung eines neuen Termins bei dem ich Teilnehmer bin, soll überprüft werden, ob die Priorität des Erstellers höher ist als die Priorität des Erstellers des bereits fixierten Termins. Sollte die Priorität nicht höher sein, soll der neue Termin automatisch abgelehnt werden.

Das `ruleEvent` bezieht sich auf alle neu erstellen Termine:

```
<ruleEvent>(ON) NEW EVENT</ruleEvent>
```

Die Bedingung wird mit folgenden Bausteinen überprüft:

```
<ruleCondition>(IF) DTSTART (AND) DTEND (=) BUSY (AND) ORGANIZER.PRIORITY
(&lt;) BUSY.PRIORITY</ruleCondition>
```

Die Aktion soll das Ablehnen des Events sein:

```
<ruleAction>(DO) DECLINE APP</ruleAction>
```

3. Regel: Wird ein Termin einer bestimmten Terminkategorie aktualisiert, soll überprüft werden, ob eine bestimmte Person zu diesem Termin zugesagt hat. Hat sie zugesagt, soll mein eigener Teilnehmerstatus bei diesem Termin auf abgesagt geändert werden.

Das `ruleEvent` bezieht sich bei dieser Regel auf Aktualisierungen von Terminen:

```
<ruleEvent>(ON) UPDATE EVENT</ruleEvent>
```

Die Bedingung wird mit folgenden Bausteinen dargestellt:

```
<ruleCondition>(IF) CATEGORY (IS) CategoryResource (AND) ATTSTATUS (OF)
AttendeeResource (TO) CONFIRMED </ruleCondition>
```

Die Aktion soll das das Ablehnen des Termins sein:

```
<ruleAction>(DO) DECLINE APP</ruleAction>
```

4. Regel: Wird ein neuer Termin einer bestimmten Terminkategorie bei dem ich als Teilnehmer eingeladen werde erstellt, soll überprüft werden, ob der Ersteller eine gewisse Priorität überschreitet und ob ich keinen weiteren Termin in dieser Zeit habe. Trifft dies zu, soll der Termin automatisch zugesagt werden.

Das `ruleEvent` bezieht sich auf alle neu erstellen Termine:

```
<ruleEvent>(ON) NEW EVENT</ruleEvent>
```

Die Bedingung wird folgendermaßen dargestellt:

```
<ruleCondition>(IF) DTSTART (AND) DTEND (!=) BUSY (AND) CATEGORY (IS)
CategoryResource (AND) ORGANIZER.PRIORITY (&gt;) 8</ruleCondition>
```

Die Aktion soll das automatische Zusagen des Termins sein:

```
<ruleAction>(DO) CONFIRM APP</ruleAction>
```

5. Design

Im folgenden Kapitel werden die einzelnen Teilsysteme der Architektur näher erläutert.

Um den in Kapitel 1.1 angeführten Anforderungen, welche Einfluss auf die Architektur haben, gerecht zu werden, müssen mehrere Design-Entscheidungen getroffen werden. Diese Anforderungen waren:

- die Anwendung soll Web-basierend sein
- es sollen verteilte, semantische Daten verwendet werden
- es soll das Resource Description Framework (RDF) verwendet werden
- es soll versucht werden, die Anwendung generisch zu halten, dass man, mit Ausnahme der Repräsentation, sie auch für andere mögliche Anwendungsbereiche (z.B. eine Reiseveranstaltungsplanung) verwenden kann

Durch die Anforderung, dass die Anwendung Web-basierend sein soll, ergibt sich die erste Komponente, das `web interface` (vgl. Abbildung 7). Es dient als Kommunikationsschnittstelle zwischen dem Benutzer der am Client sitzt und der Geschäftslogik, welche am Server liegt. Auf der Serverseite wird dadurch ein Architekturbaustein benötigt, der die Anfragen des Clients verarbeiten kann, welche in Abbildung 7 durch die `view` repräsentiert wird. Die Planung der einzelnen Teilsysteme soll dabei an das Model-View-Controller Prinzip angelehnt werden, um ein Austauschen einzelner Komponenten zu ermöglichen und so eine Erweiterbarkeit für andere Anwendungen gegeben ist (wie in den Anforderungen vorgegeben). Die serverseitige Geschäftslogik soll durch den `controller` bereitgestellt werden, der ein RDF Framework zur Manipulation der RDF Daten verwendet. Die Daten sollen semantischer Natur sein und verteilt im Internet auf einem vom Benutzer selbst gewählten Webserver liegen und in RDF dargestellt sein. Dies spiegelt die Komponente RDF in Abbildung 7 wieder. Die Kommunikation zwischen `controller` und den verteilten RDF Daten wird über das HTTP bereitgestellt. Der Schreibzugriff im Speziellen soll mit der HTTP Request Methode HTTP PUT durchgeführt werden. Die letzte Komponente ist das `model`, welches das Datenmodell darstellt und durch den `controller` erstellt und manipuliert wird.

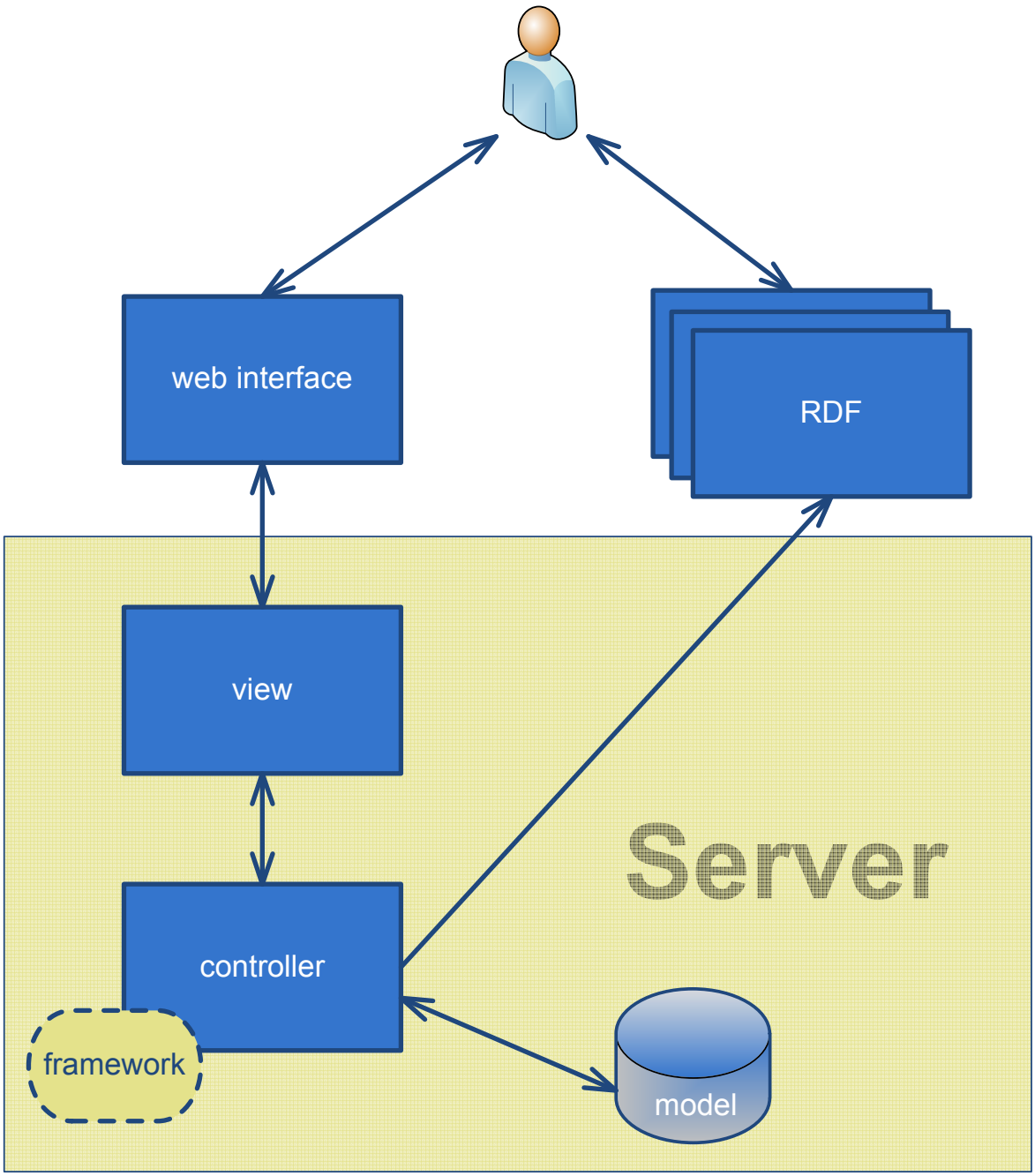


Abbildung 7 Architektur der Umsetzung

6. Implementierung

Das folgende Kapitel behandelt die Implementierung und alle dazu benötigten Komponenten der Anwendung. Zu Beginn wird die Auswahl der verwendeten Technologien begründet und der Auswahlprozess näher beschrieben. Anschließend wird das notwendige Grundgerüst zur Umsetzung der Anwendung näher erläutert. Dabei wird auf die Änderung des JENA Frameworks und auf die notwendigen Änderungen des verwendeten Webservers eingegangen. Im Anschluss daran wird die konkrete Umsetzung der Anwendung in Java mit JENA detailliert beschrieben.

6.1. Auswahl der zu verwendenden Technologien

In diesem Kapitel soll kurz erklärt werden, warum genau die ausgewählten Technologien verwendet werden. Zur Auswahl standen mehrere unterschiedliche RDF Vokabulare, RDF Frameworks, Visualisierungsmöglichkeiten und zwei mögliche Schreibverfahren auf entfernten Webservern.

6.1.1. Vokabulare

Wie in Kapitel 2.2.2.3 bereits erwähnt, ist RDF Calendar an den Standard iCalendar angelehnt und verspricht damit ein standardisiertes Format, welches auch von anderen Anwendungen akzeptiert wird und die Erweiterbarkeit der Anwendung um zusätzliche Funktionen erlaubt. Eine mögliche Alternative dazu war das Vokabular hCalendar, welches auch an iCalendar angelehnt ist, und das Darstellen von Terminen im Web ermöglicht. Es wird in HTML/HTML5 eingebettet und unterstützt eine XML Notation. Da jedoch RDF als Technologie in dieser Arbeit vorgegeben wurde, und hCalendar über einen Parser zwar auch wieder in RDF Calendar umgewandelt werden kann, jedoch dies einen Zwischenschritt bedeutet, viel die Wahl auf RDF Calendar.

Für die Beschreibung der Ressourcen bzw. die Eigenschaften der Ressourcen gab es mehrere mögliche Vokabulare zur Auswahl. Zum einen das Vokabular FOAF, welches nicht nur die Möglichkeit bietet Ressourcen zu beschreiben, sondern auch Beziehungen zwischen diesen Ressourcen herzustellen. Zweite Möglichkeit zur Darstellung der Eigenschaften ist das Vokabular vCard, welches keine zusätzlichen Beziehungen abbilden kann, und ähnlich wie das Mikroformat hCalendar gibt es auch das Vokabular hCard, welches eingebettet in HTML Code Informationen über eine Resource angibt. Da von diesem Vokabular im Grunde nur die Beschreibung von Namen, Adressen und Telefonnummern erwartet wird, reicht hier hCard oder vCard aus. Da mit hCard ein zusätzlicher Mappingschritt benötigt wird, soll vCard verwendet werden. Für die in

dieser Arbeit zu erstellende Anwendung würde man mit der Verwendung eines so mächtigen Vokabulars wie FOAF etwas über das Ziel hinaus schießen. Sollte man die Anwendung um komplexe Freundesbeziehungen erweitern wollen, kann man das Vokabular jedoch jederzeit einbauen. Als Alternative dazu soll an dieser Stelle noch das XFN[9] (XHTML Friends Network) erwähnt werden, welches auch die Möglichkeit bietet, Beziehungen zwischen Freunden darzustellen.

6.1.2. RDF Framework

Es gibt mehrere mögliche vorhandene Frameworks, welche RDF unterstützen. Bei der Auswahl muss auf Erweiterbarkeit geachtet werden (z.B. OWL) und es soll versucht werden, eine generische Lösung der Logik zu ermöglichen um das System auch für andere Zwecke (als eine Terminorganisation) einsetzen zu können.

Im Folgenden sollen mehrere RDF Frameworks miteinander auf die Punkte Aktualität, mögliche Erweiterbarkeit, und ob die Datenspeicherung im eigenen oder in einem weiteren Framework stattfindet, verglichen und anschließend beurteilt werden. Das Kriterium Erweiterbarkeit soll die Anforderung, dass die Anwendung auch für andere Bereiche als eine Terminorganisation eingesetzt werden können soll (vgl. Kapitel 1.1), bedienen. Das Kriterium der Datenspeicherung im Framework wurde durch den in Kapitel 4.7 erwähnten Grund, dass Regeln direkt an das Datenmodell gebunden werden sollen, notwendig.

Zwei weitere Kriterien sollen der Vollständigkeit halber angeführt werden, welche jedoch eine geringere Rolle in der Auswahl des Frameworks spielen. Diese zwei Kategorien sind die Unterstützung von SPARQL und ob es bereits eine Erweiterung für eine Termin- oder Kalenderorganisation gibt oder nicht.

In Tabelle 11, wird der Vergleich aufgeführt. Ein ‚+' bedeutet, dass das jeweilige Kriterium erfüllt wurde, ein ‚-' bedeutet, das Kriterium wird nicht erfüllt. Ein ‚?' bedeutet, dass keine Information zu diesem Kriterium und diesem Framework gefunden werden konnte.

Framework	aktiv	Datenspeicherung im Framework	Erweiterbarkeit für OWL	Kalendererweiterung	SPARQL
JRDF	-	?	-	-	+
AJAR/Tabulator	-	?	+	+	+
Apache Jena	+	+	+	-	+
OpenRDF Sesame	+	-	?	?	+

[9] http://www.gmpg.org/xfn/.

CubicWeb	+	+	+	?	-
Graphite	+	-	?	?	+
EasyRdf	+	-	+	?	+
SemWeb	-	-	?	?	+
dotNetRDF	+	-	+	-	+
RDF.rb	?	?	+	?	-
Mulgara	+	-	?	?	+

Tabelle 11 Ein Vergleich von RDF Frameworks

Es erfüllen mehrere RDF Frameworks die Kriterien, dass sie noch aktiv entwickelt und unterstützt werden und eine OWL Erweiterung ermöglichen. Viele dieser Frameworks verwenden jedoch eine weitere Schnittstelle zur Datenspeicherung, welche die Umsetzung der Regeln direkt am Datenmodell erschweren würde. Weiters soll in Betracht gezogen werden, dass für die Kommunikation über HTTP mit HTTP PUT ein geeignetes Framework oder Bibliotheken existieren sollten. Aus der Sicht der Popularität[10] überragt Apache Jena seine Konkurrenten und ist gleichzeitig ein serverseitiges state-of-the-art Framework, welches Bilbiotheken zur Erweiterung (in Bezug auf HTTP PUT) zur Verfügung stellt. Aus diesem Grund wurde das Framework Apache Jena für diese Arbeit gewählt.

6.1.3. View

Da mit der Wahl des JENA Frameworks ein JAVA Framework ausgewählt wurde, kann die Auswahl der Umsetzung der View auf einige mögliche Darstellungen eingegrenzt werden. Da die Anwendung nur ein Prototyp zur Darstellung der Funktionalität der Terminorganisation werden soll, wird daher kein besonderer Wert auf Usability, Look and Feel, etc. gelegt.

Um die Erweiterbarkeit (z.B. um Sessionhandling) zu gewährleisten, soll jedoch eine Trennung vom Loginbereich und der eigentlichen Anwendung umgesetzt werden. Der Loginbereich soll deshalb als Java Server Page getrennt von der Anwendung, welche als Java Servlet dargestellt werden soll, umgesetzt werden.

6.1.4. Hypertext Transfer Protocol

In Kapitel 5 wurde für den Schreibzugriff über HTTP auf die RDF Dateien die Anforderung gestellt, dass dies mit HTTP PUT passiert. Um diesen Dienst zu unterstützen, muss der Webserver auf dem geschrieben werden soll, um ein Script erweitert werden, welches die HTTP PUT Anfragen verarbeitet. Leichter für den Benutzer, welcher den Webserver konfigurieren muss, ist es jedoch,

[10] http://db-engines.com/en/ranking/rdf+store.

die Alternative WebDAV zu verwenden, da mit den Konfigurationseinstellungen, die dem Anhang entnommen werden können, sowohl das erstmalige Konfigurieren als auch die Wartung klarer ist. Ein weiterer Vorteil ist, dass es WebDAV-Frameworks für Java gibt, die im Controller direkt eingebaut werden können, wie z.B. Sardine[11]. In dieser Arbeit soll deshalb genau dieses Framework zum Einsatz kommen. Sardine ist ein kompaktes Framework zum Lesen, Schreiben und Erstellen von Dateien auf entfernten Webservern, und stellt auch (nur) Methoden für diese Operationen zur Verfügung.

6.2. Grundgerüst für die Umsetzung der Anwendung

In diesem Kapitel soll das Grundgerüst für die Umsetzung der Anwendung in Java mit dem JENA Framework geschaffen werden. Der Controller verwendet die Datenkomponente RDF, das JENA Framework zur Erstellung des Datenmodells, WebDAV zur Kommunikation mit den jeweiligen Webservern für Schreibzugriffe und ein Servlet zur Kommunikation mit dem Client. Diese Teilbereiche sollen nun näher beschrieben werden.

6.2.1. Architektur und Komponenten

Um die ausgewählten Komponenten zu verdeutlichen, soll die Architektur zur Implementierung der Anwendung nochmals, aktualisiert, dargestellt werden (vgl. Abbildung 8).

[11] https://github.com/lookfirst/sardine.

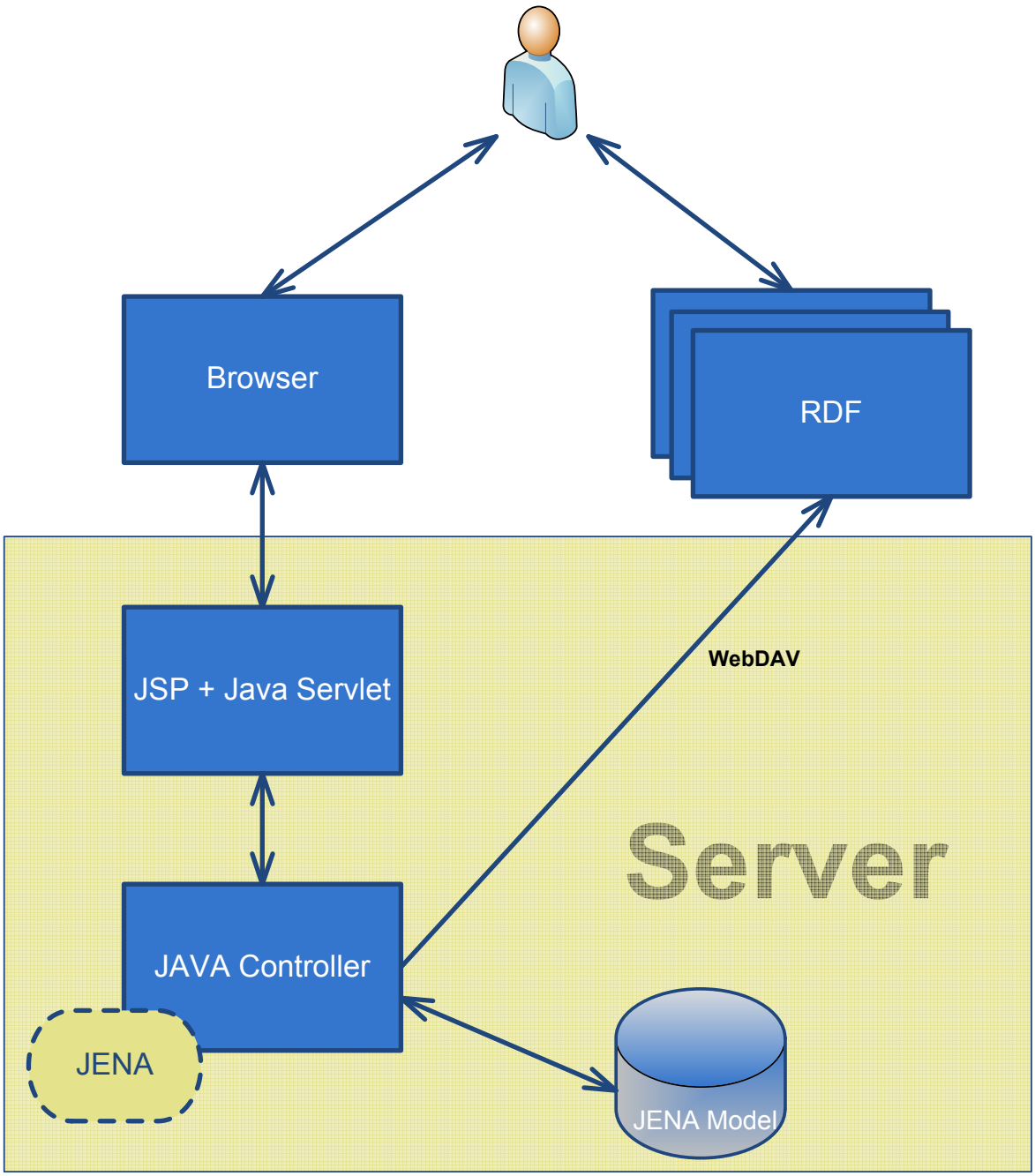


Abbildung 8 Konkretisierte Anwendungsarchitektur

Umzusetzen sind somit der Loginbereich mittels einer Java Server Page und das Java Servlet, welches die Eingaben des Benutzers annimmt und mit dem Controller kommuniziert. Weiters muss der Controller selbst (in Java), welcher mit dem JENA Framework die Daten einliest, das JENA Model erstellt und die Daten damit manipulieren kann, erstellt werden.

6.2.2. Klassenübersicht

Wie in Kapitel 4.7 bereits erwähnt, sollen die ECA-Regeln wie Trigger in einer relationalen Datenbank direkt beim Datenmodell umgesetzt werden. Daraus ergibt sich eine Erweiterung der

Funktionalität des JENA Frameworks um Trigger. Folgende Klassenaufteilung soll diese Erweiterung umsetzen (vgl. Abbildung 9):

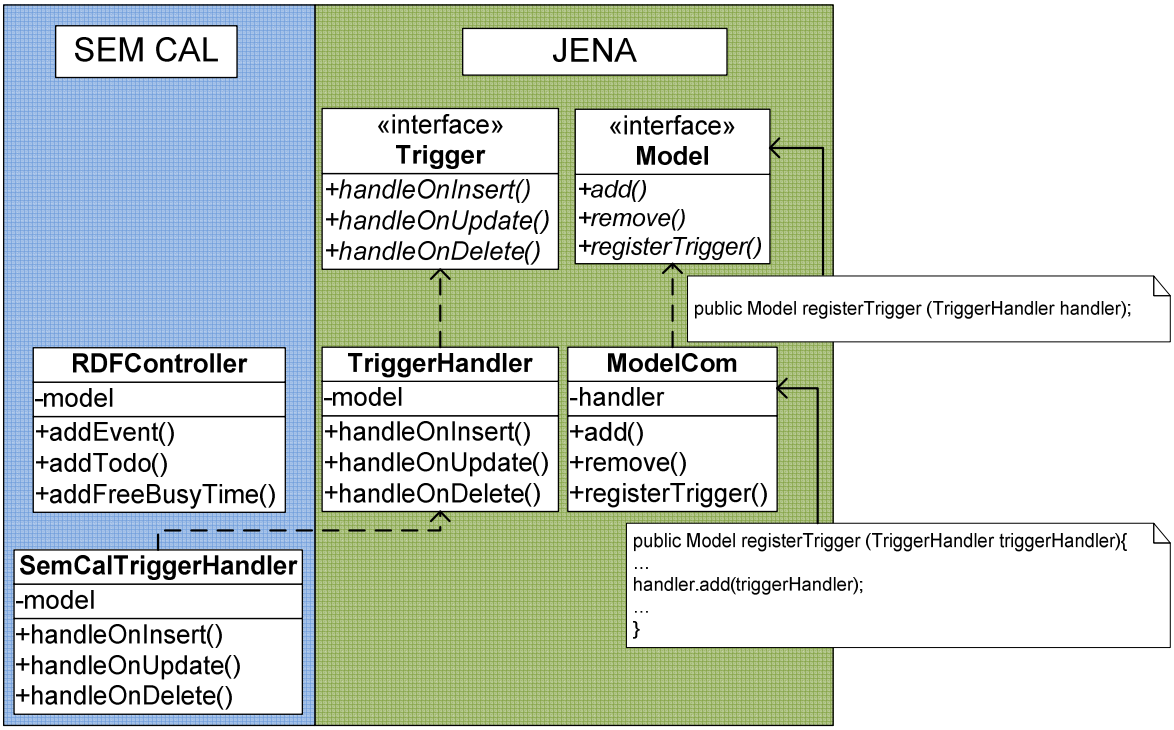


Abbildung 9 Klassenübersicht zur Erweiterung von JENA

In JENA existiert ein Interface *`Model`*, welches Methoden zum Manipulieren von RDF Daten im Datenmodell zur Verfügung stellt. Die Klasse `ModelCom` stellt die Umsetzung bzw. Realisierung dieses Interfaces innerhalb von JENA dar. Zur Umsetzung der Regeln wurde jetzt das JENA Framework um ein Interface *`Trigger`*, welches ein generisches Konzept für Schreibereignisse einer Regel darstellen soll, und um die Klasse `TriggerHandler`, welches die Umsetzung dieses Konzepts sein soll, erweitert. *`Trigger`* enthält drei Methoden zur Repräsentation von Schreibereignissen in einer Datenbank, nämlich insert, update und delete. `TriggerHandler` stellt dabei ein generisches Konzept dar, welches nicht nur für eine Terminorganisation, sondern auch für andere Zwecke verwendet werden kann. Weiters wurde das Interface *`Model`* um eine Methode zum Registrieren von `Triggern` und die Methode `ModelCom` um eine Liste von möglichen `TriggerHandlern` und einer Methode zum Registrieren dieser `TriggerHandler` erweitert.

In der Anwendung Sem Cal wird eine Klasse `SemCalTriggerHandler` von `TriggerHandler` abgeleitet und für den speziellen Fall einer Terminorganisation umgesetzt. Die Methoden `handleOnInsert()`, `handleOnUpdate()` und `handleOnDelete()` verwenden dabei die vom Benutzer festgelegten Regeln zur Überprüfung und zur Entscheidung der Aktion. Wird im Controller nun eine Einfügeoperation in das `Model` durchgeführt (dabei wird die Methode `add()`

der Klasse ModelCom aufgerufen), wird der TriggerHandler SemCalTriggerHandler registriert und in der add() Methode der ModelCom die Methode handleOnInsert() aller registrierten TriggerHandler ausgeführt (somit auch die Methode handleOnInsert() des SemCalTriggerHandlers).

6.3. Umsetzung der Anwendung in Java mit JENA

Die Umsetzung der Anwendung soll im folgenden Kapitel näher erläutert werden. Dabei werden zu Beginn die Grundfunktionen näher beschrieben und anschließend werden die vorgegebenen Regeln näher betrachtet. Darauf folgt ein kurzer Einblick in Besonderheiten die diese Art von Implementierung mit sich bringt und abschließend wird in einem Benutzerhandbuch die grafische Oberfläche dargestellt.

6.3.1. Grundfunktionen

Der Benutzer loggt sich mit der Adresse seiner privaten RDF Datei ein und übergibt dem System die WebDAV Authentifizierungsparameter. Die Anwendung versucht im nächsten Schritt von der eingegebenen URI einzulesen. Es werden alle in dieser privaten RDF Datei gefundenen seeAlso-Verknüpfungen verfolgt und die gefundenen Dateien rekursiv abgearbeitet und alle darin befindlichen RDF Daten in ein JENA Datenmodell Model geladen. Das Model stellt nun die eigentliche Datenquelle dar. Zusätzlich dazu wird im gleichen Schritt ein namedModel erstellt, welches auch die physikalischen Dateinamen zu jeder eingelesenen Zeile speichert, welches für das Zurückschreiben in die jeweiligen Dateien notwendig ist. Nach dem erfolgreichen Einlesen und Erstellen des Datenmodells, ermittelt das System zu verarbeitende Änderungen seit dem letzten Login. Das können Teilnehmerstatusänderungen zu Terminen sein, bei denen der Benutzer Organisator ist, oder zusätzliche Überprüfungen ob Regeln des Benutzers beim Erstellen von Terminen anderer Personen verletzt wurden. Anschließend werden Termine die dem Benutzer gesondert angezeigt werden sollen, weil sie seine Aufmerksamkeit oder seine Entscheidung benötigen, markiert. In einem letzten Schritt vor dem Anzeigen der Termine für den Benutzer, werden nicht mehr gebrauchte Aktualisierungs- und Regelbenachrichtigungen in den Dateien des Benutzers entfernt.

Anschließend werden die aktuellen Termine des Benutzers angezeigt. Dabei wird in der Übersicht zwischen bestätigten Terminen, noch nicht bestätigten Terminen und abgesagten Terminen unterschieden, wobei Multievents farblich als zusammengehörend dargestellt werden. Termine, welche die Aufmerksamkeit des Benutzers benötigen, werden gesondert in einer eigenen Tabelle dargestellt.

6.3.1.1. Hinzufügen eines neuen Termins

Im Controller werden für das Hinzufügen der unterschiedlichen Terminklassen separate Methoden zur Verfügung gestellt. Stellvertretend für alle Hinzufügemethoden soll die Methode `addEvent(...)` beschrieben werden:

```
public boolean addEvent(String description, String dtstart, String dtend,
String location, String priority, String category, String[]
attendees){...}
```

Nach der Erstellung der einzufügenden Statements und der einzufügenden RDF/XML Darstellung, wird die Einfügeoperation vorgenommen. Dabei wird zuvor der `SemCalTriggerHandler` mit der Methode `registerTrigger(TriggerHandler)` in der Klasse `ModelCom` registriert und anschließend die `add()` Methode der Klasse `ModelCom` aufgerufen. Dort findet die Überprüfung der Regeln statt und es kommt eine Antwort zurück, die besagt, ob der neue Termin eingefügt werden konnte oder nicht. Anschließend wird überprüft, ob beim Einfügen Regeln verletzt wurden und ob es notwendig ist, zusätzliche Einträge (für die Überprüfung des Termins durch andere Teilnehmer) einzufügen.

Nach der Ermittlung der URI auf der sich die Datei befindet in die geschrieben werden soll, wird mit der Methode `SardineFactory.begin(...)` eine Verbindung zum Webserver, auf dem sich die RDF Dateien befinden, aufgebaut und der Inhalt der aktuellen RDF Datei gelesen. Darauffolgend wird eine neue Datei, am Webserver der Anwendung, mit dem um den neuen Termin und den zusätzlichen Einträgen erweiterten RDF/XML Inhalt angelegt und anschließend mit der Methode `Sardine.put(...)` die gesamte Datei auf den Webserver, auf dem die alte Datei liegt, kopiert und die alte Datei damit überschrieben.

Die zusätzlichen Einträge können Teilnehmerzustandsänderungen für andere Teilnehmer sein oder eine Benachrichtigung für das gesonderte Anzeigen des Termins bei einem Teilnehmer. Diese zusätzlichen Einträge werden sowohl durch System- als auch durch private Regeln ausgelöst.

6.3.1.2. Manipulation von Kategorien, Freundesgruppen und Prioritäten von Freunden

Der Controller stellt Methoden zum Hinzufügen von Freunden, Erstellen von neuen Freundesgruppen, dem Anzeigen von Informationen über Freunde, das Hinzufügen von Freunden in bestimmte Freundesgruppen, zum Ändern von Prioritäten von Freundesgruppen, Erstellen von neuen Kategorien und dem Ändern von Kategorien zur Verfügung.

Der Aufbau dieser Methoden ist sehr ähnlich. Stellvertretend für diese Methoden soll hier die Methode `addFriendGroup(...)` beschrieben werden, da diese eine Besonderheit besitzt, da sie im Vergleich zu den anderen Methoden in zwei unterschiedliche Dateien schreiben muss.

```
public boolean addFriendGroup(String friendGroupToAdd, String
priorityOfGroup){...}
```

Die Priorität einer Freundesgruppe ist eine private Information und steht in der Datei, die den Private-Graph darstellt, während hingegen die Information, welche Freunde sich in welcher Gruppe befinden, in der Datei mit dem Protected-Graph stehen.

Nach der Ermittlung der physikalischen Adressen der beiden Dateien werden die einzufügenden RDF/XML Darstellungen für beide Dateien erstellt. Die Darstellung für den Private-Graph enthält dabei den Namen und die Priorität, während die Darstellung für den Protected-Graph nur das Gerüst der Freundesgruppe ohne Freunde beinhaltet (da noch keine Freunde in diese Gruppe eingefügt wurden). Anschließend werden die Dateien auf dem lokalen Webserver erstellt und auf dem entfernten Webserver überschrieben.

6.3.1.3. Antworten auf Einladungen

Es werden Methoden zum Zusagen (`confirm`), Absagen (`decline`) und Abbrechen (`cancel`) von Terminen vom Controller zur Verfügung gestellt. Es wird dabei zwischen der Teilnehmerstatusänderung durch den Organisator eines Termins und der Teilnehmerstatusänderung eines anderen Teilnehmers unterschieden, da die Änderung durch einen Organisator vom System sofort durchgeführt werden kann, während bei einem anderen Teilnehmer eine Benachrichtigung hinterlassen werden muss, dass die Änderung später in die RDF Datei des Organisators geschrieben werden muss. Die Methoden `confirmEvent(...)`, `declineEvent(...)` und `cancelEvent(...)` rufen dabei eine von zwei Methoden auf, welche sowohl `confirm`- und `decline`- als auch `cancel`- Änderungen vornehmen können. Zum Einen ist dies die Methode `updateStatusForSpecificEventByOrganizer(...)`…

```
updateAttendeeStatusForSpecificEventByOrganizer(String loggedInPerson,
String uid, String status) throws IOException {...}
```

… und zum Anderen die Methode `updateStatusForSpecificEventByAttendee(...)`:

```
updateAttendeeStatusForSpecificEventByAttendee(String loggedInPerson,
String uid, String status) throws IOException {...}
```

Beiden Methoden wird die zu ändernde Termin-UID und der zu ändernde Status mitgeteilt. Die Methode `updateAttendeeStatusForSpecificEventByAttendee(...)` ermittelt zunächst die Adresse jener Datei in die geschrieben werden soll, manipuliert den existierenden Inhalt

dahingehend, dass ein `IDUpdate` (vgl. Kapitel 4.6.2) hinzugefügt wird, welches später vom Organisator des Termins behandelt werden soll und überschreibt anschließend die alte Datei.

Die Methode `updateAttendeeStatusForSpecificEventByOrganizer(...)` hingegen registriert nach der Ermittlung der zu Schreibenden physikalischen Adresse den `SemCalTriggerHandler` und führt anschließend die Aktualisierung des Teilnehmerstatus durch. Dabei wird die Methode `handleOnUpdate(...)` im `SemCalTriggerHandler` ausgelöst und sämtliche Regeln für alle Teilnehmer dieses Termins überprüft. Sollten durch dieses Update zusätzliche Einträge erforderlich sein (Benachrichtigungen anderer Teilnehmer oder Statusänderungen), werden diese erstellt und gemeinsam mit der Teilnehmerstatusaktualisierung in die zu überschreibende Datei geschrieben.

6.3.1.4. Terminstatusänderungen

Für das Fixieren und Abbrechen eines Termins wird vom Controller die Methode `updateEventStatusForSpecificEventByOrganizer(...)` zur Verfügung gestellt, welche ausschließlich vom Organisator aufgerufen werden kann. Der Organisator kann mit dieser Methode den von ihm erstellten Termin fixieren und abbrechen.

Möchte ein Teilnehmer einen bereits fixierten Termin absagen, so muss er das über seinen Teilnehmerstatus ausdrücken und die eben beschriebene Methode `updateAttendeeStatusForSpecificEventByAttendee(...)` mit dem Status `cancel` aufrufen. Beim nächsten Login des Organisators wird ihm der Termin mit dem abgebrochenen Status gesondert angezeigt und er muss den Termin selbst absagen. Die Methode für den Organisator soll nun kurz beschrieben werden:

```
updateEventStatusForSpecificEventByOrganizer(String loggedInPerson,
String uid, String status) throws IOException
```

Nach der Ermittlung der physikalischen Adresse der zu bearbeitenden Datei, wird der Inhalt mit dem neuen Eventsstatus manipuliert und nach Erstellen der Datei am Webserver der Anwendung wird die Datei am entfernten Webserver überschrieben.

Eine Ausnahme hierbei stellt ein `Multievent` dar. Wird eine mögliche Abhaltung vom Organisator fixiert, werden alle anderen möglichen Abhaltungen automatisch abgebrochen.

6.3.2. Reaktives Verhalten

Wie und wann die Anwendung auf Ereignisse reagieren soll, wurde mit System- und benutzerdefinierten Regeln festgelegt. Im Folgenden soll nun die Umsetzung dieser Regeln in Java näher erläutert werden.

6.3.2.1. Systemregeln

Die Systemregeln sind speziell nur für die Terminorganisation gültig und werden somit im Controller bei eintretenden Events automatisch überprüft. In Kapitel 3.1.5.4 wurden die umzusetzenden Regeln bereits festgelegt. Es soll nun kurz beschrieben werden, wie diese im Controller umgesetzt wurden.

Beim Absagen eines Termins durch einen Teilnehmer soll überprüft werden, ob alle anderen Teilnehmer ebenfalls bereits abgesagt haben. Ist dies der Fall, soll der Termin abgebrochen werden. Um diese Regel umzusetzen, wird die Methode `updateAttendeeStatusOfSpecificEventByOrganizer(...)` nach dem Ermitteln der Adresse der physikalischen Datei und dem Auslesen dessen Inhalts, dahingehend modifiziert, dass sie zusätzlich alle Teilnehmerstatus (mit Ausnahme den des Organisators) überprüft. Sind diese Status alle auf abgelehnt oder abgebrochen, wird der Termin automatisch abgebrochen.

Die nächste Systemregel besagt, dass man über abgebrochene Termine informiert werden soll, da man fixierte nicht regelmäßig auf deren Teilnehmerstatus überprüft. Um diese Regel umzusetzen, wird beim Erstellen der für den Benutzer angezeigten Termine (`displayEvents(...)`) in der Methode `getActiveVevents(...)` eine separate Tabelle für zu benachrichtigende Termine kreiert. Nach dem Einlesen werden diese Termine durchforstet und aussortiert. Jene Termine die einen Teilnehmerstatus abgebrochen beinhalten, deren Terminstatus jedoch bestätigt ist, werden in die Tabelle der zu benachrichtigenden Termine hinzugefügt.

Eine weitere Systemregel besagt, dass der Organisator eines Termins zwingend an diesem Termin teilnehmen muss. Für die Umsetzung dieser Regel wird bei jeder Teilnehmerstatusänderung mittels der Methode `updateAttendeeStatusOfSpecificEventByOrganizer(...)`, sei es durch den Benutzer oder durch eine Regel, überprüft, ob der zu ändernde Status jener vom Organisator ist. Ändert der Organisator diesen Teilnehmerstatus auf abgelehnt oder abgebrochen, wird der Temin abgebrochen.

Die verbleibenden Systemregeln nehmen Bezug auf Statusüberprüfungen von Teilnehmern und Terminkollisionsüberprüfungen. Ein Teilnehmer der zu einem Termin zugesagt hat, gilt für diesen Zeitraum als beschäftigt. Bei jeder Überprüfung auf mögliche Terminkollisionen werden sämtliche Termine zum Vergleich herangezogen die den Teilnehmerstatus zugesagt dieses Teilnehmers haben. Bei der Überprüfung wird dabei auf Überschneidungen von Start- und Endzeitpunkten geachtet.

6.3.2.2. Umzusetzende benutzerdefinierte Regeln

Die vom Benutzer definierten Regeln werden zuerst vom `SemCalTriggerHandler` in der jeweiligen Methode (`handleOnInsert(...)`, `handleOnUpdate(...)` oder `handleOnDelete(...)`) überprüft, und im Anschluss daran werden bestimme Aktionen durchgeführt. Es wird nun die Umsetzung der vier vorgegebenen benutzerdefinierten Regeln näher erläutert.

1. Regel: Ich möchte über alle neu erstellten Termine, die die höchste Wichtigkeitskategorie besitzen informiert werden, egal ob ich bereits einen Termin zu diesem Zeitpunkt habe oder nicht.

Beim Einfügen eines neuen Termins wird die Methode `Model.add(...)` aufgerufen, welche vor der Einfügeoperation die Methode `handleOnInsert(...)` im `SemCalTriggerHandler` auslöst. Hier wird nun jede für den Benutzer gültige Regel überprüft.

Nehmen wir als Beispiel an, dass Matthew Sobol diese Regel in seinem Kalender eingetragen hat. In diesem konkreten Fall sieht die Regel folgendermaßen aus:

```
01 <sem:rule
rdf:about="http://localhost/rdf/matthew_protected.rdf#rule12350">
02   <sem:ruleID>12350</sem:ruleID>
03   <sem:ruleDescription>ReportDueToHighImportance</sem:ruleDescription>
04   <sem:ruleEvent>(ON) NEW EVENT (AND) NEW TODO</sem:ruleEvent>
05   <sem:ruleCondition>(IF) IMPORTANCE (&gt;)
IMPORTANT</sem:ruleCondition>
06   <sem:ruleAction>(DO) NOTIFY APP</sem:ruleAction>
07   <sem:ruleOwner rdf:resource="http://localhost/rdf/Matthew"/>
08   <sem:ruleStatus>active</sem:ruleStatus>
09 </sem:rule>
```

John Grady lädt Matthew Sobol nun zu einem Termin ein, welcher die höchste Terminwichtigkeit besitzt (vgl. Abbildung 10).

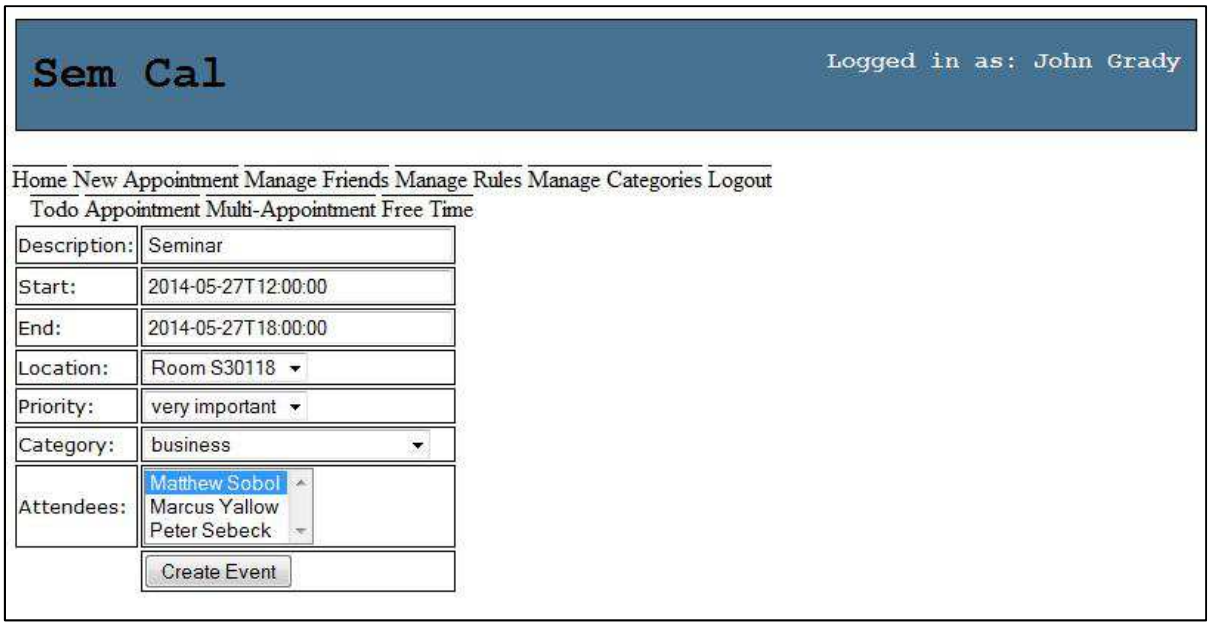


Abbildung 10 Termineinladung mit der höchsten Terminwichtigkeit

Beim Einfügen wird nun in der Methode `handleOnInsert(...)` zuerst überprüft, ob diese Regel auch für dieses Ereignis gilt, danach ob es die Bedingung erfüllt oder verletzt und im Anschluss daran die Aktion gesetzt. Bei dieser Regel ist keine SPARQL ASK Abfrage notwendig um die Bedingung zu überprüfen, und wird daher direkt in Java überprüft. Gleiches würde für eine bestimmte Kategorie gelten. Nach dem Feststellen der Terminwichtigkeit muss im Anschluss die Benachrichtigung für den Regelbesitzer erstellt werden. Diese Benachrichtigung sieht in diesem Fall folgendermaßen aus:

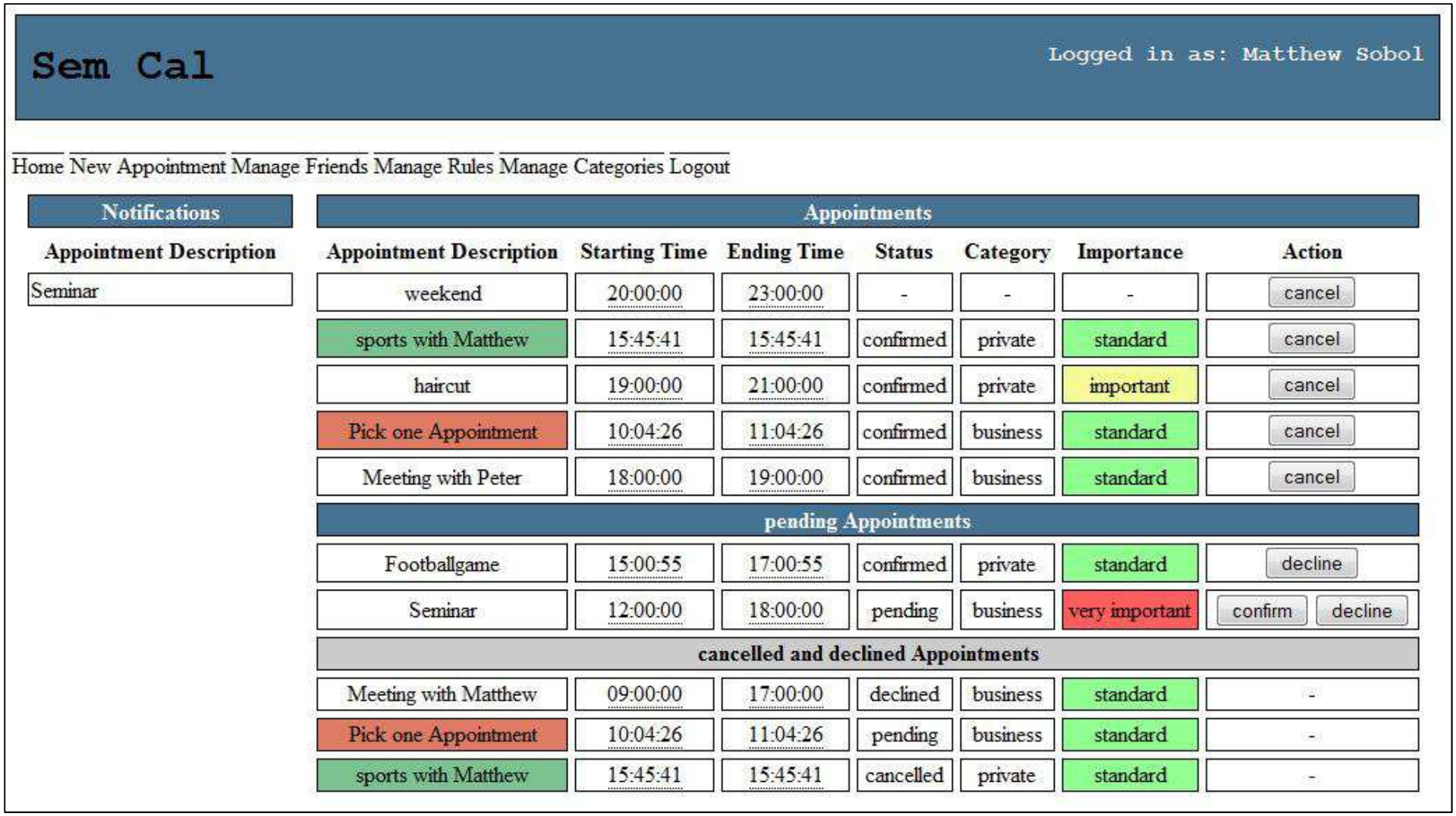
```
01 <sem:IDCheck>
02   <sem:hasIDCreator rdf:resource="http://localhost/rdf/John"/>
03   <sem:hasIDOwner rdf:resource="http://localhost/rdf/Matthew"/>
04   <sem:hasIDToCheck>20140527-32</sem:hasIDToCheck>
05   <sem:hasIDRuleId>12350</sem:hasIDRuleId>
06 </sem:IDCheck>
```

Da die Regel nichts darüber aussagt, ob der Termin abgelehnt oder zugesagt werden soll, wird der Termin in die Public-Graph-Datei von John geschrieben, wobei John den Teilnehmerstatus zugesagt und Matthew den Teilnehmerstatus abwartend besitzt:

```
01 <Vevent rdf:about="20140527-32">
02 <uid>20140527-32</uid>
03 <dtstamp
rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-05-
27T09:16:25</dtstamp>
04 <dtstart
rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-05-
27T12:00:00</dtstart>
05 <dtend rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-
05-27T18:00:00</dtend>
06 <description>Seminar</description>
```

```
07 <location>http://localhost/rdf/s30118.rdf#room</location>
08 <status rdf:resource="http://localhost/rdf/eStatus_pending"/>
09 <sem:Attendee>
10   <rdf:Description rdf:about="20140527-32_01">
11     <sem:attendeePerson rdf:resource="http://localhost/rdf/John"/>
12     <sem:attendeeStatus
rdf:resource="http://localhost/rdf/aStatus_confirmed"/>
13   </rdf:Description>
14 </sem:Attendee>
15 <sem:Attendee>
16   <rdf:Description rdf:about="20140527-32_02">
17     <sem:attendeePerson rdf:resource="http://localhost/rdf/Matthew"/>
18     <sem:attendeeStatus
rdf:resource="http://localhost/rdf/aStatus_pending"/>
19   </rdf:Description>
20 </sem:Attendee>
21 <priority
rdf:resource="http://localhost/rdf/ePriority_veryimportant"/>
22 <categories rdf:resource="http://localhost/rdf/category_2"/>
23 <organizer rdf:resource="http://localhost/rdf/John"/>
24 </Vevent>
```

Beim nächsten Login von Matthew erkennt die Methode `handleEventsToCheck(...)` den eben eingefügten `IDCheck` und fügt diesen Termin in die Liste zu benachrichtigender Termine hinzu. In der Methode zum Anzeigen der Termine `displayEvents(...)` wird diese Liste separat in einer eigenen Tabelle dargestellt (vgl. Abbildung 11).



Sem Cal

Logged in as: Matthew Sobol

Home New Appointment Manage Friends Manage Rules Manage Categories Logout

Notifications

Appointment Description
Seminar

Appointments

Appointment Description	Starting Time	Ending Time	Status	Category	Importance	Action
weekend	20:00:00	23:00:00	-	-	-	cancel
sports with Matthew	15:45:41	15:45:41	confirmed	private	standard	cancel
haircut	19:00:00	21:00:00	confirmed	private	important	cancel
Pick one Appointment	10:04:26	11:04:26	confirmed	business	standard	cancel
Meeting with Peter	18:00:00	19:00:00	confirmed	business	standard	cancel
pending Appointments						
Footballgame	15:00:55	17:00:55	confirmed	private	standard	decline
Seminar	12:00:00	18:00:00	pending	business	very important	confirm decline
cancelled and declined Appointments						
Meeting with Matthew	09:00:00	17:00:00	declined	business	standard	-
Pick one Appointment	10:04:26	11:04:26	pending	business	standard	-
sports with Matthew	15:45:41	15:45:41	cancelled	private	standard	-

Abbildung 11 Benachrichtigung eines Termins mit der höchsten Terminwichtigkeit

Bevor mit der nächsten Regel fortgefahren wird, soll kurz erläutert werden, wie die Anwendung Terminkollisionen ermittelt. Bei Überprüfungen auf Zeitüberschneidungen werden 5 mögliche Konflikte der zwei zu vergleichenden Termine verglichen. Angelehnt an das Allen-Kalkül [Alle83], werden die folgenden 5 Fälle überprüft (vgl. Abbildung 12):

1. Fall: Der Startzeitpunkt des neuen Termins ist kleiner oder gleich dem Startzeitpunkt des existierenden Termins und der Endzeitpunkt des neuen Termins ist größer als der Startzeitpunkt des existierenden Termins.

2. Fall: Der Startzeitpunkt des neuen Termins ist kleiner als der Endzeitpunkt des existierenden Termins und der Endzeitpunkt des neuen Termins ist größer oder gleich dem Endzeitpunkt des existierenden Termins.

3. und 4. Fall: Der Startzeitpunkt des neuen Termins ist größer oder gleich dem Startzeitpunkt des existierenden Termins und der Endzeitpunkt des neuen Termins ist kleiner oder gleich dem Endzeitpunkt des existierenden Termins.

5. Fall: Der Startzeitpunkt des neuen Termins ist kleiner als der Startzeitpunkt des existierenden Termins und der Endzeitpunkt des neuen Termins ist größer als der Endzeitpunkt des existierenden Termins.

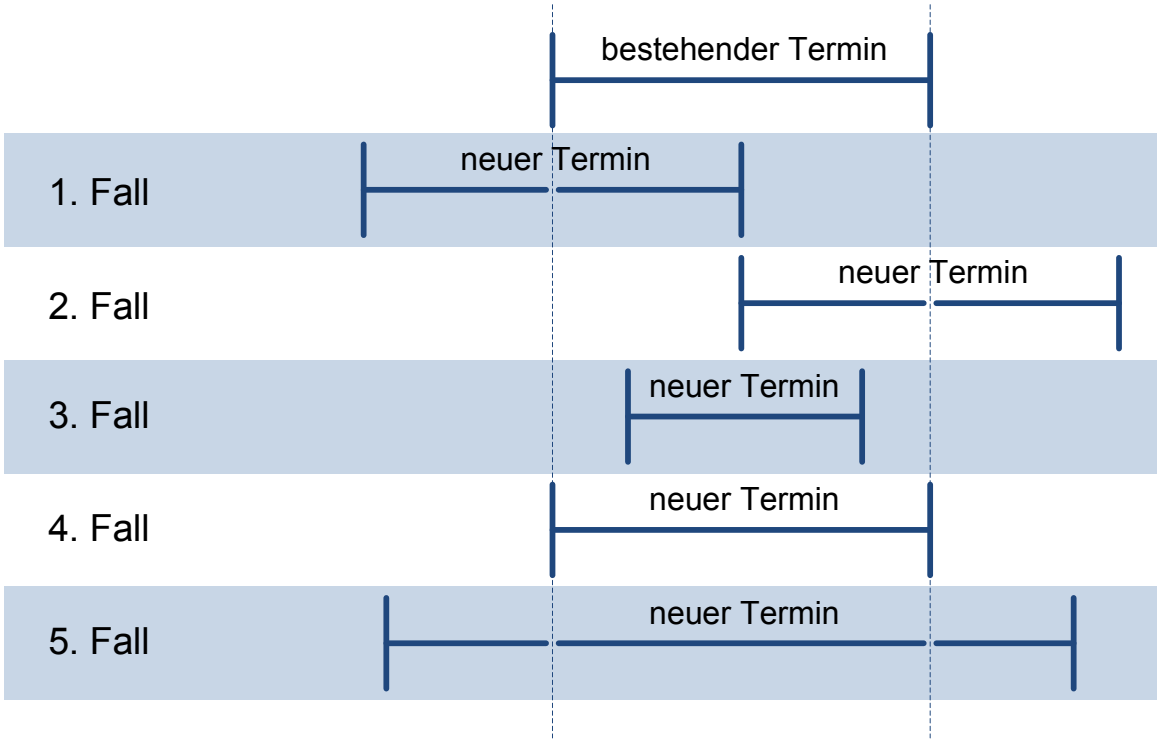


Abbildung 12 Überschneidungsmöglichkeiten zweier Termine (nach [Alle83])

2. Regel: Bei einer Terminkollision bei der Erstellung eines neuen Termins bei dem ich Teilnehmer bin, soll überprüft werden, ob die Priorität des Erstellers höher ist als die Priorität des Erstellers des bereits fixierten Termins. Sollte die Priorität nicht höher sein, soll der neue Termin automatisch abgelehnt werden.

Matthew hat bereits einen fixierten Termin mit Peter, welche sich zu einer gemeinsamen Partie Schach treffen möchten (vgl. Abbildung 13).

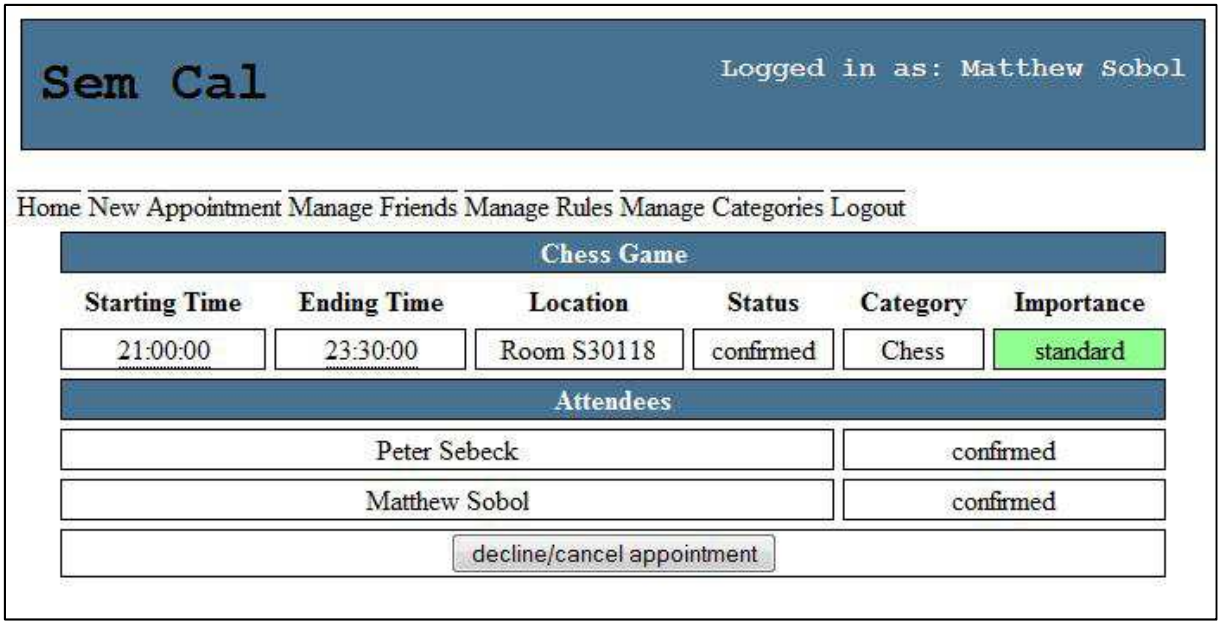


Abbildung 13 bestehender, fixierter Termin zwischen Peter und Matthew

Matthew hat für Peter die Priorität 6 und für John die Priorität 2 vergeben. Matthew hat darüber hinaus diese vorgegebene Regel in seinem Kalender eingetragen:

```
01 <sem:rule
rdf:about="http://localhost/rdf/matthew_protected.rdf#rule12346">
02   <sem:ruleID>12346</sem:ruleID>
03   <sem:ruleDescription>DeclineDueToBusyAndLowerPriority
</sem:ruleDescription>
04   <sem:ruleEvent>(ON) NEW EVENT</sem:ruleEvent>
05   <sem:ruleCondition>(IF) DTSTART (AND) DTEND (=) BUSY (AND)
ORGANIZER.PRIORITY (&lt;) BUSY.PRIORITY</sem:ruleCondition>
06   <sem:ruleAction>(DO) DECLINE APP</sem:ruleAction>
07   <sem:ruleOwner rdf:resource="http://localhost/rdf/Matthew"/>
08   <sem:ruleStatus>active</sem:ruleStatus>
09 </sem:rule>
```

Nun wird Matthew Sobol von John Grady zu einem Termin eingeladen, der sich mit dem Zeitraum der Schachpartie überschneidet (vgl. Abbildung 14). Dieser Termin hat die UID `20140527-34`.

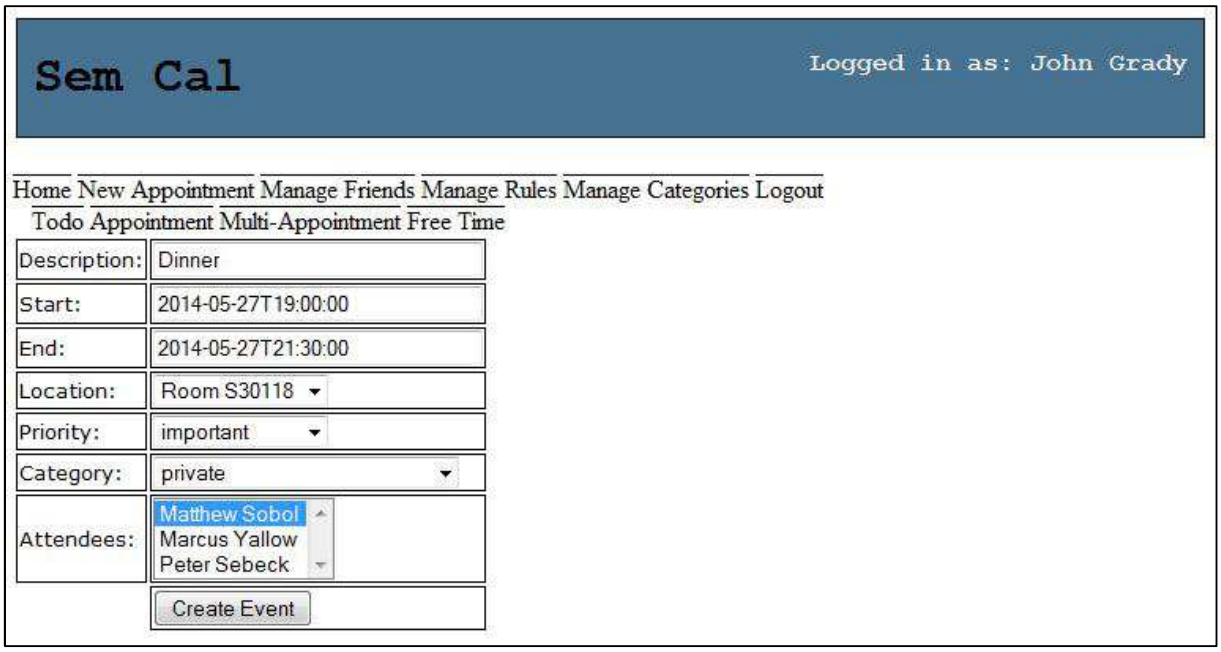


Abbildung 14 Einladung zu einem Termin mit einer Terminkollision

Beim Einfügen wird nun in der Methode `handleOnInsert(...)` zuerst überprüft, ob diese Regel auch für dieses Ereignis gilt, danach ob es die Bedingung erfüllt oder verletzt, und im Anschluss daran die entsprechende Aktion gesetzt. In diesem Fall werden die Bausteine der Regel in eine SPARQL ASK Abfrage umgewandelt. Zuerst muss überprüft werden, ob bei diesem Event der Regelinhaber auch Teilnehmer des neuen Termins ist. Anschließend wird überprüft, ob eine Terminkollision besteht. Dies wird mit folgender SPARQL ASK Abfrage überprüft:

```
01 PREFIX sem: <http://localhost/rdf/sem_cal.rdf#>
02 PREFIX ical: <http://www.w3.org/2002/12/cal/icaltzd#>
03 PREFIX  xsd:    <http://www.w3.org/2001/XMLSchema#>
04 ASK{
05 ?y sem:attendeePerson ?b .
06 ?y sem:attendeeStatus ?c .
07 FILTER ((str(?b) = \"...Teilnehmerressource...\") && (str(?c) =
\"confirmed\") )
08 ?x ical:dtstart ?dtstart
09 ?x ical:status ?status.
10 ?x sem:Attendee ?y.
11 OPTIONAL {?x ical:dtend ?dtend .}
12 OPTIONAL {?x ical:due ?dtend .}
13 FILTER ((str(?status) != \"cancelled\") &&
(xsd:dateTime(?dtstart) >= xsd:dateTime('...Startzeitpunkt...') &&
xsd:dateTime(?dtstart) < xsd:dateTime('...Endzeitpunkt...')) || // #1
(xsd:dateTime(?dtend) > xsd:dateTime('...Startzeitpunkt...') &&
xsd:dateTime(?dtend) <= xsd:dateTime('...Endzeitpunkt...')) ||// #2
(xsd:dateTime(?dtstart) <= xsd:dateTime('...Startzeitpunkt...') &&
xsd:dateTime(?dtend) >= xsd:dateTime('...Endzeitpunkt...')) || #3 + #4
(xsd:dateTime(?dtstart) > xsd:dateTime('...Startzeitpunkt...') &&
xsd:dateTime(?dtend) < xsd:dateTime('...Endzeitpunkt...'))") .   //#5
14 }
```

`Teilnehmerressource` und `Start-` und `Endzeitpunkte` sind dabei jene Werte, die der neu einzufügende Termin besitzt.

Nun müssen die Prioritäten sowohl des Organisators des neuen Termins als auch des Erstellers des bisherigen Termins (d.h. John Grady und Peter Sebeck) verglichen werden. Das System sieht jedoch die Prioritäten, welche in der Private-Graph-Datei von Matthew stehen, nicht. D.h. eine Benachrichtigung für die weitere Überprüfung beim nächsten Login von Matthew muss erstellt werden, in welcher festgehalten wird, welche Regel diese Benachrichtigung ausgelöst hat. In diesem Fall sieht die Benachrichtigung folgendermaßen aus:

```
01 <sem:IDCheck>
02   <sem:hasIDCreator rdf:resource="http://localhost/rdf/John"/>
03   <sem:hasIDOwner rdf:resource="http://localhost/rdf/Matthew"/>
04   <sem:hasIDToCheck>20140527-34</sem:hasIDToCheck>
05   <sem:hasIDRuleId>12346</sem:hasIDRuleId>
06 </sem:IDCheck>
```

Beim nächsten Login von Matthew wird nun in der Methode `handleEventsToCheck(...)` die Regel `12346` erneut überprüft und festgestellt, dass der Ersteller des neuen Termins (John Grady) eine niedrigere Priorität als der Ersteller des vorhandenen Termins (Peter Sebeck) hat. Daraus folgt ein Absagen des neuen Termins, welches durch eine erneute Benachrichtigung für John Grady bedeutet. Diese neue Benachrichtigung sieht folgendermaßen aus:

```
01 <sem:IDUpdate rdf:about="update_20140527-34_1">
02   <sem:hasIDUpdateID>update_20140527-34_1</sem:hasIDUpdateID>
03   <sem:hasIDUpdateUid>20140527-34</sem:hasIDUpdateUid>
04   <sem:hasIDUpdateCreator
rdf:resource="http://localhost/rdf/Matthew"/>
05   <sem:hasIDUpdateOwner rdf:resource="http://localhost/rdf/John"/>
06   <sem:hasIDUpdateStatus>decline</sem:hasIDUpdateStatus>
07 </sem:IDUpdate>
```

Beim erneuten Einloggen von John Grady wird diese Benachrichtigung nun gelesen und der Teilnehmerstatus von Matthew beim Termin mit der UID `20140527-34` auf ablehnen gesetzt (vgl. Abbildung 15).

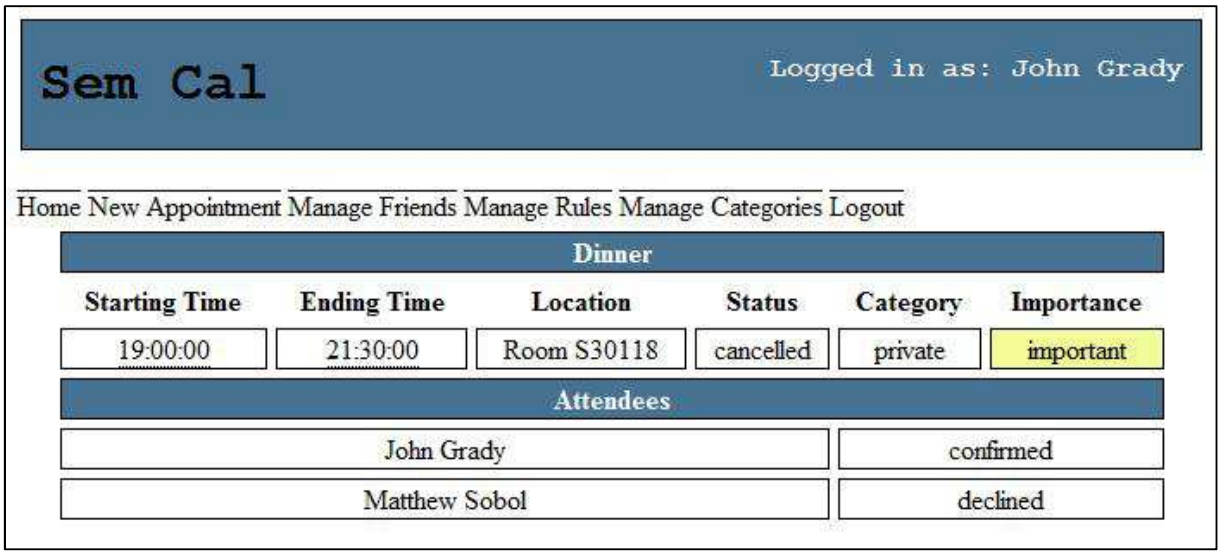


Abbildung 15 Teilnehmerstatusänderung ausgelöst durch eine Regel

Durch die Systemregel, dass, wenn der letzte Teilnehmer (mit Ausnahme des Organisators) einen Termin absagt, wurde hier der Terminstatus automatisch auf abgebrochen gesetzt.

3. Regel: Wird ein Termin einer bestimmten Terminkategorie aktualisiert, soll überprüft werden, ob eine bestimmte Person zu diesem Termin zugesagt hat. Hat sie zugesagt, soll mein eigener Teilnehmerstatus bei diesem Termin auf abgesagt geändert werden.

Bei jedem Aktualisieren eines Teilnehmer- oder Terminstatus wird die Methode `handleOnUpdate(...)` im `SemCalTriggerHandler` ausgelöst. Diese ermittelt alle Regeln, welche bei Updates ausgelöst werden sollen und überprüft die jeweilige Bedingung. In diesem Fall hat John Grady folgende Regel angelegt:

```
01 <sem:rule rdf:about="http://localhost/rdf/john_protected.rdf#12352">
02   <sem:ruleID>12352</sem:ruleID>
03   <sem:ruleDescription>DeclineDueToConfirmFromSpecificFriend
</sem:ruleDescription>
04   <sem:ruleEvent>(ON) UPDATE EVENT</sem:ruleEvent>
05   <sem:ruleCondition>(IF) CATEGORY (IS)
http://localhost/rdf/category_11 (AND) ATTSTATUS (OF)
http://localhost/rdf/Peter (TO) CONFIRMED</sem:ruleCondition>
06   <sem:ruleAction>(DO) DECLINE APP</sem:ruleAction>
07   <sem:ruleOwner rdf:resource="http://localhost/rdf/John"/>
08   <sem:ruleStatus>active</sem:ruleStatus>
09 </sem:rule>
```

Die Ressourcen `http://localhost/rdf/category_11` und `http://localhost/rdf/Peter` beziehen sich dabei einerseits auf die Kategorie mit dem Namen `Matthew Department Meeting` und der Ressource von Peter Sebeck. D.h. wenn Peter Sebeck den Teilnehmerstatus bei einem Termin der Kategorie `Matthew Department Meeting` auf zusagen ändert, soll der Teilnehmerstatus von John Grady automatisch auf ablehnen geändert werden.

Matthew Sobol legt nun einen neuen Termin der Kategorie Matthew Department Meeting an, und lädt dazu John Grady und Peter Sebeck ein (vgl. Abbildung 16).

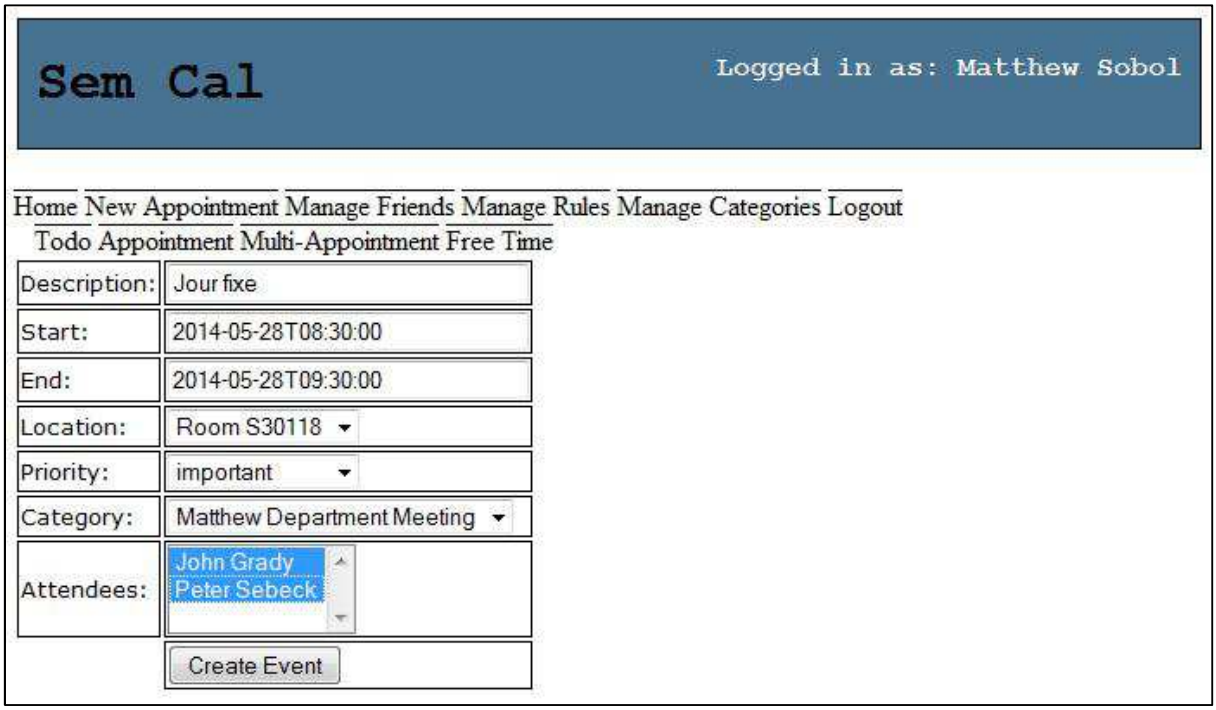


Abbildung 16 Einladung zu einem Termin mit einer bestimmten Kategorie

Anschließend ändert Peter Sebeck seinen Teilnehmerstatus zu diesem Termin auf zugesagt. Nun wird die Methode `handleOnUpdate(...)` aufgerufen und es werden die Bedingungen aller Regeln, die die Teilnehmer dieses Termins besitzen, überprüft. Darunter ist auch die Regel von John, die in diesem Fall zu einer weiteren Teilnehmerzustandsänderung führt. Für diese Änderung wird deshalb eine Benachrichtigung für Matthew angelegt, die besagt, dass der Teilnehmerstatus von John automatisch auf abgelehnt geändert werden soll. Beim nächsten Login von Matthew wird diese Benachrichtigung umgesetzt und der Termin hat folgenden Status (vgl. Abbildung 17):

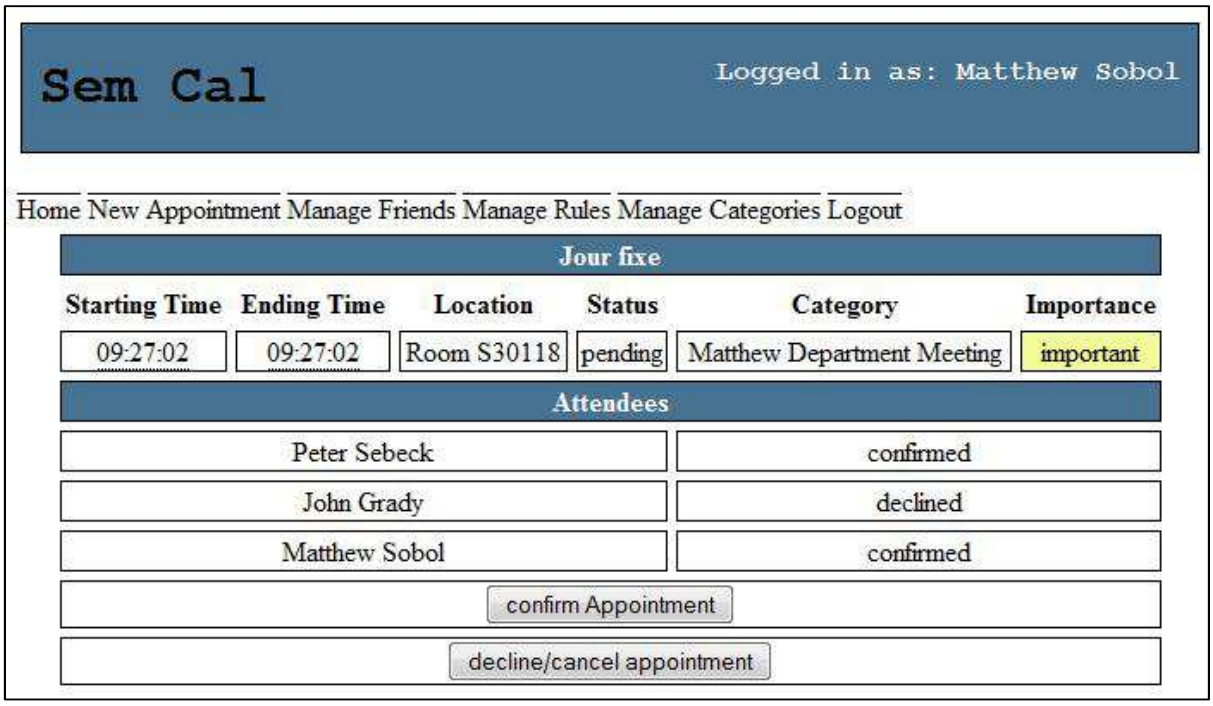


Abbildung 17 Teilnehmerstatusänderung aufgrund einer Regel

4. Regel: Wird ein neuer Termin einer bestimmten Terminkategorie, bei dem ich als Teilnehmer eingeladen werde, erstellt, soll überprüft werden, ob der Ersteller eine gewisse Priorität überschreitet und ob ich keinen weiteren Termin in dieser Zeit habe. Trifft dies zu, soll der Termin automatisch zugesagt werden.

John Grady hat in seinem Terminkalender folgende Regel eingetragen:

```
01 <sem:rule rdf:about="http://localhost/rdf/john_protected.rdf#12353">
02   <sem:ruleID>12353</sem:ruleID>
03   <sem:ruleDescription>ConfirmDueToHighPrioAndCategoryAndNotBusy
</sem:ruleDescription>
04   <sem:ruleEvent>(ON) NEW EVENT</sem:ruleEvent>
05   <sem:ruleCondition>(IF) DTSTART (AND) DTEND (!=) BUSY (AND)
ORGANIZER.PRIORITY (&gt;) 6 (AND) CATEGORY (IS)
http://localhost/rdf/category_7 </sem:ruleCondition>
06   <sem:ruleAction>(DO) CONFIRM APP</sem:ruleAction>
07   <sem:ruleOwner rdf:resource="http://localhost/rdf/John"/>
08   <sem:ruleStatus>active</sem:ruleStatus>
09 </sem:rule>
```

Der Baustein `http://localhost/rdf/category_7` steht dabei für die Terminkategorie `Matthew Business`. Diese Regel sagt nun aus, dass wenn John für einen Temin mit der Kategorie `Matthew Business` von einem Organisator eingeladen wird, der bei John eine höhere Priorität als 6 besitzt und er zu diesem Zeitpunkt keine Termine hat, der Teilnehmerstatus von John bei diesem Termin automatisch auf zugesagt geändert werden soll.

Matthew Sobol erstellt nun einen Termin mit der Terminkategorie Matthew Business. Die höchste Priorität, die Matthew bei John besitzt ist 7 (vgl. Abbildung 18).

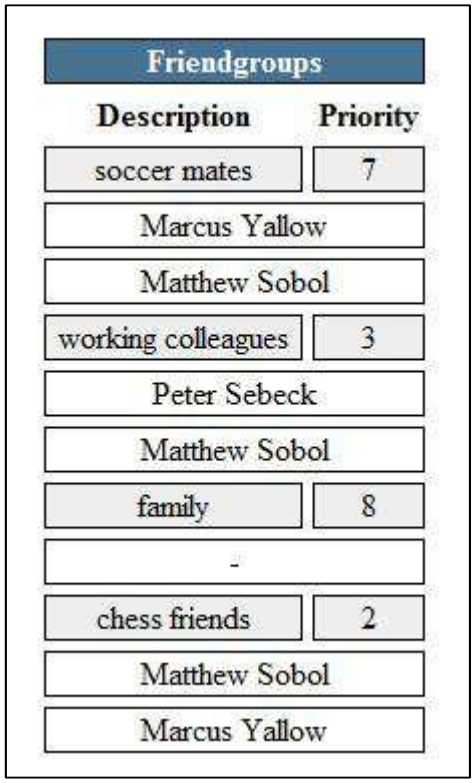


Abbildung 18 Freundesgruppen mit Prioritäten von John Grady

Nach der Erstellung des neuen Termins, wird beim Einfügen die Regel von John von der Methode `handleOnInsert(...)` mittels einer SPARQL ASK Abfrage überprüft und festgestellt, dass er in dieser Zeit keinen weiteren Termin hat. Da die Priorität von Matthew zu diesem Zeitpunkt nicht überprüft werden kann, wird eine Benachrichtigung zur Überprüfung dieses Termins für John erstellt:

```
01 <sem:IDCheck>
02   <sem:hasIDCreator rdf:resource="http://localhost/rdf/Matthew"/>
03   <sem:hasIDOwner rdf:resource="http://localhost/rdf/John"/>
04   <sem:hasIDToCheck>20140529-39</sem:hasIDToCheck>
05   <sem:hasIDRuleId>12353</sem:hasIDRuleId>
06 </sem:IDCheck>
```

Beim nächsten Login von John Grady wird in der Methode `handleUpdates(...)` der Termin `20140529-39` erneut mit der Bedingung der Regel `12353` überprüft und festgestellt, dass Matthew's Prioriät höher als die in der Regel verlangte Priorität ist. Somit wird eine weitere Benachrichtigung für Matthew erstellt, in der steht, dass der Teilnehmerstatus von John automatisch auf zugesagt geändert werden soll.

Beim nächsten Login von Matthew wird der Teilnehmerstatus von John automatisch geändert (vgl. Abbildung 19).

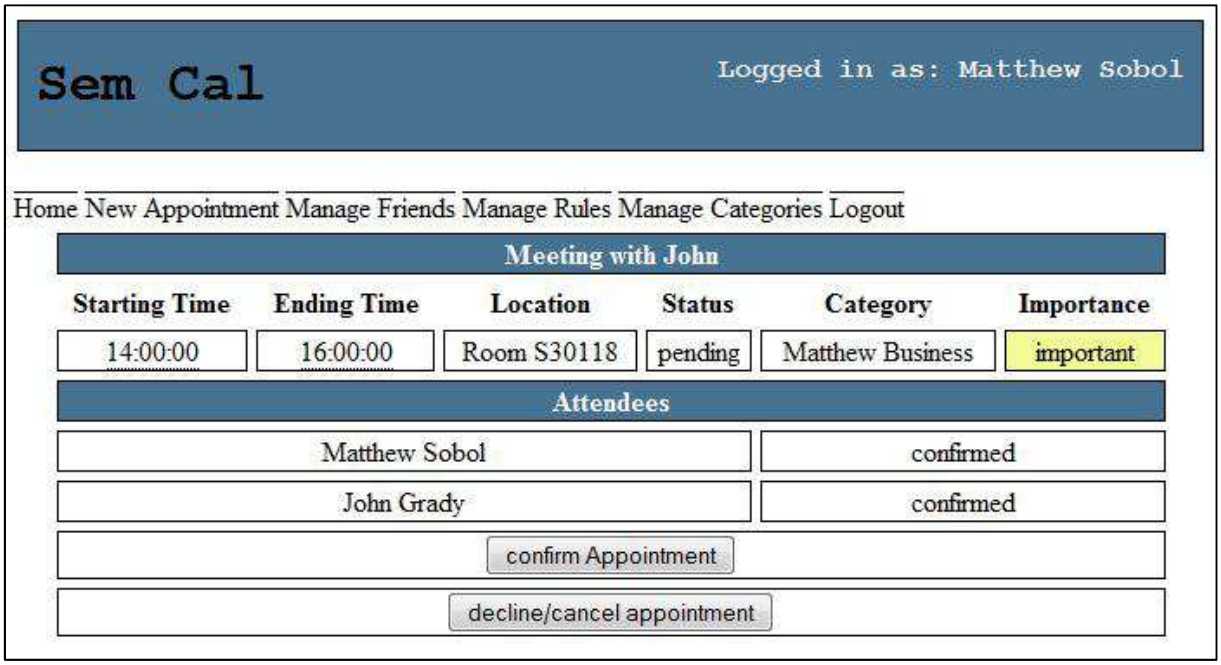


Abbildung 19 Automatische Teilnehmerstatusänderung ausgelöst durch eine Regel

6.3.3. Methoden zur Automatisierung von Änderungen

Die Anwendung benötigt mehrere Methoden, welche die erstellten Benachrichtigungen abarbeiten und im Anschluss wieder entfernen müssen. Eine besondere Methode ist dabei die Methode `handleEventsToCheck(...)`, die jene Benachrichtigungen abarbeitet, welche nach der Überprüfung von Regeln im `SemCalTriggerHandler(...)` durch den Controller erstellt wurden.

`handleEventsToCheck(...)` wird nach dem Einloggen des Benutzers, bevor die Termine angezeigt werden, aufgerufen und geht jeden für diesen Benutzer relevanten `IDCheck` durch, und muss die vorher mit SPARQL ASK überprüften Bedingungen zum Teil erneut überprüfen. In dieser Methode werden auch die bis dahin für das System unsichtbaren Prioritäten dieses Benutzers ausgelesen und zur Regelüberprüfung herangezogen. Wird zum Beispiel ein `IDCheck` bearbeitet, welcher besagt, dass ein Teilnehmerstatus bei einem neuen Termin auf zugesagt geändert werden soll, der Benutzer jedoch zu diesem Zeitpunkt bereits zwei weitere Termin hat, müssen die alten Termine abgesagt werden. Dieses Absagen wird ebenfalls in dieser Methode ausgeführt. Wurden alle `IDChecks` abgearbeitet, werden Antworten mit Statuswerten zur Aktualisierung erstellt. Nach dem Abarbeiten dieser Antworten können alte `IDChecks` gelöscht werden.

Die Methode `handleUpdates(...)` wird ebenfalls nach dem Login des Benutzers ausgeführt und kümmert sich um Teilnehmerstatusänderungen, welche durch diesen Benutzer durchgeführt werden müssen. Wurden alle `IDUpdates` abgearbeitet, werden ebenfalls Antworten dafür erstellt. Durch das Abarbeiten dieser Antworten können alte `IDUpdates` später gelöscht werden.

Die Methode `handleAnswersToCheck(...)` ist die dritte und letzte Methode, die Benachrichtigungen abarbeitet. Sie kümmert sich um die durch die Methoden `handleUpdates(...)` und `handleEventsToCheck(...)` erstellten Antworten. Im Anschluss daran werden alte Einträge mit den Methoden `deleteOldIDUpdates(...)`, `deleteIDAnswers(...)`, `deleteIDChecks(...)` und `deleteOldIDUpdateAnswers(...)` gelöscht.

6.3.4. Besondere Eigenschaften dieser Implementierung

Wie bereits erwähnt wurde, ergibt sich durch die Anforderung, dass die RDF Dateien verteilt auf entfernten Webservern liegen, eine halb-automatische Durchführung von Statusaktualisierungen. Das heißt, bis zum Zeitpunkt des neuen Logins eines Terminorganisators oder des eingeladenen Teilnehmers, werden die Zustandsänderungen nicht übernommen. Das bedeutet, dass diese Art von Anwendung eine nicht synchronisierte Datenhaltung erlaubt und Regeln oft erst zu einem späteren Zeitpunkt, zum Beispiel beim nächsten Login des eingeladenen Teilnehmers, abgearbeitet werden.

Verstärkt wird dieser Effekt dadurch, dass eine beschränkte Sichtbarkeit auf Daten besteht. D.h. Prioritäten können nur von Inhabern der Private-Graph-Datei eingesehen werden. Das bedeutet, dass die Dauer der nicht-synchronen Daten erhöht wird, weil mehrere Login-Vorgänge von Terminorganisatoren benötigt werden.

Um dies zu verdeutlichen, soll das Beispiel aus der 2. Regel aus Kapitel 6.3.2.2 noch einmal herangezogen werden, wobei der Termin zu dem Matthew eingeladen wird, einen weiteren Teilnehmer, nämlich Peter Sebeck hat. D.h. John hat nun Matthew und Peter zum neuen Termin eingeladen. Bei der Erstellung des Termins wird eine Benachrichtigung für Matthew erstellt, weil dessen Regel verletzt wurde. Peter sieht währenddessen den Teilnehmerstatus von Matthew auf abwartend. Beim nächsten Login-Vorgang wird bei Matthew die Regel erneut überprüft und eine Antwort (in diesem Fall ein Absagen des Termins) erstellt. Peter sieht diese Änderung jedoch erst ab dem Zeitpunkt, in dem sich der Organisator, in diesem Fall John, neu einloggt. Zum Zeitpunkt des neuen Logins von John, wird der Teilnehmerstatus von Matthew aktualisiert. Nun erst ist für Peter klar, dass Matthew den Termin abgesagt hat.

Die nicht synchronen Daten stellen in diesem Beispiel kein Problem dar, da der Teilnehmerzustand von Matthew Sobol nur abwartend war. Tritt jedoch der Fall auf, dass Matthew einen bereits zugesagten (oder fixierten) Termin absagt, wird die Änderung auch erst beim nächsten Login von John übernommen. In dieser Zeit könnte er jedoch bereits von anderen Teilnehmern zu neuen Terminen eingeladen werden.

Es sollen nun zwei Möglichkeiten vorgestellt werden, die Zeitspanne für die verspätete Reaktion auf verteilte Terminänderungen zu verringern.

6.3.4.1. Zentrale Datenhaltung

Eine Möglichkeit, die Dauer in der die Daten nicht synchronisiert sind zu verringern ist, dass man die Daten nicht verteilt im Internet hält, sondern zentral auf einem Server speichert. Dies ermöglicht der Anwendung die Abarbeitung von Teilnehmerstatus- und Terminstatusänderungen zum Zeitpunkt des Bekanntwerdens des Änderungswunsches. D.h. sollte Matthew Sobol eine Teilnehmerstatusänderung zu einem Termin äußern, welcher von John Grady erstellt wurde, kann dies unmittelbar passieren und sofort in der Public-Graph-Datei von John Grady aktualisiert werden, da man keine eigenen Anmeldeinformationen für den Webserver auf dem John's Dateien liegen benötigt.

Die Sichtbarkeit von Prioritäten wird dadurch jedoch nicht beeinflusst. D.h. eine Prioritätsüberprüfung würde trotzdem bis zum Login von John warten müssen.

6.3.4.2. Periodic polling

Eine weitere Möglichkeit die verspätete Reaktion auf Änderungen zu reduzieren wäre es, dass man regelmäßig einen Loginversuch der Teilnehmer mittels periodic polling simuliert und die zu aktualisierenden Änderungen vom System durchführen lässt. Diese Möglichkeit hat den Vorteil, dass der Benutzer weiterhin über seine Daten selbst verfügen kann und an der Architektur der Anwendung selbst nichts geändert werden muss. Ein sinnvolles Wiederholungsintervall liegt hierbei im einstelligen Minutenbereich.

6.3.5. Benutzerhandbuch

Um die Anwendung Sem Cal zu starten bedarf es keiner Installation seitens des Benutzers. Es muss jedoch auf dem Webserver, auf dem die RDF Dateien liegen, ein Verzeichnis für WebDAV–Zugriffe konfiguriert sein (Konfigurationseinstellungen befinden sich im Anhang).

Der Benutzer kann mit dem Aufruf der index.jsp-Datei auf dem Sem Cal Webserver die Anwendung starten. Er landet dabei auf einer Loginseite, welche WebDAV Zugangsdaten und die Adresse der Private-Graph-Datei des Benutzers verlangt (vgl. Abbildung 20).

Abbildung 20 Loginbereich von Sem Cal

Nach erfolgreichem Login landet der Benutzer auf der Hauptseite, die eine Terminübersicht und ein Menü beinhaltet (vgl. Abbildung 21). Der Benutzer sieht hierbei alle fixierten, abwartenden und abgebrochenen Termine. Mit einem Mouse-Over über Start- und Endzeitpunkten von Terminen wird ein genaues Datum des Termins angezeigt. Mit einem Klick auf den Namen eines Termins, gelangt man zu einer detaillierten Ansicht dieses speziellen Termins (vgl. Abbildung 22). Der Benutzer kann sowohl in der Detailansicht als auch auf der Hauptseite mit den Schaltflächen `confirm`, `decline` und `cancel` seine eigenen Teilnehmerstatus ändern. Das Menü auf der Hauptseite erlaubt es dem Benutzer Termine hinzuzufügen, seine Freunde zu organisieren, neue Regeln hinzuzufügen und neue Kategorien zu erstellen.

Sem Cal

Logged in as: John Grady

Home New Appointment Manage Friends Manage Rules Manage Categories Logout
Todo Appointment Multi-Appointment Free Time

Appointments

Appointment Description	Starting Time	Ending Time	Status	Category	Importance	Action
Easterholidays	12:44:54	22:44:54	-	-	-	cancel
holiday trip	20:00:0	2014-04-12T12:44:5	-	-	-	cancel
sports with Matthew	15:45:41	15:45:41	confirmed	private	standard	cancel
Pick one Appointment	10:04:26	11:04:26	confirmed	business	standard	cancel
Meeting with Peter	18:00:00	19:00:00	confirmed	business	standard	cancel

pending Appointments

Meeting with John	14:00:00	16:00:00	confirmed	Matthew Business	important	decline
Seminar	12:00:00	18:00:00	confirmed	business	very important	decline
Johns Birthday Party	15:59:43	16:59:43	cancelled	PeterPrivate	standard	confirm decline

cancelled and declined Appointments

Jour fixe	09:27:02	09:27:02	declined	Matthew Department Meeting	important	-
Dinner	19:00:00	21:30:00	confirmed	private	important	-
Pick one Appointment	10:04:26	11:04:26	cancelled	business	standard	-
sports with Matthew	15:45:41	15:45:41	confirmed	private	standard	-

Abbildung 21 Die Hauptseite mit Terminübersicht von Sem Cal

Mit dem Menüpunkt Home gelangt der Benutzer wieder zu dieser Hauptseite zurück, und mit dem Menüpunkt Logout gelangt er zum Loginbereich. Durch einen Klick auf den Menüpunkt New Appointment wird dem Benutzer ein neues Menü angezeigt, welches ihm erlaubt, eine Auswahl aus dem Anlegen eines Todos, eines neuen Termins, eines neuen Multitermins oder einer FreeBusyTime zu treffen. Stellvertretend für alle Terminklassen wird in Abbildung 23 das Anlegen eines Termins mit mehreren möglichen Abhaltungen gezeigt.

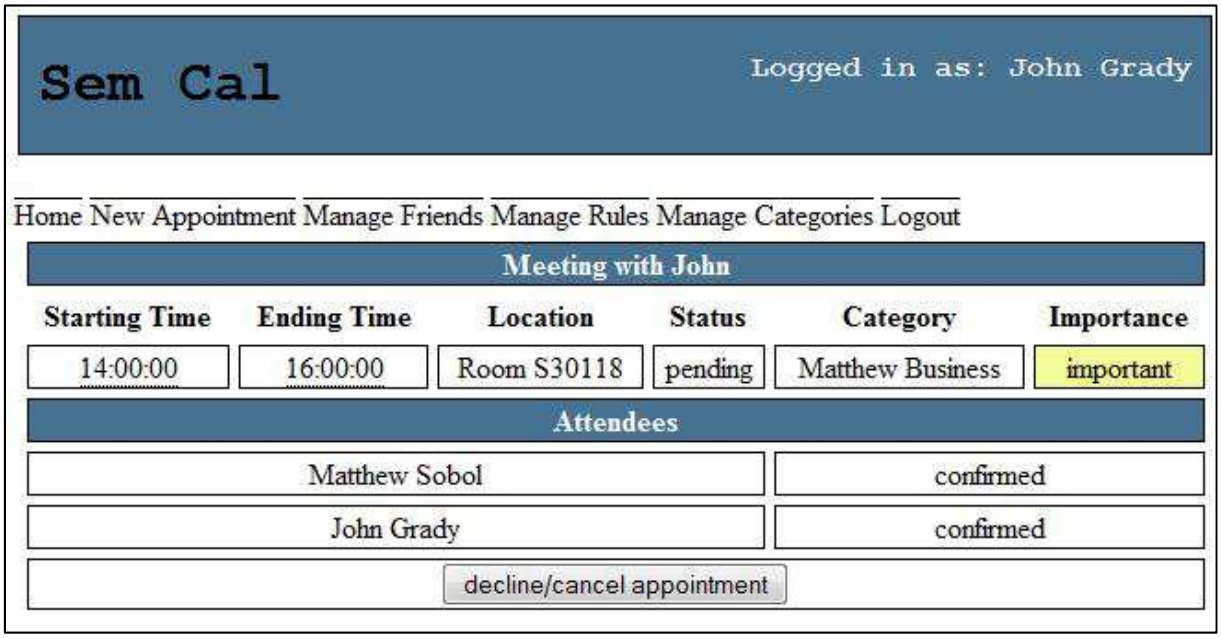


Abbildung 22 Detaillierte Darstellung eines Termins

Bei der Erstellung eines Termins mit mehreren möglichen Abhaltungen wird der Benutzer zuerst nach der Anzahl der möglichen Abhaltungen gefragt, und nach Eingabe deren Anzahl, das Formular zur Eingabe der unterschiedlichen Start- und Endzeitpunkte sowie deren abzuhaltender Ort angezeigt. Mit der Schaltfläche Create Multievent wird schlussendlich die Einfügeoperation gestartet.

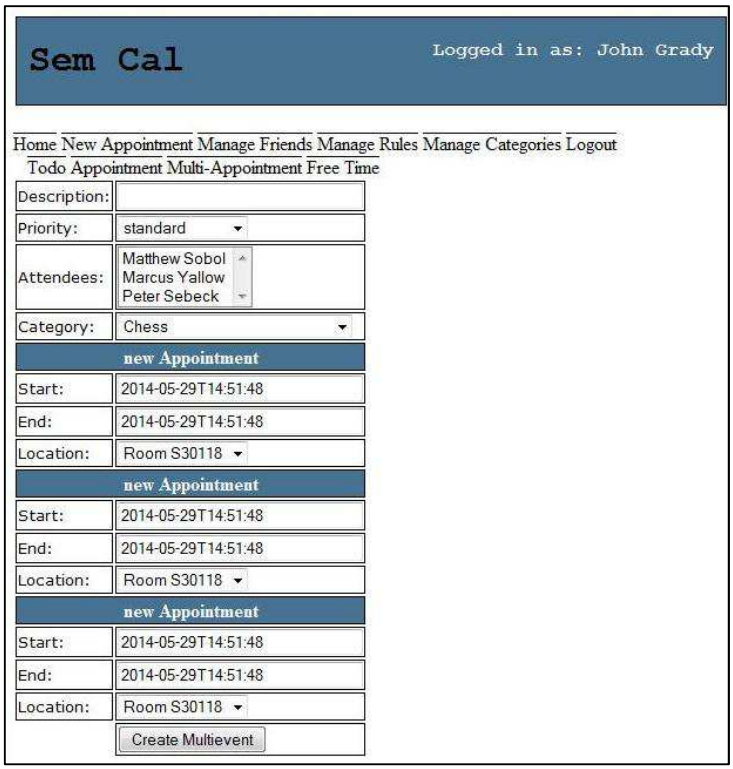


Abbildung 23 Anlegen eines neuen Termins mit mehreren möglichen Abhaltungen

Unter dem Menüpunkt `Manage Friends` kann der Benutzer neue Freunde mittels der Adresse deren Protected-Graph-Dateien hinzufügen, neue Freundesgruppen erstellen, diesen Freundensgruppen Freunde hinzufügen und sowohl die Namen der Freundesgruppen als auch deren Priorität modifzieren (vgl. Abbildung 24). Weiters kann man sich über dieses Menü sowohl die Kontaktdaten als auch die Adressen seiner Freunde anzeigen lassen.

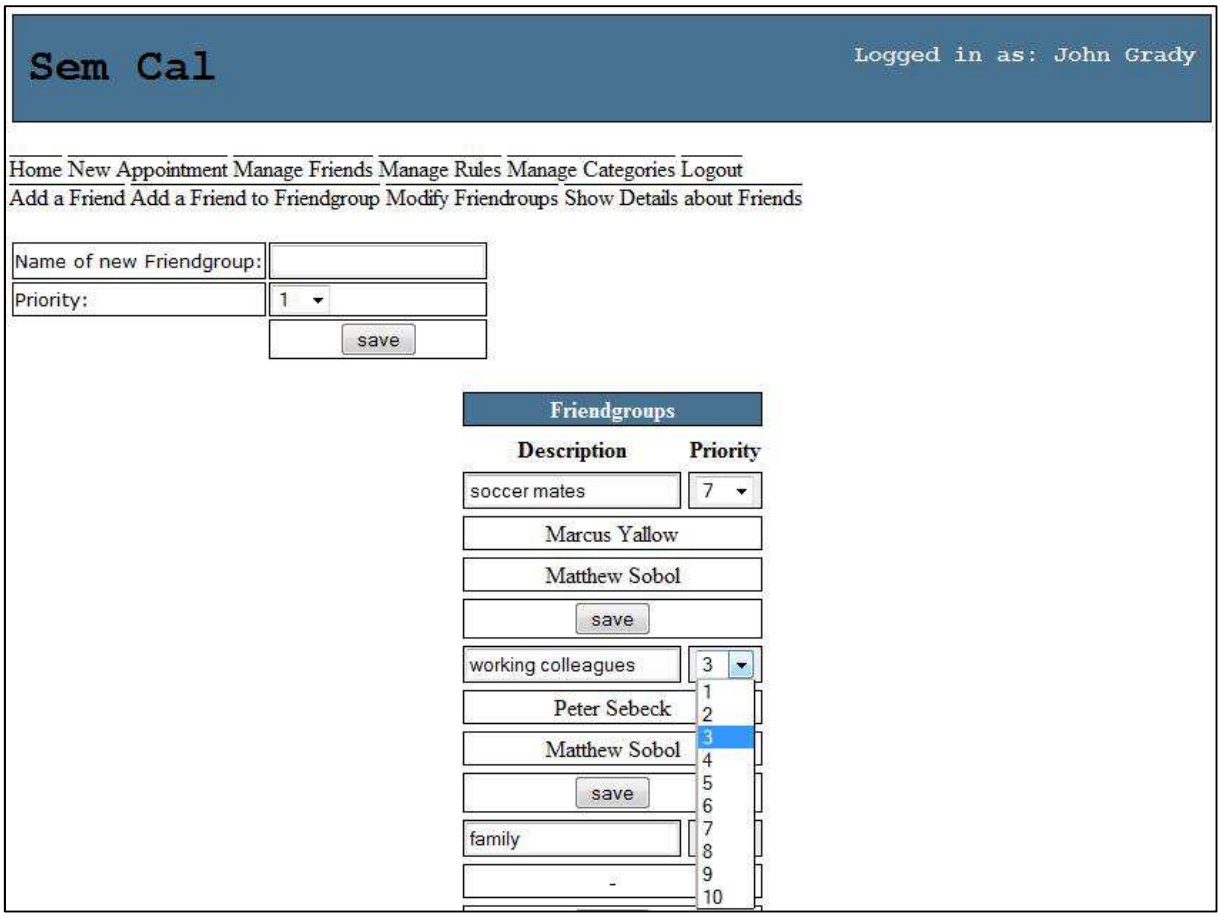


Abbildung 24 Modifzieren von Freundesgruppen und Prioritäten

Der Menüpunkt Manage Rules erlaubt es dem Benutzer neue private Regeln hinzuzufügen und sich bestehende private Regeln anzusehen. Unter dem Menüpunkt `Add a Rule` kann der Benutzer sowohl eine Regel über Bausteine, als auch Regeln mit einer SPARQL ASK Bedingung erstellen (vgl. Abbildung 25). Unter dem Menüpunkt `Manage Categories` kann der Benutzer neue Terminkategorien hinzufügen.

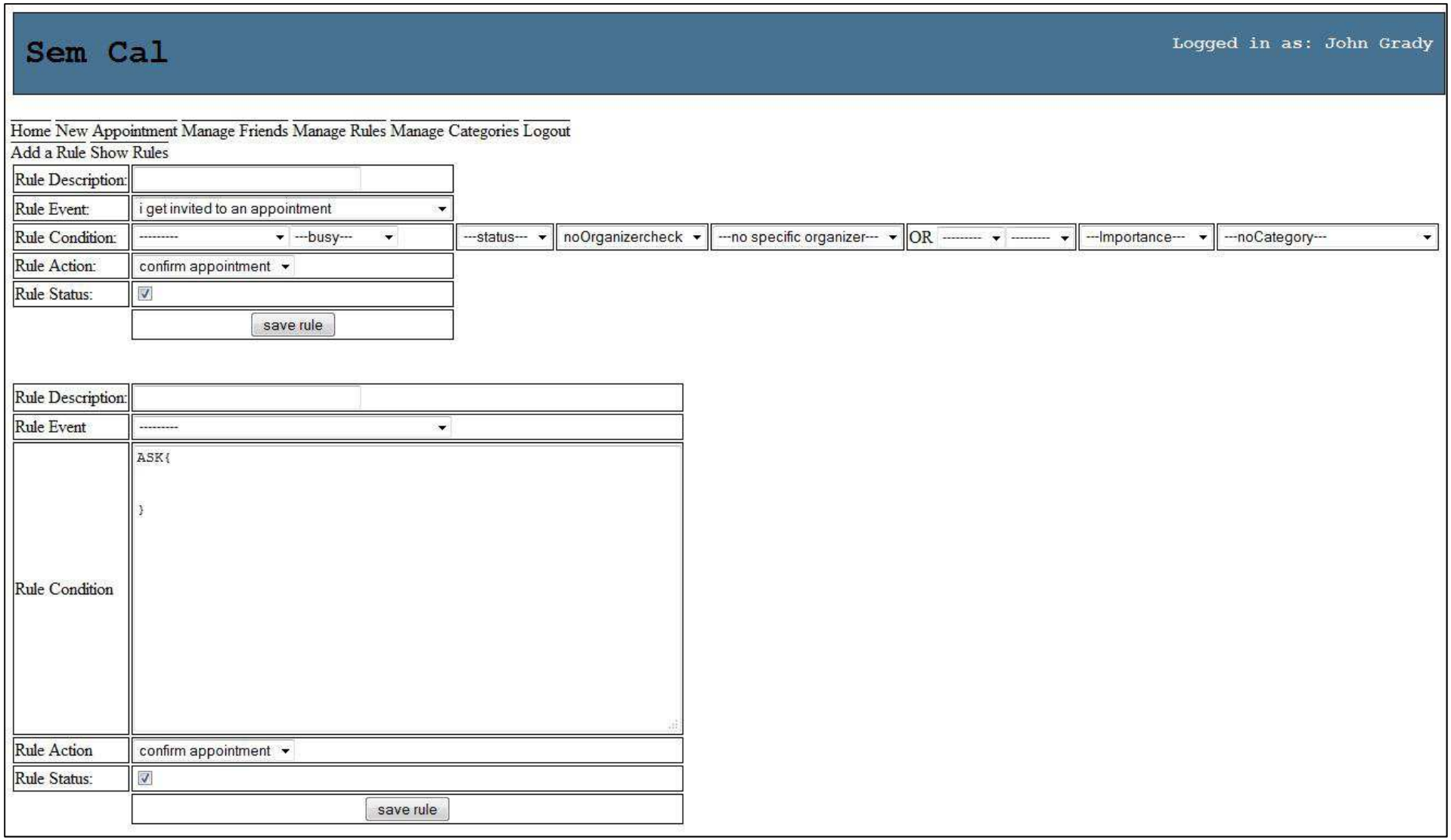


Abbildung 25 Hinzufügen einer neuen Benutzerregel

7. Zusammenfassung und Ausblick

Aufbauend auf vorgegebenen Anforderungen zur Umsetzung einer reaktiven Anwendung, welche Web-basiert sein und das Resource Description Framework verwenden soll, wurden die Aufgaben in Teilkomponenten aufgeteilt und analysiert.

Nach der Erstellung eines Konzepts zur Lösung der Problematik von verteilten Daten mit einer zentralen Verarbeitungslogik und reaktivem Verhalten der Anwendung, wurde ein möglicher Lösungsvorschlag erarbeitet. Aufbauend auf diesem Lösungsvorschlag, wurde ein RDF Vokabular entworfen, welches für die Umsetzung einer Terminorganisation geeignet ist. Es wurden dabei bestehende Vokabulare im Sinne des Semantic Web Gedanken verwendet und erweitert. Anschließend wurde ein Konzept zur Umsetzung von ECA-Regeln in RDF mittels Textbausteinen entworfen und in Verbindung mit SPARQL ASK Abfragen umgesetzt. Die Umsetzung selbst wurde in Java in Verbindung mit dem RDF-Framework JENA durchgeführt, welches um eine Triggerfunktionalität erweitert wurde.

Im Zuge der Implementierung sind Grenzen und Hindernisse dieses Lösungsansatzes klar geworden, welche näher betrachtet und mögliche Lösungsalternativen dafür vorgestellt wurden. Durch eine beschränkte Sicht auf sensible Daten (Prioritäten von Freunden), muss das reaktive Verhalten der Anwendung halb-automatisiert in mehreren Schritten durchgeführt werden.

Der vorgestellte Prototyp kann in verschiedene Richtungen erweitert und eingesetzt werden. Mögliche Erweiterungen sind unter anderen wiederkehrende Termine und die Archivierung von Termindaten. Die Erweiterung von Jena um eine Triggerfunktionalität sowie die Abbildung von Teilen der ECA-Regeln in SPARQL kann neben der Terminorganisation auch für andere reaktive und proaktive Anwendungen eingesetzt werden.

8. Literaturverzeichnis

[Alle83] James F. Allen: *Maintaining knowledge about temporal intervals*. Commun. ACM (CACM) 26(11):832-843, (1983).

[AnHa08] Antoniou, G., van Harmelen, F.: *A Semantic Web Primer.* Second Edition, The MIT Press, Cambridge, Massachusetts, London, England, (2008).

[Apac11] The Apache Software Foundation: *Apache Jena, The core RDF API, Core concepts*. http://jena.apache.org/documentation/rdf/, (2011), abgerufen am 18.05.2014.

[BeBe11] Beckett, D., Berners-Lee, T.: *Turtle – Terse RDF Triple Language*. W3C Team Submission 28 march 2011. http://www.w3.org/TeamSubmission/turtle/, (2011), abgerufen am 08.05.2014.

[Beck04] Beckett, D.: *RDF/XML Syntax Specification (revised)*. W3C Recommendation 10 February 2004. http://www.w3.org/TR/REC-rdf-syntax/, (2004), abgerufen am 30.04.2014.

[Bern92] Berners-Lee, T.: *Basic http as defined in 1992*. http://www.w3.org/Protocols/HTTP/HTTP2.html, (1992), abgerufen am 04.12.2013.

[BeHL01] Berners-Lee, T., Hendler, J., Lassila, O.: *The Semantic Web*. In: Scientific American, (2001): 34-41.

[Bern98] Berners-Lee, T.: *Semantic Web Roadmap*. http://www.w3.org/DesignIssues/Semantic.html, (1998), abgerufen am 04.12.2013.

[Bern00] Berners-Lee, T.: *Semantic Web – XML 2000*. http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html, (2000), abgerufen am 12.12.2013.

[Bern06] Berners-Lee, T.: *Linked Data – Design Issues.* http://www.w3.org/DesignIssues/LinkedData.html, (2006), abgerufen am 13.12.2013.

[Bern07] Berners-Lee, T.: *Semantic Web: Linked Data on the Web*. http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/#%2824%29, (2007), abgerufen am 12.12.2013.

[BFDW+05] Berners-Lee, T., Fielding, R., Day Software, W3C/MIT, Masinter, L., Adobe Systems: *Uniform Resource Identifier (URI): Generic Syntax*. Internet Official Protocol Standards. http://tools.ietf.org/html/rfc3986.html, (2005), abgerufen am 30.04.2014.

[BiCH07] Bizer, C., Cyganiak, R., Heath, T.: *How to Publish Linked Data on the Web.* http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/, (2007), abgerufen am 19.12.2013.

[BrEP06] Bry, F., Eckert, M., Patranjan, P.L.: *Querying Composite Events for Reactivity on the Web.* In: Heng Tao Shen, Jinbao Li, Minglu Li, Jun Ni, Wei Wang (Eds.): Advanced Web and Network Technologies, and Applications, APWeb 2006 International Workshops: XRA, IWSN, MEGA, and ICSE, Harbin, China, January 16-18, 2006, Proceedings. Springer 2006 Lecture Notes in Computer Science ISBN 3-540-31158-0. APWeb Workshops 2006:38-47.

[BrEc06] Bry, F., Eckert, M.: *Twelve Theses on Reactive Rules for the Web.* In: Torsten Grust, Hagen Höpfner, Arantza Illarramendi, Stefan Jablonski, Marco Mesiti, Sascha Müller, Paula-Lavinia Patranjan, Kai-Uwe Sattler, Myra Spiliopoulou, Jef Wijsen (Eds.): Current Trends in Database Technology - EDBT 2006, EDBT 2006 Workshops PhD, DataX, IIDB, IIHA, ICSNW, QLQP, PIM, PaRMA, and Reactivity on the Web, Munich, Germany, March 26-31, 2006, Revised Selected Papers. Springer 2006 Lecture Notes in Computer Science ISBN 3-540-46788-2. EDBT Workshops 2006:842-854.

[BrGu14] Brickley, D., Guha, R.V.: *RDF Schema 1.1*. W3C Recommendation 25 February 2014. http://www.w3.org/TR/rdf-schema/, (2014), abgerufen am 08.05.2014.

[CoMi05] Connolly, D., Miller, L.: *RDF Calendar – an application of the Resource Description Framework to iCalendar Data*. W3C Interest Group Note 29 September 2005. http://www.w3.org/TR/2005/NOTE-rdfcal-20050929/, (2005), abgerufen am 24.11.2013.

[DGHH+14] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, Wei Zhang: *Knowledge vault: a web-scale*

approach to probabilistic knowledge fusion. In: Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, Rayid Ghani (Eds.): The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014. ACM 2014 ISBN 978-1-4503-2956-9. KDD 2014:601-610.

[DiGa96] Dittrich, K., Gatziu, S.: *Aktive Datenbanksysteme, Konzepte und Mechanismen*. Internat. Thomson Publ, (1996).

[Duss07] Dusseault, L.M.: *HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)*. Internet Official Protocol Standards. http://www.webdav.org/specs/rfc4918.html, (2007), abgerufen am 17.05.204.

[Ferr12] Ferrucci, D.A.: *Introduction to "This is Watson".* IBM Journal of Research and Development (IBMRD) 56(3):1, (2012).

[FUGC+99] Fielding, R., UC Irvine, Gettys, J., Compaq/W3C, Mogul, J., Frystyk, H., W3C/MIT, Masinter, L., Xerox, Leach, P., Microsoft, Berners-Lee, T.: *Hypertext Transfer Protocol – HTTP/1.1*. Internet Official Protocol Standards. http://www.w3.org/Protocols/rfc2616/rfc2616.html, (1999), abgerufen am 30.04.2014.

[HaPa14] Hayes, P., Patel-Schneider, P.: *RDF 1.1 Semantics.* W3C Recommendation 25 February 2014. http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/, (2014), abgerufen am 30.04.2014.

[HaSe13] Harris, S., Seaborne, A.: *SPARQL 1.1 Query Language.* W3C Recommendation 21 March 2013. http://www.w3.org/TR/sparql11-query/, (2013), abgerufen am 18.05.2014.

[Heat09] Heath, T.: *Linked Data? Web of Data? WTF?.* http://tomheath.com/blog/2009/03/linked-data-web-of-data-semantic-web-wtf/, (2009), abgerufen am 19.12.20013.

[HeBi11] Heath, T., Bizer, C.: *Linked Data: Evolving the Web into a Global Data Space* (1st edition). In: Synthesis Lectures on the Semantic Web: Theory and Technology, 1:1, Morgan & Claypool, (2011): 1-136.

[HKRS08] Hitzler, P., Krötzsch, M., Rudolph, S., Sure, Y.: *Semantic Web: Grundlagen*. Springer, Berlin, Heidelberg, (2008).

[HPPH05] Horrocks, I., Parsia, B., Patel-Schneider, P., Hendler, J.: *Semantic Web Architecture: Stack or Two Towers?* In: François Fages, Sylvain Soliman (Eds.): Principles and Practice of Semantic Web Reasoning, Third International Workshop, PPSWR 2005, Dagstuhl Castle, Germany, September 11-16, 2005, Proceedings. Springer 2005 Lecture Notes in Computer Science ISBN 3-540-28793-0. PPSWR 2005:37-41.

[IaMc13] Ianella, R., McKinney, J.: *vCard Ontology, For describing People and Organisations*. W3C Working Draft 24 September 2013. http://www.w3.org/TR/vcard-rdf/, (2013), abgerufen am 08.05.2014.

[Iann10] Iannella, R.: *Representing vCard Objects in RDF*. W3C Member Submission 20 January 2010. http://www.w3.org/Submission/vcard-rdf/, (2010), abgerufen am 24.11.2013.

[JaWa04] Jacobs, I., Walsh, N.: *Architecture of the World Wide Web, Volume One*. W3C Recommendation 15 December 2004. http://www.w3.org/TR/webarch/, (2004), abgerufen am 01.05.20014.

[KlCa04] Klyne, G., Carroll, J.: *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation 10 February 2004. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/, (2004), abgerufen am 06.05.2014.

[Pato99] Paton, N.W.: *Active Rules in Database Systems*. Springer, (1999).

[PaPW06] Papamarkos, G., Poulovassilis, A., Wood, P.T.: *Event-condition-action rules on RDF metadata in P2P environments.* Computer Networks (CN) 50(10): 1513-1532, (2006).

[PeVi11] Perreault, S., Viagenie: *vCard Format Specification*. http://tools.ietf.org/html/rfc6350, (2011), abgerufen am 08.05.2014.

[PoPW06] Poulovassilis, A, Paparmarkos, G., Wood, P.T.: *Event-Condition-Action Rule Languages for the Semantic Web.* In: Torsten Grust, Hagen Höpfner, Arantza Illarramendi, Stefan Jablonski, Marco Mesiti, Sascha Müller, Paula-Lavinia Patranjan, Kai-Uwe Sattler, Myra Spiliopoulou, Jef Wijsen (Eds.): Current Trends in Database Technology - EDBT 2006, EDBT 2006 Workshops PhD, DataX, IIDB, IIHA, ICSNW, QLQP, PIM, PaRMA, and Reactivity on the Web, Munich, Germany, March

26-31, 2006, Revised Selected Papers. Springer 2006 Lecture Notes in Computer Science ISBN 3-540-46788-2. EDBT Workshops 2006:855-864.

[Patr05] Patranjan, P.L.: *The Language XChange: A Declarative Approach to Reactivity on the Web.* PhD thesis, Institute for Informatics, University of Munich, (2005).

[SaCy08] Sauermann, L., Cyganiak, R.: *Cool URIs for the Semantic Web*. W3C Interest Group Note 03 December 2008. http://www.w3.org/TR/cooluris/, (2008), abgerufen am 30.04.2014.

[ScBe01] Schrefl, M., Bernauer, M.: *Active XML Schemas.* In: Hiroshi Arisawa, Yahiko Kambayashi, Vijay Kumar, Heinrich C. Mayr, Ingrid Hunt (Eds.): ER 2001 Workshops, HUMACS, DASWIS, ECOMO, and DAMA, Yokohama Japan, November 27-30, 2001, Revised Papers. Springer 2002 Lecture Notes in Computer Science ISBN 3-540-44122-0. ER (Workshops) 2001:363-376.

[ScRa14] Schreiber, G., Raimond, Y.: *RDF 1.1 Primer*. W3C Working Group Note 25 February 2014. http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140225/, (2014), abgerufen am 06.05.2014.

[SnBH06] Shadbolt, N., Berners-Lee, T., Hall, W.: *The Semantic Web Revisited.* In: IEEE Intelligent Systems, 21(3), (2006): 96-101.

[WiCe96] Widom, J., Ceri, S.: *Active Database Systems: Triggers and Rules for Advanced Database Processing*, Morgan Kaufmann, (1996).

9. Anhang

Der Anhang beinhaltet verwendete Programmversionen bzw. Programmbibliotheken sowie Beispiele für verwendete RDF Dateien. Ebenfalls enthalten ist eine kurze Anleitung zur Konfiguration eines WebDAV Verzeichnisses.

A.1 Verwendete Programmversionen bzw. Programmbibliotheken

Im Rahmen dieser Arbeit wurden für die Entwicklung der Benutzerschnittstelle und des Controllers folgende Programmversionen und Bibliotheken eingesetzt:

- Jena 2.6
- ARQ 2.8.7
- Sardine 5.0.1
- HttpComponents-Client 4.3.3
- Ng4j 0.9.3
- Jena-tdb-0.8.10
- Iri-0.6

Für die Webanwendung wurde der Webserver Apache 2.2.3 auf dem Betriebssystem CentOS 5.10 verwendet.

A.2 Konfiguration des Apache Webservers für WebDAV

Für das Schreiben auf entfernten Webservern wurde WebDAV eingesetzt. Dies ist jedoch bei einer Neuinstalltion von Apache 2.2.3 nicht konfiguriert, und muss vom Benutzer selbst eingerichtet werden. Hier werden kurz einige Einstellungen des Webservers erläutert.

Die Module, welche von Apache für WebDAV verwendet werden sind `dav` und `dav_fs`. In der `httpd.conf` Datei (im Verzeichnis `/etc/httpd/conf/httpd.conf`) scheinen folgende Zeilen auf:

```
...
LoadModule dav_module modules/mod_dav.so
LoadModule dav_fs_module module/mod_dav_fs.so
...
```

Es wurde ein Virtueller Host eingerichtet, über den die Abarbeitung der Zugriffe erfolgt. Dafür wurden am Ende der `httpd.conf`-Datei folgende Einträge ergänzt:

```
NameVirtualHost *:80
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html/rdf/
        <Directory /var/www/html/rdf/>
                Options Indexes MultiViews
                AllowOverride None
                Order allow,deny
                allow from all
        </Directory>
</VirtualHost>
```

Das Verzeichnis `/var/www/html/rdf` hat als Besitzer und Gruppenzuordnung `apache`. Dies wurde mit folgendem Befehl gesetzt:

```
chown apache:apache /var/www/html/rdf
```

Für den Zugriff von außen wurden in einer eigenen Datei Passwörter für die jeweiligen Zugriffsbenutzer gesetzt und die Zugriffsrechte auf diese Datei auf `root` und `apache` beschränkt:

```
htpasswd -c /var/www/html/passwd.dav bernhard
```

Folgende Einträge lenken den Aufruf des Verzeichnisses `/webdav` auf den Ordner, in welchem die RDF-Dateien liegen:

```
Alias /webdav /var/www/html/rdf/rdf
       <Location /webdav>
          DAV On
          AuthType Basic
          AuthName "bernhard"
          AuthUserFile /var/www/html/passwd.dav
          Require valid-user
      </Location>
```

A.3 Erstelltes Vokabular für die Terminorganisation

Dateiname: sem_cal.rdf

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:sem="http://localhost/rdf/sem_cal.rdf#"
xmlns="http://www.w3.org/2002/12/cal/icaltzd#">
<!-- classes -->
<rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#Resource"/>
<rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#Person">
<rdfs:subClassOf rdf:resource="http://localhost/rdf/sem_cal.rdf#Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#Room">
<rdfs:subClassOf rdf:resource="http://localhost/rdf/sem_cal.rdf#Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#Equipment">
<rdfs:subClassOf rdf:resource="http://localhost/rdf/sem_cal.rdf#Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#Attendee">
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/12/cal/icaltzd#Vevent"/>
```

```
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/12/cal/icaltzd#Vtodo"/>
</rdfs:Class>
<rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#Rule"/>
<rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#Category"/>
<rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#Friendgroup"/>
<rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#IDCheck"/>
<rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#IDCheckAnswer"/>
<rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#IDUpdate"/>
<rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#IDUpdateAnswer"/>
<rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#Eventstatus"/>
<rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#Attendeestatus"/>
<rdfs:Class rdf:about="http://localhost/rdf/sem_cal.rdf#Eventpriority"/>
<!-- properties -->
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#eventPriorityValue">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#Eventpriority"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#statusValue">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#Eventstatus"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#attendeeStatusValue">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#Attendeestatus"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasCategoryOwner">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#Category"/>
  <rdfs:range rdf:resource="http://localhost/rdf/sem_cal.rdf#Resource"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasCategoryValue">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#Category"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasIDAnswerid">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#IDUpdateAnswer"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasIDAnswerCreator">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#IDCheckAnswer"/>
  <rdfs:range rdf:resource="http://localhost/rdf/sem_cal.rdf#Person"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasIDAnswerOwner">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#IDCheckAnswer"/>
  <rdfs:range rdf:resource="http://localhost/rdf/sem_cal.rdf#Person"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasIDAnswerUid">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#IDCheck"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasIDAnswer">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#IDCheck"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasIDOwner">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#IDCheck"/>
  <rdfs:range rdf:resource="http://localhost/rdf/sem_cal.rdf#Person"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasIDToCheck">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#IDCheck"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasIDUpdateID">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#IDUpdate"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasIDUpdateUid">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#IDUpdate"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
```

```
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasIDUpdateCreator">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#IDUpdate"/>
  <rdfs:range rdf:resource="http://localhost/rdf/sem_cal.rdf#Person"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasIDUpdateOwner">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#IDUpdate"/>
  <rdfs:range rdf:resource="http://localhost/rdf/sem_cal.rdf#Person"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasIDUpdateStatus">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#IDUpdate"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#ruleID">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#Rule"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#ruleDescription">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#Rule"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#ruleEvent">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#Rule"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#ruleCondition">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#Rule"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#ruleAction">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#Rule"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#ruleOwner">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#Rule"/>
  <rdfs:range rdf:resource="http://localhost/rdf/sem_cal.rdf#Person"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#ruleStatus">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#Rule"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasIDruleId">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#IDCheck"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasIDCreator">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#IDCheck"/>
  <rdfs:range rdf:resource="http://localhost/rdf/sem_cal.rdf#Person"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasFriendgroupOwner">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#Friendgroup"/>
  <rdfs:range rdf:resource="http://localhost/rdf/sem_cal.rdf#Person"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasFriends">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#Friendgroup"/>
  <rdfs:range rdf:resource="http://localhost/rdf/sem_cal.rdf#Person"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasRDescription">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#Room"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasEDescription">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#Equipment"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#hasDescription">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#Friendgroup"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#fpriority">
```

```
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#Friendgroup"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchemadatatypes#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#attendeePerson">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#Attendee"/>
  <rdfs:range rdf:resource="http://localhost/rdf/sem_cal.rdf#Person"/>
</rdf:Property>
<rdf:Property rdf:about="http://localhost/rdf/sem_cal.rdf#attendeeStatus">
  <rdfs:domain rdf:resource="http://localhost/rdf/sem_cal.rdf#Attendee"/>
  <rdfs:range rdf:resource="http://localhost/rdf/sem_cal.rdf#Attendeestatus"/>
</rdf:Property>
<!-- values -->
<sem:Eventstatus rdf:about="eStatus_confirmed">
       <sem:statusValue>confirmed</sem:statusValue>
</sem:Eventstatus>
<sem:Eventstatus rdf:about="eStatus_pending">
       <sem:statusValue>pending</sem:statusValue>
</sem:Eventstatus>
<sem:Eventstatus rdf:about="eStatus_cancelled">
       <sem:statusValue>cancelled</sem:statusValue>
</sem:Eventstatus>
<sem:Attendeestatus rdf:about="aStatus_confirmed">
       <sem:attendeeStatusValue>confirmed</sem:attendeeStatusValue>
</sem:Attendeestatus>
<sem:Attendeestatus rdf:about="aStatus_pending">
       <sem:attendeeStatusValue>pending</sem:attendeeStatusValue>
</sem:Attendeestatus>
<sem:Attendeestatus rdf:about="aStatus_declined">
       <sem:attendeeStatusValue>declined</sem:attendeeStatusValue>
</sem:Attendeestatus>
<sem:Attendeestatus rdf:about="aStatus_cancelled">
       <sem:attendeeStatusValue>cancelled</sem:attendeeStatusValue>
</sem:Attendeestatus>
<sem:Eventpriority rdf:about="ePriority_standard">
       <sem:eventPriorityValue>standard</sem:eventPriorityValue>
</sem:Eventpriority>
<sem:Eventpriority rdf:about="ePriority_important">
       <sem:eventPriorityValue>important</sem:eventPriorityValue>
</sem:Eventpriority>
<sem:Eventpriority rdf:about="ePriority_veryimportant">
       <sem:eventPriorityValue>very important</sem:eventPriorityValue>
</sem:Eventpriority>
</rdf:RDF>
```

A.4 Beispiel einer Public-Graph-Datei

Stellvertretend für alle Teilnehmer wird an dieser Stelle die Public-Graph-Datei von Matthew Sobol angeführt. Diese RDF Datei enthält die Daten zur Kontaktaufnahme mit Matthew, und jeweils einen Termin jeder Terminkategorie.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:sem="http://localhost/rdf/sem_cal.rdf#"
xmlns:v="http://www.w3.org/2006/vcard/ns#"
xmlns="http://www.w3.org/2002/12/cal/icaltzd#">
<rdf:Description rdf:about="http://localhost/rdf/Matthew">
       <v:fn>Matthew Sobol</v:fn>
       <v:email>matthew@sobol.exa</v:email>
</rdf:Description>
<Vcalendar rdf:about="http://localhost/rdf/matthew.rdf#Calender1">
<component>
<Vtodo rdf:about="20140326-08">
<uid>20140326-08</uid>
```

```
<dtstamp rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-03-
26T13:14:50</dtstamp>
<dtstart rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-03-
26T19:00:00</dtstart>
<due rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-03-
26T21:00:00</due>
<location rdf:resource="http://localhost/rdf/s30118.rdf#room"/>
<description>haircut</description>
<categories rdf:resource="http://localhost/rdf/category_2"/>
<status rdf:resource="http://localhost/rdf/eStatus_confirmed"/>
<sem:Attendee>
<rdf:Description rdf:about="20140326-08_01">
<sem:attendeePerson rdf:resource="http://localhost/rdf/Matthew"/>
<sem:attendeeStatus rdf:resource="http://localhost/rdf/aStatus_confirmed"/>
</rdf:Description>
</sem:Attendee>
<organizer rdf:resource="http://localhost/rdf/Matthew"/>
<priority rdf:resource="http://localhost/rdf/ePriority_important"/>
</Vtodo>
</component>
<component>
<Vfreebusy rdf:about="20131122-02">
<uid>20131122-02</uid>
<dtstamp rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2013-11-
03T21:14:50</dtstamp>
<dtstart rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2013-11-
04T20:00:00</dtstart>
<dtend rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2013-11-
04T23:00:00</dtend>
<location rdf:resource="http://localhost/rdf/s30118.rdf#room"/>
<description>weekend</description>
<!-- has no priorities!   has no locations!    has no categories!  has no
status!-->
<organizer rdf:resource="http://localhost/rdf/Matthew"/>
</Vfreebusy>
</component>
<component>
<Vevent rdf:about="20140416-20">
<uid>20140416-20</uid>
<dtstamp rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-04-
16T10:44:19</dtstamp>
<dtstart rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-04-
16T15:00:55</dtstart>
<dtend rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-04-
16T17:00:55</dtend>
<description>Footballgame</description>
<location>http://localhost/rdf/s30118.rdf#room</location>
<status rdf:resource="http://localhost/rdf/eStatus_pending"/>
<sem:Attendee>
<rdf:Description rdf:about="20140416-20_01">
<sem:attendeePerson rdf:resource="http://localhost/rdf/Matthew"/>
<sem:attendeeStatus rdf:resource="http://localhost/rdf/aStatus_confirmed"/>
</rdf:Description>
</sem:Attendee>
<sem:Attendee>
<rdf:Description rdf:about="20140416-20_02">
<sem:attendeePerson rdf:resource="http://localhost/rdf/Peter"/>
<sem:attendeeStatus rdf:resource="http://localhost/rdf/aStatus_pending"/>
</rdf:Description>
</sem:Attendee>
<priority rdf:resource="http://localhost/rdf/ePriority_standard"/>
<categories rdf:resource="http://localhost/rdf/category_1"/>
<organizer rdf:resource="http://localhost/rdf/Matthew"/>
</Vevent>
</component>
<component>
<Vevent rdf:about="20140530-40">
<uid>20140530-40</uid>
```

```
<dtstamp rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-05-
30T10:09:39</dtstamp>
<dtstart rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-06-
02T20:00:00</dtstart>
<dtend rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-06-
02T23:30:00</dtend>
<description>Celebration after Exams</description>
<location>http://localhost/rdf/s30118.rdf#room</location>
<status rdf:resource="http://localhost/rdf/eStatus_cancelled"/>
<sem:Attendee>
<rdf:Description rdf:about="20140530-40_01">
<sem:attendeePerson rdf:resource="http://localhost/rdf/Matthew"/>
<sem:attendeeStatus rdf:resource="http://localhost/rdf/aStatus_cancelled"/>
</rdf:Description>
</sem:Attendee>
<sem:Attendee>
<rdf:Description rdf:about="20140530-40_02">
<sem:attendeePerson rdf:resource="http://localhost/rdf/John"/>
<sem:attendeeStatus rdf:resource="http://localhost/rdf/aStatus_pending"/>
</rdf:Description>
</sem:Attendee>
<sem:Attendee>
<rdf:Description rdf:about="20140530-40_03">
<sem:attendeePerson rdf:resource="http://localhost/rdf/Peter"/>
<sem:attendeeStatus>confirmed</sem:attendeeStatus>
</rdf:Description>
</sem:Attendee>
<priority rdf:resource="http://localhost/rdf/ePriority_important"/>
<categories rdf:resource="http://localhost/rdf/category_6"/>
<organizer>http://localhost/rdf/Matthew</organizer>
<related-to>20140530-41</related-to>
<related-to>20140530-42</related-to>
</Vevent>
</component>
<component>
<Vevent rdf:about="20140530-41">
<uid>20140530-41</uid>
<dtstamp rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-05-
30T10:09:40</dtstamp>
<dtstart rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-05-
31T20:00:00</dtstart>
<dtend rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-05-
31T23:30:00</dtend>
<description>Celebration after Exams</description>
<location>http://localhost/rdf/s30118.rdf#room</location>
<status rdf:resource="http://localhost/rdf/eStatus_cancelled"/>
<sem:Attendee>
<rdf:Description rdf:about="20140530-41_01">
<sem:attendeePerson rdf:resource="http://localhost/rdf/Matthew"/>
<sem:attendeeStatus rdf:resource="http://localhost/rdf/aStatus_cancelled"/>
</rdf:Description>
</sem:Attendee>
<sem:Attendee>
<rdf:Description rdf:about="20140530-41_02">
<sem:attendeePerson rdf:resource="http://localhost/rdf/John"/>
<sem:attendeeStatus rdf:resource="http://localhost/rdf/aStatus_confirmed"/>
</rdf:Description>
</sem:Attendee>
<sem:Attendee>
<rdf:Description rdf:about="20140530-41_03">
<sem:attendeePerson rdf:resource="http://localhost/rdf/Peter"/>
<sem:attendeeStatus rdf:resource="http://localhost/rdf/aStatus_pending"/>
</rdf:Description>
</sem:Attendee>
<priority rdf:resource="http://localhost/rdf/ePriority_important"/>
<categories rdf:resource="http://localhost/rdf/category_6"/>
<organizer>http://localhost/rdf/Matthew</organizer>
<related-to>20140530-40</related-to>
<related-to>20140530-42</related-to>
```

```
</Vevent>
</component>
<component>
<Vevent rdf:about="20140530-42">
<uid>20140530-42</uid>
<dtstamp rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-05-
30T10:09:41</dtstamp>
<dtstart rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-05-
30T20:00:00</dtstart>
<dtend rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-05-
30T23:30:00</dtend>
<description>Celebration after Exams</description>
<location>http://localhost/rdf/s30118.rdf#room</location>
<status rdf:resource="http://localhost/rdf/eStatus_confirmed"/>
<sem:Attendee>
<rdf:Description rdf:about="20140530-42_01">
<sem:attendeePerson rdf:resource="http://localhost/rdf/Matthew"/>
<sem:attendeeStatus rdf:resource="http://localhost/rdf/aStatus_confirmed"/>
</rdf:Description>
</sem:Attendee>
<sem:Attendee>
<rdf:Description rdf:about="20140530-42_02">
<sem:attendeePerson rdf:resource="http://localhost/rdf/John"/>
<sem:attendeeStatus rdf:resource="http://localhost/rdf/aStatus_confirmed"/>
</rdf:Description>
</sem:Attendee>
<sem:Attendee>
<rdf:Description rdf:about="20140530-42_03">
<sem:attendeePerson rdf:resource="http://localhost/rdf/Peter"/>
<sem:attendeeStatus rdf:resource="http://localhost/rdf/aStatus_confirmed"/>
</rdf:Description>
</sem:Attendee>
<priority rdf:resource="http://localhost/rdf/ePriority_important"/>
<categories rdf:resource="http://localhost/rdf/category_6"/>
<organizer>http://localhost/rdf/Matthew</organizer>
<related-to>20140530-40</related-to>
<related-to>20140530-41</related-to>
</Vevent>
</component>
</Vcalendar>
</rdf:RDF>
```

A.5 Beispiel einer Protected-Graph-Datei

Stellvertretend für alle Teilnehmer wird an dieser Stelle die Protected-Graph-Datei von Matthew Sobol angeführt. Diese RDF-Datei enthält Daten über Telefonnummern und Adressen von Matthew, seine erstellten Regeln und Kategorien.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns="http://www.w3.org/2002/12/cal/icaltzd#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:sem="http://localhost/rdf/sem_cal.rdf#"
xmlns:v="http://www.w3.org/2006/vcard/ns#">
<rdf:Description rdf:about="http://localhost/rdf/Matthew">
<rdfs:seeAlso rdf:resource="http://localhost/rdf/matthew_public.rdf"/>
      <v:fax>+43 758 946 124</v:fax>
      <v:tel>
      <rdf:Description>
            <rdf:value>+43 758 946 125</rdf:value>
            <rdf:type rdf:resource="http://www.w3.org/2006/vcard/ns#Home"/>
      </rdf:Description>
      </v:tel>
      <v:tel>
```

```
      <rdf:Description>
            <rdf:value>+43 777 555 44</rdf:value>
            <rdf:type rdf:resource="http://www.w3.org/2006/vcard/ns#Work"/>
      </rdf:Description>
      </v:tel>
      <v:adr>
      <rdf:Description>
            <v:street-adress>Darknetstreet 15</v:street-adress>
            <v:locality>Vienna</v:locality>
            <v:postal-code>1020</v:postal-code>
            <v:country-name>Austria</v:country-name>
            <rdf:type rdf:resource="http://www.w3.org/2006/vcard/ns#Home"/>
      </rdf:Description>
      </v:adr>
</rdf:Description>
<sem:rule rdf:about="http://localhost/rdf/matthew_protected.rdf#rule12346">
<sem:ruleID>12346</sem:ruleID>
<sem:ruleDescription>CancelDueToBusyAndLowerPriority</sem:ruleDescription>
<sem:ruleEvent>(ON) NEW EVENT</sem:ruleEvent>
<sem:ruleCondition>(IF) DTSTART (=) BUSY (AND) ORGANIZER.PRIORITY (&lt;)
BUSY.PRIORITY</sem:ruleCondition>
<sem:ruleAction>(DO) DECLINE APP</sem:ruleAction>
<sem:ruleOwner rdf:resource="http://localhost/rdf/Matthew"/>
<sem:ruleStatus>active</sem:ruleStatus>
</sem:rule>
<sem:rule rdf:about="http://localhost/rdf/matthew_protected.rdf#rule12350">
<sem:ruleID>12350</sem:ruleID>
<sem:ruleDescription>ReportDueToHighImportance</sem:ruleDescription>
<sem:ruleEvent>(ON) NEW EVENT</sem:ruleEvent>
<sem:ruleCondition>(IF) IMPORTANCE (&gt;) IMPORTANT</sem:ruleCondition>
<sem:ruleAction>(DO) NOTIFY APP</sem:ruleAction>
<sem:ruleOwner rdf:resource="http://localhost/rdf/Matthew"/>
<sem:ruleStatus>active</sem:ruleStatus>
</sem:rule>
<sem:rule rdf:about="http://localhost/rdf/matthew_protected.rdf#rule12347">
<sem:ruleID>12347</sem:ruleID>
<sem:ruleDescription>CancelDueToBusy</sem:ruleDescription>
<sem:ruleEvent>(ON) NEW EVENT (AND) NEW TODO (AND) NEW FREEBUSY</sem:ruleEvent>
<sem:ruleCondition>(IF) DTSTART (=) BUSY</sem:ruleCondition>
<sem:ruleAction>(DO) DECLINE APP</sem:ruleAction>
<sem:ruleOwner rdf:resource="http://localhost/rdf/Matthew"/>
<sem:ruleStatus>inactive</sem:ruleStatus>
</sem:rule>
<sem:Friendgroup rdf:about="Friendsgroup_3">
      <sem:hasFriendgroupOwner rdf:resource="http://localhost/rdf/Matthew"/>
      <sem:hasFriends>
      <rdf:Bag rdf:about="bag_3">
      <rdf:li>
            <rdf:Description rdf:about="http://localhost/rdf/John"/>
      </rdf:li>
      </rdf:Bag>
</sem:hasFriends>
</sem:Friendgroup>
<sem:Category rdf:about="category_6">
      <sem:hasCategoryOwner rdf:resource="http://localhost/rdf/Matthew"/>
      <sem:hasCategoryValue>Matthew Private</sem:hasCategoryValue>
</sem:Category>
<sem:Category rdf:about="category_7">
      <sem:hasCategoryOwner rdf:resource="http://localhost/rdf/Matthew"/>
      <sem:hasCategoryValue>Matthew Business</sem:hasCategoryValue>
</sem:Category>
<sem:Category rdf:about="category_11">
      <sem:hasCategoryOwner rdf:resource="http://localhost/rdf/Matthew"/>
      <sem:hasCategoryValue>Matthew Department Meeting</sem:hasCategoryValue>
</sem:Category>
<sem:Friendgroup rdf:about="Friendsgroup_8">
      <sem:hasFriendgroupOwner rdf:resource="http://localhost/rdf/Matthew"/>
      <sem:hasFriends>
      <rdf:Bag rdf:about="bag_8">
```

```
      <rdf:li>
            <rdf:Description rdf:about="http://localhost/rdf/Peter"/>
      </rdf:li>
      </rdf:Bag>
      </sem:hasFriends>
</sem:Friendgroup>
</rdf:RDF>
```

A.6 Beispiel einer Private-Graph-Datei

Stellvertretend für alle Teilnehmer wird an dieser Stelle die Private-Graph-Datei von Matthew Sobol angeführt. Diese RDF-Datei enthält die Adressen der Protected-Graph-Dateien von Matthews‘ Freunden und den Orten die er kennt. Weiters enthält diese Datei die von ihm selbst verteilten Prioritäten für seine Freundesgruppen.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:sem="http://localhost/rdf/sem_cal.rdf#">
<sem:Person rdf:about="http://localhost/rdf/Matthew">
      <rdfs:seeAlso rdf:resource="http://localhost/rdf/matthew_protected.rdf"/>
      <rdfs:seeAlso rdf:resource="http://localhost/rdf/john_protected.rdf"/>
      <rdfs:seeAlso rdf:resource="http://localhost/rdf/peter_protected.rdf"/>
      <rdfs:seeAlso rdf:resource="http://localhost/rdf/s30118.rdf"/>
</sem:Person>
<rdf:Description rdf:about="Friendsgroup_3">
      <sem:hasDescription>Fishing Buddies</sem:hasDescription>
      <sem:fpriority>2</sem:fpriority>
</rdf:Description>
<rdf:Description rdf:about="Friendsgroup_8">
      <sem:hasDescription>Chess Friends</sem:hasDescription>
      <sem:fpriority>6</sem:fpriority>
</rdf:Description>
</rdf:RDF>
```