A REST API for the Groupware Kolab with support for different Media Types

bachelor thesis

Thomas Koch

March 29, 2012

Fernuniversität Hagen
Faculty of mathematics and computer science

matriculation number 7250371

Professor Dr.-Ing. Bernd J. Krämer
Dipl.-Inf. Silvia Schreier

Aufgabenstellung

Entwicklung einer REST-konformen Schnittstelle für die Opensource-Groupware Kolab mit Unterstützung verschiedener Medientypen

Für die Opensource-Groupware Kolab[1] gibt es bisher ein PHP-basiertes Web-Frontend. Als Alternative dazu soll eine REST-konforme Schnittstelle[2] für die Kontaktfunktionalität entwickelt werden. Um die Anbindung an verschiedene Clienten zu unterstützen sollen die folgenden Medientypen unterstützt werden:

- vCard[3]: Für die Darstellung von Kontaktdaten eignet sich vCard, auch hier muss untersucht werden inwiefern die Daten aus Kolab abgebildet werden können.
- Contact Schema von portablecontacts.net[4]: Dieses JSON-Format, das auf vCard basiert, findet inzwischen auch in Open Social[5] Verwendung.
- XHTML: XHTML eignet sich primär für menschliche Clients und kann beliebige Daten enthalten. Hierbei soll auch untersucht werden, inwiefern die Daten mit Hilfe von Microdata angereichert werden können, so dass dieses Format auch für maschinelle Clienten nutzbar wird.

Bei der Implementierung soll untersucht werden, welche Komponenten des Entwurfs für die Unterstützung verschiedener Medientypen gemeinsam genutzt bzw. wiederverwendet werden können. Außerdem soll die Hypermediaunterstützung der verschiedenen Formate untersucht werden: Wie viel muss ein Client vorher wissen und wie viel kann er durch Hyperlinks entdecken?

[1] http://kolab.org
[2] http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
[3] http://datatracker.ietf.org/doc/draft-ietf-vcarddav-vcardrev/?include_text=1
[4] http://portablecontacts.net/draft-spec.html
[5] http://docs.opensocial.org/display/OS/Home

Contents

1 Introduction

Although computers became ubiquitous for some time now, they still don't help their users with their most basic information management needs: Make contacts, calendars, notices and to do items available across different devices and share them with family and peers.

Most existing solutions are either based on non-free software (Microsoft Outlook), or require the user to trust his personal data to the commercial interests of a company.

2 Background and Related Work

This section introduces the existing standards and technologies that were the starting point for this work. Some of these are meant to be augmented and used in the following sections, i.e., REST, Kolab, vCard, iCalendar and PortableContacts. For the others, CardDAV, CalDAV and the IMAP use in Kolab, it is argued why it seems worthwhile to explore an alternative approach.

2.1 REST architectural style

The API to be designed in this work must obey the constraints of a REST architecture, especially its four interface constraints[Fie00, sec. 5.1.5]:

- Every resource is referenceable by a unique URI.
- Resources are manipulated by the submission of resource representations.
- All exchanged messages are self-descriptive which is achieved by using a set of Mediatypes understood by server and client.
- The client only knows the entrance URI of an API beforehand. All other permitted URIs are discovered in responses of the server.

These constraints are further discussed in subsubsection 2.7.1 when discussing how the OpenSocial API violates them.

The requirement to obey the constraints of REST should cause the system to have some characteristics that may be especially advantageous for the use case of a Groupware API. Those characteristics are, according to [Fie00] (cited sections in parentheses):

- Cacheability (sec. 5.1.4) can keep the data available also in offline mode, which improves performance and scalability.
- Simplicity (sec. 2.3.3) helps to integrate the Groupware with other applications, e.g., publishing birthdays of employees in an intranet portal.
- Modifiability (sec. 2.3.4) allows to adapt the Groupware to changes in the organization.
- Reliability (sec. 2.3.7) can be of great importance, if the ability to work depends on the correct working of the Groupware system.
- Anarchic scalability (sec. 4.1.4.1) allows a Groupware to function with other components that are not under the control of the Groupware administrator.

2.2 Kolab

This work presents an API usable for the Groupware system Kolab. "Kolab Groupware" is the name for a system comprising several independent free (as in speech) software products, a relatively small amount of Kolab specific "glue code" and a special way to configure the involved components. On the server side, Kolab's main components are the directory server OpenLDAP, the Mail Transfer Agent Postfix and the IMAP server Cyrus. Kolab works with specialized client software (see subsection 2.3) which is officially available as free extensions to KDE Kontact, Gnome Evolution and the PHP web clients Horde and Roundcube.

The development of Kolab started 2002, when the Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik) commissioned the development of a free alternative to the Microsoft Exchange Server. Kolab has been developed by a joint venture of three companies[Sto04]. Today Kolab is still developed, supported and distributed collaboratively by multiple independent companies.

A REST API for Kolab aims to make it easier for clients to connect to the Kolab server. The REST API should also be implementable by other server solutions and maybe evolve into a standardized Groupware API.

A couple of related terms and concepts exist that all more or less overlap with the functionality provided by Kolab: Groupware, Personal Information Management/Manager (PIM), Group Information Management (GIM), Computer-supported cooperative/collaborative work, Knowledge management, (Enterprise) Content Management. Especially scientific literature uses the term Groupware for other kind of systems then Kolab[Sto04, sec. 2.1]. For the rest of this work however, a Groupware is understood as a software managing address books, calendars, todo items, journals and probably more data for a group of collaborating users.

2.3 IMAP as a collection synchronization protocol

The Kolab Groupware Server is special in that it uses the Internet Message Access Protocol (IMAP) as a synchronization protocol for all its data and thus the IMAP server as a database. Every resource managed by Kolab is attached as an XML document to a dummy mail message in one of several specially annotated[6] IMAP folders. The current Kolab version still uses its own XML format. Kolab version 3 will use XCard and XCal where applicable[7].

Kolab Clients connect to the (Cyrus) IMAP server, filter all folders of a user to find those managed by Kolab and fetch all attachments of mails in those folders. Write operations also use the IMAP protocol.

There are a few appealing advantages to this approach:

- The IMAP infrastructure used for mail can be reused (authentication, backup, quotas, shareable folders).

[6] with the IMAP METADATA Extension (RFC 5464)

[7] `http://blogs.fsfe.org/greve/?p=470` (2012-03-28)

- Data is stored as file attachment. Thus the probably complicate mapping of groupware data items to the needs of a (relational) database is avoided.
- IMAP already supports offline work and later synchronization.

The simplicity of just dropping files in a store used by several concurrent clients however also has drawbacks, e.g: there is no moderating logic on the server site that could verify the correctness of stored data and there are no query or report capabilities.

IMAP in general also comes with its own challenges:

- "The" IMAP standard does not exist. The 108 pages of "core" IMAP VERSION 4rev1 specification (RFC 3501) have been augmented with around two dozens of other specifications[8], of which every IMAP server and client implements another subset. The METADATA extension needed for Kolab for example is not (yet) included in the also very popular IMAP server Dovecot.
- IMAP imposes a folder structure and does not permit alternative structures like tags, as used by Google's GMail service.
- Sam Varshavchik, author of the Courier Mail Transfer Agent, argues that IMAP standard documents are "contradictory" and that implementations define their own understanding of what IMAP is[9].

2.4 vCard, xCard, iCal, xCal

vCard is an IETF standardized Mediatype to "capture and exchange [...] information normally stored within an address book or directory application" [Per11a] e.g., about individuals, groups, organizations or locations (see the vCard KIND property). Closely connected by format, standards body and usage is iCalendar (short iCal), for "representing and exchanging calendaring and scheduling information such as events, to-dos, journal entries, and free/busy information"[Des09]. Both formats together cover most information usually managed by a Groupware system are the base of Kolab's internal storage, the underlying format of CardDav and CalDAV and thus of most free Groupware systems (subsection 2.6).

The vCard and iCalendar media types seem a bit archaic, since they're not based on XML or JSON but on the older Internet Message Format[Res08] (IMF) first defined in RFC822 in 1982. Version 3 of vCard was published in 1998[HSD98] only a few months after the W3C published Version 1.0 of XML[PSMB98] and eight years before JSON became an official standard[Cro06].

Thus vCard and iCalendar look a lot like email or HTTP headers. Fortunately, the xCard[Per11b] and xCal[DDL11] standards are now available as alternative serializations, so that XML tooling can be used. The standards aim for full compatibility between the XML and IMF formats so that no information is lost when converting in either direction.

[8] http://www.apps.ietf.org/rfc/ipoplist.html (2012-3-5)
[9] http://www.courier-mta.org/fud (2012-3-5)

2.5 PortableContacts

PortableContacts[10] is a specification initiated in 2008 by Joseph Smarr while working for the Address Book internet service Plaxo.com[11]. It comprises of a JSON schema for contacts information derived from vCard version 3[12] and a protocol for authorized retrieval of contacts. The schema misses many properties of the current vCard version 4 standard[Per11a] and introduces properties inspired by social networks, e.g. describing social behavior or preferences.

The schema and protocol has been adopted by OpenSocial (subsection 2.7) which is now its main user. The schema part of PoCo is thus the most appropriate format currently available to represent contacts information in JSON and make it easily consumable by JavaScript browser applications.

2.6 WebDAV, CardDAV, CalDAV

The most widely implemented Groupware protocols (in free software) today seem to be CalDAV[DDD07] for calendaring and CardDAV[Dab11] for contacts[13]. Both protocols extend WebDAV[Dus07] and thus inherit its characteristics.

WebDAV extends HTTP to enable "Distributed Authoring and Versioning" (DAV). For this purpose it introduces additional HTTP methods (PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, UNLOCK) and interprets the URI path component as a hierarchic file system.

Two characteristics of WebDAV motivate an investigation of alternative approaches. The first is the protocol's complexity that complicates correct implementation. Unfortunately complexity is hard to assess. Therefor only some indications are provided at this point.

Lisa Dusseault, author of a WebDAV book[Dus04] and the standard itself expressed her dissatisfaction with CalDAV[14]:

> "Were I to propose CalDAV today it would probably be CalAtom."[15]

The three standards WebDAV (127p), CalDAV (107p) and CardDAV (48p) add up to 282 pages of highly specific standards. This is nearly double as much text as necessary for the standards this work is based on (149 pages)[16]. In contrast to WebDAV, Feeds and related technologies are also more widely used so that a web developer might already know the latter standards.

[10] `http://portablecontacts.net` (2012-03-23)
[11] `http://josephsmarr.com/2008/12/31/portable-contacts-the-half-year-in-review` (2012-03-23)
[12] `http://wiki.portablecontacts.net/w/page/17776141/schema` (2012-03-23)
[13] only full free server implementations: Apple Calendar Server, Bedeworks, DAViCal, eGroupWare, Owncloud, SOGo, Tine2.0
[14] `http://nih.blogspot.com/2008/02/nearly-two-years-ago-i-made-prediction.html` (2012-1-6)
[15] CalAtom is presented in subsection 4.3
[16] Atom (43), AtomPub (53), Feed paging (15), OpenSearch (28), Atom Deleted Entry (10)

The large amount of specifications for the WebDAV family is also caused by the many different areas touched, like locking, versioning or authentication. This work deliberately only focuses on a minimal set of features and delegates additional details to other, specialized specifications. The WebDAV requirements excluded from consideration are discussed in 3.7.

The second, and for this work more important characteristic of WebDAV is, that it is not restful, as explained by Roy Fielding[17]:

> PROP* methods conflict with REST because they prevent important resources from having URIs and effectively double the number of methods for no good reason. [...] It really doesn't matter how uniform they are because they break other aspects of the overall model, leading to further complications in versioning (WebDAV versioning is hopelessly complicated), access control (WebDAV ACLs are completely wrong for HTTP), and just about every other extension to WebDAV that has been proposed.
>
> [...]
>
> The problem with MOVE is that it is actually an operation on two independent namespaces (the source collection and destination collection). The user must have permission to remove from the source collection and add to the destination collection, which can be a bit of a problem if they are in different authentication realms. COPY has a similar problem, but at least in that case only one namespace is modified. I don't think either of them map very well to HTTP.

Given the comprehensiveness of CalDAV and CardDAV one would expect these protocols to cover all common use cases. However the calconnect consortium additionally develops two alternative protocols, CalWS-SOAP and CalWS-REST[18].

2.7 OpenSocial

OpenSocial[Ope11] specifies a how data of social networks can be accessed by clients, especially Javascript browser widgets. The broad adoption not only by social networks but also for collaboration software[19] demonstrates a variety of use cases for Browser accessible Groupware data. It has also been proposed to implement an OpenSocial system for the Fernuniversität Hagen[Hü09].

Unfortunately the so called OpenSocial REST API is a poster child for a non restful API that does not warrant its name. It is rather service oriented, as the specification truthfully points out[Ope11, Social API Server, sec 2,Services]:

> OpenSocial defines several services for providing access to a container's data.

[17] `http://tech.groups.yahoo.com/group/rest-discuss/message/5874` (2012-3-5)

[18] `http://calconnect.org/CD1012_Intro_Calendaring_V1.1.shtml` (2012-3-5)

[19] wiki, issue tracker (Confluence, Jira both Atlassian), Groupware (Lotus from IBM), Content Management System (Alfresco, Nuxeo)

2.7.1 Fieldings Critique

This section examines a critique of Fielding of the API[Fie08][20] which helps to further clarify the characteristics of a restful API that must be obeyed in this work and to justify the proposal of a competing API to an already widely adopted one.

> A REST API should not be dependent on any single communication protocol, [...] any protocol element that uses a URI for identification must allow any URI scheme to be used for the sake of that identification. *[Failure here implies that identification is not separated from interaction.]*

OpenSocial defines a construct called "REST-URI-Fragment" which is criticized by Fielding because "identification is not separated from interaction". This URI fragment is in fact an encoding of query parameters as elements of the URI path component [Ope11, Core API Server, sec 2.1.1.2.2, REST-URI-Fragment]:

> Each service type defines an associated partial URI format. The base URI for each service is found in the URI element associated with the service in the discovery document. Each service type accepts parameters via the URL path. Definitions are of the form:
>
> `{a}/{b}/{c}`

An even worse misuse of URIs is present in OpenSocial's service to retrieve multiple albums. There the "c" parameter from above is actually a slash separated list of albums to retrieve. The URI standard however makes clear, that the path component of an URI is intended to indicate some kind of hierarchic order[BLFM05, sec 3.3].

The main part of the OpenSocial API describes how to form URIs to access information or which methods to use on which URIs for different actions. Fielding writes:

> A REST API should spend almost all of its descriptive effort in defining the media type(s) used for representing resources and driving application state[...]. *[Failure here implies that out-of-band information is driving interaction instead of hypertext.]* A REST API must not define fixed resource names or hierarchies[...] *[Failure here implies that clients are assuming a resource structure due to out-of band information[...]].*

The API consequently does not show the kind of simplicity that come with embedded Hyperlinks but force developers to hard code URI construction in client implementations. Such hard coded clients in turn hinder further evolution of the API, the modifiability property or a restful API[Fie00, sec 2.3].

Tables 1 and 2 show some examples of OpenSocial's hard coded URIs. They also outline a minimal set of functionality that is considered in this work for a restful API suitable as a replacement (subsection 3.2):

[20] Fielding referred to a concrete implementation, the "SocialSite REST API".

- Mediatypes to represent users and groups
- CRUD functions for collections of users and, groups
- Representation of group membership
- Representation of relations between Users
- Hyperlinks to a user's collection of albums

URI fragment	Method	Description
`/people/{User-Id}/@self`	GET	profile for User-Id
`/people/{User-Id}/@self`	DELETE	remove User
`/people/{User-Id}/{Group-Id}`	GET	full profiles of group members
	POST	Create relationship, target specified by `<entry><id>` in body
	POST	Update Person
`/people/{Initial-User-Id}/{Group-Id}/{Related-User-Id}`	GET	???
`/people/@supportedFields`	GET	list of supported person profile fields
`/groups/{User-Id}[/{Group-Id}]`	GET	one or all groups of a user
	PUT	update group
	DELETE	delete group
`/groups/{User-Id}`	POST	create group

Table 1: URI fragments for peoples and groups in OpenSocial

URI fragment	Method	Description
`/albums/{User-Id}/@self`	POST	create album
`/albums/{User-Id}/{Group-Id}[/Album-Id]*`	GET	one or multiple albums
`/mediaItems/{User-Id}/{Group-Id}/{Album-Id}/{MediaItem-Id}`	GET	one mediaitem
`/mediaItem/{User-Id}/@self/{Album-Id}` (sic!)	POST	create mediaitem

Table 2: URI fragments for albums and mediaitems in OpenSocial

The last URI in Table 2 is obviously missing an "s" behind `mediaItem`. This typo is present and unfixed in the OpenSocial spec since Version 1.0, released in march 2010. This is of course not a big issue in itself, but rather a sign that the specification is too verbose and does over-specify things that should rather be auto-discovered through hyperlinks.

2.7.2 Possible Improvements

Fielding mentions in a comment to the same blog post[Fie08] that the OpenSocial API "could be made so [restful] with some relatively small changes" but does not specify these

changes. However some issues can be easily identified.

First, the data structures defined in OpenSocial do not use URIs to refer to other resources. Instead they use Object-Ids that must then be inserted in the appropriate URI templates. Examples are the `recipients`, `senderId`, `collectionIds` of messages and the `ownerId` of albums. The person structure does not contain fields referencing other resources. Thus it does not obviously violate REST like the albums and messages. However it does so even worse since there are hidden references only defined out-of-band in the specification. One can retrieve the albums, relations or messages of a user by filling in the `userId` in one of the specified URI templates. If Users would just contain references to other resources related to a user, the specification could already be shortened a lot.

Another missed opportunity for a much more intuitive API is the relation of media items and albums. This seems to be poster child example for a collection (album) to collection-element (media item) relation which could have made use of the hierarchical character of URI paths. OpenSocial however requires the client developer to use two different URI templates. (Table 2)

A not so small change to OpenSocial would be to either use already standardized and registered media types where possible or to register new types where necessary. It seems that there are some already existing media types that could be a good fit for OpenSocial but only miss a canonical json representation for easy consumption by javascript applications. These are vCard for persons,[21] ATOM entries[NS05] for messages, activities and media items and ATOM categories, collections or workspaces[Gh07] for albums and groups. ATOM and vCard both also provide extension mechanism.

OpenSocial even referenced ATOM for some time as a wrapper format for its own data structures. This was however done in such a way that it only added complexity and totally ignored ATOM's own features[hÓ09]. Consequently the newest specification version deprecates any reference to the ATOM format.

In Jan Algermissen's "Classification of HTTP-based APIs"[22], the OpenSocial REST API would actually be "HTTP-based Type I" due to the lack of media types and direct hyper links between related resources. Algermissen writes that this level has the lowest possible initial cost of all HTTP APIs. Or in other words: The OpenSocial specification authors might not have had to invest a lot to come up with this API specification but maintenance and evolution cost may be medium or high.

2.8 Others

The Calendar Access Protocol (CAP)[RBM05] was published in December 2005 about one year before CalDAV and CalAtom. The standard comprises 131 pages. No evidence of any successful implementation could be found[23]. Cyrus Daboo, author of some calendaring

[21] OpenSocial persons are based on portable contacts which in turn borrowed field names from vCards.
[22] `http://nordsc.com/ext/classification_of_http_based_apis.html` (2011-12-08)
[23] One free implementation project `http://opencap.sourceforge.net` (2012-3-5) seems inactive since 2005

standards, attributes the failure of CAP to its complexity[24].

CalAtom and CardAtom build on top of the Atom Publishing Protocol and are therefor discussed in subsection 4.3.

The idea of using Feeds for collection synchronization has also been adapted by Microsoft's FeedSync[25]. FeedSync's most important contribution according to [Sne07b] was the concept of a "tombstone" element to indicate the deletion of entries from a collection. An RFC to standardize the tombstone concept[Sne12] for Atom feeds is currently in the late stages of the IETF standardization process.

[24] http://lists.calconnect.org/pipermail/caldeveloper-l/2012-January/000135.html (2012-01-04)

[25] http://feedsyncsamples.codeplex.com (2012-3-8)

3 Requirements and Analysis

The requirements of the Kolab REST API are derived in this section from the way how Kolab uses IMAP, from the characteristics of the managed data set and the supposed characteristics of typical clients. In addition it is also considered that the API may even be usable as a restful alternative for CardDAV, CalDAV and parts of OpenSocial. The last subsection explicitly lists why some features of WebDAV are not considered as useful requirements for a Groupware API.

3.1 Scope and General Requirements

The software system designed in this work should provide a web based interface for most common interactions with a Groupware system like Kolab. It must obey the constraints of the REST architectural style[Fie00].

Guidelines of the design are:

- The system should be extensible to support different kind of personal information resources like contacts, events, todo items, journal items and free-busy informations.
- "CRUD" operations must be supported: Create, Read, Update, Delete.
- The client must be able to synchronize collections of resources for offline read access and manipulation.
- The design should be considerably "easier" to implement than CalDAV, CardDAV or IMAP as well for the server as for the client.
- The design should reuse existing standards where possible.
- The design should support all client types listed in subsection 3.3.
- The design should support different Media Type representations of resources.

3.2 Replace Kolab IMAP, CardDAV and OpenSocial

Personal evaluation[26] suggests, that many Groupware clients and servers use CardDAV exclusively to synchronize contacts collections, allowing concurrent modifications of individual contacts via optimistic locking (subsubsection 3.7.6). This is in principle also the way how Kolab uses IMAP.

Thus the principal requirement is to support discovery and synchronization of Groupware collections (Adressbook, Calendar) and CRUD operations on Groupware items (Contacts, Events).

A restful alternative for OpenSocial is not in the scope of this work. However since PortableContacts is discussed it makes sense to highlight which features of OpenSocial,

[26] from using, contributing to or evaluating eGroupware, Horde, Kolab, Kontact, Thunderbird

as presented in subsubsection 2.7.1, are accidentally also supported. The possibility of an unified, restful API for CardDAV and OpenSocial should provide further motivation for the presented approach.

OpenSocial is mainly used by short living Javascript browser widgets. A synchronization protocol may therefor not be seen as adequate on first sight. However a restful protocol enables the proper use of the Browser cache so that synchronization may not need to start from scratch on every page load and modern browsers can keep additional meta data or indexes in HTML5 Webstorage[Hic11c].

Nevertheless, the main interaction considered here as an alternative to OpenSocial is not collection synchronization but the discovery of information related to one person, the media belonging to a person and the traversal of the "social graph", i.e., the relations between persons.

3.3 Client Classes and Characteristics

Different kinds of clients should be able to use the API. Table 3 lists exemplary clients whose constraints and characteristics should be respected by the design. The choice of clients and their characteristics is intentionally conservative to cover a wide range of real world use cases.

	Memory	Bandwidth	pref. format	comment
bad HTML5	none	56 kbit/s	JSON	internet cafe
good HTML5	5 MB	1 Mbit/s	JSON	workplace
Mobile Device	512 MB	384 kbit/s	any	Smart Phone, Tabled
Desktop app.	1 GB	10 Mbit/s	any	PIM suite
Server app.	4 GB	100 Mbit/s	any/HTML	intranet application

Table 3: Constraints of different API clients

The first line in Table 3 "bad HTML5" represents a one time browser session in an untrusted internet cafe with a very bad connection, expecting the data in JSON format. This client does not need to be fully supported, but should be considered.

All other clients are expected to be able to cache data from previous sessions and have a fairly good internet connection at least for an initial synchronization session. The second table line "good HTML5" should represent the use cases commonly handled by OpenSocial enhanced collaboration applications. The last line "Server app." could be an intranet crawler or public search engine consuming HTML pages with the ability to parse semantic annotations.

3.4 Data Characteristics

Lacking sources for more accurate numbers, a couple of conservative estimates are made for the size and number of resources in the scope of this work. This guesswork is not

perfect but it provides a rationale for later design decisions (section 4) and outlines their applicability for a concrete use case.

Contacts It is believed that humans have regular social contacts to around 150 people[27]. So an estimation of 1500 contacts in a user's address book should serve most cases.

The average textual data size associated to a contact is expected to be around 840 bytes[28]. 100 kb are enough for an image file to identify a face.

So a collection with a data size of $1500 contacts * 840 \frac{bytes}{contact} \approx 1MB$ should be a usable address book without profile pictures for many users.

Events A very busy person may have 10 events per day. A two years calendar thus contains $2 * 365 * 10 = 730$ events. The core data of an event is estimated to comprise 356 bytes[29]. So a useful calendar collection has a data size of $730 events * 356 \frac{bytes}{event} \approx 0.25MB$

Conclusions The size of full, useful collections of personal information items has the same order of magnitude then the size of a digital image taken with today's smart phones. With the worst case bandwidth from Table 3 the download of a full, uncompressed collection lasts around $\frac{2*1MB}{56kbit/s} \approx 5min$[30]. Even with a drastic data compression of 90% the transfer would still last over 30 seconds. With the next better bandwidth of the mobile device however, the transfer duration, even for the uncompressed case, is already under one minute ($\approx 42sec$).

For all but the first client the storage capacity is large enough to hold at least a few collections.

3.5 Operation Environment

The application is expected to be installed in a Java servlet container like Tomcat or Jetty and to contact a separate storage component. The primarily targeted storage component is an IMAP server with a Kolab conform set of groupware folders. However the design should not restrict the extension to a document database like CouchDB, plain files, relational or XML databases.

3.6 Caching instead of Performance optimization

The system is meant to inherit the benefits of a restful architecture, especially Cacheability. It should therefor be possible to attach separate caching intermediaries for read requests. Rather then concentrating on the performance of the implementation of read requests it

[27] `http://en.wikipedia.org/wiki/Dunbar's_number` (2012-2-29)

[28] estimated average bytes per common fields: id 100, name 30, 2 * address 100, 2 * mail 50, instant messenger 50, 2 phone numbers 15, comments 30, 3 * url 100

[29] field sizes: start 8, end 8, title 40, location 100, free text 200

[30] The factor 2 accounts for field names and syntax elements. Besides other inaccuracies, latency is not taken into account.

should be taken care that the architecture supports external and internal caching and thus avoids to serve the same read request multiple times.

3.7 Excluded WebDAV requirements

This section discusses a couple of features that are not considered as requirements for this work but are features of WebDAV and thus inherited by CardDAV and CalDAV, increasing at least the complexity of their specifications. It is however doubtable whether any CardDAV implementation supports all the following features.

3.7.1 Reports, Filters, Projections

CalDAV and CardDAV define elaborate report, filter and projection capabilities. This work considers reports or search only when an important use case is not implementable without it and existing, well known specifications can be reused.

3.7.2 Access Control

WebDAV defines specific access control semantics and thus imposes those also on CalDAV and CardDAV. This work does not consider access control but relies on HTTP mechanics to take care of those, especially recent efforts like OpenID and OAuth.

3.7.3 Copying and Moving

WebDAV introduces HTTP verbs to COPY and MOVE resources. The usefulness of such functionality must of course be compared to the complexity of the implementation and the drawback of incompatibility to plain HTTP.

It is possible to enhance a restful API with copy and move functionality without extending HTTP. The only requirement is that additional hyperlinks can be attached to the resources[31]. Allamaraju [All10, Ch. 11] proposes "controller resources" that act on POST requests and are linked from the resources they act on. Custom link relations are used to indicate the semantic of the controller resource.

This work therefor does not include initial support for copy or move.

3.7.4 Versioning

WebDAV and therefor CalDAV and CardDAV support the versioning of resources as an extension to the HTTP protocol. Versioning is an important feature for a text authoring system that may have been the main target for the WebDAV protocol. It does however seem to be of little use for the resources considered here. Individual contacts or events are mostly created in one session by one user and not modified in several sessions like text documents.

[31] which is possible through web linking[Not10]

3.7.5 Make Collections

WebDAV introduces the MKCOL HTTP verb to create collections. CardDAV recommends that implementations support this to allow users to "organize their data better". An alternative would be to make use of ATOM categories for grouping[Gh07]. Instead of creating a new (empty) collection the user would thus create a contact resource with a new category. An ATOM service document could then link to a new (virtual, read-only) collection that only contains resources of this category.

The Atom Publishing Protocol does not define how Atom collection resources could be created. Practitioners recommend a pattern wherein collections of collections exist and new collections can be created by posting to the former[32].

3.7.6 Locking

As with Versioning, this feature of WebDAV is not considered. Instead of locking a resource, HTTP supports conditional updates and leaves conflict resolution to the client.

[NL99, sec. 1] provides three rules, formulated as questions, to help deciding whether a protocol should support locking. In the present case, all three rules advise against locking: *The content is mergeable.* Conflicting changes in vCard and iCal resources can be easily presented to the user. An unmergeable resource would be for example an image. *The editing is expected to be localized to isolated points in the document*, e.g., changing just one field in a content or event. And it is required that *the content can be edited while the user is offline.*

[32] `http://www.imc.org/atom-protocol/mail-archive/msg11565.html` (2012-3-7)

4 REST Interactions Design

4.1 Discovery of Collections

An ideal Rest API is accessed by one main URI and all other resources can be discovered by following links. A useful Media type to discover available collections is the Atom Service Document[Gh07, sec. 8]. It contains links to collections organized in workspaces and annotated with meta data.

A Groupware client most likely needs to discern the available collections by the contained resources as to consume and present them with the appropriate user interfaces for contacts, calendar data, etc. A first idea could be to use the Media types declared in the "accept" tag of a collection to identify types of collections. However the specification explicitly states that this tag "specifies a type of representation that can be POSTed to a Collection". If a collection can only be read then no accept tag should be present and thus also not available for interpretation.

A standard conform approach is demonstrated by Google's Data Protocol[33] and by an internal project at IBM[34]. Both use atom categories[Gh07, sec. 8.3.6] to mark the type of atom entries. James Snell proposed a standard URI to identify the semantic of categories[34] but no follow up to this could be found. The use of categories to attach arbitrary meaning, e.g "event type (product or promotion), and its status (new, updated, or cancelled)" to feeds and entries is also recommended in [Web10, p. 200].

To make categories usable for a common Groupware API, the server needs to use a categorization scheme understood by the client. If different clients don't agree on one scheme the server could still support several.[35]

An alternative Media Type to Service Documents in JSON could not be found. The most promising approach seems to list available collections in a `application/vnd.collection+json` representation. (subsubsection 7.1.3)

4.2 Personalized Service Documents

For a Groupware that manages confidential information it would make sense to provide personalized Service Documents for authenticated users that list only collections that the user is authorized to read.[36] Personalized Service Documents for different users should have different URIs to make them cacheable and to acknowledge that each personalized Service Document is indeed an individual entity. This however conflicts with the previous goal of using one unique Service Document URI as entrance to the API. A solution would be to require the user to authenticate when requesting the unique entrance URI and to

[33] `http://code.google.com/apis/gdata/docs/2.0/elements.html` (2012-2-28)

[34] `http://www.imc.org/atom-syntax/mail-archive/msg18208.html` (2012-2-28)

[35] As a last resort a client could of course also fetch the feeds and identify the media types of the included media entries.

[36] For this use case it would be convenient, if HTTP would support optional authentication, but it does not or only poorly. `http://computerstuff.jdarx.info/content/optional-http-authentication` (2012-2-28)

```
<atom:category
    scheme="http://schemas.google.com/g/2005#kind"
    term="http://schemas.google.com/g/2005#contact" />

<atom:category
    scheme="http://ibm.com/oa/type"
    term="task" />

<atom:category
    scheme="http://www.w3.org/2005/Atom/Entry-Kind"
    term="http://schemas.google.com/g/2005#contact"
    label="Contact" />

<atom:category
     scheme="http://www.w3.org/2005/Atom/Entry-Kind"
     term="http://ibm.com/oa/type#task"
     label="Task" />
```

Listing 1: ATOM categories as used by Google and IBM to mark entry types and a proposal to use a standard scheme URI for type terms

answer with a HTTP code "307 Temporary Redirect" to the user's personalized Service Document after successful authentication.[37]

4.3 Atom Publishing Protocol

The idea to not only use Service Documents but the complete Atom Publishing Protocol as the foundation for a Groupware API is not novel. Rob Yates described this idea under the titles "CalAtom" and "CardAtom" already in 2006[38].

The CalAtom[Yat07] proposal uses a "features" tag and associated IANA registry to mark collection types and their features. But the examples of category usage above (subsection 4.1) and the availability of OpenSearch for time range searches (subsection 4.8) provide confidence that a new tag is not required. The features tag was proposed in 2007 by [Sne07a] but did not become a standard.

The Atom format is also used by the Google Data Protocol to publish contacts, events and other data types[39]. Google's use of Atom however is a bit special. The resource data is not included in the content tag of an entry. Instead a new namespace is used to put the data with additional tags directly inside the entry tag[40].

[37] Alternative all Service Documents could be served under the entrance URI with different HTTP Content-Location headers[FGM+99, sec. 14.14]. In that case the personalized Service Document must however also be available at the indicated location.

[38] `http://robubu.com/?cat=2` (2012-3-2)

[39] `http://code.google.com/apis/gdata` (2012-3-2)

[40] `http://web.archive.org/web/20081120001246/http://www.snellspace.com/wp/?p=314` (2012-01-05)

4.4 Synchronizing Collections

If a Groupware client can synchronize an entire collection to its local memory, then there is no need for more sophisticated queries that provide only a subset of the collection. The client can answer all queries from its local copy of the collection.

In subsection 3.4 it has been shown that the time necessary to synchronize a full collection is under one minute in most cases. This should be acceptable for an initial synchronization that is only done once on rare occasions when a desktop machine or mobile device is first used. If subsequent synchronizations only transfer a few resources, that have changed since the last synchronization then such updates can be made in the order of a second.

All client scenarios except of a Web Browser client that is used only once, can profit from the above scenario. In such a case other interaction patterns need to be used (subsection 4.8).

The Atom Publishing Protocol identifies collections of resources as Atom Feeds. Feeds can also be used to synchronize collections. The necessary ingredients are the link relation "next"[Not07], the concept of a "deleted entry"[Sne12] and the prerequisite that the feed entries must "be ordered by their "app:edited" property, with the most recently edited Entries coming first in the document order"[Gh07, sec. 10].

The API server design has the notion of a logical feed that can be split up in multiple real Atom feeds linked with the relation "next". Updated or new entries are always inserted as first element of the first feed since their "app:edited" property is the most recent. Inserting a new entry at the top of a feed can lead to entries at the end of the feed being pushed to the subsequent feed. This push needs to be atomic such that a client loading subsequent feeds may see an entry twice, at the end of a previous feed and the top of the next feed, but will never miss an entry in this scenario.

In the case of an initial synchronization, the client loads the initial feed and all subsequent feeds linked with the "next" relation and adds all Resources associated with the feeds entries to its local storage. Resources can either be included completely in the content tag of an entry or be linked to by the entry. The client memorizes the "app:edited" value of the first entry of the first feed for subsequent synchronizations.

It is possible, that the collection has been modified during the synchronization. Therefor the client should directly conclude with an update synchronization. This means that the client starts again to load the first feed and applies all updates until it sees an entry with an "app:edited" value older then the one memorized from the last synchronization. It is possible that the client must follow several "next" links or even load all feeds in the extreme case.

If the client followed a "next" link during a synchronization then it will make sure at the end of the synchronization that the first feed has not changed meanwhile most probably with a conditional GET request. After this last request indicates no further changes the client knows that its local collection is in the state of the servers location at the time of the last GET request.

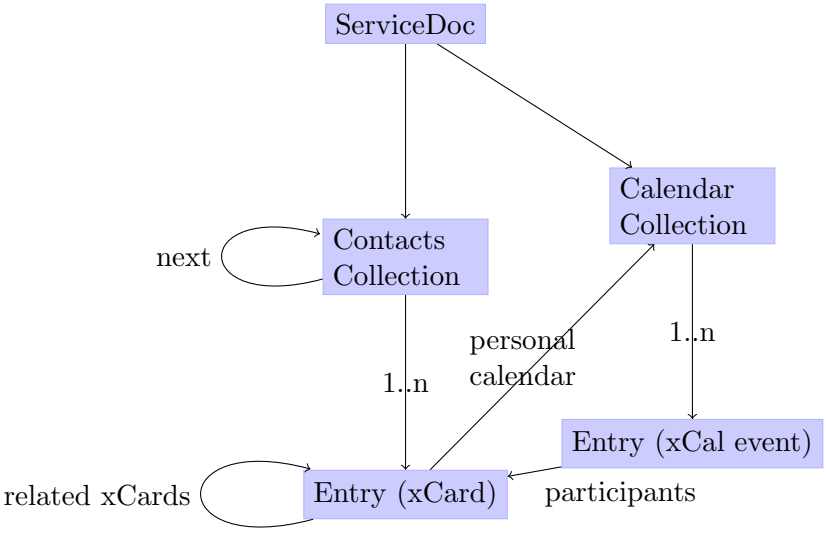

Figure 1: Discovery paths from the Service Document to individual Groupware Resources

4.5 Efficient Synchronization with HTTP Delta encoding

The synchronization method described in subsection 4.4 can be enhanced to reduce the bandwidth usage and general resource usage of both client and server. The necessary extension has been described in [Wym04] and is commonly referred to as RFC3229+Feed since it adds a feed specific Instance Method (IM) to the "Delta encoding in HTTP" standard[MKD+02]. Unfortunately nobody has yet invested the effort to drive this method through a formal standardization process[41]. It is however reported to be widely implemented[42], even in the popular Microsoft Internet Explorer[43] and the author claims substantial bandwidths saving opportunities [44].

The idea of delta encoding is that a server can respond to conditional GET requests with only a small, special patch. The client applies the patch to its cached representation of the requested resource which results in the new version of the resource. All currently IANA registered IMs are byte oriented[45]. These methods however don't add substantial benefit for the case of synchronization feeds.[46]

In the case of the proposed feed IM, the client sends a conditional GET to request the synchronization feed but indicates in the "A-IM" header that it understands the feed IM, as shown in Listing 2.

[41] `http://bob.wyman.us/main/2006/04/microsoft_to_su.html` (2012-3-9)
[42] `http://www.wyman.us/main/2004/09/implementations.html` (2012-1-6)
[43] `http://blogs.msdn.com/b/rssteam/archive/2006/04/08/571509.aspx` (2012-3-9)
[44] `http://wyman.us/main/2004/10/massive_bandwid.html` (2012-3-9)
[45] `http://www.iana.org/assignments/inst-man-values/inst-man-values.xml` (2012-3-9)
[46] Byte oriented IMs might however be very beneficial to serve updates of xCard/xCal resources if only one or a few fields changed.

```
GET /api/collections/contacts HTTP/1.1
Host: bar.example.net
If-None-Match: "3631-@2147483647"
A-IM: feed, gzip
```

Listing 2: HTTP GET request using feed delta encoding

The server responds with a valid feed including the normal head elements but can use the etag from the "If-None-Match" header to include only those entries in the response, that have changed since the time when the etag was valid. This implies of course, that the server is able to match the given etag to a corresponding list of changes[47]. The server response uses HTTP code "226 IM Used"[MKD+02] to mark the response as a special one that is not the regular, cachaable representation.

It is advisable to also include a "next" link to the subsequent feed to keep compatibility with the synchronization process from subsection 4.4 and prevent the client from accidentally considering the returned feed to contain the full collection. The "next" link however would probably cause the client to unnecessarily follow it, since it has not yet seen an entry with an old enough "app:edited" value. The server could include additional entries from its database to satisfy the client's terminating condition. Or the server could include an artificial, minimal deleted-entry[Sne12] tag with a non-existent ref value and a when value just older then the etag sent by the client:

```
<at:deleted-entry
   xmlns:at="http://purl.org/atompub/tombstones/1.0"
   ref="tag:example.org,2005:NONEXISTENT"
   when="2005-11-29T12:11:12Z"/>
```

The above precautions not to break the client's synchronization logic is necessary to permit the server to also respond to RFC3229+Feed requests with paginated feeds in cases where more entries have changed then the server is comfortable to include in a single response.

4.6 Media Entries and the content tag

The Atom format provides the opportunity to include a full representation of a resource in the content tag of an entry[NS05, sec. 4.1.3]. It is thus possible to embed complete xCard or xCal resources in the Atom feed and so to relieve the client from issuing many GET requests for each individual resource.

The benefit of saved GET requests must be balanced with the possible disadvantage of serving the client resource representations already seen. A client that does regular updates may probably be interested only in the first one or two entries of a feed while the server might have made the effort to produce tens of entries.

On the other hand the Atom Format mandates that an entry without embedded content must provide a summary element. It may not make much of a difference in bandwidth

[47] The given example already suggests that the etag itself could include database ids or timestamps.

and processing whether a summary is produced or the full content is provided.

Different optimization strategies are possible here, e.g.

- The first feed in a sequence of paged feeds could contain only very few entries to optimize for regular updates and have more entries in all following feeds.
- The server could remember the entries already consumed by an authenticated client and serve only new entries in the first feed.

In any case it is mandatory that a client can handle embedded content as well as linked content.

4.7 Modifying Resources and Offline editing

Editing, Updating and Deleting of media entries is specified in the Atom Publishing Protocol and is useful for this work without modifications.

In addition to the normal online workflow, a client should offer the user the possibility to create, update and delete resources while being offline and to apply this modifications during the next synchronization, much like the IMAP protocol used by Kolab. This requirement is trivial to fulfill as long as no concurrent edits happen on the server site. In that case the client just PUTs the changes the next time it is connected.

In the case of edit conflicts however, the client needs to perform an automated or user assisted merge of the conflicting resources. Therefor the client should always preserve a copy of a resource version as last seen from the Server to be able to perform a three-way-merge.

The problem of offline edits and conflicts is thus similar to the case of a failed conditional PUT request due to a concurrent edit. [NL99] describes this case and resolutions in detail.

4.8 Special Reports, Queries, Search

In few cases it may not be feasible for a client to synchronize a full collection, e.g. due to low bandwidth. This section explores restful ways to let the client request only a subset (selection) of a collection. More specifically the client should be informed about possible query facilities without relying on out-of-band information.

A promising approach is to use the de-facto standard OpenSearch[Cli]. According to its homepage it is implemented by most major browsers, search engines and many other sites. OpenSearch is also recommended for the link type “search” in the HTML5 standard[Hic11b, sec. 4.12.4.12]. The default format of an OpenSearch result list is an Atom (or RSS) feed.

OpenSearch defines the (not yet IANA registered) media type application/opensearchdescription+xml, which provides necessary information for a client to perform queries against a search service. Since possible search queries are usually unlimited it is not possible anymore to provide a set of static links. Instead the server provides an “URI Template”[GFH+12] that instructs the client how to perform an “URI construction”[48].

[48] OpenSearch is the older standard and referenced as Level 1 URI Templates in [GFH+12].

The basic OpenSearch standard defines a simple full text search. Thus a user could search contacts by name, address or any other field value. Equally events, todo items or notes could be searched by keywords.

The next important use case is to show calendar events in a given interval, e.g. to present the events for a month, week or day. This can be achieved with the OpenSearch Time extension that provides the temporal start and end parameters. Rob Yates CalAtom[Yat07] proposal included a similar time range search as the only but mandatory special report.

Probably useful might be the OpenSocial Geo extension. It could allow to search contacts or events in a given geographic region. Even more search types become possible with the SRU extension that wraps the "Search/Retrieval via URL" standard with its "Contextual Query Language" (CQL)[49]. The latter provides the possibility to sort result sets which might be interesting to present an address book sorted by names.

Search result Atom feeds can make use of annotated HTML (subsection 5.2) in the summaries of entries and should not embed full resources in the content tag. Thus the client can still provide a structured view of the data, like calendar views or a tabular contacts list without the need to transfer full representations.

The OpenSearch specification suggests that links to the OpenSearch Description Document for an Atom feed might be added inside a feed tag. There is however no reason not to add such a link inside the collection tag of a Service Document. This allows a client to directly search a collection without the need to get the feed first.

[49] `http://www.loc.gov/standards/sru` (2012-3-1)

5 Other Design Considerations

5.1 Media Type conversion and non-isomorphism

Two media types are non isomorphic, if at least one of them can express information which the other could not express. For example the vcard media type defines many property parameters that have no equivalent in portable contacts, like language, altid or sort-as. So a conversion of a vcard into portable contacts will most likely lose this data.

This data loss could first be a problem when a client receives a representation. However since the client negotiated the media type with the server it is most likely that it is satisfied with only the data representable in that data type.

Now if the client uses such a media type in a put request to update a resource, it may not be clear how to deal with the information the client could not express in the submitted resource. Should it be deleted or merged with the new representation?

Different strategies are possible in such scenarios and must be selected for the individual use case:

1. The server accepts updates only for one media type while serving other media types in a "read-only" mode.
2. The server accepts PATCH requests[DS10] as a compromise while still not accepting certain media types for updates (subsubsection 7.1.2).
3. The implementer decides to either merge or deletes information not representable in a received media type and lives with the consequences. In the case of contact information this can be a valid strategy since the most essential information is representable in all media types. The server practically only works with data in the intersection of all supported media types.
4. Available facilities to extend media types are used to establish isomorphism. Vcard for example allows the addition of arbitrary properties prefixed with "x-".
5. The server implements version control so that the situation can be resolved manually later.

The creation of resources can be handled more liberate then updating, since no state on the server exists that could be lost.

5.2 Microformats, Microdata, RDFa

HTML documents are primarily meant to be rendered by browsers and interpreted by humans. It is hard for a machine to interpret the meaning of text and data included in an HTML document. To help this, different techniques have evolved to add additional meta data to HTML that allows machines to identify structured data in HTML without having an impact on the rendering. The most popular ones, Microformats, Microdata and RDFa, are presented and discussed in [Ten12].

There is not yet an established term to refer to the three different formats. Practitioners use "structured data languages"[Spo11], "machine-readable data format"[Hic11a], "structured data markup"[GG11] or just "structured markup". Scientific publications seem to use the term "Semantic annotation"[RGJ05] to refer to HTML with machine readable semantic data. This work will use the term "Semantic annotation format" to refer to Microformats, Microdata, RDFa and similar formats.

5.2.1 Use Cases

One major use case for semantic annotations is to help search engines to better index the annotated site. The Microformats project was started by a blog search engine (Technorati)[Çe06] and the recent schema.org effort came from the three big search engines Google, Bing and Yahoo.[GG11] Another use case is demonstrated by the Firefox plugin "Operator".[50] It allows to extract annotated entities from web pages. A user could thus import contact or event data from arbitrary web pages in his personal information manager with one click[51]. Semantic annotations can also be used to make web content accessible to disabled people[YSHG07].

A third use case is currently under development as part of the European Union Research Project "Interactive Knowledge Stack" (IKS) that builds a semantic content management stack. The sub-project "Vienna IKS Editables" (VIE)[52] uses semantic annotations to make content on a web site editable. It does so by searching the HTML document for semantically annotated entities and dynamically building editing interfaces for those. A modified entity can then be sent to the server via AJAX in a format called "json-ld" that serializes semantic data to JSON.[53] For a Groupware, this editor could be used to automatically create HTML forms instead of creating them on the server site.

In the context of this work, semantic annotations could be used inside the `summary` tag of Atom entries, as shown in listing 3. A consumer of a feed of contact elements could thus use the data extracted from the annotated summary data to provide a tabular overview of the entries even without fetching the associated media resource of the entry, which is especially useful for search results as discussed in subsection 4.8.

5.2.2 Format selection

With at least three different semantic annotation formats, a developer needs to decide which to implement. It is possible to implement multiple formats in parallel inside the same HTML document, but this means more markup and a more complex publishing task[Ten12]. This choice is not a choice of different Media Types, but a choice in the scope of the Media Type text/html (or application/xhtml+xml).

[50] `https://addons.mozilla.org/en-US/firefox/addon/operator/` (2012-2-20)

[51] Apparently, Android phones can import annotated addresses from web pages directly to their address books.

[52] `http://www.iks-project.eu/projects/vienna-iks-editables` (2012-2-20)

[53] `http://json-ld.org` (2012-2-20) the iana registration of the mime type `application/ld+json` is currently discussed

```
<summary type="html">
  <div itemscope itemtype="http://schema.org/Person">
    <a itemprop="url" href="www.maxpattern.name">
      <div itemprop="name"><strong>
        <span itemprop="givenName">Max</span>
        <span itemprop="familyName">Pattern</span>
      </strong></div>
    </a>
    <div itemscope
         itemtype="http://schema.org/Organization">
      <span itemprop="name">
        Andorian Mining Cooperation
      </span>
    </div>
    <div itemprop="email">some@mail.com</div>
    <div>
      <meta itemprop="birthDate" content="1970-01-02">
      DOB: 01/02/1970
    </div>
  </div>
</summary>
```

Listing 3: Microdata used in the summary of an ATOM entry summary (markup not escaped for clarity)

A first consideration has to be the ability of expected consumers to handle the format, a second consideration the available tooling to produce a particular format. The different Semantic annotation formats impose certain requirements for the used HTML dialect. Microformats can be used with all versions of HTML, RDFa with XHTML or HTML5 and Microdata introduces special attributes that work only with HTML5[Ten12].

Microdata is part of HTML5 and a standard effort of the W3C[Hic11a]. It is also backed up by the schema.org effort of Google and Microsoft.[54] The schema.org vocabulary in turn has been mapped to the semantic world by researchers working on linked data.[55] Thus by using Microdata with the schema.org vocabulary, the data can easily be combined with other semantic data. The rest of this work therefor concentrates on Microdata. Many good arguments to also consider RDFa can be found in the blog of Manu Sporny[56], chair of the RDF Web Applications Working Group at the World Wide Web Consortium.

5.3 HTML Forms

A web based user interface for a Groupware today has many means to provide data editing and submission facilities thanks to powerful Javascript libraries like the VIE Editor (subsubsection 5.2.1). The traditional, standardized and most compatible way however is

[54] http://schema.org/docs/gs.html#microdata_why (2012-2-17)
[55] http://schema.rdfs.org/about.html (2012-2-17)
[56] http://manu.sporny.org/category/rdfa/ (2012-2-20)

the use of HTML forms. Unfortunately these lack a few features that could improve their use for restful systems.

HTML has no means to send an etag when submitting a form and no support for other HTTP verbs then GET and POST, most importantly PUT and DELETE. A Discussion to include these however seems to be underway[Amu11c].

All forms have the same media type of application/x-www-form-urlencoded, although they may represent totally different kind of resources. In practice this is often not a problem since the server knows which form to expect and selects its parsing routine accordingly.

In cases, where different forms can be expected to be submitted to the same URI, e.g. to the URI of a collection, the server needs to be informed about the resource type, probably by a hidden form input element.

The manual creation of HTML forms and associated form parsers and validators is involved and error prone. Therefor many approaches and implementations exist to automate this task. If a machine can work on an existing data model for the resource, then this can be used as basis for the automation.

Not yet answered is the question, where or under which condition an HTML form should be submitted to the user. It is certainly not desirable to present HTML forms by default in every HTML representation of any resource that the user is authorized to modify. And even then, there would still not be any mean for the user to reach an HTML form to create a new resource.

The edit case can be solved with the IANA registered "edit" link-relation. An HTML page representing an editable resource could just use a link element for the purpose of signaling the client the location where it can retrieve an editable resource:

```
<link rel="edit" href="?edit=true"/>
```

The above link is a relative Link that just appends a query part to the URI of the current page (assuming that the page URI does not already contain a query part).

It may be noted here, that making different representations of one resource available under different URIs is no violation of rest principles and even encouraged for similar use cases by [Ram06].

Unfortunately no link relation is standardized to retrieve an empty form for the creation of a new resource.[57] So for the time being server and client would need to agree on a custom link relation for that purpose. The empty form can be regarded as an entity of its own, linked from the collection where newly created resources are expected to appear.

5.4 VCard's (social) network properties

This section investigates, whether the VCard Mediatype is also usable to represent basic concepts of a social network (OpenSocial) in a restful way.

[57] The `Collection+JSON` Mediatype solves this issue by providing templates for new items inside the collection representation (subsubsection 7.1.3).

Important concepts of OpenSocial are persons, their relations, their media and groups of persons (subsection 2.7). The normal discovery path of OpenSocial starts with a person, probably the authenticated user.

It is assumed, that the client has somehow (e.g., by redirection after authentication) reached a VCard of a person. At this point the following defined properties of a VCard can provide useful information for a social network:

- The RELATED property expresses a typed relationship to another entity. The possible types have been adopted by the "XHTML Friends Network" project[58] (contact, acquaintance, friend, met, co-worker, colleague, co-resident, neighbor, child, parent, sibling, spouse, kin, muse, crush, date, sweetheart, me) and augmented with agent and emergency.
- PHOTO contains or links to a photo of the person.
- IMPP provides a reference to contact the person via instant messenger. No website built-in chat (Facebook) is required.
- URL is an untyped reference to any website associated with the VCard, intended for "personal web sites, blogs, and social networking site identifiers".
- [GLLM11] proposes additional VCard properties explicitly for social network information[59]. It includes an ALBUM property to link to a collection of media items belonging to the person.

VCards can not only represent persons but also groups. A VCard with the KIND property set to "group" may include a MEMBER property listing URIs for its members. Thus group VCards with dereferenceable MEMBER URIs could be used to list the members of an OpenSocial group. The client would however still need to load every member's VCard to display the names or photos of the group members.

Unfortunately the VCard type does not define an inverse link relation "GROUP" to link to a group that a person is a member of. VCard does not even have a universal link property like HTML or ATOM which could be used for that purpose.

The recent Web Linking standard[Not10] can help in this situation. It allows the inclusion of links as HTTP headers explicitly for Mediatypes without the ability to include links. The link relation type collection[Amu12] can then be used to link to all group VCards related to a person.

It would of course be preferable, if VCard would be augmented with a property to link to the groups of a person, maybe as part of the social network extension draft[GLLM11]. This situation however also shows a limitation of the way how the type of hyperlinks is indicated in VCard. VCard contains a lot of properties, that can contain URIs. The

[58] `http://gmpg.org/xfn/11` (2012-03-29)

[59] The draft has been abandoned in March 2012 due to lack of interest from social network sites. `http://www.ietf.org/mail-archive/web/vcarddav/current/msg02509.html` (2012-03-29)

type of those URIs then depends of the containing property type like PHOTO, IMPP, RELATED, etc.

VCard however lacks a universal LINK property that could refer to the IANA link relation registry[60] to indicate its type and thus directly benefit from the semantic standardization efforts underlying this registry.

PortableContacts also contains fields equivalent to RELATED, PHOTO, IMPP, URL, also misses an ALBUM field but it can not represent a group.

In conclusion, VCard seems to be a viable option to represent social network properties and structure if header links are used or with the help of two not yet standardized properties ALBUM and GROUP.

[60] `http://www.iana.org/assignments/link-relations` (2012-03-29)

6 Implementation

6.1 Control Flow Overview

Figure 2 outlines the most important classes for the control flow. The implementation relies on the JAX-RS[HS09] implementation Jersey to route calls to the four different "Jersey Resources" classes, representing Atom Service Documents, Atom Collections, Atom Entries and Media Resources of different Mediatypes.

The Jersey Reader/Writer providers are called by Jersey to transform in- and output for the Resource classes. The `AbderaWriterJerseyProvider` just uses the Abdera library. The other two providers are special because they work on the universal `Resource` class or the related `UnparsedResource` class. These classes represent the concept of resources that can be represented with different Mediatypes. They are discussed in detail in subsection 6.2.

One instance of the CollectionStorage interface is responsible for the administration of one collection of Resources. The first four methods implement CRUD functionality and the `listUpdates` method provides a partial, time ordered list of updated resources, including special Resources to indicate deletions. This list can be directly transformed in a corresponding AtomPub feed.

Like the other classes, the CollectionStorage deals with the universal `Resource` class. The Precond(itions) parameter is a wrapper class around the corresponding HTTP headers[61]. It provides `shouldPerform(etag, updated)` bool: methods that the storage must call with the resource's etag, last update timestamp or both. The CollectionStorage indicates with each methods return value, whether it actually performed any action. The GetResult and ResultList classes are simple tuple classes wrapping one or multiple `Resource` instances in case the Preconditions failed or otherwise an indication that a "304 Not Modified" response should be returned.

6.2 Resource handling

The `Resource` class is the generalization of a restful HTTP resource without a binding to a specific Mediatype. This corresponds to the distinction between a resource and its representations in Fielding's Dissertation[Fie00, sec. 5.2.1.1]. A representation in a certain format is a property "selected dynamically based on the capabilities or desires of the recipient and the nature of the resource"[Fie00, p. 87].

6.2.1 Resource properties

The interface of the `Resource` class marks the border between code that handles the control flow of an HTTP request, as outlined in Figure 2, and code that provides information about properties of a resource (subsubsection 6.2.3). In the context of this work, four different kind of properties of resources are distinguished:

- essential administration properties: unique Id, last update time, HTTP entity tag

[61] If-Match, If-None-Match, If-Modified-Since, If-Unmodified-Since

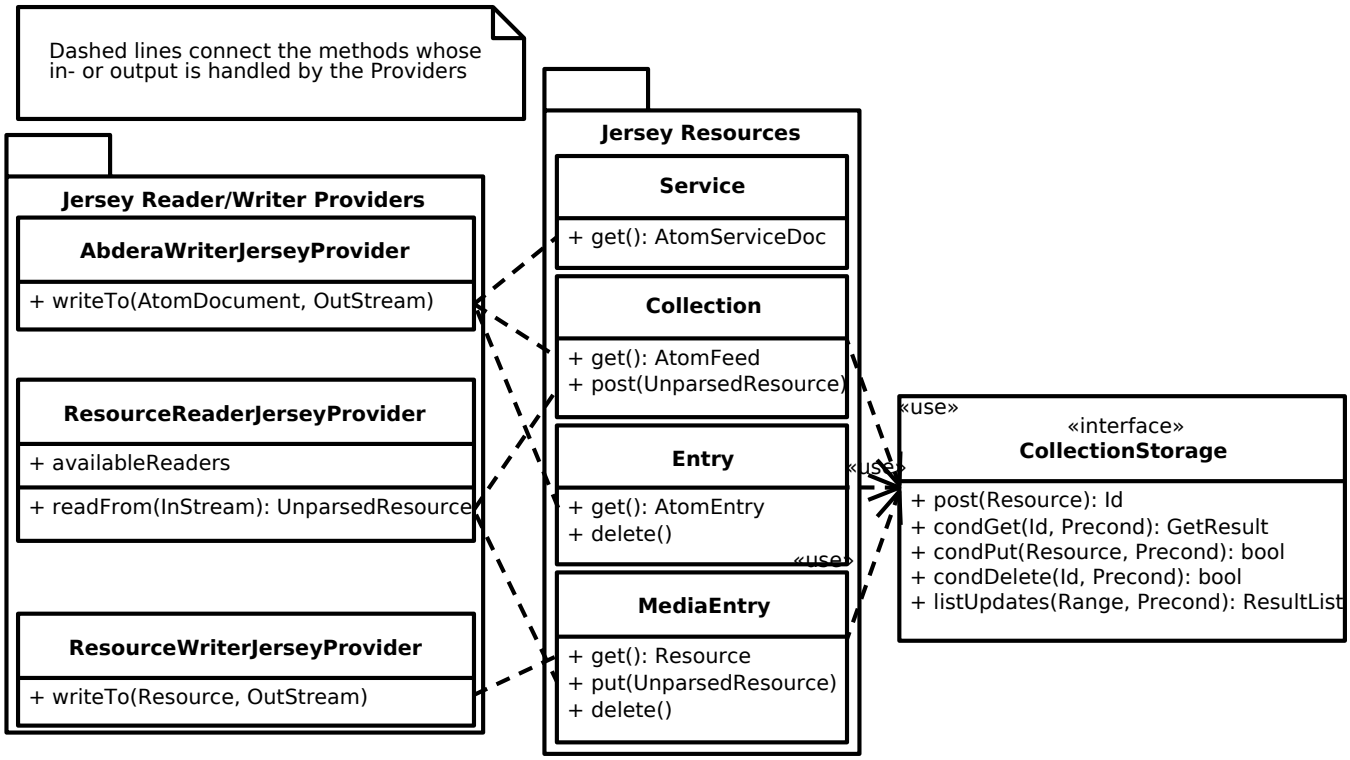

Figure 2: main classes controlling the execution flow

- generic meta properties: title, summary, author
- a Mediatype independent interface corresponding to the concept represented by the resource e.g., a person, location, event, product, ...
- a mediatype specific serialization (representation) of the resource

The id and update time are required for the synchronization protocol outlined in section 4. They do not depend of the nature or content of the resource and are attached to the resource by code outside of the `Resource` class. The entity tag must differ for each new version of a resource. The code to produce and check a resource's entity tag should be adjusted carefully with the concrete CollectionStorage implementation to ensure efficient processing of conditional HTTP requests.

The generic meta properties of the resource can be used to fill the corresponding tags of an atom entry. They can either be extracted from a meaningful property of the resource or be provided to the resource. E.g., the author property could be extracted from the meta data of an image file (EXIF), set to the organizer of an ical event. The title of a contact resource in the implementation is set to its full name and email. The summary also contains the address and phone number.

One use of two mediatype independent interfaces or "facades" or a resource is exemplified in Figure 3. The `Contact` interface is implemented by two classes that can extract the necessary information from either a `VCard` or a `PortableContact` instance. The `Contact` interface in turn is used by an implementation of the `TitleAndSummary` interface. The `PlainTextTitleAndSummary` class in comparison does not work on an intermediary interface but directly on the original data structure.

The above mechanism is exposed by the `getFacade(Interface)` method of the `Resource` class and used to retrieve a `TitleAndSummary` facade in order to build an Atom Entry. The code using the facade does not need any further knowledge about the type of resource it is working with.

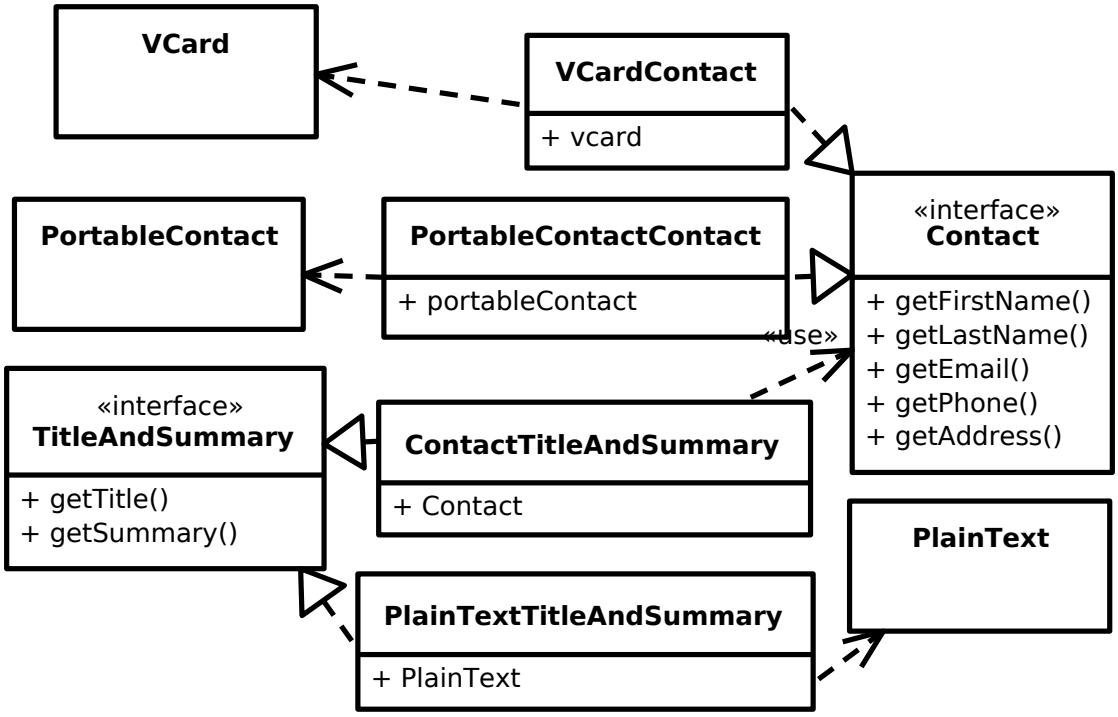

Figure 3: Dependency of facades of resources to provide the TitleAndSummary interface

The serialization property is exposed by the asMediaType(MediaType, OutputStream) method of the `Resource` class. Internally this method also uses the `getFacade()` mechanism to requests an instance of a `Writer` interface with the additional constraint that the writer must produce the requested Mediatype (subsubsection 6.2.3). Accordingly the `ResourceWriterJerseyProvider` of Figure 2 is trivially simple: It just calls the asMediaType method of the provided resource. The Resource is responsible for providing a mediatype specific representation of itself.

The Resource class outlined in this section does not correspond to the equally named resource class concept in JAX-RS[HS09]. The JAX-RS resource classes in this work are

found in the "Jersey Resources" package of Figure 2. But they do not really represent resources but rather the binding of resources to URIs and their processing logic.

6.2.2 Resource life cycle

Resource classes in this work have a four staged life cycle. The first stage is represented by the `UnparsedResource` class, instantiated by the ResourceReaderJerseyProvider class for post or put requests. In this stage the Resource has already been assigned an appropriate `Reader` implementation according to the Content-Type request header but it has not yet received an Id and update timestamp.

The `Resource` class represents the second stage, a parsed request body with an Id and update timestamp. Only in this stage the getFacade() method can be used.

Once a resource has been deleted, it does not vanish entirely, but enters stage three. Only the associated data originally submitted in the request body is discarded but the Id and update timestamp (now referring to the time of deletion) is preserved. Such a "deleted resource" is used to generate the "deleted-entry" entries in the AtomPub feed (subsection 4.4).

Deleted resources don't need to be preserved eternally. A deleted resource with the oldest timestamp of all resources managed by a particular CollectionStorage can be safely purged completely (stage four). A client that synchronizes the collection can still infer that the resource has been deleted since it is no longer included anywhere in the list of updates. Repeated application of the above rule makes sure that the last element of the updates list points to a "living" Resource of the second stage.

6.2.3 Resource Facades

subsubsection 6.2.1 introduced and motivated the concept of Resource Facades. This section explains the inner workings of the classes providing this mechanism as drafted in Figure 4.

The Resource class does not hold any attribute that directly corresponds to its "main data". Instead it holds a FacadeProvider instance to request a specific data facade to access data. Facades are primarily referenced by Java interfaces.

A FacadeProvider in turn is instantiated with a FacadeRegistry of available FacadeFactories and one or more "seed" facades, making up the initial content of the resolvedFacades attribute. An instance of FacadeFactory is capable of building one specific facade object and has a set of dependencies needed for that purpose. Therefor the concrete FacadeFactory used to build a facade and thus the resulting implementation of the facade interface depends on the facades already available in the resolvedFacades attribute of the FacadeProvider.

In the example of Figure 3, the TitleAndSummary interface is implemented by two different classes. The factory responsible for the PlainTextTitleAndSummary would declare a dependency on a PlainText facade. The factory producing the ContactTitleAndSummary declares a dependency on a Contact facade, which can again be produced by two

different factories with their dependencies finally pointing to the "root" facades.

Facades already resolved are added to the resolvedFacades attribute of the FacadeProvider to speed up future facade requests. This is possible since facades are required to only provide read access to the data. Any manipulation of the Resource should result in a new Resource instance thus reflecting the REST characteristic that Resources are manipulated by the submission of new Representations.

Requests for facades can be further parameterized with a Predicate. The Predicate has one apply(FacadeFactory):bool method which is called only for FacadeFactories producing the desired interface. This mechanism is used in the implementation to check an `isWriteable(MediaType)` method on factories producing Writer instances and thus to select the correct Writer according to the Mediatype accepted by the client. Future work could considerably enhance this rather brittle mechanism e.g., to check for annotations on the class produced by the factory.

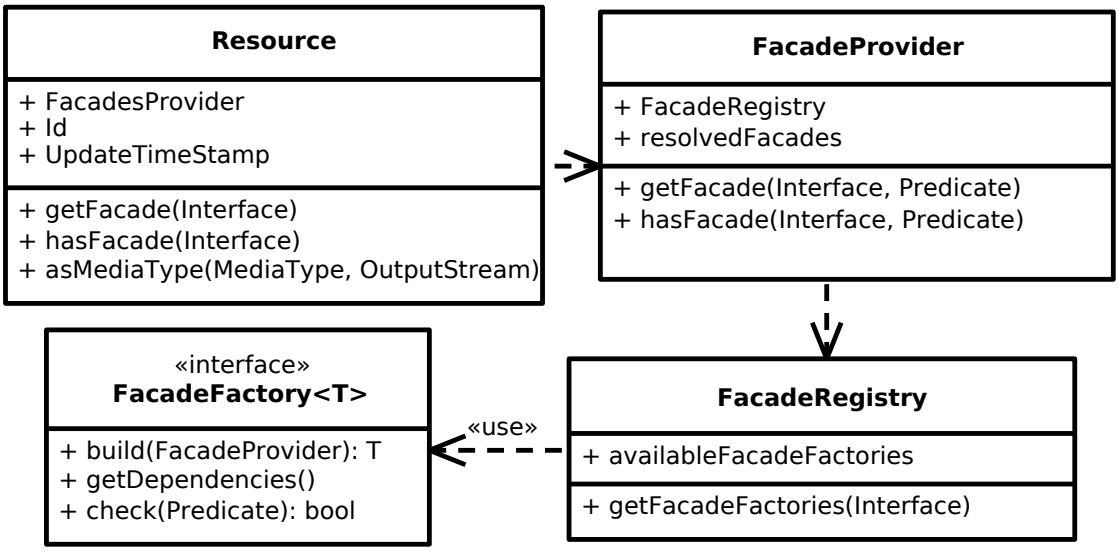

Figure 4: The API of the Resource Facades mechanism

The Resource Facades concept is implemented only as a proof of concept. subsubsection 7.1.5 outlines a lot of future work that might be worth of consideration in this direction.

6.3 CollectionStorage

The CollectionStorage interface[62] is intended to be implementable for a variety of persistency providers like relational databases, the file system, document databases like

[62]The interface may be further broken down in a read-only and a write part which has been avoided for this overview.

```
ResponseBuilder rb = request.evaluatePreconditions(etag);
if (rb == null)
  return doUpdate(foo);
```

Listing 4: Potential lost-update problem with JAX-RS

CouchDB, MongoDB, eXist. In this context the IMAP folders used by Kolab are just a kind of document store.

A CollectionStorage is instantiated with the knowledge of the collection for which it is responsible. Each collection managed by the application corresponds to a separate CollectionStorage instance. Several CollectionStorage instances might however share the same underlying connection to a persistency provider.

The only uncommon requirement for the persistency provider is to provide the time ordered list of updates. This list could be thought of as just another kind of search or report like the full text and time range search defined by OpenSearch (subsection 4.8). A search index library like Apache Lucene[63] can provide indexes for full text search on text fields, time range on events or a list of all documents ordered by an "updated" field.

The IMAP based persistency of Kolab does not provide the described indexes[64]. Those must therefor be implemented by a separate component.

To make implementation of the interface easy and to correspond to the REST characteristic that every request is atomic,

The CollectionStorage does not expose any support for transactions. This should make the interface easier to implement and also corresponds to the REST characteristics of statelessness and transfer of full representations. As a consequence, the check for HTTP preconditions must be made inside the CollectionStorage. Otherwise an update could be lost as demonstrated in Listing 4 from the JAX-RS specification[HS09, p. 28]. In this example a concurrent update by a separate HTTP request that would happen between the etag check and the `doUpdate` call would be overwritten.

6.4 Dependency Injection

6.4.1 Preparsed Request Components with Dependency Injection

It seems like an obvious fact that could not be further deduced, that any response action to a request must be preluded by a parsing of the request. In the case of a REST application this parsing could be further divided in two steps:

1. Parse URI, Accept Header and HTTP verb to select the Resource method

2. Resource method specific parsing defined by JAX-RS parameter annotations or performed in the Resource method

[63] `http://lucene.apache.org` (2012-03-26)

[64] The IMAP SEARCH command[Cri03, sec 6.4.4] searches only the message body and headers but not attachments.

```
@Get public Response get(
    @QueryParam("query") String query,
    @QueryParam("sort-by") String sortBy,
    @QueryParam("offset") int offset,
    @QueryParam("limit") int limit ) {
```

Listing 5: Verbosity of parsing Requests with JAX-RS

```
@RequestScoped @Provides
def paginationRange(uriInfo:UriInfo):PaginationRange = {
  val queryParams = uriInfo getQueryParameters
  val offset = intQueryParam(queryParams, "offset", 0)
  val limit = intQueryParam(queryParams, "limit", 20)
  return new PaginationRange(offset, limit)
}
```

Listing 6: Scala Dependency Injection provider for the PaginationRange class; intQueryParam extracts a named query parameter or returns the provided default value

JAX-RS defines only rudimentary support for the second step by means of inflexible annotations. Listing 5 shows as an example the verbosity of parsing a set of standard query parameters for a search interface.

The implementation for this work instead uses value objects and dependency injection to isolate request parsing. This can be seen for example in the PaginationRange class which should just hold the values of the URI query parameters limit and offset. The provider function in Listing 6 is invoked by guice when this class is required. It depends in turn on UriInfo, extracts the necessary information and returns the simple value class PaginationRange.

Other similar value classes in the implementation provide injectable access to the parsed path parameters (PathParam) or conditional request HTTP headers (Preconditions). The main advantages of this approach are supposed to be:

- Classes parsing commonly used query parameters can be reused, even across unrelated applications.
- The request method declaration gets much easier to read.
- Sophisticated validation can be applied without obfuscating the request method.
- Value classes are the right place for additional logic related to the wrapped values. The Preconditions class for example contains the logic to check the If-* headers of a conditional request against the current entity tag or update time of a resource.
- Default values for unspecified input could depend on information only available at runtime instead of being provided as static value to the applications source code.

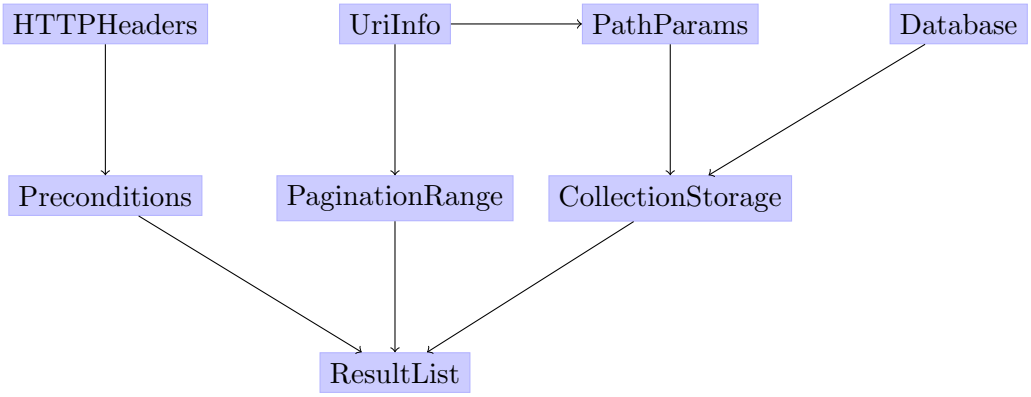

Figure 5: Building a processing pipeline with Dependency Injection

6.4.2 Driving Dependency Injection further

The use of dependency injection can be extended to comprise several levels of dependencies and thus to build processing pipelines. The information from the above PaginationRange class is in the implementation just forwarded to the CollectionStorage's listUpdates method.

Consequently the resource method could as well use dependency injection to directly request the corresponding ResultList instance. Figure 5 visualizes the resulting, hypothetic dependency graph of this approach.

The figure shows how the CollectionStorage relevant for the request is identified by the URI path. It depends of course on some kind of database. The dependency injection is configured to produce a ResultList class by calling the listUpdates method of CollectionStorage with instances of PaginationRange and Preconditions.

The idea might be an alternative implementation of processing pipelines to the one proposed in [DM11], which uses XProc, An XML Pipeline Language. One advantage of the dependency injection approach would be that the processing pipeline can be defined and configured in the same language then the rest of the application.

6.5 Producing Semantically annotated HTML

A recent discussion of possibilities to produce semantically annotated HTML can be found in [CDDM09, sec. 9.1.3]. The authors describe a method developed as part of a larger "Web Semantics Design Method" (WSDM), consists of two mappings. The first one is the "data source mapping (DSM), which describes exactly how the reference ontology maps to the actual data source." The second mapping links HTML tags to elements of the reference ontology from the first mapping. Neither the book nor referenced papers however go in any more detail about the final step of generating the annotated HTML tags.

One important point can be learned from the WSDM description. The production of

```
-@ var vcard: VCard

div( itemscope itemtype="http://schema.org/Person"
     itemid=#{vcard.getProperty("uid")} )
  span( itemprop="name" )
    #{vcard.getProperty("fn")}
  span( itemprop="telephone" )
    #{vcard.getProperty("tel")}
```

Listing 7: Defining all Microdata attributes manually in an HTML template

```
-@ var md: MicroData

= md.scope
  div
    = md.prop("name")
      span( style="color:red" )
    = md.prop("telephone")
    = md.prop("email")
```

Listing 8: Using a Microdata-aware data structure in a template

semantically annotated HTML can become a lot easier if the entity is already available represented with the targeted vocabulary. A very naive approach to produce annotated HTML would be to just manually write the necessary attributes in the template and fill them with values from an arbitrary data object, as demonstrated in listing 7. Even with the conciseness of the used template language Jade[65], the developer still has a lot to type.

Compared to the above listing 8 shows a template using a data structure that is aware of the used Microdata vocabulary and wraps an instance of a typed Microdata item with its properties. The `scope` method of the Microdata interface will add the `itemscope`, `itemtype` and `itemid` attributes to the nested `div` element. The `prop` method either augments a nested element as shown for the `name` property or creates the correct nested element. The method adds the `itemprop` attribute and puts the value for this property inside the element.

An implementation of this approach must take care of a few peculiarities[Hic11a]. Some properties don't necessarily use simple `span` elements, e.g. dates can be better expressed with `time` elements or URI values most likely appear in an `a`, `img`, `link` or `object` element. Property values could also be put in a `content` attribute while the element's nested text content is optimized for human consumption. Items can be nested, e.g. an item of type PostalAddress could be nested inside a Person item.

The proposed approach can be implemented on any template engine as long as it permits to capture and manipulate nested HTML elements and to call methods of passed in

[65] `http://scalate.fusesource.org/documentation/jade-syntax.html` (2012-2-22) Jade is the most concise among several supported template languages of the Scalate Template Engine.

objects.[66]

[66] https://github.com/Paxa/green_monkey (2012-3-7) provides helpers to produce microdata in rails but is not as automated as the design proposed here.

7 Results and Discussion

7.1 Further work

7.1.1 Browser Caches as Collection Store

The example of OpenSocial shows how important the Browser has become as an application platform. However browsers also restrict the options of developers, especially in terms of available Webstorage (subsection 3.3).

Clever use of the Browser cache could raise this limit. The average available Cache size is unknown but expected to exceed the Webstorage size by several orders of magnitude.

One interesting caching strategy in combination with collections is the use of "caching tokens", i.e. each new version of a resource is available under a new URI and served with extremely long cache expiration times. Thus a client only learns about a new version of a resource, if another resource, in our case the collection, updates its hyperlink to the resource. This is a perfect match to the Atom Publishing Protocol which provides updated entries each time a linked resource changes.

In this context the exact caching behavior of popular browsers is of interest, especially in combination with Content-Location headers or the only-if-cached directive.

7.1.2 Patching Resources

subsection 4.5 introduced Delta Encoding[MKD+02] and mentioned that it would also be useful to efficiently respond to GET requests on resources that only slightly changed (e.g., by one property field) compared to the resource cached by the client.

The same argument applies for the other direction, if a client updates a large resource. The HTTP PATCH method[DS10] has been especially standardized for this case in March 2010. Support for the PATCH method can be advertised in the "Allow" header of server responses and the "Accept-Patch" header specifies the Mediatypes of accepted patch formats.

However until today no patch format Mediatypes are listed in the official IANA registry[67] and a draft for a JSON patch format has expired[68].

More critical for this work is that no means have been foreseen to specify the exact representation of a resource to which a PATCH should apply. So given a server that can represent contacts in vCard, xCard and PortableContacts under the same URI depending on the Mediatype accepted by the client and a byte oriented patch Mediatype like the output of the unix `diff -e` command[69]. How should the server know against which format the patch should be applied?

At least three solutions may be possible:

[67] `http://www.iana.org/assignments/media-types/index.html` (2012-03-24)
[68] `https://datatracker.ietf.org/doc/draft-pbryan-json-patch` (2012-03-24)
[69] `http://www.iana.org/assignments/inst-man-values` (2012-03-24)

- A different patch Mediatype is used that is defined to apply to a specific representation. (This is recommended in [All10, ch. 11.9].)
- The server uses separate URIs for different Mediatype representations as suggested by [Ram06] and accepts PATCH request only against those URIs.
- The server uses different entity tags for different representations that it can later use to parse the original resource Mediatype from the etag supplied in the If-MAtch header of the PATCH request.

A further discussion of patch requests should also consider, whether this request type still conforms to the REST interface constraint "manipulation of resources through representations"[Fie00, sec. 5.1.5].

7.1.3 JSON based Mediatypes for Collections

This work examined an API that can serve contact information in different Mediatypes based on XML, JSON and RFC822. However it only considered one XML based format for the discovery of those. It would be desirable to be able to provide an API variant exclusively based on JSON. Therefor a JSON alternative for the ATOM Publishing Protocol is needed.

Collection+JSON The Collection+JSON Mimetype (IANA registered in July 2011) by Mike Amundsen[Amu11b][Amu11a, ch. 3] looks like a promising candidate for a JSON alternative. The author even states that is has been explicitly designed after the model of the Atom Publishing Protocol[70].

The Mediatype defines a JSON structure which contains:

- query templates for the construction of query URIs
- an array of collection items
- one write template to create or edit items
- meta data about the collection

Unfortunately, Collection+JSON in its current state is not yet fully usable to implement the interactions described in 4.

Collection+JSON does not enforce an order of the elements in a collection. The proposed synchronization interaction however is based on the assumption that the collection feed is ordered by the time of the last modification. Consequently, there is also no equivalent to an ATOM "deleted-entry"[Sne12], which enables the use of an updates feed for synchronization.

[70] `http://amundsen.com/media-types/collection` (2012-03-22)

The facility to include full item representations directly in the collection (the "data" property) is restricted to simple key/value pairs. This excludes more complex data structures, like PortableContacts. It would still be possible to omit the optional data property and only fill the href property with a link to the full representation.

An AtomPub collection declares its accepted media types and assumes that the client knows how to produce those. The Collection+JSON Mediatype instead provides a write template which the client must fill in order to create new items. The write template only supports basic key value pairs in accordance to the data property. More complex schemes can not be expressed[71].

A Collection+JSON document is furthermore restricted to contain only one write template. This excludes mixed collections of different types.

Collection+JSON provides query templates but those come without a defined semantic. A mapping from OpenSearch to JSON would be helpful to reuse the semantic definitions. Also the URI template part should be updated to reuse the specification of the new URI templates standard[GFH+12].

Pagination link relations for feeds[Not07] could be reused to express paginated JSON collections as encouraged by the format documentation[Amu11b, sec. 5.5].

Direct Mapping of ATOM XML to JSON James Snell described a mapping of ATOM XML to JSON that should not loose any information[Sne08][72]. The attempt however to map every feature of ATOM and the inherited extensibility and expressiveness of XML results in a very complex and deeply nested JSON structure. In detail, Snell identifies a couple of problems that need to be dealt with in such a mapping[Sne08]:

- JSON has no equivalent for the xml:lang attribute.
- Dereferencable IRIs must be transformed to URIs.
- URIs relative to an xml:base attribute must be resolved, also inside XHTML content elements.
- Repeatable elements must be converted to arrays.
- The ATOM date format (RFC 3339) differs from the JavaScript Date serialization.
- ATOM content elements are versatile but should be represented more meaningful in JSON then just a plain String.
- ATOM supports arbitrary extensions via namespaces.

Considering all the problems, it is understandable that no effort could be found since 2008 to formalize the outlined mapping in an IANA registered Mediatype.

[71] It would make sense to rely on JSON schemas to define valid item structures: `http://json-schema.org` (2012-03-22)

[72] also implemented in Apache Abdera `https://cwiki.apache.org/ABDERA/json-serialization.html` (2012-1-7)

Conclusion Other related formats considered are the "Hypertext Application Language" (HAL)[73] and Microsoft's OData[74]. HAL is a simple container format that only standardizes linking and embedding of resources and is rather meant as a building block or foundation for more specialized formats. Collection+JSON as the more specialized format is therefor preferable here.

OData shares many similarities with the Atom Publishing Protocol and also provides a JSON variant of it. However, despite announcements in March 2010[75], Microsoft has not taken any steps so far to make OData an open standard that could be safely used for free software projects.

In summary, the most promising approach for a JSON variant of ATOM seems to enhance Collection+JSON in the following points:

- an indication, that a collection is ordered by modification time
- means to indicate deletion of items
- means to indicate accepted Mediatypes as an alternative to the write template
- development and adoption of a JSON variant of OpenSearch

7.1.4 Push notifications

This work does not include any means to actively notify (push) a client about changes happening on the server. The client needs to initiate a request (pull) to the server to look for changes. However separate solutions exist[76] to enable a push workflow on top of a feed based application[WM09]. It may therefor not be seen as a disadvantage that push notifications have been omitted as a requirement.[77]

7.1.5 Resource Facades

The idea for the Resource Facades concept was triggered by the use of the JavaBeans Activation Framework[78] (JAF) in the JAX-RS specification. In this framework the DataHandler interface provides access to available commands for a specific MediaType via the getCommand method. The framework however was designed with the needs of a Desktop clipboard in mind. Since JAF has been released for Java version 1.4 it also does neither support Generics nor uses the advantages of immutability.

[PO08] presents an approach and implementation in Scala to attach roles to arbitrary objects. The work achieves type safe roles without extending the underlying language.

[73] `http://stateless.co/hal_specification.html` (2012-03-23)
[74] `http://www.odata.org` (2012-03-23)
[75] `http://web.archive.org/web/20110103120930/http://www.odata.org/blog/2010/3/16/welcome-to-the-new-odataorg!` (2012-03-23)
[76] most notable PubSubHubBub `http://code.google.com/p/pubsubhubbub/` (2012-1-5)
[77] [Dab11, sec. 1] explicitly mentions missing "change notifications" as a "key disadvantage" of CardDAV.
[78] `http://www.oracle.com/technetwork/java/javase/downloads/index-135046.html` (2012-2-24)

Using this library has been considered but it was discovered too late to be included. Open questions are, how the declared media type of a Resource could be considered in the selection of a role implementation and how roles could depend on other roles. Another challenge would be to preserve role instances and thus to avoid recreating them for every invocation. If is furthermore required that roles implement a given interface. The Resource Facade approach presented here is slightly different in that creation of the facades is implemented independent from the facades themselves by the factory classes.

JAX-RS provides the MessageBodyReader and -Writer interfaces. However these interfaces are expected to be used only once per request. The resource method afterwards needs to work with whatever interface was produced by the MessageBodyReader. There exists no facility to request additional transformations or facades of a Resource.

It is possible in JAX-RS to request a MessageBodyReader instance from the javax.ws.rs.ext.Providers interface. This couldn't however help to get additional Facades since the InputStream has already been consumed.

The concept shows similarities with Dependency Injection since dependencies of a facade are also provided by an external component. It may be possible that the concept could even be implemented on top of an existing Dependency Injection framework.[79] Some aspects however may require extra care:

- Resolving the dependencies of Facade factories must consider the Media Type of the input data.
- The scope of an instance is bound to the ResourceHandler which in most cases may be equivalent to the Request scope, but this can't be guaranteed.
- Each ResourceHandler manages its own view of available Facades.

The Apache Wink Rest Framework implements a concept called "Assets".[80] Assets are containers for the resource data injected in or returned from resource methods. Assets provide methods annotated with `@Produces` or `@Consumes` to handle different Media types. In contrast to Resource Facades, the set of supported media types of assets can only be extended by extending the asset classes. It is also not possible like in Figure 3 to provide generic Facades for a TitleAndSummary or Contact.

Scala's type system The proposed Java class diagram in this section has the disadvantage that the availability of a facade can not be checked at compile time. It seems however, that a more advanced type system could help in this regard.

Listing 9 demonstrates features of the Scala type system[Ode11] that could be of interest here. In the example a post method handler has the requirement to access the posted data

[79] Scala can provide Dependency Injection solely with language features via the so called "Cake Pattern". `http://www.warski.org/blog/2011/04/di-in-scala-cake-pattern-pros-cons/` (2012-2-24) or Odersky: "Scalable Component Abstractions"

[80] `https://cwiki.apache.org/WINK/59-assets.html` (2012-2-28)

```
1  trait Storage[ReqFacade] {
2   def create(id: String,
3              body: ResourceHandler
4                    with FacadeFactory[ReqFacade])
5  }
6
7  class PostToCollection[StorageReqFacade]
8              (storage: Storage[StorageReqFacade]) {
9   type MessageBody = ResourceHandler
10                     with FacadeFactory[VCard]
11                     with FacadeFactory[TextSummary]
12                     with FacadeFactory[StorageReqFacade]
13
14  def post(body:MessageBody) : Response = {
15    ...
16    storage.create("id", body)
17    ...
18  }
19 }
```

Listing 9: Implementing the facades approach with Scala's type system

through the facades `VCard` and `TextSummary`. Additionally the data should be forwarded to an implementation of the trait Storage which has its own requirement for a facade.

Scala's "compound types" feature is used in line 9 to combine these requirements into an anonymous type. The "type alias" feature allows it to assign the identifier `MessageBody` to this anonymous type and thus to keep the declaration of the `post` method short and readable.

This example and the mentioned work on Scala roles shows that an advanced type systems may be able to considerably improve the presented facades approach. A more detailed study however is out of the scope of this work and the author's comprehension of type systems.

8 Conclusions

References

[All10] Allamaraju, Subbu: *RESTful Web Services Cookbook*. O'Reilly, 2010. – 314 S.

[Amu11a] Amundsen, Mike: *Building Hypermedia APIs with HTML5 and Node*. O'Reilly, 2011. – ISBN 9781449306571

[Amu11b] Amundsen, Mike: *Collection+JSON - Document Format*. July 2011. – available online at `http://amundsen.com/media-types/collection/format`; accessed on March 22nd, 2012

[Amu11c] Amundsen, Mike: *Supporting PUT and DELETE with HTML FORMS*. December 2011. – available online at `http://amundsen.com/examples/put-delete-forms`; accessed on March 8th, 2012

[Amu12] Amundsen, Mike: The Item and Collection Link Relations / IETF Secretariat. 2012 (draft-amundsen-item-and-collection-link-relations-05). – Internet-Draft. – available online at `https://datatracker.ietf.org/doc/draft-amundsen-item-and-collection-link-relations/`; accessed on March 28th, 2012

[BLFM05] Berners-Lee, Tim ; Fielding, Roy T. ; Masinter, Larry: Uniform Resource Identifier (URI): Generic Syntax. RFC Editor, January 2005 (3986). – RFC

[CDDM09] Casteleyn, Sven ; Daniel, Florian ; Dolog, Peter ; Matera, Maristella: *Engineering Web Applications*. Springer, 2009. – I–XIII, 1–349 S. – ISBN 978–3–540–92200–1

[Cli] Clinton, DeWitt: *OpenSearch Specification 1.1 Draft 5*. – available online at `http://opensearch.org`; accessed on March 1st, 2012

[Cri03] Crispin, Mark R.: Internet Message Access Protocol - VERSION 4rev1. RFC Editor, March 2003 (3501). – RFC

[Cro06] Crockford, Douglas: The application/json Media Type for JavaScript Object Notation (JSON). RFC Editor, July 2006 (4627). – RFC

[Dab11] Daboo, Cyrus: CardDAV: vCard Extensions to Web Distributed Authoring and Versioning (WebDAV). RFC Editor, August 2011 (6352). – RFC

[DDD07] Daboo, Cyrus ; Desruisseaux, Bernard ; Dusseault, Lisa: Calendaring Extensions to WebDAV (CalDAV). RFC Editor, March 2007 (4791). – RFC

[DDL11] Daboo, Cyrus ; Douglass, Mike ; Lees, Steven: xCal: The XML Format for iCalendar. RFC Editor, August 2011 (6321). – RFC

[Des09] Desruisseaux, Bernard: Internet Calendaring and Scheduling Core Object Specification (iCalendar). RFC Editor, September 2009 (5545). – RFC

[DM11] Davis, Cornelia ; Maguire, Tom: XML technologies for RESTful services development. In: *Proceedings of the Second International Workshop on RESTful Design*. New York, NY, USA : ACM, 2011 (WS-REST '11). – ISBN 978–1–4503–0623–2, S. 26–32

[DS10] Dusseault, Lisa ; Snell, James M.: PATCH Method for HTTP. RFC Editor, March 2010 (5789). – RFC

[Dus04] Dusseault, Lisa: *WebDav: next generation collaborative Web authoring*. Prentice Hall PTR, 2004. – ISBN 9780130652089

[Dus07] Dusseault, Lisa: HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV). RFC Editor, June 2007 (4918). – RFC

[FGM+99] Fielding, Roy T. ; Gettys, James ; Mogul, Jeffrey C. ; Nielsen, Henrik F. ; Masinter, Larry ; Leach, Paul J. ; Berners-Lee, Tim: Hypertext Transfer Protocol – HTTP/1.1. RFC Editor, June 1999 (2616). – RFC

[Fie00] Fielding, Roy T.: *REST: Architectural Styles and the Design of Network-based Software Architectures*, University of California, Irvine, Doctoral dissertation, 2000

[Fie08] Fielding, Roy T.: *REST APIs must be hypertext-driven*. October 2008. – available online at `http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven`; accessed on March 8th, 2012

[GFH+12] Gregorio, Joe ; Fielding, Roy T. ; Hadley, Marc ; Nottingham, Mark ; Orchard, David: URI Template. 2012 (6570). – RFC

[GG11] Goel, Kavi ; Gupta, Pravir: *Introducing schema.org: Search engines come together for a richer*. June 2011. – available online at `http://googlewebmastercentral.blogspot.com/2011/06/introducing-schemaorg-search-engines.html` accessed on March 7th, 2012

[Gh07] Gregorio, Joe ; hOra, Bill de: The Atom Publishing Protocol. RFC Editor, October 2007 (5023). – RFC

[GLLM11] George, Robins ; Leiba, Barry ; Li, Kapeng ; Melnikov, Alexey: vCard Format Extension: To Represent the Social Network Information of an Individual / IETF Secretariat. 2011 (draft-ietf-vcarddav-social-networks-00). – Internet-Draft. – available online at `http://datatracker.ietf.org/doc/draft-ietf-vcarddav-social-networks`; accessed on March 28th, 2012

[Hic11a] Hickson, Ian: HTML Microdata / W3C. 2011. – W3C Working Draft. – available online at `http://www.w3.org/TR/microdata/`; accessed on February 17th, 2012

[Hic11b] Hickson, Ian: HTML5. A vocabulary and associated APIs for HTML and XHTML / W3C. 2011. – W3C Working Draft. – available online at `http://www.w3.org/TR/html5`; accessed on March 1st, 2012

[Hic11c] Hickson, Ian: Web Storage / W3C. 2011. – W3C Candidate Recommendation. – available online at `http://www.w3.org/TR/webstorage`; accessed on March 24th, 2012

[HS09] Hadley, Marc ; Sandoz, Paul: *JSR 311: JAX-RS: The Java API for RESTful Web Services Version 1.1.* September 2009 available online at `http://www.jcp.org/en/jsr/detail?id=311`; accessed on March 7th, 2012

[HSD98] Howes, Tim ; Smith, Mark ; Dawson, Frank: A MIME Content-Type for Directory Information. RFC Editor, September 1998 (2425). – RFC

[hÓ09] hÓra, Bill de: *Extensions v Envelopes.* November 2009. – available online at `http://www.dehora.net/journal/2009/11/28/extensions-v-envelopes`; accessed on March 24th, 2012

[Hü09] Hübner, Harry: *Implementierung der OpenSocial-API in der Communityumgebung für das Fernstudium*, Fernuniversität Hagen, Lehrgebiet Informationssysteme und Datenbanken, Bachelor thesis, 6 2009. `harry011.files.wordpress.com/2009/06/opensocial_containerimpl.pdf`

[MKD+02] Mogul, Jeffrey C. ; Krishnamurthy, Belachander ; Douglis, Fred ; Feldmann, Anja ; Goland, Yaron Y. ; Hoff, Arthur van ; Hellerstein, Daniel M.: Delta encoding in HTTP. RFC Editor, January 2002 (3229). – RFC

[NL99] Nielsen, Henrik F. ; LaLiberte, Daniel: Editing the Web. Detecting the Lost Update Problem Using Unreserved Checkout / W3C. 1999. – W3C Note. – available online at `http://www.w3.org/1999/04/Editing`; accessed on March 1st, 2012

[Not07] Nottingham, Mark: Feed Paging and Archiving. RFC Editor, September 2007 (5005). – RFC

[Not10] Nottingham, Mark: Web Linking. RFC Editor, October 2010 (5988). – RFC

[NS05] Nottingham, Mark ; Sayre, Robert: The Atom Syndication Format. RFC Editor, December 2005 (4287). – RFC

[Ode11] Odersky, Martin: *The Scala Language Specification Version 2.9.* website scala-lang.org, section Documentation/Manuals/Scala Language Specification, May 2011. – available online at `http://www.scala-lang.org/sites/default/files/linuxsoft_archives/docu/files/ScalaReference.pdf` accessed on February 14th, 2012

[Ope11] OpenSocial and Gadgets Specification Group: *OpenSocial Specification Version 2.0.1.* `mailto:opensocial-and-gadgets-spec@googlegroups.com`, 11 2011. – available online at `http://docs.opensocial.org/display/OSD/Specs`; accessed on January 10th, 2012

[Per11a] Perreault, Simon: vCard Format Specification. RFC Editor, August 2011 (6350). – RFC

[Per11b] Perreault, Simon: xCard: vCard XML Representation. RFC Editor, August 2011 (6351). – RFC

[PO08] Pradel, Michael ; Odersky, Martin: Scala Roles - A Lightweight Approach towards Reusable Collaborations. In: *International Conference on Software and Data Technologies (ICSOFT '08)*, 2008

[PSMB98] Paoli, Jean ; Sperberg-McQueen, C. M. ; Bray, Tim: XML 1.0 Recommendation / W3C. 1998. – first Edition of a Recommendation. – http://www.w3.org/TR/1998/REC-xml-19980210

[Ram06] Raman, T. V.: On Linking Alternative Representations To Enable Discovery And Publishing / W3C. 2006. – W3C TAG Finding. – available online at `http://www.w3.org/2001/tag/doc/alternatives-discovery.html`; accessed on March 2nd, 2012

[RBM05] Royer, Doug ; Babics, George ; Mansour, Steve: Calendar Access Protocol (CAP). RFC Editor, December 2005 (4324). – RFC

[Res08] Resnick, Peter W.: Internet Message Format. RFC Editor, October 2008 (5322). – RFC

[RGJ05] Reif, Gerald ; Gall, Harald C. ; Jazayeri, Mehdi: WEESA - Web Engineering for Semanitc Web Applications. In: *Proceedings of the 14th International World Wide Web Conference.* Chiba, Japan, May 2005, S. 722–729

[Sne07a] Snell, James: Atom Publishing Protocol Feature Discovery / IETF Secretariat. 2007 (draft-snell-atompub-feature-12). – Internet-Draft. – available online at `http://tools.ietf.org/id/draft-snell-atompub-feature-12.txt`; accessed on March 2nd, 2012

[Sne07b] Snell, James: *Sync!* December 2007. – available online at http://web.archive.org/web/20081114142152/http://www.snellspace.com/wp/?p=818; accessed on March 8th, 2012

[Sne08] Snell, James: *Convert Atom documents to JSON.* IBM developerWorks, January 2008. – Available online at http://www.ibm.com/developerworks/library/x-atom2json/index.html; accessed on January 7th, 2012

[Sne12] Snell, James: The Atom "deleted-entry" Element / IETF Secretariat. 2012 (draft-snell-atompub-tombstones-14). – Internet-Draft. – available online at http://tools.ietf.org/id/draft-snell-atompub-tombstones-14.txt; accessed on February 28th, 2012

[Spo11] Sporny, Manu: *An Uber-comparison of RDFa, Microdata and Microformats.* June 2011. – available online at http://manu.sporny.org/2011/uber-comparison-rdfa-md-uf/ accessed on March 7th, 2012

[Sto04] Stoermer, Magnus: *Open Source Groupware am Beispiel Kolab*, FOM Fachhochschule für Oekonomie & Management Essen / Neuss, Diplomarbeit, October 2004. – available online at http://ftp.kolab.org/contrib/diplom_thetis_stoermer/OSS_Groupware_Kolab.pdf; accessed on March 27th, 2012

[Ten12] Tennison, Jeni: HTML Data Guide - Working Draft / W3C. 2012. – W3C Working Draft. – available online at http://www.w3.org/TR/2012/WD-html-data-guide-20120112/; accessed on February 16th, 2012

[Web10] Webber, Jim: *REST in Practice: Hypermedia and Systems Architecture.* O'Reilly, 2010. – ISBN 978–0–596–80582–1

[WM09] Wilde, Erik ; Marinos, Alexandros: Feed Querying as a Proxy for Querying the Web. In: *Proceedings of the 8th International Conference on Flexible Query Answering Systems.* Berlin, Heidelberg : Springer-Verlag, 2009 (FQAS '09). – ISBN 978–3–642–04956–9, S. 663–674

[Wym04] Wyman, Bob: *Using RFC3229 with Feeds.* September 2004. – available online at http://bob.wyman.us/main/2004/09/using_rfc3229_w.html; accessed on March 9th, 2012

[Yat07] Yates, Rob: *CalAtom.* April 2007. – available online at http://robubu.com/CalAtom/calatom-draft-00.txt; accessed on March 7th, 2012

[YSHG07] Yesilada, Yeliz ; Stevens, Robert ; Harper, Simon ; Goble, Carole: Evaluating DANTE: Semantic transcoding for visually disabled users. In: *ACM Transactions on Computer-Human Interaction* 14 (2007), September

[Çe06] Çelik, Tantek: *Introducing Microformats Search and Pingerati.* May 2006. – available online at `http://tantek.com/log/2006/05.html`; accessed on March 7th, 2012

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt.

Kreuzlingen, 4. April 2012 *Thomas Koch*