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

bachelor thesis

Thomas Koch

March 25, 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?

optionale Ergänzungen:

- CardDAV als alternatives Protokoll mit Untersuchung der Wiederverwendbarkeit der Komponenten
- Mitbetrachtung der Kalenderfunktionalität bei der Hypermediaunterstützung
- zumindest lesender Zugriff auf die Kalenderfunktionalität über die REST-Schnittstelle

[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 multinational corporation.

2 Background and Related Work

2.1 REST architectural style

[Fie00, sec. 5.1.5]

"REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state."

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]:

- Cacheability (sec. 5.1.4) can keep the data available also in offline mode, 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

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. For the rest of this work a Groupware is 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.

There are a few appealing advantages to this approach:

- The IMAP infrastructure used for mail can be reused.
- 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 capabilities.

IMAP in general also comes with its own challenges:

- The standard documents describing and extending IMAP are many[6].
- IMAP imposes a folder structure and does not permit alternative structures like tags as used by Google's gmail.
- 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[7].

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, 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.

The iCalendar standard defines several properties that can link to external representations of the properties value by specifying an "Alternate Text Representation" parameter. These are comment, summary, description, contact, location, resources. The properties attendee and organizer can have a "Directory Entry Reference" parameter that should contain an URI to a person resource. One property that can not be dereferenced is "Related To": It only contains the globally unique identifier of another calendar component.

One problem that arises with the use of hyperlinks in personal information management is identification across administrative boundaries. Take for example an event that gets sent from one organization to another and contains

[6] `http://www.apps.ietf.org/rfc/ipoplist.html` (2012-3-5)

[7] `http://www.courier-mta.org/fud` (2012-3-5)

hyperlinks to person representations. These hyperlinks most likely point to an internal addressbook of the organization and may not be accessible by the receiver of the event information. The receiver however may have his own addressbook containing information about the person.

2.5 PortableContacts

Portable Contacts[8] is a specification initiated in 2008 by Joseph Smarr while working at the Address Book internet service Plaxo.com[9]. It comprises of a JSON schema for contacts information derived from vCard[10] and a protocol for authorized retrieval of contacts. The schema omits many properties of the current vCard 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 thus 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[11]. Both protocols extend WebDAV[Dus07] and thus inherit its characteristics.

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 wrote[12]:

> Were I to propose CalDAV today it would probably be CalAtom.

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)[13]. In contrast to WebDAV, Feeds and related technologies are also widely used so that a web developer might already know the latter standards.

The second and for this work more important characteristic of WebDAV is, that it is not restful, as explained by Roy Fielding[14]:

[8] `http://portablecontacts.net` (2012-03-23)

[9] `http://josephsmarr.com/2008/12/31/portable-contacts-the-half-year-in-review` (2012-03-23)

[10] `http://wiki.portablecontacts.net/w/page/17776141/schema` (2012-03-23)

[11] only full free server implementations: Apple Calendar Server, Bedeworks, DAViCal, eGroupWare, Owncloud, SOGo, Tine2.0

[12] `http://nih.blogspot.com/2008/02/nearly-two-years-ago-i-made-prediction.html` (2012-1-6)

[13] Atom (43), AtomPub (53), Feed paging (15), OpenSearch (28), Atom Deleted Entry (10)

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

> 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[15].

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[16] 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.

2.7.1 Fieldings Critique

This section examines a critique of Fielding of the API[Fie08][17] 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

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

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

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

> 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:

- 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,[18] 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"[19], 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[20]. Cyrus Daboo, author of some calendaring standards, attributes the failure of CAP to its complexity[21].

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[22]. 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.

[18] OpenSocial persons are based on portable contacts which in turn borrowed field names from vCards.

[19] `http://nordsc.com/ext/classification_of_http_based_apis.html` (2011-12-08)

[20] One free implementation project `http://opencap.sourceforge.net` (2012-3-5) seems inactive since 2005

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

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

3 Requirements and Analysis

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.
- The 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 then 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 Replacement for Kolab IMAP, CardDAV and OpenSocial

Personal evaluation[23] suggests, that many Groupware clients and servers use CardDAV exclusively to synchronize contacts collections, allowing concurrent modifications of individual contacts via optimistic locking (subsubsection 3.6.6). This is also exactly the operation mode of Kolab's IMAP use.

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, 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. Additional indexes could be saved in HTML5 Webstorage[Hic11c].

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

The main interaction considered for OpenSocial is not the synchronization of full collections. Instead it is required that a client can start with the representation of a system's user and explore the user's network of groups, social relations and media collections.

3.3 Client Classes and Characteristics

Different kinds of clients should be able to use the API. Table 3 lists clients whose constraints and characteristics should be respected in 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 behind 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[24]. So the number of contacts in a users address book collection should at least handle 1500 contacts.

The average textual data size associated to a contact is expected to be around 840 bytes[25]. 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.

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

[25] 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

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[26]. So a useful calendar collection has a data size of $730 events * 356 \frac{bytes}{event} \approx 0.25 MB$

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 $\frac{2*1MB}{56kbit/s} \approx 5min$[27]. 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 for the uncompressed case is already under one minute ($\approx 42sec$).

For all described clients the client's storage is large enough to contain 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 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.

3.6.1 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.

3.6.2 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.

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

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

3.6.3 Copying and Moving

WebDAV introduces the HTTP verbs to COPY and MOVE. 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 of the API. 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 does not include initial support for copy or move.

3.6.4 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.

3.6.5 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. 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 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[28].

3.6.6 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 questions to help deciding whether a protocol should support locking and which in the present case advise against locking: *The content is mergeable* in contrast to binary data like images. *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 *the content can be edited while the user is offline*.

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

3.7 Other excluded Requirements

3.7.1 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[29] to enable a push workflow on top of a feed based application[WM09]. It may therefor not be seen as a disadvantage to omit push notifications as a requirement.[30]

3.7.2 Performance optimization

The system is meant to inherit the benefits of a restful architecture. 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 should be taken care that the architecture supports external caching and thus avoids to serve the same read request multiple times.

[29] most notable PubSubHubBub `http://code.google.com/p/pubsubhubbub/` (2012-1-5)

[30] [Dab11, sec. 1] explicitly mentions missing "change notifications" as a "key disadvantage" of CardDAV.

```
<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

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[31] and by an internal project at IBM[32]. 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[32] 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.[33]

An alternative Media Type to Service Documents in JSON could not be

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

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

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

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.[34] 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 answer with a HTTP code "307 Temporary Redirect" to the user's personalized Service Document after successful authentication.[35]

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[36].

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[37]. 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[38].

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.

[34] 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)

[35] 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.

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

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

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

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.

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

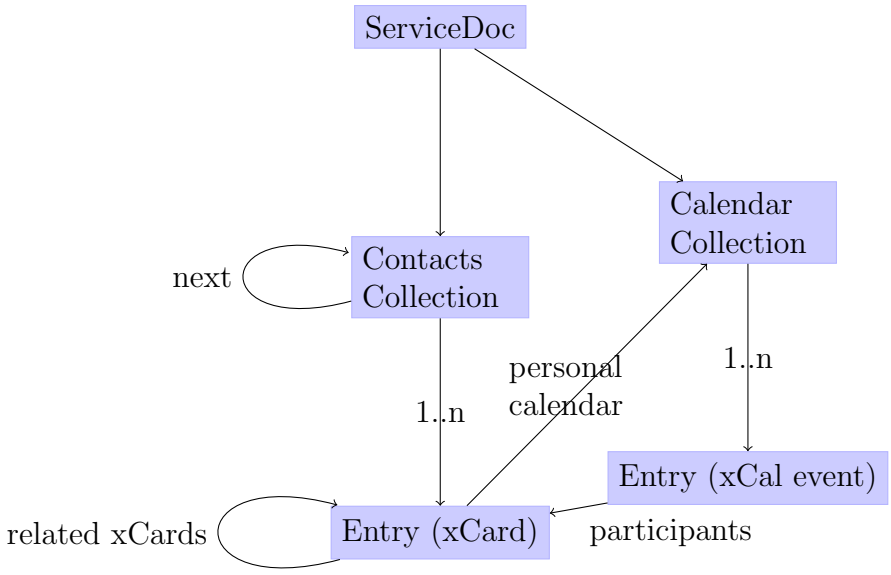

Figure 1: Discovery paths from the Service Document to individual Groupware Resources

has yet invested the effort to drive this method through a formal standardization process[39]. It is however reported to be widely implemented[40], even in the popular Microsoft Internet Explorer[41] and the author claims substantial bandwidths saving opportunities [42].

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[43]. These methods however don't add substantial benefit for the case of synchronization feeds.[44]

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:

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

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

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

valid. This implies of course, that the server is able to match the given etag to a corresponding list of changes[45]. 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 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.

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

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"[46].

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

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

URL" standard with its "Contextual Query Language" (CQL)[47]. 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.

[47] `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".[48] 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. 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)[49] 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.[50] 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 2. 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).

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

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

[50] `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 2: 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.[51] The schema.org vocabulary in turn has been mapped to the semantic world by researchers working on linked data.[52] 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[53], chair of the RDF Web Applications Working Group at the World Wide Web Consortium.

5.3 HTML Forms

TODO

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 the use of HTML forms. These

[51] `http://schema.org/docs/gs.html#microdata_why` (2012-2-17)
[52] `http://schema.rdfs.org/about.html` (2012-2-17)
[53] `http://manu.sporny.org/category/rdfa/` (2012-2-20)

however 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.

TODO: Means to automatically build forms from descriptions of the data: "Dynamic Object Model" (Dirk Riehle) or "Adaptive Object-Model" (Yoder, Johnson) implemented e.g. by eZPublish, Drupal

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.[54] 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.

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

6 Implementation

6.1 Used Frameworks and Libraries

6.2 Overview

Figure 2 outlines the most important classes for the control flow. The implementation relies on 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 6.3.

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[55]. 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.

6.3 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 made between a resource and its representations made by Fielding[Fie00, sec. 5.2.1.1]. In the context of this work, four different kind of properties of resources are distinguished:

- two essential properties: unique Id, last update time
- generic atom entry properties: title, summary, author
- a Mediatype independent interface corresponding to the concept represented by the resource e.g., a person, location, group, organization, event, ...
- a mediatype specific serialization (representation) of the resource

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

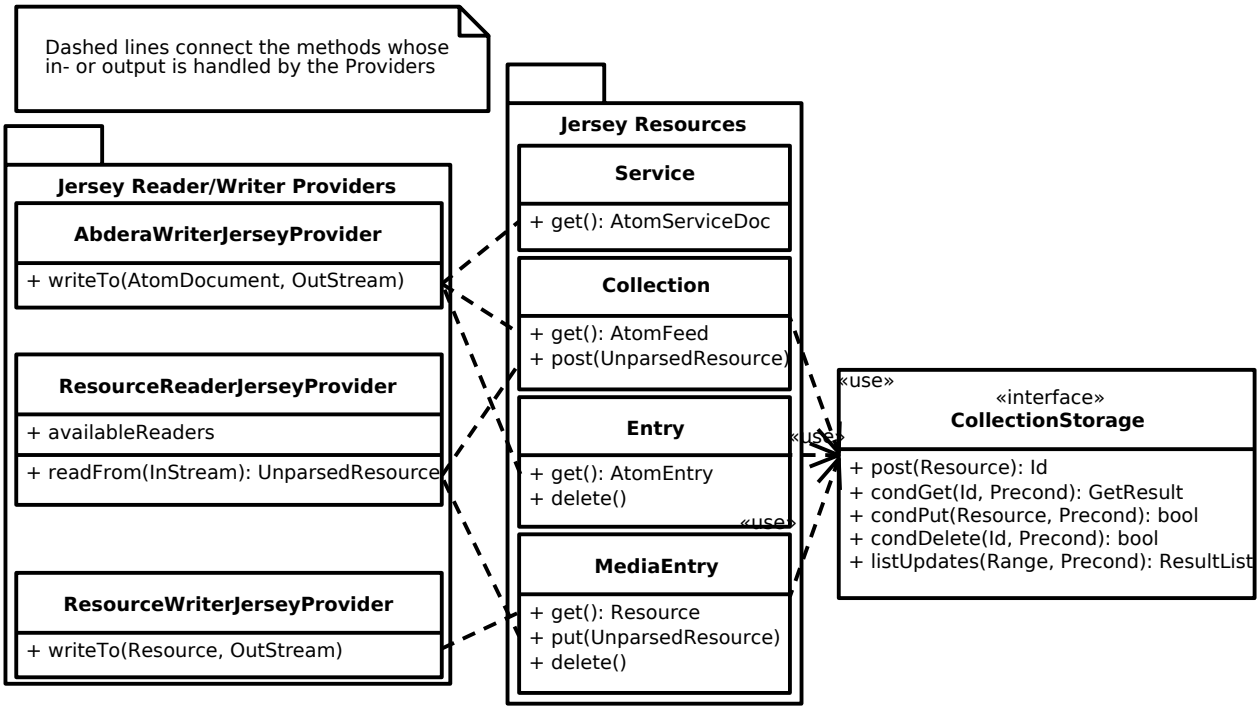

Figure 2: Overview of the execution flow

6.4 CollectionStorage

6.5 Dependency Injection

6.5.1 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 as defined by annotations or done in the Resource method

JAX-RS defines only rudimentary support for the second step by means of inflexible annotations. Listing 3 shows the verbosity of parsing a set of standard query parameters for a search interface. An alternative is shown in listing 4. The parsing of query parameters is delegated to the class SearchRequest.[56] The request method "handleGet" can access the parameters easily through the injected SearchRequest instance.

The main advantages of this approach would be:

- Classes parsing commonly used query parameters can be reused.
- The request method declaration gets much easier to read.

[56]The QueryParams class is supposed to be an easy interface to access query parameters and apply rudimentary validation in one step.

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

Listing 3: Verbosity of parsing Requests with JAX-RS

```
@GET @Inject
public Response handleGet(SearchRequest sr) { ... }

@RequestScoped
public class SearchRequest {
    public final String query, sortBy;
    public final int offset, limit;

    @Inject public SearchRequest(QueryParams qp) {
        query = qp.getNotEmpty("query");
        sortBy = qp.getOrElse("sort-by", "score");
        offset = qp.getPositiveIntOrElse("offset", 0);
        limit = qp.getPositiveIntOrElse("limit", -1);
    }
}
```

Listing 4: Separating Request parsing from the Resource method

- Sophisticated validation can be applied without obfuscating the request method.
- 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.

This approach is possible to implement for example with the dependency injection support provided by the Jersey framework.[57]

6.5.2 Driving Dependency Injection further

The use of dependency injection could be extended to comprise several levels of dependencies and thus to build processing pipelines. The information from the above SearchRequest class is probably just forwarded by "handleGet" to another component that executes the search on a given collection. Thus the request methot is ultimately interested on the search result set to transform it into a response. Consequently the "handleGet" method could use dependency injection to request the result set and only start working on this. Figure 3

[57] `http://codahale.com/what-makes-jersey-interesting-parameter-classes/` (2012-2-5), `http://codahale.com/what-makes-jersey-interesting-injection-providers/` (2012-2-5)

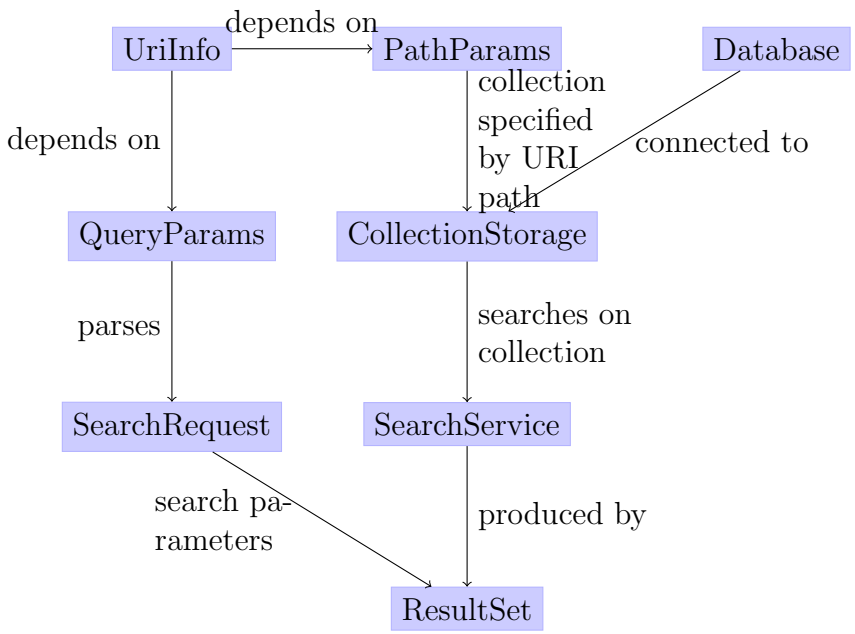

Figure 3: Building a processing pipeline with Dependency Injection

visualizes the hypothetic dependency graph of an application specific ResultSet class.

The figure shows how the CollectionStorage to search on is identified by the URI path and the search parameters by the SearchRequest class of listing 4. The dependency injection is configured to produce a ResultSet class by executing a SearchService with the request scoped CollectionStorage and SearchRequest.

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.

6.6 Reusability of Components

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[58] and a draft for a JSON patch format has expired[59].

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[60]. How should the server know against which format the patch should be applied?

At least three solutions may be possible:

- 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.

7.1.3 JSON based Mediatypes for Collections

Collection+JSON Collection+JSON Mime-Type (approved in July 2011) by Mike Amundsen[Amu11b][Amu11a, ch. 3].

The author states that is has been explicitly designed after the model of Atom Publishing Protocol[61]

[58] `http://www.iana.org/assignments/media-types/index.html` (2012-03-24)
[59] `https://datatracker.ietf.org/doc/draft-pbryan-json-patch` (2012-03-24)
[60] `http://www.iana.org/assignments/inst-man-values` (2012-03-24)
[61] `http://amundsen.com/media-types/collection` (2012-03-22)

The Mediatype consists of

- Query Templates
- items array
- Write Template - to create or edit items
- Meta data about the collection

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.

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 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[62].

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 upcoming 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 JSON ATOM serialization implemented by Apache Abdera[63]

Some problems in loss-less conversion of ATOM to json:[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.

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

[63] [Sne08] `https://cwiki.apache.org/ABDERA/json-serialization.html` (2012-1-7)

- 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.

Conclusion Other related formats considered are the "Hypertext Application Language" (HAL)[64] and Microsoft's OData[65]. 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[66], 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

[64] `http://stateless.co/hal_specification.html` (2012-03-23)
[65] `http://www.odata.org` (2012-03-23)
[66] `http://web.archive.org/web/20110103120930/http://www.odata.org/blog/2010/3/16/welcome-to-the-new-odataorg!` (2012-03-23)

8 Conclusions

9 alles alt hier unten

10 Media Types

> To some extent, people get REST wrong because I failed to include enough detail on media type design within my dissertation. – Roy T. Fielding, [Fie08]

[PZL08, sec. 7.2] identifies the support of different media types as an issue that "can complicate and hinder the interoperability" and "requires more maintenance effort".

[DM11] proposes a XML based REST framework that uses XForms, XQuery, XProc, XSLT and an XML database. It can benefit from the constraint that it only supports XML based media types. It is to be seen, which ideas from this work could be reused in the case of a broader variety of supported media types.

10.1 Syntax vs. Semantic (Vocabulary)

The use of standardized media types is one key difference between an API and a restful API[Fie00, sec. 5.2.1.2]. Only if the client has knowledge about the media type can it do something meaningful with it besides just receiving it. In that sense, the often used mime types application/xml or application/json are not really media types. They don't tell the browser or user anything meaningful beside the *syntax* of the data.[67]

To do anything meaningful with plain json or xml, the client programmer must normally look up the meaning or *semantic* of the data in the API documentation. The data therefor fails the self-descriptive constraint of REST.

Compare this with a mime type like `application/atom+xml`. It specifies the syntax (xml) and the semantic (atom) of the data. Of course somebody once needed to read the atom specification and program the client with the knowledge of how to process this media type. The purpose of standardized media types however is that their number is limited enough so that there is a fair chance that a client might have implemented a given media type.

Large sites like Google, Facebook or Twitter have the market power to attract developers to read their specifications and program clients accordingly. They thus don't necessary need to rely on standardized media types. REST however envisions a decentralized web in which parties can interact without previous knowledge of each other. This becomes possible through the usage of well known predefined media types.

[67] `http://blog.programmableweb.com/2011/11/18/rest-api-design-putting-the-type-in-content-type` (2011-21-20) and Web Resource Modeling Language `http://www.wrml.org` (2011-12-20) both by Mark Massé

10.1.1 XML vs. JSON

This section investigates the two most common syntaxes used by media types and the issues that arise if an application needs to support both of them.

The application section of the IANA mime type registration has 294 entries ending in "+xml" and only 3 ending in "+json".[68] This stands in contrast to the rise of public JSON APIs and the decline of XML APIs.[69]

A strong argument for JSON as the preferred format for public APIs may be that JSON is a subset of JavaScript and thus easily consumable in a web browser.[70]

A drawback of this mismatch between the preference of media type designers and API consumers is a possible duplication of work and incompatibilities across different APIs. An author that wants to offer a public API as JSON is likely to find only an existing XML media type, but no one in JSON. The situation would be eased, if a standard mapping from XML schemes to JSON would be possible, but that is not the case.

Instead, possible mappings have to trade of the preservation of all structural information against the "friendliness" of the resulting JSON structure[BGM+11]. Without going into detail, a JSON structure can be seen as friendly if it makes best use of JSON's data types, is compact and easy to process. Listing 5 shows two different examples how to map data from XML to JSON with one of them using JSON number values, being more compact and probably easier to process.

Listing 5: XML fragment

```
<lang pref="1"
      id="fr" />
<lang pref="3"
      id="en" />
```

unfriendly JSON

```
"languages": [
  { "id":"fr",
    "pref":"1" },
  { "id":"en",
    "pref":"3" }
]
```

friendly JSON

```
"languages": {
  "fr":1,
  "en":3
}
```

Activity Streams has avoided the misalignment of an official XML format and an unofficial JSON deviate by defining an XML (ATOM) and JSON format from the beginning.[71]

[68] `http://www.iana.org/assignments/media-types/application/index.html` (2011-12-20)

[69] `http://blog.programmableweb.com/2011/05/25/1-in-5-apis-say-bye-xml/` (2011-12-20) `http://www.readwriteweb.com/cloud/2011/03/programmable-web-apis-popping.php` (2011-12-21)

[70] ECMAScript for XML (E4X) makes XML a first class language construct in the browser but is only supported by Mozilla `http://en.wikipedia.org/wiki/ECMAScript_for_XML` (2012-2-2)

[71] `http://activitystrea.ms/` (2012-01-21)

```
@Path("atm/{cardId}")
public class AtmResource {
‘
  @GET
  @Path("balance")
  @Produces("text/plain")
  public String balance(@PathParam("cardId") String card,
                        @QueryParam("pin") String pin) {
    return Double.toString(getBalance(card, pin));
  }
```

Listing 6: Example of a JAX-RS annotated Resource class (by Marek Potociar)

11 Design

11.1 Reusable Patterns and Components

11.2 Components

11.2.1 Dispatcher

The dispatcher selects the Java method (see 11.3.1) that should handle the request. The selection can depend at least on the path component of the requested URI, the media types accepted by the client as indicated in the request's ACCEPT header and the HTTP verb.

Every project implementing JAX-RS[HS09] needs to have some kind of dispatcher component. The specification itself does not identify this component. It does however specify the algorithm a dispatcher needs to follow and a set of Java annotations which must be used to configure the dispatch. These annotations (PATH, GET for the HTTP verb and Produces) are demonstrated in listing 6.

Alternative approaches to configure the dispatcher are not designated by JAX-RS. One possible alternative would be to expose an API to manually add dispatch routes at runtime and remove the corresponding annotations from the source code.

This approach is indeed implemented e.g. by Restlet[72], Apache Wink[73] and probably others. Jersey 2.0 is also expected to provide an API for the dispatcher.[74]:

Advantages of a dynamic dispatcher configuration would be:

- The path under which a resource type is served is decoupled from the code defining the behavior of the resource. This could enable the reuse of resource classes or methods in other contexts.

[72] `http://wiki.restlet.org/docs_2.1/13-restlet/27-restlet/326-restlet.html` (2012-2-6)

[73] called "Dynamic Resources" `http://incubator.apache.org/wink/1.1/html/5.1RegistrationandConfiguration.html` (2012-2-7)

[74] `http://java.net/jira/browse/JERSEY-842` (2012-2-6)

- The decision which media types can be consumed or produced may not depend solely on the resource class or method. A resource method may work on a domain specific data type and the set of supported media types may depend on the available converter between media types and the data type. A photo album for example resource may be able to consume any number of different image formats that a separate component can convert to an internal image representation.

- The list of supported media types could be created programmatically. This enables reuse of set of equivalent media types or combination of media type categories for example to combine the sets of image, video and audio media types.

- The concept of resource classes could be replaced altogether. The life cycle of a resource class in JAX-RS defaults to the request scope. During one request only one resource method is called. Resource methods therefor by default don't share state through resource class attributes. It would therefor be possible to bind individual functors to dispatcher routes and thus composing the equivalent of a resource class at runtime.

The dispatching as defined in JAX-RS does not define any facility for a resource method to decline its possibility to handle a method at runtime. Such a facility could either be implemented by a boolean precondition method associated with the resource method or by a special Exception type that would restart the request dispatch but this time ignoring the method that threw the exception. If no alternative request method could be found, the Exception would be propagated and subsequently transformed into an appropriate error response.

Thus it would be possible to define generic and special purpose request methods even for cases where the static JAX-RS dispatch algorithm does not provide sufficient granularity.

While all this flexibility can provide many advantages it has to be kept in mind how the framework can gather enough knowledge to still help by autogenerating e.g. WADL documents and responses to HEAD and OPTION requests.

11.2.2 Resource Facades

Fielding discerns between a resource and the representation of a resource in a certain format, "selected dynamically based on the capabilities or desires of the recipient and the nature of the resource"[Fie00, p. 87]. According to this notion, the media type used to represent a resource should not influence the processing logic. In an ideal case all possible media types should be handled by the same resource method.

This ideal contrasts with JAX-RS concepts where the media type can be one parameter of the dispatcher logic. This section outlines a pattern tentatively named "Resource Facades" that should make it easier to handle different media types with the same code and thus to facilitate code reuse.

A resource method should contain the programming logic executed to serve a request of a specific type (e.g. GET, PUT) against a specific resource. The programming logic could execute common tasks like the following:

- validate the correctness of a submitted resource
- check the clients authorization
- persist the submitted resource data
- trigger notifications containing a summary of the resource
- submit the submitted resource to an indexing system
- check the submitted resource to be of a certain accepted domain type, like contact, event, todo item or any set of such types

All the above processing tasks should in theory be independent of the media type of a resource and only be programmed once to work on any resource format. This could be made possible by applying the concept of roles to resources. Roles have been described already 15 years ago by [Fow97] or a bit later by [BRSW00]. However no evidence could be found whether roles have been used to implement restful systems.

According to [Ste08], there exists several definitions for roles which mostly share a few core properties:

> This includes the property that a single object can play several roles of different or the same kind both simultaneously and sequentially, and that the same role can be played by different objects of the same and different kinds. Raised to the type level, this means that the relationship between role types and class types (as sources of role players) is generally m:n.

A popular example for roles is a person, that can have the different roles over their lifetime (student, professor, single, husband, widower) or in different contexts (teacher, father, husband, customer, politician).

Exemplified with the above tasks, a resource can have the role of being validated, persisted, summarized or checked for being of a certain type. So like in the above quote a facility is needed that can provide m different roles of resources that come in n different shapes.

It can be noted, that unlike in the previous example with roles of a person, this resource roles examples do not extend the original resource with new attributes. A person surely gets additional attributes as a father (references to children) or professor (member of faculty). Thus the term “facade” in favor of role should indicate that only different views of the same data are provided.

Listing 7 shows interfaces of a minimal framework to provide Facades for Resources. The idea is, that any code that needs information from a Resource requests the appropriate Facade from the ResourceHandler. The ResourceHandler was instantiated with a FacadeRegistry from which it can request Factories for requested Facades. A ResourceHandler must have been instantiated with at least one initial input Facade, e.g. an InputStream.

```
interface FacadeFactory<T> {
 T build(ResourceHandler resourceHandler);

 /**
  * Dependency Facades needed by this factory.
  */
 Iterable<? extends Class<?>> getDependencies();
}

interface FacadeRegistry {
 /**
  * Returns Facade factories that could probably
  * build the requested Facade.
  *
  * @param mediaType MediaType of the original Resource
  * @param clazz requested Facade interface
  */
 Iterable<FacadeFactory<?>> getFacadeFactories(MediaType mediaType,
                                               Class<?> clazz);
}

interface ResourceHandler {
 /**
  * Returns the unique instance of a Facade for this Resource
  *
  * Subsequent calls with the same parameter receive the
  * _same_ unique Facade instance!
  *
  * @param clazz requested Facade interface
  * @return Facade implementation instance
  */
 <T> T getFacade(Class<T> clazz);

 /**
  * Is the requested Facade interface available for this Resource?
  *
  * @param clazz Facade interface
  */
 boolean hasFacade(Class<?> clazz);

 /**
  * The MediaType of the original Resource from which this
  * ResourceHandler was instantiated.
  */
 MediaType getMediaType();
}
```

Listing 7: API of the ResourceFacades component

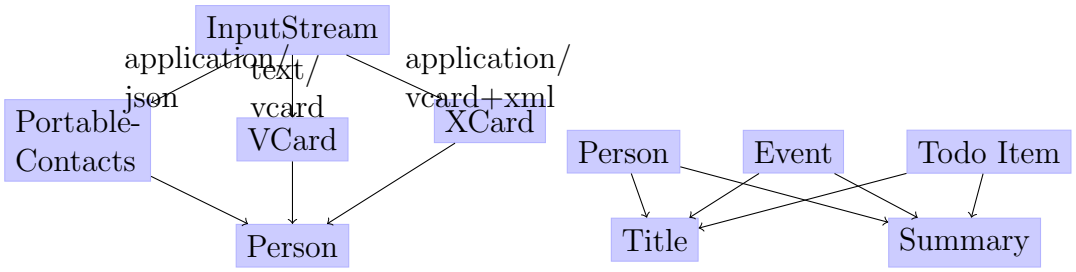

Figure 4: Facade examples with dependencies

Figure 4 presents two example use cases for Facades. On the left site the request method might want to know the full name of a submitted contact resource. It therefor requests a Person facade. Different Person Facade factories are registered. The different factories in turn have each a dependency on an InputStream parameterized with a Media Type. The provided Media Type makes the resolution path unambiguous.

The right site shows dependencies of Title and Summary Facades. Different Factories would be provided that knows to create meaningful titles and summaries for Persons, Events or Todo items independent of the original Media Types. A title of a person surely includes the full name, for an event the date and event title would be combined and a todo item could include the priority in the title.

Related work The idea for the Resource Facades concept was triggered by the use of the JavaBeans Activation Framework[75] (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. 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.

[75] `http://www.oracle.com/technetwork/java/javase/downloads/index-135046.html` (2012-2-24)

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.[76] 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".[77] 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 listing 4 to provide generic Facades for a title or a summary.

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 8 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 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 8 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.

[76] 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"

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

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

Listing 8: Implementing the facades approach with Scala's type system

11.3 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 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 9. Even with the conciseness of the used template language Jade[78], the developer still has a lot to type.

Compared to the above listing 10 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.

[78] `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.

```
-@ 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 9: 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 10: Using a Microdata-aware data structure in a template

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 objects.[79]

11.3.1 Other components

Actions An action is basically the code that should be executed to respond to a client request. An action receives all information about a request and is connected to the application. It can use and manipulate the application state and produces a data structure representing the response. It can be compared to the "Request method" defined in JAX-RS.

It is desirable to reuse actions across different consumed media types. Typical tasks to perform in a POST or PUT resource method are:

- Transform the input format in a format suitable for the storage component.
- Check the validity of the received data.

[79] `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.

- Extract information to be sent to another component, e.g. to notify users about changes or to index the new data for search.

References

[All10] Allamaraju, Subbu ; Treseler, Mary E. (Hrsg.): *RESTful Web Services Cookbook*. O'Reilly, 2010. – 314 S.

[Amu11a] Amundsen, Mike: *Building Hypermedia APIs with HTML5 and Node*. O'Reilly Media, 2011 (Oreilly and Associate Series). – ISBN 9781449306571

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

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

[BGM+11] Boyer, John ; Gao, Sandy ; Malaika, Susan ; Maximilien, Michael ; Salz, Rich ; Simeon, Jerome: Experiences with JSON and XML Transformations. In: *Workshop on Data and Services Integration* W3C, 2011

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

[BRSW00] *Kapitel* 2. In: Bäumer, Dirk ; Riehle, Dirk ; Siberski, Wolf ; Wulf, Martina: *Role Object*. Reading, Massachusetts : Addison-Wesley, 2000 (Pattern Languages of Program Design 4), S. 15–32

[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*. `http://opensearch.org`. – available online at `http://opensearch.org`; visited at 1st March 2012

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

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

[DDD07] Daboo, C. ; Desruisseaux, B. ; Dusseault, L.: Calendaring Extensions to WebDAV (CalDAV) / RFC Editor. RFC Editor, March 2007 (4791). – RFC

[DDL11] Daboo, C. ; Douglass, M. ; Lees, S.: xCal: The XML Format for iCalendar / RFC Editor. RFC Editor, August 2011 (6321). – RFC

[Des09] Desruisseaux, B.: Internet Calendaring and Scheduling Core Object Specification (iCalendar) / RFC Editor. 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, 26–32

[DS10] Dusseault, L. ; Snell, J.: PATCH Method for HTTP / RFC Editor. RFC Editor, March 2010 (5789). – RFC

[Dus04] Dusseault, L.: *WebDav: next generation collaborative Web authoring*. Prentice Hall PTR, 2004 (Prentice Hall series in computer networking and distributed systems). `http://books.google.com/books?id=LN6PRtgiwNgC`. – ISBN 9780130652089

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

[FGM+99] Fielding, R. ; Gettys, J. ; Mogul, J. ; Frystyk, H. ; Masinter, L. ; Leach, P. ; Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1 / RFC Editor. 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*. `http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven`. Version: October 2008. – available online at `http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven`; visited 8th March 2012

[Fow97] Fowler, Martin: Dealing with Roles. In: *4th Pattern Languages of Programming Conference*, 1997. – Available online at `http://martinfowler.com/apsupp/roles.pdf`; visited at 23th February 2012

[GFH+12] Gregorio, Joe ; Fielding, Roy T. ; Hadley, Marc ; Nottingham, Mark ; Orchard, David: URI Template / IETF Secretariat. Version: January 2012. `http://datatracker.ietf.org/doc/draft-gregorio-uritemplate`. 2012 (draft-gregorio-uritemplate-08). – Internet-Draft. – available online at `http://datatracker.ietf.org/doc/draft-gregorio-uritemplate`; visited 1st March 2012

[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 visited at 7th March 2012

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

[Hic11a] Hickson, Ian: HTML Microdata / W3C. Version: May 2011. http://www.w3.org/TR/microdata/http://dev.w3.org/html5/md/Overview.html. 2011. – W3C Working Draft. – Available online at http://www.w3.org/TR/microdata/; visited at 17th February 2012

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

[Hic11c] Hickson, Ian: Web Storage / W3C. Version: December 2011. http://www.w3.org/TR/rdfa-in-html/. 2011. – W3C Candidate Recommendation. – available online at http://www.w3.org/TR/webstorage; visited 24nd March 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; visited at 7th March 2012

[HSD98] Howes, T. ; Smith, M. ; Dawson, F.: A MIME Content-Type for Directory Information / RFC Editor. RFC Editor, September 1998 (2425). – RFC

[hÓ09] hÓra, Bill de: *Extensions v Envelopes.* November 2009. – available online http://www.dehora.net/journal/2009/11/28/extensions-v-envelopes; visited 24th March 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, J. ; Krishnamurthy, B. ; Douglis, F. ; Feldmann, A. ; Goland, Y. ; Hoff, A. van ; Hellerstein, D.: Delta encoding in HTTP / RFC Editor. RFC Editor, January 2002 (3229). – RFC

[NL99] Nielsen, Henrik F. ; LaLiberte, Daniel: Editing the Web. Detecting the Lost Update Problem Using Unreserved Checkout / W3C. Version: May 1999. http://www.w3.org/1999/04/

Editing. 1999. – W3C Note. – Available online at http://www.w3.org/1999/04/Editing; visited at 1st March 2012

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

[NS05] Nottingham, M. ; Sayre, R.: The Atom Syndication Format / RFC Editor. 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 visited on February 14th 2012.

[Ope11] OpenSocial and Gadgets Specification Group: *OpenSocial Specification Version 2.0.1.* http://docs.opensocial.org/display/OSD/Specs. Version: 11 2011. – available onliine at http://docs.opensocial.org/display/OSD/Specs; visited at 10th January 2012

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

[Per11b] Perreault, S.: xCard: vCard XML Representation / RFC Editor. 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. Version: Februar 1998. http://www.w3.org/TR/1998/REC-xml-19980210. 1998. – first Edition of a Recommendation. – http://www.w3.org/TR/1998/REC-xml-19980210

[PZL08] Pautasso, Cesare ; Zimmermann, Olaf ; Leymann, Frank: Restful web services vs. "big"' web services: making the right architectural decision. In: *Proceedings of the 17th international conference on World Wide Web.* New York, NY, USA : ACM, 2008 (WWW '08). – ISBN 978–1–60558–085–2, 805–814

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

[RBM05] Royer, D. ; Babics, G. ; Mansour, S.: Calendar Access Protocol (CAP) / RFC Editor. RFC Editor, December 2005 (4324). – RFC

[Res08] Resnick, P.: Internet Message Format / RFC Editor. 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`; visited at 2nd March 2012

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

[Sne08] Snell, James M.: *Convert Atom documents to JSON*. IBM developerWorks, January 2008. – Available online at `http://www.ibm.com/developerworks/library/x-atom2json/index.html`; visited 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`; visited at 28th February 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/` visited at 7th March 2012

[Ste08] Steimann, Friedrich: Role + counter-role = relationship + collaboration. In: *OOPSLA '08: 23rd Annual ACM Conference on Object-Oriented Programming. Systems, Languages and Applications*. New York, NY, USA : ACM, October 2008. – ISBN 978–1–60558–215–3. – available online at `http://www.fernuni-hagen.de/ps/veroeffentlichungen/57336.shtml`; visited 2nd March 2012

[Ten12] Tennison, Jeni: HTML Data Guide - Working Draft / W3C. Version: January 2012. `http://www.w3.org/TR/2012/WD-html-data-guide-20120112/`. 2012. – W3C Working Draft. – online available at `http://www.w3.org/TR/2012/WD-html-data-guide-20120112/`; last visited at 16th February 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, 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`; visited 9th March 2012

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

[YSHG07] Yesilada, Yeliz ; Stevens, Robert ; Harper, Simon ; Goble, Carole: Evaluating DANTE: Semantic transcoding for visually disabled users. In: *ACM Trans. Comput.-Hum. Interact.* 14 (2007), September. `http://dx.doi.org/http://doi.acm.org/10.1145/1279700.1279704`. – DOI http://doi.acm.org/10.1145/1279700.1279704. – ISSN 1073–0516

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