Thomas Koch
thomas@koch.ro
matriculation number 7250371

February 24, 2012
Fernuniversität Hagen
Faculty of mathematics and computer science

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 my peers.

Existing solutions are either based on non-free software (Microsoft Outlook), brittle and unreliable[1] or require the user to trust his personal data to the commercial interests of a multinational corporation.[2]

This work uses an ongoing effort to draft a meta model for restful applications. Applying this meta model to the domain of this work may provide further insights into its general applicability and fit.

1.1 Definitions

Kolab is the name of a software product ... TODO

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.

TODO: keinen der Terme benutzen.

1.2 Related work

1.2.1 IMAP used by Kolab

Kolab uses an IMAP server as the data store and synchronization protocol for calendar and contact informations. I want to compare this approach to a restful one.

Advantages of IMAP:

- already there, since Mail uses it
- can store blobs/files so no need to map the iCal/vCard files to a relational scheme
- out of the box support for offline work and later synchronization (How does it solve editing conflicts?)

Disadvantages of IMAP:

- Complicate, 38 RFCs according to `http://de.wikipedia.org/wiki/Internet_Message_Access_Protocol` see also: `http://www.apps.ietf.org/rfc/ipoplist.html`
- All clients directly access the iCal/vCard files with no moderation layer in between. This means that no validation or normalization can be done. Schema updates can only be done if all clients cooperate.
- IMAP imposes a folder structure. Google's gmail is an example for another, tag based approach. Messages could have several tags. It is therefor hard to access Gmail via IMAP.
- Sam Varshavchik, the author of the courier Mail Transfer Agent explains the history of IMAP and claims that the IMAP standard is broken: http://www.courier-mta.org/fud/

[1] See comments on the individual projects in appendix ...

[2] Experiences with Android and Data in the cloud `http://keithp.com/blogs/calypso` (?)

- IMAP is so complicate that the IMAP wiki holds 10 pages of advises for IMAP client authors: http://www.imapwiki.org/ClientImplementation RFC 2683 "IMAP4 Implementation Recommendations" is a 23 pages document (cut 5 pages for verbosity) explaining how to implement another RFC standard. Is there any widely used standard that needs another RFC explaining how to implement it?
- `http://en.wikipedia.org/wiki/Internet_Message_Access_Protocol#Disadvantages`
- Some attempts to create a simpler alternative to IMAP:
  - http://en.wikipedia.org/wiki/POP4
  - `http://en.wikipedia.org/wiki/Simple_Mail_Access_Protocol` also here http://www.co mta.org/cone/smap1.html
  - `http://en.wikipedia.org/wiki/Internet_Mail_2000`
  - HTTP restful: http://tools.ietf.org/id/draft-dusseault-httpmail-00.txt mailing list: https://www.ietf.org/mailman/listinfo/httpmail
  - BikINI is not IMAP http://bikini.caterva.org
  - Outlook uses HTTP to communicate with Hotmail
  - another rest mail proposal: http://www.prescod.net/rest/restmail/
- more rants: http://blog.gaborcselle.com/2010/02/how-to-replace-imap.html
- IMAP issues found by the chandler project http://chandlerproject.org/bin/view/Jungle/IntrinsicI

1.2.2 WebDAV, CalDAV, CardDAV

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

– Lisa Dusseault, February 29, 2008[3]

> I gave a big sigh of relief when I read that, and I hope that the CardDAV folks take this to heart. Some parts of WebDAV (e.g., properties [...]) deserve to be taken out back and shot – although, as Lisa says, they were necessary because of the state of the art at the time. That doesn't mean we can't do better now.

– Ted Leung, March 6, 2008[4]

Roy Fielding says WebDAV is not restful: `http://tech.groups.yahoo.com/group/rest-discuss/message/5874`

> PROP* methods conflict with REST because they prevent important resources from having URIs and effectively double the number of methods for no good reason. Both Henrik and I argued against those methods at the time. 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

[3] `http://nih.blogspot.com/2008/02/nearly-two-years-ago-i-made-prediction.html` (2012-1-6)
[4] `http://www.sauria.com/blog/2008/03/06/google-contacts-and-carddav/` (2012-1-6)

> 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.

The discussion also continued on the microformats mailing list `http://microformats.org/discuss/mail/microformats-rest/2006-April/thread.html#217`.

see [Amu10] for a restful approach to properties.

Is ATOM an alternative to WebDAV?

> AtomPub is different from DAV in two key respects:
>
> - The client doesn't control where things go, the server does
> - It is allowed and expected that an AtomPub server will look at the incoming information and change it (generate ID, timestamps, sanitize HTML, etc)

Tim Bray, http://www.imc.org/atom-protocol/mail-archive/msg11271.html

1.2.3 OpenSocial

Roy Fielding wrote a blog post about the "SocialSite REST API", stating that it isn't restful at all but clearly an RPC style API.[5] Fielding was referring to SocialSite, which is however an implementation of the OpenSocial specification. Dave Johnson, a contributor to SocialSite, reacted on this critique by opening a discussion on an OpenSocial mailing list:[6]

> I must admit, it is not clear to me how OpenSocial REST API violates the six rules that Roy stated.

The above quote warrants a short comment. I also thought before, that REST would be so simple that there wouldn't be much need for further studying. Every web developer has some understanding of URIs, HTTP and a bit less of Hypermedia. So it is easy to fall into the trap that everything build on top of HTTP would be restful. Now however, after some more reading about REST, I can easily find violations of the REST constraints in the OpenSocial specification.

Restful APIs are modeled around resources, their representations and links between them. The authors of the OpenSocial API however seem to have modeled their API around the concept of services:[Ope11, Social API Server, sec 2, Services]

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

1.2.3.1 Fielding's critique Fielding listed some rules that a restful API must obey, but did not give explicit examples how OpenSocial violates this rules. The following section will provide such examples.

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

OpenSocial defines a construct called "REST-URI-Fragment" which is a clear violation of the above rule. This URI fragment is simply an encoding of procedural parameters as elements of an HTTP URI:[Ope11, Core API Server, sec 2.1.1.2.2, REST-URI-Fragment]

[5] `http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hyperhtext-driven` (2011-12-06)

[6] `http://groups.google.com/group/opensocial-and-gadgets-spec/browse_thread/thread/aff4ba7373e21284/201a413efa67c26e` (2011-12-06)

> 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}`

URIs can contain a query component that would be more appropriate to contain parameters. This would also have made it clearer to see that the specification actually defines services instead of resources. One test showing the misfit is to ask how dot-segments ('.' and '..') inside the URI fragment are interpreted and whether this conforms with the letter and spirit of the URI standard.[BLFM05, sec 3.3] Another misfit can be seen in the URI fragment to retrieve one or multiple albums. In this case the 'c' part in the quoted definition above is actually a list of albums to retrieve separated by a slash.

Fielding's second bullet point most likely refers to the `X-HTTP-Method-Override` header. This header is a widely used[7] workaround to allow the use of other HTTP methods than GET and POST from HTML forms or through firewalls.

The next two points again refer to a more serious issue:

> 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 OpenSocial specification contains a lot of out-of-band information describing how to form URIs to access information or which methods to use on which URIs for different actions. This means that the OpenSocial API is not simple or intuitive to use but requires a client developer to read a lot of specification, thus violating the simplicity property of a restful architecture. Since the URIs are fixed in the specification and necessarily also in clients, the modifiability property is also violated.[Fie00, sec 2.3]

The following tables give some examples of the specified URIs:

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 the OpenSocial REST API

[7] `http://www.subbu.org/blog/2008/07/another-rest-anti-pattern` (2011-12-06)

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 the OpenSocial REST API

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.

Fielding mentions in a comment to the same blog post that the OpenSocial API "could be made so [restful] with some relatively small changes" but does not specify these changes. However some issues can easily be 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,[8] ATOM entries[NS05] for messages, activities and media items and ATOM categories, collections or workspaces[Gh07] for albums and groups. It would probably be necessary to add extensions to the mentioned media types but vCard and ATOM both already anticipated this need and provided mechanisms to do so.

The use of the ATOM format could promote the adoption of OpenSocial because developers could either reuse existing knowledge about ATOM or would be more motivated to learn about a system that is based on an already widespread format. In fact OpenSocial already mentions ATOM as a way to wrap OpenSocial data. However this wrapping does not build extend and reuse ATOM semantics as proposed above but just puts the OpenSocial data structures inside the entry/content element of ATOM. This kind of misuse of ATOM does of course not deliver any advantage on top of the existing plain JSON or XML representations.[9] Consequently the newest OpenSocial specification deprecates any reference to the ATOM format.

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

[9] compare Bill de Hora, Extensions v Envelopes. 11/2009 `http://www.dehora.net/journal/2009/11/28/extensions-v-envelopes` (2011-21-07)

In Algermissen's classification (**??**), 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 did not have to invest a lot to come up with this API specification but maintenance and evolution cost may be medium or high.

1.2.4 CAP

RFC 4324 "Calendar Access Protocol (CAP)"

> [...] CAP failed because most of the active implementors at the time felt it had become too complex to implement - partly because it required implementing a "brand new" protocol (BEEP). The upshot of that was the CalDAV effort, which was based on an existing fairly well understood protocol - WebDAV. One of the benefits of using WebDAV was that it was easy to take "off the shelf" WebDAV software (server and client) and get a basic CalDAV implementation done very quickly. Indeed, several months after CalDAV was published, CalConnect held an interop event at which several server and client implementations were present and demonstrated basic interoperability - something that would have been hard to achieve with CAP.

– Cyrus Daboo[10]

2 Requirements

2.1 Scope

This work defines a protocol to share

2.2 General Requirements

- Lesen/Schreiben der verwalteten Resourcen: Kontakte, Kalender-Events, Todos, Journal-Einträge, Free-Busy, ...
- Synchronisation von Collections für Offline-Nutzung
- einfach zu implementieren (vgl. CalDAV, CardDAV, IMAP) → ReST
- Standardkonform → xCard, xCal, ATOM
- nutzbar durch JavaScript: JSON basierte Medientypen
- Groupware Elemente: Kontakte, Kalender-Events, Todos, Journal-Einträge, Free-Busy

2.2.0.1 Replacement for CardDAV The web application should provide at least the same features as the CardDAV protocol. It should be demonstrated that a restful application can serve for the same purpose and thus that the additional complexity of WebDAV and CardDAV is not necessary.

The standard lists the supposed main features of CardDAV[Dab11, sec. 1]:

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

1. Ability to use multiple address books with hierarchical layout.
2. Ability to control access to individual address books and address entries as per WebDAV Access Control List (ACL) [RFC3744].
3. Principal collections can be used to enumerate and query other users on the system as per WebDAV ACL [RFC3744].
4. Server-side searching of address data, avoiding the need for clients to download an entire address book in order to do a quick address 'expansion' operation.
5. Well-defined internationalization support through WebDAV's use of XML.
6. Use of vCards [RFC2426] for well-defined address schema to enhance client interoperability.
7. Many limited clients (e.g., mobile devices) contain an HTTP stack that makes implementing WebDAV much easier than other protocols.

There are some minor features of CardDAV, that are mainly inherited from WebDAV and whose general usefulness outside of the scope of a content management system could be argued. See subsection 2.7 for a discussion of those.

2.2.0.2 Restful

The application should obey the constraints of a rest application as specified in [Fie00].

TODO: 4 Grundconstraints von REST auflisten.

The above constraints are not an end in itself but lead to the following required or desirable properties:

- Cacheability (5.1.4) can keep the data available also in offline mode, improves performance and scalability.
- Simplicity helps to develop glue code to connect the application to other systems or to extend it.
- Modifiability allows to adapt the Groupware to changes in the organization.
- Reliability should not need additional justification.
- Administrative scalability means that intermediary components can be deployed independent of the administrator of the main application.

Other outcomes described by Fielding that may not be of importance for the present work are: scalability in terms of users, network performance and efficiency.

2.3 User Classes and Characteristics

TODO: plain Web Browsers, Desktop applications (PIM Suites), JavaScript Gadgets/Widgets (see OpenSocial) probably with WebStorage[11]

Aus den verschiedenen Benutzerern leitet sich ab, dass verschiedene Medientypen unterstützt werden sollen: xCard, JSON, HTML

[11] `http://www.w3.org/TR/webstorage/` (2012-2-2)

2.4 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.

2.5 Design and Implementation Constraints

2.6 Specific Requirements

2.6.0.3 Nested and mixed collections The design should not unnecessarily hinder that collections could be nested or contain different kinds of media types, e.g. calendar items and contacts.

[Dab11, sec. 5.2] forbids nested and mixed collections to ease the implementation of clients. However both may make sense in certain scenarios and the protocol should not exclude such scenarios. A collection could for example represent all items related to a project, which include contacts, events, todos and notes.

2.7 Excluded requirements

2.7.0.4 Search It is not required that the server implements any means to search its data. It is not excluded that such a facility could be added later. It is however expected that searching could be implemented separately. This could be done either on a synchronizing client or as a separate system in the same administrative domain as the server.

2.7.0.5 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.

2.7.0.6 Access Control The aspect of access control would broaden the scope of this work to wide. However it could be kept in mind, whether the proposed design could be enhanced by a separate access control design as proposed in [GZLW11].

2.7.0.7 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.

2.7.0.8 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. The resources are mostly created in one session by one user and seldom modified.

2.7.0.9 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) collection that only contains and accepts resources of this category.

TODO: What are the proposed ways to create collections? Post a feed to the service document? Put a new service document? Put a feed document to the desired location?

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

2.7.0.11 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[12] 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.[13]

3 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

from: Rest APIs must be hypertext driven[14]

[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.

3.1 Syntax vs. Semantic (Vocabulary)

The usage 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.[15]

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

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

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

[14] `http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven` (2011-12-20)

[15] `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

type. The purpose of standardized media types however is that there is a limited number of them and that they are reused by many sites and clients.

Large sites like Google, Facebook or Twitter can successfully attract enough developers to read their specifications and program clients accordingly. Usually one can also find client libraries that can help a lot. 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.

3.2 Data Models of Media Types

TODO:

- Ein generelles Daten Modell wäre hilfreich, um alle Medien Typen darauf zu projezieren und mit einer solchen Projektion dann innerhalb der Applikation zu arbeiten (TODO Schreier: warum muss dass Datenmodell total allgemeingültig sein, reicht es nicht vielleicht auch für eine Domäne?)
- Ein allgemeines Datenmodell könnte auch eine Hilfe sein als Zwischenschritt für Conversions zwischen Medientypen
- Es gibt kein allgemeines, sinnvolles Datenmodell für alle Medientypen
- Trotzdem können bestimmte hilfreiche Generalisierungen vorgenommen werden
  - Die meisten Resourcen haben bestimmte generische Metadaten die entweder im Medientyp kodiert werden können oder mit dem Medientyp zusammen persistiert werden müssen
  - Diese Metadaten finden sich auch in atom:entry wieder und sind: Autor, Updated, Titel, Summary, etag, id, name, links
  - Transitional Links vs Structural Links: `http://java.net/projects/jax-rs-spec/pages/Hypermedia`
  - Different categories of data: CSV, binary/plain text, large binary (video), tree (XML/JSON) (Referenz?)

3.2.1 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".[16] This stands in contrast to the rise of public JSON APIs and the decline of XML APIs.[17]

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.[18]

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.

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

[17] `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)

[18] 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)

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 1 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 1: 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.[19]

3.2.2 Hypermedia Support in JSON

TODO: discuss HAL

3.3 vCard, iCalendar, xCard and xCal

TODO:

- Textbasierte vs. XML Formate

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]

3.3.1 Hypermedia Support

The iCalendar standard defines a 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 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.

3.4 Derived JSON formats for PIM data

TODO Übereinstimmungen und Unterschiede vCard, portable contacts: Which fields of portable contacts are derived from vCard: `http://wiki.portablecontacts.net/w/page/17776141/schema`

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

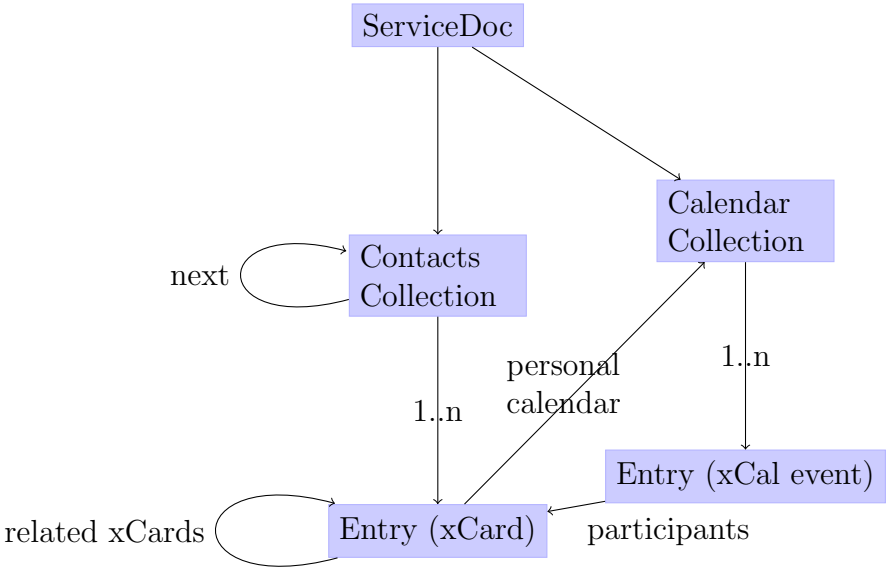

Figure 1: Hypermedia support in xCard and xCal

Calendar Formate nur kurz

3.5 Microformats, Microdata, RDFa

HTML documents are primarily meant to be rendered by browsers and interpreted by humans. It is hard for a computer to interpret the meaning of text and data included in an HTML document. Microformats, Microdata and RDFa define ways to add additional meta data to HTML that allows computers to identify structured data in HTML without having an impact on the rendering.[Ten12]

There is not yet an established term to refer to the three different formats. Practitioners use "structured data languages"[20], "machine-readable data format"[Hic11], "structured data markup"[21] 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.

3.5.1 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)[22] and the recent schema.org effort came from the three big search engines Google, Bing and Yahoo. Another use case is demonstrated by the Firefox plugin "Operator".[23] 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]

In the context of this work, Semantic annotations could be used inside the summary tag of Atom entries. A consumer of a feed of contact elements could thus use the data extracted

[20] `http://manu.sporny.org/2011/uber-comparison-rdfa-md-uf/` (2012-2-20)
[21] `http://googlewebmastercentral.blogspot.com/2011/06/introducing-schemaorg-search-engines.html` (2012-2-20)
[22] `http://tantek.com/log/2006/05.html` (2012-2-20)
[23] `https://addons.mozilla.org/en-US/firefox/addon/operator/` (2012-2-20)

from the annotated summary data to provide a tabular overview of the entries even without fetching the associated media resource of the entry.

TODO example

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)[24] uses semantic annotations to make content on a web site editable. In a traditional content management system, content is editable via HTML forms that are available as separate sites in addition to the normal view. The VIE Javascript library instead searches the HTML document for semantically annotated entities and dynamically builds 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.[25]

3.5.2 Format selection

With at least three different formats, a developer needs to settle on one to implement.[26] 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.[Hic11] It is also backed up by the schema.org effort of Google and Microsoft.[27] The schema.org vocabulary in turn has been mapped to the semantic world by researchers working on linked data.[28] 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[29], chair of the RDF Web Applications Working Group at the World Wide Web Consortium.

3.5.3 Producing Semantically annotated HTML

A recent discussion of possibilities to produce semantically annotated HTML pages can be found in [CDDM09, sec. 9.1.3]. The authors "discern two different ways in which the Semantic Web plays a key role in current Web engineering approaches: one is by the creation of Web applications starting from semantically described data, and the second is by the generation of semantic annotations from the Web engineering process."

In the case of this work only the second way is of interest, since we don't assume the data to be persisted in a semantic model and aim to generate semantic annotations. As an example for this way the book describes a method developed as part of a larger "Web Semantics Design Method" (WSDM). This method 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 uses XPointer expressions to link 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.

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

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

[26] It is possible to implement multiple formats in parallel.[Ten12]

[27] `http://schema.org/docs/gs.html#microdata_why` (2012-2-17)

[28] `http://schema.rdfs.org/about.html` (2012-2-17)

[29] `http://manu.sporny.org/category/rdfa/` (2012-2-20)

```
-@ 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 2: 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 3: Using a Microdata-aware data structure in a template

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 2. Even with the conciseness of the used template language Jade[30], the developer still has a lot to type.

Compared to the above listing 3 shows a template using a data structure that is aware of the used Microdata vocabulary and wraps an instance of a typed Microdata item with its properties. The scope method of the Microdata interface will add the itemscope, itemtype and itemid attributes to the nested div element. The prop method either augments a nested element as shown for the name property or creates the correct nested element. The method adds the itemprop attribute and puts the value for this property inside the element.

An implementation of this approach must take care of a few peculiarities.[Hic11] 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.

A template engine that should be extended as described above should allow to capture and manipulate nested HTML elements and to call methods of passed in objects.

3.6 Media Types for Collections

Vergleich ATOM mit Medientyp Collection+JSON

JSON formats for collections: Collection+JSON Mime-Type (approved in July 2011) by

[30] `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.

Mike Amundsen[31] JSON ATOM serialization implemented by Apache Abdera[32]

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.
- 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.

3.7 Media Type conversion

Konvertieren zwischen verschiedenen Representationen die nach aussen gegeben werden vs. Konvertieren zwischen äußerer Representation und Representation für die Datenbank.

An welchen Punkten in der Architektur muss/kann/soll konvertiert werden?

3.8 Updates with non isomorphic Media Types

How to handle updates, if the mediatypes are not isomorph?

How does Google handle PATCH in the calendar API?

Schnittmenge der Medientypen identifizieren

Einen kanonischen Medientypen festlegen für Updates?

Erweiterungsmöglichkeiten v. Medientypen nutzen um Isomorphie herzustellen

4 Design

4.1 Overview

4.2 Reusable Patterns and Components

Reuse is of course in general a good thing. In the context of Model Driven Development (MDD) and code generation it is especially import to identify code that is general enough to be provided by a library of framework and does not need to be generated.

Minimizing the generated code also minimizes the extend of drawbacks associated with code generation, most importantly conflicts between updates by the code generator and manual modification.

Concerns regarding Media Types that needs to be implemented differently for each different Media Type:

- validate the Media Type
- provide accessors to read, write parts of the Media Type

[31] `http://amundsen.com/media-types/collection/` (2012-1-7)

[32] `http://www.ibm.com/developerworks/library/x-atom2json/index.html` (2012-1-7) `https://cwiki.apache.org/ABDERA/json-serialization.html` (2012-1-7)

- serialize, deserialize the whole Media Type
- converters to other formats
- accessors to common interfaces (projection), e.g. common generic resource attributes or common attributes of a contact

Candidate areas for re-usability:

- link building, URL parsing
- HTML form building, parsing
- generic properties of resources, id
- resource types
- question to storage: does resource still match ETag? Has changed since?
- all links of a resource: Link: intern/extern/undefined, href, rel, title, text, media type
- bool function matchesMediaType(), getMediaType() auf WrappedEntry
- Prüfung, ob ein Update durchgeführt werden soll, gemäß ETAG, ifnotchanged
- Möglichkeit, DatenKlassen mit DatenTypen zu definieren wie in eZ Publish um automatische Views und Edit Ansichten zu ermöglichen.
- Creation of resources: POST to collection with SLUG Header, PUT to URI, normalization of SLUG Header
- Pagination (building and parsing of next and previous URIs, implementation of RFC5005), querying the collections entries provider with the correct parameters (offset, limit).
- Storage interface with transaction support. An application may for example need to notify an indexing component after some resource has been changed. – No transaction support: Every action that must happen in a transaction together with the resource change must be handled by the storage layer, must be aware of the storage technologie.

4.3 Interactions

4.3.1 Discovery of Collections

4.3.2 Synchronizing Collections

4.3.3 Modifying Resources

TODO Atompub requires the client to download the full representation before editing. This however could be avoided, if the feed would contain etags, probably as attributes to the link tag.

Google adds an etag attribute to the entry tag in its data api.[33]

[33] `http://code.google.com/apis/gdata/docs/2.0/reference.html#ResourceVersioning` (2012-2-13)

4.3.4 Special Reports, Search

TODO full text search in contacts -¿ OpenSearch

The client can compare the update time of any entry with the update time of an entry that it may have cached to avoid unnecessary loading.

complete collections [Not07] for months, weeks, days, time ranges - should not contain full representations but etags

TODO It would be good, if OpenSearch links would already be available in the service document to avoid loading the feed. TODO Google://opensearch service document

Microdata (subsection 3.5) enhanced summaries could be used by the client to display the results list.

OpenSearch already defines an extension to limit a search to a time range.[34]

4.3.5 Structural and Behavioral Rest Model

TODO Modelierung der Anwendung mit dem Meta Model nach Schreier.

Primary Resources: Contacts, Calendars, ... List Resources:

4.3.6 AtomPub for PIM data

Rob Yates `mailto:robert_yates@us.ibm.com`: CalATOM `http://robubu.com/?p=6` (2012-01-05) Draft: CardATOM `http://robubu.com/?p=10` (2012-01-05)

Rob Yates drafted a CalATOM specification.[Yat07] It references another old draft (draft-snell-atompub-feature-12) to mark a collection as a CalATOM collection. This seems unnecessary since we can specify that a collection accepts calendar resources and thus marking a collection as a calendar.

Furthermore the draft mentions a capability to query for events in a given time range. There is no equivalent for such a request for contacts.

In rest the draft only repeats or references the atompub standard and the xcal format. This draft could therefor be seen as indication that the existing standards are almost sufficient to define GroupWare APIs.

Google Data API also uses ATOM, but puts their tags directly inside the "atom:entry" tag instead of putting all of the content in the content element of an atom entry.[35]

- Atom entries could contain multiple link tags referring to alternative representations of the resource with different (media) type attributes.
- The synchronization algorithm is so simple that it does not even warrant an activity diagram:

  > Consume all entries of a (paged[36]) feed unless you see an entry older then the newest entry from your last synchronization run.

- If the content of the entry is presented inline then the client does not know the etag of the resource representation? TODO

[34] `http://www.opensearch.org/Specifications/OpenSearch/Extensions/Time/1.0/Draft_1` (2012-2-13)

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

[36] RFC 5005 - Feed Paging and Archiving

```
@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 4: Example of a JAX-RS annotated Resource class (by Marek Potociar)

4.4 Components

4.4.1 Dispatcher

The dispatcher selects the Java method (see 4.4.3.1) 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 4.

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[37], Apache Wink[38] and probably others. Jersey 2.0 is also expected to provide an API for the dispatcher.[39]:

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.
- 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

[37] `http://wiki.restlet.org/docs_2.1/13-restlet/27-restlet/326-restlet.html` (2012-2-6)
[38] called "Dynamic Resources" `http://incubator.apache.org/wink/1.1/html/5.1RegistrationandConfiguration.html` (2012-2-7)
[39] `http://java.net/jira/browse/JERSEY-842` (2012-2-6)

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.

TODO:

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.

4.4.2 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:

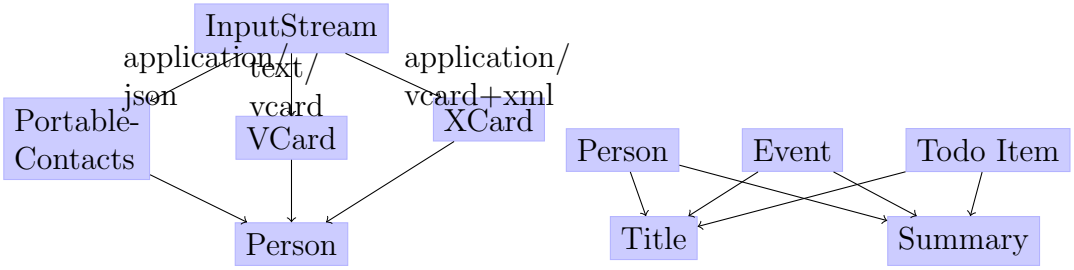

Figure 2: Facade examples with dependencies

> 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 5 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.

Figure 2 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.

4.4.2.1 Related work The idea for the Resource Facades concept was triggered by the use of the JavaBeans Activation Framework[40] (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.

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

```
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 5: API of the ResourceFacades component

[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.

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.[41] 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.

4.4.2.2 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 6 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.

4.4.3 Other components

4.4.3.1 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

[41] 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"

```
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 6: Implementing the facades approach with Scala's type system

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

4.4.3.2 CollectionStorage The collection storage interface offers the necessary means to store and retrieve resources. For clarity this interface is not further broken down into a read-only part and a full read-write interface.

A collection storage is instantiated with the knowledge of the collection it is responsible for. It therefor typically only returns resources that were previously stored through it although it may share its underlying persistency provider with other collection storage instances.

The life cycle of a collection storage is scoped to the application. It is therefor possible to attach memory based caching to this component.

The storage does not expose any support for transactions. Instead every method call represents one atomic action independent from other actions. Conditional request execution is therefor in the responsibility of the storage. Listing 7[42] shows a possibility for a lost update. Another request could have updated the resource between the etag check and the doUpdate() call.

4.4.3.3 Preparsed Request Components 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

[42]found in the JAX-RS specification on page 28.

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

Listing 7: Potential lost-update problem with JAX-RS

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

Listing 8: Verbosity of parsing Requests with JAX-RS

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 8 shows the verbosity of parsing a set of standard query parameters for a search interface. An alternative is shown in listing 9. The parsing of query parameters is delegated to the class SearchRequest.[43] 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.
- 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.[44]

4.4.3.4 Exkurs: Driving Dependency Injection further Paragraph 4.4.3.3 used dependency injection to cause the instantiation of a request scoped class that prepares information for the request method ("handleGet" in the example). This idea could be extended.

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

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

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

```
@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 9: Separating Request parsing from the Resource method

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 9. 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 as proposed in [DM11] and worth exploring in a separate work. One advantage of this approach would be that the processing pipeline is defined and configured in the same language then the rest of the application.

4.4.3.5 GenericResourceAttributes

4.5 Detailed Design Considerations

4.5.1 Inline feeds or feeds with links?

ATOM entries can either carry the full content inline in the content tag or link to the content.

- feeds could be dynamically created and inline only those entries that the client has not yet seen as indicated by the provided ETag or timestamp.
- Feeds with links will most probably result in many further requests by the client to get the individual entries
- Inline feeds may cause unnecessary work and deliver entries that the client has already seen
- A compromise could be to provide the first page of a paginated feed with links and all additional pages with inlined entries

A final decision about this subject should be done based on utilization and performance data of a real world installation.

4.6 Client Design

What needs a client to know, how does it need to work?

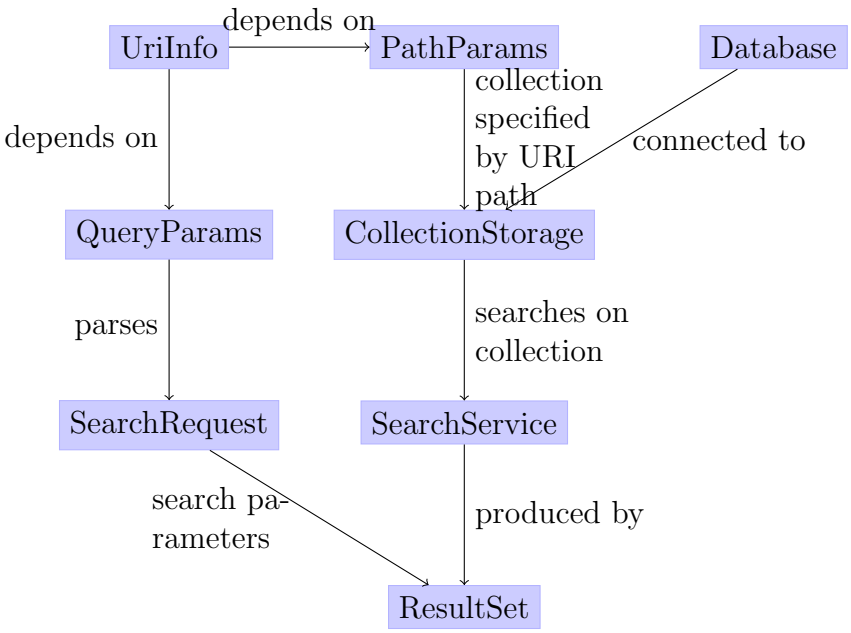

Figure 3: Building a processing pipeline with Dependency Injection

5 Summary and Conclusions

References

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

[Amu10] Amundsen, Mike: *Fielding Property Maps*. `http://amundsen.com/examples/fielding-props/`, 3 2010

[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

[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

[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

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

[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

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

[GZLW11] Graf, Sebastian ; Zholudev, Vyacheslav ; Lewandowski, Lukas ; Waldvogel, Marcel: Hecate, managing authorization with RESTful XML. In: Alarcón, Rosa (Hrsg.) ; Pautasso, Cesare (Hrsg.) ; Wilde, Erik (Hrsg.): *WS-REST*, ACM, 2011. – ISBN 978–1–4503–0623–2, S. 51–58

[Hic11] 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

[HS09] Hadley, Marc ; Sandoz, Paul: *JSR 311: JAX-RS: The Java API for RESTful Web Services Version 1.1.* http://www.jcp.org/en/jsr/detail?id=311, September 2009

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

[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

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

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

[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

[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

[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

[Ste08] Steimann, Friedrich: Role + counter-role = relationship + collaboration. In: *23rd Annual ACM Conference on Object-Oriented Programming. Systems, Languages, and Applications*. Nashville, Tennessee, October 2008

[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

[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

[Yat07] Yates, Rob: *CalAtom*. `http://robubu.com/CalAtom/calatom-draft-00.txt`. `http://robubu.com/CalAtom/calatom-draft-00.txt`. Version: April 2007

[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