Thomas Koch
thomas@koch.ro
matriculation number 7250371

February 14, 2012
Fernuniversität Hagen
Faculty of mathematics and computer science

Contents

1. Introduction and Motivation

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.

2. Requirements

2.1. Scope

This work defines a protocol to share

2.2. 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.

2.3. 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

2.3.0.1. 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]:

1. Ability to use multiple address books with hierarchical layout.

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

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

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 2.8 for a discussion of those.

2.3.0.2. 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.

2.4. User Classes and Characteristics

TODO: plain Web Browsers, Desktop applications (PIM Suites), JavaScript Gadgets/Widgets (see OpenSocial) probably with WebStorage[3]

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

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

2.5. 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.

2.6. Design and Implementation Constraints

2.7. Specific Requirements

2.7.0.3. 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.

2.8. Excluded requirements

2.8.0.4. 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.

2.8.0.5. 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.

2.8.0.6. 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].

2.8.0.7. 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.

2.8.0.8. 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.

2.8.0.9. 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?

2.8.0.10. 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.

2.8.0.11. 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[4] 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.[5]

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[6]

[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

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.[7]

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

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

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

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

[7] `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
- 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 data models 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".[8] This stands in contrast to the rise of public JSON APIs and the decline of XML APIs.[9]

Web developers often prefer JSON over XML because it is perceived as easier to parse, process and smaller in size. XML in comparison is seen as complicate, slow to process and larger in size. A strong argument for JSON as the preferred format for JavaScript applications is that JSON is a subset of JavaScript.[10]

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. If two developers independently take the same XML media type as a model to develop a JSON format, they can still come up with different results. There is no standard mapping from XML schemes to JSON structures.

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

[9] `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)

[10] 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)

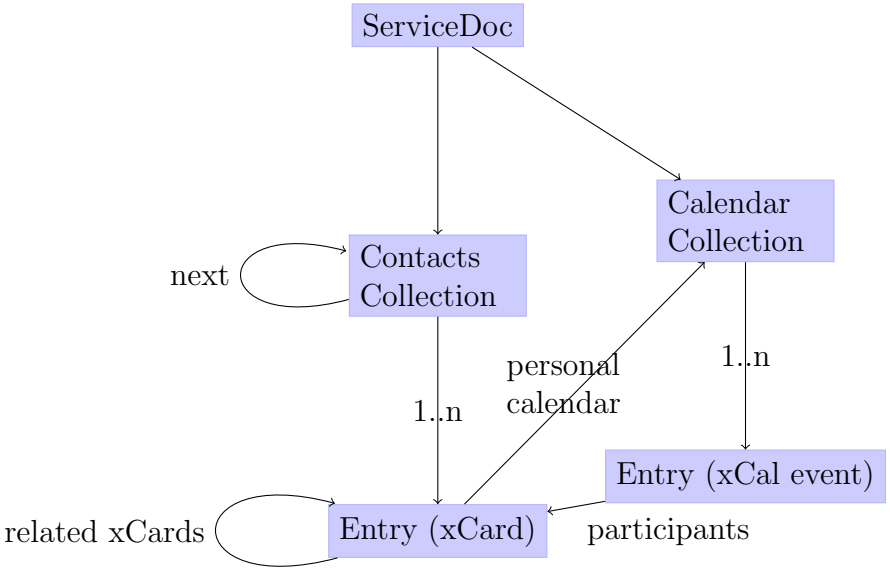

Figure 1: Hypermedia support in xCard and xCal

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. **??** 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.

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.[11]

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.

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

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`

Calendar Formate nur kurz

3.5. Microdata

TODO Microformats could also be useful to markup data elements in a (X)HTML summmary of an Atom entry. In this case a web client could use the data from the summary in a tabular view without fetching the full enclosed data in case it is not contained in the content tag.

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 Mike Amundsen[12] JSON ATOM serialization implemented by Apache Abdera[13]

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

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

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

4. Design

4.1. Overview

4.2. Interactions

4.2.1. Discovery of Collections

4.2.2. Synchronizing Collections

4.2.3. 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.[14]

4.2.4. 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.[15]

4.2.5. Structural and Behavioral Rest Model

TODO Modelierung der Anwendung mit dem Meta Model nach Schreier.

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

4.2.6. 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.

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

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

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

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.[16]

- 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[17]) 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

4.3. Components

4.3.1. Dispatcher

The dispatcher selects the Java method (see 4.3.3.1) 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 1.

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[18], Apache Wink[19] and probably others. Jersey 2.0 is also expected to provide an API for the dispatcher.[20]:

Advantages of a dynamic dispatcher configuration would be:

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

[17] RFC 5005 - Feed Paging and Archiving

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

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

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

- 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 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.

4.3.2. DataFacades handle different Media Types

hide the details of the Resource Representation from the resource method

Resource methods have access to a DataHandler[2] that wraps the declared Media Type of the data and the DataSource. Users can then request Facades to the data from the DataHandler. The DataHandler has access to a registry of Facade factories.

Facades provide read-only access to the data Facades can depend on other Facades and thus reuse work Facades could do: Parse the InputStream with a common JSON/XML framework Could check whether the data represents a concept like a contact or calendar item, independent from the underlying Media Type (vCard, xCard, portable contacts, iCal, xCal) Could provide a transformation to another representation (XML¡-¿JSON, xCard to portable contacts) could extract generic informations from different media types: Title, Authors, Updated, Summary A Facade (Java Interface) that provides access to the Name of a Contact Resource could be implemented by different classes that could work either on vCard, xCard or Portable Contacts. The resource method does not need to be aware or the original media type of the data.

The concept is powerful because Facades can easily be added without changing existing classes. In JAX-RS the resource Method needs to decide, on which representation of the

```
interface FacadeFactory<T> { T build(DataHandler) }

interface FacadeRegistry {
  // Lookup in this map is first done for the full MediaType, then only for
  // the Top level media Type and afterwards for */*
  Map<MediaType, FacadeFactory>

  Object getFacade(DataHandler, Class)

  // calls itself recursively for transient dependencies
  private hasFacade(DataHandler, Class)

  register(FacadeFactory xy, Iterable<MediaType> matchingTypes,
           Iterable<Class> requiredDependencyFacades)
  ))
}

class DataHandler {
  private FacadeRegistry
  private MediaType
  private DataSource // encapsulates e.g. an InputStream

  getFacade(Class) // delegate call to FacadeRegistry
  hasFacade(Class) // delegate call to FacadeRegistry
}
```

Listing 2: API of the DataFacades component

data it wants to work. With this concept, a resource method could use several different available Facades to access the data.

A sketch of the API:

Comparison to slightly similar things:

The JAF's (AWT's) java.awt.datatransfer.Transferable interface is similar, but has no concept of reuse or composition of a DataFlavor by another. It is designed with the needs of a Desktop clipboard in mind. No use of Generics.

It is possible in JAX-RS to get a MessageBodyReader for the creation of another facade. The necessary code to do so however is verbose and every MessageBodyReader would again parse the message body InputStream without reuse of work already done by another MessageBodyReader.

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. Some aspects however may require extra care:

- Dependencies are parameterized with the Media Type of the data.
- The scope of an instance is bound to the DataHandler which in most cases may be equivalent to the Request scope, but this can't be guaranteed.
- Each DataHandler manages its own view of available classes.

```
1  trait Storage[ReqFacade] {
2   def create(id: String,
3              body: DataHandler with FacadeFactory[ReqFacade])
4  }
5
6  class PostToCollection[StorageReqFacade]
7              (storage: Storage[StorageReqFacade]) {
8   type MessageBody = DataHandlery
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 3: Implementing the facades approach with Scala's type system

4.3.2.1. 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 3 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 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.

4.3.3. Other components

4.3.3.1. 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 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.

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

Listing 4: 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 5: Verbosity of parsing Requests with JAX-RS

4.3.3.2. 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 4[21] shows a possibility for a lost update. Another request could have updated the resource between the etag check and the doUpdate() call.

4.3.3.3. 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 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 5 shows the verbosity of parsing a set of standard query parameters for a search interface. An alternative is shown in listing 6. The parsing of query parameters is delegated to the class SearchRequest.[22] 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.

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

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

```
@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 6: Separating Request parsing from the Resource 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.[23]

4.3.3.4. Exkurs: Driving Dependency Injection further Paragraph 4.3.3.3 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 2 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 6. 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.

4.3.3.5. GenericResourceAttributes

4.4. Detailed Design Considerations

4.4.1. Inline feeds or feeds with links?

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

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

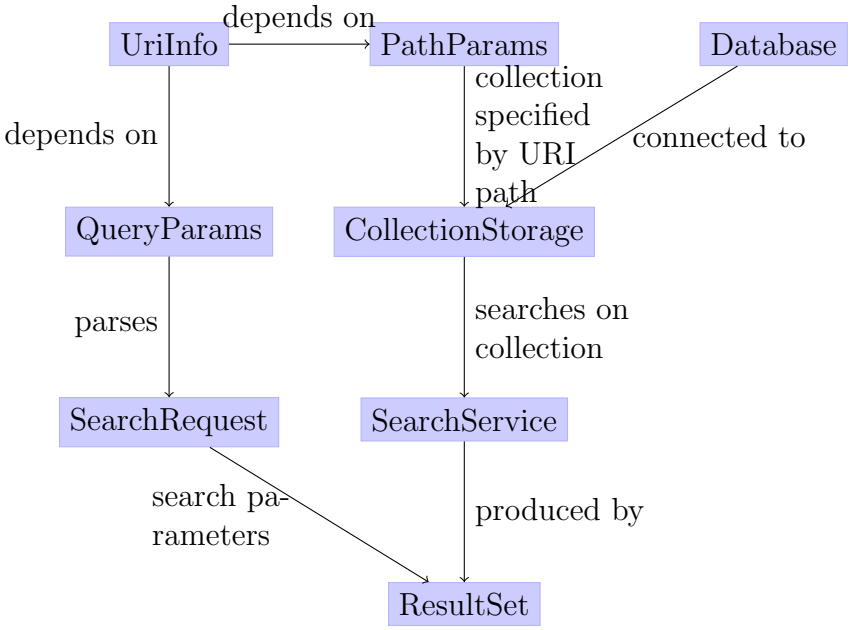

Figure 2: Building a processing pipeline with Dependency Injection

- 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.

4.5. Client Design

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

4.6. Related work

4.6.1. 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/
- 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

4.6.2. WebDAV, CalDAV, CardDAV

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

– Lisa Dusseault, February 29, 2008[24]

> 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.

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

– Ted Leung, March 6, 2008[25]

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 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

4.6.3. 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.[26] 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:[27]

> 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.

[25] http://www.sauria.com/blog/2008/03/06/google-contacts-and-carddav/ (2012-1-6)

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

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

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.

4.6.3.1. 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]

> 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[28] 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:

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

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

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,[29] 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.[30] Consequently the newest OpenSocial specification deprecates any reference to the ATOM format.

In Algermissen's classification (A.0.5), 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.

4.6.4. 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[31]

5. Reusable 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:

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

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

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

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

Why vCard/CardDav: many clients
Why OpenSocial / Portable Contacts:

- used by Google, LinkedIn,
- used in Enterprise applications like Attlassian tools (Jira, Confluence, ...), Nuxeo CMS, ...
- OpenSocial can be used to implement inhouse portals and populate it with data from the companies GroupWare

A. Evaluation of APIs

A.0.5. Algermissen's Classification of HTTP-based APIs

Jan Algermissen, proposes a "Classification of HTTP-based APIs" in February 2010.[32] He identifies and names a five level order of HTTP based APIs. Each level adds one constraint to be obeyed. The last level obeys all five constraints and is RESTful. (Table 3)

level name	constraint violated	violation description
WS-*	Identification of Resources	Only service endpoint is identified by URI. No resources exposed.
RPC URI-Tunneling	Manipulation of Resources through Representations	SOAP body contains operation name, message not transferred to manipulate resource state.
HTTP-based Type I	Self-Descriptive Messages	Message semantics depend on action specified in message body.
HTTP-based Type II	Hypermedia as the Engine of Application State	Application state machine known at design time.

Table 3: Classification of HTTP-based APIs after Algermissen

In addition to that, Algermissen also provides a list of "Effect on System Properties and Cost" of the different API styles and acknowledges that the initial costs of developing a REST API may be higher compared to the other styles but lower in the long run.

B. Persistency for Groupware Data

Relational Databases vs. NoSQL databases vs. plain files

Relational databases are not practical for contacts, events or todos. Common patterns in systems that use relational DBs for that purpose:

- artificial limits of entries, e.g. only 3 email addresses per contact, because there are only three columns email1, email2 and email3.
- Fields for custom data like custom1 to customX
- EAV pattern: tables like: id, foreign_id, type, value

C. Synchronizing a large collection

How to efficiently synchronize a large collection of contacts with the server without checking each contact for changes?

Portable Contacts has a filter "updatedSince".

How is synchronization done in CardDAV?

"Feed Paging for Efficient Feed Synchronization" thread at ATOM-Syntax `http://www.imc.org/atom-syntax/mail-archive/msg20060.html`

HTTP Delata encoding with Feed: `http://www.wyman.us/main/2004/09/using_rfc3229_w.html` (2012-1-6), implementations: `http://www.wyman.us/main/2004/09/implementations.html` (2011-1-6)

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

type of data	XML	JSON	semantic	microformat
Calendar	xCal	Google calendar API	http://www.w3.org/TR/rdfcal	hCalendar
Contact	xCard	portable contacts, jCard	friend of a friend	hCard
Resume	HR XML		Description of a Career	hResume

Table 4: data in different formats

"There's also an advanced technique involving etags which keeps track of when each etag was issued and if an old etag is sent with the request then everything since then is sent back in the response. Someone will chime in with the reference." Eric Scheid, http://www.imc.org/atom-syntax/mail-archive/msg19832.html (2012-1-6)

Microsoft specification FeedSync (formerly Simple Sharing Extension) http://feedsyncsamples.codeplex.com, comment http://notes.kateva.org/2008/01/microsoft-feedsync-what-heck-is-html http://web.archive.org/web/20081114142152/http://www.snellspace.com/wp/?p=818

D. Media Types

D.1. Media Type conversion

RFC 5023 "Collections are represented as Atom Feeds"

Is conneg (content negotiation) useful? No: Norman Walsh, 2003, it can lead to hard to debug bugshttp://norman.walsh.name/2003/07/02/conneg (2011-1-9), Joe Gregorio, 2003,I can't communicate the mime type to request to a third service if I can only give an URIhttp://bitworking.org/news/WebServicesAndContentNegotiation (2011-1-9) Yes: Jerome Louvel, 2006, I could additionally provide URIs that override the accept headers with query parameters like ?format=json.http://blog.noelios.com/2006/11/15/reconsidering-content (2011-1-9)

> No single data representation is ideal for every client. This protocol defines representations for each resource in three widely supported formats, JSON [RFC4627], XML, and Atom [RFC4287] / AtomPub [RFC5023], using a set of generic mapping rules. The mapping rules allow a server to write to a single interface rather than implementing the protocol three times.

[Ope11, Core API Server]

In 2007, a project called microjson wanted to standardize json representations of microformat data structures.[33]

The project identified the need for a json schema:[34]

> If there are standard microJSON formats for transfer of certain datasets, there will be a need to validate that data to ensure that it is infact valid format. To validate a format you need something that details the structure, data content types and required data. Sounds like we'll be needing a schema for each microJSON format.

jCard example from microjson.org[35]

[33] http://notizblog.org/2007/09/16/microjson-microformats-in-json/ (2011-12-19)
[34] http://web.archive.org/web/20080524003749/http://microjson.org/wiki/Schemas (2022-12-19)
[35] http://web.archive.org/web/20080517003233/http://microjson.org/wiki/JCard (2011-12-19)

```
{
"vcard":{
  "name":{
    "given":"John",
    "additional":"Paul",
    "family":"Smith"
  },
  "org":"Company Corp",
  "email":"john@companycorp.com",
  "address":{
    "street":"50 Main Street",
    "locality":"Cityville",
    "region":"Stateshire",
    "postalCode":"1234abc",
    "country":"Someplace"
  },
  "tel":"111-222-333",
  "aim":"johnsmith",
  "yim":"smithjohn"
}
```

D.2. Example: vCard

```
<?xml version="1.0" encoding="UTF-8"?>
<vcards xmlns="urn:ietf:params:xml:ns:vcard-4.0">
  <vcard>
    <fn><text>Simon Perreault</text></fn>
    <n>
      <surname>Perreault</surname>
      <given>Simon</given>
      <additional/>
      <prefix/>
      <suffix>ing. jr</suffix>
      <suffix>M.Sc.</suffix>
    </n>
    <bday><date>--0203</date></bday>
    <anniversary>
      <date-time>20090808T1430-0500</date-time>
    </anniversary>
    <gender><sex>M</sex></gender>
    <lang>
      <parameters><pref><integer>1</integer></pref></parameters>
      <language-tag>fr</language-tag>
    </lang>
    <lang>
      <parameters><pref><integer>2</integer></pref></parameters>
      <language-tag>en</language-tag>
    </lang>
    <org>
      <parameters><type><text>work</text></type></parameters>
```

```
      <text>Viagenie</text>
    </org>
    <adr>
      <parameters>
        <type><text>work</text></type>
        <label><text>Simon Perreault
2875 boul. Laurier, suite D2-630
Quebec, QC, Canada
G1V 2M2</text></label>
      </parameters>
      <pobox/>
      <ext/>
      <street>2875 boul. Laurier, suite D2-630</street>
      <locality>Quebec</locality>
      <region>QC</region>
      <code>G1V 2M2</code>
      <country>Canada</country>
    </adr>
    <tel>
      <parameters>
        <type>
          <text>work</text>
          <text>voice</text>
        </type>
      </parameters>
      <uri>tel:+1-418-656-9254;ext=102</uri>
    </tel>
    <tel>
      <parameters>
        <type>
          <text>work</text>
          <text>text</text>
          <text>voice</text>
          <text>cell</text>
          <text>video</text>
        </type>
      </parameters>
      <uri>tel:+1-418-262-6501</uri>
    </tel>
    <email>
      <parameters><type><text>work</text></type></parameters>
      <text>simon.perreault@viagenie.ca</text>
    </email>
    <geo>
      <parameters><type><text>work</text></type></parameters>
      <uri>geo:46.766336,-71.28955</uri>
    </geo>
    <key>
      <parameters><type><text>work</text></type></parameters>
      <uri>http://www.viagenie.ca/simon.perreault/simon.asc</uri>
    </key>
```

```
    <tz><text>America/Montreal</text></tz>
    <url>
      <parameters><type><text>home</text></type></parameters>
      <uri>http://nomis80.org</uri>
    </url>
  </vcard>
</vcards>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<vcards xmlns="urn:ietf:params:xml:ns:vcard-4.0">
  <vcard>
    <fn><text>Simon Perreault</text></fn>
    <n>
      <surname>Perreault</surname>
      <given>Simon</given>
      <suffix>ing. jr</suffix>
      <suffix>M.Sc.</suffix>
    </n>
    <bday day="02" month="03" />
    <anniversary format="date-time">20090808T1430-0500</anniversary>
    <gender>M</gender>
    <lang pref="1">fr</lang>
    <lang pref="2">en</lang>
    <org type="work">Viagenie</org>
    <adr type="work">
      <label>Simon Perreault
2875 boul. Laurier, suite D2-630
Quebec, QC, Canada
G1V 2M2</label>
      <street>2875 boul. Laurier, suite D2-630</street>
      <locality>Quebec</locality>
      <region>QC</region>
      <code>G1V 2M2</code>
      <country>Canada</country>
    </adr>
    <tel>
      <type>work</type>
      <type>voice</type>
      <uri>tel:+1-418-656-9254;ext=102</uri>
    </tel>
    <tel>
      <type>work</type>
      <type>text</type>
      <type>voice</type>
      <type>cell</type>
      <type>video</type>
      <uri>tel:+1-418-262-6501</uri>
    </tel>
    <email type="work">simon.perreault@viagenie.ca</email>
    <geo type="work">
      <uri>geo:46.766336,-71.28955</uri>
```

```
      </geo>
      <key type="work">
        <uri>http://www.viagenie.ca/simon.perreault/simon.asc</uri>
      </key>
      <tz>America/Montreal</tz>
      <url type="home">
        <uri>http://nomis80.org</uri>
      </url>
    </vcard>
  </vcards>
```

D.3. HFactor

Mike Amundsen defines a method to asses media types that he calls "HFactor".[36] The HFactor distinguishes different types of support for links and indicates which of those are provided by a reviewed media type.

Amundsen did reviews of a couple of media types. Unfortunately these do not include `vcard+xml` or `calendar+xml`. I'll try to identify the HFactors of both here.

The different types of link support have two letter acronyms and fall in two categories: Link support values, with the first letter "L" and Control data support, first letter "C".

- Link Support for
  - `LE` embedded links (HTTP GET)
  - `LO` out-bound navigational links (HTTP GET)
  - `LT` templated queries (HTTP GET)
  - `LN` non-idempotent updates (HTTP POST)
  - `LI` idempotent updates (HTTP PUT, DELETE)
- Control Data Support to
  - `CR` modify control data for read requests (e.g. `HTTP Accept-*` headers)
  - `CU` modify control data for update requests (e. g. `Content-*` headers)
  - `CM` indicate the interface method for requests (e.g. HTTP GET,POST,PUT,DELETE methods)
  - `CL` add semantic meaning to link elements using link relations (e.g. HTML rel attribute)

E. Hypermedia in RESTful applications

> The model application is therefore an engine that moves from one state to the next by examining and choosing from among the alternative state transitions in the current set of representations.

[Fie00, sec. 5.3, p.103]

[36] `http://amundsen.com/hypermedia/` (2011-12-21)

E.1. Hypermedia in OpenSocial

Webfinger, e.g. get a profile picture from an email address

Danger: One can trigger na http request by sending an email.

F. Selection of components

Apache Shindig for Open Social, includes client tests

http://code.google.com/p/kolab-android/

https://evolvis.org/projects/kolab-ws/

http://packages.ubuntu.com/source/maverick/dovecot-metadata-plugin https://launchpad.net/ubu
metadata-plugin/8-0ubuntu1

F.1. REST framework

Jersey recommended by [Kai11] above Restfulie and RESTeasy because of maturity and flexibility.

Jersey has a atompub-contact client/server example app.

Why not Jersey in the end?

- JAX-RS assumes, that Paths are defined on the classes that represent the resources.
  - This couples the "location" of a resource to its implementation.
  - This leads to copied code. Given an URL pattern like `/{AUTHORITY}/{COLLECTION}/{ENTRY}` In this case the resource classes for authority, collection and entry would each need to parse the authority section of the path.
  - If paths are not defined on resource classes, it is not possible to make use of JAX-RS' capabilities of declarative hyperlink building (@REF annotation).
- The dispatch to a request handler method has in our case three orthogonal parameters: HTTP verb, Media type, path. It would be preferable to handle these parameters independent of each other. The only way to handle at least the path dispatch separately is with the help of sub resources. This still leaves HTTP verb and Media type to be handled together.

  The sub resource mechanism additionally suffers from the shortcoming that it does not allow to specify an empty path.[37] This makes it impossible to return a sub resource and annotate a method that should handle the case that no additional path elements remain to be matched.
- Debugging is hard. It's not trivial to find out, why Jersey did not select a request handler or provider as the developer intended.
- Jersey's parameter injection can not be used together with a dependency injection framework like Guice or Spring.

Comments on Restlet:

- A couple of core classes of Restlet extend a class called Restlet whose purpose is only vaguely defined but the type inheritance does not correspond to an "is-a" relationship. This might indicate a questionable architecture of the framework.

[37] `http://java.net/jira/browse/JERSEY-536` (2012-01-21)

- Classes in Restlet are generally mutable. The Javadocs of several classes, e.g. org.restlet.Restlet and subclasses even come with a warning note but do not expose any information about the thread-safety of their methods:

  > Concurrency note: instances of this class or its subclasses can be invoked by several threads at the same time and therefore must be thread-safe. You should be especially careful when storing state in member variables.

-

F.2. VCard

ical4j best documented best code is used by most active also supports icalendar is immutable!!!

G. Testing

How to test the ReST/CardDAV interface?

Portable Contacts test client at plaxo `http://www.plaxo.com/pdata/testClient` `http://code.google.com/p/rest-assured/` `http://restfuse.com/`

H. Standards

H.1. Contacts / Persons

RFC 6450 vCard Format Specification
This document defines the vCard data format for representing and exchanging a variety of information about individuals and other entities (e.g., formatted and structured name and delivery addresses, email address, multiple telephone numbers, photograph, logo, audio clips, etc.). This is the new version and obsoletes RFCs 2425, 2426, and 4770, and updates RFC 2739.

RFC 6351 xCard: vCard XML Representation
This document defines the XML schema of the vCard data format.

Portable Contacts, OpenSocial
Portable Contacts defines contact data structures and a ReST API. It has been integrated in the OpenSocial standard.

Nepomuk Semantic Desktop Contact Ontology

Friend of a friend (FOAF)
FOAF is a

hCard

jCard

H.2. Calendaring

RFC 5545 Internet Calendaring and Scheduling Core Object Specification

iCalendar is the core data schema for calendaring information. This is the new version and obsoletes RFC2445

RFC 6321 xCal: The XML format for iCalendar

This specification defines a format for representing iCalendar data in XML. More specifically, is to define an XML format that allows iCalendar data to be converted to XML, and then back to iCalendar, without losing any semantic meaning in the data. Anyone creating XML calendar data according to this specification will know that their data can be converted to a valid iCalendar representation as well.

CalWS RESTful Web Services Protocol for Calendaring

This document, developed by the XML Technical Committee, specifies a RESTful web services Protocol for calendaring operations. This protocol has been contributed to OASIS WS-CALENDAR as a component of the WS-CALENDAR Specification under development by OASIS.

Google Calendar API V3

While not being a standard, the Google Calendar API is RESTful and will surely be implemented by many client applications. It's remarkable that the API supports partial GETs returning only specified fields and the HTTP PATCH verb to update only specified fields.

Open Services for Lifecycle Collaboration (OSLC)

uses FOAF person `http://open-services.net/bin/view/Main/OSLCCoreSpecAppendixA?sortcol=table;up=#foaf_Person_Resource`

provides change management, some overlapping to iCal TODOs `http://open-services.net/bin/view/Main/CmSpecificationV2`

reference implementation: `http://eclipse.org/lyo`

H.3. Scheduling

RFC 5546 iCalendar Transport-Independent Interoperability Protocol (iTIP)

Scheduling Events, BusyTime, To-dos and Journal Entries; Specifies the mechanisms for calendaring event interchange between calendar servers. This is the new version and obsoletes RFC2446

RFC 6047 iCalendar Message-Based Interoperability Protocol (iMIP)

Specifies how to exchange calendaring data via e-mail. This is the new version and obsoletes RFC2447.

H.4. Relations and Links

Xhtml Friends Network (XFN)

One of the relations returned by Google's webfinger.

Webfinger
Webfinger in Firefox Contacts Add-On http://mozillalabs.com/blog/2010/03/contacts-in-the-

RFC 6415 Web Host Metadata

Extensible Resource Descriptor (XRD)

H.5. out of scope

HR XML

The HR-XML Consortium is the only independent, non-profit, volunteer-led organization dedicated to the development and promotion of a standard suite of XML specifications to enable e-business and the automation of human resources-related data exchanges.

OMA Converged Address Book V1.0

Standard by the Open Mobile Alliance defining data structures and synchronization of contact data. It references vCard.

Open Collaboration Services

Also contains data structures for persons and events but does not reuse any known standard. See this thread: http://lists.freedesktop.org/archives/ocs/2011-December/000136.html

W3C Contacts API

A standard on how address books cold be accessed on devices or from JavaScript inside a Web Browser. The standard references vCard, OMA Converged Address Book and Portable Contacts.

W3C vCard ontology

W3C PIM ontology

I. People, Groups and Organizations

I.0.0.1. People

Eran Hammer-Lahav
http://hueniverse.com Yahoo!, OAuth

Eliot Lear ¡lear@cisco.com¿
IETF Calsify WG chair

James Snell
`http://chmod777self.blogspot.com/`

Apache Abdera committer, OpenSocial, IBM

Joseph Smarr

former Plaxo now Google presentation about portable contacts at vcarddav wg http://tools.ietf.org/age
2.pdf http://josephsmarr.com http://anyasq.com/79-im-a-technical-lead-on-the-google+-team

Julian Reschke ¡julian.reschke@gmx.de¿

Lisa Dusseault

Lisa Dusseault is a development manager and standards architect at the Open Source Applications Foundation, where she's involved in the Chandler, Cosmo and Scooby projects. Previously, Lisa came from Xythos, an Internet startup where she was development manager for four years. She has also been an IETF contributor on various Internet applications protocols for eight years now, and continues to do this kind of work at OSAF. She co-chairs the IETF IMAP extensions and CALSIFY (Calendaring and Scheduling Standards Simplification) Working Groups. She is also the author of a book on WebDAV and co-author of CalDAV, an open and interoperable protocol for calendar access and sharing.

Mark Nottingham

Mike Amundsen ¡mamund@yahoo.com¿
`http://amundsen.com`

Mike Conley

`http://mikeconley.ca/blog/` working on a new address book for Thunderbird: `https://wiki.mozilla.org/Thunderbird/tb-planning`

Peter Saint-Andre ¡stpeter@stpeter.im¿

IETF Calsify WG area director

J. Implementations

J.1. Servers

Cyn.in
Python, Open Core

DAViCal

PHP, SQL storage, CalDAV, CardDav

eGroupWare

eXo Platform
Open Core, Java, AGPL, participates in OpenSocial?

Group-Office
PHP, AGPL

Horde

OBM Groupware
PHP, GPL

Open-Xchange
Java, In 2006 a Debian packaging attempt was canceled because upstream decided not to publish security updates for the open source version anymore.[38]

owncloud

ownCloud supports syncing of calendar and contacts information via the CalDAV and CardDAV protocols.

Scalix
Open Core Scalix Public License (SPL) based on MPL, requires to show the Scalix Logo

Simple Groupware
PHP, GPL, SQL

SOGo
CalDAV and CardDAV, written in Objective-C

Tiki Wiki
PHP, SQL Contacts `http://doc.tiki.org/Contacts`, Calendar `http://doc.tiki.org/Calendar` iCal export apparently no CardDAV/CalDAV many many features!

Tine 2.0
Tine is not eGroupWare

Zarafa
PHP, MySQL IIRC it uses an Entity-Attribute-Value pattern to store its data in the relational db.

Zimbra
Open Core, Own license (Zimbra Public License), RFP since 2008 open: http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=498316

[38] `http://web.archive.org/web/20100510133805/http://seraphyn.deveth.org/archives/10-Keine-Zukunft-in-der-freien-Version-von-Open-Exchange-auf-Debian.html` (2011-12-19)

J.2. Clients

Spicebird
built on top of Thunderbird with Calendar

Thunderbird

CardDAV via SoCO connector http://www.sogo.nu/fr/downloads/frontends.html

WebiCal
Java, YUI, Web frontend for a CalDAV server, uses iCal4J

Evolution, Evolution Data Server

KDE Kontact, Akonadi

more CardDAV
http://wiki.davical.org/w/CardDAV/Clients http://en.wikipedia.org/wiki/CardDAV#Implementations

more CalDAV
http://wiki.davical.org/w/CalDAV_Clients http://en.wikipedia.org/wiki/CalDAV#Implementations

J.3. Web Services

J.4. Others

K. Links

- http://thesocialweb.tv
- http://www.vogella.de/articles/REST/article.html REST with Java (JAX-RS) using Jersey - Tutorial
- https://addons.mozilla.org/de/firefox/addon/restclient/
- http://dataportability.org/ still active?
- http://tech.groups.yahoo.com/group/rest-discuss/messages/17242?threaded=1&m=e&var=1&tidx=1 REST and Semantic
- http://stackoverflow.com/questions/2669926/practical-advice-on-using-jersey-and-
- http://macstrac.blogspot.com/2009/01/jax-rs-as-one-web-framework-to-rule.html
- Calypso — CalDAV/CardDAV/WebDAV for Android and Evolution
- http://www.xfront.com/files/articles-1.html
- http://buzzword.org.uk/swignition/uf
- http://json-schema.org/

- Resource Directory Description Language (RDDL)
- http://blogs.oracle.com/sandoz/entry/jersey_and_abdera_with_a http://weblogs.java.net/blog/mhadley/archive/2008/02/integrating_jer_2.html

IANA link relations registry http://www.iana.org/assignments/link-relations/link-relatio
xml

K.1. ATOM

ATOM landscape overview http://dret.typepad.com/dretblog/atom-landscape.html WebDAV vs. ATOM: http://intertwingly.net/wiki/pie/WebDav http://intertwingly.net/wiki/pie/WebDavVsAtom google webdav atom Why didn't ATOM succeed (more)? http://bitworking.org/news/425/atompub-is-a-failure (2012-01-06)

K.2. XML and JSON

- http://blog.jclark.com/
- http://code.google.com/p/jaql/wiki/Builtin_functions#xml
- http://www.webmasterworld.com/xml/3603303.htm
- http://www.xml.com/pub/a/2006/05/31/converting-between-xml-and-json.html?page=3
- http://goessner.net/download/prj/jsonxml/
- http://www.w3.org/2011/10/integration-workshop/agenda.html
- http://jsonml.org/

K.3. Apache Shindig

RPC vs. REST API for Shindig/OpenSocial: http://groups.google.com/group/opensocial-and-ga
browse_thread/thread/a4ddf7cd09f90237/5cfa1658e1c1d698?lnk=gst&q=rest#5cfa1658e1c1d698
http://groups.google.com/group/opensocial-and-gadgets-spec/browse_thread/thread/d1a5627fb6e686ce/d27d47dee92a87b2 One argument was support for batching. A restful batching proposal didn't get consensus: https://docs.google.com/View?docid=dc43mmng_23fdbpp7hd&pli=1

Flow of REST requests in Shindig https://sites.google.com/site/opensocialarticles/Home/shindig-rest-java

Google+ is likely to become OpenSocial enabled: http://groups.google.com/group/opensocial-and-gadgets-spec/browse_thread/thread/1187241df6759a9a

Shindig issues to implement OpenSocial 2.0 https://docs.google.com/spreadsheet/ccc?key=0AihdZBncP3KzdGN3dVl3MFpIUlk2TXIyR3hfUDhHZUE&hl=en_US#gid=0

How Shindig supports extensions: https://cwiki.apache.org/confluence/display/SHINDIG/Arbitrary+Extensions+to+Apache+Shindig%27s+Data+Model

Videos about some 2.0 OS features http://groups.google.com/group/opensocial-and-gadgets-browse_thread/thread/7b911edfb1bb3b4d

OS and RDF http://groups.google.com/group/opensocial-and-gadgets-spec/browse_thread/thread/20f62d627003509b

OpenSocial Development Environment (OSDE, Eclipse Plugin) https://sites.google.com/site/opensocialdevenv

https://cwiki.apache.org/confluence/display/SHINDIG/Providing+your+own+data+service+implementation

K.4. Socialsite

Oracle's (former Sun's) extension to Apache Shindig. Blog http://blogs.oracle.com/socialsite

L. TODO

- Does funambol.org has interesting implementations?

support Plain Text Format (text/plain), RFC5147 URI fragment identifier for plain text?

> Sowohl Atom als auch AtomPub definieren XML-Vokabulare, die eine Erweiterung mit zwei Mechanismen unterstützen. Zum einen ist im Standard definiert, dass neue Elemente in diesen Vokaularen selbst von standardkonformen Prozessoren ignoeriert werden müssen. [...] Gleichzeitig ist es überall dort, wo es nicht explizit verboten ist, möglich, Elemente ausanderen XML-Namespaces einzubetten.

[Til11, p. 102]

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

[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

[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

[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

[Kai11] Kaiser, Guido: *Modellierung und Vergleich von Implementierungen einer REST-Anwendung in verschiedenen Sprachen*, Fernuniversität Hagen, Master thesis, 1 2011. `http://www.fernuni-hagen.de/dvt/studium/masterarbeiten_9.shtml`

[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

[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

[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

[Til11] Tilkov, Stefan: *REST und HTTP: Einsatz der Architektur des Web für Integrationsszenarien*. 2. DPUNKT VERLAG, 2011

[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