Technische Universität Dresden
Fakultät Informatik
Institut für Systemarchitektur
Prof. Dr. rer. net. habil. Dr. h. c. Alexander Schill

Großer Beleg

XMPP-based Media Sharing for Mobile Collaboration with Android Phones

Benjamin Söllner
born November 2, 1986 in Plauen, Germany
benjamin.soellner@mail.inf.tu-dresden.de

October 29, 2009

Advisor: Dr.-Ing. Daniel Schuster
daniel.schuster@tu-dresden.de

Selbstständigkeitserklärung

Hiermit erkläre ich, die vorliegende Studienarbeit zum Thema

XMPP-based Media Sharing for Mobile Collaboration with Android Phones

selbstständig und ausschließlich unter Verwendung der im Quellenverzeichnis aufgeführten Literatur- und sonstigen Informationsquellen verfasst zu haben.

Dresden, den 30. Oktober 2009

Unterschrift

Acknowledgements

I would like to thank the staff members of the Mobilis team for their valuable guidance and helpful input concerning both the theoretical as well as the practical part of this work. This includes Maximilian Walther, Thomas Springer and in particular my thesis supervisor Daniel Schuster, who motivated me by showing great interest into my ideas and always provided valuable feedback, especially concerning the practice of academic writing which I was fairly new to.

Thanks go also to the my student collegues who worked on other aspects of the Mobilis project in parallel to or before this thesis: István Koren, Dirk Hering, Christopher Friedrich, Jacobo Eduardo Miranda and Lukas Vierhaus. I am thankful for the passionate atmosphere in which we shared knowledge, feedback and hints about each others area and I enjoyed the illuminating conversations and inspirations we had. I also thank my student collegues from student lab INF/3074 for the creative and social working environment.

I finally owe my family a big debt of gratitude. Ich danke euch vielmals für die allzeitliche Unterstützung meiner Studienpläne, speziell aber für Hilfe und Ruhe in den letzten hektischen Tagen, in denen ich diese Arbeit fertiggestellt habe.

Abstract

With the recent tremendous innovation on the mobile handset, network and operating system market, the use of mobile phones as content creation and content sharing device have become commonplace. Although bluetooth, MMS and email obviously seem outdated in the social cloud the "Web 2.0" creates, neither the academic nor the economic world has developed an open media sharing protocol committed to collaborative work and adapted to a mobile environment.

This thesis has been created within the Mobilis project as a subproject called "Mobilis Media". Mobilis is a project developing a service platform for collaborative work with Android phones. Mobilis chose XMPP, the Extensible Messaging and Presence Protocol, as a collaborative protocol. Besides this document, a client and a server prototype was developed which fits into the Mobilis architecture.

The thesis starts by motivating the need of a mobile media sharing platform and by introducing a user scenario of travel picture sharing. It then examines several file transfer technologies, both media sharing technologies used within XMPP itself (SI File Transfer and Jingle) and second-stack technologies (WebDAV, Atom Publishing Protocol, Google Wave Federation Protocol). Afterwards, related work is evaluated and requirements to the media sharing platform as well as the mobile client are settled. Moving on, a concrete file sharing technology is chosen and the XMPP interface of the media sharing platform using it is described as a custom extension to the XMPP protocol. Subsequently, the structure of the Mobilis platform is presented and how Mobilis Media fits into it. System boundaries and extension points for later projects are outlined. Some aspects of the internal structure and some implementational considerations of the Mobilis Media prototype are highlighted. Finally, the presented implementation is evaluated qualitatively and quantitatively and a prospect for future work is given.

Keywords mobile collaboration, services, mobile framework, media sharing, content sharing, metadata, repository, multidimensional, cube, hypercube, database, social networks, Hibernate, Android, XMPP, Jingle, SI File Transfer, Publishing Stream Initiation Requests, WebDAV, APP, Google Wave

Contents

1. Introduction

The capabilities of mobile networks and handset devices has dramatically in the last few years. Broadband mobile network technologies like UMTS, HSDPA or HSUPA allowing high speed data connections are finally available for reasonable prices to end users. Additionally, global players like Apple or Google joined the competition in the smart phone market recently with the Android Operating System or the iPhone, intensivating competition, supporting innovation and lowering prices for those ubiquitous gadgets. On the other hand, the so-called "Web 2.0" phenomena has achieved a social change in developing for and using the web. The user is not longer solely involved as the consumer of contents but rather of the creator.

Current development of mobile handset and network technologies suggests that the "Web 2.0" will reach the mobile phone for a bigger amount of consumers in the near future. That is mostly to the fact, that the mobile phone is permanent companion of the owner. That way, the mobile phone is used as a content creation tool based on the built-in camera. It's use is huge and still growing. Compared to still cameras, mobile cameras have been sold 4 times as much in 2008 and are estimated to reach a ratio of 7:1 in 2010 [art03]. The desire of the user to share this created content in social platforms can be percieved when observing the Web 2.0 phenomena.

The conventional technologies to share mobile phone images are bluetooth, MMS or email. Bluetooth relies on proximity and user-initiated device pairing and therefore is fairly inflexible for a big or spacially distributed group formed by a social network. MMS is to restricted in terms of file size, annotations and number of recieptients, and to cost intensive. Even email is not vivid enough concerning the dynamics of mobile networks - it does not allow any repository-like browsing or pull functionality.

Economy has percieved this need and social network providers offer the possibility of content submission in their APIs. Of course, the submission will happen to the specific social network only. Successing bluetooth and MMS, there is no current open standard for ubiquitous media sharing. Instant messaging protocols, like Skype, ICQ or XMPP define how to send files between single users and are partially already ported to mobile operating systems but there is no method to share content using a central repository with browsing, pulling and management functionality.

1.1. The Mobilis Project

The development of a mobile media sharing repository suggest integrating a media repository service into the architecture of the Mobilis platform. The Mobilis platform is a server-client system using XMPP, the open Extensible Messaging and Presence Protocol, as a communication protocol. The platform is developed within the Mobilis project, a brasilian-german research project of TU Dresden's chair of computer networks coorporating cooperating with the Pontifícia Universidade Católica do Rio de Janeiro and the

Universidade Federal de Minas Gerais in Belo Horizonte.

The goal of the Mobilis project is to develop a generalized and open framework for mobile collaboration with a set of central services offered to the mobile clients via the XMPP interface. In earlier work already a number of such services was developed: a mobile tourist guide [Kor08a] (MobilisGuide), a geolocation service [DS09] (MobilisBuddy) and a collaborative drawing application [Kor08b] (MobilisMapDraw) to mention a few. Along with the development of the respective services matching sample mobile client applications were created. These clients are usually developed on top of the Google Android Operating System and make use of the services based on a concrete user scenario.

The prototype developed within this thesis will uses the name **Mobilis Media** and consists of the named media sharing service a sample client allowing picture sharing based on a tourism scenario.

1.2. User Scenario: Travel Picture Sharing

> *Travel is only glamorous in retrospect.*
> – Paul Theroux, born 1941, US novelist, in The Washington Post

Reviewing and reminiscencing a past tourist trip is an activity highly enjoyed by travelers and tourists. Pictures, which are taken during the journey are exchanged with travel mates. They are shown to friends at home, shared with the world to give an impression for future travelers or simply kept in a private library for personal memories.

During a voyage, it is desirable to have a tool at hand which can accomplish the exchanging and sharing task automatically. Given the advancements of mobile networks and mobile phone cameras, the mobile phone camera might finally be used instead of still image camera during a journey. Images taken with the mobile phone camera are also tagged with a timestamp and are geotagged, if the mobile phone posesses a GPS sensor. Hence, management of the metadata of the images should be taken into account.

The goal of this thesis is to develop such a tool as a prototype additional to theoretical elaborations concerning a mobile media sharing repository. The tool should be able to store taken pictures into the repository adding a user defined title and taken into account metadata stored with the image. It should be possible to upload new pictures or replace current ones as well as delete them. On the other hand, it should be possible to browse existing pictures in an intuitive GUI. Remembering the geotagging feature, a map is a possible artifact where images could be visualized.

1.3. Structure of this Thesis

Starting with chapter 2, this work lays the foundations of this work by introducing the XMPP protocol and a set of possible protocols to allow file transfer or transfer of binary data between entities. In the following chapter 3 related work is introduced which copes with the task of mobile media repositories. Chapter 4 presents the requirements set to the media repository and the client prototype. The actual media system is then designed in chapter 5 and implementation specific issues are adressed in chapter 6. Chapter 7 evaluates the presented solution and chapter 8 finally concludes the accomplishments of this work and gives a prospect for future work.

2. Foundations

Mobile media sharing can be split up into two general problems: file transfer, content repository management and mobile collaboration. Earlier work in the Mobilis Project (see section 1.1) has shown, that XMPP, a protocol for messaging and presence, suits exceptionally well for the purpose of mobile collaboration, enabling cloud computing with both the users and the services being equally participating entities of a XMPP session. In this chapter, we will provide a brief introduction into XMPP in section 2.1.

Moreover, the basic task of this work is to find a proper file transfer and content repository management technologies. That is why in section 2.2 we will present how the XMPP comunity developed extensions to XMPP and which extensions would suit well for our purpose of media sharing. We will contrast two general approaches: SI File Transfer with Published Stream Initiation Request (2.2.1) and on the other side an XMPP signalling protocol called "Jingle" (2.2.2). In section 2.3, we will introduce two other technologies, Atom and WebDAV, which are commonly used for media sharing. Finally, section 2.4 will compare introduced technologies side-by-side, especially examine library support facilitating future development.

2.1. The XMPP Protocol

The Extensible Messenging and Presence Protocol (XMPP) is an implementation of the Internet Engineering Task Force (IETF) model for near real-time instant messaging and presence (e.g., buddy lists). Beyond, XMPP recently evolved into a protocol used in the realm of message oriented middleware [Joh05]. Thus, it can be used in much broader domains, e.g. shared editing or collaborative drawing - and file transfer. XMPP is an open standard and many free and open source client and server implementations exist. The origins of XMPP lie in the Jabber project, which was formed in 1998. IETF formed an XMPP Working Group in 2002 and produced four specifications, which were approved by the IESG as Proposed Standards in 2004. RFC 3920 [SA04a] and RFC 3921 [SA04b] are currently undergoing revisions to promote them to a Draft Standard.

One of the key strengths of XMPP are built-in security mechanisms. Since XMPP is an open standard, everybody can run their own XMPP server. XMPP servers can be isolated from the public, so they can also be installed inside of a company network. The XMPP core furthermore specifies robust security mechanisms like SASL or TSL to encrypt the transport stream. The XMPP Standards Foundation also runs an own certification authority at xmpp.net to encourage the use of channel encryption. Moreover, certain security threats are defeated by automated identity check of connected users using a dialback protocol.

Additionally, no multi-hop routing is possible. In email, if a server A needs to deliver a message to server B the message might be routed via a number of itermediate servers C_i. In XMPP, server A would perform an appropriate DNS lookup and then open a direct

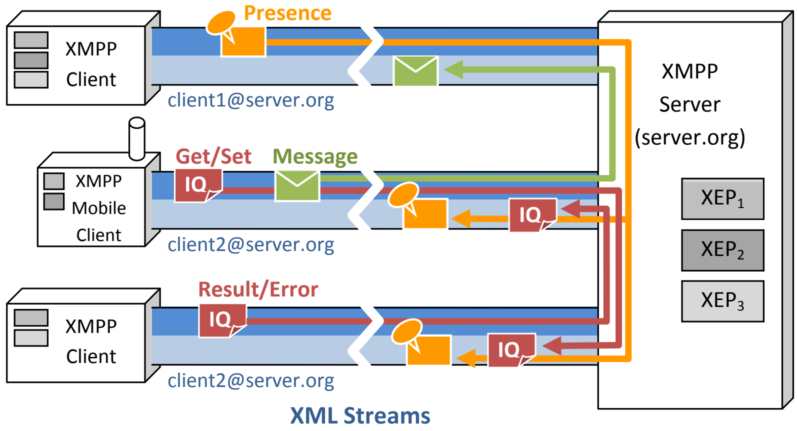

Figure 2.1.: A basic XMPP architecture scenario

connection to the server B. This prevents changes of the messages along the way, so that addresses cannot be modified. On the other hand, it means that XMPP servers have to maintain "always-on" connections to the network, i.e. more reliable uptime than email servers.

XMPP users communicate via an XMPP-address similar to an email-address. The XMPP-address is formed out of an username and the XMPP server's domain, both separated by an "at" sign ("@"). Multiple server logins are allowed by definition. The distinction between multiple clients connected to the same XMPP-address is made by appending a ressource identifier immediately following the address after a slash ("/").

After connection establishment and possible encryption and authorization initialization, information is exchanged on top of the TCP/IP protocol as two XML streams, one transmitting client-to-server and one server-to-client. A transmission unit exchanged from client to server or vice versa is called stanza: each stanza is a well-formed piece of XML, carrying at least the information about sender, receiver and identifier. Further on, stanzas are devided into three classes: Message, Presence and Info/Query. The meaning of every stanza type is illustrated in figure 2.1 showing a scenario with one server three clients by bidirectional XML streams. We will detail each stanza type in the following sections.

2.1.1. Message

A Message can be seen as pushed data from one entity of the XMPP network to another. In Instant Messanging scenarios, this usually corresponds to a chat message sent to another user with a body part similar to an email. But messages can be used also in other scenarios, for example when notifying XMPP entities about an event, for which the entity has subscribed.

A basic message stanza would be looking like this:

```
<message from='juliet@shakespeare.lit/balcony'
    to='romeo@shakespeare.lit/yard'
    id='m940AE74'>
  O romeo...
```

```
</message>
```

Notice, how a random unique number is assigned to the message and the sender and recieptient of the message are defined as attributes of the `<message/>` tag. The actual payload of the `<message/>` tag is not restricted to plain text but can rather be any well-formed XML content – possibly qualified by an `xmlns` attribute. For example, in case of event notification in the publish/subscribe extension [MSAM08], the message body would consist of an `<event/>` tag containing the event information.

2.1.2. Presence

Presence Information can generally be regarded as multicast information of the user to the XMPP network. In particular, multicast would mean delivery to all addresses which have subscribed to a user's presence updates by having the specific user on their buddy list, in XMPP terms called "roster". The roster – and also the delivery of presence multicasts – is managed by the server. In instant messaging scenarios, a Presence stanza would contain all information about the users current context, like availability (online/offline/away/...), status message, location or any other relevant data.

The `<presence/>` tag contains a `type` attribute describing the overal presence state of the sender, which is mentioned in the `from` attribute. A corresponding example of a presence tag is:

```
<presence from='juliet@shakespeare.lit/balcony' type='
   unavailable'/>
```

2.1.3. Info/Query

Info/Query (IQ) stanzas are a mean to realize a request-response mechanism between any two XMPP entities. IQ stanzas can be on either of the type get, set, result or error. Therefore, each get- or set-IQ (request) has to be answered by a result- or error-IQ (response). The IQ stanza contains usually custom XML as child element describing the information which is to be obtained (get) or changed (set) and the answer in a positive (result) or negative case (error).

A well-known application of IQ-stanzas is the service discovery used to identify the capabilities offered by an XMPP entity, that is, a list of feature sets explaining which particular stanzas a XMPP entity can understand. [HMESA08] An XMPP entity would request information from another entity about which XMPP extension this entity supports. This is done using a get-IQ, containing a `<query/>` tag qualified by an appropriate `xmlns` attribute showing that the query is about service discovery:

```
<iq from='romeo@shakespeare.lit/yard'
    to='juliet@shakespeare.lit/balcony'
    id='m940AE76' type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

The receiver would normally reply sending back a corresponding result-IQ (with the same `id` attribute):

```
<iq from='juliet@shakespeare.lit/balcony'
    to='romeo@shakespeare.lit/yard'
    id='m940AE76' type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <feature var='urn:xmpp:jingle:1'/>
    <feature var='urn:xmpp:jingle:apps:rtp:1'/>
    <feature var='urn:xmpp:jingle:apps:rtp:audio'/>
    <feature var='urn:xmpp:jingle:apps:rtp:video'/>
  </query>
</iq>
```

The answer contains a list of “features” the entity supports. Features are qualified by the identifier of an XML namespace mentioned in the `var` attribute. This namespace concept is inherent to all XMPP extensions. Every XMPP extension belongs to a namespace. If the entity supports an extension, it should mention its namespace upon service discovery. There is a number of extensions standardized by the XMPP community and we will introduce some of them in section 2.2. However, defining your own proprietary namespaces is also possible.

When using an extension, i.e. sending a stanza which is characteristic to the extension, the extension-specific tags are qualified by the respective namespace using an `xmlns` attribute. This is a well-known concept in every extension. In chapter 2.2 we will oftenly mention “qualified elements”, which means elements (tags) with a `xmlns="..."` according to the namespace of the Extension. In our example above, Juliet supports the Jingle protocol (which we will describe in section 2.2.2). Here is a Jingle-related IQ – notice, how the Jingle payload is qualified with the correct namespace (“`urn:xmpp:jingle:1`”):

```
<iq from='juliet@shakespeare.lit/balcony'
    to='romeo@shakespeare.lit/yard'
    id='m940AE77' type='set'>
  <jingle xmlns='urn:xmpp:jingle:1'
          action='session-initiate'
          initiator='juliet@shakespeare.lit/balcony'
          sid='a7840fe940ece580'>
    <!-- ... -->
  </jingle>
</iq>
```

2.2. The XMPP Extension Protocols

On top of the above mentioned building blocks, the XMPP community has developed further specifications to standardize XMPP communication in various areas like file sharing, collaborative drawing, event publication/subscription, avatars and much more. Each standard is published as a so-called XMPP Extension Protocol (XEP) which runs through a commonly agreed standardization process.

Below, a few XEPs related to media exchange and sharing are introduced. We concentrate on two broad concepts: First, in subsection 2.2.1, we step-by-step introduce a technology called SI File Transfers with Published Stream Initiation Requests. Second,

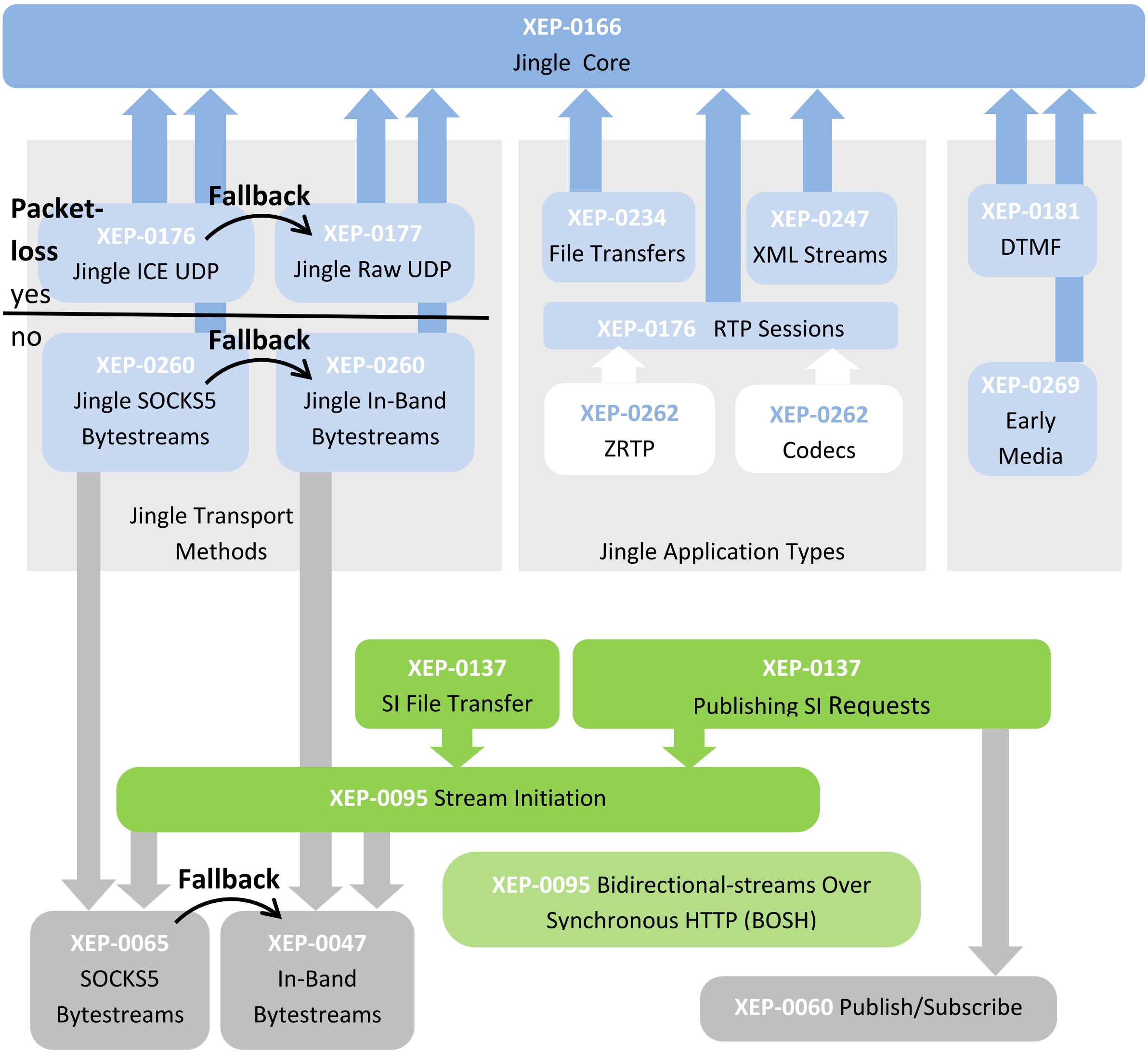

Figure 2.2.: Selected XEPs for media transport and their interdepecincies.

in subsection 2.2.2ff., we will have a look at a more sophisticated and advanced stream initiation protocol “Jingle” and present it as an alternative to the approach mentioned before. Figure 2.2 shows both set of XEPs, the SI File Transfer XEPs sitting on top of XEP-0095 “Stream Initiation” and the Jingle XEPs, building on their core XEP-0166 “Jingle Core”.

2.2.1. SI File Transfers with Published Stream Initiation Requests

The need to develop an extension to exchange binary data was percieved early by the XMPP community. Already 2002, it was possible to exchange URLs of external resources using the **Out Of Band Data** extension. Later, XEPs were published, which allowed controlled streams of binary data: first as **In-Band Bytestreams**, later using out-of-band **SOCKS5 Bytestreams**. Around those streaming methods, a standard called **Stream Initiation** was developed for initiating a bytestream independent from streaming method and application. Different applications, like **SI File Transfer**, were classified in profiles and standardized. Now, push-like file transfer was possible with XMPP. Another extension, called **Publishing Stream Initiation Requests**, made pull-like transfers possible by allowing information about a stream to be published to interrested subscribers or to a repository. In this subsection, we will step-by-step introduce every of the mentioned extensions and their functionality.

XEP-0066: Out Of Band Data

The first XMPP extension protocol specified for binary data exchange was XEP-0066 [SA06]. It provides a mean to inform XMPP entities about available ressources under a certain URL. It is the receiving entities responsibility to retrieve the data from this URL, if desired. This method can still be used in file transfer scenarios as a fallback solution, if more sophisticated technologies fail.

The URI is communicated by a Set-IQ with a qualified `<query/>` element which then contains a `<url/>` and optionally a `<desc/>` element which hold the URI and the metadata of the shared resource respectively. The receiver of this Set-IQ will retrieve the ressouce and send an empty Result-IQ, if it succeeds.

The XEP specifies more advanced technologies, where this combination of `<url/>` and `<desc/>` may be used: for example, it can be included inside data forms [EHM+07]. Data forms are a mechanism similar to web forms, which are exchanged between and filled out by XMPP entities to communicate arbitrary record-like information between XMPP entities. With the combination of Out Of Band Data and Data Forms it is possible to exchange complete files inside of Data Forms, similar to the “Choose file...” field of web forms. Furthermore the XEP provides a mean to include the out-of-band transport inside of Stream Initiation Requests [MME04b], which will be detailed later in this section.

XEP-0047: In-Band Bytestreams

In-Band bytestreams [KSA09] send binary data directly along the XMPP channel. Since the XMPP channel is pure XML, the stream has thus to be base64-encoded. This results in a large overhead and high server load. For those reasons, that technology is - especially in the mobile sector - only a fallback option but may still be used in other low-bandwidth applications like games, shell-sessions or encrypted messaging.

An in-band bytestream is initiated by a Set-IQ with an qualified `<open/>` element. The `<open/>` tag provides all necessary information, like `block-size`, a `sid` which is unique for this transfer and the attribute `stanza`, which describes, if IQs (`iq`) or messages (`message`) should be used to exchange the actual data. IQs should be preferred, since they are always acknowledged by Result-IQs. The receiving entity confirms the opening of the stream with an empty Result-IQ.

After the stream is opened, it may be used bidirectionally. Chunks of data are exchanged by the two participating entities inside of Set-IQs with `<data/>` elements containing the base64-encoded data. Each `<data/>` element is tagged with a `seq` and `sid` attribute which identify the data chunck's stream and its position inside of the stream. Finally, the stream may be closed by either party with a `<close sid="..." />` element.

XEP-0065: SOCKS5 Bytestreams

SOCKS5 Bytestrams `xep:0065` are a mean to initiate a out-of band binary connection between two XMPP entities. It is NAT-safe (Network Address Translation) since it uses the SOCKS5 protocol which makes it possible to mediate the connection through so-called StreamHosts, which are nothing else than SOCKS5 proxies enhanced by XMPP functionality.

The SOCKS5 protocol `rfc:1928` is an internet proxy protocol between application and transport layer. It makes it possible for any client-server application to use the services of a so-called SOCKS proxy server transparently and independently from the actual underlying protocol. That means, if both clients are behind a NAT, they can connect to a SOCKS proxy and exchanged data is transparently forwarded to the opposite entity. A SOCKS connection is established by the client sending first its supported authentication methods to the server. Then, the server chooses one authentication method and replies. The client authenticates and then issues a connection request containing, among others, the destination address, port and used protocol (TCP open, TCP accept or UDP). The server finally confirms the connection request.

The initiation of a SOCKS5 Bytestream via XMPP is as follows (see figure ...):

1. The initiator queries the target for SOCKS5 bytestream support.
2. The initiator tires to find a StramHost using service discovery by querying its XMPP server for a list of potential StreamHosts and then querying each potential StreamHost to find out, if it really is a StreamHost.
3. The initiator requests the full network address from the StreamHost using a Get-IQ with a qualified `<query/>`.
4. The StreamHost replies with an respective Result-IQ whose `<query/>` contains a `sid` attribute used as an unique identifier from now on and furthermore a `<streamhost/>` child elements with information about `host` and `jid`.
5. The initiator informs the target about all gathered streamhosts by sending a Set-IQ with a list of `<streamhost/>` elements inside an appropriate qualified `<query/>` element.

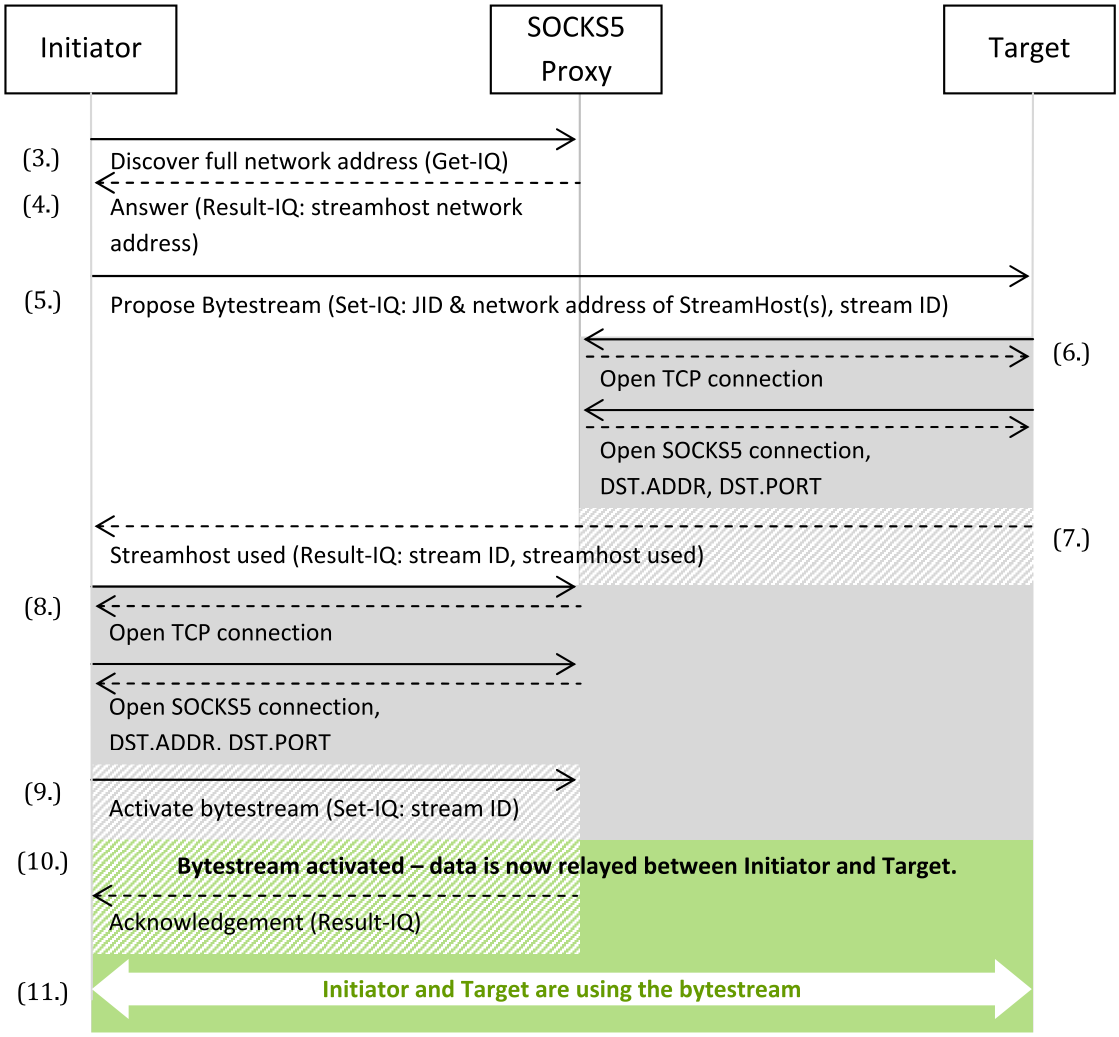

Figure 2.3.: SOCKS5 Bytestreams negotiation

6. The target tries to connect to one of the StreamHosts by opening a SOCKS5 TCP connection. The reported SOCKS5 destination address (DST.ADDR) is the SHA1-encrypted concatenation of `sid`, initiator JID and target JID. The destination port (DST.PORT) is set to 0.

7. The target notifies the initiator about the used StreamHost by sending a Result-IQ containing an appropriate qualified `<query/>` element with a `<streamhost-used/>` subelement.

8. The initiator opens a SOCKS5 TCP connection to the StreamHost. The destination address (DST.ADDR) and port (DST.PORT) are set like in (6.). That way, the server can be sure that both the initiator and the sender are willing to accept the connection.

9. The initiator opens the bytestream by sending a Set-IQ with an appropriate qualified `<query/>` tag containing an `<activate/>` tag.

10. The StreamHost activates the bytestream and replies with a Result-IQ.

11. The media may now be exchanged over the TCP connection via the SOCKS5 StreamHost.

There are situations, where the use of a StreamHost is not necessary, that is, when NAT is not applied and a direct TCP connection between the both XMPP entities can be established. In this case, the protocol flow is simplified: Only steps (5.) and (7.) are executed where the initiator itself is advertised as the StreamHost. Afterwards, the initiator will activate the bytestream and the media may be be exchanged over the TCP connection.

XEP-0095: Stream Initiation

The methods presented inside this section up to now are rather different streaming methods where different forms of streamed communication may be executed on top. There has been little told about stream negotiation and metadata exchange. This is the scope of XEP-0095 about Stream Initiation: it negotiates an out-of-band content stream between any two XMPP entities, i.e. choses the streaming method, provides sufficient metadata in advance and may be used for file transfers, audio/video chat and other applications.

Streams are initialized by a Set-IQ from the initiator containing a qualified `<si/>` element. The content of the `<si/>` element has two parts. Firstly, it contains a so-called profile, which describes the use case of the bytestream and its metadata. This may, for example, be a qualified `<file/>` element in case of file transfer. Secondly, it contains a qualified `<feature/>` element by `http://jabber.org/protocol/feature-neg`. This element posesses a `<x/>` tag carrying a data form [EHM+07] which is used to negotiate the file streaming parameters, i.e. the streaming method used. The data form may offer several possible methods from which the responder will choose one. One example of a complete Set-IQ might look like (taken from [MME04b]):

```
<iq type='set' id='offer1' to='receiver@jabber.org/resource'>
 <si xmlns='http://jabber.org/protocol/si' id='a0'
```

```
    mime-type='text/plain'
    profile='http://jabber.org/protocol/si/profile/file-
       transfer'>
  <file xmlns='http://jabber.org/protocol/si/profile/file-
     transfer'
        name='test.txt' size='1022'>
   <desc>This is info about the file.</desc>
  </file>
  <feature xmlns='http://jabber.org/protocol/feature-neg'>
   <x xmlns='jabber:x:data' type='form'>
    <field var='stream-method' type='list-single'>
     <option><value>
      http://jabber.org/protocol/bytestreams</value></option>
     <option><value>
      jabber:iq:oob</value></option>
     <option><value>
      http://jabber.org/protocol/ibb</value></option>
    </field>
   </x>
  </feature>
 </si>
</iq>
```

Here, the responder has the choice between SOCKS5-bytestreams, Out-Of Band Data and In-Band Bytestreams. In the case of success, the receiver will respond by with Result-IQ having a similar `<si/>` element with a filled-out data form. The stream will then be opened according to the chosen streaming method.

XEP-0096: SI File Transfer

The SI File Transfer XMPP extension protocol [MME04a] finally adds metadata to a file transfer and provides, together with the streaming methods and negotiation technologies presented before, the possibility to carry out seamless file transfers, enhanced with metadata, reliable, even via NATs and optionally even featuring ranged transfers.

XEP-0096 defines a profile to be used with Stream Initiation Requests, which announces a file transfer: An appropriately qualified `<file/>` element may be specified inside a stream initiation request (`<si/>`). This element contains attributes with metadata of the file (`size`, `name`, `date`, `hash`) and optionally a `<desc/>` element to provide a human-readable description plus an empty `<range/>` element to indicate support for ranged transfers which makes it possible for the receiver to specify `offset` and `length` when requesting a portion of the file.

XEP-0137: Publishing Stream Initiation Request

XEP-0137 [MM05] introduces a pull model for streams by bringing the XMPP extension “Publish-Subscribe” [MSAM08] and Stream Initiation Requests together. Publish-Subscribe is a generic XMPP extension to allow entities to publish items to a PubSub service. The item is published to a node, which covers an area of interest. Other entities may subscribe to this node and receive events when new items are published. An entity may also create new nodes or subnodes or delete them. In fact, the PubSub-service hosts a whole node-tree. Entities may query the service for its nodes or for the published items. Owners of a node – i.e. entities which created that node – may query or manage subscriptions to their node or modify the items inside of the node. Subscriptions and nodes may be configured using Data Forms [EHM+07] to specify the behaviour of the PubSub service regarding the notification or access control.

By combining both PubSub and Stream Initiation Request, one gets a powerful model to publish links to binary ressources in a hierarchic content repository. Note, that only Stream Initiation Requests are published, the data itself is located elsewhere. It is retrieved from the publisher, when the stream is initiated based on the published Stream Initiation Request stored in the PubSub-tree and retrieved from subscribers or from entities which browse that tree.

A generic publish-IQ is a Set-IQ with a qualified `<pubsub/>` element containing a `<publish/>` element. In the case of Stream Initiation Requests, `<publish/>` posesses a `<sipub/>` element, which is similar to `<si/>` but does not contain any feature negotiation. The `<sipub/>` element is then pushed to all subscribers in accordance to "Publish-Subscribe" using XMPP-Messages containing an `<event/>` with a list of `<items/>` and the `<sipub/>` inside of a final `<item/>`. Note, that this is not the real stream initiation request yet but only a notification, that there is such a source available at the respective entity.

After the subscriber has recieved the `<sipub/>` notification, it may request the actual Stream Instantiation from the initiator. This is done by sending a Get-IQ with `<start sid="..." />` to the initiator. The initiator will answer with a Result-IQ containing a similar `<starting/>` element and will then issue the actual Stream Initiation Request (`<si/>`) in accordance to "Stream Initiation" described earlier [MME04a].

2.2.2. Jingle – An XMPP Signalling Protocol

Jingle is a signaling protocol first introduced to allow simple video and voice chat. It is specified in XEP-0166 [LBSA+09]. It resembles the Session Initiation Protocol (SIP) [RSC+02] and was introduced after a long discussion inside the XMPP community which showed that two-stack clients working on both SIP and XMPP are difficult to realize. However, interworking of Jingle entities with SIP entities is made possible since many Jingle-related XEPs define direct mapping between SIP and Jingle so that translating gateways can be developed easily.

Jingle was designed with a maximum of flexibility concerning application types (i.e., what data is transmitted - voice/video chat, file transfer etc.), transport methods (i.e., how the data is transmitted - via UDP, TCP, Socks 5 etc.) and security preconditions. The XMPP core leaves the specification of those technologies to further XEPs while specifying only a common template to which all the related specifications have to comply. We will have a look onto application types in subsection 2.2.4 and onto transport methods in subsection 2.2.3. Further design goals of Jingle was strict separation of signalling from data, lightweighted clients and the support of session management functions.

A basic Jingle stanza is usually an IQ containing a qualified `<jingle/>` tag with attributes such as `action`, `initiator`, `responder` or `sid` (session-ID). `action` takes a spe-

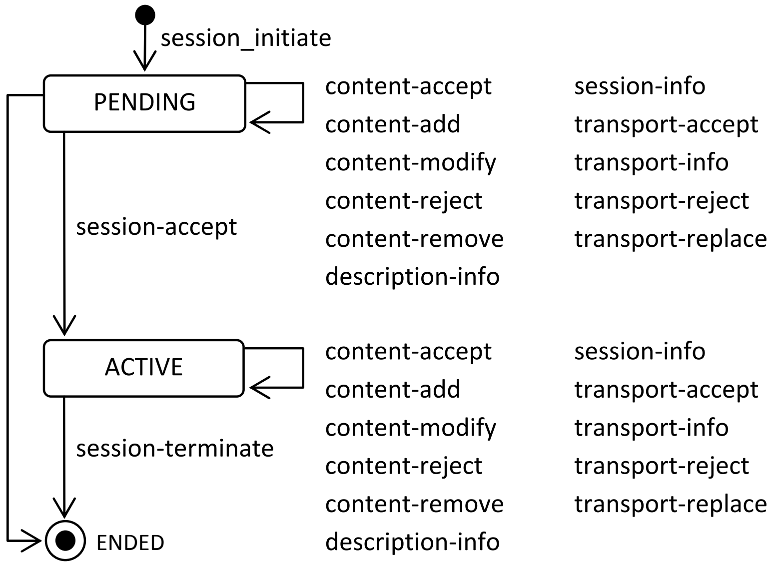

Figure 2.4.: The Jingle protocol states.

cial role, since it defines how one XMPP user wishes to modify the session with their counterpart.

The `<jingle/>` element itself usually contains one or more `<content/>` elements - one for each stream to be established. The `<content/>` element then consists of one `<description/>` and one `<transport/>` element, specifying the application data (the "what") and the transport method (the "how"). Application data is further described by a `<payload-type/>` subelement and a transport method by a `<candidate/>` subelement.

A basic Jingle session passes through the 3 states "pending", "active" and "ended" (see figure 2.4.

1. After resource determination, the initiator sends an Set-IQ initialization request to the responder (`action="session-initiate"`). The responder has to answer with an Result-IQ or an Error-IQ. In the case of succes, the session is in the transits to the state "pending".

2. Now, further negotiation can be performed, like adding/removing/editing transports or contents (`action="transport-info"`, `transport-replace`, `content-modify`, `content-add`, `content-replace`).

3. Then, the session will be accepted by the responder (`session-accept` IQ-set), which will be acknowledged by the initiator (IQ-result). The sender will choose a subset of listed `<payload-type/>`s and `<transport/>`s which she supports herself. The session is now in the state of being "active".

4. Even after, further modifications to transports and contents can be carried out.

5. Finally, one party will end the session (`session-terminate` IQ-set with `<reason/>` element). Also this will be confirmed with a IQ-result.

During the whole session process, informational messages with `action="session-info"` can be sent, e.g. the IQs containing a `<ringing/>` tag are used to signalize that the session initiation was received but not accepted yet.

2.2.3. Jingle Transport Method Specifications

XEP-0176: Jingle Raw UDP Transport Method

XEP-0176 defines a transport method for establishing and managing out-of-band sessions using raw-UDP between two entities defined by their IP address and port. This transport method is applicable where some packet loss is tolerable, e.g. in audio/video chats. Raw-UDP works as a "hit-or-miss" protocol: the transfer might work end-to-end, especially when the sending entity is a gateway / relay, e.g. when the back-to-back user agent sends an early media offer to the initiator on behalf of the responder.

XEP-0177: Jingle ICE-UDP Transport Method

XEP-0177 defines a transport method for establishment and management of out-of-band sessions using ICE-UDP. ICE-UDP stands for Interactive Connectivity Establishment and is suitable for the use of media applications communicating over Network Address Translators (NATs), where some packet loss is tolerable. ICE-core is currently available as a RFC draft [Ros07]. However, this RFC focusses on sessions negotiated via SIP, while XEP-0177 makes the use of ICE-UDP possible with XMPPs signalling protocol Jingle. XEP-0177 specifies furthermore a mapping between SIP and Jingle in this particular case.

XEP-0260: Jingle SOCKS5 Bytestreams Transport Method

The Jingle SOCKS5 Bytestream Transport Method (XEP-0260 [SAM09]) brings together the SOCKS5 Bytestream Protocol defined in XEP-0065 [SMSA07] (see section 2.2.1) and the Jingle protocol defined in XEP-0166 [LBSA+09]. With this extension, it is possible to instantiate Jingle Sessions in accordance to the Jingle protocol with data flowing over SOCKS5 Bytestreams.

XEP-0261: Jingle In-Band Bytestreams Transport

The Jingle In-Band Bytestream Transport Method (XEP-0261 [SA09c]) combines In-Band Bytestreams defined in XEP-0047 [KSA09] (see section 2.2.1 and the Jingle protocol defined in XEP-0166 [LBSA+09]. With this extension, it is possible to instantiate Jingle Sessions in accordance to the Jingle protocol with data flowing directly over the XMPP channel itself, that is, not over any signalling channel. Because of that, this transport method should be only a failsafe solution. The In-Band Bytestream Transport Method is a lossless transport method.

2.2.4. Jingle Application Format Specifications

XEP-0167: Jingle RTP Sessions

XEP-0167 [LSAE+06] describes an application format for negotiating Jingle RTP media sessions and complies to the standard template for Jingle application formats defined alongside with the Jingle core specification in XEP-0166 [LBSA+09]. Special attention has been paid to the coverage of all possible RTP-parameters and their mapping to SDP. Also, all necessary informational messages (e.g. ringing, on hold, mute) are defined.

The Jingle RTP application format is usually used with datagram transports (raw-UDP as in XEP-0177 or ICE-UDP as in XEP-0176, see subsection 2.2.3) if the media is light and the latency low – this may, e.g., apply to streamed media. Usually, the transported content consists of two components: an RTP channel (1) and an RTCP channel (2).

XEP-0262: Use of ZRTP in Jingle RTP Sessions

ZRTP [Ros07] is a variant of RTP supporting secure RTP transmission. It can be used as an alternative to the Secured Real Time Protocol (SRTP). Negotiating ZRTP happens rather on the signal level base. However, in the SDP protocol a `zrtp-hash` attribute is required with ZRTP which communicates version and Hello Message.

XEP-0262 describes how this SDP attribute translates to Jingle. In Jingle, a `session-info` action would be sent after session initiation, containing a `<zrtp-hash/>` element.

XEP-0266: Codecs for Jingle RTP Sessions

XEP-0266 [SA09a] is strictly informational and provides suggestions about which codecs a Jingle entity should support. Since codecs are often subject to patents, the discussion about this topic has been very controversial in the XMPP community. In XEP-0266, some audio and video codecs are discussed according to criterias like quality, RTP packetization standard, cross-platform availability and patents. The audio codecs mentioned are Speex and G.711, the video codecs are Theora, Dirac and H.264.

The extension protocol suggests, that support for patent-clear, freely implementable and commonly deployed codecs should be supported. For audio, this would apply to both Speex and G.711. For video, no recommendation can be made yet, but Theora and Dirac are seen to have the most chances for the future, when they are deployed to more platforms.

2.2.5. File transfers and XML Streams using Jingle

XEP-0234: Jingle File Transfer

XEP-0234 [SA09b] shall improve SI File Transfer defined in XEP-0096 [MME04a] (see section 2.2.1, "File Transfer using Stream Initiation Requests"). The XMPP community identifies two drawbacks in that early standard: first, it supports no negotiation of File Transfer parameters but only acceptance or denial. Second, it is the only technology which uses Stream Initiation Requests – instead, one could use Jingle, which is much more powerful.

The extension protocol defines how the Jingle Protocol defined in XEP-0166 [LBSA+09] and the file description format in "SI File Transfer" (XEP-0096 [MME04a]) work together and describes a clear update path how to move from SI File Transfer to Jingle File Transfer.

File transfer is usually accomplished using the SOCKS5 or the In-Band transport methods of Jingle (see section 2.2.3), since loss of data is not tolerated. The XMPP community announces the development of another transport method, ICE-TCP, to provide more effective TCP transport over NAT.

A file transfer is initiated like every other Jingle session by sending a set-IQ with a `<jingle/>` tag with `action="session-initiate"`. The `<description/>` subtag will contain a `<offer/>` or `<request/>` element, depending on if the file is pushed or pulled from the initiator. This element then contains a qualified `<file/>` element. The `<file/>` element describes the file according to the structure defined in XEP-0096 [MME04a].

2.2.6. Further XEPs concerning Jingle

XEP-0181: Jingle DTMF

XEP-0181 [PSA08] defines an extension for XMPP to send Dual Tone Multi-Frequency (DTMF) events for dialing and issuing commands, e.g. of interactive voice response applications. Normally native RTP methods (like "audio/telephone event" or "audio/tone" media type) should be preferred, but when communicating with RTP-unaware entities, e.g. gateways to the PSTN, this protocol may be used. A DTMF event is signalled by sending a `<jingle/>` set-IQ with `action="session-info"`, which contains a tag of the following structure:

```
<dtmf xmlns='urn:xmpp:jingle:dtmf:0' code='0-9,#,*,A-D'
      duration='milliseconds' volume='0-63'/>
```

XEP-0269: Jingle Early Media

Jingle Early Media, defined in XEP-0269 [CSA09], defines a mean of exchanging media before the session is definitively accepted. It is comparable with the SIP header `Early-session` and accomplished using a `content-add` Jingle action. The media may be generated by the initiator or an intermediary. If an intermediary generates the early media, it has to use a codec and a transport method advertised by the initiator. This protocol may be used when dealing with ringtones or announcements using audio streams or Dual Tone Multi Frequency events (DTMF).

2.3. Second-Stack Technologies

This section will concisely present two other technologies – WebDAV (2.3.1) and the Atom Publishing Protocol (2.3.2). We name these protocols "Second-Stack Technologies" since they are independent from XMPP, which is used in the Mobilis Project. The use of those second-stack technologies would definitively require either wrapping or changing of the protocol fragments to fit into the XMPP stanza concept or it would require to introduce a second protocol stack and negotiation between XMPP and the respective protocol. We will also shortly motivate, how this can be done.

2.3.1. WebDAV

The WebDAV protocol (Web based Distributed Authoring and Versioning Protocol) is an extension to the Hypertext Transfer Protocol 1.1 (HTTP) which removes certain restrictions of the classical HTTP protocol. Originally, HTTP only allows Get and Post Request, i.e. downloading and uploading of information. WebDAV is developed by the WebDAV Working Group, the DASL Working Group and the Delta-V Working Group of the Internet Engineering Task Force. It is specified by numerous RFCs, mainly RFC 4918 [Dus07].

WebDAV adds support for much more operations like deletion, directory creation or modification or even versioning. To be precise, the following methods are added: `PROPFIND` (to access the metadata of a ressource), `PROPPATCH` (to modify the metadata of a ressource), `MKCOL` (to create a directory – called "Collection" in WebDAV), `COPY` (to copy a ressource), `MOVE` (to move a ressource), `LOCK` (to disallow modifications of a ressource temporarily), `UNLOCK` (to remove a lock).

More complex technologies, like the version controlling system Subversion [Ste02] are based on WebDAV. There is also a WebDAV extension called CalDAV [web08] which makes it possible to manage a calendar using the WebDAV protocol – this extension is specified in RFC 4791 [DDD07] and used e.g. by Google Calendar [Goob].

On the one side WebDAV seams like a promising technology: It is wide-spread – supported in many desktop operating systems like Windows XP and Linux – and first applications have been built which bring WebDAV to mobile phones (see section 3.2 and 3.5). WebDAV can handle the transport of binary data like HTTP can do – without the need of base64-encoding. When integrating WebDAV into an XMPP protocol stack, one can use a dedicated repository server to store the media ressources and use e.g. Out-Of-Band Data (see section 2.2.1) to communicate the URLs of the ressource. A very similar approach to managage binary ressources inside of collaborative documents has been taken by Google with their product Google Wave – we will present this solution in more detail in section 3.6.

However a WebDAV repository has the drawback. It uses a directory structure as the underlying ressource classification system. This concept is too restrictive. What we are looking for is an architecture, which distributes ressources according to it's metadata – not according to a physical location like a webserver path. Also, there are no open frameworks for WebDAV available except Jakarta SLIDE [1], which is discontinued.

2.3.2. Atom Publishing Protocol

The Atom Syndication Format (ASF) is a standard for platform independent information exchange and an alternative to RSS (Really Simple Syndication) [webe]. On top of that, the Atom Publishing Protocol (AtomPub or APP) is a platform independent XML format for editing web ressources. It was originally developed to provide a mean for webfeed administrators to publish feed items to an Atom feed in a standardized way. Today, it is standardized by the IETF in RFC 5023 [Gdh07].

The Atom Publishing Protocol runs on top of the HTTP protocol. A client, who is interested in publishing an entry to a collection first retrieves a so-called service document via HTTP-`GET` from a dedicated URL. This service document is from the content type `application/atomserv+xml` and contains one single `<service/>` element with one or more `<workspace/>` element containing `<collection href="..." />`" elements. The

[1] `http://jakarta.apache.org/slide/`

client will then issue a `POST` request to the address specified in `href`. Also requests like `DELETE` are possible.

The idea of collections makes the Atom protocol more attractive for the use with metadata-driven content repositories, since the APP server may decide itself, where it should put the uploaded content and in which context it should offer the ressource to other participants. However, certain other drawbacks limit the use of APP in our situation: firstly, APP was developed for XML data where Media sharing, especially image sharing handles pure binary data. Secondly, again, APP is a protocol which needs a separate web server and thus a second protocol stack. And thirdly, library support in this area is not mature yet and there have been no efforts to bring APP to a mobile platform. Two libraries on java basis have been published to our knowledge: ROME propono, currently available in version 0.6 [2] and Apache Abdera, currently available in version 0.4 [3].

2.4. Conclusion

In this chapter, we introduced four possible technologies to allow media sharing in collaborative environments: SI File Transfers with Published Stream Initiation Requests (2.2.1) and Jingle File Transfers (2.2.5) as a solution integrated into XMPP as well as WebDAV (2.3.1) and the Atom Publishing Protocol (2.3.2) as second-stack technologies. Table 2.1 summarizes the insights made in this chapter according to support of metadata-driven repositories, a suiting transport method for media sharing purposes and library support for mobile applications.

	Metadata-driven repositories	**Suiting Transport Method**	**Library support**
SI FT	Yes	SOCKS5	Smack
Jingle FT	Yes	SOCKS5, later: ICE-TCP	Not available
WebDAV	No	HTTP	Apache Slide (discontinued)
APP	Yes	HTTP	Rome Propono, Apache Abdera (unsure)

Table 2.1.: Overview of presented technologies

[2] `http://wiki.java.net/bin/view/Javawsxml/RomePropono`
[3] `http://abdera.apache.org/`

3. Related Work

Media sharing, or more general, file sharing on mobile devices is a widely discussed topic in the academic as well as in the economic sector. However, there are only a few elaborations who combine media sharing techniques with social collaboration mechanisms provided by mobile systems and, in particular, XMPP.

In the following section, we want to present a selection of promissing solutions. We will start by discussing academic papers. In section 3.1, we will introduce the work of Belimpasakis et al. [BLB07], who developed an independent content sharing middleware which can connect to a given list of content repositories. Then, in section 3.2, we will present a paper from Tolvanen et al. [TSLA06] who integrated remote file systems into the mobile device. Section 3.3 shows the work of Martuszewski et al. [MBLH06], who implemented a distributed file sharing service using on the Session Initiation Protocol. Section 3.4 introduces the work of Sarvas et al. [SVPN04] who outlined how to build up a collaborative photo album enhanced with contextual metadata provided by the mobile phone.

In section 3.5 and 3.6, we will present recent approaches from the economic sector: first, we introduce file sharing applications for mobile devices running on the Google Android platform (3.5) and second, we will present how the transfer and storage of binary data is handled in a new collaborative tool called Google Wave (3.6). Section 3.7 concludes this chapter and compares presented work to our vision of a collaborative media sharing repository.

3.1. Belimpasakis et al.: Content Sharing Middleware for Mobile Devices

Belimpasakis et al. [BLB07] focus on the use of smart phone cameras as primary source of digital images to be shared. Like us, they distinguish between two primary use cases: sharing media with third party servers which act as intermediary hosts (e.g. Flickr) and sharing media device-to-device (e.g. based on proximity based ad hoc networks or based on remote connections).

In their work, they develop a middleware, which abstracts from specific sharing protocols and lets the user connect to arbitrary sharing repositories to browse or search them or to upload and download files. Also aggregated browsing across multiple repositories is possible. The middleware consists of the following parts (see figure 3.1:

1. An **API** which provides a common interface of the functionality offered by all sharing protocols to third party applications ("sharing applications"). This includes methods for uploading and downloading media, (aggregated) browsing a repository or accessing media's metadata (like EXIF-metadata or thumbnails).

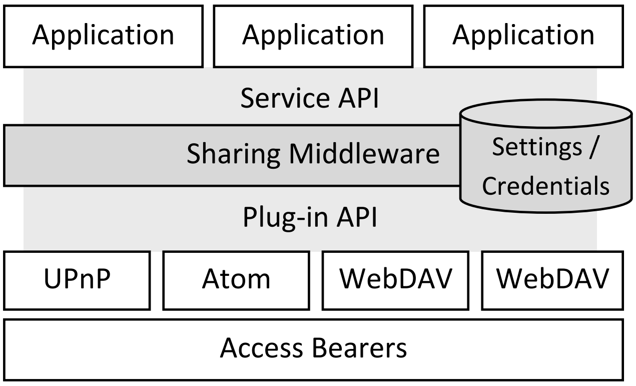

Figure 3.1.: Sharing Middleware Simplified Design

2. **Extension points** to add maximum flexibility for upper and lower layers. New sharing protocols can be added "at the botton" as so-called "sharing plugins" and new functionality for the overlaying sharing applications can be added "at the top" as so called "sharing services".

3. A **middleware configuration utility** to manage the sharing repositories and their respective techniques. The authors anticipate that in future work, this configuration will be much more context-aware or user-centric. One could e.g. use ad-hoc or proximity-based connectivity mechanisms to discover available repositories or one could fetch a list of the available media repositories from a user database, e.g. from the phone book, which could containing a respective field for each person. In our work, we will be presenting a way, how to come closer to this goal using the Mobilis platform with it's services, which feature integration into social networks [DS09].

To pay special attention to restrictions in a mobile environment (traffic cost, battery power, slow CPU), downloaded data is only updated by client request, i.e. when downloaded again. This avoids development of sophisticated caching and conflict resolving mechanisms and is also the way we choose. Furthermore, the API provides clear distinction between the use case of browsing a repository and downloading the content from it. In the first case, only thumbnails and metadata is accessed to save bandwidth.

The discussed sharing protocols are UPnP AV (for the "Home environment domain"), Atom (for the "Internet domain") and WebDAV (for the "Business domain"). In our work, we will concentrate on one single sharing protocol.

The client-server related sharing is driven by the idea to use arbitrary third party servers as sharing repositories. That is, because the introduced sharing application is middleware solution which doesn't provide it's own client-server architecture but rather enables the client to connect to any already existing sharing service. This enables the integration of well-established repositories with associated communities (e.g. Flickr). In this sense, the introduced work is different from our work as we will concentrate on a client-server solution with our own, central Mobilis server. However, there have been already efforts to integrate social networks into the Mobilis platform, so we refer in this case to [HFV+] [DS09].

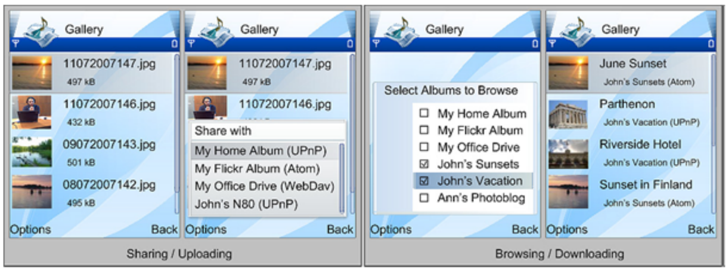

Figure 3.2.: Concept UI for uploading & downloading content [BLB07]

3.2. Tolvanen et al.: Remote Storage for Mobile Devices

Tolvanen et al. [TSLA06] designed and implemented a solution to integrate remote repositories transparently into the Symbian file system. The framework makes it possible to mount practically every arbitrary WebDAV or FTP repository onto a new drive letter of the phone. The contents of the repository can then be used like local files on the device. Even offline access and modification with later synchronisation is possible.

The foundations of this project are inspired by Coda and WebDAV. Pure Coda was not considered to be appropriate in the mobile sector, since it was mainly designed for high-bandwidth carriers. The decision was finally made to use WebDAV, although there were certain limitations in comparison to Coda (no callback-promise, no replication mechanisms, no path-independent identifiers).

The underlaying file systems have been adapted for disconnected operations using sophisticated file caching techniques. In particular, two general approaches of file caching are presented with their trade-offs, especially when applied in the mobile sector.

- **File block caching for immediate file access** Only a part of the file is requested from the server and stored in a cache. The size of the transfered block should be at least the size needed to fulfill the issued `read(..)`-request. In practise, however, multiple `read(..)`-requests are issued in a row which would result in too many round trips. This problem is resolved by requesting bigger blocks, depending on the file structure of the according MIME type and on the typical application behaviour to access this file type (e.g. skimming for metadata, streaming). Once a block is received, it is stored in a cache having a user-defined size and running with a LRU strategy. However, with this technique incompletely cached files cannot be accessed in disconnected mode.

- **Aggressive (whole-file) caching for disconnected access** The complete file is requested from the server and stored in a cache. This allows the user to access the file also in disconnected mode but disables “skimming” files in real time, e.g. for browsing or streaming, without the files being transmitted completely.

To combine the best of both worlds, a hybrid solution is used: files are requested blockwise. Once transmitted completely, they are made available for disconnected operations. The framework allows user intervention (“cache hoarding”), e.g. the definition of sticky flags for the cache using a dedicated user interface. It also provides means to modify files (i.e., the cache contents) when disconnected or “weakly connected” (using e.g. GPRS).

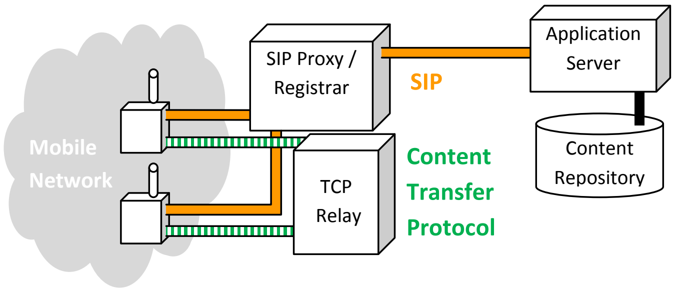

Figure 3.3.: Prototype Architecture

Several methods to resolve conflicts when re-integrating local changes are also discussed in the paper.

The framework consists of an "Enhanced File Browser" for cache control and a "Mounter" on the user interface side. Furthermore, a Remote Storage Client component is plugged in the symbian-own file server which communicates with the "Remote File Engine" managing the communication over arbitrary protocols - e.g. WebDAV and FTP. Multiple mounted repositories all have their own file server thread, so caching can be realized parallely.

3.3. Matuszewski et al.: Mobile Peer-to-Peer Content Sharing Application

The work of Matuszewski et al. [MBLH06] presents a mobile peer-to-peer content sharing service in cellular networks based on SIP (the Session Initiation Protocol), which is part of the IP Multimedia Subsystem (IMS). It introduces an architecture which is based on peer-to-peer media sharing rather then centralized client-server media sharing, since a client-server approach is considered to have poor scalability, a slow content publishing process, a low variety of offered content and the lack of possibility to experiment due to high usage of the centralized server.

A demo is developed using "Registrars" and "Finders" as core components. The "Registrar" provides information to super-peers about the current state of the sub-peers (service started/stopped). The super-peer can subscribe to the information of the sub-peers. The "Finder" component is responsible for generating and receiving SIP XML-messages used for content retrieval. Alongside the core, a "Transfer" component exists for managing the transport of files between two peers (e.g. using TCP/IP). It is responsible for session-initialization, hash-checking and managing local shared content.

This work is only of marginal relevance for our research. Building dual-stack applications using both XMPP and SIP is considered to be difficult [LBSA+09], which is why we consider purely XMPP-based solutions, like introduced in section 2.4 to be a good alternative for file transfer session initialization. Also, the related project was developed relying on GPRS architecture, while nowadays 3G connections like UMTS/HSDPA provide much more opportunities.

Figure 3.4.: The posting process where (a) the images are selected, (b) posting initiated, (c) new folder created, d) named, (e) the people selected, and (f) finally the images are uploaded. [SVPN04]

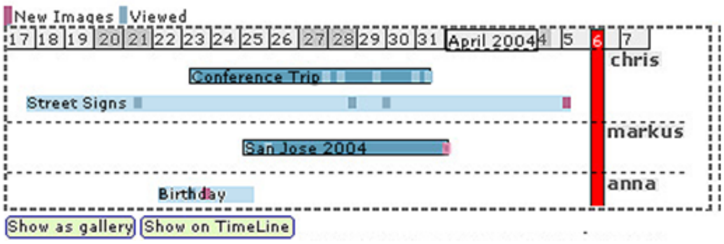

Figure 3.5.: The Horizontal Timeline View of the user's (Chris) own folders and folders shared by others (Markus and Anna). The two selected folders are in darker color. [SVPN04]

3.4. Risto Sarvas et al.: MobShare: Controlled and Immediate Sharing of Mobile Images

Sarvas et al. [SVPN04] developed an architecture for personal image management and photo sharing using mobile phone cameras. The designed and implemented prototype allows posting mobile images into an organized web album. The teoretical work in this paper focusses on issues of user interface design and user habits in taking and sharing photographs and outlines the characteristics of a mobile phone camera being rather a communication device with several network connections, advanced computing ressources (like J2ME or Symbian) and access to contextual or social information which may enrich metadata assigned to a picture.

This work differs from other work presented before in the dual architecture which has been applied: On one side, a mobile phone application based on Symbian is used. On the other side, a web platform on a Struts-driven Tomcat-server exists. This distinction is made because of two general identified use cases:

The first is capturing the picture and sharing it with other users. This is done via the mobile phone (see figure 3.4. After pictures are selected for sharing, they are distributed in two steps. First, the user is asked explicitly to assign the pictures to a folder, which forces a strong organization of the images. Then, the user selects a set of persons to share the pictures with from her phonebook. The phone number (MSISDN) is used as an unique identifier to retrieve the user – users who are not part of the system get invited to the system by a short message.

The second use case is viewing and commenting the picture. This is done on a web platform using a desktop PC, where every user can login using her phone number and password. The contents of the repository are displayed in 3 granularity levels: The highest level is a horizontal timeline view with separated users and their folders aligned vertically. New pictures are highlighted (see figure 3.5). One level down, a user can compare the content of two folders using a vertical timeline view with both folders side-by-side. The second-lowest level is a folder view showing a chronological thumbnail list of one folder. The lowest level finally is the view of a single image with a caption and discussion.

To compare this solution with our approach, we can admit, that todays mobile phones have advanced and the use of them has changed. The distinction between a desktop and a mobile platform is not necessary anymore since todays smartphones, networks and users can handle tasks like displaying and navigating through big repositories on the phone. Nevertheless, the principles defined in this paper, like the dependency on 3 dimensions - user, folder and time - is an important aspect which we incorporated into our design, adding another dimension: location. However, using the phonebook as user base does not apply to our work, since we bet on XMPP and JIDs.

3.5. Android Applications: OnAir and ES File Explorer

The Developer Community of Google's Mobile Phone Operating System Android [1] has developed first applications to facilitate file sharing on mobile devices. Those applications are “OnAir” [Clo] and “EStrongs File Explorer” [webb].

OnAir [Clo] is a free Android software which opens a WebDAV or a AppleTalk server on the Android phone where it is installed. One can then connect to this server with any device running a WebDAV client and having access to the IP address and port of the WebDAV server running on the mobile phone. That way, OnAir allows simple file transfers only under “happy circumstances” – i.e., if the phone and the client are logged in to the same WLAN network. Also, contextual aspects are not taken into consideration, e.g. to tag the transmitted files with metadata and thus allow rich media sharing. Furthermore, this program provides only a simple, directory-based client-server solution and social information is not taken into account to allow collaborative sharing.

EStrongs File Explorer [webb] is another free Android application which can browse shared directories to NetBIOS servers running e.g. on Windows platforms or as Samba servers on Linux platforms. In contrast to OnAir, the application is realized as a client while the paired device has to run the server. However, similar restrictions apply: no

[1] http://developer.android.com/

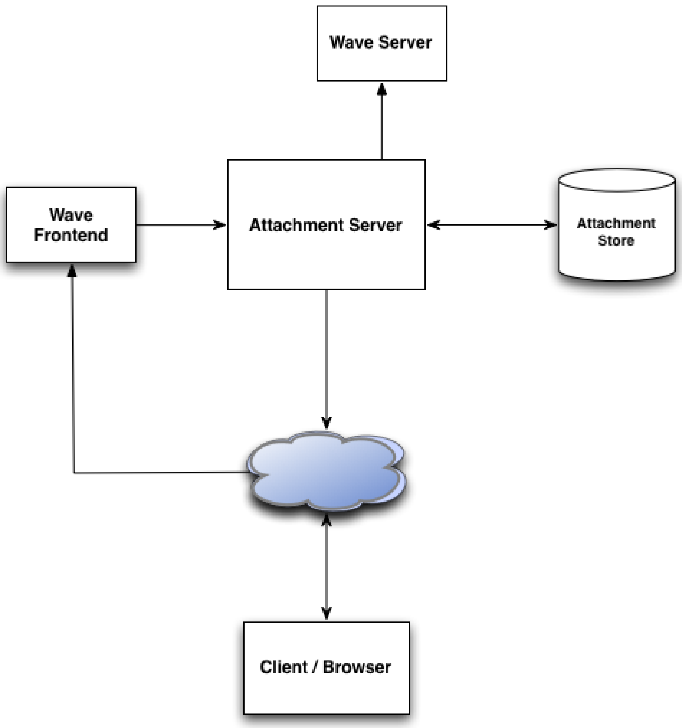

Figure 3.6.: The Google Wave Attachment Server Architecture [LT09]

context or social information are used and no collaborative sharing is possible. Also, the server has to be reachable by its IP address and port. Note, since we consider the paired device to be a desktop station, this is easier since desktop stations are more likely accessible from a known IP address than mobile phones.

3.6. Google Wave Attachments (Google Wave Federation Protocol)

Google Inc. recently published a collaborative product called "Google Wave" [2] which tries to combine communication tools like email, instant messanging, wikis, collaborative editors and other collaborative "gadgets" into one single extensible product. The elementary unit in "Google Wave" is a "wave", a conversation object hosted on the server in which multiple users participate. A wave is split up into a tree structure of wavelets which user can read, modify or reply to in near real-time. The protocol used to propagate those changes is called the Google Wave Federation Protocol, and it will be be open-source according to Google Inc. Already now, multiple white-papers are available [web09].

Inside of wavelets, binary data can be embedded. In [LT09], Michael Lancester explains how binary data is stored on so-called attachment servers, how it is referenced inside of XML-based wavelets and how changes to the attachment repository are communicated (see figure 3.6).

[2] http://wave.google.com/

The attachment server, where all binary data of the attachments is stored, is an HTTP server. It supports upload (HTTP POST) and download (HTTP GET) of the attachments, which are stored in a database. For every attachment, one row exists, storing the thumbnail, the actual attachment content and the metadata of the attachment. Thumbnail and content are stored in BLOB fields (Binary large objects), the metadata is stored in a protocol buffer [Gooa], a format developed by Google Inc. to store serialized data efficiently. Every attachment is further referenced by a globally unique ID. Inside of the wavelet object, which embeds the attachment, a portion of the metadata is repeated – if changes are done to the metadata at the attachment server side, they are pushed via Remote Procedure Calls (RPC) to the wave server.

Creation and modification of attachments can be done in various ways: a thumbnail can be uploaded via HTTP POST, the attachment content (or non-overlapping parts of it) can be uploaded via HTTP POST or a link to an existing attachment can be created. These operations are idempotent, that means, they can be executed in parallel. With every attachment operation, the attachment ID, wavelet name and other metadata is specified as HTTP headers. An attachment is downloaded by an HTTP GET request with special HTTP headers specifying the attachment ID and a token, stored inside the wavelet object.

In contrast to our work, Google Wave concentrates on hosted conversations (wave objects) with participants instead of locations as the key concept of media exchange. It also is not targeted primarily to the mobile sector. Also, it uses XMPP only marginally to store binary media – in fact, most of the communication takes part using HTTP and RPC. Furthermore, the introduced technology is not standardized by any authority.

3.7. Conclusion

In the previous sections, numerous previous efforts in media sharing on mobile devices were introduced which covered the basic scenario introduced in 1 by using technologies described in 2. However, none of them provided the necessary features to fulfill the requirement of a collaborative social mobile media sharing platform integrable into an XMPP-powered system. Most of the introduced examples concentrated on directory-based file sharing or file transfers without taking the numerous affiliations of the user and available context information or metadata into account. The work of Sarvas et al. (3.4) comes closest to our vision but it lacks the support for a standardized technology like XMPP. Furthermore, location awareness is not supported. Also, not all mobile use cases are covered, since management, viewing and discussing the media ressources is done in a web browser even though mobile phone operating systems, SDKs and handsets have dramatically improved which allows to build applications with advanced functionality.

As we showed in chapter 2, the XMPP standards foundation lately developed appropriate standards to make media sharing with annotated metadata possible using XMPP. In fact, none of the works specified before ran applications using the XMPP protocol – despite of the Google Wave Foundation Protocol, which, although, uses proprietary negotiation mechanisms not developed by the XMPP standards foundation. Also, most of the communication in this protocol takes part out-of-band from the XMPP stream without proper XMPP negotiation.

In the next chapter (4) we will thus settle requirements to detail our vision of a new collaborative metadata-driven mobile media sharing platform based on XMPP.

4. Requirements Analysis

Based on the preceeding introduction of the tourism scenario (1.2) which showed the business need for a collaborative mobile media sharing platform, we will now introduce the requirements set to such a system to improve currently available systems introduced in the previous chapter (3). This system will run on top of available collaborative media sharing technologies evaluated in chapter 2 and be integrated in the current Mobilis system presented in section 1.1.

Requirements are descriptions of what a system should do and how this goal should be reached. Therefore, we will start in section 4.1 by introducing functional requirements based on the presented user scenario. We will oppose them to the minimum device capabilities needed to accomplish this goal in section 4.2. Then, non functional requirements are introduced in section 4.3 and finally Human-Computer Interaction considerations are presented in section 4.4. At the end of each section, the elaborated requirements are summarized in a prioritized table (tables 4.1, 4.2, 4.3 and 4.4). The last section (4.5) of this chapter concludes the key guidelines for developing the prototype system.

4.1. Functional Requirements

The primary functional requirements in our media sharing scenario is the publication of pictures (FR-1). In mobile environment, pictures are usually taken with the mobile phone camera, so it should be possible to load them directly in the sharing application (FR-1.1). However, the system should not be restricted to store pictures only – moreover it should be possible transmit any file to the repository (FR-1.2). The selected picture or file may be sent to a picture repository for other users to retrieve it (FR-1.3) or only a thumbnail file with metadata may be sent to the repository and the actual file may stay at the users device for users to retrieve it later (FR-1.4). The user may also choose the option to share the image with one single user only (FR-1.5), where a direct and instant file transfer would be initiated without the repository seeing any of the transmitted data.

The prototype should support the management of metadata stored alongside with the content (FR-2). First, the image is geotagged with the location where the image is taken or, if this information is not available, with the location where the image is stored into the repository (FR-2.1). The image is also tagged with a date from when it is taken or, if this information is not available, with the date of the upload (FR-2.2). Furthermore, the image was shared by a specific owner, which classifies images according to their user (FR-2.3). The user may finally enter a free title which is stored as well (FR-2.4). All those classifications should be further classifications (FR-2.5).

The repository should be browsable by other users (FR-3). Browsing the repository means retrieving information about a set of pictures meeting a specific condition. Browsing has to be done using easy-to-use filtering controls (FR-3.1). The repository should be browsable by the location classification using map view (FR-3.2), by date by entering a start and end date in a calendar view (FR-3.3) and by the creator of the picture (FR-3.4). The system should moreover support custom types of browsing (FR-3.5).

After the repository has been browsed and the user has found a picture of interest, she can invoke certain actions on it (FR-4). This includes displaying all metadata associated with the picture (FR-4.1), replacing the picture by another (FR-4.2), downloading the item to the local disk (FR-4.3) or deleting the item (FR-4.4).

Security considerations have to apply during the whole process (FR-5). The repository should have provide access control – i.e., not every user should be able to browse or download from the repository, what may be accomplished by using the roster as a whitelist (FR-5.1). On a picture level, all modifications to the picture, i.e. deletion (FR-4.4) and replacement (FR-4.2) should be possible for the owner only, if not more sophisticated access control mechanisms are introduced (FR-5.2). If possible, the data transfer connection should be encrypted (FR-5.3).

Finally, it should be possible to use existing functionality provided by the Mobilis Platform (FR-6). The JID should be the central entity, which identifies users and services (FR-6.1). Functionality from the previous prototypes MobilisBuddy [HFV+] and MobilisGuide [Kor08a] should be integrated (FR-6.2) and the status of other work in the Mobilis Project has to be followed (FR-6.3).

	Description	**Priority**
FR-1	Publishing of content	High
FR-1.1	Publishing of images taken with the phone camera	High
FR-1.2	Publishing of arbitrary files	High
FR-1.3	Publishint to a group of users (client-server)	High
FR-1.4	Publishing to a group of users (hybrid)	Low
FR-1.5	Publishing to a single user (peer-to-peer)	Medium
FR-2	Metadata classifications	High
FR-2.1	Spacial classification	High
FR-2.2	Temporal classification	High
FR-2.3	Classification concerning ownership	High
FR-2.4	User provided title	Medium
FR-2.5	Arbitrary classification	Medium
FR-3	Browsing of content	High
FR-3.1	Specialized views for every type of browsing	High
FR-3.2	Browsing based on location by map view	High
FR-3.3	Browsing based on date by calendar view	High
FR-3.4	Browsing based on creator	High
FR-3.5	Browsing based on arbitrary criteria	Medium
FR-4	Interacting with content	High
FR-4.1	Listing of the complete metadata set of one item	High
FR-4.2	Replacing the item by another	High
FR-4.3	Downloading the item	High
FR-4.4	Deleting the item	High
FR-5	Security	Medium
FR-5.1	Access control on repository level (e.g. using Roster)	Medium
FR-5.2	Access control to content level (e.g. using ownership)	Medium

	Description	Priority
FR-5.3	Privacy considerations (e.g. encryption)	Low
FR-6	Support of previous and future work	High
FR-6.1	JID as central identity	High
FR-6.2	Seamless integration of MobilisBuddy and MobilisGuide	Medium
FR-6.3	Coordination with current Mobilis project	Medium

Table 4.1.: Summary of Functional Requirements

4.2. Device Capabilities

To realize the preceedingly explained use cases, the mobile device and the mobile network have to fullfil some necessities concerning capacity, performance and equipment. First of all, collaborative solutions, especially such, which are based on client-server or peer-to-peer architecture, require constant connectivity to a wireless networks (DC-1). Given the relatively large size of media files sent media sharing scenarios, even newer wireless technologies with sufficient bandwidth and data rate – so-called 3G networks – should be available and supported by the device (DC-1.1). The device may also connect to pure IP networks like WLAN or be handovered between several networks (DC-1.2). However, generally the user can be assumed to be “always online” (DC-1.3).

To allow spacial classification of shared images (see FR-2.1) the device has to provide sensors to determine its location (DC-2.1). This can be done by GPS triangulation (DC-2.1) or by location determination using the service provided through the mobile network, like Assisted GPS (A-GPS), location determination by the networks CellID or a database containing WLAN-Footprints (DC-2.2). If all automated or assisted location determination functionality fails, the device should offer the possibility to enter the location manually (DC-2.3).

For publishing images taken by the mobile phone camera (see FR-1.1), the mobile phone flash drive can be accessed (DC-3.1). Furthermore, the Camera-API should allow aquiring images directly from the mobile phone camera (DC-3.2). Geotagging of taken images (DC-3.3) and date-tagging (DC-3.4) should be possible for further classification (see FR-2).

Finally the development environment for applications on the mobile device should support common software technology practises (DC-4) to facilitate easy and modular development (see FR-6). The operating system which suits this purpose best is the Android Operating System with the Android Software Development Kit (SDK), currently available in release 1.6, which is used in the Mobilis Project (DC-4.1). Android applications are written in Java (DC-4.2) and are portable between any handset, which runs the Android platform (DC-4.3).

	Description	Priority
DC-1	Connectivity to a wireless network	High
DC-1.1	Wireless network has sufficient bandwidth	High
DC-1.2	Occasional availability of higher capacity wireless networks like WLAN	Medium

	Description	Priority
DC-1.3	"Always-online" assumption	High
DC-2	Location sensitivity (see FR-2.1)	Medium
DC-2.1	Automatic location retrieval using GPS	High
DC-2.2	Assisted location retrieval using A-GPS, CellID or WLAN-Footprint	Medium
DC-2.3	Manual setting of location	Low
DC-3	Mobile images accessible	High
DC-3.1	Aquiring media from images stored on the mobile phone flash drive	High
DC-3.2	Aquiring media directly from the camera	Low
DC-3.3	Geo-tagging of taken pictures	High
DC-3.4	Date-tagging of taken picture	High
DC-4	Developer-friendly Platform (compare to UC-6)	High
DC-4.1	Operation system: Android 1.6	High
DC-4.2	Programming language: Java	High
DC-4.3	Portability	Medium

Table 4.2.: Summary of Device Capabilities

4.3. Non Functional Requirements

In addition to functional requirements to realize and the device capabilities to be assumed, there are a number of non functional requirements concerning design and later implementation of the collaborative media sharing platform. First, the designed architecture should provide a maximum of exchangability, variability and reusability according to best-practices of software technology (NF-1): It should fit into the service oriented architecture of the Mobilis Project defined in [Mac07] (NF-1.1) and thus be a layered framework architecture using design patterns [GHJV95] (NF-1.2). Concerning extensibility, especially support for new media types (other than images) (NF-1.3) and future media sharing technologies (e.g. Jingle) (NF-1.4) should be added.

During development, standardized and elaborated technologies and frameworks should be used, where applicable (NF-2). We use XMPP as collaborative protocol (NF-2.1). If possible, the transfer of binary content should underly a standardized protocol (NF-2.2). The implementation of this protocol should be reusable for future work (NF-2.3).

Finally, the mobile sector requires some special considerations which do not apply to the desktop sector (NF-3). A mobile connection may face sudden loss or handover, to which the application has to adapt (NF-3.1). Also, transmitting data should be considered expensive and thus minimized. (NF-3.2). Firstly, because data fares are still higher than on fixed networks, especially in the tourism case, where roaming rates apply (NF-3.2). Secondly, because sending data over the air is battery consuming (NF-3.3). Thus, file transfers have to be optimized, that is, compressed (NF-3.4). On the other side, a certain trade-off has to be made considering computational expensive operations, which should be delegated from the weak mobile device to the server (NF-3.5). What comes to peer-to-peer connections, one should consider, that the opposite party may use an unreliable connection, like oneself may do (NF-3.6).

	Description	Priority
NF-1	Exchangability, variability, reusability	High
NF-1.1	Integration into the Mobilis architecture (see UC-6)	High
NF-1.2	Layered service architecture, Use of Design Patterns	High
NF-1.3	Support for other media types	High
NF-1.4	Support for other media sharing technologies (e.g. Jingle)	Low
NF-1.5	Server support for other clients / client platforms	High
NF-1.6	Reusability on client level for other applications	High
NF-2	Standardized or elaborated technologies	Medium
NF-2.1	Collaborative protocol: XMPP	High
NF-2.2	Standardized file transfer protocol	High
NF-2.3	Implementation of standardized file transfer reusable for other applications	High
NF-3	Considerations concerning the mobile environment	High
NF-3.1	Adaptions to the mobile network	High
NF-3.2	Low traffic cost approach (consider roaming!)	High
NF-3.3	Energy awareness	High
NF-3.4	Efficient file transfers, use of compression	Medium
NF-3.5	Server-client balance (traffic vs. computation power)	High
NF-3.6	Keep in mind peer-to-peer connection problems	Medium

Table 4.3.: Summary of Device Capabilities

4.4. Human Factors

Although the mobile user interface is not the main focus of this work, there are some issues which should be highlighted and cannot be underestimated. Comparing using a mobile phone and using desktop PC, one can recognize, that the mobile phone is used in a context, where the user may not be primilary focused on solving tasks with the mobile phone itself. Especially in the tourism case, the mobile phone is rather an accessoire than a device where oneself puts all ones attention to. Thus, input behaviour differs from working on a PC (HF-1). Operations should be executed with the fewest clicks possible (HF-1.1) and suggestions should be provided upon typing considering small mobile phone keyboards (HF-1.2). UI elements should be arranged in a clear way for the user to find her desired operation immediately (HF-1.3). Multitasking should be provided to execute file transfers in the background, so users can deal with other tasks in the meantime (HF-1.4).

The user interface should be goal-oriented to imitate the users thinking (HF-2). This means, that multiple image collection views have to be shown (see NF-3) to allow the user to solve the task of finding one image without the need to search manually (HF-2.1). The map, however, should be the main artifact (HF-2.2), since location is the most important concept in mobility [Lau]. Finally, it should be possible to interlink between different social objects (HF-2.3), that is, e.g., to contact a user who published a certain picture or to show all pictures of a certain user.

	Description	Priority
HF-1	Mobile phone input considerations	High
HF-1.1	Fewest clicks possible	High
HF-1.2	Suggestions upon typing	Low
HF-1.3	Clear arrangement of UI elements	High
HF-1.4	Multitasking / background downloads	High
HF-2	Goal-oriented user interface	High
HF-2.1	Multiple image collection views (see UC-4)	High
HF-2.2	Map as main artifact (see UC-4.1)	High
HF-2.3	Interlink between social objects	Medium

Table 4.4.: Summary of Human Factors

4.5. Conclusion

In the previous sections, functional requirements were elaborated from the introducing user stories and minimum device capabilities to realize those requirements where identified. Besides, non-functional requirements and human factors where gathered. While non functional requirements and device capabilities focus mostly on the collaborative media sharing aspect of our work, non-functional requirements and human factors take the special conditions of a mobile environment into account, stressing both the nature of mobile networks as well as mobility and thus the necessarity for location sensivity. In the following chapter, we will introduce the design of such a media sharing system, which will meet the requirements preceedingly defined.

5. Conceptual Design

This chapter introduces an architecture of distributed entities communication via the XMPP protocol to exchange media (or, in more specific, images), store that media in a repository or access it according to the requirements settled in the previous chapter. In the first section (7.1) we will review file transfer technologies from the foundations chapter (2) and decide upon one technology for one-to-one file transfers. We will then work out a repository architecture which fits best our needs for a multimedia repository of our requirements (5.2), continue by discussing methods to break down the repository into several so-called "broker services" and finally choose a decomposition which fits best to the mobilis architecture (5.3). For this decomposition, we will work out the service primitives and how they translate to sequences of XMPP-IQ messages (5.4) issued between the broker services. Section 5.5 concludes this chapter.

5.1. Finding an XMPP-based File Transfer Protocol

In chapter 2 we presented four possible file transfer mechanisms: SI File Transfers, Jingle File Transfers, WebDav and APP. We compared these technologies based on different criteria in table 2.1. In this section, we explain our decision for one of them and the resulting protocol architecture.

5.1.1. SI File Transfers

According to our findings, we decided to build a media sharing environment based on SI File Transfers. With that come several advantages: First, the underlaying XMPP protocol stack does not have to be extended by a second protocol stack (SIP, WebDav, APP) possibly resulting in a parallel client-server architecture which has to be maintained asynchroneously. Instead, the Mobilis framework can continue to use XMPP as the only core technology and that way build on a future-proof protocol.

Another argument for SI File Transfers is the available library support. The Mobilis Framework runs on Android devices (DC-4.1), with user applications based on Java (DC-4.2). At Mobilis, we are using the Ignite Realtime Smack library to realize communication via XMPP. Smack also supports stream initiation requests. The library was found to be portable to the Android platform while other java-based file transfer libraries, like Apache Slide (for WebDav) or Rome Propono and Apache Abdera (for APP) might not be so cooperrative.

For Jingle, no suitable library existed, since this technology is a relatively new, but newertheless promising technology. An argument for Jingle is that it is generic and usable in other out-of-band-sessions. However, this protocol fastly becomes heavyweight in that context, especially for mobile devices. Furthermore, a complete implementation of the Jingle-standard is behind the scope of this work.

5.1.2. One-to-one File Transfers

A one-to-one file transfer using stream initiation requests is executed as described in the "Foundations" chapter in section 2.2.1. We take this one-to-one file transfer as the atomar building block to publish a file peer-to-peer to another user (FR-1.5). Later (in subsections 5.4.4 and 5.4.5) we will show, how this atomar file transfer can be used in a custom XMPP extension to store files into a repository (FR-1.3) or to allow a directory-based hybrid technique with pulling the file from the user on-demand (FR-1.4).

5.1.3. One-to-many File Transfers

The XMPP community already developed a mechanism to allow file storage and pulling from a central repository: Published Stream Initiation Requests (XEP-0137 [MM05]). It is based on the idea to hold a pubsub trees containing pubsub items, each one representing one repository item enhanced with metadata. However, plain published stream initiation request does not fit our requirements of in two ways: First, multidimensional metadata (FR-2) cannot be mapped logically onto a pubsub tree. Second, a hybrid solution (FR-1.4) where the actual file content stays on the sender side until it is pulled by a receiver, is hard to realize, since the server cannot store any actual media content but only a reference where it can be found. In the next section, we will introduce a solution to this problem.

5.2. The Cube Media Repository

According to our requirements, media content – in our scenario, images – are classified by a range of metadata aspects. For our prototype requirements, we chose location, date and ownership (FR-2.1, FR-2.2, FR-2.3). Since these aspects are independent from each other, they cannot be mapped onto a tree structure without loosing this independence. Instead of a tree structure, which lays the foundation of the pubsub mechanism, a **"cube"** structure is desired. This cube structure is visualized in figure 5.1. The "cube" contains every repository item at a discrete position inside a multidimensional hyperspace spanned by all orthogonal dimensions of metadata classification. We call the position of an item in every dimension a **slice**, so that every item is assigned to a number of slices, one slice for each dimension.

When browsing this repository (FR-3) it should be possible for a requester to filter only repository items located in a well-defined area of the hyperspace. Additionally to slices, a repository item holds a set of rigid properties: alongside with a unique identifier **uid**, the user which has **ownership** of the item (i.e., who uploaded it) is saved. Also, a reference to a **content store** is held.

The content store physically stores the binary content of the repository item along with some information needed for efficiently carrying out SI File Transfers. This is, to large parts, the mime-type and file size of the stored content.

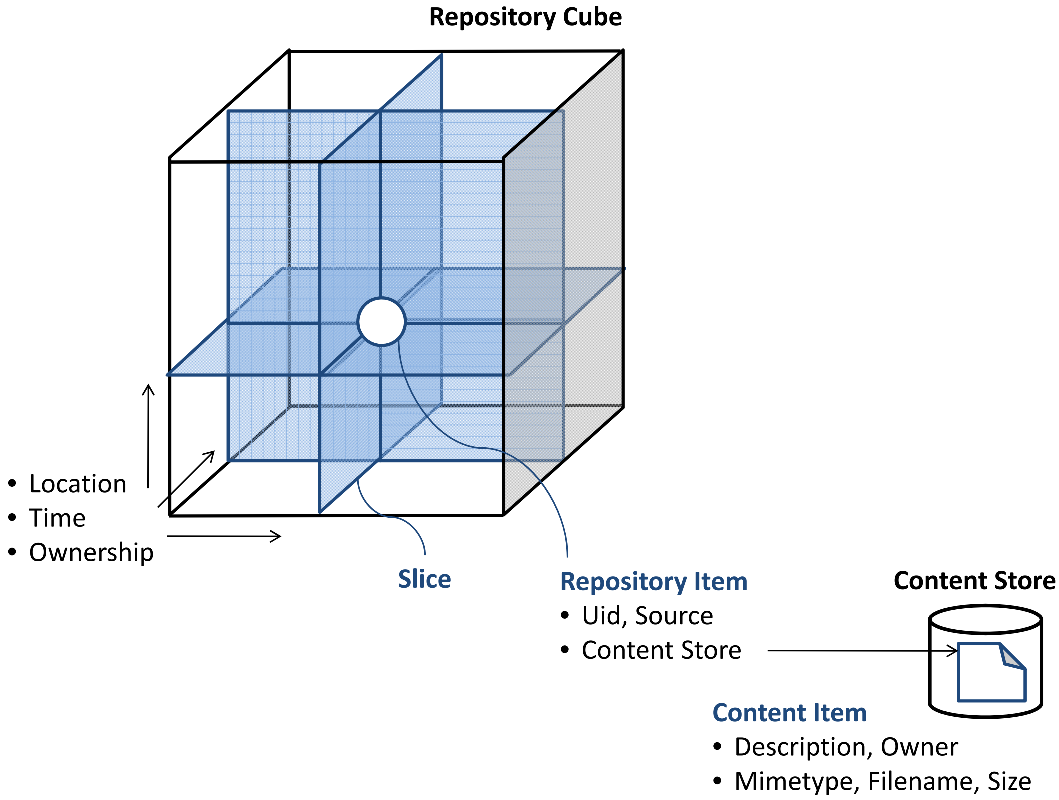

Figure 5.1.: Multidimensional metadata classification system “cube”

5.3. Breaking down the Architecture

5.3.1. Decomposition into Entities

The decomposition of the design into two areas of concern – management of a **repository cube** and a **content store** – is an important principle for the future design. It allows load-balancing: while the repository cube is in charge of classifying repository items and browsing the repository in – possibly complex – queries (FR-2.5), the content store only holds the actual content but therefore needs sufficient storage space.

There are various approaches how both repository cube and content store can be incorporated into a mobile networked system. Three general ways are shown in figure 5.2. The most simple realization (**combined scenario**) is done by combining repository cube and content store in one single entity, which is connected to its mobile clients. This approach is easy to implement and has few management overhead but also allows no form of load balancing like described above.

A **distributed scenario** realizes content store and repository cube in different entities. This allows repository cubes to distribute contents to multiple content stores for load-balancing. However, it requires the development of a sophisticated algorithm to choose the content store which should be used, if a new repository item should be stored. A content store then could also have multiple repository cubes registered and a mobile client might finally be able to connect to one or more different repository cubes.

The **hybrid scenario** facilitates a peer-to-peer media sharing architecture with central

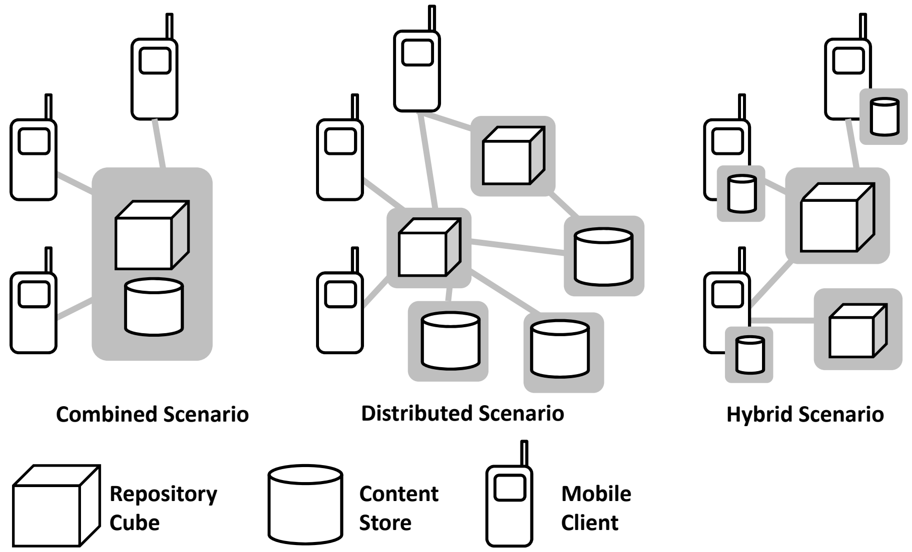

Figure 5.2.: Decomposition approaches of architecture in repository service and content broker

repository cubes working as directories. However, here, the actual file content stays at the client side, so the client itself plays the role of a content store. This scenario is therefore close to the hybrid scenario introduced in requirement FR-1.4. In this situation no additional server storage is required, but client outage due to disconnections from the network must be carefully considered, in which case contents from the repository might not be available (NF-3.6).

It should be noted, that these three approaches are not discreet options but rather principles, which might be combined. It is indeed possible to develop a heterogenous system, in which some content items are stored on the client devices (hybrid scenario) while other files are stored in a central independent content store. Since an item in a repository cube always points to the correct content store, the content may always be localized by a requester.

5.3.2. Decomposition regarding the Mobilis Architecture

The Mobilis architecture introduces a Mobilis server which offers several services to a client, while a set of services is always encapsulated by one single "broker services" [DS09] [DS09]. To allow communication of a client with its broker service, Mobilis server and client connect to an XMPP server opening a bidirectional XML-stream over TCP. The overall architecture of the Mobilis platform is shown in figure 5.3.

By convention, the XMPP server acts under one XMPP user, e.g., `mobilis@xmpp`, with each broker service having its own XMPP connection using a seperate XMPP resource, i.e., `mobilis@xmpp/Buddy`, `mobilis@xmpp/Coordinator` etc. This convention brings several advantages:

First, an end user, who configures her client, has to know only the Mobilis server XMPP user (`mobilis@xmpp`) and does not have to take care about all the broker services which

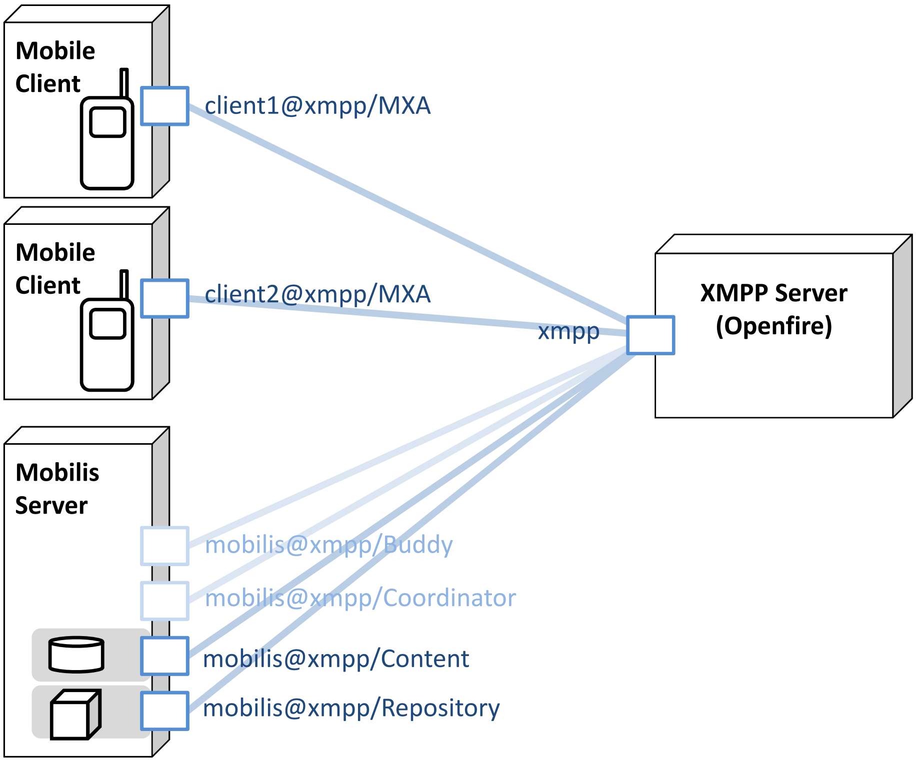

Figure 5.3.: Decomposition with regard to the Mobilis archtiecture

are running on the server. The broker services, that is, the resources of the connected XMPP user (`mobilis@xmpp`), however, might be easily queried from the XMPP server by sending a Service Discovery message for all items to mobilis@server. The XMPP server will reply to this message with a list of connected resources if both Mobilis server (`mobilis@xmpp`) and the client XMPP entity have each other at their XMPP roster with a subscription state set to "both" (see [HMESA08]). This mechanism provides also a simple trust mechanism to be negotiated between Mobilis server and client before any communication can occur between both (FR-5.1).

Second, since every broker service has their own XMPP connection, the broker services can be deployed as well on different physical machines as also be realized in one software package only. Having all the same XMPP user (`mobilis@xmpp`), the only requirement for setting up a new broker service next to existing ones is to know the account data of the server XMPP user (`mobilis@xmpp`) which is a fair amount of trust that a broker service can work with the other broker services under the name of a combined XMPP server.

For our media repository, we introduce two new broker services: a repository broker (mobilis@xmpp/Repository) and a content broker (mobilis@xmpp/Content). Although they are realized in the same software package (the Mobilis server prototype) they are not connected to each other but communicate with each other via XMPP only. Hence, the realization can be classified as a **distributed scenario** (see subsection 5.3.1) with one repository cube and one content store. However, we will see, that the realization also contains elements of the hybrid scenario, as the Mobilis server can decide to leave the content on the client device without an upload process to the content broker (FR-1.4). In

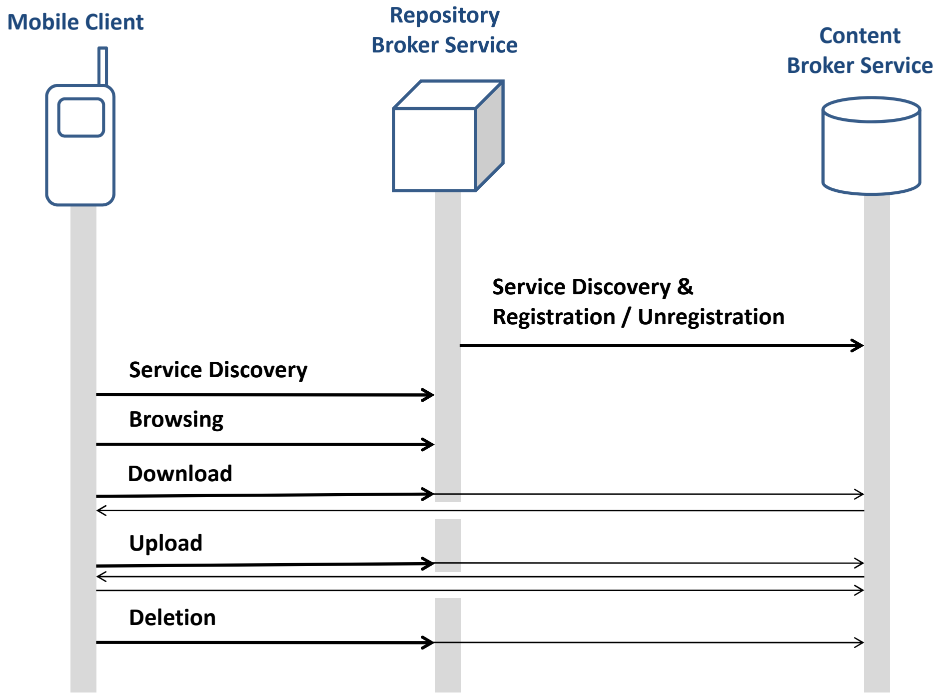

Figure 5.4.: Service primitives of content service and repository service

this case, the client plays the role of the content broker and can be contacted to deliver its contents right to requesters.

5.4. Service Primitives

Figure 5.4 shows the service primitives, that is, the fundamental requests which can be issued by a user to the media repository. The fundamental service primitives are browsing of the repository (FR-3 and FR-4.1) as well as upload (FR-1) / replacing (FR-4.2), download (FR-4.3) and deletion (FR-4.4) of content. Additionally, a service discovery and registration / unregistration mechanism is used to couple the user, repository broker and content broker together.

It should be outlined, that the mobile client issues its requests to the repository service only. However, despite of service discovery, browsing the repository is the only primitive which the repository broker may fulfill on its own. In all other processes (download, upload and deletion), a content broker is involved in handling a request. The repository service will then itself generate a specific request and issue it to the content broker service. This would be, for content deletion, a request to remove the content from the content store. In case of download, this is a request to send the content to the requester. And finally in case of upload – the most complex scenario – the repository broker requests the content broker to ask the mobile client to transfer the file to the content broker.

Figure 5.5 shows that each service primitive translates to a set of XMPP IQs issued between the entities. These IQs are either taken from a public XEP or they are custom IQs. Requests are always sent by Set-IQs and Get-IQs, where Set-IQs are ment to change the persistent state of a receiver and Get-IQs are a mean to retrieve information. According

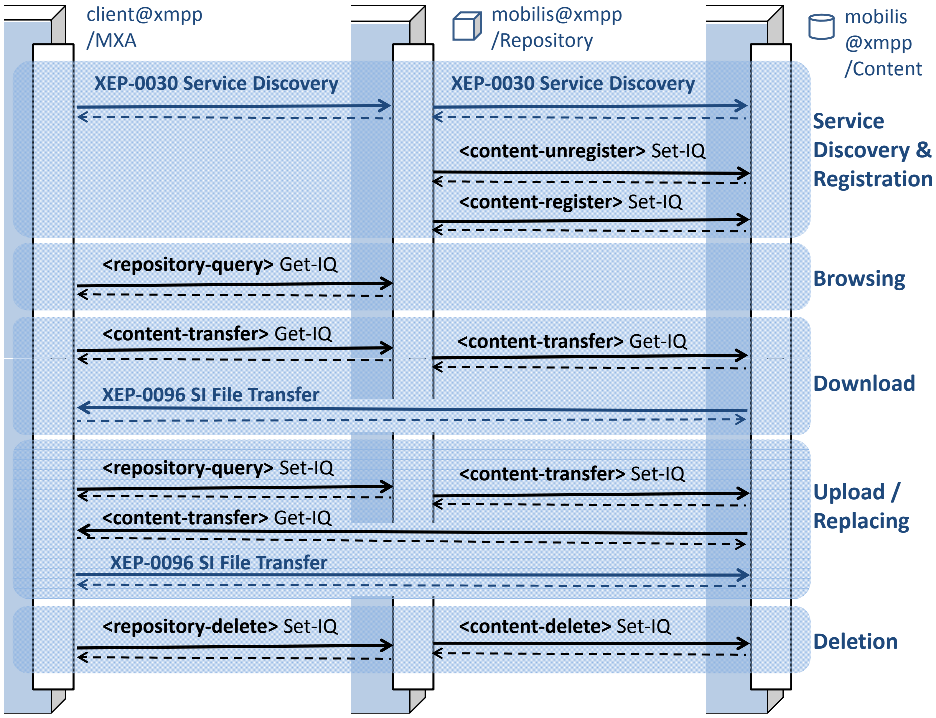

Figure 5.5.: Service primitives of content service and repository service detailed by IQ

to the XMPP specification [SA04b] every Set-IQ and Get-IQ has to be acknowledged by a Result-IQ or Error-IQ, depending on if the request could be fulfilled or not. In our design, the Result-IQ following a Get-IQ will always carry the requested information. A Result-IQ following a Set-IQ may contain nothing or the repeated request payload, parts of it or detailed information about the performed actions.

5.4.1. Custom IQs

The custom IQs used to manage the media repository contain either a child element of the form `<repository-.../>` or `<content-.../>`, depending on if they are handled by the repository broker or the content broker. The only exception is the `<content-transfer/>` IQ, which is sent to the repository broker but will be directly forwarded then to the respective content broker. The child elements `<repository-.../>` are declared in the namespace `http://rn.inf.tu-dresden.de/mobilis#services/RepositoryService` and the child elements of the form `<content-.../>` are declared in `.../ContentService`.

Every child element contains further child elements as a payload to detail the request or acknowledgement. Figure 5.6 illustrates both namespaces with their containing IQs. Refer to section A.1 of the appendix for a complete XSD schema definition of all custom IQs.

The Mobilis Namespace

<condition/> is an XML element to describe arbitrary boolean conditions on a set of key-value pairs. It is used inside of a **<repository-query/>** tag to browse a

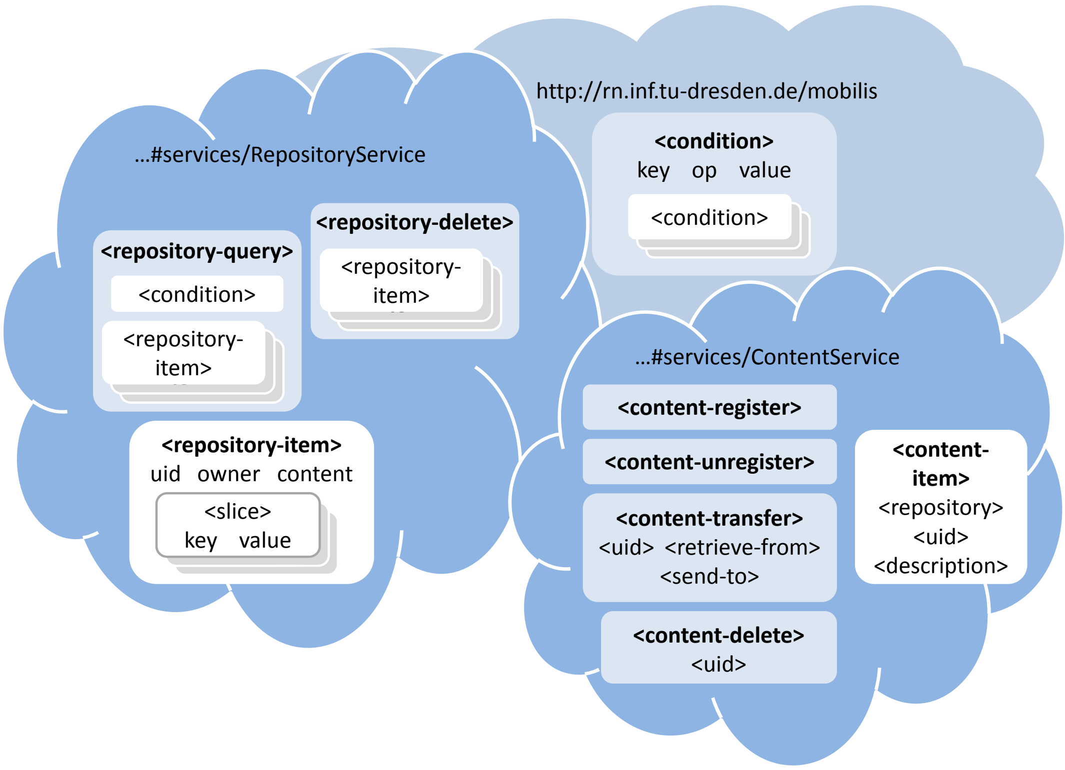

Figure 5.6.: Mobilis Namespaces and their Custom IQs

repository and filter out certain items of it in an arbitrary way (FR-3.4). However, since the backing concept is more widely applicable, it was decided that the element should be included in the core Mobilis namespace. A `<condition/>` contains an `op` attribute describing the operator. `op` may be either one of the comparison operators `eq` (equal), `ne` (not equal), `gt` (greater than), `lt` (lower then), `ge` (greater or equal), `le` (lower or equal) or one of the logical operators `and`, `or` or `not`. In case of a comparison operator, the `key` and `value` attribute represent the values to be compared. In case of a logical operator, one or more `<condition/>` elements may be used as subelements to represent subterms.

The RepositoryService Subnamespace

<repository-query/> is used as an IQ child element in the primitive to **browse** the repository (FR-3) and to **upload/replace** items into the repository (FR-1 and FR-4.2). Depending on the actual primitive and whether it is used as a request or as an answer, the payload of the element is either a `<condition/>` element, that describes which repository items should be queried, or one or more `<repository-item/>` elements describing the repository elements which are inserted into the repository or read-out while browsing the repository.

<repository-delete/> is used as an IQ child element in the primitive to **delete** items (FR-4.4) from the repository. The element contains as a payload one or more `<repository-item/>` elements which should be deleted.

<repository-item/> is a child element used in both `<repository-delete/>` and `<repository-query/>`. It provides information about a item stored in the repository. According to our model specification (see section 5.2) these can be rigid values like a unique identifier (`uid` attribute), the owner / creator of the repository item (`owner` attribute) or a reference to the content broker where the item is stored (`content` attribute). Furthermore, the slices of the repository item may be listed as a sequence of `<slice/>` child elements inside of the `<repository-item/>` element. Every `<slice/>` element will contain a `key` and a value attribute to identify the dimension and position of the slice. Not all information about a repository item has to be represented inside of a `<repository-item/>` element. If it is, we speak of a **complete** `<repository-item/>` element. If only `<slice/>` elements are specified, we call the `<repository-item/>` element **simple**, if additionally a `uid` attribute is present **concrete**. If *only* a `uid` attribute (and optionally a `content` attribute) is present we speak of a **referencing <repository-item/>** element.

The ContentService Subnamespace

<content-register/> and <content-unregister/> are used as IQ child elements to **register and unregister** a repository broker service at a content broker service. Since the intent of the request (repository and content brokers JIDs) can be determined from the sender and receiver of the IQ packets, no payload has to be attached to these elements.

<content-transfer/> is used as IQ cild element during the **download** (FR-4.3) and **upload** (FR-1) / **replace** (FR-4.2) primitive to request the initiation of a file transfer. It consists of three child elements containing plain text only: `<retrieve-from/>` being the source where the content item should be retrieved from, `<send-to` containing the destination of the file transfer and `<uid/>` to identify, together with the repository from which the `<content-transfer/>` request was issued the content item itself.

<content-item/> is used as a child element of the `<desc/>` subelement inside a Stream Initiation Request based file transfer following on a `<content-transfer/>` request to assign the SI File Transfer uniquely to a content item.

<content-delete/> is used as IQ child element in the **deletion** (FR-4.4) primitive to delete content from a content broker. It as the only child element an `<uid/>` element to identify the content item to be deleted. Since the `<content-delete/>` can only be issued from the repository broker, the content item can be uniquely identified that way.

5.4.2. Service Discovery & Register / Unregister Service Pimitive

To start communication of the participating entities, it is first necessary, that the mobile client knows the JID of the repository broker and the repository broker the one of the content broker. Per default, the address of the mobilis server (`mobilis@xmpp`) is already known to both entities. The challenge is to find out the correct ressource(s) of the repository and content broker(s).

In case of the client discovering the repository broker service, it will first query the connected resources of `mobilis@xmpp` from the XMPP Server. This is possible using the following XEP-0030 service discovery mechanism [HMESA08]:

```
<iq type='get' id='mobilis_1'
    from='client@xmpp/MXA' to='mobilis@xmpp'>
  <query xmlns='http://jabber.org/protocol/disco#items' />
</iq>
```

If both Mobilis client and Mobilis server are on each others roster and subscribed to each other, the XMPP server will return the connected resources, in this case equivalent to the list of broker services:

```
<iq type='result' id='mobilis_1'
    from='mobilis@xmpp' to='client@xmpp/MXA'>
  <item jid='mobilis@xmpp/Repository' />
  <item jid='mobilis@xmpp/Content' />
  <item jid='mobilis@xmpp/Coordinator' />
  <item jid='mobilis@xmpp/Buddy' />
</iq>
```

The mobile client will then discover the supported service namespaces of each broker service by querying the items of the Mobilis service namespace node:

```
<iq type='get' id='mobilis_2'
    from='client@xmpp/MXA' to='mobilis@xmpp/Repository'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='http://rn.inf.tu-dresden.de/mobilis#services' />
</iq>
```

The repository broker will answer with a list of items, each item mentioning one service namespace.

```
<iq type='get' id='mobilis_2'
    from='mobilis@xmpp/Repository' to='client@xmpp/MXA'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='http://rn.inf.tu-dresden.de/mobilis#services'>
    <item node='http://rn.inf.tu-dresden.de/mobilis#services/
        RepositoryService'
          name='ContentService' />
  </query>
</iq>
```

That way, the mobile client will look for a broker service supporting the correct namespace `.../RepositoryService`. If the discovery process yielded more than one repository broker service, a list will be shown to the end user to choose one repository broker.

The repository broker will discover the content broker in an analogue way (by discovering broker service supporting the `.../ContentService` namespace). Once it has found a content broker, it will register at the content broker to show the desire to manage contents of the store:

```
<iq type='set' id='mobilis_3'
    from='mobilis@xmpp/Repository'
    to='mobilis@xmpp/Content'>
  <content-register xmlns='http://rn.inf.tu-dresden.de/mobilis#services/
      ContentService' />
</iq>
```

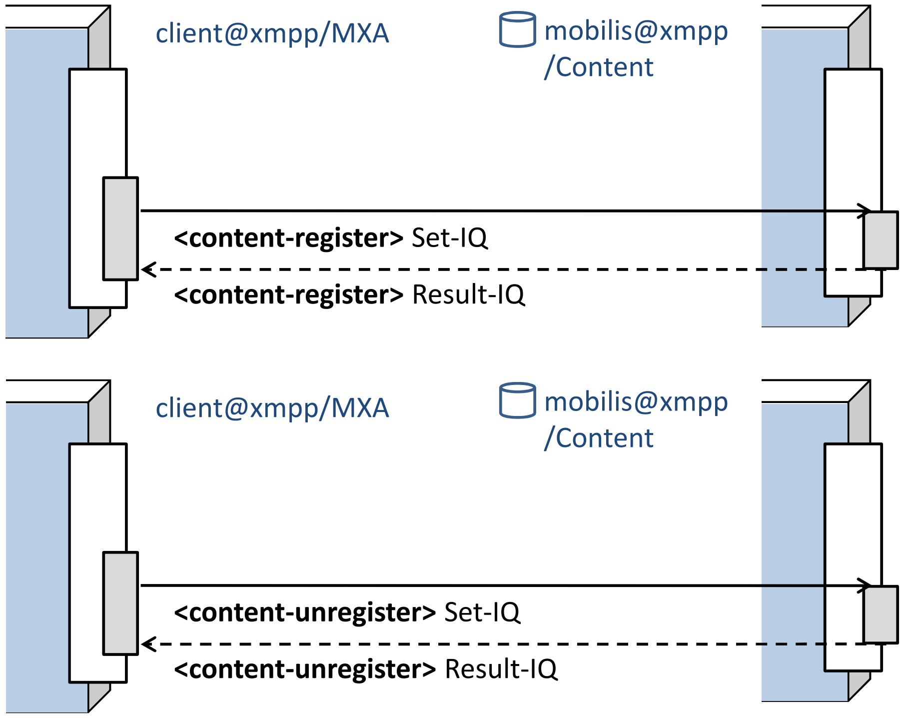

Figure 5.7.: Service registration and unregistration service primitive

The content service will confirm the registration with a respective Result-IQ:

```
<iq type='result' id='mobilis_3'
    from='mobilis@xmpp/Content'
    to='mobilis@xmpp/Repository'>
  <content-register xmlns='http://rn.inf.tu-dresden.de/mobilis#services/
     ContentService' />
</iq>
```

It may also send an Error-IQ instead, if the repository already was registered or, for whatever reason, it decides that the repository cannot register at the content broker service. Currently, there is no specific security mechanism developed, but future work can include such mechanisms in the registration process.

Before the repository broker service finishes its work, it will unregister from the content service:

```
<iq type='set' id='mobilis_4'
    from='mobilis@xmpp/Repository'
    to='mobilis@xmpp/Content'>
  <content-unregister xmlns='http://rn.inf.tu-dresden.de/mobilis#
     services/ContentService' />
</iq>
```

This will be acknowledged by an Result-IQ (or an Error-IQ, if unregistration is not possible, that is, if, e.g., the repository is not registered):

```
<iq type='result' id='mobilis_4'
    from='mobilis@xmpp/Repository'
```

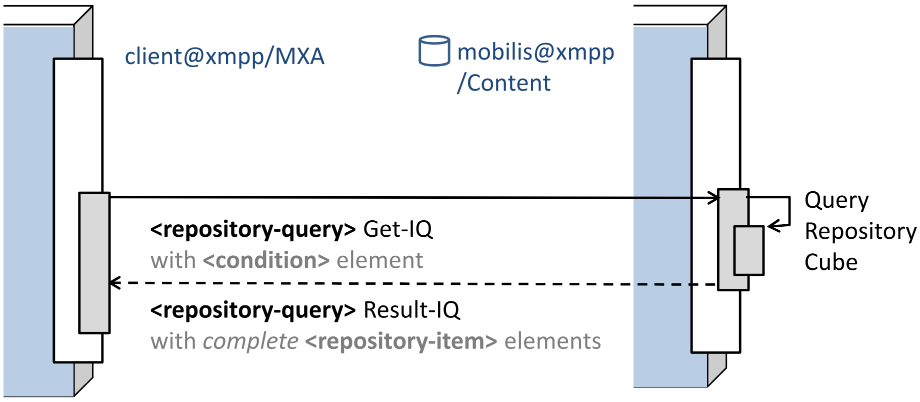

Figure 5.8.: Browsing service primitive

```
    to='mobilis@xmpp/Content'>
  <content-unregister xmlns='http://rn.inf.tu-dresden.de/mobilis#
     services/ContentService' />
</iq>
```

Figure 5.7 shows the sequence diagram of both registration and unregistration.

5.4.3. Browsing Service Pimitive

Figure 5.8 shows the sequence diagram for the browsing service primitive. Browsing the repository (FR-3) means requesting the metadata of all those repository items which slices match a certain condition (FR-3.4). The mobile client thus issues a `<repository-query/>` Get-IQ with a `<condition/>` element stating the filter conditions:

```
<iq type='get' id='mobilis_5'
    from='client@xmpp/MXA'
    to='mobilis@xmpp/Repository'>
  <repository-query xmlns='http://rn.inf.tu-dresden.de/mobilis#services/
     RepositoryService'>
    <condition op='and' xmlns='http://rn.inf.tu-dresden.de/mobilis'>
      <condition key='taken' op='lt' value='1256468400000' />
      <condition key='taken' op='gt' value='1256461200000' />
    </condition>
  </repository-query>
</iq>
```

The `<condition/>` element of the above example states that the date when the photo was taken (`taken`) should be before (`lt`) October 25 2009 12:00 (`1256468400000`) and (`and`) after (`gt`) October 25 2009 10:00 (`1256468400000`). The value of the date is represented as a unix-timestamp (milliseconds since 1970). While the mobile client is waiting for the result, the repository broker will query the repository cube according to the `<condition/>` element and finally return a IQ-Result with one `<repository-item/>` element for every matching item to the mobile client:

```
<iq type='get' id='mobilis_5'
    from='client@xmpp/MXA'
    to='mobilis@xmpp/Repository'>
```

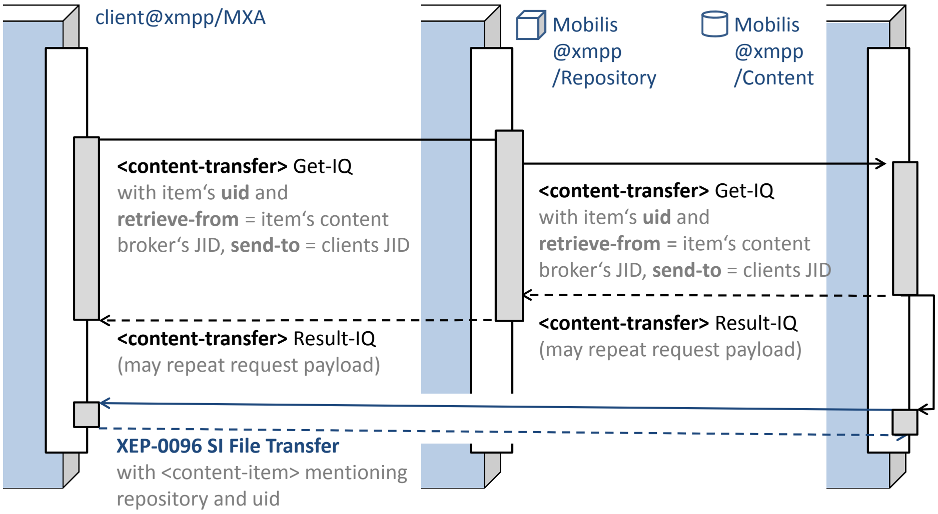

Figure 5.9.: Download service primitive

```
  <repository-query xmlns='http://rn.inf.tu-dresden.de/mobilis#services/
     RepositoryService'>
    <repository-item uid='8a808081246fc4e201246fc54f480002'
                     owner='client@xmpp/MXA'
                     content='mobilis@xmpp/Content'>
      <slice key='taken' value='1256464800000' />
      <slice key='longitude_e6' value='12105000' />
      <slice key='latitude_e6' value='47080000' />
      <slice key='owner' value='client@xmpp/MXA' />
      <slice key='title' value='Image.jpg' />
    </repository-item>
  </repository-query>
</iq>
```

The returned `<repository-item/>` elements are *complete*, that means, they mention all the data which is available about the item in the repository: the uid (`uid`), a reference to the content broker (`content`), the owner/creater (`owner`) as well as a sequence of all `<slice/>`s the item is assigned to. In our case, these are the dimensions time (`taken`) (FR-2.2), origin (`owner`) (FR-2.3), user defined name (`title`) (FR-2.4) and place (`longitude_e6`, `latitude_e6` – longitude and latitude multiplied with 10^6) (FR-2.1).

5.4.4. Download Service Pimitive

The downlaoad service primitive (illustrated in figure 5.9) aims to retrieve the actual content of a repository item from its content broker service. The mobile client will therefore send a `<content-transfer/>` Get-IQ to the repository service stating that it wants the content with the specified `uid` to be transferred from the content broker (`retrieve-from`) to itself (`send-to`):

```
<iq type='get' id='mobilis_6'
    from='client@xmpp/MXA'
    to='mobilis@xmpp/Repository'>
```

```
  <content-transfer xmlns='http://rn.inf.tu-dresden.de/mobilis#services/
      ContentService'>
    <uid>8a808081246fc4e201246fc54f480002</uid>
    <retrieve-from>content@xmpp/Content</retrieve-from>
    <send-to>client@xmpp/MXA</send-to>
  </content-transfer>
</iq>
```

Both <retrieve-from/> and <send-to/> are optional since they can be derived from the IQ sender and the content broker reference of the repository item.

The repository broker will forward this element to the content broker (eventually with additional retrieve-from and <send-to/> elements).

```
<iq type='get' id='mobilis_7'
    from='mobilis@xmpp/Repository'
    to='mobilis@xmpp/Content'>
  <content-transfer xmlns='http://rn.inf.tu-dresden.de/mobilis#services/
      ContentService'>
    <uid>8a808081246fc4e201246fc54f480002</uid>
    <retrieve-from>content@xmpp/Content</retrieve-from>
    <send-to>client@xmpp/MXA</send-to>
  </content-transfer>
</iq>
```

Finally, the content broker will initiate a SI file transfer of the actual content to the requester mentioned by <send-to/>. The file transfer request's <desc/> element will contain an XML coded <content-item/> element referencing repository and uid, so the file transfer can be assigned to the correct request on client side.

```
<iq type='set' id='mobilis_8'
    from='client@xmpp/Repository'
    to='mobilis@xmpp/Content'>
  <si xmlns='http://jabber.org/protocol/si'
      id='68081925' mime-type='binary/octet-stream'
      profile='http://jabber.org/protocol/si/profile/file-transfer'>
    <file xmlns='http://jabber.org/protocol/si/profile/file-transfer'
        name='Image.jpg' size='9170'>
      <desc>
&lt;content-item xmlns='http://rn.inf.tu-dresden.de/mobilis#Services/
   ContentService'&gt;
&lt;repository&gt;mobilis@content/Repository&lt;/repository&gt;
&lt;uid&gt;8a808081246fc4e201246fc54f480002&lt;/uid&gt;
&lt;description&gt;the actual description&lt;/description&gt;
&lt;/content-item&gt;
      </desc>
    </file>
    <feature xmlns='http://jabber.org/protocol/feature-neg'>
      <x xmlns='jabber:x:data' type='form'>
        <field var='stream-method' type='list-single'>
          <option><value>
            http://jabber.org/protocol/bytestreams
          </value></option>
          <option><value>
            http://jabber.org/protocol/ibb
          </value></option>
        </field>
```

```
      </x>
    </feature>
  </si>
</iq>
```

In parallel to the initiation of the file transfer, a Result-IQ is sent back to the repository broker:

```
<iq type='result' id='mobilis_7'
    to='mobilis@xmpp/Content'
    from='mobilis@xmpp/Repository'>
  <content-transfer xmlns='http://rn.inf.tu-dresden.de/mobilis#services/
      ContentService' />
</iq>
```

... and from there to the mobile client:

```
<iq type='result' id='mobilis_6'
    to='mobilis@xmpp/Repository'
    from='client@xmpp/MXA'>
  <content-transfer xmlns='http://rn.inf.tu-dresden.de/mobilis#services/
      ContentService' />
</iq>
```

If an error occured, Error-IQs are used instead of Result-IQs. An error occurs, if the content item cannot be found on the content server, if a file transfer cannot be established or if the request reaching the content broker was not sent by a registered repository broker. For future implementation, also more enhanced security mechanisms (on the side of the repository broker) concerning the request of content items are possible (FR-5.2).

5.4.5. Upload/Replacing Service Primitive

Uploading new items into the repository or replacing existing items in the repository are the most complex service primitives. Uploading/Replacing happens in two stages. First, the item is created at the repository broker but the actual content stays on the mobile client. In the second stage, the repository broker may require an assigned content broker to request handover of the content from the mobile client to the content broker. The content will finally be transferred by an SI file transfer from the mobile client to the content broker after which the content broker informs the repository broker, that the content now is stored at another location. The repository broker will thus update its repository cube to hold the new reference to the content broker. The whole process is illustrated in the sequence diagram of figure 5.10.

When uploading an item, the mobile client will first send a `<repository-query/>` Set-IQ to the repository broker.

```
<iq type='set' id='mobilis_9'
    from='client@xmpp/MXA'
    to='mobilis@xmpp/Repository'>
  <repository-query xmlns='http://rn.inf.tu-dresden.de/mobilis#services/
      RepositoryService'>
    <repository-item>
      <slice key='taken' value='1256464800000' />
      <slice key='longitude_e6' value='12105000' />
      <slice key='latitude_e6' value='47080000' />
```

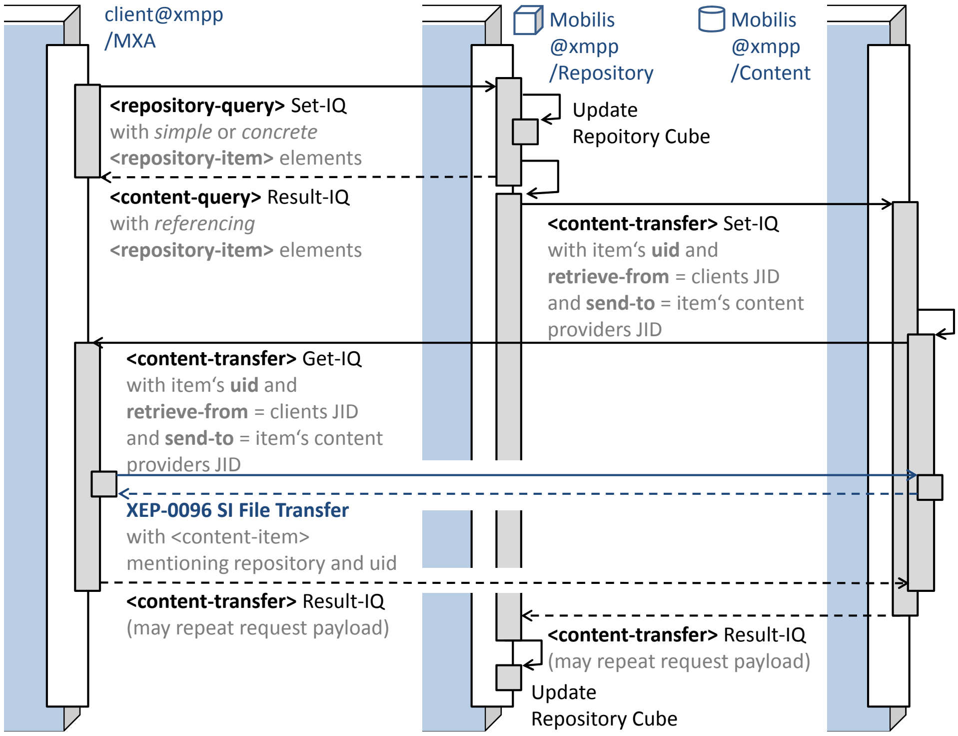

Figure 5.10.: Upload / Replacing service primitive

```
      <slice key='title' value='Image.jpg' />
    </repository-item>
  </repository-query>
</iq>
```

It contains one or more *simple* or *concrete* `<repository-item/>` elements representing the items which should be added to or replaced in the repository. A *simple* item, as shown in the above example, contains `<slice/>` elements only and creates a new repository-item inside the repository. A *concrete* item has an additional `uid` attribute which references an already existing item, which should be replaced.

The repository broker will insert the item with its slices into the repository cube or update the existing items depending on the request. It will then send back a Result-IQ to confirm the operation:

```
<iq type='set' id='mobilis_9'
    from='mobilis@xmpp/Repository'
    to='client@xmpp/MXA'>
  <repository-query xmlns='http://rn.inf.tu-dresden.de/mobilis#services/
     RepositoryService'>
    <repository-item uid='8a808081246fc4e201246fc54e480205'
                     content='mobilis@xmpp/Content' />
  </repository-query>
</iq>
```

The Result-IQ will contain a *referencing* repository-item for every inserted or updated repository item. The referencing repository-item mentions only the `uid` of the item and which `content` broker may be expected to ask for handover of the content. Instead of a

Result-IQ, an Error-IQ may be sent, if any error occurs while storing the repository item into the repository cube. This is the case when an item, which should be replaced, cannot be found, is not owned by the requester or if a database error occurs.

Note, that the actual content item is still at the mobile client side. That means, that any other mobile client, that requests the content file, will contact the client (FR-1.4). That means, the client should implement all capabilities of a content broker. After publishing an item to the repository broker, the client should be ready to transfer the content to any third party, which requests it, until the content is finally handed over to a content broker at some point in the future. The content broker, which will request this handover is mentioned in the `content` attribute.

To initiate the handover, the repository broker will at some point in the future notify a content broker to ask the mobile client to handover the content file. Therefore, a `<content-transfer/>` Set-IQ is sent from the repository broker to the content broker:

```
<iq type='set' id='mobilis_10'
    from='mobilis@xmpp/Repository'
    to='content@xmpp/Repository'>
  <content-transfer xmlns='http://rn.inf.tu-dresden.de/mobilis#services/
      ContentTransfer'>
    <uid>8a808081246fc4e201246fc54e480205</uid>
    <retrieve-from>client@xmpp/MXA</retrieve-from>
    <send-to>mobilis@xmpp/Repository</send-to>
  </content-transfer>
</iq>
```

The content broker then sends a `<content-transfer/>` Get-IQ to the mobile client to request handover of the content item:

```
<iq type='get' id='mobilis_11'
    from='content@xmpp/Repository'
    to='client@xmpp/MXA'>
  <content-transfer xmlns='http://rn.inf.tu-dresden.de/mobilis#services/
      ContentTransfer'>
    <uid>8a808081246fc4e201246fc54e480205</uid>
    <retrieve-from>client@xmpp/MXA</retrieve-from>
    <send-to>mobilis@xmpp/Repository</send-to>
  </content-transfer>
</iq>
```

The actual handover is then carried out by a SI file transfer similar to that of the download service primitive (see 5.4.4). Afterwards, the client confirms the request by an Result-IQ to the content broker:

```
<iq type='result' id='mobilis_11'
    from='client@xmpp/MXA'
    to='mobilis@xmpp/Content'>
  <content-transfer xmlns='http://rn.inf.tu-dresden.de/mobilis#services/
      ContentTransfer' />
</iq>
```

This will trigger another Result-IQ sent back from the content broker to the repository broker who will upon reception of this IQ know, that the location where the content has been stored has changed. It will therefore update the repository cube to hold the new content broker reference.

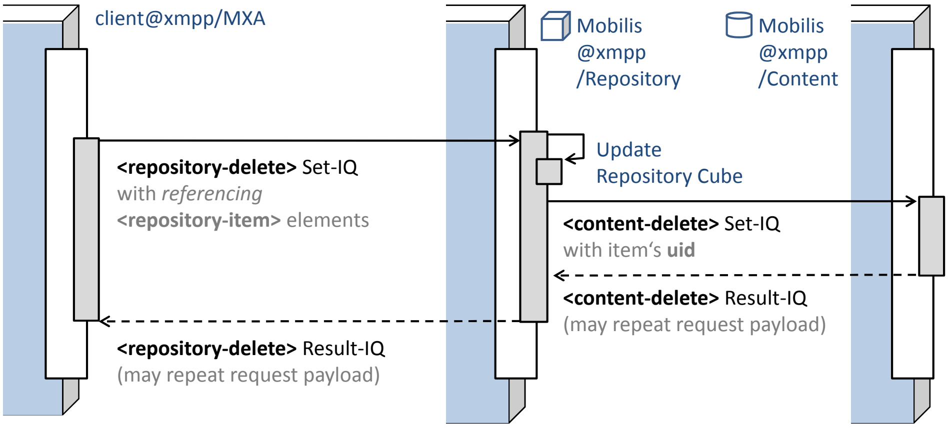

Figure 5.11.: Deletion service primitive

```
<iq type='result' id='mobilis_11'
    from='mobilis@xmpp/Content'
    to='mobilis@xmpp/Repository'>
  <content-transfer xmlns='http://rn.inf.tu-dresden.de/mobilis#services/
     ContentTransfer' />
</iq>
```

5.4.6. Deletion Service Primitive

Deleting an item from the repository is depicted in figure 5.11. The mobile requests deletion using a `<repository-delete/>` Set-IQ referencing all the items it wants to delete in `<repository-item/>` elements:

```
<iq type='set' id='mobilis_12'
    from='client@xmpp/MXA'
    to='mobilis@xmpp/Repository'>
  <repository-delete xmlns='http://rn.inf.tu-dresden.de/mobilis#services
     /RepositoryService'>
    <repository-item uid='8a808081246fc4e201246fc54e480205' />
  </repository-delete>
</iq>
```

If the item exists and if the mobile client owns the item (FR-5.2), the repository broker will remove the repository item from the repository cube and send a `<content-delete/>` Set-IQ to the associated content broker:

```
<iq type='set' id='mobilis_13'
    from='mobilis@xmpp/Repository'
    to='mobilis@xmpp/Content'>
  <content-delete xmlns='http://rn.inf.tu-dresden.de/mobilis#services/
     ContentService'>
    <uid>8a808081246fc4e201246fc54e480205</uid>
  </repository-delete>
</iq>
```

The content broker will remove the content from its content store and send back `<content-delete/>` a Result-IQ to the repository broker:

```
<iq type='set' id='mobilis_13'
    from='mobilis@xmpp/Content'
    to='mobilis@xmpp/Repository'>
  <content-delete xmlns='http://rn.inf.tu-dresden.de/mobilis#services/
      ContentService' />
</iq>
```

... which will confirm the deletion by sending back a `<repository-delete/>` Result-IQ to the mobile client:

```
<iq type='result' id='mobilis_12'
    to='client@xmpp/MXA'
    from='mobilis@xmpp/Repository'>
  <repository-delete xmlns='http://rn.inf.tu-dresden.de/mobilis#services
      /RepositoryTransfer' />
</iq>
```

5.5. Conclusion

This chapter introduced an architecture for a XMPP-based metadata-structured repository. The designed repository is general enough to manage all kinds of media (FR-1.2) which includes image files (FR-1.1) with arbitrary metadata classification (FR-2.5), which includes the classification location, time, ownership and title (FR-2.1 through FR-2.5). Using the elaborated architecture, both centralized server scenarios (FR-1.3) as well as hybrid scenarios (FR-1.4) are possible. Also user-to-user sharing is possible using the building block of SI File Transfers (FR-1.5).

Service primitives have been introduced to allow interaction with the repository items (FR-4) and security mechanisms have been considered where applicable (FR-5.1 and FR-5.2). Encryption (FR-5.3) has not been covered but can be achieved by higher layer protocols. The architecture has been described to fit into the mobilis platform (FR-6.2).

Hence, most functional requirements can be covered by the current architecture. What is left is the implementation of a prototype which actually uses the described architecture and XMPP extension for the presented picture sharing scenario. Also, non-functional requirements have to be considered during implementation. The next chapter will concentrate on this topic.

6. Implementation Considerations

Having defined the architecture of a general repository system in the previous, this chapter concentrates on the implementation of a prototype to proof the concept of the architecture as well as realize the picture sharing usage scenario introduced in the beginning. First, we introduce the layered implementation of both Mobilis client and server (6.1) and then point out, how the same code can be reused on both client and server side (6.2). We proceed by explaining how the Mobilis Media prototype fills extension points of client and server (6.3) and finally highlight both system boundaries and internal structure of the server (6.4) and the client (6.5) prototype. Section 6.6 concludes this chapter.

6.1. The Mobilis Architecture

The general architecture of the Mobilis platform is shown in 6.1. Mobilis media, the mobilis media sharing repository platform, has to be included into this architecture (FR-6.2 and FR-6.3). In the Mobilis environment, multiple (mobile) Mobilis clients communicate with a Mobilis server using an XMPP server as communication relay. The Mobilis client and server applications are realized in Java (DC-4.2). The server application runs on a **JDK 1.6.0_10** runtime environment.

On client side, multiple Mobilis **applications** may be installed independently, each one using another set of broker services of the Mobilis Server application. Such a set of broker services used by one client application is referred to as a **package**. The package on the server side and the application on the client side together make up a **project**.

All client applications run on the **Android SDK 1.6**, that is, as Java applications on the Android operating system inside a special Dalvik virtual machine optimized for mobile environments. Client applications make also heavily use of Android SDK framework classes for the purpose of retrieving content or contextual information or for displaying a GUI to the user.

Both client and server use the **Smack 1.3** library to communicate via XMPP. To avoid deployment of the (quite heavyweight) Smack library in every client application, the Smack library is encapsulated into one central application: the **MXA application** developed in parallel by István Koren, also a member of the Mobilis team. Overlaying applications can bind by means of interprocess communication to the MXA and use services of the MXA to send and receive packets over XMPP using Smack. The MXA consists of one central **XMPPService**, used to exchange core XMPP packets, like IQs and messages, and, on top of that, several **extension services**, which realize XMPP Extensions like file transfer, service discovery etc.

The logic representation of the messages exchanged between client and server – the custom IQs – are incorporated into a lightweight **XMPPBean** layer which can be used both on Mobilis server and on Mobilis client side. There is one set of XMPPBeans for every Mobilis project. Since the interface to the MXA on the server side is different from

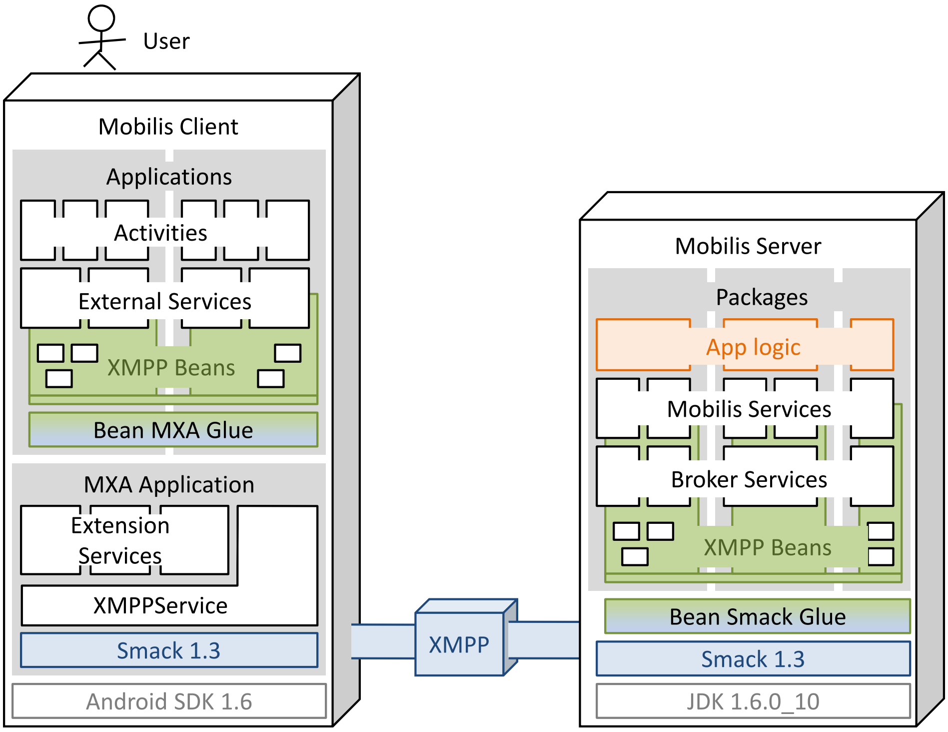

Figure 6.1.: The inner architecture of the Mobilis platform

the interface to Smack on the client side, two glue layers are required (**Bean MXA glue** and **Bean Smack glue**) to serialize and deserialize XMPPBeans to match the MXA or Smack interface respectively.

On the server side, on top of the XMPPBean layer, **broker services** reside. A Broker Service is an entity which manages a single XMPP connection of the Mobilis Server and has a set of **Mobilis services** assigned to it. It can receive a set of IQs and knows to which Mobilis service these IQs should be forwarded to. These overlying Mobilis Services finally interprete those IQs, test requests on validity and execute respective actions in the Server's **app logic**. The Mobilis Services will also decide upon the requests answer sent over the broker service back to the requester.

On the client side, on top of the XMPPBean layer, so called **external services** are running. External services are services which make use of MXA functionality but realize high-level non-standard XMPP protocols. They are implemented as Android Service classes, that means, they run in the background of the Android operating system and other processes can connect to them by means of interprocess communication to request certain actions. External services do not have a GUI – this final layer is provided by **activity** layer. Activities in sense of Android are a screen where a user tries to accomplish a certain task. Each activity binds to one or more external services and executes certain commands of these services upon request by the user.

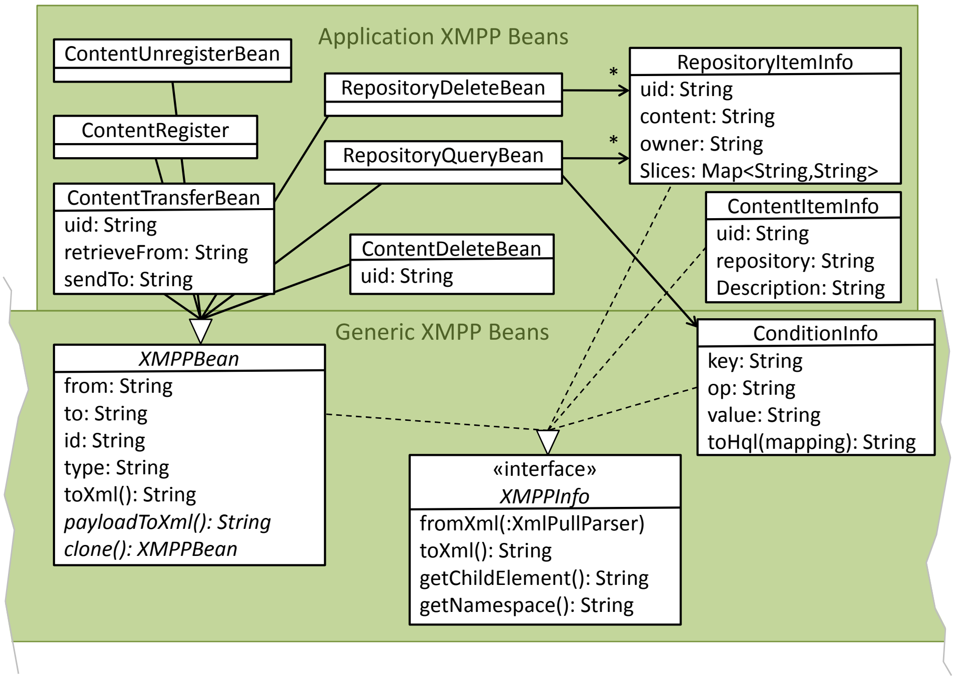

Figure 6.2.: XMPP Beans as a representation for IQ packets and XML snippets

6.2. Reuse of the XMPP layer using XMPPBeans

A special role in the Mobilis architecture is given to the the layer of the so-called XMPP-Beans, which represent XMPP IQs on class level. This layer is the only one which is re-used in client and server, which provides both easy maintainablility and on the other hand assures constraints concerning the syntax of exchanged IQs on both client and and server. The XMPPBean layer of the Mobilis Media project is shown in figure 6.2.

The XMPPBean layer provides some basic classes, which are used in every Mobilis project, like the class `XMPPBean` representing an XMPP IQ or the more general interface `XMPPInfo` representing a snippet of XML. `XMPPInfo` provides methods for unserializing (`fromXml(...)`) and serializing (`toXml()`). It also provides methods to get the root element of the carried XML (`getChildElement()`) and the corresponding namespace (`getNamespace()`). An `XMPPBean` object supports additionally information carried by an XMPP IQ like `from`, `to`, `id` etc.

For every custom IQ, there will be one subclass of XMPPBean. Hence, every Mobilis project has its own concrete XMPP Bean subclasses and this set of subclasses is linked into the Android application and into the Server package. In the example of Mobilis Media, the `<content-.../>` and `<repository-.../>` IQs are represented by respective `Content...Bean` and `Repository...Bean` classes. All those classes implement methods for serialization and deserialization as well as the methods to retrieve the class-specific namespace and child element. Sub elements may be included into the bean by aggregating other concrete XMPPInfo classes.

An `XMPPBean` object can neither be understood by Smack nor by MXA. Therefore, it has to be converted to the representation used on the respective system. This is done by a `bean glue layers`. The pattern of this process is shown in figure 6.3.

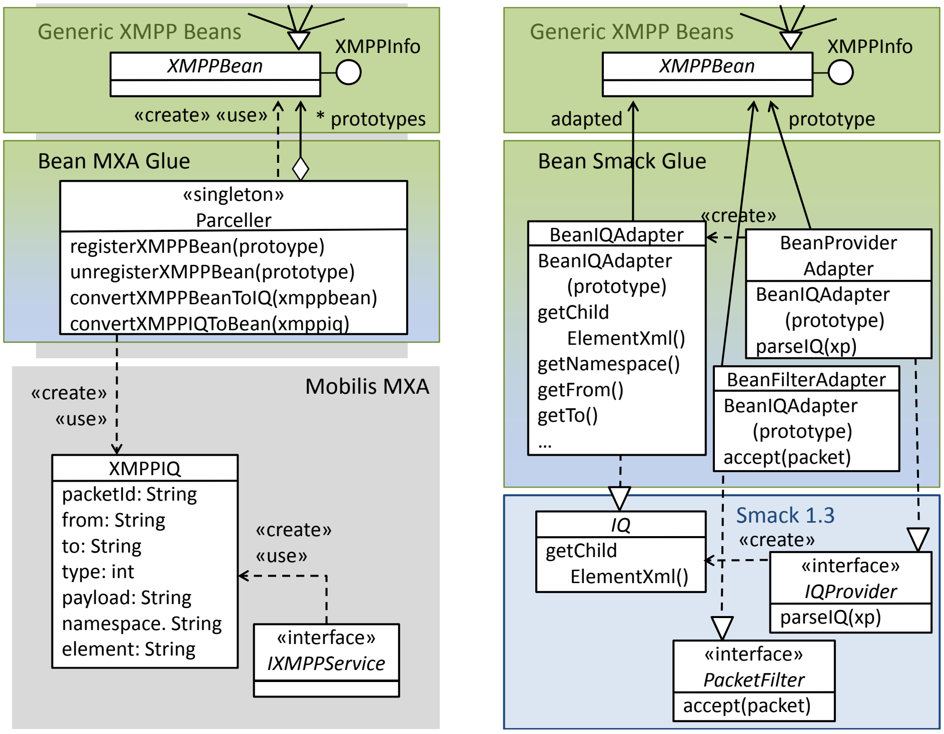

Figure 6.3.: Translation of XMPP Beans to the Smack or MXA layer

In case of MXA, the concrete XMPPBean must be converted to a XMPPIQ class, which holds the XML payload of the bean as plaintext. Converting XMPPIQ classes to XMPPBeans and vice versa is done by an entity called the `Parceller`. Converting to XMPPIQ is done straightforward by deserializing the bean. However, converting an incoming XMPPIQ to a bean is more difficult: the `namespace` and `element` pair has to be resolved to a concrete `XMPPBean` implementation first. To achieve this, the `Parceller` holds a list of `prototypes`, which it may scan for the correct subclass, then invoke the `clone()` method upon this prototype and perform unserialization upon this copy.

On the server side the responsibility is spread to three classes: Since Smack represents IQs as subclasses of `IQ`, a `BeanIQAdapter` simply wraps an XMPPBean and adapts the IQ interface to the XMPPBean interface. Deserialization in Smack is done by implementations of the `IQProvider` interface. Here, a prototype pattern is used again to create a respective `BeanIQAdapter` for a `XMPPBean` with characteristic namespace and child element. The third class, the `BeanFilterAdapter`, is an implementation of the Smack interface `PacketFilter` which may be used to filter incoming packets to match the namespace / child element combination of a specific XMPPBean.

6.3. Mobilis Media as a Mobilis Project

The Mobilis Media project prototype is included into the Mobilis platform as one package on server side and one sample Android application on client side. Figure 6.4 shows how the project fits into the architecture. The server package implements all the functionality introduced for the repository architecture in chapter “Conceptual Design” (5). The sample

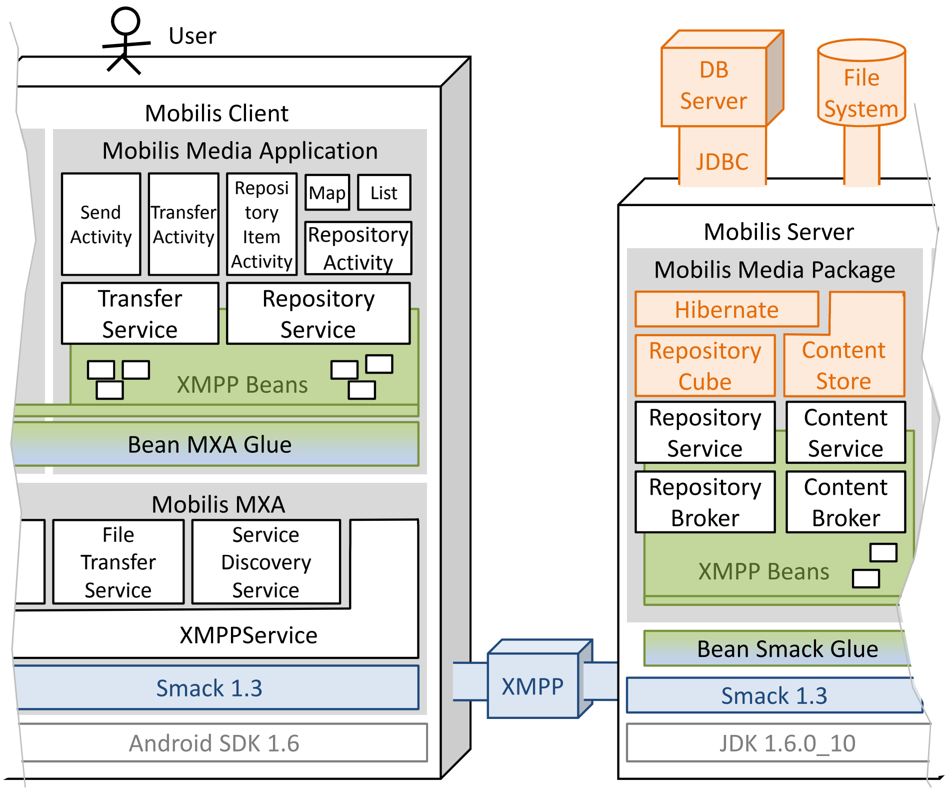

Figure 6.4.: The inner architecture of Mobilis Media as a Mobilis project

Android application provides functionality to browse an image repository using a map (FR-3.1) and a filtered list (FR-3.2) as well as an user interface to upload new items to the repository. As a side-effect, this user interface also allows one-to-one file transfer based on the XMPP extension for SI File Transfer (FR-1.5).

6.3.1. Server Prototype

The server prototype includes a **repository broker** and a **content broker**, as introduced in chapter 5. Connected to those broker services are respective Mobilis services, which interprete incoming `<repository-.../>` and `<content-.../>` IQs, perform security and other checks and finally manipulate the overlying **application logic** according to the requests.

The application logic is more precisely a **Repository Cube** and a **Content Store** respectively. Both entities make use of a database to persist the modeled data. For this purpose, the **Hibernate framework** [1] is used. The file contents, however, are stored by the content store to the file system using JDK core functionality.

6.3.2. Client Prototype

The client prototype introduces two new external services: The **transfer service** allows one-to-one file transfers and transfers to a cube repository (upload service primitive, see

[1] `https://www.hibernate.org/`

subsection 5.4.5). To accomplish the first, it makes use of the file transfer service and the XMPPService, both offered by the MXA. It also adds another layer around the MXA File Transfer Service by displaying notifications about ongoing transfers in the Android notification manager. The transfer service will be more detailed in subsection 6.5.2.

The second service, called **repository service**, provides an interface to communicate with the repository broker through all other service primitives despite of upload, that is, repository service discovery, browsing of the repository, download and deletion of repository items. The repository service uses the service discovery service and the XMPPService of the MXA and will be detailed in subsection 6.5.3.

Both services are used by overlying activities, which form the GUI layer. However, it should be noted that any other Android application may register to the services and use their interface by means of interprocess communication.

The activities which form the user interface are shown in figure 6.5. The **send activity** is used if the user chooses to send a single image either to another XMPP user or to a repository. In this activity, the user may choose the destination and enter a description for the transfer. The **transfer activity** shows all ongoing transfers, their origin or destination, the file and the progress of the transfer. Furthermore, a user can accept or deny incoming transfers by interacting with this window. Both send activity and transfer activity make use of the transfer service.

The **repository activity** shows the contents of a repository selected by the user. Therefore, it embeds either a **repository list activity** or a **repository map activity**. In those activities, the user may select a repository item, what would invoke the **Repository Item Activity**. That activity shows information about the repository item and commands which may be invoked upon them, like replacing, downloading and deletion. All repository activities make use of the repository service.

6.4. The Mobilis Media Server Prototype

The Mobilis server prototype has already undergone vast development in previous Mobilis projects and therefore has reached a high software majurity concerning the integration of new brokers and services. Currently, it includes broker service for a mobile tourist guide [Kor08a], a buddy finder [HFV$^+$] and collaborative editing [Her09]. This section describes, how broker services and mobilis service can be integrated on class level (6.4.1) and how this is done at Mobilis Media in particular (6.4.2). Finally the realization of the repository cube data model on database level is introduced (6.4.3).

6.4.1. General Mobilis Server Class Model

Figure 6.6 shows the core classes to implement service brokers (`MobilisBroker` class) and Mobilis services (`MobilisService` class). Instances of both classes are configured dynamically via a configuration file (`MobilisSettings.xml`) and managed by the `MobilisManager` singleton. The basic task of every `MobilisBroker` object is to maintain a XMPP connection (`XMPPConnection` class) and hold references to concrete `MobilisService` objects.

The `MobilisService` class is abstract and will be subclassed by every concrete Mobilis service. In the abstract method `registerPacketListener()`, every concrete service

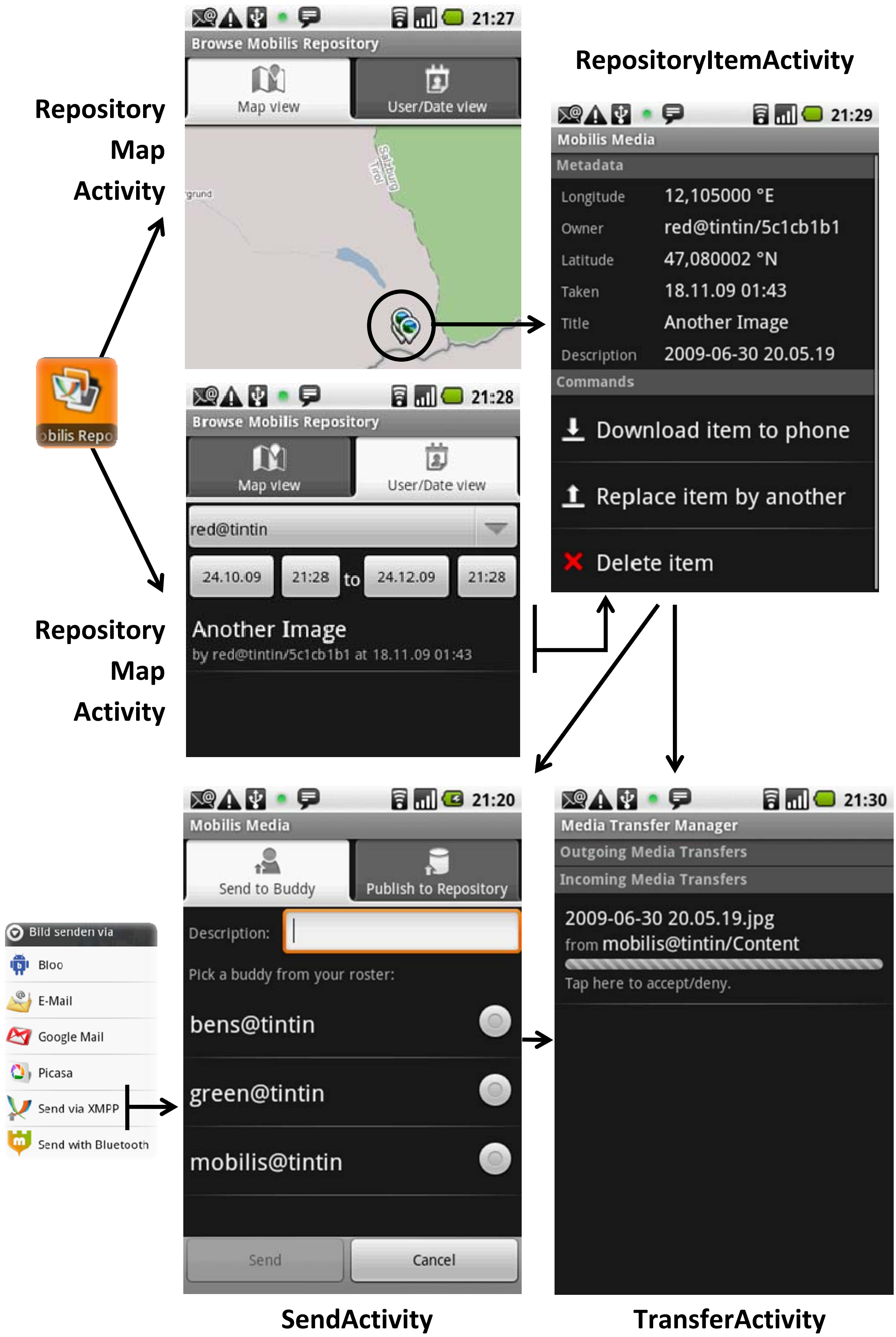

Figure 6.5.: Activities forming the User Interface of the Mobilis Media Android Application

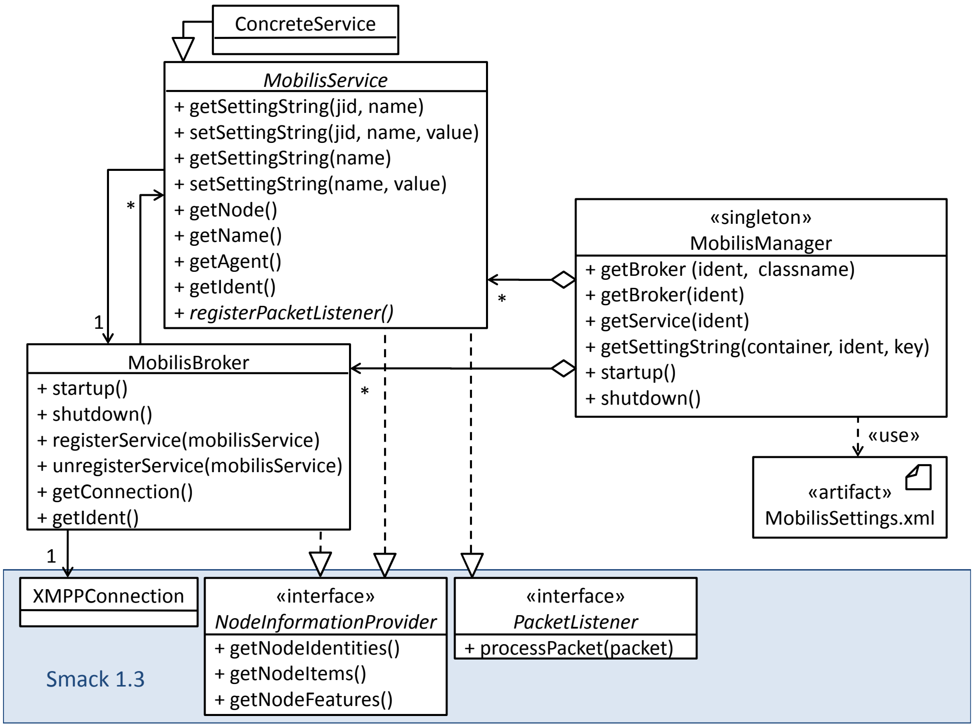

Figure 6.6.: Mobilis Server Classes for Service Brokers and Mobilis Services

registers its `PacketListener` interface to receive a certain set of IQ messages arriving at the XMPP connection held by the assigned `MobilisBroker`. Both `MobilisServices` and `MobilisBrokers` implement the interface `NodeInformationProvider` to allow the client to query the service broker for connected Mobilis services via XMPP Service Discovery [HMESA08].

6.4.2. Mobilis Media Server Class Model

Figure 6.7 shows how Mobilis Media is integrated into the Mobilis Server on class level. The content broker and the repository broker are represented as instances of the `MobilisBroker` class. They hold a reference to an instance of the concrete `MobilisService` classes `ContentService` and `MobilisService` respectively. These classes define various method which are called upon reception of related IQs (`in...(bean)`) or if IQs should be sent out from the service to another XMPP entity (`out...()`).

The backing application logic consists of the two classes `ContentStore` and `RepositoryCube` which provide an interface to arbitrarily manipulate the data structures laying behind both entities. The backing data model is stored both on the file system (in a folder `store/ContentService`) and in a database, managed by the Hibernate framework [2], in detail, a `org::hibernate::Session` instance.

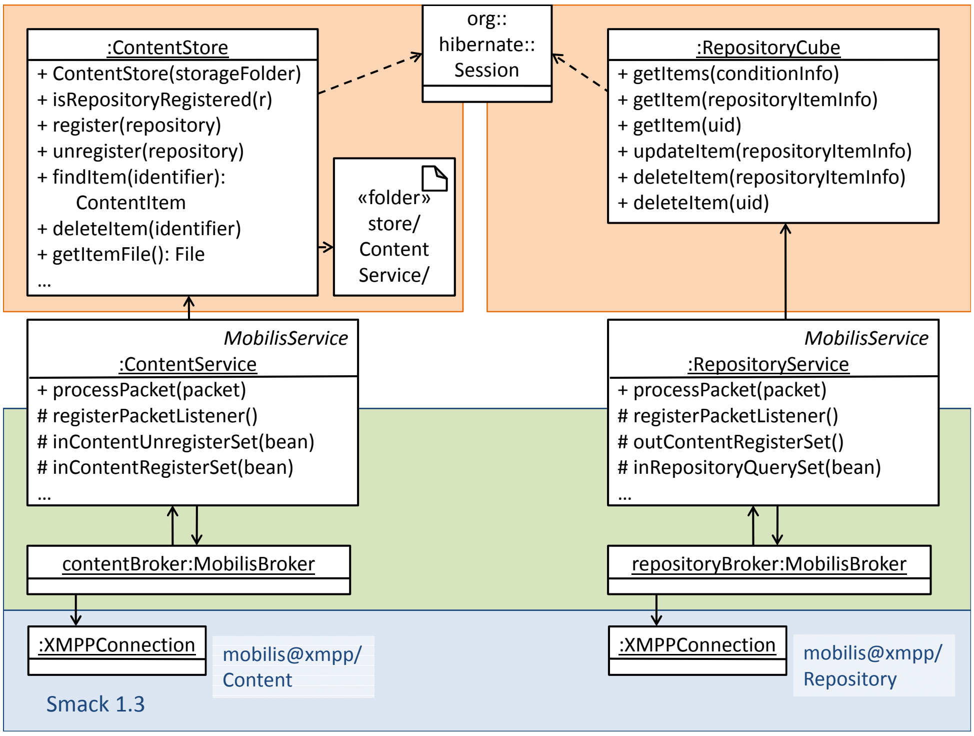

Figure 6.7.: Mobilis Media Server Core Classes

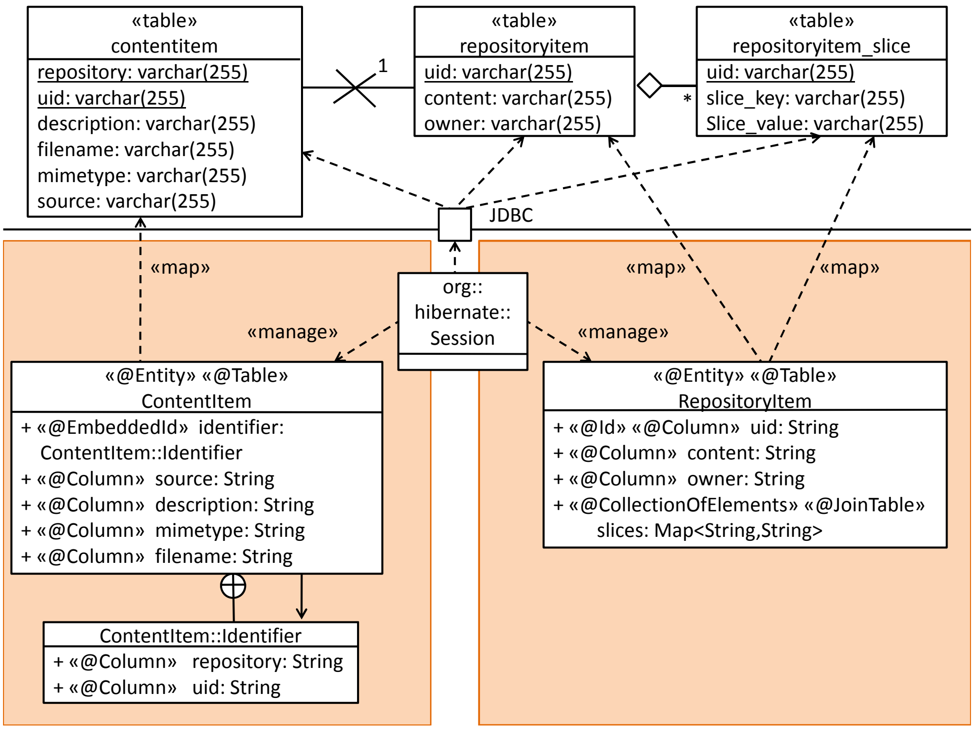

Figure 6.8.: Mobilis Media Server Database Model for the Repository Cube Data Model

6.4.3. Mobilis Media Database Model

The data model described in chapter "Conceptual Design" (section 5.2) is realized using a relational database. Using the Hibernate framework the relational records are mapped to instances of classes of the server prototype. Figure 6.8 shows both the relational model as well as its mapping to the class model using Hibernate. The object-relational mapping is declared on Java sourcecode level by Java annotations assigned to the respective Java types and their members. Those annotations specify the related relational entity, that is, which elements are tables, primary keys etc.

Once an object of such an annotated type is handed over to a the Hibernate session by calling `session.save(object)`, the object becomes managed by Hibernate – any change to the object is henceforth written to the database (possibly with a certain buffered delay). In similar ways, objects may be read out from the database using the session. Hibernate uses the JDBC (Java Database Connectivity) interface which can connect to a large amount of different database systems. In our prototype implementation, we connect to a MySQL 5.1 database [3] using the Java MySQL Connector 5.1.10 [4].

In Mobilis Media, there are three tables stored in the database: `contentitem` which represents instances of the `ContentItem` class. Those records/objects represent items in the `ContentStore`, which is managable by the `ContentService`. The other two tables `repositoryitem` and `repositoryitem_slice` represent instances of the `RepositoryItem` class and it's slice assignment. `RepositoryItem` objects are stored in the `RepositoryCube` which can be managed by the `RepositoryService`. The slice assignment of repository items is represented as a simple `String-to-String-Map` on class level. Both `repositoryitem` and `contentitem` are completely independend from each other on database and on class level. The link between both items is built on semantics of the more abstract service broker layer.

6.5. The Mobilis Media Client Prototype

This section describes some system boundaries and internal structures of the client prototype. We start by introducing some fundamentals of interprocess communication on Android in subsection 6.5.1 since interprocess communication was extensively used in the prototype to allow reusability of the prototype (NF-1.6). One manner of interprocess communication is the use of Android services, or, in terms of the Mobilis platform, external services. The two Mobilis Media external services – `TransferService` and `RepositoryService` – are introduced in subsections 6.5.2 and 6.5.3. Subsection 6.5.4 gives a brief overview of overlying GUI layer.

6.5.1. Interprocess Communication on Android

A process of the Android framework is always running inside a `Context`. This context can be either an `Activity` or a `Service`. An activity is a piece of GUI where the user accomplishes a certain task. An (Android) service may be started independently and

[2] https://www.hibernate.org/
[3] http://dev.mysql.com/downloads/mysql/5.1.html
[4] http://www.mysql.com/products/connector/

perform background tasks. It is also possible for a service to offer an interface where another context can connect to. Services and activities have a lifecycle managed by the android system, i.e. a service runs until a task is complete and it stops itself or until a consumer unbinds from its interface. An activity normally runs until it is hidden by the user. This lifecycle mechanism allows the Android operating system to manage memory efficiently, that is, to kill processes automatically, where the lifecycle of services and activities has ended. More information about process management on android can be found in [weba].

An easy mean of interprocess communication are `Intents`. An intent is an request to the system to execute a specific action and it may either start another activity, start a service or bind to a service interface. An `Intent` object may be sent from any context object to the system by calling `context.startActivity(intent)`, `context.startService(intent)` or `context.bindToService(intent)`. The system will then find the activity or service which is responsible for handling that intent, start it and send the intent to the respective context. This is possible because all Android applications declare in their manifest which intents they can handle. (There are also more ways to handle intents, for more information, see [webd]). If more than one applications is found capable, the user is requested to choose one application.

When a context binds to a service's interface, it will receive an `IBinder` object which represents a remote interface, in this case, the remote interface of the service thread. Using an Android own interface description language named AIDL (Android Interface Description Language) this `IBinder` may be typecasted to the concrete remote interface and thereafter be be used to issue remote calls to the service. More information about AIDL can be found at [webc].

AIDL generated interfaces only accept parameters and return types which implement the `Parcelable` interface. Framework classes which are "parcelable" are for example `Intent`, `IBinder`, `Message` (a class containing fields like `what`, `when` and a hash map of arbitrary other `Parcelables`) or `Messanger` but the programmer is free to make any class parcelable that she wishes to.

A `Messenger` is a target for `Messages`. In the owning thread, it is coupled to a `Handler`. Once sent to another process, this process can invoke `messenger.sendMessage(Message)` what will enqueue the message in a message queue at the destination thread and call `handler.handleMessage(Message)` when the thread is ready. Such a message queue, however, must be tied to a `Looper`. This means, a thread using a `Handler` must run in an infinite loop to check for new messages. This loop can be entered by calling `Looper.prepare()` before creating the `Handler` object and finally calling `Looper.loop()` to start enter the infinite message loop.

6.5.2. External Service: TransferService

Service Boundaries

There are two ways to interact with the TransferService: The first is by starting it to execute a file transfer – either to another XMPP entity or to a repository. This is done by sending an `Intent` with the action `de.tudresden.inf.rn.mobilis.media.intent.SEND_TO_JID` or `....SEND_TO_REP` in the following way:

```
Intent i = new Intent("de.tudresden.inf.rn.mobilis.media.intent.
    SEND_TO_REP");
// has to be provided always - also if SEND_TO_JID is used.
i.putExtra("STR_TO", "mobilis@xmpp/Repository");
i.putExtra("STR_DESCRIPTION", "Any␣description");
i.putExtra("STRA_PATHS", new String[] { "path/to/file/1", "path/to/file
    /2" });
// the following lines only if the file is send to a Repository
Bundle b = new Bundle[2];
b[0] = new Bundle();
b[0].putString("file1_slice1", "value_for_file1_slice1");
b[0].putString("file1_slice2", "value_for_file1_slice2");
b[1] = new Bundle();
b[1].putString("file2_slice1", "value_for_file2_slice1");
b[1].putString("file2_slice2", "value_for_file2_slice2");
i.putExtra("BDLA_SLICES", b);
// this line only if other items in the repository should be replaced.
i.putExtra("STRA_REPOSITORYITEMS_UIDS", new String[] { "uidforfile1", "
    uidforfile2" });
// send the intent
context.startService(i);
```

The string extras `STR_TO` and `STR_DESCRIPTION` name the recieptient and the description used during file transfer. `STRA_PATHS` is an array of paths where the files can be found on the mobile device's file system. `BDLA_SLICES` contains the desired slice assignment for every file and `STRA_REPOSITORYITEMS_UID` the UIDs of the repository items which should be replaced. If the last extra is not present or any of the string array contents is set to null, the repository item will be newly created and not replace another.

The second way to interact with the TransferService is to bind to its AIDL interface. Therefore a `de.tudresden.inf.rn.mobilis.media.services.ITransferService` intent has to be sent:

```
Intent i = new Intent("de.tudresden.inf.rn.mobilis.media.services.
    ITransferService");
context.bindService(i);
```

The `IBinder` which is sent back to the context's `onBind(...)` method once the service is bound provides the following remote interface:

int startTransferToJid(FileTransfer) initiates a file transfer to another XMPP entity. The properties of the file transfer are described in the given FileTransfer object. The service will return an unique ID of the transfer.

int startTransferToRep(String, RepositoryItemParcel, FileTransfer) initiates a file transfer to a Cube Repository. The arguments name the repository, the desired properties of the repository item and the desired properties of the file transfer. An unique ID for this file transfer will be returned.

void registerMediaTransferMessenger(Messenger, int) registers a `Messenger` to be informed when the state of file transfers change or a new file transfer arrives. The second argument indicates the direction about which the `Messenger` should be informed (incoming / outgoing file transfers).

void unregisterMediaTransferMessenger(Messenger, int) unregisters a `Messenger` registered using `registerMediaTransferMessenger`.

boolean acceptTransferFromJid(String, int) accepts an incoming transfer with a given id and and stores it to a given file. Returns, wheter the transfer has been accepted successfully.

boolean denyTransferFromJid(int) denies an incoming file transfer with a given id. Returns, if this action has been executed successfully.

intgetIds(int) gets the ids of all file transfers. The parameter indicates if ids from incoming or outgoing transfers should be returned.

TransferParcel getTransferParcel(int) returns information about the state of a specific file transfer (with its id given as argument) represented in a `TransferParcel` object.

Internal Structure

The internal structure of the `TransferService` is shown in figure 6.9. The class `TransferService` itself manages the lifecycle of the service. It inherits from `XMPPConsumerService`, a base class which is responsible for binding to the underlying `IXMPPService` provided by the MXA. It also initiates the XMPP connection procedure if the MXA didn't connect to the XMPP server yet. The `TransferService` class defines an inner class called `ServiceBinder` which inherits from `ITransferService::Stub`, a stub class automatically code-generated by AIDL implementing the `ITransferService` interface and enriched by code allowing interprocess communication.

The `ServiceBinder` class aggregates a `TransferManager` which is responsible of managing all `Transfer` objects, each one representing one file transfer and having its own lifecycle. An object can register as `TransferObserver` at a `Transfer` object to be informed when the state of the file transfer changes. A class may also register as `TransferRequestObserver` at the `TransferManager` to be informed about incoming file transfers. Per default, the `ServiceBinder` implements and registers with both interfaces to be informed about every change of state and about every new incoming transfer. It then notifies all `Messengers` which have been registered using the `registerMediaTransferMessenger(...)` method of the remote interface.

A `Transfer` knows that its state has changed by a call sent to the `ServiceHandler`. The `ServiceHandler` is an inner class of `TransferService` which is wrapped into a `Messenger` and send to the `IFileTransferService` to be informed about transfer state updates, i.e. completion of negotiation processes, transfer of a single file block etc. The `XMPPConsumerService` provides the base implementation of `ServiceHandler` from which the `ServiceHandler` in `TransferService` inherits. The base implementation catches messages concerning XMPP connection and reacts to them.

Another internal class of `TransferService` is `ServiceNotifier`. It is responsible for updating the notification window during a file transfer or in case a new file arrives.

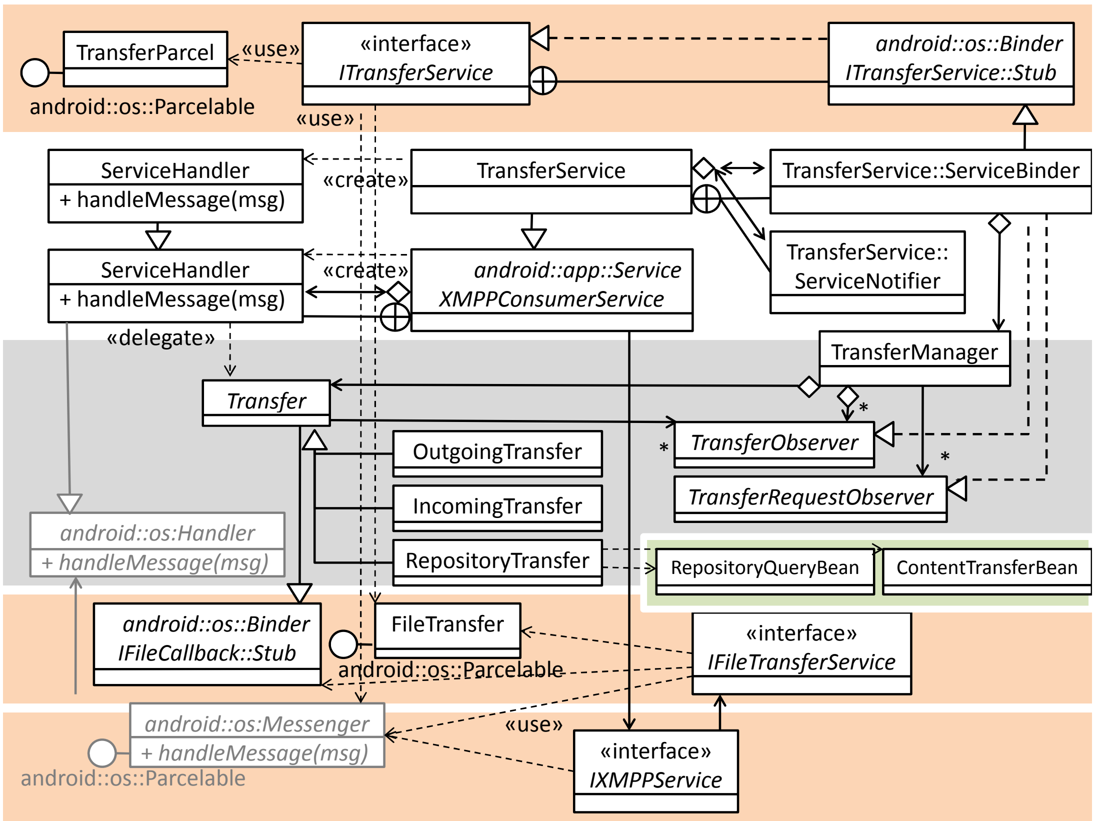

Figure 6.9.: Internal Structure of the Transfer Service

6.5.3. External Service: RepositoryService

Service Boundaries

The `RepositoryService` can be used by connecting to its AIDL interface sending an Intent with the action `de.tudresden.inf.rn.mobilis.media.services.IRepositoryService`:

```
Intent i = new Intent("de.tudresden.inf.rn.mobilis.media.services.
    IRepositoryService");
context.bindService(i);
```

The IBinder which is sent back to the context's `onBind(...)` method once the service is bound provides the following remote interface. Every method is used for one service primitive, as it was introduced in the chapter "Conceptual Design" in section 5.4.

void discover(String, Messenger, int) – **Discovery Service Primitive**: Discovers all repository brokers from a mobilis server with the given bare JID (first parameter). The result will be sent to a `Messenger`.

void query(String, ConditionParcel, Messenger, int) – **Browsing Service Primitive** (FR-3): Queries a repository broker with a given JID (first argument) for repository items. The filtering condition is given by a `ConditionParcel` object.

void delete(String, String[, Messenger, int)] – **Deletion Service Primitive** (FR-4.4): Deletes a list of uids (second parameter) from a repository broker with a given JID (first parameter).

void transfer[String, String, String, Messenger, int) – **Download Service Primitive** (FR-4.3): Requests the initiation of a content transfer from a given repository broker (first parameter) and given content broker (second parameter) given the item's uid (third parameter).

The last two parameters always specify the `Messenger` where the result has to be sent to and a result code, which is used to identify the result with the request.

Internal Structure

Like the `TransferService`, the `RepositoryService` is responsible for for managing its own lifecycle and inherits from `XMPPConsumerService` which maintains the connection to the `IXMPPService` offered by the MXA. The `RepositoryService` also contains a `ServiceHandler`, an inner class inheriting from `Handler`. This class is wrapped into a `Messenger` and sent to the IXMPPService to notify it about the progress of every tasks carried out in the name of the `RepositoryService`. Messages issued to the `ServiceHandler` are forwarded to the respective `Task` objects.

The `ServiceBinder` is another internal class of the `RepositoryService` which implements `IRepositoryService` via the abstract AIDL-generated class `IRepositoryService::Stub`. An instance of `ServiceBinder` is returned as an `IBinder` to any context which binds to the service. For every call to the interface, a new `Task` is created – there is a concrete implementation of the abstract `Task` class for every of the four methods of the `IRepositoryService` interface.

6.5.4. User Interface

System Boundaries

The User Interface of the client prototype is represented by a set of `Activity` classes. The following `Intent` actions can be used to start an activity of the Mobilis Media application from any other Android application:

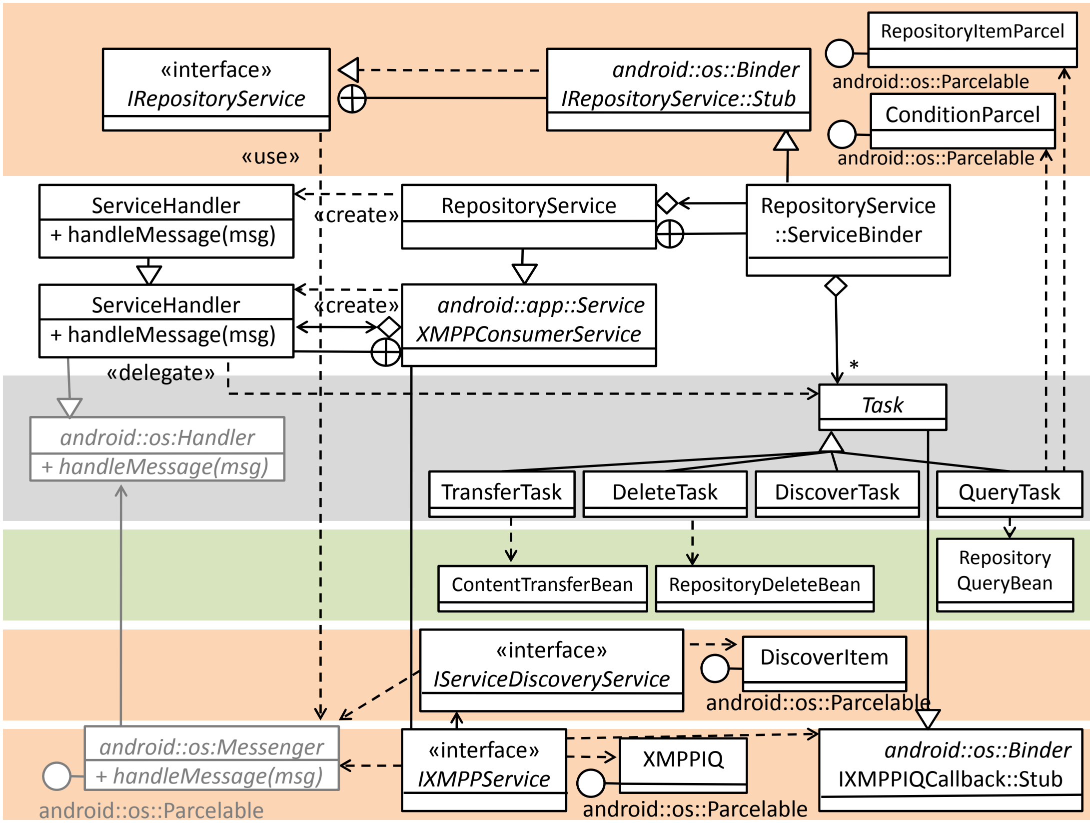

Figure 6.10.: Internal Structure of the Repository Service

de.tudresden.inf.rn.mobilis.media.intent.SEND Opens the `SendActivity` which will first let the user choose an image from a picture chooser.

android.intent.action.SEND Has the same effect despite of the fact that the image is determined by image.getData(). This intent is called by the system image gallery application which is pre-installed on Android OS when a user selects an image and clicks the "Share" button.

de.tudresden.inf.rn.mobilis.media.intent.CHECK_TRANSFER Opens the `Transfer-Activity` with the list of ongoing incoming and outgoing transfers.

de.tudresden.inf.rn.mobilis.media.intent.DISPLAY_REPOSITORYITEM Opens the `RepositoryItemActivity` to show metadata and commands of a single repository item. The intent has to have two extras: `STR_REPOSITORY`, a `String` holding the repository JID and `PAR_REPOSITORYITEM`, an instance of the parcelable `RepositoryItemParcel` which holds the contents of the repository item.

Internal Structure

Given the use of the underlying services, despite of some subtleties handling with framework UI classes, the implementation of the GUI layer is straightforward. An interesting

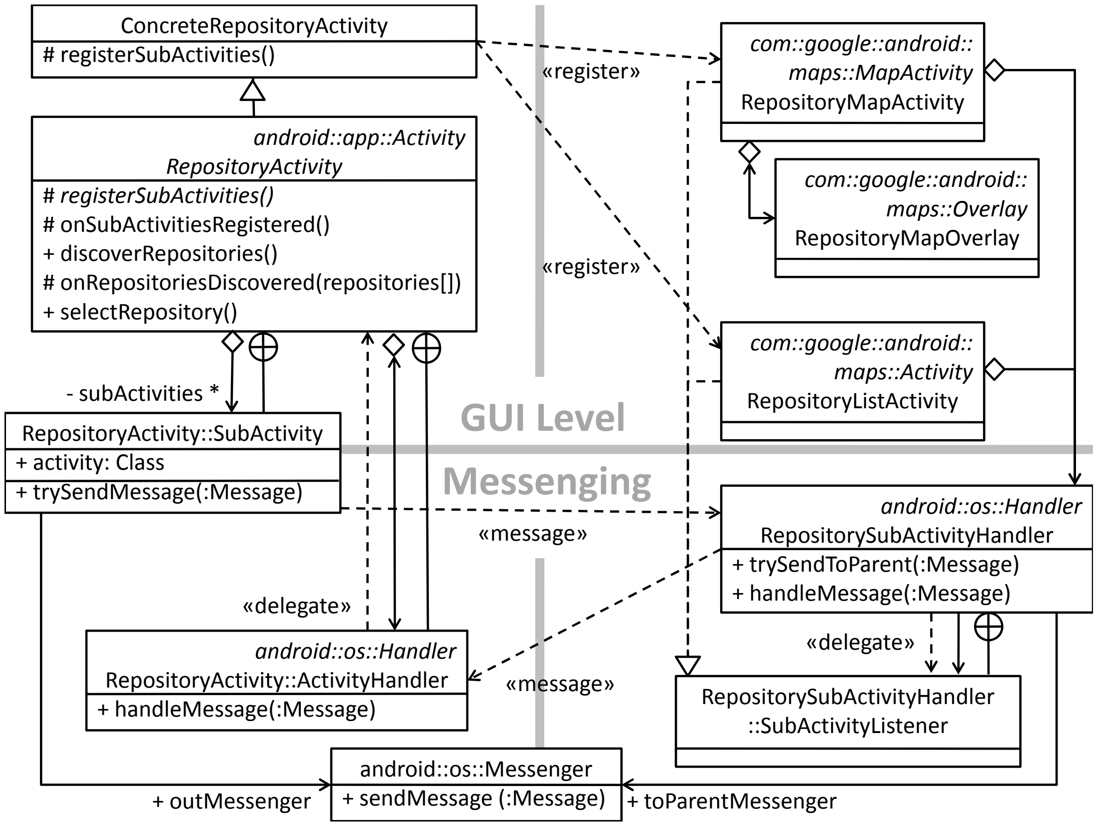

Figure 6.11.: Architecture for exchanging messages between a RepositoryActivity and its subactivities

issue is the implementation of the `RepositoryActivity` since it calls and communicates with multiple subactivities. The corresponding class structure is shown in figure 6.11.

`RepositoryActivity` is a subclass of `TabActivity` and therefore able to host a set of tabs, each one displaying another `Activity`. While the `RepositoryActivity` is responsible for querying the repository items using the `RepositoryService`, the sole task of the `Activity` classes hosted inside the `RepositoryService` is to let the user choose a filtering and to display the filtered items.

To allow communication between `RepositoryActivity` and subactivities, both activities posess a Messenger which is used for bidirectional communication to notify the counterpart upon change of the filterin condition, need to refresh the view etc. The messenger of the `RepositoryActivity` is given to the subactivity as an `Intent` extra when the subactivity is started by the `RepositoryActivity`. The subactivity then immediately sends a `Message` to this messenger to give its own messenger to the `RepositoryActivity`. This functionality is encapsulated by the class `RepositorySubActivityHandler`, which is aggregated by every subactivity. It inherits from `Handler` and delegates incoming messages to a `SubActivityListener` which can be registered using `repositorySubAcrtivityHandler.setSubActivityListener(...)`. In the current implementation, the subactivities act as `SubActivityListener` themselves.

6.6. Conclusion

In this chapter, an prototype implementation of the Mobilis media sharing platform designed in chapter 5 was presented and served as a proof of concept for the design. The prototype implementation was related to the Mobilis architecture and integrated into it.

The Mobilis server has been enhanced by two broker services realizing a media repository open for different usage scenarios. Relational databases administered by Hibernate serve as realization of the data model for the repository. As stated in the design and requested by the requirements (FR-3.5 and FR-2.5), the repository is highly generic concerning storaged data and therefore allows various usage scenarios.

One scenario, the scenario of picture sharing was realized by a the implementation of an Android client with a UI suitable for this task. But also this client implementation provides code which can be reused in future client prototypes which make use of the repository functionality. Android services have been realized which implement file transfer and transfer to repositories. The remote interface of the service was described as well as the internal structure of them.

7. Evaluation

While the preceeding chapters to large parts delt with the formulation of the problem and the elaboration of a possible solution, this chapter evaluates the solution in terms of practicability, performance, efficiency and energy consumption in a mobile environment. In doing so, non functional requirements which have been set in chapter 4 are revisited and performance tests are carried out. [Kor08a] already analyzed applicability of XMPP and Android in a mobile development. This chapter will concentrate on evaluating the design decision for SI File Transfer (7.1) and then continue by architecture of the cube repository (7.2). Afterwards the implementation will be evaluated (7.3) – to large parts concerning the introduction of a multiprocess client prototype with interprocess communication and concerning the use of relational databases managed by Hibernate on server side. The chapter will be concluded summarizing known issues and challenges (7.4).

7.1. Applicability of SI File Transfer

In section of the chapter "Conceptual Design" Stream Initiation File Transfers where introduced as a file transfer protocol. Indeed this protocol has proven to be very reliable for wireless networks: since both entities that transfer a file can connect to a SOCKS5 proxy server after the transfer is negotiated, no incoming TCP stream has to be accepted by the mobile client, especially not when a file arrives at the client. This is especially important because incoming TCP streams are still blocked by most mobile network providers.

Another argument for SI File transfer is the fact, that it is a technology which can send binary data unencoded over the wire. The only stage with unneccessary redundancy is the the negotiation of the file transfer. The redundancy, however is due to the XML character of the XMPP stanzas what is rather a general XMPP problem. Other problems concerning XMPP also apply: for example, it is not possible to restart the file transfer from a specific offset when one entity disconnects, e.g., due to mobile handover or weak signal reception.

To quantitatively evaluate the SI File Transfer protocol, several performance measurements have been executed.

7.1.1. Test Environment and Methodology

The test was carried out by sending images from the mobile client prototype to the repository broker with different file sizes and different block sizes. Two file sizes where used: 95833 Bytes and 1020249 Bytes. The block size indicates how many bytes are transferred in one row before sending `Message` objects to the above layers and proceeding with the next block. The small image of 95833 Bytes was transferred at block sizes of 1024, 2048 and 4096 Bytes. The bigger file was transferred with block sizes of 2048 Bytes and 4096 Bytes.

The used mobile phone was a HTC G1 connected to a 54kbps WLAN access point. The mobile phone was rebooted before every measurement to provide the same starting conditions. The mobilis server ran on an ASUS F3JM laptop with 1.83 GHz and 2.0 GB RAM. Both devices were connected to a WLAN access point with 54Mbps.

The system time in milliseconds was measured at the initiation of the SI File Transfer, after negotiation and after the transfer of every block. The measurement took place on MXA level and was written to the console, which was read out by USB debugging.

7.1.2. Measurement of Transfer Time

Figures 7.3 and show the progress of the transfer for the filesize of 95833 Bytes and 1020249 Bytes respectively. The diagrams show at which time (y axis) a specific amount of bytes is transferred (x axis). In figure 7.3 some kinks are visible every 10 blocks where the transfer seems to stop for about 500ms.

In the case of a small filesize (95833 Bytes), the speed of the transfer with a smaller block size seems faster than with larger block size. However, in case of bigger files (1020249 Bytes) this effect is reversed. Figure 7.4 shows this effect: starting from ca. 200kB, the speed suddenly grows for the bigger block size. It is assumed that this effect is due to the change of quality of service parameters for bigger files, e.g. the TCP slow start mechanism.

Figures 7.3 and 7.4 show the speed of the file transfer (y axis) for the amount of bytes transferred (x axis). The speed is calculated by taking the derivate of the original graph

$$v_i = \frac{\Delta TransferredBytes_i}{\Delta t_i} = \frac{TransferredBytes_i - TransferredBytes_{i-1}}{t_i - t_{i-1}}$$

Another measurement of interest is the time needed to negotiate the file transfer. In the current collected data, this time varies between 300ms and 900ms.

Table 7.1 summarizes our findings by t_{nego}, t_{end} as well as the average and standard derivate values for transfer speed ($\bar{v}$ and $\tilde{v}$) are and illustrate our findings:

Scenario	t_{nego}/ms	t_{end}/ms	$\bar{v}/kBs^{-1}$	$\tilde{v}/kBs^{-1}$
Filesize 97833kB, Blocksize 1024B	392	9031	17.869	10.409
Filesize 97833kB, Blocksize 2048B	918	6841	24.747	14.771
Filesize 97833kB, Blocksize 4096B	325	5559	20.908	6.177
Filesize 1020249kB, Blocksize 2048B	345	68775	32.073	32.505
Filesize 1020249kB, Blocksize 4096B	508	373426	12.726	16.358

Table 7.1.: Maximum Transfer Time and Speed of the test transfers.

7.2. Evaluation of the Repository Architecture

The custom XMPP extension protocol defined in chapter 5 provide all necessary functionality to realize to realize the cube repository architecture are flexible enough to allow extension of the service primitives to allow more complex query mechanisms. This is necessary to let the mobile client specify its request more precisely to allow unnecessarily transmitted data. Imagine a user querying the database with a `<condition>` matching

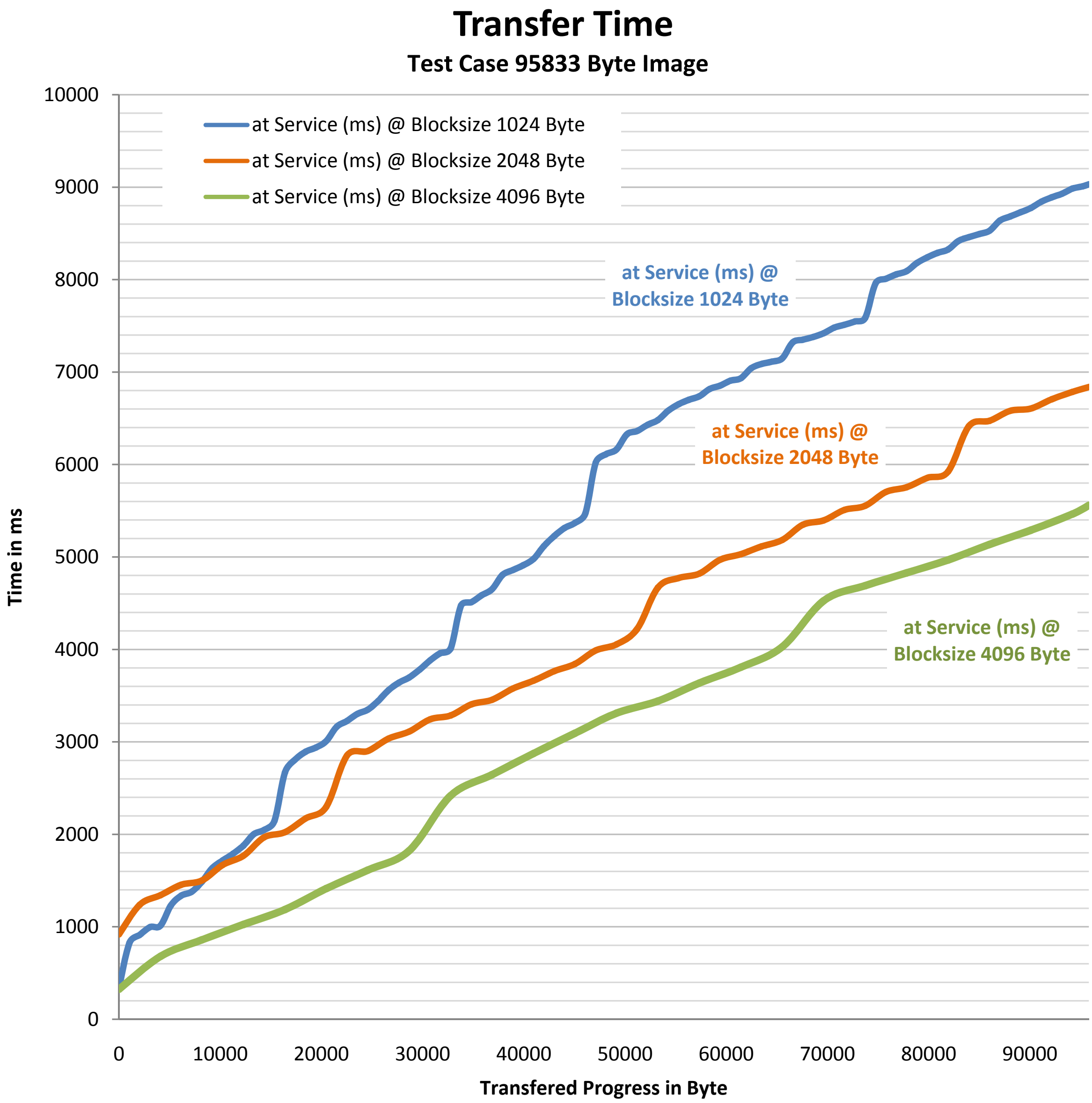

Figure 7.1.: Transfer time of a 95833 bytes image

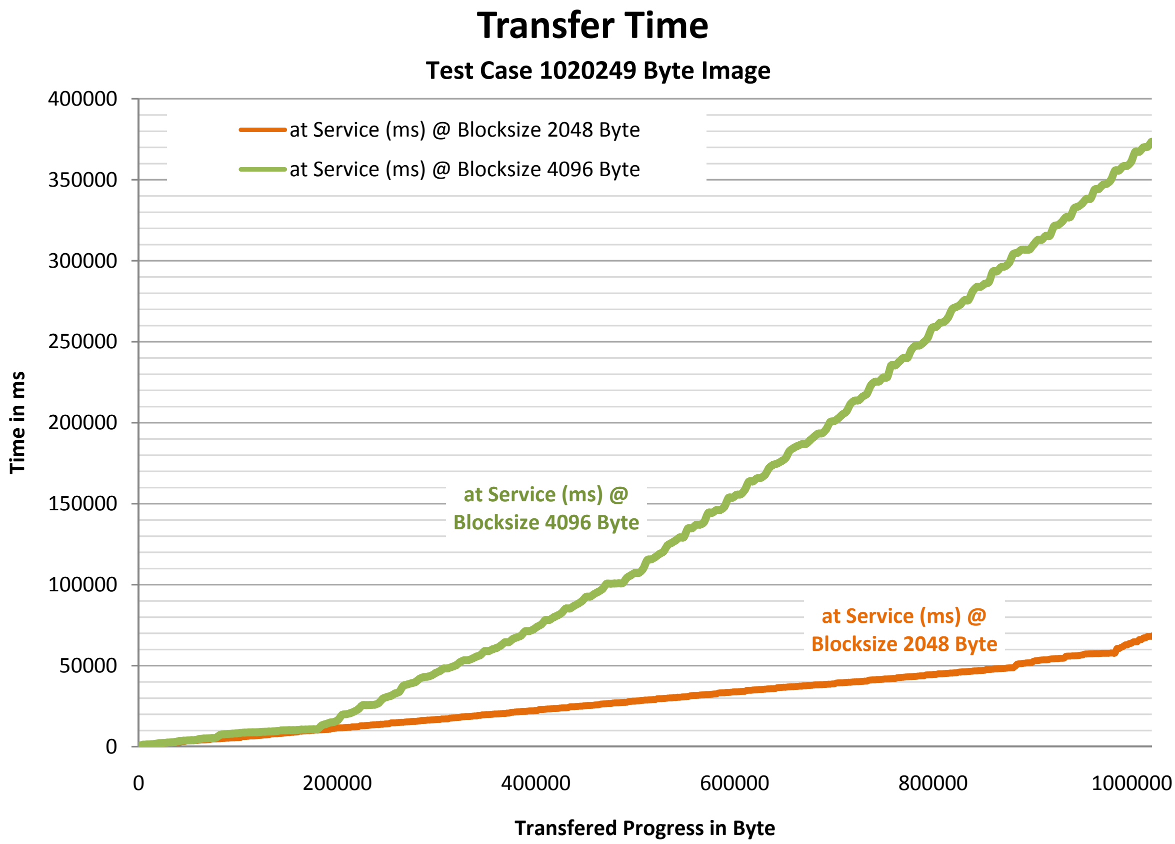

Figure 7.2.: Transfer time of a 1020249 bytes image

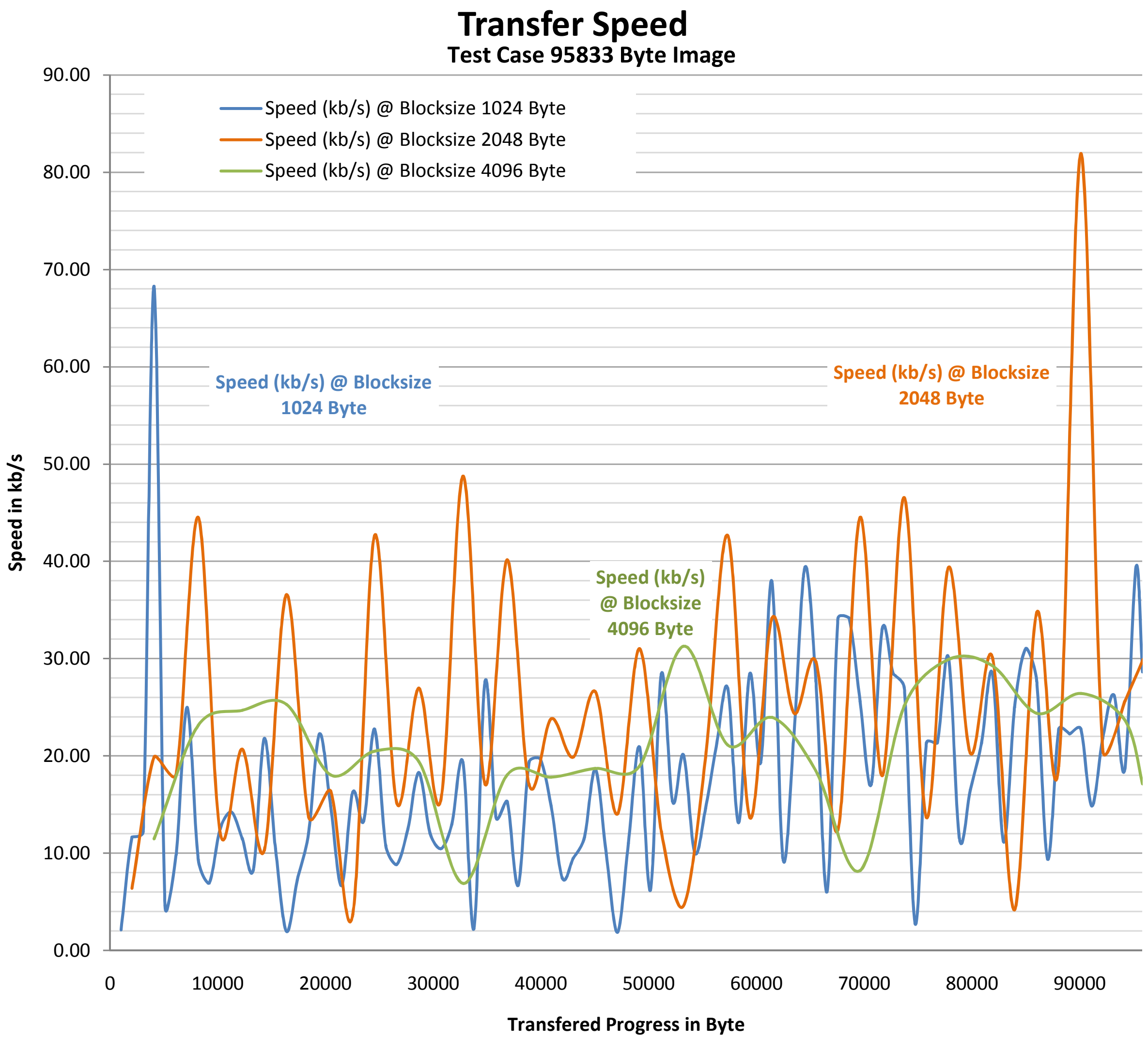

Figure 7.3.: Transfer time of a 95833 bytes image

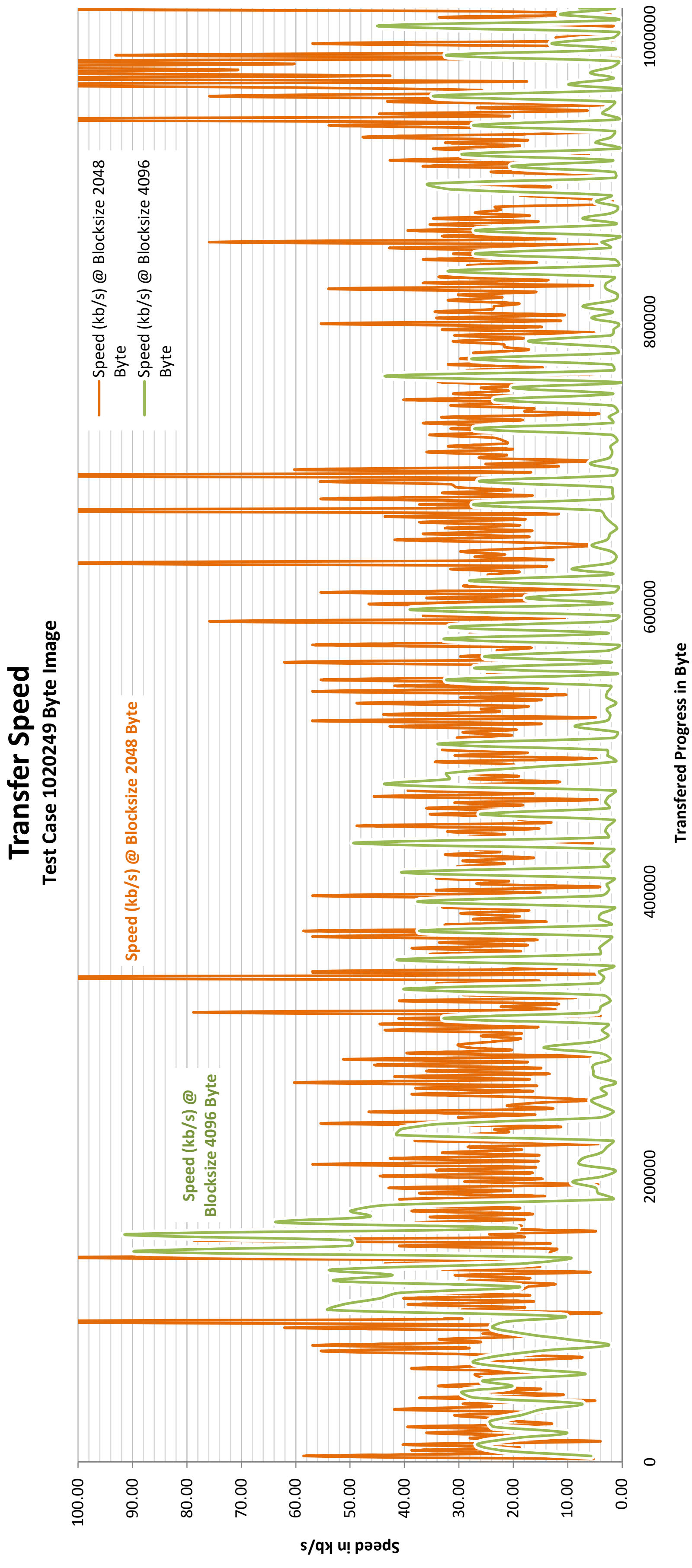

Figure 7.4.: Transfer time of a 1020249 bytes image

an enormous high number of repository items. In the current architecture they will all be returned - moreover - with all slice assignments what may also be a high number. Taking into account the redundancy overhead of XMPP the size of the returned package might be enormous and slow down both mobile client and server.

A solution to this problem is to allow fine tuning of the `<repository-query>` package. Compared to an SQL statement, the `<condition>` corresponds to the `where` clause. Future implementation should also add the possibility to restrict the returned information in number (SQL `limit` clause), granularity (SQL `select` clause) and sorting (SQL `order by` clause).

The same applies to the replacement of repository items. Currently, a repository item has to be replaced completely, that means, its slices are completely overwritten and even the content has to be stored newly. This is an enormous overhead which can be avoided, if the update logic would be fixedly implemented in the custom cube repository protocol.

More ideas concerning the custom cube extension protocol can be found in the prospect of chapter 8.

7.3. Evaluation of the Implementation

7.3.1. Server Side

One core problem on the server side is the database connection. The implementation of the repository slicing with a simple HashMap is far from optimal. Assume the mobile client issues a simple request like the following:

```
<iq type='get' id='mobilis_5'
    from='client@xmpp/MXA'
    to='mobilis@xmpp/Repository'>
  <repository-query xmlns='http://rn.inf.tu-dresden.de/mobilis#services/
      RepositoryService'>
    <condition op='and' xmlns='http://rn.inf.tu-dresden.de/mobilis'>
      <condition key='taken' op='lt' value='1256468400000' />
      <condition key='taken' op='gt' value='1256461200000' />
    </condition>
  </repository-query>
</iq>
```

The request will be translated by the `RepositoryCube` to the following so-called HQL statement:

```
from de.tudresden.inf.rn.mobilis.server.services.media.RepositoryItem it
where (it.slices['taken']<='1256468400000') and (it.slices['taken']>='
    1256461200000')
```

This HQL statement will be translated by Hibernate to an equivalent SQL statement:

```
select
  repository0_.uid as uid1_,
  repository0_.content as content1_,
  repository0_.owner as owner1_
from
  mobilis_repositoryitem repository0_,
  mobilis_repositoryitem_slice slices1_,
  mobilis_repositoryitem_slice slices2_
```

```
where
  repository0_.uid=slices1_.uid
  and repository0_.uid=slices2_.uid
  and slices1_.slice_key = 'taken'
  and slices2_.slice_key = 'taken'
  and slices1_.slice_value<='1256468400000'
  and slices2_.slice_value>='1256461200000'
```

The example shows that for every `<condition>` one cross join with two `where`-conditions are introduced. However the statement could be optimized into into three independent nested statements:

```
select
  repository0_.uid as uid1_,
  repository0_.content as content1_,
  repository0_.owner as owner1_
from
  mobilis_repositoryitem repository0_
where
  (select slices1_.slice_value
      from mobilis_repositoryitem_slice slices1_
      where slices1_.slice_key = 'taken'
      and slices1_.uid = repository0_.uid)
    <='1256468400000'
  and (select slices2_.slice_value
      from mobilis_repositoryitem_slice slices2_
      where slices2_.slice_key = 'taken'
      and slices2_.uid = repository0_.uid)
    >='1256461200000'
```

The time needed to process both requestes ws analyzed using the MySQL console. The second, optimized statement executed in 50ms while the Hibernate-generated statement executed in about 300ms.

The result of the two inner statements depend on the currently visited record of the outer statement so they cannot be precalculated before the outer statement is run. However, they should be executable quite fastly since they are dependent on the `uid` and `slice_key` field of the `mobilis_repositoryitem_slice` table which are primary keys and therefore stored in a search-efficient datatype. The outer statement, however, has to traverse all repository items.

Having a `<repository-query>` with c conditions and a repository with N repository items each having n slide assignments, the database hence has to perform c hash accesses on N repository items. Since a hash access is of order $O(n \log n)$ the complexity of a single request is described by

$$O(N \cdot c \cdot n \log n)$$

.

7.3.2. Client Side

One key experience we made when using the client prototype was the finding, that the User Interface sometimes felt “sluggish” when the file transfer was ongoing. That means, the progress bar notifying the user about the transfer progress was updated delayedly.

The reason to this fact has been researched: In parallel to the actual file transfer measurements (as introduced in section 7.1), the time was measured, when the user interface was notified about the progress. Figure 7.5 and 7.6 show this effect: the dashed lines represent the transfer as shown to the user while the solid lines show the transfer as it is in fact happening.

It turns out, that this effect is percieved less for bigger block sizes. The smaller the block size the much more significant this **lag** is, especially for very large files, since the lag accumulates over time. In our example the accumulated lag at the end of the file transfer is in the following order (table 7.2):

Scenario	Lag_{max}/ms
Filesize 97833kB, Blocksize 1024B	52227
Filesize 97833kB, Blocksize 2048B	12959
Filesize 97833kB, Blocksize 4096B	2471
Filesize 1020249kB, Blocksize 2048B	1759111
Filesize 1020249kB, Blocksize 4096B	53621

Table 7.2.: Lag between Transfer Time and UI Response.

1759111ms – that is almost 30 minutes! This is a totally unacceptable value for productive use. During this time, the mobile phone seems unusable because the GUI thread takes up very much resources and even heats up the phones processor, what can be literally felt. This is a clear violation to the non functional requirements NF-3.3 and NF-3.5.

The reason of this defect lies probably in a design failure concerning the interprocess communication between MXA, external services and GUI layer. For synchronizing threads, we make use of the Android framework classes `Message`, `Handler` and `Messenger`. However, these classes use Message queues, so if receiptient threads are currently busy, e.g. drawing an UI element, incoming messages are added to a queue. If handling one `Message` takes longer then transferring a block over the wire, a bottleneck is created which slows down the UI, creating waiting Messages, taking up ressources, eventually slowing the thread down even more etc. pp.

A possible solution would be the use of other means of content sharing, e.g., a pulling mechanism. However, the time of this work was limited so broader research on this issue could not be accomplished more intensively.

7.4. Conclusion

In this chapter, both the architecture of the custom XMPP extension for cube media repositories as well as its implementation on client and server side have been examined qualitiativedly and in measure. Three potential issues where identified: The first was the inflexibility of the repository query and upload mechanism, which is solvable by extending the related protocol by further elaborations. The second is the question if the current relational database model is efficient and scalable enough. To answer this question, further measurements and calculations should be carried out. And the final issue, the sluggishness of the GUI layer, is a clear defect on implementation level. It represents

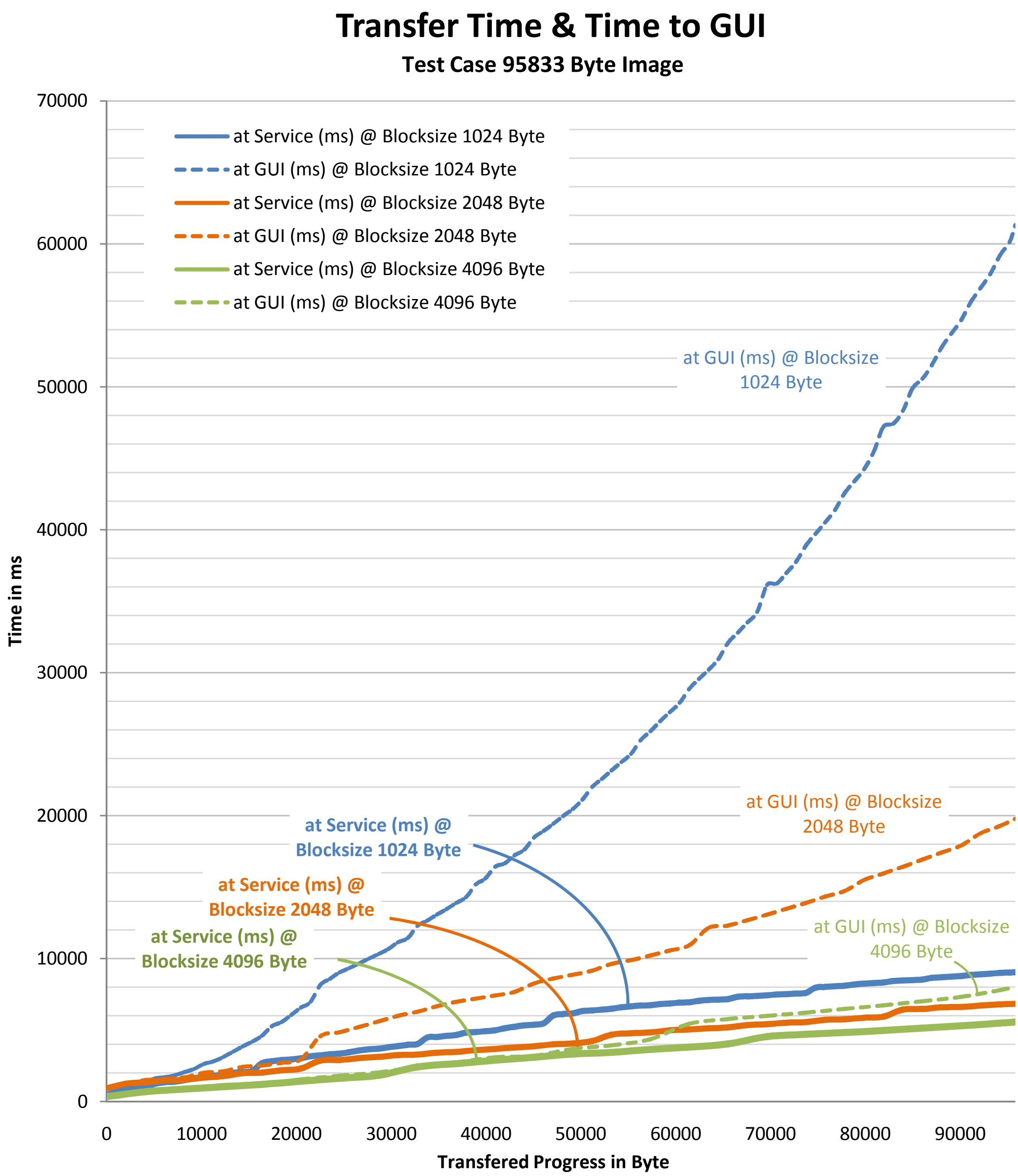

Figure 7.5.: Speed of the transfer of a 95833 bytes image

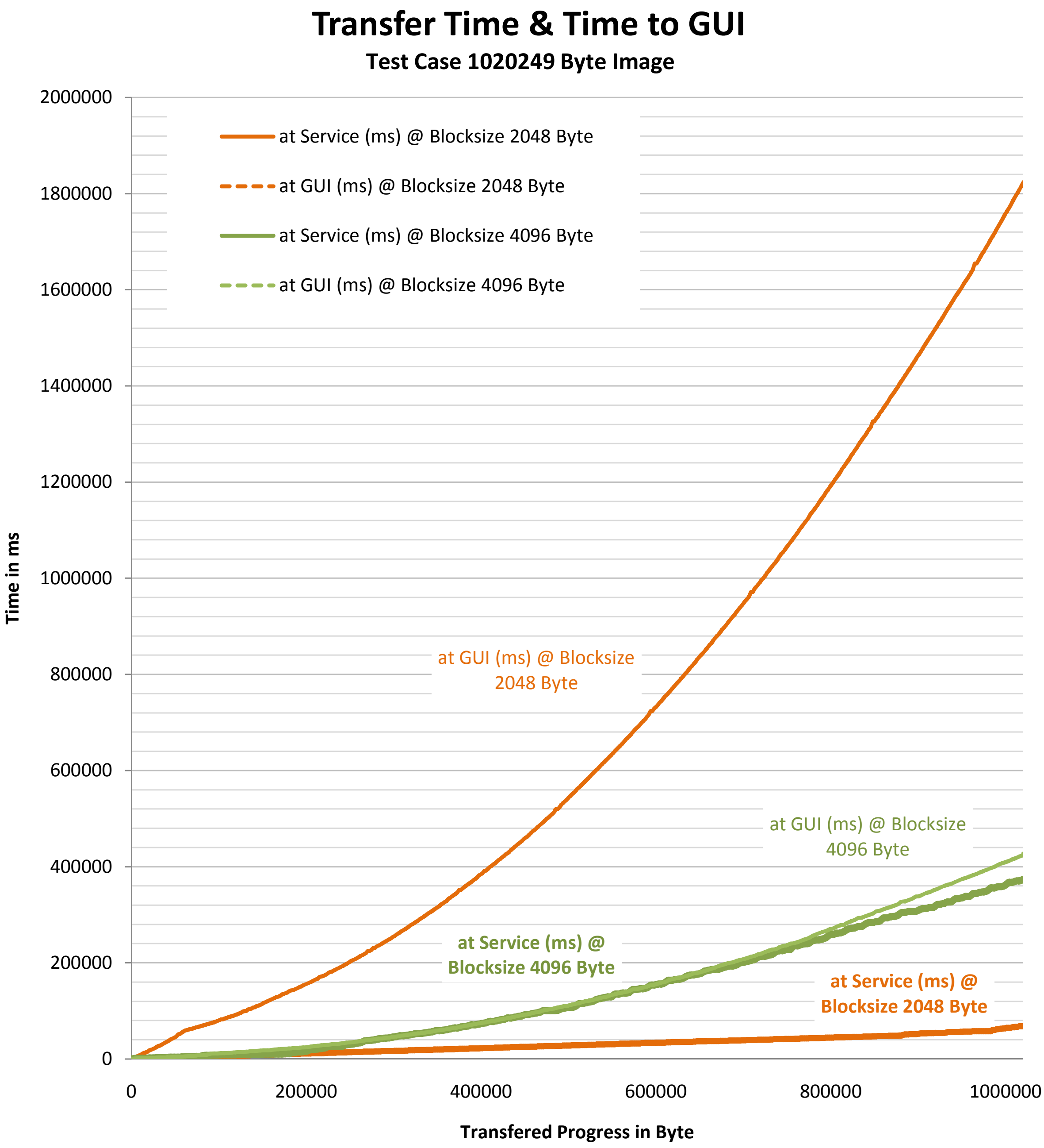

Figure 7.6.: Speed of the transfer of a 1020249 bytes image

a lesson-learned which should be considered when dealing with other real-time updated GUIs. The following chapter presents a prospect with ideas for future work which have not been considered in requirements and design.

8. Prospect

From the introductory user scenario indicating a problem, this thesis has developed a long bend over the whys, whats, hows to a solution of that problem. The presented solution was evaluated and lessons learned where shared. During the whole process, further problems were identified and new ideas came to life. This chapter presents problems, which where not in the scope of this thesis and therefore also have not been solved by the current prototype. They provide a starting point for future work.

Section 8.1 introduces the idea for new prototypes based on the current architecture and suggests possible enhancements to the current prototype which would improve the user experience. Section 8.2 shows how the underlying architecture can be modified to be more flexible and adapted to the mobile environment.

8.1. Possible Enhancements of the Prototype

While the repository is general enough to provide a framework for realization of other user stories, especially the client prototype user interface is constructed to strictly implement the settled requirements of an image sharing tool. A simple task would be to implement prototypes which use the repository architecture for other means of media sharing, like music or video sharing. Another possible task could be to implement a client prototype running in another environment, e.g., on a desktop system. Together with a mobile prototype, the architecture may then be used to ubiquitously manage a users personal image library.

Finally, some functionality is not implemented, which would be nice-to-have when using the prototype in daily life: this concerns handy features like uploading or downloading multiple files at once, seeing a thumbnail view of online content or filtering the items by user-defined conditions. Also an overview over downloaded content allowing synchronization of it with uploaded content would be an enhancement to the user experience.

8.2. Possible Enhancements of the Repository Architecture

We already analyzed the repository architecture in the previous chapter (see section 7.2) and expressed the need for more sophisticated query and update mechanisms. In particular, updates and requests should be possible on a more fine-granular level than repository items. Also the query condition language (syntax and semantics of the `<condition/>` tag) could be enriched by more complex terms, functions and operators than it currently supports. Sorting the result, limiting it to a specific amount of items or specifying the granularity of the returned items could be implemented. Partly this is already introduced in the current design by the distinction between *concrete*, *complete*, *referencing*, or *simple* `<repository-item/>`s (see subsection 5.4.1).

Moreover, security mechanisms could be dramatically improved by implementing proper authentication mechanisms or access control lists. This might include the possibility to split up a repository into sub-repositories, which is currently only supported by introducing multiple repository brokers.

8.2.1. Practical Comparison with other File Transfer Technologies

We use SI File Transfer as a building block of our repository architecture to transfer binary data between XMPP entities. The reason for this design decision has been broadly discussed (see 7.1). However, the architecture of the repository allows replacing this mean of transfer by another, probably more advanced technologies, like Jingle [LBSA+09], if practical reasons like the availability library support would make the effort to integrate the technology reasonable. Future work may modify the introduced media sharing by replacing SI File Transfers by another one-to-one binary transfer and evaluate those methods practically.

8.2.2. Practical Comparison with other Repository Models

The presented repository architecture is backed by a data model which has a cube shape. This shape was chosen based on the introductory user scenario. In other file sharing scenarios, hierarchical storage might be beneficial. However, hierarchical information can only be mapped difficultly to the repository.

Future work could develop a hierarchical repository architecture and compare the introduced cube structure to the hierarchical structure. During the development of this thesis, a Pub-Sub library [1] from Ignite Realitime was announced. This library could be used to develop a hierarchical repository based on Published Stream Initiation Requests and, if possible, suggest a hybrid solution which combines the multi-dimensional and hierarchical model.

Also one core strength of XMPP has not been taken into account yet: the possibility to use it as a push service. Using a publish-subscribe mechanism, a user could be automatically informed about newly available content. This would save bandwidth, which is a core issue in mobile environments.

8.2.3. Thinking Big: Replication and Partitioning

A media sharing platform can grow quite big with the amount of participating users and files. Good scalability is therefore an important property. The presented architecture introduced some means of load balancing by allowing multiple content stores and repository cubes on different physical entities. However, in todays media sharing platforms far more professional technologies exist: Files are partitioned and spread among different entities or they are held in multiple replicas on different storages which creates the need for a proper synchronization algorithms.

[1] `http://nixbit.com/cat/programming/libraries/smack-pubsub-extensions/`

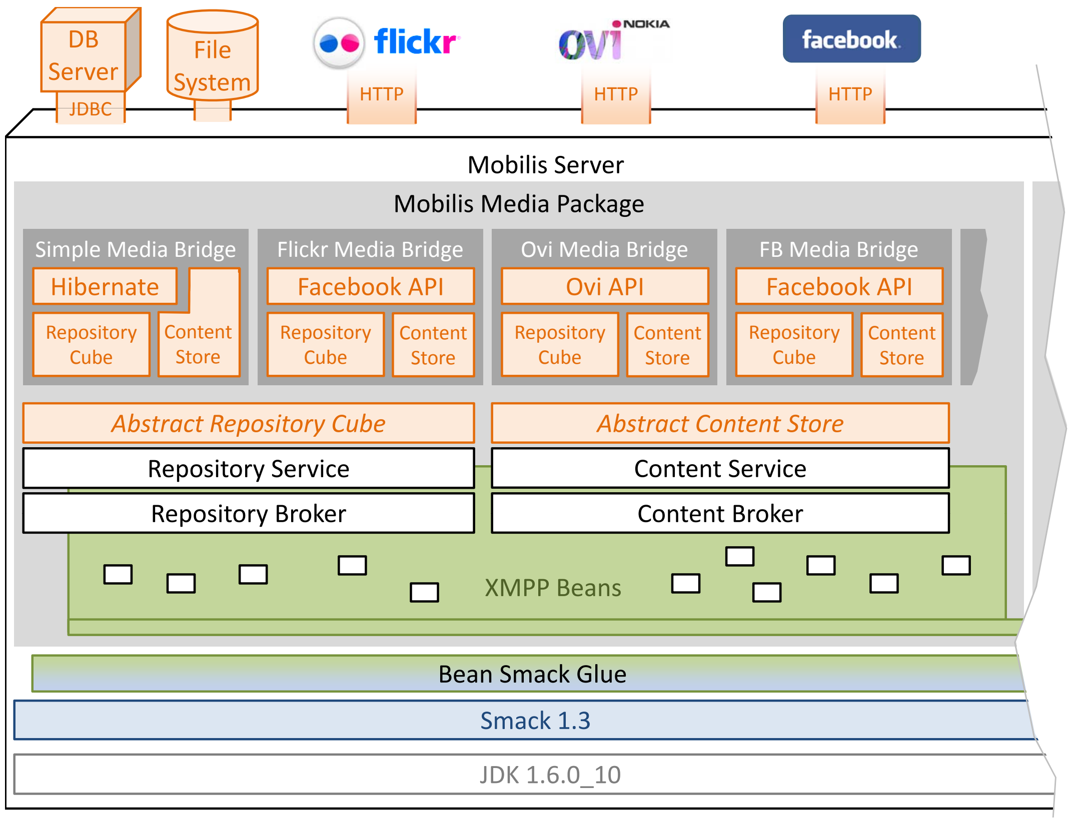

Figure 8.1.: Media Bridges as a mean for a genaralized social cube based media repository.
Icons are registered trademarks of Yahoo Inc., Nokia Oy and Facebook respectively.

8.3. Coupling with other Media Repositories

The introduced user scenario of travel picture sharing provides a fairly well-known practical background to be extended to many other use cases. One possible extension lies in the weakness of the social aspect of the presented solution: our platform introduces just another social network to a possible new user and the user probably already has other social network accounts where she shares and publishes pictures. An idea would therefore be to integrate an interface to other social networks into the presented architecture.

Indeed, this is easily possible due to the flexible data model of content with assigned arbitrary metadata. The implemented content store and repository cube, by design, store images into the file system and into a relational database currently. This has been a simple design decision, but of course, they may also store images into any other media repository, like Flickr [2], Nokia Ovi [3] or Facebook [4]. The advantage of this is, that a user already has a community of friends there waiting to see her pictures.

On implementation level, the presented idea would result in an abstraction of `RepositoryCube` and `ContentBroker` and a set of replacable **media bridges**, which provide their own implementation of the two entities. This concept is shown in figure 8.1. The

[2] `http://www.flickr.com`
[3] `http://www.ovi.com`
[4] `http://www.facebook.com`

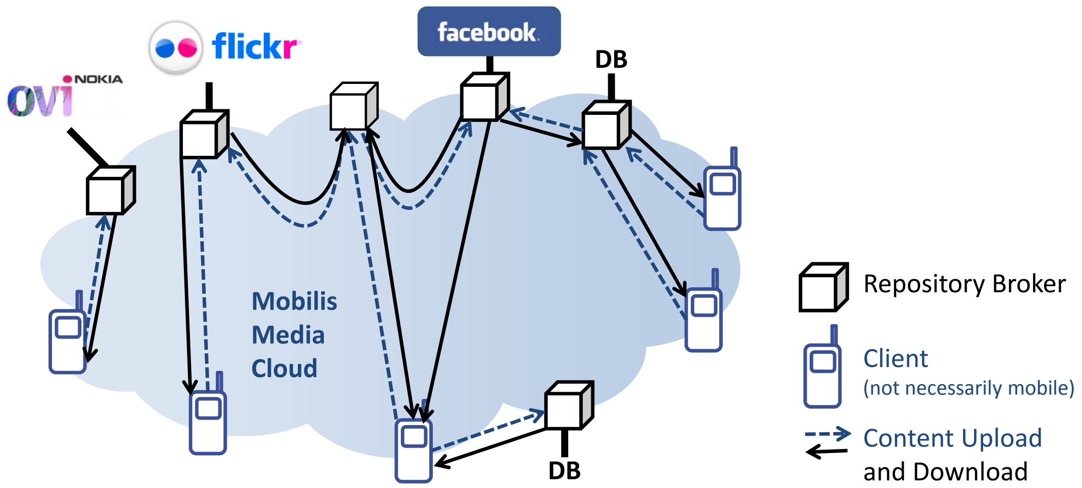

Figure 8.2.: The Mobilis Media Cloud

Icons are registered trademarks of Yahoo Inc., Nokia Oy and Facebook respectively.

idea can be realized in a similar way to the generalized handling of friend lists using a `BuddyBroker`, which has been described in [DS09].

8.4. Conclusion

To conclude, the Mobilis Media repository can be extended to build up a central **Media Sharing Cloud** which allows media exchange between different content creation devices like Mobile phones and arbitrary content storage entities being either tied to the Media Sharing Cloud or located remotedly. That way, platforms like Facebook, Flickr etc. can be connected to. Imposing that repository brokers can also play the role of clients in the sense of the Media Sharing Cloud, they may also be used as adapters or aggregators to link several repositories together. The overall idea of the **Mobilis Media Cloud** is depicted in figure 8.2.

Given the possible size of such a media cloud, future work should beyond the extension of the IQ interface also focus on the design of content distribution, replication and synchronization algorithms. And supposing the variety of user stories, various user prototypes addressed to different tasks could be developed. The introduced Mobilis Media picture sharing solution can in fact be seen as the part of a generalized content sharing platform, which to build up provides of room for quite a high number of research challenges and economical innovations.

A. Appendix

A.1. XSD Schema of used custom IQs

The following listings show the XSD (XML schema definition) of the IQ namespaces defined for the media repository in section 5.4.1.

The Mobilis Namespace

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.rn.inf.tu-dresden.de/mobilis">

<xs:simpleType name="mobilis:opLogicalClause">
  <xs:restriction base="xs:string">
    <xs:enumeration value="or"/><xs:enumeration value="and"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="mobilis:opLogicalUnary">
  <xs:restriction base="xs:string">
    <xs:enumeration value="not"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="mobilis:opComparison">
  <xs:restriction base="xs:string">
    <xs:enumeration value="lt"/><xs:enumeration value="gt"/>
    <xs:enumeration value="le"/><xs:enumeration value="ge"/>
    <xs:enumeration value="ne"/><xs:enumeration value="eq"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="mobilis:uid">
  <xs:restriction base="xs:string" />
</xs:simpleType>

<xs:simpleType name="mobilis:jidFull">
  <xs:restriction base="xs:string">
    <xs:pattern value="^[\w.-]+@[\w.-]+$" />
  </xs:restriction>
</xs:simpleType>

<xs:element name="condition" targetNamespace="http://www.rn.inf.tu-
   dresden.de/mobilis">
  <xs:complexType>
    <xs:choice minOccurs="1" maxOccurs="1">
      <xs:all>
```

```
        <xs:attribute name="op" type="mobilis:opLogicalUnary"  use="
           required" />
        <xs:element name="condition">
      </xs:all>
      <xs:all>
        <xs:attribute name="op" type="mobilis:opLogicalClause" use="
           required" />
        <xsd:sequence>
          <xs:element name="condition">
        </xsd:sequence>
      </xs:all>
      <xs:all>
        <xs:attribute name="op" type="mobilis:opComparison" use="
           required" />
        <xs:attribute name="key" type="xs:NMTOKEN" use="required" />
        <xs:attribute name="value" type="xs:NMTOKEN" use="required" />
      </xs:all>
  </xs:complexType>
</xs:element>

</xs:schema>
```

The RepositoryService Subnamespace

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.rn.inf.tu-dresden.de/mobilis#services/
     RepositoryService">
<xs:import namespace="http://www.rn.inf.tu-dresden.de/mobilis#services"
    />

<xs:complexType name="mobilis:repository:item:referencing">
  <xs:attribute name="uid" type="mobilis:uid" use="required" />
</xs:complexType>

<xs:complexType name="mobilis:repository:item:simple">
  <xs:sequence>
    <xs:element name="slice">
      <xs:attribute name="key" type="xs:NMTOKEN" use="required" />
      <xs:attribute name="value" type="xs:NMTOKEN" use="required" />
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="mobilis:repository:item:concrete">
  <xs:extension base="mobilis:repository:item:simple">
    <xs:attribute name="uid" type="mobilis:uid" use="required" />
  </xs:extension>
</xs:complexType>

<xs:complexType name="mobilis:repository:item:complete">
  <xs:extension base="mobilis:repository:item:simple">
    <xs:attribute name="content" type="mobilis:jidFull" use="required" /
       >
    <xs:attribute name="owner" type="mobilis:jidFull" use="required" />
  </xs:extension>
```

```
</xs:complexType>

<xs:element name="repository-delete">
  <xs:complexType>
    <xs:sequence minOccurs="1">
      <xs:element name="repository-item" type="
          mobilis:repository:item:referencing" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="repository-query">
  <xs:complexType>
    <xs:choice minOccurs="1" maxOccurs="1">
      <xs:sequence minOccurs="1">
        <xs:choice>
          <xs:element name="repository-item" type="
              mobilis:repository:item:concrete" />
          <xs:element name="repository-item" type="
              mobilis:repository:item:simple" />
        </xs:choice>
      </xs:sequence>
      <xs:sequence minOccurs="1">
        <xs:element name="repository-item" type="
            mobilis:repository:item:complete" />
      </xs:sequence>
      <xs:element name="condition" />
    </xs:choice>
  </xs:complexType>
</xs:element>

</xs:schema>
```

The ContentService Subnamespace

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.rn.inf.tu-dresden.de/mobilis#services/
      ContentService">
<xs:import namespace="http://www.rn.inf.tu-dresden.de/mobilis#services"
    />

<xs:element name="content-delete">
  <xs:complexType>
    <xs:element name="uid" type="mobilis:uid" />
  </xs:complexType>
</xs:element>

<xs:element name="content-register" />

<xs:element name="content-unregister" />

<xs:element name="content-transfer">
  <xs:complexType>
    <xs:choice minOccurs="1">
      <xs:element name="retrieveFrom" type="mobilis:jidFull" />
```

```
      <xs:element name="sendTo" type="mobilis:jidFull" />
    </xs:choice>
    <xs:element name="uid" type="mobilis:uid" />
  </xs:complexType>
</xs:element>

<xs:element name="content-item">
  <xs:complexType>
    <xs:choice minOccurs="1">
      <xs:element name="retrieveFrom" type="mobilis:jidFull" />
      <xs:element name="sendTo" type="mobilis:jidFull" />
    </xs:choice>
    <xs:element name="uid" type="mobilis:uid" />
  </xs:complexType>
</xs:element>

</xs:schema>
```

A.2. Data Source of the Performance Evaluation

The following attached pages contain the raw data measured during evaluation of the prototype. They are used for the respective conclusions in sections 7.1 and 7.3.2. See subsection 7.1.1 for a description of the test methodology. The attachment consists of two parts: the first part contains measurements of the 9583 bytes image and includes pages A.2-1 through A.2-4. The second part contains measurements of the 1020249 bytes image and takes up pages A.2-5 through A.2-10. Horizontally, the tables are split up into the measurement of the different block sizes and vertically the measured data for each transferred block is listed. The data includes:

kb transferred – the accumulated number of kilobytes which has been transferred with this block.

at Service the number of milliseconds which has passed when the block is read out from the MXA, measured since the transfer has been initiated

at GUI – the number of milliseconds which has passed when the user is informed about the transferred block, measured since the transfer has been initiated.

Delta – the number of milliseconds which has passed since the last measurement of “at Speed” or “at GUI” respectively..

Speed the first derivate, that is, how fast the last block has been transferred in kb/s.

Lag – the number of milliseconds which passes between arrival of the block at the MXA and notification of the end user that the block has been transferred.

Test Case 9583 Bytes Image File

Block Size	1024							2048							4096						
Block #	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)
initiated		0			0		0		0			0		0		0			0		0
0	0	342	342		414	414	72	0	918	918		930	930	12	0	325	325		337	337	12
1	1024	826	484	2.12	829	415	3	2048	1239	321	6.38	1255	325	16	4096	682	357	11.47	693	356	11
2	2048	914	88	11.64	1104	275	190	4096	1342	103	19.88	1489	234	147	8192	859	177	23.14	876	183	17
3	3072	998	84	12.19	1207	103	209	6144	1455	113	18.12	1576	87	121	12288	1025	166	24.67	1146	270	121
4	4096	1013	15	68.27	1328	121	315	8192	1501	46	44.52	1675	99	174	16384	1187	162	25.28	1243	97	56
5	5120	1233	220	4.65	1612	284	379	10240	1670	169	12.12	1995	320	325	20480	1414	227	18.04	1542	299	128
6	6144	1337	104	9.85	1702	90	365	12288	1769	99	20.69	2114	119	345	24576	1614	200	20.48	1798	256	184
7	7168	1378	41	24.98	1815	113	437	14336	1968	199	10.29	2409	295	441	28672	1825	211	19.41	2030	232	205
8	8192	1488	110	9.31	2046	231	558	16384	2024	56	36.57	2520	111	496	32768	2419	594	6.90	2416	386	-3
9	9216	1636	148	6.92	2276	230	640	18432	2173	149	13.74	2692	172	519	36864	2646	227	18.04	2728	312	82
10	10240	1718	82	12.49	2624	348	906	20480	2298	125	16.38	2939	247	641	40960	2876	230	17.81	3094	366	218
11	11264	1790	72	14.22	2791	167	1001	22528	2853	555	3.69	4561	1622	1708	45056	3095	219	18.70	3191	97	96
12	12288	1879	89	11.51	3102	311	1223	24576	2901	48	42.67	4834	273	1933	49152	3311	216	18.96	3683	492	372
13	13312	2002	123	8.33	3469	367	1467	26624	3036	135	15.17	5199	365	2163	53248	3442	131	31.27	3920	237	478
14	14336	2049	47	21.79	3851	382	1802	28672	3112	76	26.95	5592	393	2480	57344	3636	194	21.11	4357	437	721
15	15360	2147	98	10.45	4187	336	2040	30720	3244	132	15.52	5980	388	2736	61440	3807	171	23.95	5424	1067	1617
16	16384	2668	521	1.97	4595	408	1927	32768	3286	42	48.76	6275	295	2989	65536	4029	222	18.45	5768	344	1739
17	17408	2809	141	7.26	5257	662	2448	34816	3406	120	17.07	6653	378	3247	69632	4528	499	8.21	5989	221	1461
18	18432	2894	85	12.05	5551	294	2657	36864	3457	51	40.16	6972	319	3515	73728	4691	163	25.13	6207	218	1516
19	19456	2940	46	22.26	6039	488	3099	38912	3577	120	17.07	7188	216	3611	77824	4828	137	29.90	6465	258	1637
20	20480	3009	69	14.84	6554	515	3545	40960	3663	86	23.81	7414	226	3751	81920	4968	140	29.26	6725	260	1757
21	21504	3163	154	6.65	6955	401	3792	43008	3766	103	19.88	7658	244	3892	86016	5136	168	24.38	7005	280	1869
22	22528	3226	63	16.25	8146	1191	4920	45056	3843	77	26.60	8258	600	4415	90112	5291	155	26.43	7317	312	2026
23	23552	3303	77	13.30	8584	438	5281	47104	3989	146	14.03	8591	333	4602	94208	5464	173	23.68	7775	458	2311
24	24576	3348	45	22.76	9030	446	5682	49152	4055	66	31.03	8869	278	4814	95833	5559	95	17.11	8030	255	2471
25	25600	3443	95	10.78	9338	308	5895	51200	4223	168	12.19	9153	284	4930							
26	26624	3559	116	8.83	9660	322	6101	53248	4670	447	4.58	9624	471	4954							
27	27648	3641	82	12.49	9974	314	6333	55296	4773	103	19.88	9878	254	5105							
28	28672	3697	56	18.29	10304	330	6607	57344	4821	48	42.67	10204	326	5383							
29	29696	3782	85	12.05	10659	355	6877	59392	4971	150	13.65	10565	361	5594							
30	30720	3880	98	10.45	11117	458	7237	61440	5031	60	34.13	10932	367	5901							
31	31744	3959	79	12.96	11426	309	7467	63488	5115	84	24.38	12151	1219	7036							
32	32768	4012	53	19.32	12267	841	8255	65536	5184	69	29.68	12301	150	7117							
33	33792	4474	462	2.22	12646	379	8172	67584	5349	165	12.41	12683	382	7334							
34	34816	4511	37	27.68	13058	412	8547	69632	5395	46	44.52	13074	391	7679							
35	35840	4586	75	13.65	13455	397	8869	71680	5509	114	17.96	13469	395	7960							
36	36864	4653	67	15.28	13875	420	9222	73728	5553	44	46.55	13878	409	8325							
37	37888	4806	153	6.69	14287	412	9481	75776	5703	150	13.65	14325	447	8622							
38	38912	4859	53	19.32	15167	880	10308	77824	5755	52	39.38	14718	393	8963							

Block Size	1024							2048							4096						
Block #	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)
39	39936	4911	52	19.69	15604	437	10693	79872	5856	101	20.28	15497	779	9641							
40	40960	4980	69	14.84	16417	813	11437	81920	5924	68	30.12	15950	453	10026							
41	41984	5118	138	7.42	16653	236	11535	83968	6414	490	4.18	16420	470	10006							
42	43008	5226	108	9.48	17213	560	11987	86016	6473	59	34.71	16915	495	10442							
43	44032	5314	88	11.64	17611	398	12297	88064	6581	108	18.96	17415	500	10834							
44	45056	5369	55	18.62	18453	842	13084	90112	6606	25	81.92	17898	483	11292							
45	46080	5472	103	9.94	18920	467	13448	92160	6705	99	20.69	18765	867	12060							
46	47104	6020	548	1.87	19421	501	13401	94208	6785	80	25.60	19260	495	12475							
47	48128	6113	93	11.01	19954	533	13841	95883	6841	56	29.91	19797	537	12956							
48	49152	6162	49	20.90	20458	504	14296														
49	50176	6327	165	6.21	21107	649	14780														
50	51200	6363	36	28.44	22005	898	15642														
51	52224	6430	67	15.28	22557	552	16127														
52	53248	6481	51	20.08	23139	582	16658														
53	54272	6583	102	10.04	23733	594	17150														
54	55296	6653	70	14.63	24312	579	17659														
55	56320	6701	48	21.33	25271	959	18570														
56	57344	6739	38	26.95	25898	627	19159														
57	58368	6817	78	13.13	26587	689	19770														
58	59392	6853	36	28.44	27221	634	20368														
59	60416	6906	53	19.32	27873	652	20967														
60	61440	6933	27	37.93	28861	988	21928														
61	62464	7041	108	9.48	29557	696	22516														
62	63488	7087	46	22.26	30207	650	23120														
63	64512	7113	26	39.38	30946	739	23833														
64	65536	7150	37	27.68	32076	1130	24926														
65	66560	7320	170	6.02	32796	720	25476														
66	67584	7350	30	34.13	33521	725	26171														
67	68608	7380	30	34.13	34253	732	26873														
68	69632	7420	40	25.60	36133	1880	28713														
69	70656	7480	60	17.07	36227	94	28747														
70	71680	7511	31	33.03	36962	735	29451														
71	72704	7547	36	28.44	37735	773	30188														
72	73728	7585	38	26.95	38889	1154	31304														
73	74752	7961	376	2.72	39700	811	31739														
74	75776	8009	48	21.33	40492	792	32483														
75	76800	8057	48	21.33	41364	872	33307														
76	77824	8091	34	30.12	42552	1188	34461														
77	78848	8180	89	11.51	43377	825	35197														
78	79872	8242	62	16.52	44228	851	35986														

Test Case 9583 Bytes Image File

Block Size	1024							2048							4096						
Block #	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)
79	80896	8290	48	21.33	45448	1220	37158														
80	81920	8326	36	28.44	47186	1738	38860														
81	82944	8418	92	11.13	47420	234	39002														
82	83968	8459	41	24.98	48302	882	39843														
83	84992	8492	33	31.03	49883	1581	41391														
84	86016	8529	37	27.68	50547	664	42018														
85	87040	8638	109	9.39	51539	992	42901														
86	88064	8683	45	22.76	52760	1221	44077														
87	89088	8729	46	22.26	53732	972	45003														
88	90112	8774	45	22.76	54647	915	45873														
89	91136	8843	69	14.84	55922	1275	47079														
90	92160	8890	47	21.79	56920	998	48030														
91	93184	8929	39	26.26	57869	949	48940														
92	94208	8984	55	18.62	59175	1306	50191														
93	95232	9010	26	39.38	60151	976	51141														
94	95833	9031	21	28.62	61308	1157	52277														
Average			95.063	17.896		645.347	18288.432			142.521	24.747		412.438	5077.667			222.360	20.908		321.200	844.080
Std Deriv			103.015	10.409		364.883	15529.149			158.993	14.771		278.076	3896.084			113.376	6.177		182.850	869.190

Test Case 1020249 Bytes Image File

Block Size	2048							4096						
Block #	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)
initiated		0			0		0		0			0		0
0	0	345	345		349	349	4	0	508	508		540	540	32
1	2048	739	394	5.20	2075	1726	1336	4096	1230	722	5.67	1243	703	13
2	4096	774	35	58.51	3280	1205	2506	8192	1414	184	22.26	1515	272	101
3	6144	908	134	15.28	5723	2443	4815	12288	1568	154	26.60	1702	187	134
4	8192	961	53	38.64	7103	1380	6142	16384	1771	203	20.18	1915	213	144
5	10240	1070	109	18.79	8447	1344	7377	20480	2175	404	10.14	2197	282	22
6	12288	1121	51	40.16	11005	2558	9884	24576	2349	174	23.54	2456	259	107
7	14336	1619	498	4.11	12496	1491	10877	28672	2518	169	24.24	2610	154	92
8	16384	1693	74	27.68	13940	1444	12247	32768	2739	221	18.53	3253	643	514
9	18432	1804	111	18.45	15344	1404	13540	36864	3015	276	14.84	3548	295	533
10	20480	1861	57	35.93	17868	2524	16007	40960	3552	537	7.63	3685	137	133
11	22528	1980	119	17.21	19496	1628	17516	45056	3704	152	26.95	4115	430	411
12	24576	2032	52	39.38	21158	1662	19126	49152	3843	139	29.47	4282	167	439
13	26624	2183	151	13.56	22711	1553	20528	53248	4046	203	20.18	4702	420	656
14	28672	2288	105	19.50	25104	2393	22816	57344	4208	162	25.28	5085	383	877
15	30720	2374	86	23.81	26709	1605	24335	61440	4810	602	6.80	5580	495	770
16	32768	2441	67	30.57	28310	1601	25869	65536	5030	220	18.62	6103	523	1073
17	34816	2578	137	14.95	30139	1829	27561	69632	5180	150	27.31	6501	398	1321
18	36864	2627	49	41.80	32665	2526	30038	73728	5351	171	23.95	7236	735	1885
19	38912	2712	85	24.09	34413	1748	31701	77824	5621	270	15.17	7537	301	1916
20	40960	2783	71	28.85	36114	1701	33331	81920	7283	1662	2.46	7832	295	549
21	43008	3189	406	5.04	38675	2561	35486	86016	7690	407	10.06	9052	1220	1362
22	45056	3244	55	37.24	41000	2325	37756	90112	7917	227	18.04	10840	1788	2923
23	47104	3431	187	10.95	42773	1773	39342	94208	8089	172	23.81	11275	435	3186
24	49152	3515	84	24.38	44644	1871	41129	98304	8282	193	21.22	11604	329	3322
25	51200	3651	136	15.06	46621	1977	42970	102400	8640	358	11.44	11994	390	3354
26	53248	3712	61	33.57	50975	4354	47263	106496	8716	76	53.89	12343	349	3627
27	55296	3795	83	24.67	52089	1114	48294	110592	8796	80	51.20	12680	337	3884
28	57344	3897	102	20.08	54746	2657	50849	114688	8888	92	44.52	13094	414	4206
29	59392	3976	79	25.92	57067	2321	53091	118784	8989	101	40.55	14012	918	5023
30	61440	4052	76	26.95	59564	2497	55512	122880	9206	217	18.88	14384	372	5178
31	63488	4188	136	15.06	60127	563	55939	126976	9284	78	52.51	14877	493	5593
32	65536	4241	53	38.64	61442	1315	57201	131072	9381	97	42.23	15282	405	5901
33	67584	4326	85	24.09	62778	1336	58452	135168	9458	77	53.19	16125	843	6667
34	69632	4402	76	26.95	63359	581	58957	139264	9684	226	18.12	16573	448	6889
35	71680	4525	123	16.65	64505	1146	59980	143360	10082	398	10.29	16863	290	6781
36	73728	4790	265	7.73	65955	1450	61165	147456	10128	46	89.04	17437	574	7309
37	75776	4855	65	31.51	66648	693	61793	151552	10210	82	49.95	17869	432	7659
38	77824	4892	37	55.35	67815	1167	62923	155648	10292	82	49.95	18306	437	8014
39	79872	4965	73	28.05	68635	820	63670	159744	10337	45	91.02	18753	447	8416
40	81920	5001	36	56.89	69875	1240	64874	163840	10545	208	19.69	19570	817	9025
41	83968	5078	77	26.60	70894	1019	65816	167936	10610	65	63.02	20089	519	9479
42	86016	5139	61	33.57	71807	913	66668	172032	10698	88	46.55	20619	530	9921
43	88064	5256	117	17.50	73146	1339	67890	176128	10780	82	49.95	21167	548	10387
44	90112	5336	80	25.60	74206	1060	68870	180224	10876	96	42.67	21735	568	10859
45	92160	5441	105	19.50	75130	924	69689	184320	13098	2222	1.84	22280	545	9182
46	94208	5474	33	62.06	76555	1425	71081	188416	14037	939	4.36	23268	988	9231
47	96256	5558	84	24.38	77461	906	71903	192512	14865	828	4.95	23844	576	8979
48	98304	5575	17	120.47	79282	1821	73707	196608	15316	451	9.08	24400	556	9084
49	100352	5644	69	29.68	80983	1701	75339	200704	16604	1288	3.18	24966	566	8362
50	102400	5706	62	33.03	82243	1260	76537	204800	19604	3000	1.37	26038	1072	6434
51	104448	6238	532	3.85	82617	374	76379	208896	20129	525	7.80	26683	645	6554
52	106496	6308	70	29.26	84086	1469	77778	212992	20694	565	7.25	27716	1033	7022
53	108544	6422	114	17.96	85234	1148	78812	217088	21738	1044	3.92	28361	645	6623
54	110592	6474	52	39.38	86676	1442	80202	221184	23246	1508	2.72	29003	642	5757
55	112640	6600	126	16.25	87721	1045	81121	225280	25648	2402	1.71	29658	655	4010
56	114688	6651	51	40.16	89002	1281	82351	229376	25747	99	41.37	30683	1025	4936
57	116736	6772	121	16.93	90366	1364	83594	233472	25848	101	40.55	31396	713	5548
58	118784	6830	58	35.31	91612	1246	84782	237568	25977	129	31.75	32107	711	6130
59	120832	6939	109	18.79	92776	1164	85837	241664	26999	1022	4.01	32867	760	5868
60	122880	7058	119	17.21	94445	1669	87387	245760	29680	2681	1.53	33905	1038	4225
61	124928	7220	162	12.64	95769	1324	88549	249856	30713	1033	3.97	34618	713	3905
62	126976	7292	72	28.44	98172	2403	90880	253952	31442	729	5.62	35324	706	3882
63	129024	7413	121	16.93	99681	1509	92268	258048	32869	1427	2.87	36025	701	3156
64	131072	7480	67	30.57	100068	387	92588	262144	33932	1063	3.85	37373	1348	3441
65	133120	7831	351	5.83	102615	2547	94784	266240	37561	3629	1.13	38356	983	795
66	135168	7893	62	33.03	103905	1290	96012	270336	38386	825	4.96	38901	545	515
67	137216	8028	135	15.17	105035	1130	97007	274432	39160	774	5.29	39877	976	717
68	139264	8075	47	43.57	107144	2109	99069	278528	39948	788	5.20	40968	1091	1020
69	141312	8217	142	14.42	108010	866	99793	282624	41883	1935	2.12	42232	1264	349
70	143360	8236	19	107.79	110473	2463	102237	286720	42935	1052	3.89	43136	904	201
71	145408	8385	149	13.74	110916	443	102531	290816	43219	284	14.42	43861	725	642

Test Case 1020249 Bytes Image File

Block Size	2048							4096						
Block #	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)
72	147456	8541	156	13.13	112787	1871	104246	294912	44029	810	5.06	44769	908	740
73	149504	8711	170	12.05	114822	2035	106111	299008	45632	1603	2.56	46002	1233	370
74	151552	8761	50	40.96	117039	2217	108278	303104	46737	1105	3.71	46919	917	182
75	153600	8908	147	13.93	117995	956	109087	307200	48289	1552	2.64	48320	1401	31
76	155648	8934	26	78.77	120426	2431	111492	311296	48414	125	32.77	49158	838	744
77	157696	9045	111	18.45	121967	1541	112922	315392	49293	879	4.66	49989	831	696
78	159744	9129	84	24.38	123865	1898	114736	319488	50341	1048	3.91	50987	998	646
79	161792	9556	427	4.80	125422	1557	115866	323584	52250	1909	2.15	52261	1274	11
80	163840	9638	82	24.98	126745	1323	117107	327680	53374	1124	3.64	53399	1138	25
81	165888	9747	109	18.79	129044	2299	119297	331776	53476	102	40.16	53712	313	236
82	167936	9801	54	37.93	129786	742	119985	335872	54401	925	4.43	55002	1290	601
83	169984	9915	114	17.96	131526	1740	121611	339968	55649	1248	3.28	56299	1297	650
84	172032	9973	58	35.31	133906	2380	123933	344064	56622	973	4.21	57298	999	676
85	174080	10098	125	16.38	135445	1539	125347	348160	58986	2364	1.73	58997	1699	11
86	176128	10151	53	38.64	136705	1260	126554	352256	59085	99	41.37	59720	723	635
87	178176	10260	109	18.79	137916	1211	127656	356352	60093	1008	4.06	60591	871	498
88	180224	10305	45	45.51	140713	2797	130408	360448	61079	986	4.15	62047	1456	968
89	182272	10435	130	15.75	142625	1912	132190	364544	62441	1362	3.01	63326	1279	885
90	184320	10485	50	40.96	143166	541	132681	368640	64479	2038	2.01	64508	1182	29
91	186368	10629	144	14.22	144666	1500	134037	372736	64589	110	37.24	65220	712	631
92	188416	10684	55	37.24	146466	1800	135782	376832	66538	1949	2.10	67327	2107	789
93	190464	10784	100	20.48	148075	1609	137291	380928	67537	999	4.10	68545	1218	1008
94	192512	10832	48	42.67	149925	1850	139093	385024	68630	1093	3.75	69926	1381	1296
95	194560	11283	451	4.54	151220	1295	139937	389120	71259	2629	1.56	71299	1373	40
96	196608	11354	71	28.85	153178	1958	141824	393216	71368	109	37.58	72382	1083	1014
97	198656	11492	138	14.84	154965	1787	143473	397312	72424	1056	3.88	74058	1676	1634
98	200704	11538	46	44.52	156726	1761	145188	401408	74136	1712	2.39	75253	1195	1117
99	202752	11662	124	16.52	158423	1697	146761	405504	75589	1453	2.82	76498	1245	909
100	204800	11722	60	34.13	160206	1783	148484	409600	78236	2647	1.55	78257	1759	21
101	206848	11848	126	16.25	161736	1530	149888	413696	78337	101	40.55	79157	900	820
102	208896	11884	36	56.89	163629	1893	151745	417792	79915	1578	2.60	80547	1390	632
103	210944	12016	132	15.52	165373	1744	153357	421888	81126	1211	3.38	82617	2070	1491
104	212992	12064	48	42.67	167405	2032	155341	425984	82706	1580	2.59	83818	1201	1112
105	215040	12198	134	15.28	170285	2880	158087	430080	85281	2575	1.59	85291	1473	10
106	217088	12260	62	33.03	171585	1300	159325	434176	85364	83	49.35	86357	1066	993
107	219136	12371	111	18.45	172668	1083	160297	438272	86948	1584	2.59	87908	1551	960
108	221184	12444	73	28.05	174567	1899	162123	442368	88289	1341	3.05	89718	1810	1429
109	223232	12908	464	4.41	176486	1919	163578	446464	89894	1605	2.55	91026	1308	1132
110	225280	12962	54	37.93	178996	2510	166034	450560	92530	2636	1.55	92567	1541	37
111	227328	13073	111	18.45	180704	1708	167631	454656	92687	157	26.09	93678	1111	991
112	229376	13142	69	29.68	182215	1511	169073	458752	94360	1673	2.45	95669	1991	1309
113	231424	13240	98	20.90	184244	2029	171004	462848	95697	1337	3.06	97347	1678	1650
114	233472	13328	88	23.27	186024	1780	172696	466944	97377	1680	2.44	98438	1091	1061
115	235520	13494	166	12.34	188052	2028	174558	471040	100517	3140	1.30	101130	2692	613
116	237568	13531	37	55.35	189765	1713	176234	475136	100613	96	42.67	102532	1402	1919
117	239616	13654	123	16.65	191687	1922	178033	479232	100742	129	31.75	104018	1486	3276
118	241664	13722	68	30.12	193625	1938	179903	483328	100869	127	32.25	105257	1239	4388
119	243712	13848	126	16.25	195654	2029	181806	487424	101115	246	16.65	107270	2013	6155
120	245760	13892	44	46.55	197664	2010	183772	491520	104338	3223	1.27	108438	1168	4100
121	247808	14044	152	13.47	200230	2566	186186	495616	105869	1531	2.68	110408	1970	4539
122	249856	14141	97	21.11	202522	2292	188381	499712	107308	1439	2.85	111268	860	3960
123	251904	14271	130	15.75	204543	2021	190272	503808	107429	121	33.85	113358	2090	5929
124	253952	14619	348	5.89	206025	1482	191406	507904	109884	2455	1.67	115067	1709	5183
125	256000	14739	120	17.07	207816	1791	193077	512000	115175	5291	0.77	116268	1201	1093
126	258048	14792	53	38.64	209887	2071	195095	516096	115649	474	8.64	118589	2321	2940
127	260096	14917	125	16.38	212209	2322	197292	520192	117108	1459	2.81	120352	1763	3244
128	262144	14971	54	37.93	214216	2007	199245	524288	118996	1888	2.17	121667	1315	2671
129	264192	15099	128	16.00	216247	2031	201148	528384	120476	1480	2.77	123406	1739	2930
130	266240	15133	34	60.24	218323	2076	203190	532480	124325	3849	1.06	125087	1681	762
131	268288	15252	119	17.21	220675	2352	205423	536576	125767	1442	2.84	127273	2186	1506
132	270336	15301	49	41.80	224443	3768	209142	540672	127314	1547	2.65	128633	1360	1319
133	272384	15454	153	13.39	224797	354	209343	544768	129249	1935	2.12	130097	1464	848
134	274432	15511	57	35.93	227010	2213	211499	548864	129376	127	32.25	132928	2831	3552
135	276480	15648	137	14.95	230827	3817	215179	552960	134681	5305	0.77	134693	1765	12
136	278528	15693	45	45.51	231678	851	215985	557056	134832	151	27.13	136925	2232	2093
137	280576	15811	118	17.36	234107	2429	218296	561152	136952	2120	1.93	138166	1241	1214
138	282624	15851	40	51.20	236132	2025	220281	565248	137114	162	25.28	139427	1261	2313
139	284672	16194	343	5.97	238306	2174	222112	569344	138748	1634	2.51	141297	1870	2549
140	286720	16246	52	39.38	240524	2218	224278	573440	144338	5590	0.73	144375	3078	37
141	288768	16346	100	20.48	242445	1921	226099	577536	144463	125	32.77	145278	903	815
142	290816	16418	72	28.44	244649	2204	228231	581632	146100	1637	2.50	146807	1529	707
143	292864	16486	68	30.12	247006	2357	230520	585728	146229	129	31.75	148918	2111	2689
144	294912	16572	86	23.81	249176	2170	232604	589824	148242	2013	2.03	150877	1959	2635

Test Case 1020249 Bytes Image File

Block Size	2048							4096						
Block #	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)
145	296960	16682	110	18.62	251553	2377	234871	593920	153642	5400	0.76	153652	2775	10
146	299008	16761	79	25.92	253766	2213	237005	598016	153747	105	39.01	154682	1030	935
147	301056	16869	108	18.96	255956	2190	239087	602112	155615	1868	2.19	156798	2116	1183
148	303104	16916	47	43.57	258787	2831	241871	606208	155848	233	17.58	159123	2325	3275
149	305152	17049	133	15.40	261363	2576	244314	610304	158646	2798	1.46	160202	1079	1556
150	307200	17095	46	44.52	263237	1874	246142	614400	163624	4978	0.82	163676	3474	52
151	309248	17206	111	18.45	266085	2848	248879	618496	163770	146	28.05	165911	2235	2141
152	311296	17256	50	40.96	268485	2400	251229	622592	165618	1848	2.22	167871	1960	2253
153	313344	17699	443	4.62	270535	2050	252836	626688	166061	443	9.25	168763	892	2702
154	315392	17725	26	78.77	273307	2772	255582	630784	167943	1882	2.18	171038	2275	3095
155	317440	17880	155	13.21	275588	2281	257708	634880	172052	4109	1.00	172897	1859	845
156	319488	17972	92	22.26	278594	3006	260622	638976	174093	2041	2.01	175127	2230	1034
157	321536	18142	170	12.05	280545	1951	262403	643072	174830	737	5.56	177918	2791	3088
158	323584	18192	50	40.96	283245	2700	265053	647168	176296	1466	2.79	178817	899	2521
159	325632	18430	238	8.61	285605	2360	267175	651264	178189	1893	2.16	180997	2180	2808
160	327680	18501	71	28.85	287745	2140	269244	655360	182513	4324	0.95	183287	2290	774
161	329728	18592	91	22.51	291062	3317	272470	659456	184449	1936	2.12	185438	2151	989
162	331776	18671	79	25.92	293214	2152	274543	663552	185763	1314	3.12	188006	2568	2243
163	333824	18798	127	16.13	295956	2742	277158	667648	186738	975	4.20	190078	2072	3340
164	335872	18858	60	34.13	298198	2242	279340	671744	186888	150	27.31	193328	3250	6440
165	337920	18979	121	16.93	300865	2667	281886	675840	189047	2159	1.90	195714	2386	6667
166	339968	18995	16	128.00	303495	2630	284500	679936	191292	2245	1.82	198065	2351	6773
167	342016	19414	419	4.89	305515	2020	286101	684032	193312	2020	2.03	200207	2142	6895
168	344064	19450	36	56.89	307928	2413	288478	688128	193468	156	26.26	201239	1032	7771
169	346112	19607	157	13.04	310580	2652	290973	692224	195927	2459	1.67	203785	2546	7858
170	348160	19700	93	22.02	313205	2625	293505	696320	200383	4456	0.92	206281	2496	5898
171	350208	19827	127	16.13	315725	2520	295898	700416	201085	702	5.83	207780	1499	6695
172	352256	19883	56	36.57	318228	2503	298345	704512	202809	1724	2.38	210119	2339	7310
173	354304	20001	118	17.36	321484	3256	301483	708608	205065	2256	1.82	212628	2509	7563
174	356352	20059	58	35.31	323805	2321	303746	712704	207017	1952	2.10	214382	1754	7365
175	358400	20169	110	18.62	326544	2739	306375	716800	211733	4716	0.87	217630	3248	5897
176	360448	20222	53	38.64	328976	2432	308754	720896	213703	1970	2.08	219347	1717	5644
177	362496	20340	118	17.36	332755	3779	312415	724992	213854	151	27.13	222864	3517	9010
178	364544	20401	61	33.57	335388	2633	314987	729088	216195	2341	1.75	225269	2405	9074
179	366592	20529	128	16.00	337995	2607	317466	733184	217888	1693	2.42	227900	2631	10012
180	368640	20565	36	56.89	341128	3133	320563	737280	223173	5285	0.78	228983	1083	5810
181	370688	20698	133	15.40	343684	2556	322986	741376	225279	2106	1.94	232691	3708	7412
182	372736	20733	35	58.51	346150	2466	325417	745472	225454	175	23.41	235249	2558	9795
183	374784	21240	507	4.04	351436	5286	330196	749568	227954	2500	1.64	236437	1188	8483
184	376832	21303	63	32.51	352094	658	330791	753664	228158	204	20.08	239966	3529	11808
185	378880	21450	147	13.93	354985	2891	333535	757760	235313	7155	0.57	241090	1124	5777
186	380928	21525	75	27.31	357706	2721	336181	761856	235407	94	43.57	243224	2134	7817
187	382976	21634	109	18.79	360586	2880	338952	765952	237878	2471	1.66	245959	2735	8081
188	385024	21703	69	29.68	363201	2615	341498	770048	239986	2108	1.94	248627	2668	8641
189	387072	21823	120	17.07	366158	2957	344335	774144	240134	148	27.68	251077	2450	10943
190	389120	21885	62	33.03	369077	2919	347192	778240	245107	4973	0.82	255265	4188	10158
191	391168	22009	124	16.52	371945	2868	349936	782336	247531	2424	1.69	258267	3002	10736
192	393216	22075	66	31.03	374496	2551	352421	786432	247769	238	17.21	261335	3068	13566
193	395264	22189	114	17.96	377525	3029	355336	790528	249810	2041	2.01	264281	2946	14471
194	397312	22225	36	56.89	381772	4247	359547	794624	252477	2667	1.54	266378	2097	13901
195	399360	22358	133	15.40	383865	2093	361507	798720	258468	5991	0.68	269899	3521	11431
196	401408	22418	60	34.13	386211	2346	363793	802816	259202	734	5.58	271704	1805	12502
197	403456	22919	501	4.09	390772	4561	367853	806912	261669	2467	1.66	275580	3876	13911
198	405504	22997	78	26.26	392128	1356	369131	811008	262237	568	7.21	278078	2498	15841
199	407552	23095	98	20.90	395355	3227	372260	815104	265044	2807	1.46	279428	1350	14384
200	409600	23155	60	34.13	398584	3229	375429	819200	270145	5101	0.80	283774	4346	13629
201	411648	23255	100	20.48	401195	2611	377940	823296	271535	1390	2.95	286640	2866	15105
202	413696	23323	68	30.12	404652	3457	381329	827392	272950	1415	2.89	289316	2676	16366
203	415744	23502	179	11.44	406875	2223	383373	831488	275642	2692	1.52	292872	3556	17230
204	417792	23570	68	30.12	410472	3597	386902	835584	275770	128	32.00	294269	1397	18499
205	419840	23665	95	21.56	413742	3270	390077	839680	281083	5313	0.77	297118	2849	16035
206	421888	23735	70	29.26	416275	2533	392540	843776	283810	2727	1.50	299558	2440	15748
207	423936	23861	126	16.25	419713	3438	395852	847872	283962	152	26.95	302268	2710	18306
208	425984	23924	63	32.51	422355	2642	398431	851968	285768	1806	2.27	307082	4814	21314
209	428032	24015	91	22.51	425495	3140	401480	856064	286861	1093	3.75	308246	1164	21385
210	430080	24091	76	26.95	427997	2502	403906	860160	293319	6458	0.63	311350	3104	18031
211	432128	24202	111	18.45	430971	2974	406769	864256	293471	152	26.95	313738	2388	20267
212	434176	24579	377	5.43	434933	3962	410354	868352	296041	2570	1.59	316408	2670	20367
213	436224	24687	108	18.96	437475	2542	412788	872448	296607	566	7.24	319081	2673	22474
214	438272	24759	72	28.44	440895	3420	416136	876544	298912	2305	1.78	321847	2766	22935
215	440320	24854	95	21.56	444035	3140	419181	880640	304029	5117	0.80	326898	5051	22869
216	442368	24918	64	32.00	447325	3290	422407	884736	304888	859	4.77	327721	823	22833
217	444416	25050	132	15.52	450095	2770	425045	888832	306770	1882	2.18	330243	2522	23473

Test Case 1020249 Bytes Image File

Block Size	2048							4096						
Block #	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)
218	446464	25092	42	48.76	453205	3110	428113	892928	306903	133	30.80	332989	2746	26086
219	448512	25243	151	13.56	456816	3611	431573	897024	307019	116	35.31	337637	4648	30618
220	450560	25351	108	18.96	460045	3229	434694	901120	309967	2948	1.39	338840	1203	28873
221	452608	25455	104	19.69	462689	2644	437234	905216	312817	2850	1.44	341984	3144	29167
222	454656	25513	58	35.31	465885	3196	440372	909312	313019	202	20.28	345157	3173	32138
223	456704	25643	130	15.75	469571	3686	443928	913408	315398	2379	1.72	347785	2628	32387
224	458752	25700	57	35.93	473372	3801	447672	917504	315537	139	29.47	350837	3052	35300
225	460800	25812	112	18.29	476346	2974	450534	921600	321227	5690	0.72	353617	2780	32390
226	462848	25879	67	30.57	479515	3169	453636	925696	322073	846	4.84	356685	3068	34612
227	464896	26318	439	4.67	482865	3350	456547	929792	324133	2060	1.99	359987	3302	35854
228	466944	26363	45	45.51	486269	3404	459906	933888	326924	2791	1.47	362358	2371	35434
229	468992	26488	125	16.38	489435	3166	462947	937984	327074	150	27.31	364782	2424	37708
230	471040	26540	52	39.38	492435	3000	465895	942080	332426	5352	0.77	367967	3185	35541
231	473088	26669	129	15.88	495955	3520	469286	946176	333575	1149	3.56	371077	3110	37502
232	475136	26738	69	29.68	499924	3969	473186	950272	335379	1804	2.27	374397	3320	39018
233	477184	26917	179	11.44	502816	2892	475899	954368	338211	2832	1.45	376818	2421	38607
234	479232	26990	73	28.05	506205	3389	479215	958464	338329	118	34.71	379605	2787	41276
235	481280	27098	108	18.96	509945	3740	482847	962560	343916	5587	0.73	382101	2496	38185
236	483328	27168	70	29.26	513255	3310	486087	966656	344332	416	9.85	385072	2971	40740
237	485376	27255	87	23.54	519505	6250	492250	970752	346838	2506	1.63	388069	2997	41231
238	487424	27341	86	23.81	520146	641	492805	974848	347541	703	5.83	390981	2912	43440
239	489472	27442	101	20.28	523428	3282	495986	978944	349869	2328	1.76	393922	2941	44053
240	491520	27502	60	34.13	528689	5261	501187	983040	355645	5776	0.71	396863	2941	41218
241	493568	27935	433	4.73	532482	3793	504547	987136	355772	127	32.25	399892	3029	44120
242	495616	28002	67	30.57	535775	3293	507773	991232	358316	2544	1.61	403727	3835	45411
243	497664	28120	118	17.36	539056	3281	510936	995328	358632	316	12.96	406916	3189	48284
244	499712	28182	62	33.03	542506	3450	514324	999424	361272	2640	1.55	409197	2281	47925
245	501760	28280	98	20.90	546665	4159	518385	1003520	367195	5923	0.69	411886	2689	44691
246	503808	28355	75	27.31	550436	3771	522081	1007616	367286	91	45.01	414920	3034	47634
247	505856	28609	254	8.06	553806	3370	525197	1011712	369998	2712	1.51	418025	3105	48027
248	507904	28677	68	30.12	557796	3990	529119	1015808	370358	360	11.38	420944	2919	50586
249	509952	28778	101	20.28	561902	4106	533124	1019904	373382	3024	1.35	424020	3076	50638
250	512000	28848	70	29.26	564989	3087	536141	1020249	373426	44	7.84	427047	3027	53621
251	514048	28952	104	19.69	568546	3557	539594							
252	516096	29000	48	42.67	572286	3740	543286							
253	518144	29137	137	14.95	575935	3649	546798							
254	520192	29173	36	56.89	579745	3810	550572							
255	522240	29593	420	4.88	583395	3650	553802							
256	524288	29640	47	43.57	587076	3681	557436							
257	526336	29730	90	22.76	591738	4662	562008							
258	528384	29809	79	25.92	595202	3464	565393							
259	530432	29923	114	17.96	598825	3623	568902							
260	532480	29965	42	48.76	602375	3550	572410							
261	534528	30100	135	15.17	606215	3840	576115							
262	536576	30169	69	29.68	609876	3661	579707							
263	538624	30358	189	10.84	613525	3649	583167							
264	540672	30394	36	56.89	617605	4080	587211							
265	542720	30542	148	13.84	621295	3690	590753							
266	544768	30591	49	41.80	624875	3580	594284							
267	546816	30727	136	15.06	628556	3681	597829							
268	548864	30764	37	55.35	632455	3899	601691							
269	550912	30882	118	17.36	636068	3613	605186							
270	552960	30965	83	24.67	640395	4327	609430							
271	555008	31391	426	4.81	643975	3580	612584							
272	557056	31571	180	11.38	647596	3621	616025							
273	559104	31663	92	22.26	651547	3951	619884							
274	561152	31696	33	62.06	655438	3891	623742							
275	563200	31822	126	16.25	659975	4537	628153							
276	565248	31891	69	29.68	663648	3673	631757							
277	567296	31999	108	18.96	667815	4167	635816							
278	569344	32088	89	23.01	671525	3710	639437							
279	571392	32207	119	17.21	675464	3939	643257							
280	573440	32243	36	56.89	679902	4438	647659							
281	575488	32353	110	18.62	684002	4100	651649							
282	577536	32427	74	27.68	687922	3920	655495							
283	579584	32537	110	18.62	691295	3373	658758							
284	581632	32611	74	27.68	695495	4200	662884							
285	583680	33094	483	4.24	699336	3841	666242							
286	585728	33171	77	26.60	703535	4199	670364							
287	587776	33289	118	17.36	707177	3642	673888							
288	589824	33316	27	75.85	711895	4718	678579							
289	591872	33494	178	11.51	716399	4504	682905							
290	593920	33550	56	36.57	723404	7005	689854							

Block Size	2048							4096						
Block #	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)
291	595968	33674	124	16.52	724086	682	690412							
292	598016	33732	58	35.31	729234	5148	695502							
293	600064	33851	119	17.21	732396	3162	698545							
294	602112	33895	44	46.55	736126	3730	702231							
295	604160	34026	131	15.63	741012	4886	706986							
296	606208	34083	57	35.93	744616	3604	710533							
297	608256	34218	135	15.17	749336	4720	715118							
298	610304	34255	37	55.35	753752	4416	719497							
299	612352	34752	497	4.12	757277	3525	722525							
300	614400	34823	71	28.85	761376	4099	726553							
301	616448	34905	82	24.98	765885	4509	730980							
302	618496	34992	87	23.54	770436	4551	735444							
303	620544	35089	97	21.11	773956	3520	738867							
304	622592	35172	83	24.67	778506	4550	743334							
305	624640	35280	108	18.96	783190	4684	747910							
306	626688	35345	65	31.51	786935	3745	751590							
307	628736	35477	132	15.52	791067	4132	755590							
308	630784	35496	19	107.79	795255	4188	759759							
309	632832	35643	147	13.93	799862	4607	764219							
310	634880	35719	76	26.95	804142	4280	768423							
311	636928	35814	95	21.56	808666	4524	772852							
312	638976	35883	69	29.68	813274	4608	777391							
313	641024	35999	116	17.66	817235	3961	781236							
314	643072	36413	414	4.95	821536	4301	785123							
315	645120	36547	134	15.28	825655	4119	789108							
316	647168	36596	49	41.80	830754	5099	794158							
317	649216	36716	120	17.07	835444	4690	798728							
318	651264	36772	56	36.57	839605	4161	802833							
319	653312	36896	124	16.52	843656	4051	806760							
320	655360	36959	63	32.51	847875	4219	810916							
321	657408	37068	109	18.79	852475	4600	815407							
322	659456	37123	55	37.24	856885	4410	819762							
323	661504	37238	115	17.81	861582	4697	824344							
324	663552	37285	47	43.57	865715	4133	828430							
325	665600	37427	142	14.42	870485	4770	833058							
326	667648	37443	16	128.00	875362	4877	837919							
327	669696	37586	143	14.32	880008	4646	842422							
328	671744	37641	55	37.24	883891	3883	846250							
329	673792	37844	203	10.09	888753	4862	850909							
330	675840	37881	37	55.35	893575	4822	855694							
331	677888	38002	121	16.93	897535	3960	859533							
332	679936	38064	62	33.03	901955	4420	863891							
333	681984	38164	100	20.48	906695	4740	868531							
334	684032	38231	67	30.57	911335	4640	873104							
335	686080	38296	65	31.51	916004	4669	877708							
336	688128	38333	37	55.35	920625	4621	882292							
337	690176	38459	126	16.25	924768	4143	886309							
338	692224	38475	16	128.00	930485	5717	892010							
339	694272	38587	112	18.29	934385	3900	895798							
340	696320	38621	34	60.24	939245	4860	900624							
341	698368	38782	161	12.72	943832	4587	905050							
342	700416	38864	82	24.98	948356	4524	909492							
343	702464	39300	436	4.70	952965	4609	913665							
344	704512	39379	79	25.92	957377	4412	917998							
345	706560	39475	96	21.33	962475	5098	923000							
346	708608	39532	57	35.93	970984	8509	931452							
347	710656	39634	102	20.08	971510	526	931876							
348	712704	39698	64	32.00	977425	5915	937727							
349	714752	39795	97	21.11	982102	4677	942307							
350	716800	39888	93	22.02	986823	4721	946935							
351	718848	39974	86	23.81	991372	4549	951398							
352	720896	40032	58	35.31	996116	4744	956084							
353	722944	40155	123	16.65	1000356	4240	960201							
354	724992	40220	65	31.51	1005288	4932	965068							
355	727040	40326	106	19.32	1011442	6154	971116							
356	729088	40382	56	36.57	1015439	3997	975057							
357	731136	40494	112	18.29	1020746	5307	980252							
358	733184	40556	62	33.03	1025095	4349	984539							
359	735232	41013	457	4.48	1029793	4698	988780							
360	737280	41128	115	17.81	1034415	4622	993287							
361	739328	41254	126	16.25	1039400	4985	998146							
362	741376	41319	65	31.51	1045328	5928	1004009							
363	743424	41441	122	16.79	1050552	5224	1009111							

Block Size	2048							4096						
Block #	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)
364	745472	41492	51	40.16	1054876	4324	1013384							
365	747520	41610	118	17.36	1059556	4680	1017946							
366	749568	41676	66	31.03	1064346	4790	1022670							
367	751616	41774	98	20.90	1069575	5229	1027801							
368	753664	41853	79	25.92	1074506	4931	1032653							
369	755712	41970	117	17.50	1079383	4877	1037413							
370	757760	42031	61	33.57	1084125	4742	1042094							
371	759808	42106	75	27.31	1089319	5194	1047213							
372	761856	42199	93	22.02	1094291	4972	1052092							
373	763904	42609	410	5.00	1100116	5825	1057507							
374	765952	42681	72	28.44	1105046	4930	1062365							
375	768000	42821	140	14.63	1110027	4981	1067206							
376	770048	42885	64	32.00	1115146	5119	1072261							
377	772096	43044	159	12.88	1119900	4754	1076856							
378	774144	43113	69	29.68	1125269	5369	1082156							
379	776192	43222	109	18.79	1131149	5880	1087927							
380	778240	43297	75	27.31	1136496	5347	1093199							
381	780288	43416	119	17.21	1141160	4664	1097744							
382	782336	43511	95	21.56	1146282	5122	1102771							
383	784384	43605	94	21.79	1151566	5284	1107961							
384	786432	43671	66	31.03	1156737	5171	1113066							
385	788480	43784	113	18.12	1162647	5910	1118863							
386	790528	43851	67	30.57	1167936	5289	1124085							
387	792576	44251	400	5.12	1172631	4695	1128380							
388	794624	44313	62	33.03	1178706	6075	1134393							
389	796672	44448	135	15.17	1183377	4671	1138929							
390	798720	44485	37	55.35	1189287	5910	1144802							
391	800768	44662	177	11.57	1194822	5535	1150160							
392	802816	44722	60	34.13	1199946	5124	1155224							
393	804864	44917	195	10.50	1205486	5540	1160569							
394	806912	44977	60	34.13	1210250	4764	1165273							
395	808960	45063	86	23.81	1215883	5633	1170820							
396	811008	45150	87	23.54	1221489	5606	1176339							
397	813056	45257	107	19.14	1227056	5567	1181799							
398	815104	45321	64	32.00	1232239	5183	1186918							
399	817152	45414	93	22.02	1237490	5251	1192076							
400	819200	45482	68	30.12	1243312	5822	1197830							
401	821248	45607	125	16.38	1248917	5605	1203310							
402	823296	45645	38	53.89	1254739	5822	1209094							
403	825344	46014	369	5.55	1259884	5145	1213870							
404	827392	46070	56	36.57	1265559	5675	1219489							
405	829440	46221	151	13.56	1270804	5245	1224583							
406	831488	46282	61	33.57	1276473	5669	1230191							
407	833536	46372	90	22.76	1282095	5622	1235723							
408	835584	46442	70	29.26	1287884	5789	1241442							
409	837632	46566	124	16.52	1293026	5142	1246460							
410	839680	46638	72	28.44	1298445	5419	1251807							
411	841728	46768	130	15.75	1303666	5221	1256898							
412	843776	46824	56	36.57	1309884	6218	1263060							
413	845824	46973	149	13.74	1314886	5002	1267913							
414	847872	47039	66	31.03	1325526	10640	1278487							
415	849920	47155	116	17.66	1326306	780	1279151							
416	851968	47203	48	42.67	1331806	5500	1284603							
417	854016	47649	446	4.59	1336707	4901	1289058							
418	856064	47676	27	75.85	1343122	6415	1295446							
419	858112	47827	151	13.56	1347875	4753	1300048							
420	860160	47889	62	33.03	1353150	5275	1305261							
421	862208	48002	113	18.12	1358965	5815	1310963							
422	864256	48054	52	39.38	1364362	5397	1316308							
423	866304	48192	138	14.84	1370546	6184	1322354							
424	868352	48250	58	35.31	1376406	5860	1328156							
425	870400	48383	133	15.40	1381239	4833	1332856							
426	872448	48442	59	34.71	1386974	5735	1338532							
427	874496	48562	120	17.07	1392231	5257	1343669							
428	876544	48638	76	26.95	1397599	5368	1348961							
429	878592	48730	92	22.26	1403596	5997	1354866							
430	880640	48818	88	23.27	1409346	5750	1360528							
431	882688	49816	998	2.05	1414743	5397	1364927							
432	884736	51007	1191	1.72	1420914	6171	1369907							
433	886784	51245	238	8.61	1425946	5032	1374701							
434	888832	51354	109	18.79	1432066	6120	1380712							
435	890880	51510	156	13.13	1437278	5212	1385768							
436	892928	51606	96	21.33	1443301	6023	1391695							

Test Case 1020249 Bytes Image File

Block Size	2048							4096						
Block #	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)	kb transferred	at Service (ms)	Delta (ms)	Speed (kb/s)	at GUI (ms)	Delta (ms)	Lag (ms)
437	894976	51760	154	13.30	1449396	6095	1397636							
438	897024	51824	64	32.00	1454476	5080	1402652							
439	899072	51982	158	12.96	1460908	6432	1408926							
440	901120	52765	783	2.62	1466066	5158	1413301							
441	903168	53127	362	5.66	1471911	5845	1418784							
442	905216	53212	85	24.09	1477560	5649	1424348							
443	907264	53447	235	8.71	1483521	5961	1430074							
444	909312	53503	56	36.57	1489460	5939	1435957							
445	911360	53624	121	16.93	1495516	6056	1441892							
446	913408	53672	48	42.67	1501515	5999	1447843							
447	915456	53792	120	17.07	1506616	5101	1452824							
448	917504	54123	331	6.19	1512553	5937	1458430							
449	919552	54232	109	18.79	1518894	6341	1464662							
450	921600	54291	59	34.71	1524190	5296	1469899							
451	923648	54400	109	18.79	1530091	5901	1475691							
452	925696	54463	63	32.51	1535728	5637	1481265							
453	927744	54580	117	17.50	1541882	6154	1487302							
454	929792	54623	43	47.63	1547926	6044	1493303							
455	931840	54741	118	17.36	1554289	6363	1499548							
456	933888	55873	1132	1.81	1560275	5986	1504402							
457	935936	55954	81	25.28	1565812	5537	1509858							
458	937984	55992	38	53.89	1571830	6018	1515838							
459	940032	56080	88	23.27	1577888	6058	1521808							
460	942080	56096	16	128.00	1584222	6334	1528126							
461	944128	56186	90	22.76	1589698	5476	1533512							
462	946176	56232	46	44.52	1595346	5648	1539114							
463	948224	56541	309	6.63	1600718	5372	1544177							
464	950272	56618	77	26.60	1606724	6006	1550106							
465	952320	57190	572	3.58	1612908	6184	1555718							
466	954368	57238	48	42.67	1618486	5578	1561248							
467	956416	57307	69	29.68	1623860	5374	1566553							
468	958464	57334	27	75.85	1629638	5778	1572304							
469	960512	57414	80	25.60	1635450	5812	1578036							
470	962560	57493	79	25.92	1641930	6480	1584437							
471	964608	57525	32	64.00	1653900	11970	1596375							
472	966656	57541	16	128.00	1654736	836	1597195							
473	968704	57658	117	17.50	1660577	5841	1602919							
474	970752	57674	16	128.00	1666768	6191	1609094							
475	972800	57722	48	42.67	1673626	6858	1615904							
476	974848	57739	17	120.47	1679366	5740	1621627							
477	976896	57768	29	70.62	1685675	6309	1627907							
478	978944	57784	16	128.00	1691657	5982	1633873							
479	980992	57818	34	60.24	1698140	6483	1640322							
480	983040	57834	16	128.00	1705246	7106	1647412							
481	985088	60572	2738	0.75	1711449	6203	1650877							
482	987136	60594	22	93.09	1717447	5998	1656853							
483	989184	61341	747	2.74	1723931	6484	1662590							
484	991232	61872	531	3.86	1730284	6353	1668412							
485	993280	62803	931	2.20	1736977	6693	1674174							
486	995328	62839	36	56.89	1743514	6537	1680675							
487	997376	63692	853	2.40	1749744	6230	1686052							
488	999424	63860	168	12.19	1756596	6852	1692736							
489	1001472	64660	800	2.56	1762894	6298	1698234							
490	1003520	64795	135	15.17	1769076	6182	1704281							
491	1005568	64893	98	20.90	1775649	6573	1710756							
492	1007616	66121	1228	1.67	1782131	6482	1716010							
493	1009664	66186	65	31.51	1789137	7006	1722951							
494	1011712	67118	932	2.20	1795787	6650	1728669							
495	1013760	67179	61	33.57	1802206	6419	1735027							
496	1015808	68113	934	2.19	1808039	5833	1739926							
497	1017856	68188	75	27.31	1814793	6754	1746605							
498	1019904	68204	16	128.00	1821710	6917	1753506							
499	1021952	68775	571	3.59	1827886	6176	1759111							
Average			233.579	32.073		5913.958	1479842.726			1487.753	12.726		1701.382	9642.753
Std Deriv			392.935	32.505		1204.687	156221.401			1526.210	16.352		1706.028	9681.196

Bibliography

[art03] Strategy analytics: Camera phones outsell digital still cameras worldwide; nec, panasonic and nokia lead 25 million unit market. *Business Wire*, September 2003.

[BLB07] Petros Belimpasakis, Juha-Pekka Luoma, and Mihaly Börzsei. Content sharing middleware for mobile devices. In *MOBILWARE '08: Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, pages 1–8, ICST, Brussels, Belgium, Belgium, 2007. Nokia Research Center, Tampere, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[Clo] Johan Cloetens. Cyrket - onair (wifi disk). Online. Cited: Jul 13 2009, URL: `http://www.cyrket.com/package/com.bw.onair`.

[CSA09] Diana Cionoiu and Peter Saint-Andre. Jingle Early Media. XEP-0269 (Experimental Standard), May 2009. Version 0.1, URL: `http://xmpp.org/extensions/xep-0269.html`.

[DDD07] C. Daboo, B. Desruisseaux, and L. Dusseault. Calendaring Extensions to WebDAV (CalDAV). RFC 4791 (Proposed Standard), March 2007. URL: `http://www.ietf.org/rfc/rfc4791.txt`.

[DS09] Benjamin Söllner Dirk Hering Alexander Schill Daniel Schuster, Thomas Springer. Mobilisbuddy - integration sozialer netzwerke in umgebungs-basierte dienste auf mobilen endgeräten. In *Proceedings of GeNeMe 2009*. Gemeinschaft für Neue Medien, TUD Print, 2009.

[Dus07] L. Dusseault. HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV). RFC 4918 (Proposed Standard), June 2007. URL: `http://www.ietf.org/rfc/rfc4918.txt`.

[EHM+07] Ryan Eatmon, Joe Hildebrand, Jeremie Miller, Thomas Muldowney, and Peter Saint-Andre. Data Forms. XEP-0004 (Final Standard), September 2007. Version 2.9, URL: `http://xmpp.org/extensions/xep-0004.html`.

[Gdh07] J. Gregorio and B. de hOra. The Atom Publishing Protocol. RFC 5023 (Proposed Standard), October 2007. URL: `http://www.ietf.org/rfc/rfc5023.txt`.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Reading, 1995.

[Gooa] Google Inc. *Developer Guide – Protocol Buffers.* Cited: Jul 13 2009, URL: `http://code.google.com/intl/de-DE/apis/protocolbuffers/docs/overview.html`.

[Goob] Google Inc. *Google Calendar CalDAV support.* Cited: Jul 12 2009, URL: `http://www.google.com/support/calendar/bin/answer.py?hl=en&answer=99355`.

[Her09] Dirk Hering. *Entwicklung eines Dienstes für Real-time Collaborative Editing für die Mobilis-Plattform.* PhD thesis, September 2009.

[HFV+] Dirk Hering, Christopher Friedrich, Lukas Vierhaus, Martin Werner, and Benjamin Söllner. *Androidbuddy - Komplexpraktikum.*

[HMESA08] Joe Hildebrand, Peter Millard, Ryan Eatmon, and Peter Saint-Andre. Service Discovery. XEP-0030 (Final Standard), June 2008. Version 2.4, URL: `http://xmpp.org/extensions/xep-0030.html`.

[Joh05] Leif Johansson. XMPP as MOM. Online (Presentation), April 2005. Cited: Jul 11 2009, URL: `http://www.gnomis.org/presentasjoner/oso2005/xmpp.pdf`.

[Kor08a] István Koren. *Conceptual Design of a mobile collaborative Platform based on Android and XMPP.* PhD thesis, September 2008.

[Kor08b] István Koren. *Conceptual Design of a mobile collaborative Platform based on Android and XMPP.* PhD thesis, September 2008.

[KSA09] Justin Karneges and Peter Saint-Andre. In-Band Bytestreams. XEP-0047 (Draft Standard), March 2009. Version 1.2, URL: `http://xmpp.org/extensions/xep-0047.html`.

[Lau] Eric Laurier. Why people say where they are during mobile phone calls.

[LBSA+09] Scott Ludwig, Joe Beda, Peter Saint-Andre, Robert McQueen, Sean Egan, and Joe Hildebrand. Jingle. XEP-0166 (Draft Standard), June 2009. Version 1.0, URL: `http://xmpp.org/extensions/xep-0166.html`.

[LSAE+06] Scott Ludwig, Peter Saint-Andre, Sean Egan, Robert McQueen, and Diana Cionoiu. Jingle RTP Sessions. XEP-0167 (Draft Standard), June 2006. Version 1.0, URL: `http://xmpp.org/extensions/xep-0167.html`.

[LT09] Soren Lassen and Sam Thorogood. Google wave federation architecture. Google Inc., Google Inc., 2009. Cited: Jul 12 2009, URL: `http://www.waveprotocol.org/whitepapers/google-wave-architecture`.

[Mac07] Henrik Machela. *Entwicklung einer Infrastruktur für synchrone Groupware auf Basis von Jabber Instant Messaging und Presence.* PhD thesis, 5 2007.

[MBLH06] Marcin Matuszewski, Nicklas Beijar, Juuso Lehtinen, and Tuomo Hyyryläinen. Mobile peer-to-peer content sharing application. In *3rd IEEE Consumer Communications and Networking Conference, 2006. CCNC 2006.*, number 2, pages 1324–1325. Nokia Research Center, Tampere and Helsinki University of Technology, IEEE (Institute of Electrical and Electronics Engineers), 2006.

[MM05] Matthew Miller and Thomas Muldowney. Publishing Stream Initiation Requests. XEP-0137 (Draft Standard), September 2005. Version 1.0, URL: `http://xmpp.org/extensions/xep-0137.html`.

[MME04a] Thomas Muldowney, Matthew Miller, and Ryan Eatmon. SI File Transfer. XEP-0096 (Draft Standard), April 2004. Version 1.1, URL: `http://xmpp.org/extensions/xep-0096.html`.

[MME04b] Thomas Muldowney, Matthew Miller, and Ryan Eatmon. Stream Initiation. XEP-0095 (Draft Standard), April 2004. Version 1.1, URL: `http://xmpp.org/extensions/xep-0095.html`.

[MSAM08] Peter Millard, Peter Saint-Andre, and Ralph Meijer. Publish-Subscribe. XEP-0060 (Draft Standard), September 2008. Version 1.12, URL: `http://xmpp.org/extensions/xep-0060.html`.

[PSA08] Sean Egan Peter Saint-Andre. Jingle DTMF. XEP-0181 (Experimental Standard), September 2008. Version 0.11, URL: `http://xmpp.org/extensions/xep-0181.html`.

[Ros07] J. Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. RFC Draft, October 2007. Version 19, Expired on May 1, 2008, URL: `http://tools.ietf.org/html/draft-ietf-mmusic-ice-19`.

[RSC+02] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393, URL: `http://www.ietf.org/rfc/rfc3261.txt`.

[SA04a] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 3920 (Proposed Standard), October 2004. URL: `http://www.ietf.org/rfc/rfc3920.txt`.

[SA04b] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. RFC 3921 (Proposed Standard), October 2004. URL: `http://www.ietf.org/rfc/rfc3921.txt`.

[SA06] Peter Saint-Andre. Out of Band Data. XEP-0066 (Draft Standard), August 2006. Version 1.5, URL: `http://xmpp.org/extensions/xep-0066.html`.

[SA09a] Peter Saint-Andre. Codecs for Jingle RTP Sessions. XEP-0266 (Experimental Standard), April 2009. Version 0.2, URL: `http://xmpp.org/extensions/xep-0266.html`.

[SA09b] Peter Saint-Andre. Jingle File Transfer. XEP-0234 (Experimental Standard), February 2009. Version 0.9, URL: http://xmpp.org/extensions/xep-0234.html.

[SA09c] Peter Saint-Andre. Jingle In-Band Bytestreams Transport. XEP-0261 (Experimental Standard), March 2009. Version 0.2, URL: http://xmpp.org/extensions/xep-0261.html.

[SAM09] Peter Saint-Andre and Dirk Meyer. Jingle SOCKS5 Bytestreams Transport Method. XEP-0260 (Experimental Standard), March 2009. Version 0.2, URL: http://xmpp.org/extensions/xep-0260.html.

[SMSA07] Dave Smith, Matthew Miller, and Peter Saint-Andre. SOCKS5 Bytestreams. XEP-0065 (Draft Standard), May 2007. Version 1.7, URL: http://xmpp.org/extensions/xep-0065.html.

[Ste02] Greg Stein. Use Of WebDAV in Subversion. Online, January 2002. Cited: Jul 12 2009, URL: http://subversion.tigris.org/webdav-usage.html.

[SVPN04] Risto Sarvas, Mikko Viikari, Juha Pesonen, and Hanno Nevanlinna. Mobshare: controlled and immediate sharing of mobile images. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, pages 724–731, New York, NY, USA, 2004. ACM. URL: http://doi.acm.org/10.1145/1027527.1027690, ISBN: 1-58113-893-8.

[TSLA06] Jarkko Tolvanen, Tapio Suihko, Jaakko Lipasti, and N. Asokan. Remote storage for mobile devices. In *First International Conference on Communication System Software and Middleware, 2006. Comsware 2006.*, pages 1–9. Nokia Research Center, Helsinki, IEEE (Institute of Electrical and Electronics Engineers), August 2006.

[weba] Application fundamentals / android developers. Online. Cited: Oct 27 2009, URL: http://developer.android.com/guide/topics/fundamentals.html.

[webb] Cyrket - estrongs file explorer. Online. Cited: Jul 13 2009, URL: http://www.cyrket.com/package/com.estrongs.android.pop.

[webc] Designing a remote interface using aidl / android developers. Online. Cited: Oct 27 2009, URL: http://developer.android.com/guide/developing/tools/aidl.html.

[webd] Intents and intent filters / android developers. Online. Cited: Oct 27 2009, URL: http://developer.android.com/guide/topics/intents/intents-filters.html.

[webe] Rss 2.0 specification. Online. Cited: Jul 12 2009, URL: http://www.rssboard.org/rss-specification.

[web08] Caldav resources – about the protocol and more... Online, 2008. Cited: Jul 12 2009, URL: http://caldav.calconnect.org/.

[web09] Google wave federation protocol. Online, 2009. Cited: Jul 12 2009, URL: `http://www.waveprotocol.org/`.

List of Figures

List of Tables