Storing Personal Information Management data

Akonadi - unifying PIM data for KDE

Robert Zwerus (arzie@dds.nl)

Graduation committee:

dr. ir. Djoerd Hiemstra	University of Twente
Pavel V. Serdyukov, M.Sc.	University of Twente
prof. dr. Peter M.G. Apers	University of Twente
Till Adam	KDE
Volker Krause	KDE

Abstract

Storing Personal Information Management (PIM) data is not trivial, because of the variety in content types. Existing PIM storage systems have shortcomings in performance, data concistency and/or concurrency. In this thesis, we propose several optimisations and test them in Akonadi, KDE's new central PIM data access manager. The optimisations include using the D-Bus protocol for transmitting short commands and notifications and an IMAP-compatible protocol for data access and modification. The PIM data is kept in its native format, but compressed and split up into separate, frequently-used parts for increased performance. Both the synthetic and use case based evaluation results show that the proposed modifications perform well and help maintain data consistency in Akonadi.

Preface

Welcome to the thesis belonging to the final project that concludes my studies of Computer Science at the University of Twente!

The final project is overseen by the Database group, with Djoerd Hiemstra and Pavel Serdyukov as mentors. As a big part of the project involves KDE, they provide mentors as well, Volker Krause and Till Adam.

When I was looking for an assignment for the project, I had open source software in mind. I'd been running Linux with the KDE desktop for years and wanted to contribute something useful. After asking around and investigating some options, I came in contact with Volker, who helped me get up the speed on the Akonadi project. Attending a KDE PIM meeting in Osnabrck got me to meet the people working on KDE's PIM software.

Defining the precise working area within Akonadi took some time, but everything became more clear on the way. In the research phase, Akonadi's state was analysed, along with material on PIM storage, communication methods and evaluation data. A few months later an Akonadi meeting took place in Berlin, which was also good to get everybody on the same page again. Implementation started after that and took a while. Evaluation methods were defined and the implemented ideas were tested on performance and robustness. The thesis gained its present form along the way and was updated and restructured when necessary.

This project serves more than one goal: to find out smart methods for handling personal information and to help the KDE project.

Acknowledgements Firstly, I would like to thank my supervisors for their help with the project. Djoerd and Pavel, thanks for keeping things in perspective and helping me specify and refine the scope of my project. Volker and Till, thanks for helping with the KDE/Akonadi related parts of the project. Also, thanks to the rest of the KDE PIM developers, whom I've met in several developer meetings.

As for the non-technical side of things, I would like to thank my girlfriend Klarieke for motivating and supporting me.

Enschede,
November 2007

Robert Zwerus

Contents

Chapter 1

Introduction

1.1 Background

Computers are used for Personal Information Management (abbreviated 'PIM' in the rest of this document) more and more. The spectrum of PIM data is growing, not only emails, events, contacts and notes, but also blog entries and instant messaging history come into play nowadays. More types are expected in the future.

Storing all these different types of data efficiently is not trivial. Although largely consisting of textual information of varying size, these items contain binary parts as well (e.g., email attachments, contact photos).

In this thesis, the above-mentioned storage problem is investigated. Research is performed on how to store PIM data, how to provide concurrent access, how to allow fast searching, how to communicate between the storage and client applications and how to evaluate a PIM storage system. To test the findings of the research, the PIM storage of the KDE project is taken as a "guinea pig". Optimisations for it are designed, implemented and evaluated.

KDE (KDE07) is a powerful Free Software (FSF07) graphical desktop environment. It is available for Linux and Unix workstations, and is to become available for Microsoft Windows as an alternative to the default Explorer environment.

KDE contains a set of applications for Personal Information Management, namely KMail, KOrganizer, KAddressbook and many more. They can be used separately or from within Kontact, an application that groups the suite of PIM applications together.

In KDE 3, each application has its own method of storing and retrieving data, which has several negative aspects. Some applications store their data in a single file, which results in low performance, especially with concurrent access. Sometimes the same data is held in memory multiple times, taking up too much space and lowering the capacity of the whole system. Additionally, access to groupware servers or otherwise remote data is not always possible.

More technically, KDE 3 uses the KResources framework, where each kind of data storage is an implementation of a KResource class. It has an online

model, expecting that the storage is always available, even if it is a remote one. Adding an offline mode (for example for laptop users, wanting to use IMAP to connect to their mailbox) is not a case of extending the existing class, but rewriting it with an offline model in mind. This introduces the risk of making and solving the same programming mistakes again. The implemented KResources are not nicely separated from their related GUI items, which makes using them from console software or from within libraries harder.

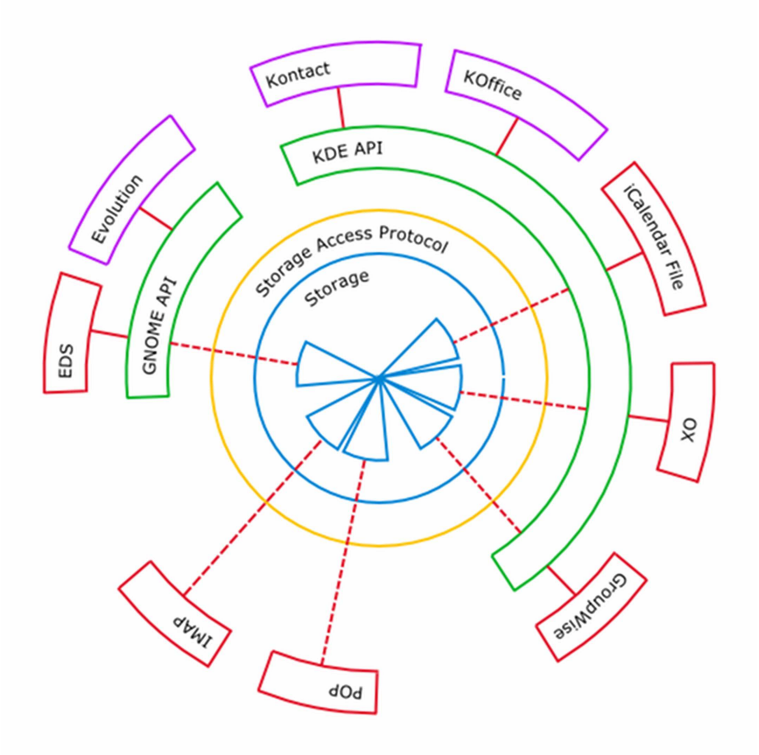

Figure 1.1: *Akonadi's global architecture*

Akonadi [1] is the proposed solution to these shortcomings, a unified data access manager for PIM data in the user's desktop. All PIM data is accessible from a central place, thus enabling the possibility of synchronous access, locking and reduced memory consumption. Additionally, optimizing Akonadi means improving a host of applications at once instead of one at a time.

Akonadi is a standalone server, separated from the client applications. This means that it is available for all kinds of software, including non-KDE applications, that are welcome to use it.

The idea of a centralised access point for KDE PIM emerged in 2006. Since then, implementation is taking place, but a lot of thinking, designing and further implementing needs to be done. As for the PIM applications, work has started porting them to KDE 4. However, the developers are waiting for Akonadi to become usable, before converting the storage access methods to it.

[1] `http://pim.kde.org/akonadi`

1.2 Research questions

- How to transfer PIM data between the storage component and the applications/libraries using it with high performance?
- How to store PIM data, consisting of tabular, textual and binary parts, efficiently and consistently?
- What are the performance and robustness of Akonadi?

1.3 Goals

- Decide upon a communication protocol and data format for transferring PIM data from and to the storage and applications/libraries.
- Investigate and specify a way of storing PIM data efficiently.
- Research and implement the possibilities to keep the stored PIM data consistent.
- Create and execute evaluation methods for testing the performance and robustness of PIM storage systems.
- Apply the found answers to Akonadi.

1.4 Approach

Initially, requirements are stated and research is done in the fields relevant to Akonadi (e.g., PIM data storage, concurrency control, data protocols and formats). Based on the results of this research, solutions are designed and implemented.

Akonadi's performance and robustness is evaluated, according to an evaluation plan and with a collection of test data.

1.5 Report structure

This document contains a standard software engineering structure. The next chapter lists and elaborates on the requirements for Akonadi's storage and its communication with other parts of the system. Following up on those, the Research chapter investigates the available solutions to the stated requirements. The Design chapter describes the anatomy of Akonadi and explains how the chosen solutions are implemented.

To investigate whether the implementation meets the requirements, the Evaluation chapter discusses the approach and the results of the performed tests.

In the end, conclusions are drawn with regard to the implemented solutions. Also, recommendations are made to inspire further developments in Akonadi.

Chapter 2

Requirements of Akonadi

This chapter poses the general requirements for the KDE project and the specific requirements for the added functionality of Akonadi.

2.1 General requirements

1. KDE uses the C++ programming language and the Qt-library created by Trolltech (Tro07).

2. Akonadi is licensed under the GNU (Library) General Public License.

3. Akonadi should not depend heavily on KDE libraries, because it needs to become a desktop-wide standard.

The work in this thesis builds on the existing basis of Akonadi. Technical documentation about the design and current state of the software can be found in KDE's API documentation (KK07).

2.2 Specific requirements

2.2.1 Communication

KDE components need the ability to communicate with each other. It should be able to send and receive commands and notification messages to specific components or to a set of listeners. Transmitting these types of messages should also be possible between Akonadi's components.

For Akonadi's main purpose (handling PIM data), a library, called libakonadi, provides an API to client applications. For communication between the Akonadi storage and libakonadi, a message-access protocol is necessary. It should allow retrieving a list of PIM items, fetching several PIM items at once, as well as adding/updating/deleting PIM items in the storage. Performance is of importance, as well as reliability. Also, a platform independent protocol would be nice to ease other implementations of libakonadi.

2.2.2 Data storage

Akonadi's purpose is to be a central access point to all the PIM data of a KDE desktop user. This does nowadays consist of email, notes, addressbook and calendar items, but in the future many new types of data will probably emerge. Because of this, it is important to store the items in a uniform, non-specific manner. The storage should not know anything about the semantics of the stored items, the client applications are responsible for this. In this way, Akonadi is future-proof, effortlessly storing any new type of information the client application developers can think of.

Akonadi is using an offline mode by default, treating online mode as a more or less special case. Both modes are implemented in a single component. A partially online mode should also be supported, where parts of the PIM items are cached by Akonadi and others only reside in the external resource (such as a groupware server). An example of this: the headers of all email messages are cached, along with the complete messages in a few important folders, but the complete messages of the other folders are fetched from the groupware server on demand.

Essentially, Akonadi is meant as a cache, not as the designated storage of the data. For groupware and other remote kinds of stores, this is trivial, because the main storage location of the data is the remote server, not the user's system. Local kinds of stores do not specifically need caching, however, it has several advantages. The cache is designed to be fast, where some PIM data formats (e.g., an iCalendar file, where for every data modification the complete iCalendar file has to be rewritten) are not. A uniform way of accessing the items makes development and maintenance easier.

The PIM items are structured, e.g., emails sometimes have attachments belonging to them. It should be possible to retrieve a description of the structure of a PIM item, as well as parts of the PIM items (for example, only the text of an email message, transferring the attachments only if the user wants to open them).

Client applications can generate additional data for an item, because they have knowledge about the content of items. Email software knows the meaning of email headers and can create an envelope of them. Akonadi should be able to store this extra information. With this ability, listing an email folder's contents only requires retrieving all the envelopes and processing them. Otherwise, all the email's headers would need to be fetched and interpreted, which costs more time for the data transfer and more for the processing.

Searching data

Nowadays desktop search plays an important role in modern desktops. Especially PIM data needs to be searchable, therefore Akonadi must allow desktop search engines to index its stored data.

The method of searching the stored data depends on the chosen backend. If direct filesystem storage is used, the indexers of desktop search applications automatically add newly stored data to their index. All other backends require creating a plugin or feeder to provide the indexers with a readable version of the backend's contents.

2.2.3 Data consistency

Because there is one instance of the Akonadi storage server per user, concurrent access has to be possible. Multiple applications can access the same data, creating the possibility of conflicting write attempts. Naturally, this has to be prevented, always leaving the storage in a consistent state and, where possible, notifying the user of the conflict without loss of data.

If data conflicts occur and they cannot be automatically solved (for example, when two different columns of the same row are modified), present the user with an option to choose between the two.

Chapter 3

Related research in PIM

In order to accomplish the set goals and to fulfill the requirements, research is done on the available options. This chapter's structure is equal to that of the specific requirements, stated in the previous chapter.

3.1 Communication

3.1.1 Protocols and formats

Transmitting commands and notifications between components and client applications can be done using an IPC (Inter-Process Communication) protocol.

Several IPC methods exist, such as DCOP (DCO07), which is used in KDE 3. However, D-Bus (PCL06) (see figure 3.1) is the accepted standard for IPC in Qt4, GNOME and KDE 4. It is a message bus system, which allows processes to communicate with one another in a simple way.

An IMAP-like (Chr03) protocol between libakonadi and the Akonadi server is implemented already. It is widely used for email access, and thus well-tested and optimised for speed, even when handling large numbers of messages. It supports streaming, which allows more efficient processing of commands. Incoming commands can be queued and reordered, results can be sent back asynchronously. Using IMAP also makes sure other software can access the Akonadi cache, which makes importing and exporting a lot more easy.

The actual PIM items can be transferred in several ways. The simplest option is to use the item's native format (e.g., RFC822 for email, VCard for addressbook entries, iCalendar for events). Existing KDE libraries can be used for parsing the native formats.

Another possibility is to use an XML or MIME based format, but that requires converting the item data whenever the native format is needed again (for example, when communicating with a groupware server). It also removes a level of backwards compatibility with other IMAP clients.

The fastest method would be to store the objects holding the items in binary form, as it only requires serialising and deserialising the object, no interpretation is needed. However, this method introduces a lot of problems, such as platform dependence (because the object's binary form differs between platforms) and

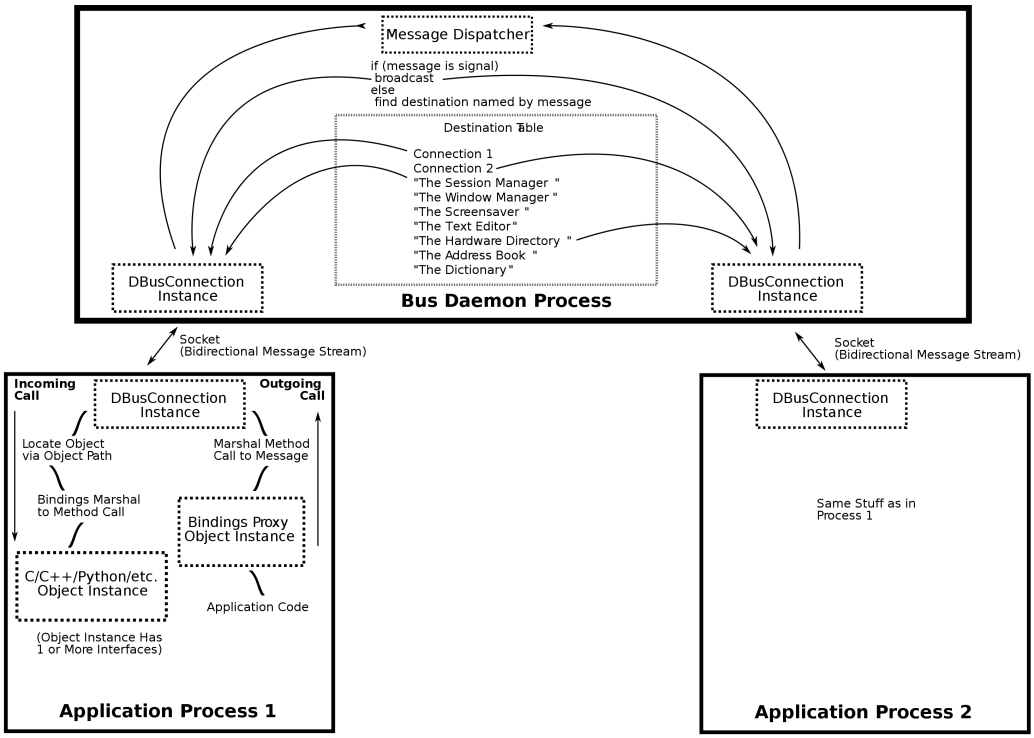

Figure 3.1: *D-Bus diagram, showing broadcast en specific messages (PCL06)*

compatibility between different versions of Akonadi (for most modifications to the object's code, the database contents needs to be rebuilt).

3.1.2 Compression

As a lot of the PIM item's formats are textual, it is interesting to have a look at data compression in the storage and the IMAP connection. The reduced storage size is not that important, as storage is cheap, but the performance might increase, especially with the powerful computer processors in use today.

Lots of different compression algorithms exist, many of which are proprietary, but free alternatives exist. Perhaps the best known free library for data compression is zlib (zli07), it is used in hundreds of applications (such as the Linux kernel, the Apache webserver, libpng, OpenSSH and OpenSSL).

A comparison of the capabilities and efficiency of some free archivers (table 3.1) shows their differences. gzip uses about the same algorithm as zlib, it boasts a good tradeoff between computational intensity and compression ratio. It does not have the best compression ratio, but the speed makes up for it.

One possible drawback of compression might be that the data cannot be searched while in the storage backend. However, Akonadi is not responsible for indexing the data, it is forwarded to designated search software (such as

Name	Text %	Text (s)	Executables %	Executables (s)	Images %	Images (s)
7-zip	19	18.8	27	59.6	50	36.4
bzip2	20	4.7	37	32.8	51	20.0
rar	23	30.0	36	275.4	58	52.7
advzip	24	21.1	37	70.6	57	41.6
gzip	25	4.2	39	23.1	60	5.4
zip	25	4.3	39	23.3	60	5.7
lha	27	3.7	40	13.2	61	9.3

Table 3.1: *Comparison of file archivers (fil07)*

Strigi) instead., so there is no drawback there.

Searching in the storage backend also is not context-aware (e.g., searching for the word 'subject' returns all email messages, because it is a standard header name), whereas search software can process all data formats intelligently.

3.2 Data storage

Because PIM data consists of a lot of different kinds, it gets scattered all over the user's desktop, being handled by separate applications and stored in different locations. Unifying PIM data (KJ06) using existing applications is hard, because of the fact that a lot of the references are ambiguous (e.g., one name can refer to two different people).

Presenting the PIM data in a unified way (Dum07) is the applications' responsibility, not Akonadi's. Supporting it where possible, however, would be nice.

Storing everything in a central location, as Akonadi does, is one step in the right direction. Extracting and storing relations between the items is another step, which can be done by Nepomuk (Con07). Making the contents of the items available for fast searching can be done by desktop search engines, such as Strigi (vdO07).

3.2.1 Storage backend

Because Akonadi uses Qt, using a natively supported database is convenient. However, Qt allows writing a driver for currently unsupported databases, so investigating the options is no waste of time.

If possible, using an embedded database is preferred. An embedded database is linked directly to the Akonadi server, relieving the server from starting the database and it usually has a higher performance, because of the direct connection between the database and the Akonadi server.

Direct filesystem storage

A straightforward method of storing PIM data is to save it directly on the filesystem. The mbox format (mbo07) stores a complete folder as a single file, whereas the Maildir format (Ber95; Pre06) stores every item separately.

Direct storage removes a layer of abstraction, and thus overhead, possibly improving performance a little. Also, in the case of filesystem corruption, there is a possibility only part of the data is lost.

On the other hand, the actual performance depends on the underlying filesystem, which is not freely selectable by developers, but by the users. This is undesirable, because it makes the user responsible for choosing the right filesystem, which is not trivial. Also, it is easy for the user to access the files, thus endangering data consistency. Normally, the PIM storage application should be the only entity handling the data.

Storage granularity is another issue, which the mbox format runs into. When a complete folder is stored in a single file, locking is necessary to avoid data corruption when multiple processes access the folder simultaneously. The performance is rather low, because reading and writing an item perform operations on the entire folder. Storing single items by themselves, as the Maildir format does, helps in this matter. Performance is much better, and the locks only lock the item, not the folder it belongs to.

Relational database

Relational database management systems (RDBMSes) are in use for decades, indicating they are mature and well-supported. SQL, the standard language for accessing the stored data, is widely known by developers.

RDBMSes store data in tables, that in turn consist of columns of a certain data type. This makes them ideal for storing tabular data, i.e., data items that are built up of a set of distinguishable parts. However, when the data is a single binary object, RDBMSes cannot offer more convenience than a filesystem does, they only add overhead. This disadvantage also holds for hierarchically structured data items and data items that are a single block of text, although RDBMSes are increasingly able to handle such kinds of data.

Using object/relational mappers for mapping objects to database records In the source code of Akonadi, objects are used to represent the PIM items. When communicating with a relational database, a mapping has to be performed between the objects and the database records containing the data. To ease this process, object/relational mappers (Amb06) are used. They automate the mapping of objects to records and vice versa, relieving the programmer from manually storing them in the database.

Unfortunately, most mature object/relational mappers (such as Hibernate [1]) are for the Java or .NET programming languages. For C++, only a few exist:

- Database Template Library [2]; not under active development
- Progress DataXtend CE [3]; professional, but not free
- Object Builder [4]; immature, not under active development

[1] `www.hibernate.org`
[2] `http://dtemplatelib.sourceforge.net`
[3] `www.progress.com/dataxtend/dataxtend_ce/index.ssp`
[4] `www.ceamus.com/objbuilder`

Unfortunately, all of the mentioned packages are unfit for usage in Akonadi's source code. Partly because some are non-free, creating a licensing conflict. The others are not actively being worked on and not mature enough.

Handling multipart data PIM items might consist of multiple parts, probably in a tree-like structure (e.g., emails in MIME-format). It is thus possible to split them up and store the parts separately. Dissecting costs a bit of performance when new items are stored, but gives a higher performance each time only a part of the item is retrieved.

When only the parts of an item are stored, retrieval of the complete item means it has to be reconstructed in its original form. An option to counteract this is to store the complete item as well. This has the disadvantage of taking up about twice the storage space, but removes the need of reconstructing the item completely. Both the individual parts and the complete item can be retrieved at a high speed.

Storing large objects Large PIM items or parts of PIM items (e.g. photos or email attachments) occur regularly and are expected to grow in the future. Computer systems are growing in performance as well, but a gain in efficiency could be made when these large items are stored in a special way.

Most databases support storing binary objects in so-called BLOBs (binary large objects), but this can mean making a sacrifice in performance.

A possibility is to store them in separate files, directly on the filesystem. Filesystems are built for this kind of data, and an abstraction layer (i.e., the database) is removed. The disadvantage is that effort has to be made to keep the data consistent, as it would be spread over multiple stores (the database and the filesystem).

Storing custom parts A simple approach is adding an extra column to the table with PIM items, containing all the extra information in one cell (encapsulated in a container format, like XML). This has the advantage of being a generic solution for all types of PIM data. The disadvantage is that it is slow and defeats the purpose of the database, because the container format will have to be dissected and interpreted.

Another option is to create an extra table for every kind of PIM item (e.g., emails, contacts, events), that contains the additional columns for that data type. The advantages and disadvantages are the inverse of the previous solution.

Some kind of solution in the middle is to create an extra table with all the extra information in the form of two columns: a key and a value. The rows of the table can then be linked to the PIM item. In this way the purpose of the database is retained, as is the genericity of the concept.

Free relational databases A number of free databases is available:

- MySQL [5]; mature, high performance, well-known, but with a relatively high footprint for an embedded system
- MySQL/Embedded [6]; same as MySQL, but no concurrency with transac-

[5] `www.mysql.com`

[6] `www.mysql.com/products/embedded`

tional InnoDB engine (see section 4.2.3)

- SQLite [7]; small, but not threadsafe, low concurrency support, no foreign keys
- PostgreSQL [8]; no embedded version
- Firebird [9]; embedded version only available for Windows™, not widely known, but worth investigating

Object database

Object databases (Obj07) are databases that allow storing objects directly, thus removing the need for a mapping between the objects and the database's native storage format. Their usage resembles using object/relational mappers to talk to a relational database, but completely transparent to the developer.

This kind of database is relatively new, not many mature ones exist yet, let alone open source versions.

Native XML database

XML databases (XML07; Sta01; Har05) use XML documents as their data holder, they store and present their contents as XML.

The advantage of this kind of database is that it is designed for hierarchically structured data, and it removes the need of data conversion for XML applications. For Akonadi, this is only a partial advantage, as the applications use the item's native format, which is not always XML.

Disadvantages include the inefficient treatment of binary and tabular data. It also is a relatively new technology, so the existing DBMSs are not as mature as older systems.

Free native XML databases Several XML databases are available:

- MonetDB/XQuery (CWI04)
- Xindice (Apa07); no C++ API
- eXist - Open Source Native XML Database (Mei07)

Storage method of widely used software

Email Email messages form a large part of the PIM data, therefore looking at the storage method of several well-known email applications can be worthwile.

Research on the different kinds of storage (mbox, Maildir and database storage) of email is performed by Elprin and Parno (EP03), concluding that database storage (using a MySQL database) is the best performer (with a notable exception to data removal).

[7] www.sqlite.org
[8] www.postgresql.org
[9] www.firebirdsql.org

A lot of well-known, email software (Postfix SMTP server, Courier IMAP server, KDE's KMail, GNOME's Evolution) uses the Maildir format. The mbox format is used in widely used software (Mozilla Thunderbird) as well. Almost no software uses a database-backed system. This contrasts with the previously mentioned research, the reason for this might be conservative motives.

Organiser For dedicated organiser software, not many storage options are in use. The iCalendar (DS98) and vCalendar formats are mostly used. These formats store events in a file, which can be located on the user's drive, but also on a remote server. Organiser data is typically small in size, a possible reason for the low amount of formats in use.

A few examples of this kind of software are Mozilla Calendar, Apple iCal, Novell Evolution and KDE's own KOrganizer.

Some of these also provide connectivity to groupware solutions, see below for their storage methods.

Addressbook Contacts stored in an addressbook are a lot like events in an organiser, but their format is generally vCard

For large addressbooks, e.g., those used in companies and universities, LDAP servers are used. LDAP (Lightweight Directory Access Protocol (LDA07)) is a directory access protocol. These store the directory entries in the LDIF format, which is comparable to vCard, but has a different syntax.

Groupware Groupware systems combine services, as described in the previous paragraphs. For storage, they sometimes rely on existing (email) servers, adding (invisible) folders for the other types of PIM data (such as calendar and addressbook items). Examples of these systems include Kolab [10] and OpenXchange [11].

Other collaborative software uses a database backend, for example OpenGroupware.org [12] and eGroupWare [13].

The reason for the wider adoption of database backends amongst groupware systems might be that they have not been around for as long as the separate applications. Another reason could be that groupware software often runs on servers, where databases are a lot more common than on desktops.

3.3 Data consistency

One of the important requirements of PIM data is its consistency, this means that the data's integrity remains ensured when modifications take place. When data depends on the existence of other data (e.g., an email message belongs to a certain folder), this poses a constraint on the possible modifications to it (when the folder is deleted, the message is deleted as well).

[10] `www.kolab.org`
[11] `www.open-xchange.com`
[12] `www.opengroupware.org`
[13] `www.egroupware.org`

3.3.1 In databases

In databases, ACID properties (ACI07) guarantee the reliability of processed transactions. It consists of these properties:

- Atomicity: a transaction is either performed completely or not at all
- Consistency: a transaction may not break the integrity constraints of the database
- Isolation: a transaction's result is only visible after its completion, not before
- Durability: a transaction's result will persist after it is successfully committed

To solve the above-mentioned email/folder problem, cascaded updates and deletes can be used. These help to maintain referential integrity when updates or deletes are executed on data in one table, by automatically performing actions on the referenced records in the other table.

3.3.2 Concurrency control

In a multithreading environment, it is possible to retrieve the same data object twice. When both threads perform modifications to the object and try to write it back to the storage, concurrency control (BG81) becomes a necessity.

To allow for concurrent access, an effort has to be made to always keep the storage in a consistent state. Data updates should therefore be performed one at a time, informing the applications and/or users of the (un-)committed action.

In Akonadi, this cannot be solved by simply using database transactions, because of the loose coupling between the client and the server via a stateless protocol. Thus, a concurrency control method has to be implemented in Akonadi.

Several types of concurrency control use a mechanism called locking, for read or write actions on data objects. A read or write lock on an object is the right to exclusively read or write that object, respectively. If a lock on an object is issued, other transactions have no or limited access to the object.

Other types are non-locking, they have validity checks instead. These check whether a transaction can commit, based on the state of the affected data objects.

Two-phase locking

This type of locking is the accepted standard kind of locking. Transactions work in two phases, the first one is acquiring the locks, the second one the release of the locks. After a lock is released, no new locks can be acquired in the transaction.

When a read lock is issued to a transaction, other transactions can get read locks as well, but no write lock. A read lock can be upgraded to a write lock, when no other read locks have been issued.

In figure 3.2, two situations are displayed. The first one shows two transactions (T_1 and T_2). T_1 acquires a read lock (RL) on object A, then reads (R) A

Figure 3.2: *Two-phase locking*

and upgrades the read lock to a write lock (WL). After that it writes (W) A and releases the lock (-WL). T_2 then starts, acquires a read lock on A, reads it and releases the lock (-RL).

A more complex situation is shown as well. T_1 and T_2 both acquire read locks on A, but T_2 wants to upgrade its lock to a write lock. This cannot be done before T_1 releases its read lock, which is done after some time. Then T_2 can upgrade its read lock to a write lock on A and write A.

Two-phase locking has a risk of deadlock, where one transactions waits for the release of a lock, held by another transaction, which in turn is waiting for a release of a lock, held by the first transaction.

Pessimistic

Pessimistic locking is a trivial locking mechanism, as it plainly locks every transaction's data until it is completed. This simplicity makes it easy to implement and ensures data consistency.

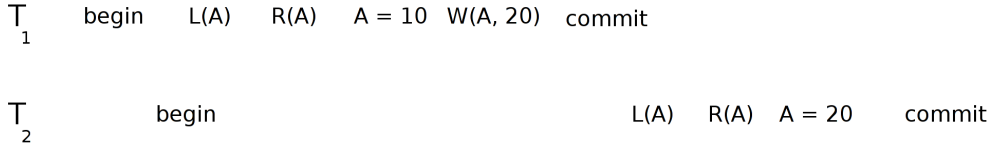

Figure 3.3: *Pessimistic locking*

Figure 3.3 shows two transactions that want a lock on the same data object (A). T_2 has to wait for T_1's lock to be released before it can perform any action on A.

The drawback is that it really slows down a multithreaded system like Akonadi. If more operations are executed simultaneously or operations take

much time, the chance that they operate on the same data increases. This makes it unscalable for increasingly large datasets.

Another disadvantage is that pessimistic locking takes up resources in the storage, because a list of open data objects is necessary. It can also cause excessive locking (where read-only data is locked, while it will not be modified), as well as deadlocks (where multiple transactions wait for each other to finish).

Optimistic

Optimistic locking (Opt07; MWZ04; HD91) is a form of concurrency control, which does not really lock data. It assigns an increasing version number to every object in the database and passes this number along with each transaction. If a transactions tries to commit and the version number of its objects matches that of the objects in the database, the commit is successful. If the version numbers differ, it means that another transaction has processed the objects and the transaction is rolled back.

Instead of rolling back the full transaction, conflict resolving is possible. If two transactions update different parts of an object, both changes can take place, as they do not conflict. When the same part of an object is updated, the user can be notified and asked for a manual correction of the conflict. A last resort is to roll back the conflicting transaction and present the user with an error message.

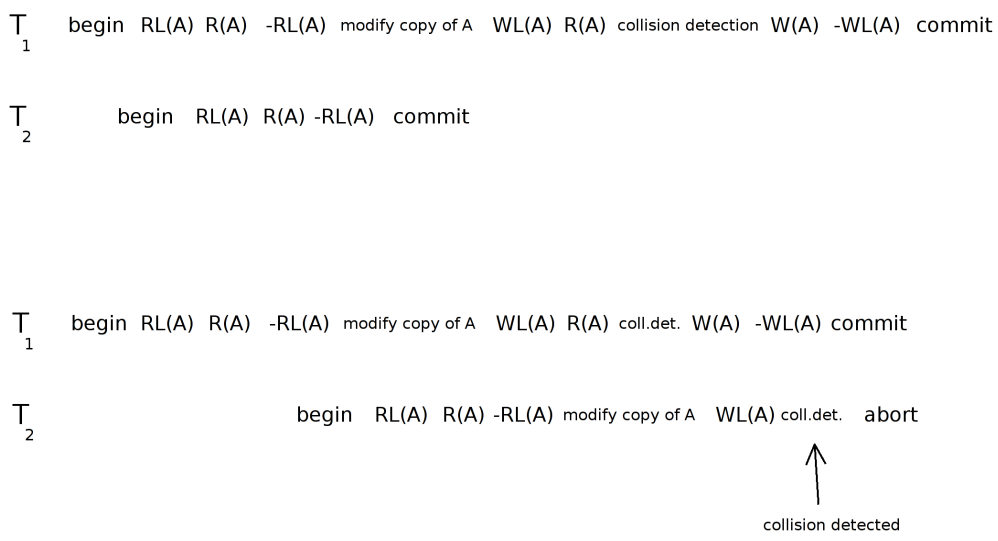

Figure 3.4: *Optimistic concurrency control*

Two example situations are shown in figure 3.4. In the first, both transactions read A, but only transaction T_1 modifies it. After modification, it gets a write lock on A, reads it (to be able to check the version number) and collision detection is performed. The version number is the same as the one at the start of the transaction, so no conflict has occurred and the write is succesful.

In the second situation, both transactions modify their copy of A and want to write it back to the storage. T_1 is successful, because A's version in the storage is the same as it was when reading it. T_2 however detects a collision, as T_1 just updated A's version number. In this case, the transaction is simply aborted.

Implementation of optimistic locking is easy and its performance is high, if a low number of conflicts is expected. The storage only needs a bit of extra information (i.e., the version number of the object), thus it has a low overhead in storage and memory consumption. A slight slowdown in data updates is induced, because the version number has to be updated in addition to the modified data.

Also, applications are not notified of conflicts until they try to save their data. This can be solved by using Akonadi's change notification system, which emits notification signals to all applications when their monitored data is updated. If such a notification applies to the data object at hand, a conflict has occurred and conflict resolving can start.

Since the Akonadi service is run per user, conflicts are not expected to occur regularly. The chance of multiple applications accessing the same data is rather low.

Timestamp-based

Timestamp-based concurrency control does not use locking for providing concurrent access. It gives transactions a timestamp to know in which order they should be executed (Tim07). However, if a newer transaction (that affects the same object as the older one) is presented first, it is aborted.

Also, every object in the database is given a read timestamp and a write timestamp. If a transaction is started and tries to read or write an object, that changed in between, it gets aborted as well.

If a transaction wants to read an object, but the object's write timestamp is later than the transaction's timestamp, the object has changed and the transaction is aborted. If the object's write timestamp is earlier than the transaction's timestamp, it is safe to read the object. The object's read timestamp is then set to the transaction's timestamp.

If a transaction wants to write to an object, and the transaction's timestamp is earlier than the object's read timestamp, another transaction has read the object. Writing to the object would invalidate the other transaction's copy, so the transaction is aborted. If the write transaction has a timestamp earlier than the write timestamp of the object, it means that the object has changed since the transaction started, and the write is skipped. In any other case, the write succeeds and the object's write timestamp is set to that of the writing transaction.

Again, two situations are presented, in figure 3.5. Updates to the read and write timestamps of objects are marked. In the first case, T_1 reads A a bit after T_2 began. T_2 then tries to write A, but gets aborted, because A's read timestamp is newer than T_2's timestamp.

The second situation ends in two successful transactions, as T_2 starts after A's timestamps have been updated by T_1.

The timestamp resolution should be high enough, to avoid duplicate timestamps. The server's clock should be reliable, its values should be strictly

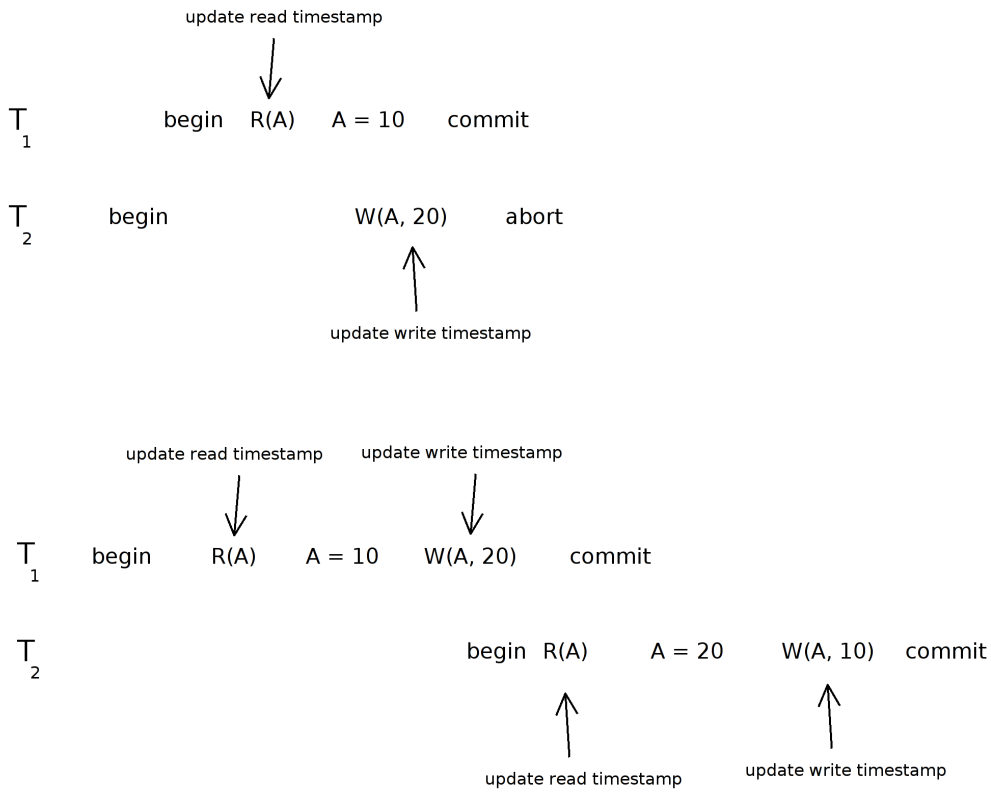

Figure 3.5: *Timestamp-based concurrency control*

increasing.

Although this technique is non-locking, a very short lock is necessary on the object, to write its read or write timestamp.

Multiversion

This type of concurrency control also uses timestamps (or version numbers) to achieve serializability. Transactions never have to wait for a database object, because several versions of an object are maintained (Mul07).

Data objects have read and write timestamps, transactions receive a timestamp when they start. A transactions can read the version of an object that has a write timestamp which is not newer than the transaction itself. If a transaction tries to write an object, it only succeeds when other transactions that write the object started later. If a transaction wants to write an object and the object's read timestamp is newer than the transaction's timestamp, the transaction is aborted and restarted. Otherwise, the transaction creates a new version of the object with the same timestamps as the transaction itself.

Another demonstration, shown in figure 3.6. As can be seen, writes create new versions of objects. Also, transactions read the version of objects, that are

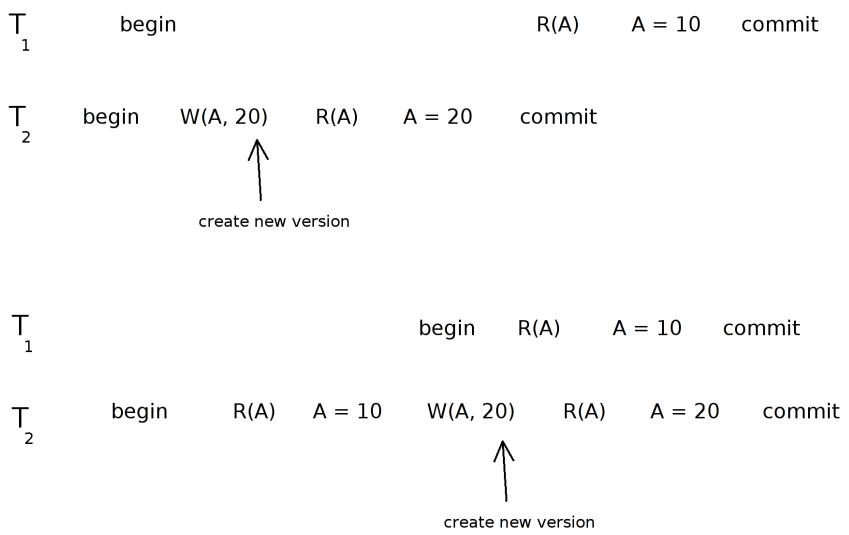

Figure 3.6: *Multiversion concurrency control*

not more recent than the transactions themselves. In the first situation, T_1 reads the version of A, before it was modified by T_2, which had a value of 10 (and not 20).

The second situation shows this more verbosely, T_2 sees A in its modified version, where T_1 sees the older version during the complete transaction.

When using multiversion concurrency control, reads will never block writes and vice versa, this makes it a fast solution. It also provides snapshot isolation at low performance cost, as every transaction views a snapshot of the storage.

The drawback is the extra storage needed for the multiple versions of an object and the timestamp calculations to be performed during the validity checks.

Chapter 4

Design

At first, this chapter portrays and explains Akonadi's global design, to help pinpoint the locations of implemented modifications. Consecutively, more detailed descriptions of the modifications follow.

4.1 Akonadi in general

4.1.1 Global design

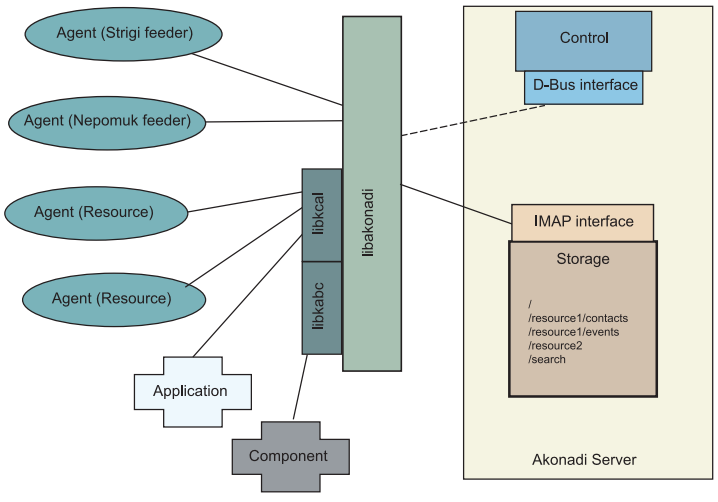

Figure 4.1: *Global design (KZ07)*

Figure 4.1 shows Akonadi's global architecture. The central part shows the server, which consists of several subprocesses. The Control process performs lifetime management of the other processes, it starts, restarts and stops them when necessary.

The storage is responsible for storing and retrieving the PIM items and other information. The D-Bus and IMAP interfaces are the access points for the clients.

A set of Agents perform work on the data in the storage, via libakonadi. Resources are a subset of them, they represent data sources (e.g., vCard files, POP3 servers, groupware servers). Resources keep the data source and the Akonadi storage synchronised, by updating both sides regularly. Autonomous agents operate on the data in the Storage by themselves, examples of these are feeders for Nepomuk (Tru07) and Strigi (vdO07). Other, not yet implemented, feeders that come to mind are those for Beagle (Bea07) or Google Desktop Search.

The left side of the figure displays how applications or application components access the Akonadi server. They use the type specific libraries, like kabc and kcal, that know how to handle addressbook and calendar items respectively. These libraries in turn use libakonadi for contacting the server. libakonadi uses the IMAP and D-Bus interfaces of the server to execute the needed commands.

In figure 4.2, a more detailed view of Akonadi's components is presented. The arrangement of the parts is the same as in the global architecture. The central part of "server" is AkonadiServer. It receives incoming requests from libakonadi via the IMAP interface and handles them appropriately by instantiating the Handler corresponding to the incoming command. This handler further processes the request. It also receives D-Bus notifications via the DBus interface and responds to the request, e.g., by adding or removing a resource.

The DataStore in "storage" communicates directly with the database backend, MySQL in this case. If no database exists at startup, it is created by the DbInitializer according to the database layout (see figure 4.3). CacheCleaner does cache cleanup (adhering to the cache policy defined for every resource, which is not completely implemented yet) on regular intervals. Notifications are sent around via the D-Bus interface, e.g., to indicate modifications to PIM items:

```
NotificationManager::notify ( Item (6, 2Q8cO5u3ZC) in collection 2 modified )
```

The "server/control" component contains multiple managers, that are responsible for instantiating and destroying Agents and Profiles. Profiles are groups of Resources, to enable applications to use different sets of Resources, tailored to their needs.

"libakonadi" is responsible for all communication between the higher level client libraries and the Akonadi server. It provides a set of Jobs, which the client applications or libraries can use. These jobs use the IMAP interface of the server to facilitate the modification and viewing of collections and items. The client libraries can use the Monitor to keep tabs on specific items or collections, it notifies them when their status changes.

4.1.2 Database layout

The database layout as used by Akonadi is shown in figure 4.3. PimItems and Locations play the central roles in the diagram. Each PimItem belongs to a Location (also known as a Collection), which is part of a tree structure. Every

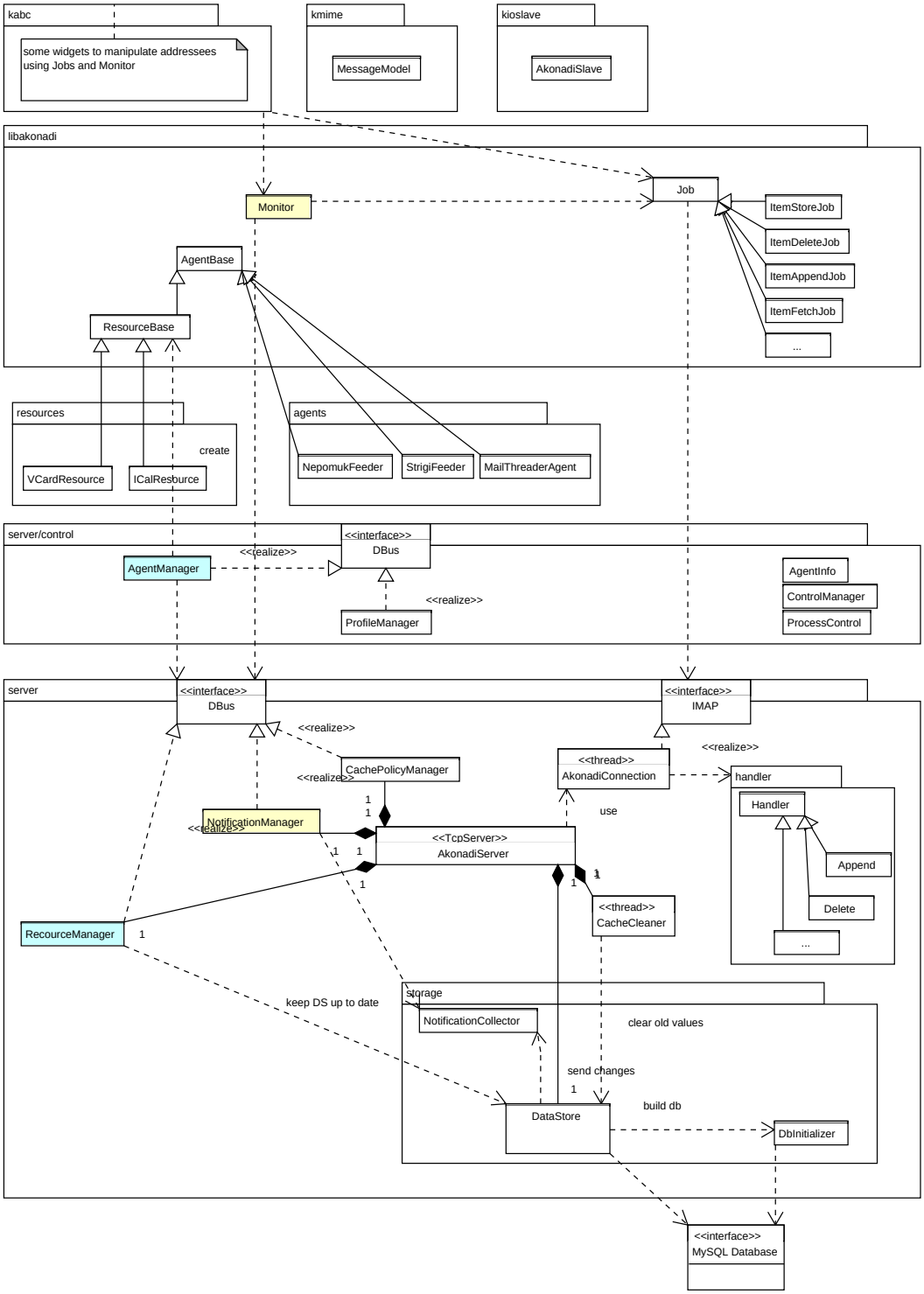

Figure 4.2: *Detailed design (Sch07)*

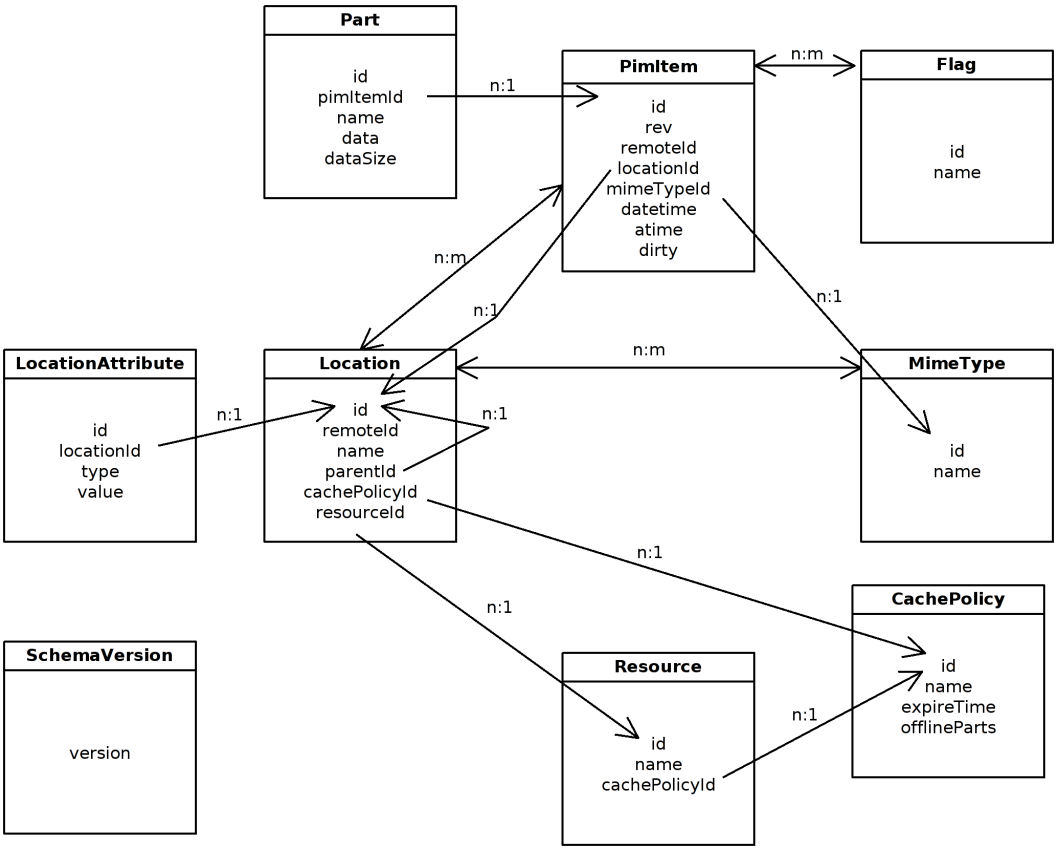

Figure 4.3: *Database layout*

Location has a Resource connected to it, which provides the items from an external location.

PimItems have a MimeType, specifying their contents, and a set of Flags (which indicate new, answered, draft, unseen or deleted items, for example) associated with them. Also, one or more Parts are attached to each PimItem. These contain the actual data of the item, possibly split into multiple parts (see below).

4.2 Implemented modifications

4.2.1 Communication

Protocols and formats

The data format, in which the PIM items are transferred, is the native format of the PIM item (for examples of these, see figure 4.4), because it is tailored to the needs of the specific data type. Interpretation is done in libakonadi, not in the resources.

This decision impacts the IMAP interface in the figures in the first part of this chapter. The items are transferred without pre- or postprocessing.

```
Date: Wed, 6 Jun 2001 02:38:00 -0700 (PDT)
From: brant.reves@enron.com
To: frank.sayre@enron.com, susan.bailey@enron.com
Subject: Glencore Commodities Ltd.

Hello,

Here is a spreadsheet from the confirmations group of all
financial trades between ENA and this entity that have ever
been entered into.

Let me know what you think.
brant
```

Figure 4.4: *Example email message in native format*

Compression

Adding data compression to the IMAP interface forms no obstacle, as the Qt library already offers zlib data compression. Compression and decompression are done in the client library (libakonadi), the former before adding and updating PIM item parts to the storage, the latter after the item parts are fetched from the storage.

This means that the uncompressed item parts only exist in libakonadi and the higher level libraries and applications, while the IMAP interface and the server all handle their compressed equivalents.

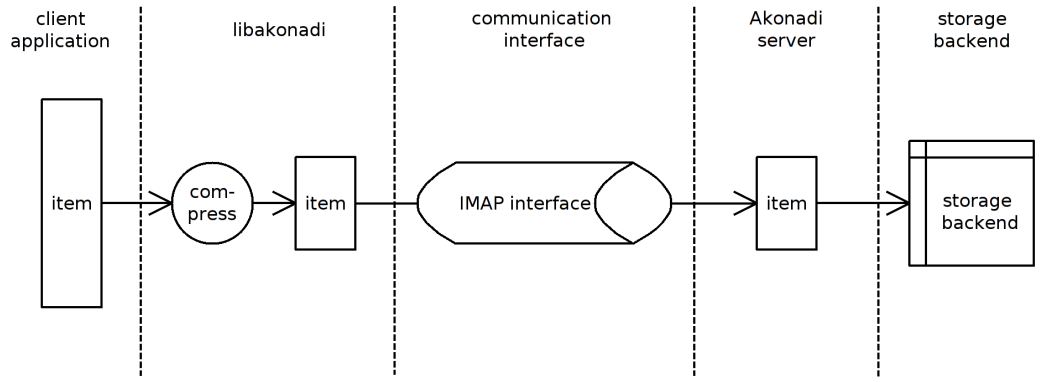

Figure 4.5: *Data compression*

A schematic view of the influence of data compression is shown in figure 4.5. The sending and retrieving of items requires a bit of processing in libakonadi, to compress and decompress the data. The IMAP interface and the storage have less data to handle, so they should be faster than with uncompressed data.

Evaluation is done on the efficiency of compressing the data in the protocol and the storage. If it lowers performance significantly instead of increasing it, while not saving much storage space, compression should not be used.

4.2.2 Data storage

Storage backend

For the data storage, a relational database is initially chosen, because it's well supported and easily usable.

Development of Akonadi started with the SQLite [1] storage backend. After some time however, problems emerged with threading. SQLite is not designed for concurrency, as it locks the full database for every read and write action. It also does not support foreign keys, needed for data which depends upon multiple tables. For these reasons, another backend is used now. SQLite support can be added again, if it gains the needed features.

MySQL is the chosen DBMS, because it is well known by most developers and can be freely distributed with the rest of KDE. It is mature and under active development.

Of course, the indexing features of MySQL are used where appropriate. All "id" columns get an index (mostly automatically, because they are the primary key of a table), as well as the columns that are frequently used for searching (such as the names of PimItems and Parts).

Multipart items

The difference between singlepart and multipart items is visually shown in figures 4.6 and 4.7. With singlepart processing, appending an item is straightforward and fast. Multipart processing requires libakonadi to split the item in several parts. They are sent over the IMAP interface and stored in the storage backend separately. Retrieving an item part turns things around. The singlepart system has to retrieve the full item and extract the item part afterwards, where the multipart system only retrieves the requested part and forwards it to the application. As appending an item is done only once and retrieving its parts many times, it is to be expected that a multipart system will give a higher performance.

All item handling classes and methods are modified to work with separate parts, instead of a single data entry.

The part data is not stored in the PimItems table anymore, but in a Parts table instead. Each entry of the PimItems table has zero or more entries in the Parts table.

As it would be nice to keep the storage access protocol compatible with IMAP, the normal APPEND command (used for inserting new items) is not modified for multipart support. A special command for appending multipart items is introduced, X-AKAPPEND ('X' to indicate an extension, 'AK' to refer to Akonadi).

The FETCH command (for retrieving items or information about items, such as their mimetypes or modification times) already supports a list of attributes to retrieve. The command is modified, so that it supports a list of item part names to be retrieved.

The STORE command (for updating existing items) is like the FETCH command, it also supports a list of attributes to update. An extra attribute is added,

[1] `www.sqlite.org`

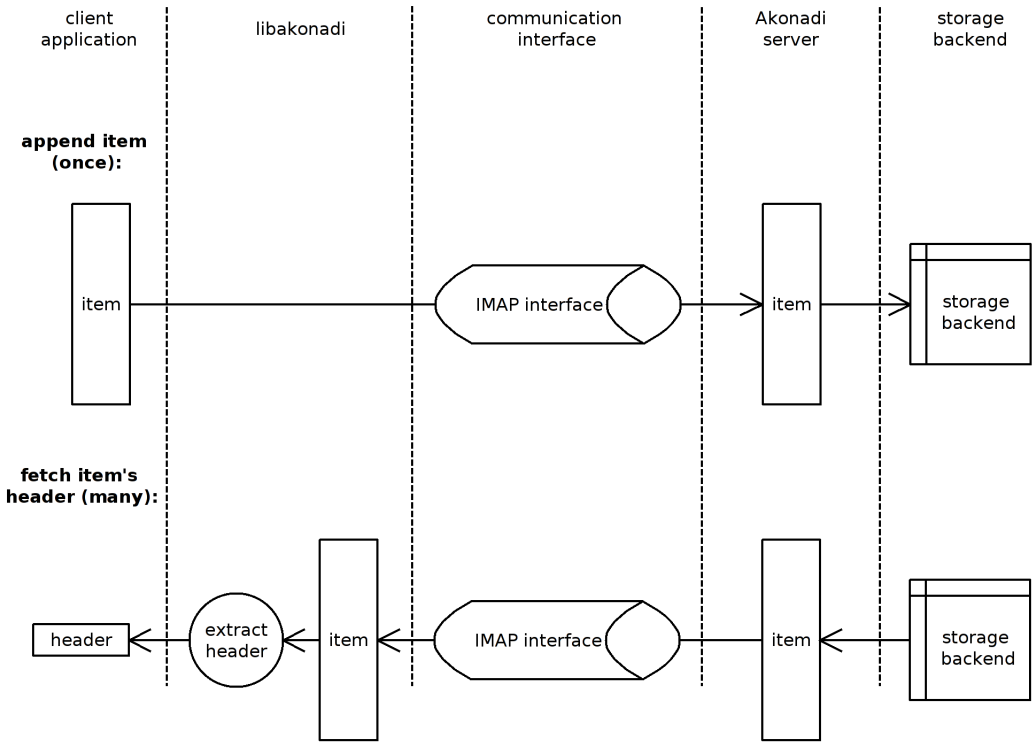

Figure 4.6: *Singlepart items*

to support the removal of item parts. Support for updating item parts is also added.

Most of the changes are made in the jobs in libakonadi and their respective handlers in the server. The database layout changes as well, because of the added Parts table.

Some example commands and the server's responses are shown and explained below. The part data is compressed and is shown as '≪compressed data≫'. If it were not compressed, the PIM item would be sent in its native format. The numbers in front of most commands and responses are so-called tags, which identify a specific command sequence. This is necessary because IMAP does not per se process commands in received order, so the responses can intertwine each other.

The first command is a simple APPEND command. It is the IMAP compatible append, able to store singlepart items. This will create a single part with the default part name (named 'RFC822' internally).

```
4 APPEND 2 (\MimeType[text/directory] \RemoteId[02jF0C3J60]) {382}
+ Ready for literal data (expecting 382 bytes)
<<compressed data>>
4 [UIDNEXT 3]
4 OK Append completed
```

When a multipart item is added, the new command X-AKAPPEND is used.

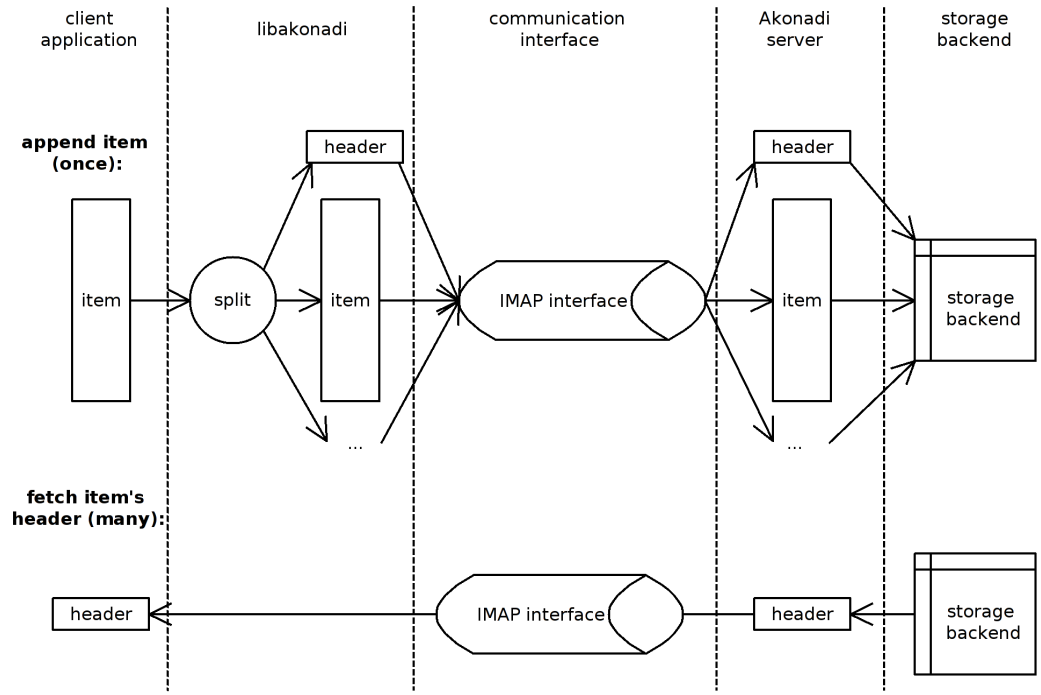

Figure 4.7: *Multipart items*

In this case, an email message is appended, consisting of 3 parts: ENVELOPE (a single line containing summary information), HEAD (containing the email headers) and RFC822 (the original message, including headers and body). These are specified and their size is given, so the server knows what to do with the incoming raw data.

```
17 X-AKAPPEND 3 (\MimeType[message/rfc822]
   \RemoteId[/home/user/.maildir/cur/18972.localhost:2,ST])
   ("ENVELOPE":176,"HEAD":350,"RFC822":351) {877}
+ Ready for literal data (expecting 877 bytes)
<<compressed data>>
17 [UIDNEXT 112]
17 OK Append completed
```

A STORE command with disabled revision checking (NOREV, see the section about concurrency control for more information), which speeds up the operation. It stores the 'RFC822' part of the item with id '1'.

```
6 UID STORE 1 NOREV RFC822.SILENT {149}
+ Ready for literal data (expecting 149 bytes)
<<compressed data>>
6 OK STORE completed
```

Another STORE command, with revision checking enabled (REV 8). It removes the item part called 'ENVELOPE'.

```
8 UID STORE 3 REV 8 -PARTS(ENVELOPE)
8 OK STORE completed
```

An example of a FETCH command, where the attribute list can be seen. It requests several attributes from the item with id '9', namely UID (the internal id of the item), REMOTEID (the remote id of the item), FLAGS (the item's flags) and the part named 'RFC822'. In the response the values of these attributes are given, along with extra information, i.e., REV (the revision of the item), MIMETYPE (the mimetype of the item).

```
10 UID FETCH 9 (UID REMOTEID FLAGS RFC822)
* 10 FETCH (UID 9 REV 2 REMOTEID "2kx8Najlvk" MIMETYPE "text/directory"
     FLAGS (\Recent) RFC822 {224}
<<compressed data>>)
10 OK UID FETCH completed
```

Storing large objects

All PIM item parts are stored as BLOBs in the database, no exception is made for larger parts. Implementing this is straightforward, optimisations can be made in the future if performance is insufficient. See the Recommendations chapter for more information.

The parts are stored in compressed form, so storage requirements are somewhat lower than with uncompressed data.

Storing custom data

The extra data for every PIM item type is stored as item parts (see figure 4.7, the extra parts are represented by '. . . '). This solution is generic, because new extra fields can be added effortlessly. They can be stored in the existing database layout as it is.

Parts are identified by their name, some fixed part names are defined (body, header and envelope), as these are used extensively by PIM applications. Programs that use extra fields can create additional parts for their PIM items, which are stored in the database automatically along with the standard parts upon saving.

4.2.3 Data consistency

Database

The MySQL database uses the MyISAM engine [2] by default. It is fast and widely used, but misses some more advanced database features, such as transactions.

Therefore, the database is updated to use the InnoDB engine [3] instead. It is included in the standard MySQL distribution, there is no need for additional configuration. This engine supports ACID-compliant transaction processing and multiversion concurrency control. It also focuses on performance by using techniques to minimise disk I/O through efficient memory and processor use.

InnoDB's support for transaction isolation and foreign keys ensure data consistency with multiple simultaneous transactions and across multiple related tables respectively.

[2] http://dev.mysql.com/doc/refman/5.1/en/myisam-storage-engine.html
[3] http://dev.mysql.com/doc/refman/5.1/en/innodb.html

Being a transactional engine, InnoDB is more reliable, because in case of hardware crashes, transactions can be replayed. Updates are either performed completely or not at al, where they might be performed partially in a non-transactional engine, thus leaving the storage in an inconsistent state.

Concurrency control

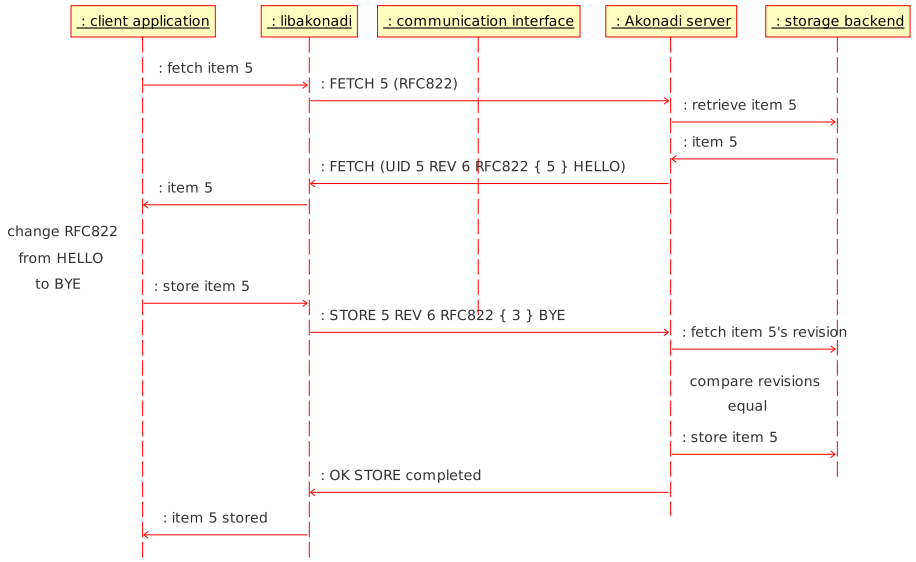

Figure 4.8: *Successful update*

Optimistic concurrency control is implemented, because of the low expected number of conflicting updates and the high number of reads. Its high performance and possibility for conflict resolving are important features for this decision.

The database is modified to supply every item with a revision number, which defaults to 0 for newly added items. Whenever an item is retrieved by a client application or component, its revision number is sent along with the rest of the data.

After retrieval, the client application may modify the item and request libakonadi to store the modifications. The item is sent to the server, which looks if the revision number is still the same in the storage backend. If so, the modifications are stored and the revision number is increased (figure 4.8).

If the revision numbers differ, the item was modified by another application, resulting in a conflict (figure 4.9). An error message is sent back to the application, which has several options to handle the situation. It can present the user with an error and just throw away the modifications. Another option is to fetch the item from the server and compare it with its own version. The modifications might not conflict, in which case the item can be stored, but if they do conflict, the user can be asked which version of the item to store.

Revision checking is enabled by default, but can be disabled (figure 4.10).

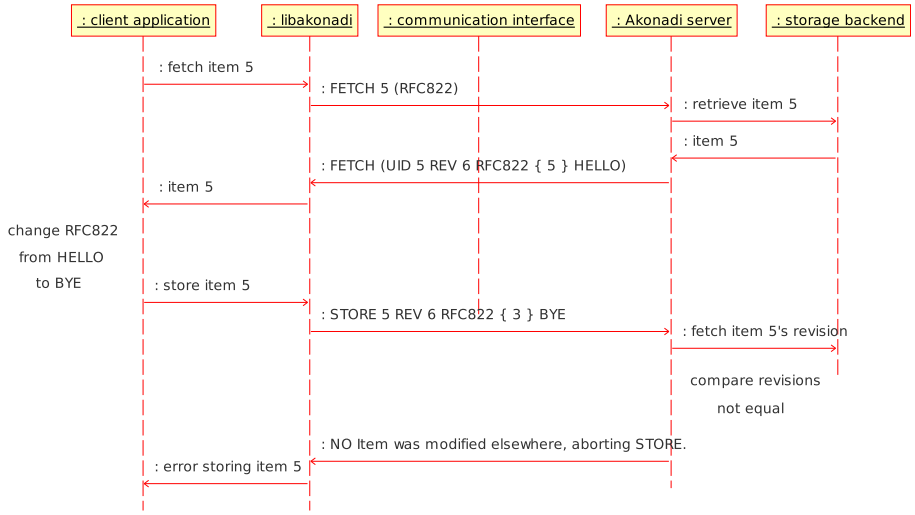

Figure 4.9: *Update failure due to conflict*

This might be desirable when items are first imported into Akonadi, or when the user explicitly requests that an item is deleted. The skipped check increases performance and avoids unnecessary error notifications in these cases.

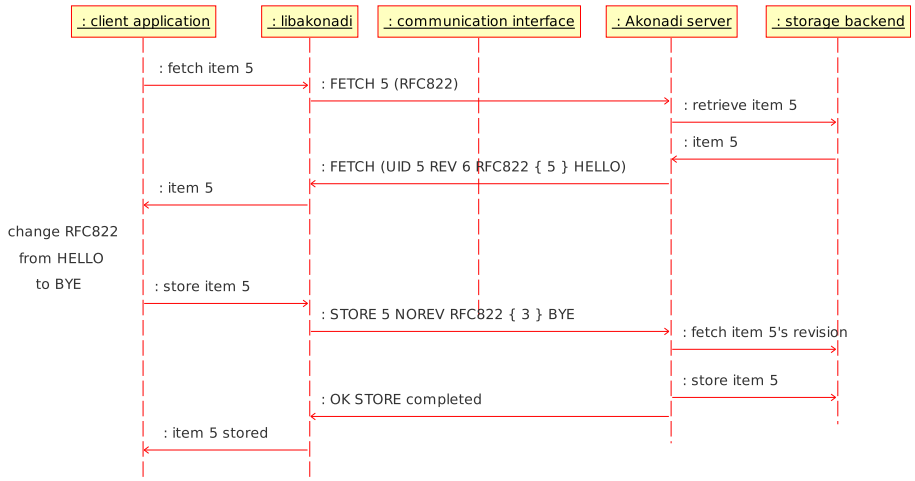

Figure 4.10: *Forced update*

Implementing revision checking has effect on the different Jobs in libakonadi and Handlers in the server. These have an added REV attribute, specifying the revision number of an item. When NOREV is used, revision checking is skipped by the server.

Chapter 5

Evaluation

To verify whether the suggested and implemented optimisations help to reach the stated goals, evaluations are performed. An evaluation plan is created, which is executed, after which its results are presented.

5.1 Evaluation plan

The evaluation will focus on performance and robustness, as this covers the implemented communication, storage and concurrency optimisations.

5.1.1 Dataset

The availability of a public set of PIM data would be useful, because it would make the tests repeatable for confirmation of their results and for evaluating future PIM storage systems.

Several public email corpora exist, some of which are listed below:

- PU123A corpora (APM03)
- SpamAssassin corpus (Mas05)
- SpamBase corpus (AN07)
- Ling-Spam (SAP+03)

These all suffer from the fact that they are relatively small (several thousand messages each). Also, they were all designed for spam classification purposes, so they contain a lot of spam messages. The purpose of Akonadi is to store normal (so-called ham) emails, which is thus what we want to use as evaluation data.

Another well-known dataset is the one from Enron (Coh05), which contains more than half a million messages from over 150 users, mostly senior management of Enron. It became public during the legal investigation of the Enron corporation (KY04). It is chosen, because it lacks the downsides of the other corpora. Unfortunately, it does not contain any attachments, therefore a script augments it with (fake) attachments.

5.1.2 Robustness tests

Unit tests The Akonadi software already had a set of unit tests, which makes it easy to continuously verify the correct working of its components.

This set of tests is extended with ones that check handling multipart items and optimistic locking. These automated tests help to ensure that the implemented modifications keep working as expected. If regressions or incorrect behaviour is introduced (unintentionally), it is detected and can be fixed immediately.

Multipart items

1. append a multipart item (should invoke the X-AKAPPEND command, instead of the APPEND command)
2. fetch several parts from a multipart item
3. fetch all parts of a multipart item
4. update an item, add extra parts
5. update an item, remove a part
6. delete a multipart item (should automatically delete all item parts, because of referential integrity)

Optimistic locking

1. - open an email in two programs
   - delete it in the first program
   - change its status to 'important' in the other program
   - required effect: the other program gives an error message, informing the user about the deleted email

   The application may present the user the option to recreate the email (effectively undeleting it).

2. - open an event in an application
   - let the event's resource modify the event
   - edit the event in the application and save it
   - required effect: the application gives an error message, informing the user about the outdated data

   The application could compare its version of the event with the modified one on the server, but that is no requirement for Akonadi itself.

3. - delete an item
   - required effect: item, along with all item parts, is deleted from the storage

   This test checks the referential integrity of the InnoDB engine. Deletes should be cascaded to the referenced tables, in this case a delete in the PimItems table should delete all referring entries in the Parts table.

5.1.3 Performance tests

For the sizes and amounts in the use cases, realistic numbers are used, extracted from the Enron dataset (KY04). According to Klimt and Yang, the number of folders of a user's mailbox is at most a log of the number of messages in the mailbox. Therefore the numbers below adhere to that analysis.

Compression To test whether the applied data compression is useful, an average-sized and the largest accounts (according to their required storage space) from the email dataset are imported into Akonadi. The import consists of reading the accounts from disk, storing them in the Akonadi storage, which signals the Strigi feeder to retrieve and index their contents. This means that the test benchmarks a roundtrip of the data to and from the Akonadi storage.

The required time and storage space are measured for multiple scenarios: the account with and without added attachments and Akonadi with and without compression.

Multipart The performance differences between singlepart and multipart item storage are tested, with the same accounts that the compression test uses. The singlepart test stores all messages of the account directly in the Akonadi storage, whereas the multipart test splits them into parts and stores them afterwards.

After the import, the retrieval performance is measured by retrieving all the message headers from the storage. The singlepart test needs to retrieve the full message and separate the header, the multipart test only needs to retrieve the already stored header part of the message.

The required time and storage space are measured in this test as well; again for the accounts with and without added attachments, and with and without using Akonadi's multipart support.

Use cases These tests evaluate the complete system, not the implemented modifications specifically. However, when a test is especially relevant to certain areas of optimisation, they are named.

The tests are run with each individual account in the complete Enron email dataset, both without and with added attachments.

1. import the complete mailbox (multipart, compression)
2. fetch all headers from each folder (multipart)
3. mark 20% of messages as read (protocol)
4. fetch headers of unread messages from each folder (multipart)
5. remove all read messages from each folder (protocol, database)
6. remove every folder sequentially (database)

5.2 Evaluation results

5.2.1 Test systems

The system on which the benchmarks are performed has the following configuration.

Hardware configuration

- AMD Athlon X2 3800+ processor
- 2x 512 MB PC3200 DDR memory
- Hitachi T7K500 320 GB S-ATA hard drive

Software configuration

- Gentoo Linux
- Linux 2.6.23 kernel with SMP-support
- GCC 4.2.2
- glibc 2.7
- MySQL 5.0.44 Embedded
- Xorg 7.3
- Qt 3.3.8 (working environment) and 4.3.2 (development)
- KDE 3.5.8 (working environment)

5.2.2 Robustness

All the executed use cases result in the required effect. Unfortunately, none of the KDE applications use the Akonadi server yet, so only synthetic tests can be done.

Also, the additionally desired effects (giving the user the option to resolve the conflicting transactions) are not implemented yet. Part of this can be done in Akonadi (simple automatic conflict resolution), whereas the client applications should implement the ability to let the user resolve the conflict.

Executing the use cases with concurrency control disabled (i.e., using the NOREV parameter instead of REV) results in the expected conflicted updates.

5.2.3 Performance

Compression

For the average sized account, the one named *hernandez-j* is chosen, it has 3265 messages and takes up 18 MB and 34 MB of disk space, respectively without and with the added attachments. The largest account of the dataset is the one named *dasovich-j*, having 28234 messages, taking up 203 MB and 343 MB of disk space, respectively without and with the added attachments.

Table 5.1 shows the results. The reduction in storage space depends on the type of data, the version of the account without attachments consists mainly of textual data and is reduced by about 50%, a lot more than the version with attachments, which is only reduced by around 40% (still a significant reduction).

The time to import and index the data is reduced considerably as well. This indicates that the data compression and decompression have less impact than the data transfer.

The compression results also indicates good future prospects, as the savings increase with larger accounts.

	uncompressed	compressed	reduction
hernandez-j			
without attachments			
time	34 s	31 s	9%
space	36 MB	19 MB	47%
with attachments			
time	35 s	32 s	9%
space	56 MB	36 MB	36%
dasovich-j			
without attachments			
time	995 s	865 s	13%
space	212 MB	96 MB	55%
with attachments			
time	1147 s	904 s	21%
space	380 MB	224 MB	41%

Table 5.1: *Compression results*

Multipart

Executing the multipart tests with accounts *hernandez-j* and *dasovich-j* renders the results as shown in table 5.2. The initial import of the mailbox takes roughly 15% more time (because of the extraction and storage of multiple parts). The required storage space also increases, by about 35%. This sounds like a big disadvantage, but as the account sizes increase, the negative effect is lower. Also, the initial import occurs only once.

When only a small part of the data is needed, performance is about 30% better than that of the singlepart version. With larger accounts, the gain rises as well, up to 42% with the largest account in the dataset. These kind of transactions occur frequently, and their performance is therefore more significant than that of low-frequency actions.

Use cases

The results of the use case tests are provided in the form of a series of figures. For each test, two figures are shown: one showing the relation between the number of messages in a mailbox and the time to execute the test, the other

	singlepart	multipart	reduction
hernandez-j			
without attachments			
import time	14.7 s	22.3 s	-52%
fetch headers time	3.5 s	2.6 s	26%
space	14 MB	19 MB	-36%
with attachments			
import time	22 s	25.9 s	-18%
fetch headers time	4.1 s	3 s	27%
space	29 MB	36 MB	-24%
dasovich-j			
without attachments			
import time	851 s	919 s	-8%
fetch headers time	33.6 s	27.9 s	17%
space	96 MB	148 MB	-54%
with attachments			
import time	899 s	1015 s	-13%
fetch headers time	50 s	29 s	42%
space	224 MB	276 MB	-23%

Table 5.2: *Singlepart and multipart results*

showing the relation between the total mailbox size on disk and the time to execute the test.

The graphs show that the size of the individual messages has almost no impact on the required time, regardless of the use case. The messages without attachments are several kilobytes each, the attachments vary between 5 and 500 kilobyte, thus serioesly increasing the message size. However, the plot of the dataset without attachments and the one with attachments are almost equal (clearly visible in the figures showing the number of messages against the execution time, while the other kind of figures shows similar looking, but horizontally stretched plots).

This can indicate that the transfer time of the actual content data of the messages is negligible in comparison to the overhead of processing the commands, creating and handling the message objects and everything else that is needed for executing the use case.

It should be noted that the largest mailboxes are several hundreds of megabytes. This influences the results, because of the hardware configuration of the test computer. The internal memory of the computer is completely filled with these mailboxes, at which point the operating system falls back to swapping, which decreases performance significantly.

Import the complete mailbox This use case (figure 5.1) shows better than lineair scaling for mailboxes under 150 MB. Larger accounts seem to hit the memory capacity of the computer, as previously mentioned.

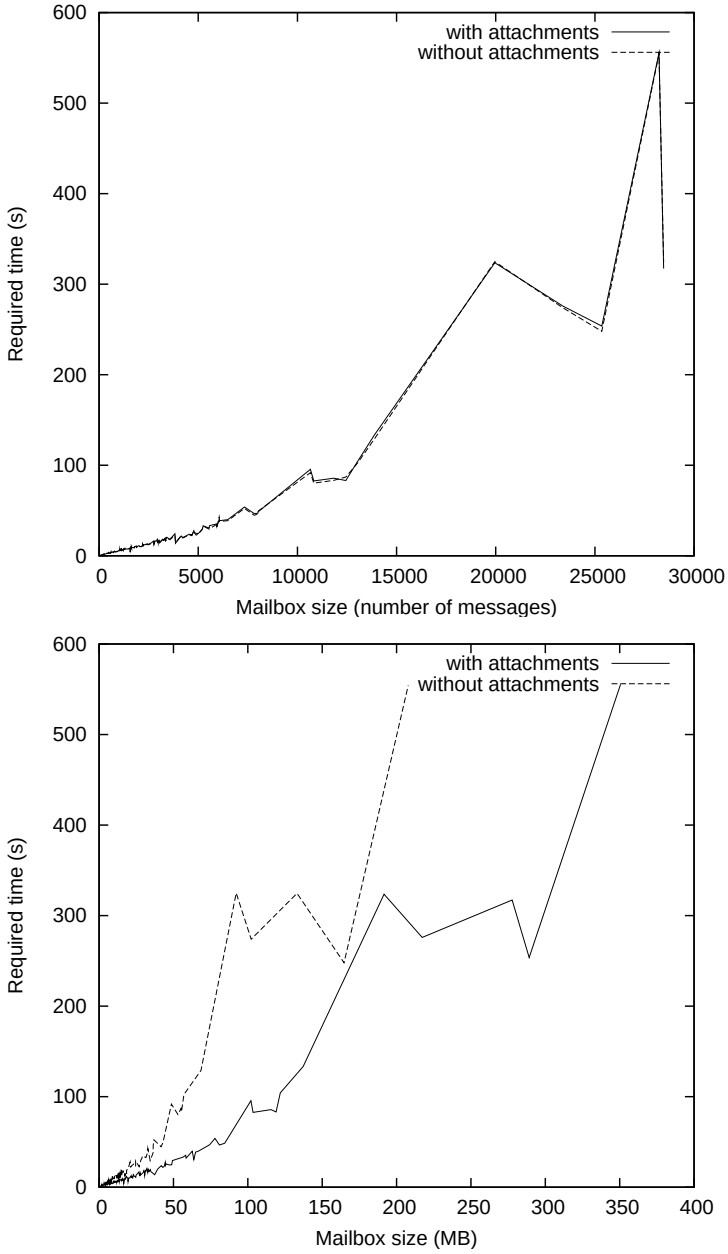

Figure 5.1: *Import complete mailbox*

Fetch all headers from each folder Up until 200000 messages, or 200 MB, the scaling is almost linear in this test (figure 5.2), which is good.

Mark 20% of messages as read Like in the previous test, the results of this one (figure 5.3) are linear, for all mailbox sizes.

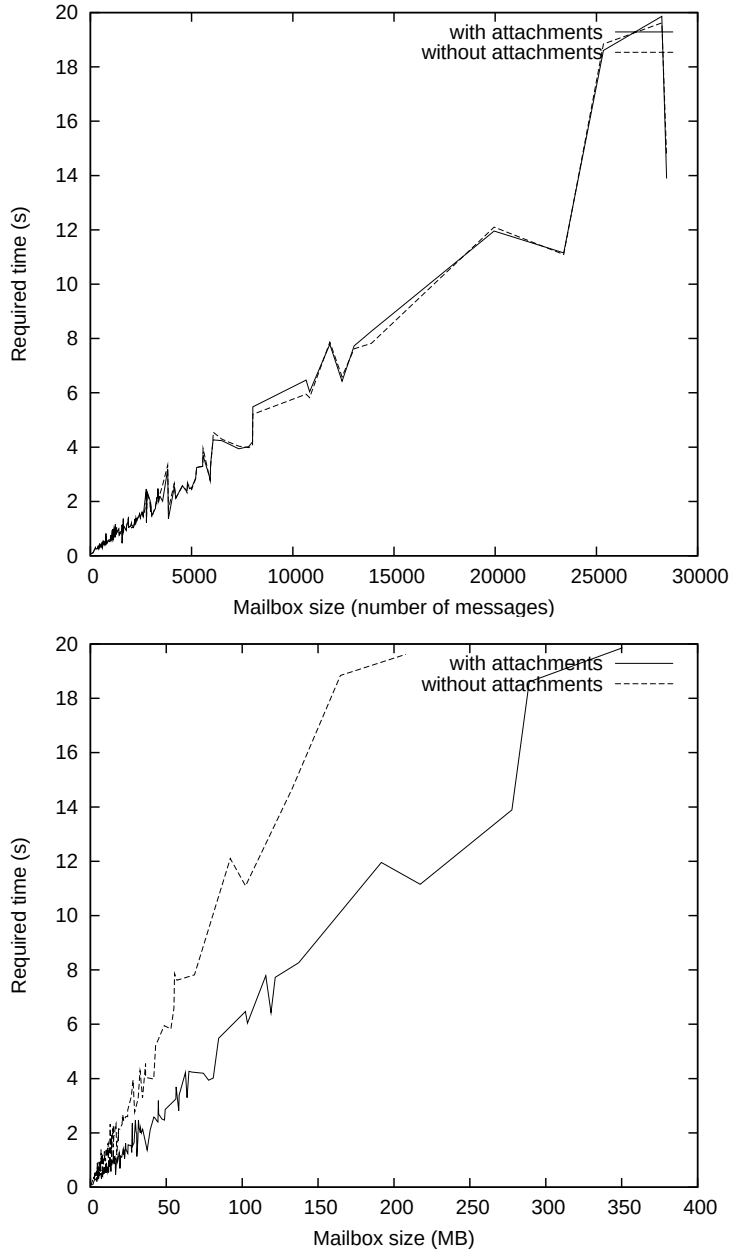

Figure 5.2: *List headers of all messages of every folder*

Fetch headers of unread messages from each folder This test's results (figure 5.4) also suffers from the capacity problem. With more average-sized accounts, it scales a bit better than linear, which is good.

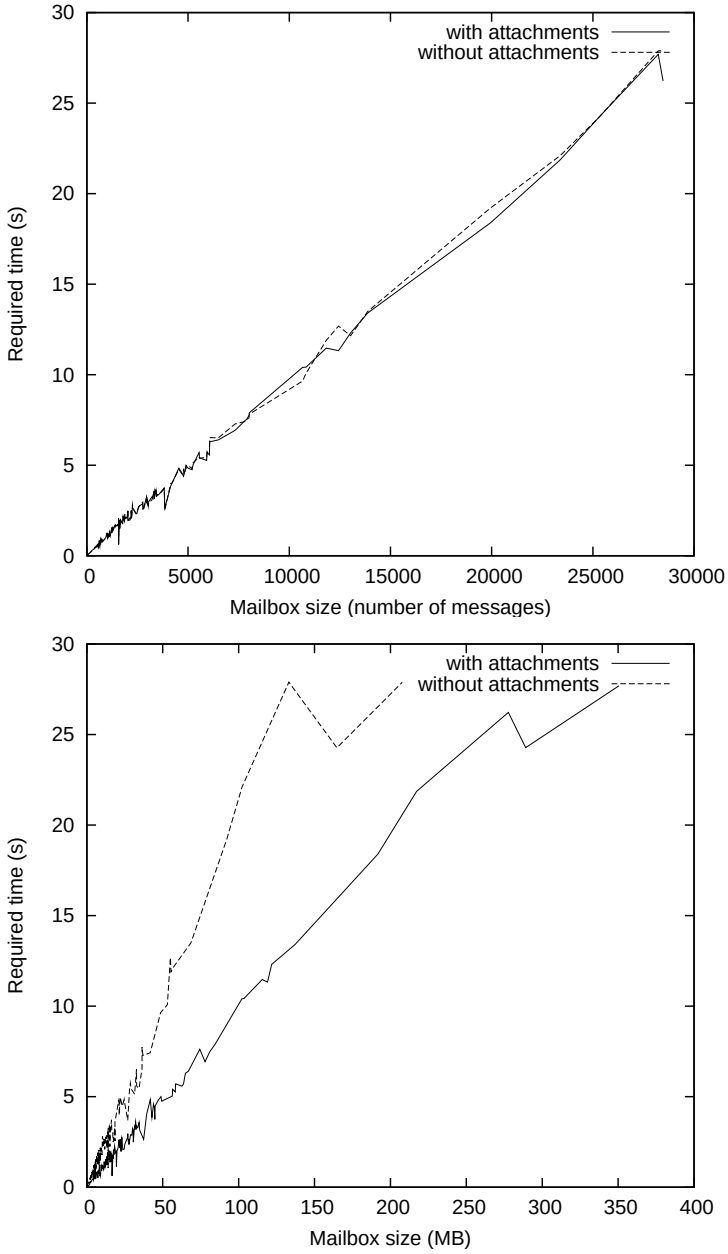

Figure 5.3: *Mark 20% of messages read*

Remove all read messages from each folder For small mailboxes, consisting of several thousands of messages, the results (figure 5.5) are much better than linear. However, with larger mailboxes it gets a bit worse. Unfortunately, there are not many large mailboxes in the dataset, so those results are not very usable.

The plots of this test show similarities to those of the "import the complete mailbox" use case.

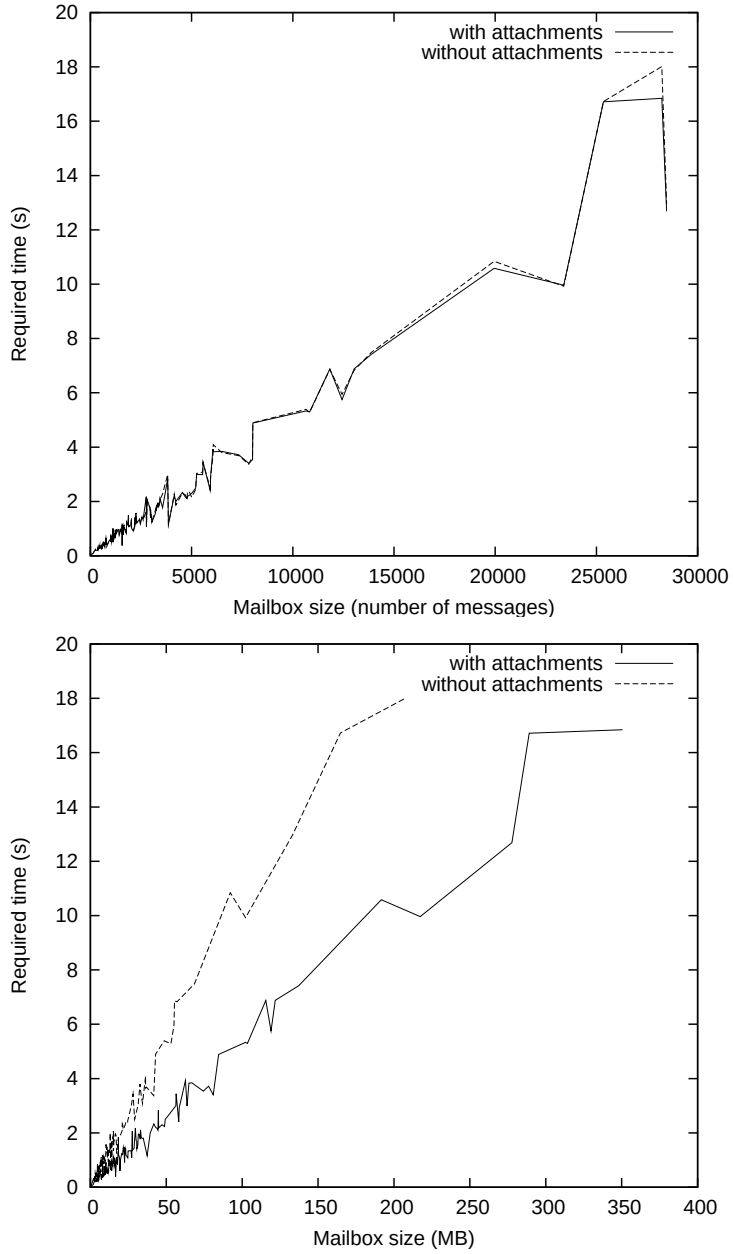

Figure 5.4: *List headers of unread messages of every folder*

Remove every folder sequentially Again, the results (figure 5.6) are a bit better than linear

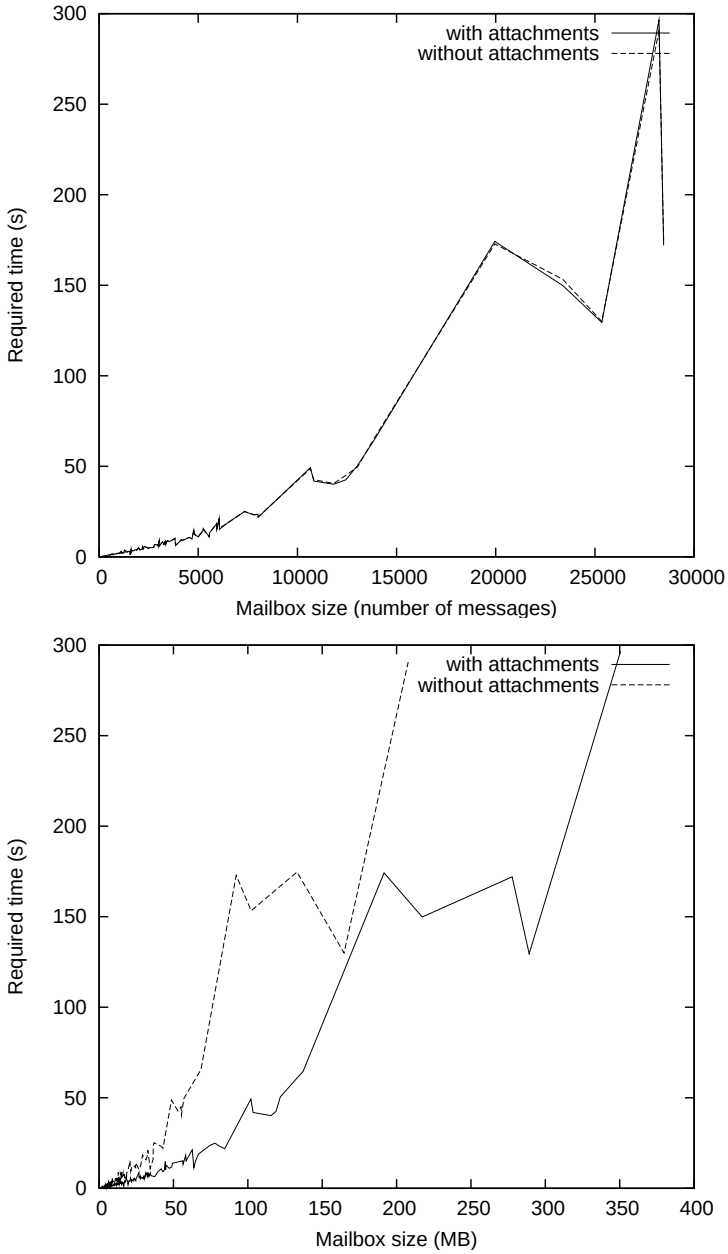

Figure 5.5: *Remove read messages from every folder*

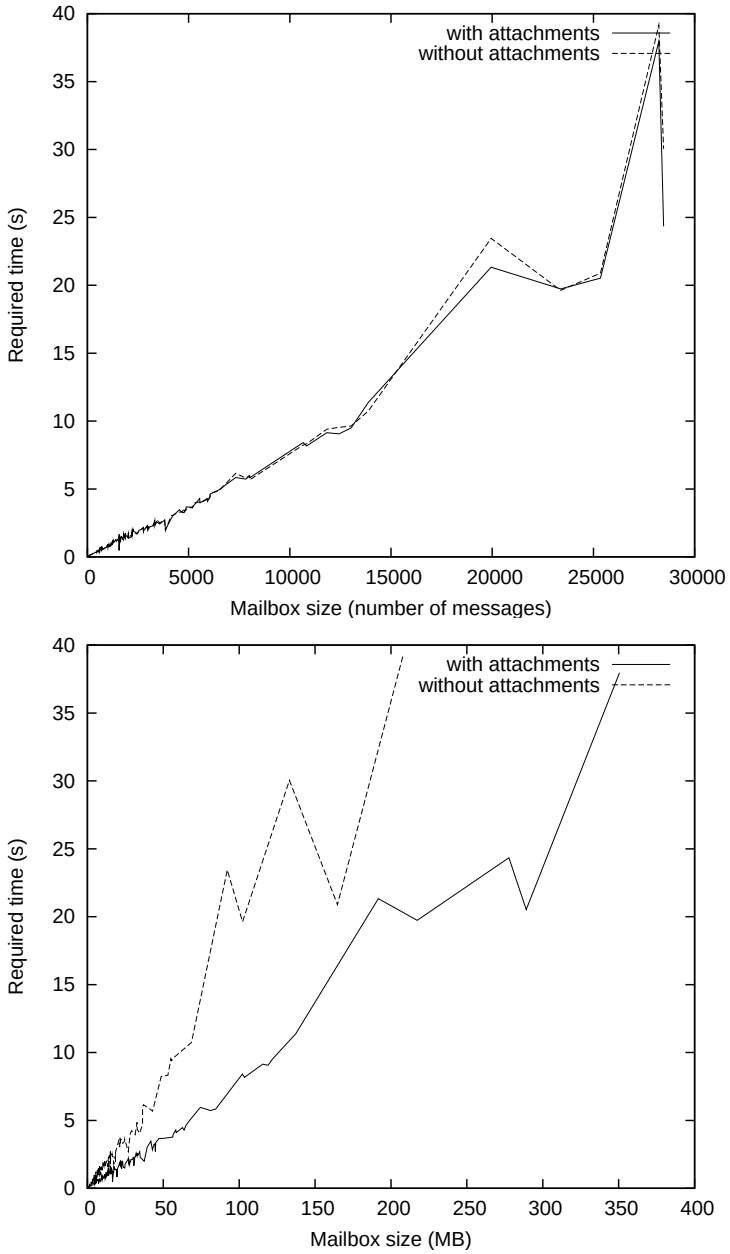

Figure 5.6: *Remove every folder sequentially*

Chapter 6

Conclusions

How to transfer PIM data between the storage component and the applications/libraries using it with high performance? The D-Bus protocol does a good job of transferring commands and notifications between Akonadi's components. Its performance is not directly evaluated, as only small messages are transmitted over it. The advantages are clear: it is compatible with a lot of desktop software, not only KDE, it works desktop-wide and is easy to use.

Akonadi's IMAP-like data access protocol is successful. It offers great performance, which can be concluded from the linearity of the evaluation results.

Keeping the PIM items in their native format prevents encapsulation and conversion. It does not lower performance, the test results show that the data size of the PIM items has a negligible impact on the required time to execute tasks.

Compressing the data before transferring and storing it helps to increase the performance by around 15%, depending on the type of data. In addition, it saves storage space by about 40%, which helps using the memory capacity to its full extent (see next paragraph). The processing requirements for the compression and decompression play an insignificant role, especially with today's high performance computers.

How to store PIM data, consisting of tabular, textual and binary parts, efficiently and consistently? Using a relational database as a storage backend works fine. Performance is good, up until the memory capacity of the computer, from where it decreases significantly.

The use of MySQL's InnoDB engine in conjunction with transactions and cascaded updates and deletes keeps the storage in a consistent state. Support for multiple, simultaneous connections to the storage backend is thus possible.

Storing large objects as BLOBs performs satisfactory, the aforementioned data compression helps in reducing their size.

Storing the data in multiple parts proves useful. The initial import of items into Akonadi's storage takes a lot more time, varying around 25%, while also using about 30% more storage space. After this one time operation, the

multipart support pays off, which is clearly visible in the (frequently happening) fetching of message headers, which consumes around 30% less time.

Storing custom generated data as extra item parts is not fully evaluated, as no client applications (that generate these parts) are using Akonadi yet. The support for it is in place, however, and works fine. The performance is comparable to that of the evaluated multipart support.

The added optimistic concurrency control makes sure that conflicting updates to PIM items are detected and allows for automated or manual recovery in case of a conflict.

What are the performance and robustness of Akonadi? Akonadi's scales well with increasing data volumes, making it ready for future PIM usage scenarios. The absolute performance needs to be improved, as well as the processing of large datasets.

The results of the robustness tests are satisfactory, but extended and continued testing is recommended, as the past shows endless possibilities of data loss and computer failure, not to mention software bugs.

The evaluation methods are useful to test the progress of Akonadi, as well as other PIM storage systems. They evaluate the results of the implemented modifications and stay valuable during further development, because the real-world scenarios they simulate.

Chapter 7

Recommendations for future work

Support for other storage backends Firebird might be a good alternative to MySQL, but performance comparisons should be done prior to implementing support for it, to prevent disappointments.

When object databases become available as free software, they might be worth investigating as an alternative to the currently used relational database. This also goes for XML databases, although to a somewhat lesser extent, because they require lots of work to the inner structure of the current implementation of Akonadi.

Object/relational mappers These simplify the database access code significantly, but no mature and free packages exist for C++ at the time of writing. The reduced code size and complexity decreases the chance of programming errors and improves readability. When they do become available, using them is advised.

Storing large objects Storing large binary objects directly in the filesystem is only a temporary solution. It is also a workaround, because databases should be able to optimally store large objects by themselves.

The problem is not urgent yet, as most multimedia content is stored on the filesystem these days. When emails grow to several hundreds of megabytes and multimedia data is seen as PIM data, Akonadi has to cope with really large objects.

The Scalable BLOB Streaming Infrastructure for MySQL (mbH07) is an effort to create a universal method for storing large objects. The MyBS engine and MySQL's own new Falcon engine (MyS07) are examples of engines with focus on efficient BLOB storage, that can be used in companion with the new infrastructure. In the mean time, MySQL's FILE privilege [1] might be an option to get higher performance. As the Akonadi server runs on a per user basis and

[1] http://dev.mysql.com/doc/refman/5.0/en/privileges-provided.html

only allows local connections, it poses no security vulnerability (the MySQL server has only read and write access to the files that the user has access to).

Storing large datasets A related issue is when more data is stored than what fits in the computer's memory, which is a realistic scenario with today's large amounts of PIM data. This is devastating for Akonadi's performance. Smart caching is a way of tackling the problem. The cache cleaner agent already exists, but it is not yet completely implemented.

Compression While data compression proves to be increasing performance, while saving storage space, this is mainly due to the textual nature of the data. When already compressed data parts (such as e-mail attachments and compressed images) are stored, these advantages vanish, but the increased computation time remains.

A solution is to make the compression dependent on the mimetype of the data. Types such as *application/x-tgz*, *application/x-zip* and *image/png* are already compressed, where *text/plain* is not. It is even possible to use different compression algorithms, depending on the data to be compressed.

Multipart support Newly developed resources and applications should make good use of Akonadi's support for multipart items. Measurements should be done on the frequency of actions, to determine what kind of data is stored and retrieved mostly. This data can then be stored as a separate item part, thus lowering the time needed for storage, retrieval and further processing.

Data consistency The PIM applications supporting Akonadi as their backend should add functionality to prevent data loss. This includes presenting the user with options to resolve conflicts and automatic conflict resolving where Akonadi is not able to do so.

System failures stay a risk, even with lots of protection measures. Integrating a way to backup Akonadi's data should therefore be done before releasing a stable version.

Performance evaluation It is recommended to execute performance tests on Akonadi and its competitors, to be able to compare the numbers and learn smart approaches from each other.

Bibliography

[ACI07] Acid properties in databases. http://en.wikipedia.org/wiki/ACID, 2007.

[Amb06] Scott W. Ambler. Mapping objects to relational databases: O/r mapping in detail. http://www.agiledata.org/essays/mappingObjects.html, 2006.

[AN07] A. Asuncion and D.J. Newman. UCI machine learning repository. 2007.

[Apa07] Xindice. http://xml.apache.org/xindice/, 2007.

[APM03] I. Androutsopoulos, G. Paliouras, and E. Michelakis. Pu123a corpora. http://www.iit.demokritos.gr/skel/i-config/downloads/PU123ACorpora_readme.txt, 2003.

[Bea07] Beagle desktop search. http://beagle-project.org, 2007.

[Ber95] Daniel J. Bernstein. Using maildir format. http://cr.yp.to/proto/maildir.html, 1995.

[BG81] Philip A. Bernstein and Nathan Goodman. Concurrency control in distributed database systems. *ACM Comput. Surv.*, 13(2):185–221, 1981.

[Chr03] M. Chrispin. Rfc 3501 - internet message access protocol - version 4rev1. http://www.faqs.org/rfcs/rfc3501.html, 2003.

[Coh05] William W. Cohen. Enron email dataset. http://www.cs.cmu.edu/~enron/, 2005.

[Con07] Nepomuk Consortium. Nepomuk - the social semantic desktop. http://nepomuk.semanticdesktop.org/, 2007.

[CWI04] CWI. Monetdb/xquery. http://monetdb.cwi.nl/projects/monetdb/Home/index.html, 2004.

[DCO07] Desktop communication protocol. http://en.wikipedia.org/wiki/DCOP, 2007.

[DS98] F. Dawson and D. Stenerson. Rfc 2445 - internet calendaring and scheduling core object specification (icalendar). http://www.ietf.org/rfc/rfc2445.txt, 1998.

[Dum07] Susan Dumais. The person in personal; 16th international world wide web conference (www2007). http://research.microsoft.com/~sdumais/WWW2007-Dumais-Share.pdf, 2007.

[EP03] Nick Elprin and Bryan Parno. An analysis of database-driven mail servers. *Proceedings of the 17th Large Installation Systems Administration Conference*, pages 15–22, 2003. https://www.usenix.org/events/lisa03/tech/full_papers/elprin/elprin_html/.

[fil07] Comparison of file archivers. http://en.wikipedia.org/wiki/Comparison_of_file_archivers, 2007.

[FSF07] The free software definition. http://www.gnu.org/philosophy/free-sw.html, 2007.

[Har05] Elliotte Rusty Harold. Managing xml data: Native xml databases. http://www-128.ibm.com/developerworks/xml/library/x-mxd4.html?ca=dnt-623, 2005.

[HD91] U. Halici and A. Dogac. An optimistic locking technique for concurrency control in distributed databases. *Software Engineering, IEEE Transactions on*, 17(7):712–724, 1991.

[KDE07] Kde homepage. http://www.kde.org/, 2007.

[KJ06] David R. Karger and William Jones. Data unification in personal information management. *Commun. ACM*, 49(1):77–82, 2006.

[KK07] Tobias König and Volker Krause. Akonadi documentation. http://www.englishbreakfastnetwork.org/apidocs/apidox-kde-4.0/kdepim-apidocs/akonadi/html/index.html, 2007.

[KY04] Bryan Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. In *Machine Learning: ECML 2004; 15th European Conference on Machine Learning, Pisa, Italy, September 20-24, 2004. Proceedings*, pages 217–226, Language Technologies Institute, Carnegie Mellon University, Pittsburgh, PA 15213-8213, USA, 2004. Springer Berlin / Heidelberg.

[KZ07] Tobias König and Robert Zwerus. Akonadi design. http://www.englishbreakfastnetwork.org/apidocs/apidox-kde-4.0/kdepim-apidocs/akonadi/html/akonadi_design.html, 2007.

[LDA07] Lightweight directory access protocol. http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol, 2007.

[Mas05] J. Mason. Spamassassin corpus. http://spamassassin.apache.org/publiccorpus/, 2005.

[mbH07] SNAP Innovation Softwareentwicklungsgesellschaft mbH. Scalable blob streaming infrastructure for mysql. http://www.blobstreaming.org/, 2007.

[mbo07] mbox storage format. http://en.wikipedia.org/wiki/Mbox, 2007.

[Mei07] Wolfgang Meier. exist - open source native xml database. http://exist.sourceforge.net/, 2007.

[Mul07] Multiversion concurrency control. http://en.wikipedia.org/wiki/Multiversion_concurrency_control, 2007.

[MWZ04] Qirong Mao, Jinfeng Wang, and Yongzhao Zhan. The optimistic locking concurrency controlling algorithm based on relative position and its application in real-time collaborative editing system. *Computer Supported Cooperative Work in Design, 2004. Proceedings. The 8th International Conference on*, 1:99–105, 2004.

[MyS07] MySQL. Falcon storage engine guide. http://dev.mysql.com/doc/falcon/en/index.html, 2007.

[Obj07] Object database. http://en.wikipedia.org/wiki/Object_database, 2007.

[Opt07] Optimistic concurrency control. http://en.wikipedia.org/wiki/Optimistic_concurrency_control, 2007.

[PCL06] H. Pennington, A. Carlsson, and A. Larsson. D-bus specification. http://dbus.freedesktop.org/doc/dbus-specification.html, 2006.

[Pre06] Double Precision. Courier extended maildir. http://www.courier-mta.org/maildir.html, 2006.

[SAP+03] G. Sakkis, I. Androutsopoulos, G. Paliouras, V. Karkaletsis, C.D. Spyropoulos, and P. Stamatopoulos. A memory-based approach to anti-spam filtering for mailing lists. *Information Retrieval*, 6(1):49–73, 2003.

[Sch07] Christian Schaarschmidt. Akonadi diagram in detail. http://www.englishbreakfastnetwork.org/apidocs/apidox-kde-4.0/kdepim-apidocs/akonadi/html/akonadi_overview_uml.html, 2007.

[Sta01] Kimbro Staken. Introduction to native xml databases. http://www.xml.com/pub/a/2001/10/31/nativexmldb.html, 2001.

[Tim07] Timestamp-based concurrency control. http://en.wikipedia.org/wiki/Timestamp-based_concurrency_control, 2007.

[Tro07] Trolltech homepage. http://trolltech.com/, 2007.

[Tru07] Sebastian Trueg. Nepomuk-kde. http://nepomuk-kde.semanticdesktop.org/xwiki/bin/view/Main/, 2007.

[vdO07] Jos van den Oever. Strigi desktop search. http://strigi.sourceforge.net/index.php/Main_Page, 2007.

[XML07] Xml database. http://en.wikipedia.org/wiki/XML_database, 2007.

[zli07] zlib data compression. http://en.wikipedia.org/wiki/Zlib, 2007.

Appendix A

GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed

under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title

page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you

must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all

of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright

holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Summary

One of the main purposes of personal computers today is to store and manage our personal information. We use organiser software instead of paper organisers, we don't clutter our colleagues' desks with written notes, but send them emails, we keep a blog for a diary, we could even think of digital photo and video albums. As the size of all this data increases, it becomes more important to think about the way we store it. We expect instant response from our computer, which makes performance a main aspect to focus on. Also, we'd hate losing information, so stability, robustness and protection from all kinds of data loss should also be taken into account.

In this thesis, several options for achieving the above-mentioned goals are looked into. Also, some of the options are implemented in an existing project, namely Akonadi, part of KDE. KDE is a desktop environment for Linux, several flavors of Unix and (in the future) Windows. It contains a complete set of applications to manage our personal information, like an email client, an organiser, a blogging application and so forth. In KDE 4, the next major version of the desktop environment, Akonadi is to be responsible for storing and accessing all the data these applications handle.

Naturally, an evaluation of the implemented options is performed and shows that they have the desired effects. Akonadi's performance is tested, as well as its protection against data loss.