Integrated Project

Priority 2.4.7

Semantic based knowledge systems

Information Society Technologies

The Social Semantic Desktop

NEPOMUK

KDE Community Involvement

Deliverable D7.2

Version 1.1

08.01.2007

Dissemination level: PU

Nature	Other
Due date	31.12.2006
Lead contractor	EDGE-IT S.A.R.L
Start date of project	01.01.2006
Duration	36 months

Authors

Sebastian Trüg, EDGE-IT S.A.R.L
Stéphane Laurière, EDGE-IT S.A.R.L
David Barth, EDGE-IT S.A.R.L

Mentors

Malte Kiesel, DFKI

Project Co-ordinator

Dr. Ansgar Bernardi
German Research Center for Artificial Intelligence (DFKI) GmbH
Erwin-Schroedinger-Strasse (Building 57)
D 67663 Kaiserslautern
Germany
Email: bernardi@dfki.uni-kl.de, phone: +49 631 205 3582, fax: +49 631 205 4910

Partners

DEUTSCHES FORSCHUNGSZENTRUM F. KUENSTLICHE INTELLIGENZ GMBH
IBM IRELAND PRODUCT DISTRIBUTION LIMITED
SAP AG
HEWLETT PACKARD GALWAY LTD
THALES S.A.
PRC GROUP - THE MANAGEMENT HOUSE S.A.
EDGE-IT S.A.R.L
COGNIUM SYSTEMS S.A.
NATIONAL UNIVERSITY OF IRELAND, GALWAY
ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE
FORSCHUNGSZENTRUM INFORMATIK AN DER UNIVERSITAET KARLSRUHE
UNIVERSITAET HANNOVER
INSTITUTE OF COMMUNICATION AND COMPUTER SYSTEMS
KUNGLIGA TEKNISKA HOEGSKOLAN
UNIVERSITA DELLA SVIZZERA ITALIANA
IRION MANAGEMENT CONSULTING GMBH

Versions

Version	Date	Reason
0.1	04.12.2006	First draft by Sebastian Trüg
0.2	22.12.2006	Second version presented to mentors
0.3	27.12.2006	Considered comments from mentors
1.0	29.12.2006	Final version
1.1	08.01.2006	Some streamlining by M. Junker and submission

Explanations of abbreviations on front page

Nature
R: Report
P: Prototype
R/P: Report and Prototype
O: Other

Dissemination level
PU: Public
PP: Restricted to other FP6 participants
RE: Restricted to specified group
CO: Confidential, only for Nepomuk partners

Executive summary

To be really successful, the Nepomuk project needs not only to realize and deploy an innovative approach for collaborative knowledge work, but also to integrate the Nepomuk framework into mainstream desktop environments and development platforms. Community work is hence a very important part of the project. Already in early stages of the project open-source communities have to be involved in the development process to make them aware of the progress in Nepomuk.

Nepomuk-KDE is a sub-project of Nepomuk which aims to provide a full implementation of the standards and APIs defined in Nepomuk on the KDE Desktop by introducing the technologies to the KDE community and helping with an integration as a central KDE technology.

KDE is a powerful Free Software graphical desktop environment for Linux and Unix workstations. It consists of an elaborate development framework, a large collection of desktop applications including a complete office suite and all-day-tools like an email client and a powerful internet browser. KDE is the leading desktop environment on Unix derivates. The semantic features found in KDE are however minimalistic if existant at all. Nepomuk-KDE sets out to change this.

Steps have been undertaken to get the KDE community involved in Nepomuk. The Nepomuk-KDE project was presented at the annual KDE developer conference aKademy 2006 in Dublin. A wiki has been created to present the goals and progress of the project, and also to raise interest across the community. Many discussions have been initiated within the KDE community.

As a first result of the Nepomuk-KDE project, the Nepomuk-KDE middleware and the core services providing important features like RDF storage have been implemented. They integrate well with the existing KDE framework. Two simple annotation and tagging applications have been developed to present the capabilities of the Nepomuk-KDE framework. An existing desktop search tool has been enhanced to also search for metadata created via the Nepomuk-KDE tools.

The Nepomuk-KDE project is already a success: a working implementation of the Nepomuk middleware including the core services has been realized and made public to the KDE community, and the Nepomuk-KDE components are scheduled to be included into the kdelibs, which is the core of KDE which each KDE component is based upon. This will drastically improve the awareness of Nepomuk-KDE and Nepomuk in general. Hopefully it will also bring new developers to the effort and help speeding up the realization of a social semantic KDE desktop.

Table of contents

1 Introduction

> Nepomuk intends to realize and deploy a comprehensive solution – methods, data structures, and a set of tools – for extending the personal computer into a collaborative environment, which improves the state of art in online collaboration and personal data management and augments the intellect of people by providing and organizing information created by single or group efforts.

To be really successful, the Nepomuk project needs not only to realize and deploy an innovative approach for collaborative knowledge work, but also to integrate the Nepomuk framework into mainstream desktop environments and development platforms. In accordance with the Metcalfe's law[1] related to the network effects of communication technologies, the value of the Nepomuk platform will depend on the number of its adopters. Thus, since Nepomuk is developed as an open and extensible platform, community work is a very important part of the project. Already in early stages of the project, open-source communities have to be involved in the development process to make them aware of the progress in Nepomuk, let them know their input is important, and finally trigger their interest in the project's technologies.

Nepomuk is a desktop project which immediately leads to the big players in open-source desktops today. The K Desktop Environment, short KDE, is certainly one of the most well known and vastly used desktop environments available on the Linux/Unix platform. An integration of Nepomuk APIs, ontologies and technologies into the KDE system would mean a big step towards real acceptance and use of the Nepomuk standards.

As a result of these considerations, in October 2006 the Nepomuk-KDE project was born. Nepomuk-KDE is a sub-project of Nepomuk which aims to provide a full implementation of the standards and APIs defined in Nepomuk on the KDE Desktop by introducing the technologies to the KDE community and helping with an integration as a central KDE technology. As a sub-project of Nepomuk, the main issues are (i) the use of interoperable metadata throughout the desktop applications (ii) powerful peer-to-peer collaboration features (iii) advanced user interfaces for dealing with semi-structured data.

Implementing the Nepomuk standards in an established system like KDE has many advantages over a reference implementation using the Java programming language as used throughout the Nepomuk project. KDE has a broad user base which means that the technologies introduced in KDE reach a broad audience and get tested thouroughly. Also components developed especially for a platform like KDE are easy to integrate with the existing framework and existing applications. This is also true for the non-technological area: application developers will be eager to pick up and use new KDE technologies in their applications. KDE already provides many useful technologies that otherwise would have to be re-implemented, like for example a full Linux inotify[2] integration, which makes monitoring file system changes very easy.

In the first phase of the Nepomuk-KDE project, the focus lay on the metadata part. There are basically three kinds of metadata to be found on the desktop:

1. Metadata that can be found in files stored on the local hard disk like tag information in audio files, timestamps, or simple indexed text. This metadata can be extracted and indexed at any time and is exactly the type of information current desktop search projects like Beagle or Strigi are based on.

[1] `http://en.wikipedia.org/wiki/Metcalfe's_law`: Metcalfe's law states that the value of a telecommunications network is proportional to the square of the number of users of the system. First formulated by Robert Metcalfe in regard to Ethernet, Metcalfe's law explains many of the network effects of communication technologies and networks

[2] `http://en.wikipedia.org/wiki/Inotify`: inotify is a Linux kernel subsystem that provides file system event notification.

2. Metadata created manually by the user. In the most simple case this can be a comment to a file or an email, but it could also mean the grouping of several resources under one topic and so on.

3. Metadata that cannot be extracted easily by an indexer and is not generated by the user manually but by an application in the background. This includes for example the URL of a file that is downloaded from the Internet. Once saved on the local hard disk this information is lost. The same goes for the (rather popular) example of email attachments: once an email attachment is saved to the local hard disk its connection to the email and with it the connection to the sender is lost. These are just two examples relating to the source of files. There are many more.

The goal of the Nepomuk-KDE project in its first phase was to create facilities to allow each KDE application to take advantage of this metadata. That means that a KDE application can easily create new metadata, search metadata, search relations between resources based on metadata, and so on.

All code produced within the Nepomuk-KDE project will be released under the GPL[3] or the LGPL[4].

This document presents what has been accomplished in the Nepomuk-KDE project so far. Section 2 gives an introduction to the KDE in general and discusses previous attempts of adding semantics to the desktop. In section 3, the steps that have been undertaken to get the community involved are presented. Section 4 discusses the actual implementation and integration of Nepomuk technologies in the KDE so far. Section 5 gives an outlook on the next phase of the Nepomuk-KDE project in 2007 and 2008. Finally, section 6 makes some concluding remarks on the progress in the first months of the project.

[3] http://www.gnu.org/copyleft/gpl.html
[4] http://www.gnu.org/copyleft/lgpl.html

2 KDE State Of The Art

This section presents an overview of what KDE is, what applications are available, and which semantic and social features can already be used or are being worked on beside the Nepomuk initiative.

The KDE web portal[5] describes KDE as follows.

> KDE is a powerful Free Software graphical desktop environment for Linux and Unix workstations. It combines ease of use, contemporary functionality, and outstanding graphical design with the technological superiority of the Unix operating system.

KDE, however, is much more than that. KDE consist of the following parts:

- A powerful graphical desktop environment.
- A network transparent application development framework based on the QT libraries.
- A vast collection of desktop applications and tools for nearly all purposes, including a complete office suite KOffice and the personal information management suite KDE-Pim including among others an email client, a calender, and an address book.
- A big community of developers and users backed up by many companies from the open-source world

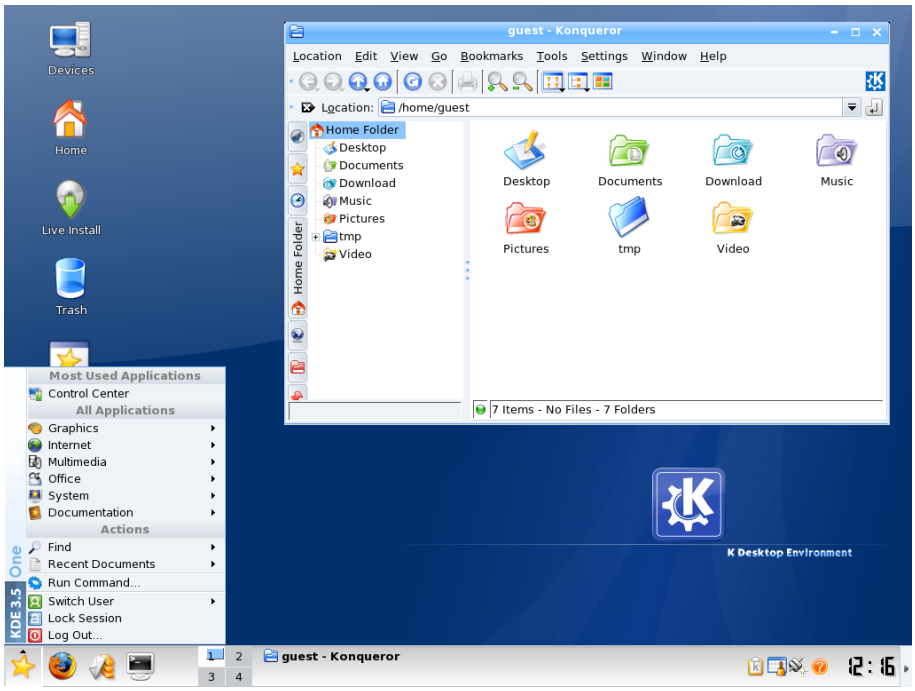

Figure 1: The KDE desktop environment in action

2.1 A Little KDE History

The KDE project was founded in October 1996 by Matthias Ettrich. He proposed a new kind of graphical desktop environment for Linux/Unix systems and shortly many interested people joined the KDE effort.

[5] KDE portal: `http://www.kde.org`

The first version of KDE was released on July 12, 1998 under the GPL[6] and was based on QT[7] 1.4. By that time, QT was not released under an open license compatible to the GPL yet, which was a problem for many and led to the start of the Gnome project[8]. On September 4th, 2000 QT was finally released under the GPL, once and for all removing all criticism in this area. From then on, KDE development went on with a fast pace. KDE 2.0 was released on October 23th, 2000 and marked a very important step for the KDE project. It also marked the official birth of the KOffice office application suite.

By that time KDE already had a vast user base and was supported by many companies. Key players in the Linux segment such as Mandrake Linux (now Mandriva Linux)[9] or SuSE[10] used KDE as their default desktop[11] and contributed to its success.

The release of KDE 3.0 on April 3rd, 2002 was the next big step in KDE development. It came with a new printing system based on CUPS[12] and was already translated into more than 50 languages. With the 3.0 release many parts of KDE had majored like KMail, the email client, or Konqueror, one of the cornerstones of the KDE desktop.

Today, the most recent version of the KDE project being 3.5.5, KDE is probably the most complete and advanced open-source desktop system and development environment available. It ships with a basic desktop and fifteen other packages covering personal information management, administration, network, edutainment, utilities, multimedia, games, artwork, web development and more. KDE's award-winning tools and applications are available in 65 languages.

2.2 KDE - Personal Information Management

This section presents some of the better known KDE applications concerned with personal information management, starting with the most important: the KDE-Pim suite[13]. The goal of KDE-Pim is to provide an application suite to manage personal information. This includes mail, time, people and more. The main result is KDE Kontact.

2.2.1 Kontact

Kontact is the shell that combines all the separate applications of the KDE-Pim suite through the power of KDE's KPart technology (a KPart is a sort of plug-in that brings its own GUI as well as menu and toolbar elements with it):

- KMail: KMail is a very powerful email client. It supports the major email transportation protocols such as POP3, IMAP, and SMTP, reading of HTML mails, has anti-spam functionalities, provides spell-checking and powerful search and filter functions . KMail also has very enhanced privacy

[6] General Public License

[7] QT: `http://www.trolltech.com/products/qt/`. QT is a cross-platform application development framework, widely used for the development of GUI programs. QT is most notably used in KDE, the web browser Opera and Qtopia. It is produced by the Norwegian company Trolltech. QT is written in C++ and comes with bindings for Python, Ruby, PHP, C, Perl, Pascal, C# and Java. It runs on all major platforms, and has extensive internationalization support.

[8] Gnome: `http://www.gnome.org`. Gnome is to date the biggest competitor of KDE on the open desktop market.

[9] Mandriva Linux: `http://www.mandrivalinux.org`

[10] SuSE: `http://www.suse.com`

[11] Mandriva and SuSE still use KDE as their default desktop environment in their latest distribution versions.

[12] CUPS: `http://www.cups.org`

[13] KDE-Pim: `http://pim.kde.org`

and encryption features and is certainly one of the most important and popular KDE applications to date.

- KOrganizer: KOrganizer is the KDE calendar application. It can be used as a personal organizer.
- KAddressbook: KAddressbook is the KDE address management application. It manages all contacts within KDE. It is tightly integrated into KMail and KOrganizer and has advanced features such as import and export to nearly every address book standard, LDAP server support, and a powerful search functionality.
- KNotes: KNotes is the KDE tool that allows writing short notes which can be sticked to the desktop.
- KNode: KNode is the default KDE news reader. It supports multiple news server, reading and composing of news articles, or inline text and image attachments.

These applications form the basis of each KDE desktop's personal information management installation. They are part of the official packages maintained within KDE and have a big fan base.

2.2.2 Kopete

Kopete[14] is the instant messenger (IM) application in KDE. It supports most used IM protocols like AIM, ICQ, MSN, Yahoo, Jabber, IRC, Gadu-Gadu, or Novell GroupWise Messenger through a sound plug-in interface which allows the addition of arbitrary additional communication protocols.

2.2.3 Basket - Advanced Note Management

Basket is a nice KDE application that allows to collect all kind of information in a so called basket. It is possible to store text, pictures, links, files, and much more in a named container. Thus, Basket provides an enhanced notepad.

The simplest usage of Basket is to just drag arbitrary pieces of information to the "basket" and group them under a certain topic.

2.3 KDE Semantic Features

Semantic features in KDE (as of this writing, the current version of the KDE is 3.5.5) are very limited. KDE itself, i.e. the kdelibs, which are described in the kdelibs section below, provides a plug-in system for metadata extraction of type 1. An example is an extractor for the Id3 tags in mp3 files. This metadata, however, is only displayed on user request for a selected file, for example from the file manager Konqueror. This is shown in figure 2.

Other semantic features are restricted to tags or comments in applications like digiKam (see section 2.3.2) or in KDE-Pim (see section 2.3.1). These semantic annotations, however, can only be used in the application itself and cannot be accessed anywhere else.

[14] Kopete: `http://kopete.kde.org`

Figure 2: Konqueror - Displaying metadata

2.3.1 Kontact

The KDE-Pim applications already provide very rough semantic information. Contacts and appointments in KOrganizer can be assigned categories. These categories, however, are again restricted to the applications themselves and even KAdressbook and KOrganizer do not share their categories. Both applications provide very simple search functionalities that allow to find contacts or appointments also by selected categories. But again, it is not possible to automatically find the connection between a contact and an appointment based on a shared category. Figure 4 shows an example of how appointments are ordered into categories in KOrganizer.

KMail, the email client does not yet offer any tagging mechanism such as the categories in KOrganizer or KAdressbook. Thus, emails can not be assigned to arbitrary categories. It only provides simple IMAP-based tagging of emails such as "mark as important" or "mark as todo" which can then be used to find emails within a folder. Figure 3 shows the procedure of marking an email in KMail. Just as with KOrganizer and KAdressbook, these markers are restricted to the application and the mail tree itself.

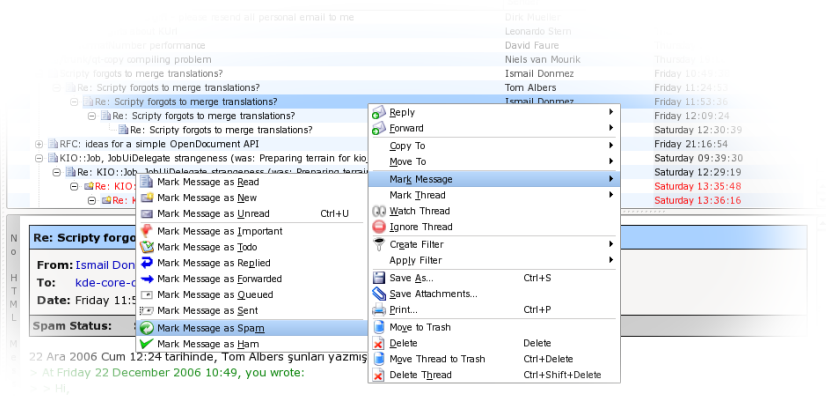

Figure 3: KMail - Marking emails within the application

The categories in KOrganizer and KAdressbook and the markers in KMail can be seen as the only pseudo-semantic features in KDE-Pim. Cross-references between the various KDE-Pim applications and searching through more than one application is not yet supported.

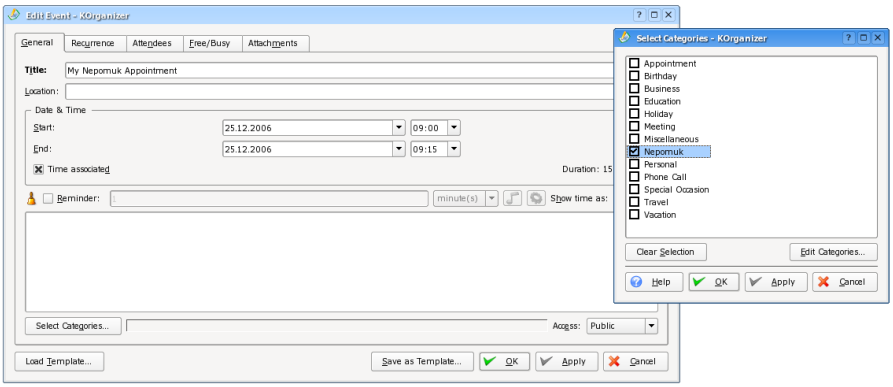

Figure 4: KOrganizer - Selecting categories for an appointment

2.3.2 Other Applications

Some applications designed for and with KDE support simple tagging. One example is the picture management application digiKam. DigiKam sorts pictures into albums which correspond to folders on the local hard disk. Additionally, arbitrary tags can be assigned to pictures. Figure 5 shows how these tags can then be used to filter pictures through virtual albums that represent the tags. The tags in digiKam, however, are restricted to the applications and cannot be used outside of it. Thus, it is not possible to tag different types of data with the same tag.

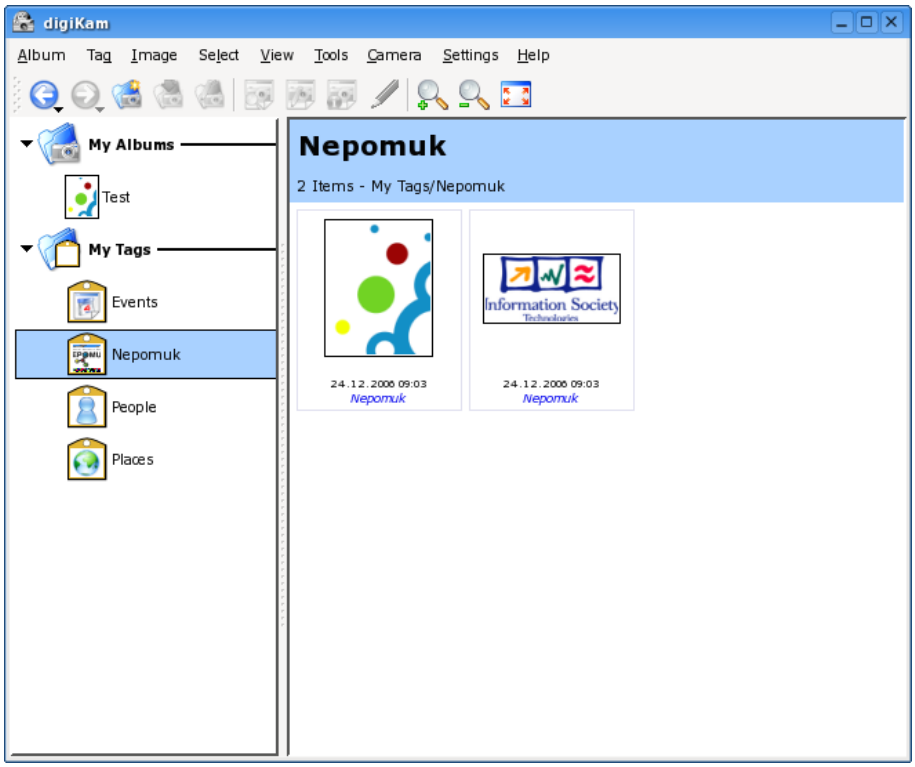

Figure 5: Tagging pictures in digiKam

2.3.3 Kerry - Beagle Desktop Search in KDE

The Kerry project[15]. Kerry aims at integrating desktop search based on Beagle[16] into KDE. It provides a KDE front-end to the Beagle search engine which allows to search for files using their metadata.

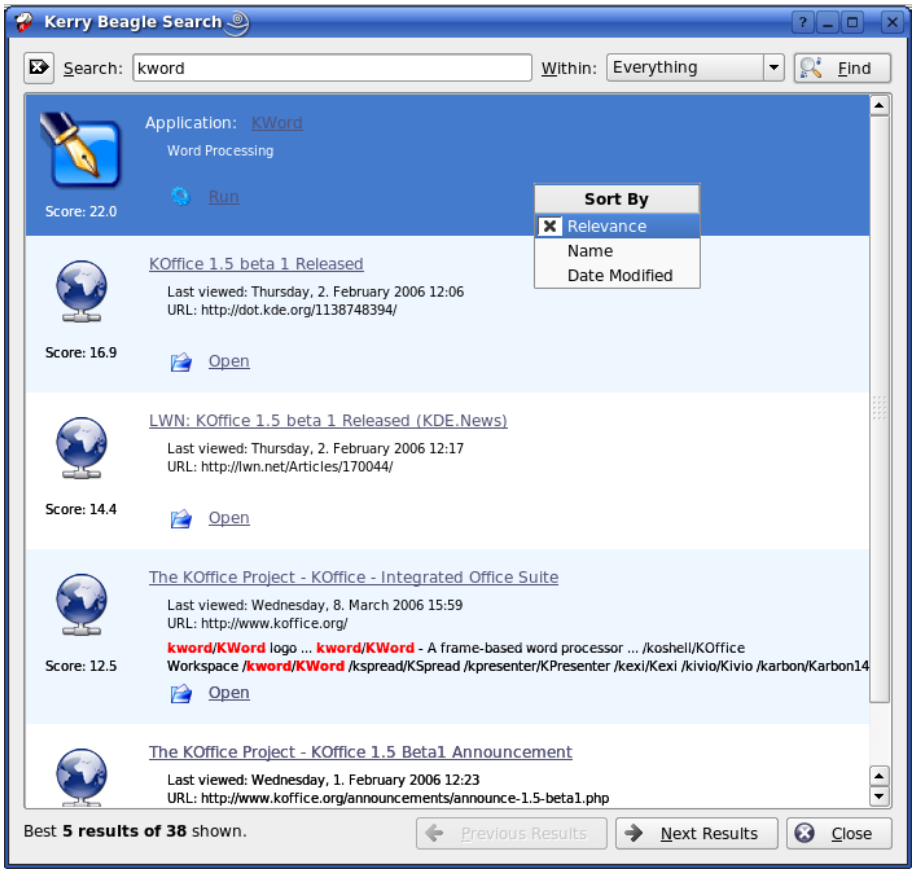

Figure 6: Kerry - KDE Beagle desktop search

The Beagle system extracts metadata of type 1 from all files on the local hard disk and stores them in an index which allows to search for files based on their metadata. Although Beagle also supports some more uncommon types of metadata like subject and sender information from emails stored locally, this information is sorted into arbitrary fields and does not follow a defined style, unlike the approach followed by the use of the Nepomuk Desktop Ontology, (appendix A).

Beagle is implemented in C#. It is considered by the KDE community as very memory hungry and thus, it can only be a temporary solution for the KDE desktop. The KDE developers favour the the Strigi approach, presented in the Strigi section 2.3.4.

2.3.4 Strigi - Fast Desktop Search

Strigi[17] is, like Beagle, a desktop metadata indexer and searching tool. Other than Beagle, however, it is written entirely in C++ and is much faster and less memory hungry. Strigi consists of a daemon which indexes files in selected folders using metadata extractor plug-ins and a KDE interface that allows to

[15] Kerry: `http://kde-apps.org/content/show.php?content=36832`
[16] Beagle: `http://beagle-project.org/`
[17] Strigi: `http://www.vandenoever.info/software/strigi/`

search the indexed information. Strigi is under heavy development and is currently favoured over Beagle by the KDE community. Its stream-based metadata extraction system is very promising and will hopefully be merged into the KDE metadata facilities.

2.3.5 Tenor

Tenor [5] is the name of a contextual linkage framework for KDE designed by Scott Wheeler. It aims at addressing the management of metadata of all types. Tenor is based on the idea of nodes and links between these nodes, which then form the Tenor-graph. Nodes represent the resources while links describe arbitrary relations between the resources. Nodes and links can have properties assigned. The primary function of node properties is to limit the domain of graph traversal while link properties provide a means to specify more fine grained information to the application space.

The idea of Tenor is to have the applications and the user fill the Tenor-graph with as much information as possible and exploit this information later on to improve the way we handle data on the desktop today.

The KDE community had very high hopes for the Tenor framework but development seems either have stalled or it continues in private. Whichever, there was no possibility to merge the existing work in Tenor into the Nepomuk-KDE project and thus, a similar system, namely KMetaData presented in section 4.2, was started from scratch.

2.4 KDE Social Features

In the context of Nepomuk, desktop social aspects mainly relate to P2P desktop communication, trust networks, collaborative recommendation, distributed metadata indexing and social networks analysis. This section gives an overview on existing initiatives covering some of these topics in the realm of KDE, in order to assess which KDE sub-communities may be targeted for an in depth integration of the Nepomuk social components - i.e. mainly Nepomuk WP4000 and WP5000 components - into the KDE desktop.

2.4.1 Decibel

As described on Wikipedia, Decibel[18] is "a new communication framework for KDE4, [aiming at] integrating all communication protocols into the desktop. As of now they have all their contacts in different applications: AOL, MSN, E-mail, Skype, etc. Decibel wants to put all contacts in one place and make it easier for the user to manage and communicate with his/her contacts. For example Alice wants to talk to Bob. Alice requests to start a connection to Bob and a service manager, known as 'Houston' takes in the requests and choose the best way to communicate to Bob (depending on telephone number, e-mail address, etc), and opens up a connection to Bob. This way Alice can contact her friends no matter what protocol they are best contacted with."

It is worth pointing out that Decibel shares some concerns with the Eclipse Communication Framework[19] in providing a high level service for abstracting the protocols used for information exchange across the desktops. In this context as well, Nepomuk can act as a federator between several existing initiatives, so that the Nepomuk APIs can be used indeed above various desktop

[18] Decibel: `http://en.wikipedia.org/wiki/Decibel_(KDE)`, `http://decibel.kde.org/`
[19] Eclipse Communication Framework: `http://www.eclipse.org/ecf/`

environment frameworks, namely KDE, Eclipse and Mozilla.

Decibel is sponsored by the company basysKom[20], a prominent actor in the adoption of the KDE desktop by industrial users.

2.4.2 File Sharing

Within the vast collection of available KDE applications, also exist many file sharing tools. KTorrent[21] for example is a BitTorrent client for KDE and probably the most popular KDE file sharing application to date. KMLDonkey[22], a client for the edonkey2000[23] p2p network or Apollon[24], a client for the giFT[25] daemon which supports file sharing protocols such as OpenFT[26], FastTrack(Kazaa!)[27], Gnutella[28], or OpenNap[29] are all dedicated to mere file sharing over fixed protocols. An integration of Nepomuk technologies with these tools seems mostly inappropriate.

It might, however, be of interest to integrate these file sharing technologies within a new Nepomuk-KDE social component which then supports the usage of arbitrary sharing and communication protocols.

2.4.3 Social Networks Visualiser

SocNetV[30] is "an application for the Linux desktop written in C++/Qt3. Its main objective is to provide a sensible means for Social Networks Analysis and Visualization on the Linux platform. [With SocNetV] you can read and visualize various network file formats and/or visually create and modify a network using your mouse. SocNetV will happily compute network and actor properties, such as distances, centralities, diameter etc."

SocNetV may be used as an entry point for exploring social networks in a KDE environment, in particular in the case of the Mandriva community case study WP11000, which requires an in depth integration with the KDE framework.

2.5 KDE Architecture

Since the goal of the Nepomuk-KDE project is to bring social semantic features into all parts of KDE, i.e. all KDE applications and tools, it is important to build upon and extend the existing architectures in KDE. This section gives a brief overview of the main development artifacts of the KDE framework.

[20] basysKom: `http://www.basyskom.de`. basysKom Managing Director is Eva Brucherseifer, President of the German KDE community (KDE e.V.). basysKom is actively involved in supporting and promoting the KDE development project.
[21] KTorrent: `http://www.ktorrent.org`
[22] KMLDonkey: `http://kmldonkey.org/`
[23] eDonkey2000: `http://en.wikipedia.org/wiki/EDonkey2000`
[24] Apollon: `http://apollon.sourceforge.net`
[25] giFT: `http://gift.sourceforge.net`
[26] OpenFT: `http://en.wikipedia.org/wiki/OpenFT`
[27] FastTrack: `http://en.wikipedia.org/wiki/FastTrack`
[28] Gnutella: `http://www.gnutella.com`
[29] OpenNAP: `http://opennap.sourceforge.net`
[30] SocNetV: `http://kde-apps.org/content/show.php?content=34591`

2.5.1 The kdelibs

The central part of the KDE development framework, and thus, the part everything else is built upon in KDE, are the kdelibs[31]. The kdelibs provide a vast collection of classes to handle all kinds of situations like network transparent file handling, advanced GUI features like editable toolbars, spell checking integration, or even a complete HTML engine. The kdelibs are developed by some of the best people in the open-source community and used by hundreds of applications.

To get an overview of the power of the kdelibs, the reference API[32] or the tutorials on the KDE developer web site[33] are a good starting point.

2.5.2 D-Bus Architecture

Overview

D-Bus[34]is the inter-process communication system used in the upcoming forth version of KDE[35]. D-Bus consists of three components: a library implemented in C that allows applications to connect to each other and exchange messages, a daemon that can route messages between applications and provides the central communication point in most D-Bus setups, and a collection of wrapper libraries (bindings) that allow a simple usage of the D-Bus library from other programming languages and environments. The most elaborate bindings to date are the QT4 bindings which are used in KDE 4.

D-Bus was especially designed for communication between desktop applications in the same session and the operating system (HAL[36] is probably the best known example for communication with the operating system). D-Bus is developed as a pseudo-standard at freedesktop.org[37] and is quickly becoming the default communication system on the Linux desktop.

Architecture

In the D-Bus system, each application gets assigned a unique identifier (for example ":1.10") and can optionally register one or more names. The Nepomuk-KDE Middleware service registry for example registers as org.semanticdesktop.nepomuk.ServiceRegistry. An application may then export an arbitrary number of objects, each providing an arbitrary number of interfaces. Each interface defines methods and signals. Thus, a method in the D-Bus system is identified by four tokens: the D-Bus service id (i.e. the unique identifier), the object path, the interface name, and the method name[38]. This is best made clear with an example.

Let some application define a method `helloWorld` which shows a window stating the words "Hello World!". The application could export this method via D-Bus. It could register as org.example.HelloWorld and export the interface org.example.helloworld.test on the /Test object:

```
org.example.HelloWorld
   - /Test
      - org.example.helloworld.Test
         - helloWorld()
```

[31] kdelibs API: `http://api.kde.org/cvs-api/kdelibs-apidocs/`
[32] kdelibs API: `http://api.kde.org/`
[33] KDE devel: `http://developernew.kde.org/`
[34] D-Bus: `http://dbus.freedesktop.org`
[35] D-Bus replaces DCOP as the default desktop communication protocol in KDE 4
[36] HAL: `http://hal.freedesktop.org/`
[37] freedesktop.org (`http://www.freedesktop.org`) is a working group focusing on interoperability and shared technology for X Window System desktops, in particular GNOME and KDE, although developers working on any Linux/UNIX GUI technology are welcome to participate.
[38] It is common practice to choose D-Bus identifiers and interface names according to the web URL of the owner of the application. Thus, all KDE applications are named *org.kde.appname*.

3 KDE Community Involvement

One of the main tasks of the Nepomuk-KDE project beside the implementation of semantic KDE features is the involvement of the KDE community. It is essential to the success of the Nepomuk project that the developed technologies are picked up and used in important projects such as KDE. Only then will the Nepomuk project have the chance to actually establish standards throughout the desktop world.

This section gives an overview of the KDE community activities in the Nepomuk-KDE project.

3.1 Nepomuk-KDE Presentation At aKademy 2006

The Nepomuk-KDE project was presented at the annual KDE developer conference aKademy 2006[39] in Dublin. This was also the first mentioning of the project to the KDE community ever. The response was positive, clearly the vision of a "social semantic KDE desktop" is very important to the KDE community. Especially the KDE-Pim team was very interested in what will be achieved in the near future and was eager to help with the integration into their new PIM data storage Akonadi[40]. It was already noted on the conference that semantic technologies developed withing Nepomuk-KDE are candidates for inclusion in the kdelibs. Section 3.3 presents another success story.

During the conference, there has been a lot of discussion, especially with Jos van den Oeven, the author of the Strigi desktop search and indexing tool, about what Nepomuk-KDE should and will achieve. The main focus of interest in the KDE community seems to be on file tagging and desktop search at the moment. Projects like Beagle or the Google desktop search which are the current state of the art mainly trigger this interest. Sections E.3 and 4.4 show the progress that has been already made in this area within the Nepomuk-KDE project.

3.2 The Nepomuk-KDE Web Portal

As part of the Nepomuk-KDE project, in order to present the goals and progress of the project, and also to raise interest across the community, a wiki has been created[41], as illustrated by figure 7. This wiki is intended to be a portal for developers and users of the Nepomuk-KDE project. It provides information about all the Nepomuk-KDE sub-projects, information for developers, and the project in general. Contrary to the internal Nepomuk wiki, it is open to the public, inviting the community to take part in the development.

In addition, a public Nepomuk-KDE mailing-list[42] has been created and has already received attention from a lot of interested people (33 subscribers to date beside Nepomuk internal participants). Most of the current development discussions, however, take place in the #nepomuk-kde IRC channel and in the KDE mailing-lists like kde-core-devel[43].

[39] aKademy 2006: http://conference2006.kde.org/

[40] Akonadi: `http://pim.kde.org/akonadi/`. The project aims at "designing an extensible cross-desktop storage service for PIM data and metadata providing concurrent read, write, and query access".

[41] Nepomuk-KDE wiki URL: `http://nepomuk-kde.semanticdesktop.org`

[42] Nepomuk-KDE mailing-list: `https://nepomuk.semanticdesktop.org/wws/info/nepomuk-kde`

[43] http://lists.kde.org

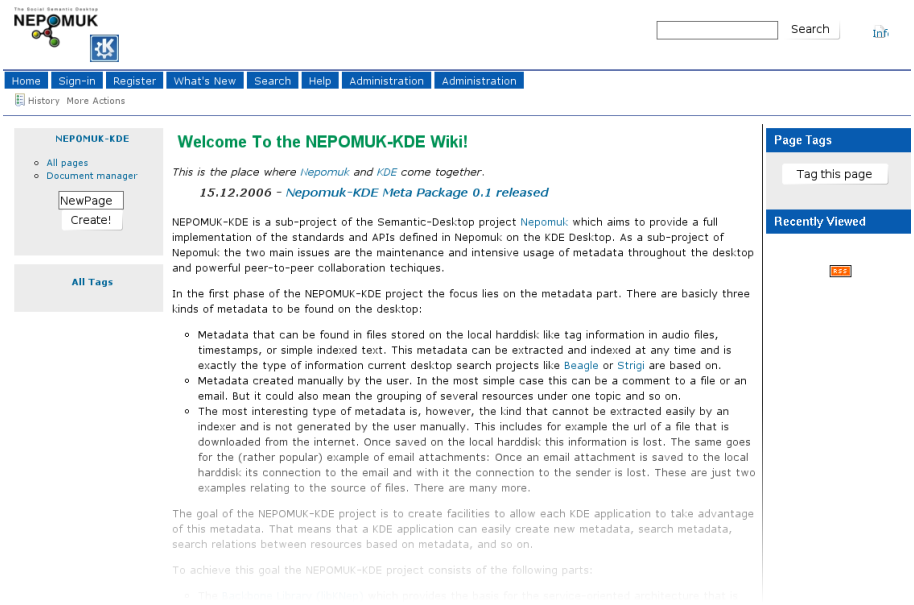

Figure 7: The Nepomuk-KDE wiki home page

3.3 Integration

Since Nepomuk-KDE is intended to provide semantic features for KDE, it is developed in the KDE subversion repository (Appendix B.1.2 describes details on how to use the repository), thus, already taking advantage of the KDE development process which for example includes translations of all GUI components. This early integration into the working branch of the KDE development favours the adoption of the Nepomuk-KDE achievements by the upcoming mainstream KDE releases.

The goals and work already done in Nepomuk-KDE have attracted some of the important characters in the KDE community which resulted in a move of parts of Nepomuk-KDE (the most important part here is KMetaData, which is presented in section 4.2.) into the main KDE libraries. As of this writing the move is planned but not performed yet. Once done, however, Nepomuk-KDE, and with it Nepomuk technologies will be available to all KDE developers: a big step towards a semantic KDE based on Nepomuk has been taken.

4 Nepomuk-KDE Prototype

Besides sparking off Nepomuk-KDE community uptake, the Nepomuk-KDE project is about implementing the standards defined in the Nepomuk project on the KDE platform. In this section, the state of the development as of this writing is presented.

4.1 The Nepomuk-KDE Middleware

The Nepomuk middleware [3] is the central communication hub in the Nepomuk system. The middleware communication is designed to be platform and technology independent. This is realized by the introduction of a federation of service registries, each of which implement a certain type of communication. Nepomuk services are developed for a specific type of communication and, thus, for a specific middleware implementation. Figure 8 shows an overview of the middleware architecture.

While the core Nepomuk middleware task force is implementing a reference implementation of the middleware based on Java/OSGi [44] the Nepomuk-KDE project realizes the same for the KDE/QT platform with D-Bus communication.

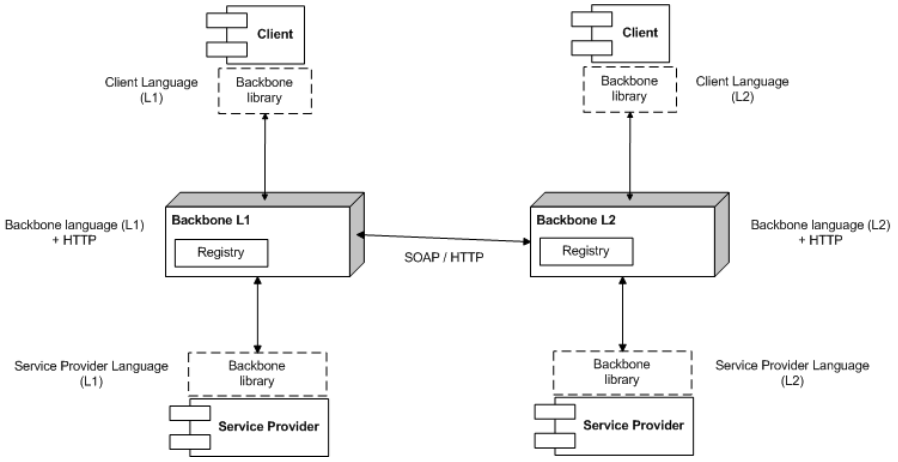

Figure 8: Nepomuk middleware components - KNep Service Registry implements the middleware registry and KNep Client implements the middleware library

Both implementations (Java/OSGI and KDE/D-Bus) were developed in parallel during the definition phase of the middleware API. This allowed the middleware task force to have direct real-life experiences and feedback. Problems could be detected and solved early and the result was not only an abstract standard but two implementations. The second one, realized in the KNep package is presented in this section.

KNep consists of two parts:

- The KNep Service Registry is a KDE daemon module that implements the service registry as defined in the Nepomuk middleware. It provides its functionality like service registration and discovery via a D-Bus interface.
- The KNepClient library implements the service client library defined in the Nepomuk middleware with a QT/KDE API. It provides a wrapper around the D-Bus communication used between the services and the service registry for simple service discovery and publishing. Appendix C presents details of the API.

[44] OSGi: `http://www.osgi.org`

4.2 KMetaData - Embedding Nepomuk Metadata into KDE

One of the first usages of the KNepClient library is the KMetaData library. It is based on and provides a convenience wrapper around the Nepomuk Desktop Ontology to handle metadata in KDE applications.

As already pointed out in the introduction three types of metadata can be identified:

1. Metadata that is stored with the data itself and is available at all times. This includes id3 tags, the number of pages in a PDF document, or even the size of a file or the subject of an email.
2. Metadata that is created by the user manually like annotations or tags that are assigned to files, emails, or whatever resources.
3. Metadata that can be gathered automatically by applications such as the source of a downloaded file or the email an attachment was saved from or the original when copying a file locally.

Type 1 is already handled in many implementations. KDE itself includes the KMetaFileInfo framework that allow extracting this kind of meta information from files.

KMetaData is intended for metadata of type 2 and 3. It provides an easy way to create and read metadata for arbitrary resources (this includes for example files or emails, but also contacts or maybe even a paragraph in a PDF file). The simplest type of metadata that can be handled with KMetaData is a comment. It is a simple string associated with a resource (a file for example). This comment is created by the user using an application that is based on KMetaData. KMetaData's core is designed to allow arbitrary types of meta data[45], i.e. any resource can be related with any other resource or value by simply naming the relation and providing the value. The power of KMetaData, however, lies in that it provides a C++ class for each type of resource. Each of these classes provide convenience methods to allow a simple handling of the metadata.

The types of resources and their properties are defined in the the Nepomuk Desktop Ontology which is discussed in appendix A. KMetaData's build system includes a code generator which creates one C++ class for each RDF class/type defined in the ontology. An example or such a class can be seen in appendix E.1.

KMetaData is resource based. Thus, working with KMetaData is always done with instances representing a certain resource. This resource has a list of properties. Properties are named and have a certain type. The type can either be another resource (compare a file that was an attachment from an email) or a literal (this means for example a string, or an integer; the comment mentioned earlier would be a string literal). Each property can either have a cardinality of 1 (again a file can only be saved from one email) or greater than 1 (i.e. infinite, like one file can have arbitrary many associated comments). This restriction naturally evolves from the usage of lists vs. single values. Appendix D shows how KMetaData handles literals and cardinalities greater than 1.

4.3 The Core Services

The Nepomuk-KDE middleware ships with a set of core services which are necessary to use the Nepomuk-KDE system at all and thus have to be installed as the core Nepomuk-KDE runtime.

[45] In fact, KMetaData roughly implements the NRL/RDFS standard as a C++ class library but is optimized for the Nepomuk Desktop Ontology.

4.3.1 RDF Storage

The RDF storage service implements the RDFRepository interface as defined in Nepomuk by work package 2 [2]. This service is based on the Soprano/QRDF package which has been developed within Nepomuk-KDE. It has been developed in close cooperation with the Nepomuk RDF-API task force. This allowed to test and optimize it in a very early stage.

The Nepomuk-KDE RDF storage service provides a D-Bus interface which allows to add, remove, and list RDF statements and also query RDF models using the SPARQL[46] query language.

4.3.2 Resource Identification

The resource identification service is implemented in Nepomuk-KDE as a very simple resource URI matcher which makes sure that local file URIs are not stored multiple times (`/home/foo/bar` = `file:///home/foo/bar`).

Section 5.2.1 states how the resource id service should be extended in the future.

4.4 Local Search Service

Metadata created via KMetaData alone does not help the user much yet. It has to be at least searchable. Thus, a plug-in for the Strigi desktop search engine[47] has been written that allows Strigi to index Nepomuk metadata. This has the effect that it is possible to find desktop resources[48] that have been annotated or tagged via KMetaData with a simple Strigi desktop search.

Appendix B.3 shows how this integration can be tested.

4.5 Annotation and Tagging

To demonstrate the power of the current Nepomuk-KDE libraries (i.e. KNep and KMetaData), simple client applications have been implemented that allow to annotate and tag files from within the Konqueror file browser or from the command line as can be seen in figure 9. The data, i.e. the annotations and the tags are stored within the Nepomuk-KDE RDF storage service. Figures 9 and 10 show example views of the applications graphical user interfaces.

The simple annotator allows to add comments to arbitrary files. The URI of the file resource has to be provided on the command line (this is exactly what Konqueror does when calling the annotator from the context menu: it adds the URL of the file to annotate to the call). The annotation is then associated with the file in question via KMetaData and thus, synchronized automatically. Section 4.4 shows how this new information can be used to simply relocate the file in question.

The simple tagger on the other hand allows the creation of new tags (simple topics which are used to group resources) and assign these tags to files (in general the simple tagger can be used to assign tags to arbitrary resources

[46] SPARQL: `http://www.w3.org/TR/rdf-sparql-query/`

[47] Strigi has been selected over Beagle because it is much faster, has a much lower memory footprint, and is written in C++. Apart from that Strigi is likely to be integrated into KDE soon as the default desktop search engine.

[48] In its current implementation state, Strigi mainly indexes files but a plug-in to handle Akonadi PIM resources is in development. One of the next steps will be to include Nepomuk metadata for PIM resources in the Strigi index.

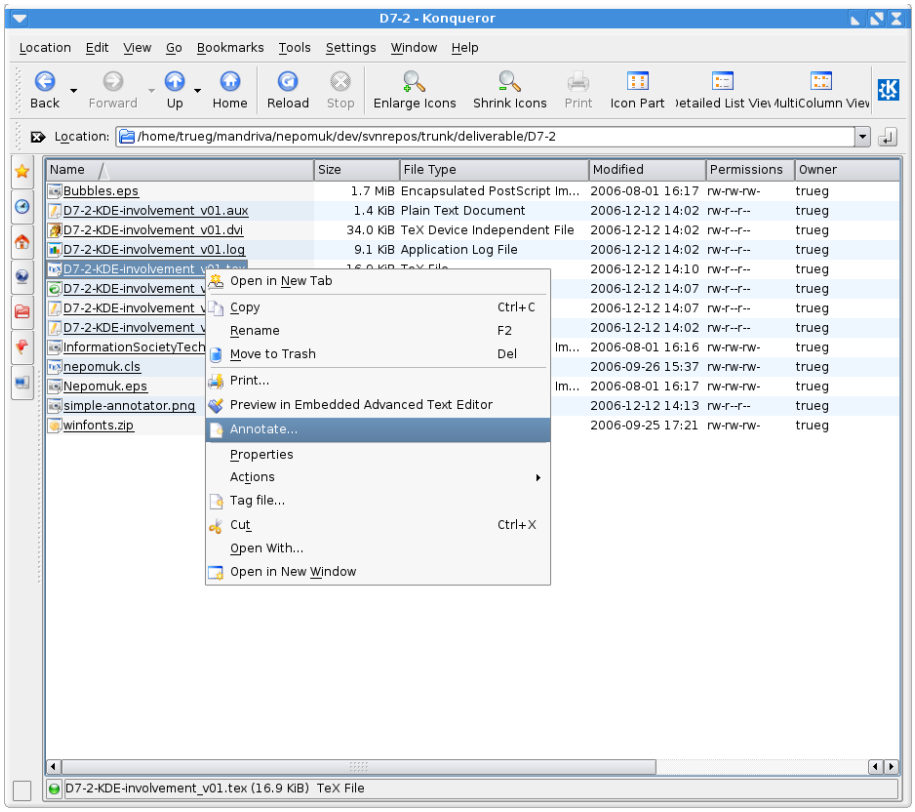

Figure 9: Simple-Annotator Konqueror 4 Integration

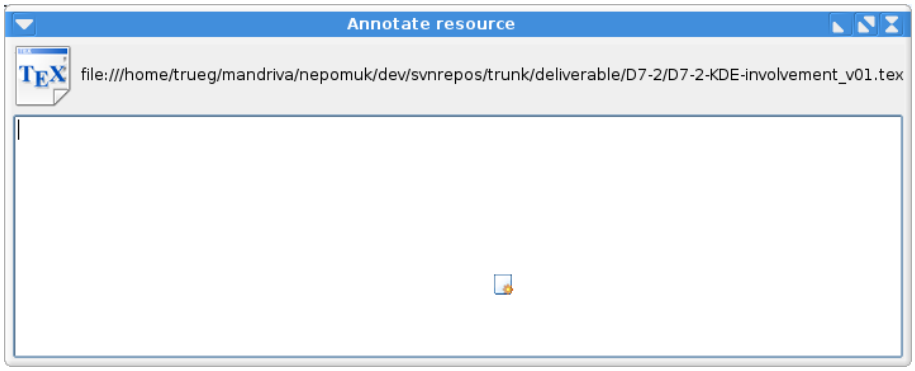

Figure 10: Annotating a resource with the Nepomuk-KDE simple annotator

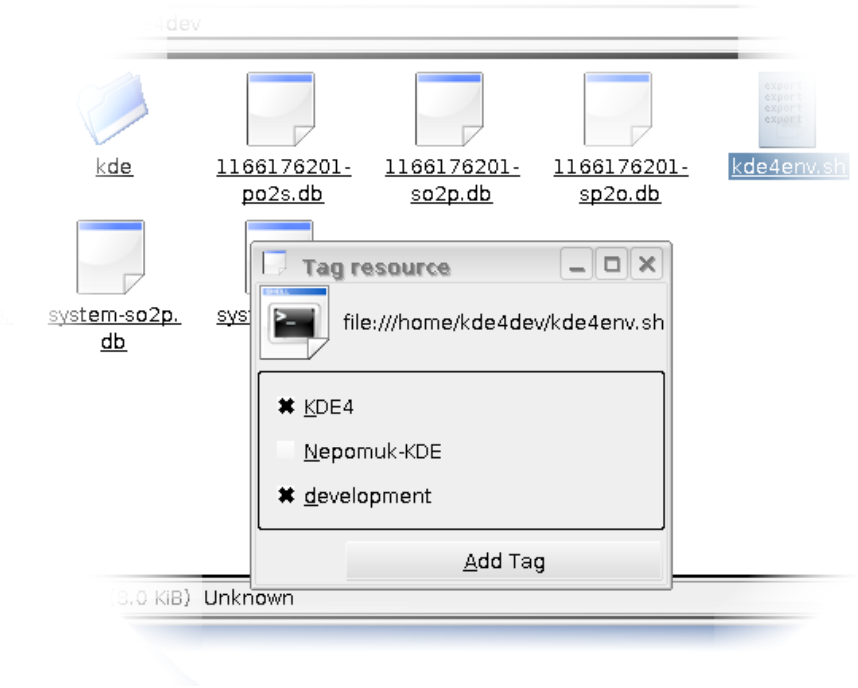

Figure 11: Tagging a file with the Nepomuk-KDE Simple Tagger

but since it is mainly called from the Konqueror which only handles files the discussion is restricted to files also).

4.6 Miscellaneous Tool Support

Before the start of the Nepomuk-KDE project Nepomuk tool development under KDE had already started. This section presents these tools that are not strictly part of Nepomuk-KDE.

4.6.1 KGense

KGense has been developed by Edge-IT to serve as a generic search front-end for various search engines such as Beagle, Strigi, or even Google web search and display all results in one convenient user interface.

4.6.2 KRDFExplorer

The KRDFExplorer is not really part of the Nepomuk-KDE prototype. Its development started before the Nepomuk-KDE project and it was not ported yet.

KRDFExplorer is intended as an all-purpose RDF data querying tool for the Nepomuk system as can be seen in figure 12. It provides advanced querying capabilities and a convenience presentation of the data in the store.

As mentioned KRDFExplorer was not ported to Nepomuk-KDE yet. Thus, it is still based on the Sesame RDF storage solution which is accessed via HTTP. Porting the application to using KNepClient and the Nepomuk-KDE RDF storage service instead should be an easy task.

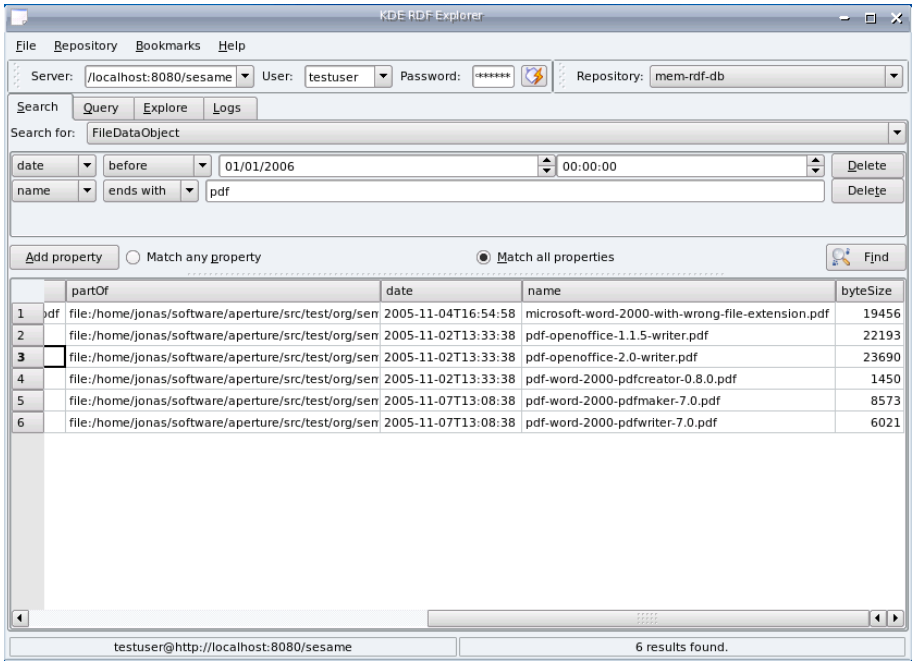

Figure 12: RDF browsing with KRDFExplorer

5 NEPOMUK-KDE Next Steps

5.1 KDE Development

Roughly since the release of QT 4 the development on KDE 4 has started. KDE 4 is intended as the next big step in Linux/Unix desktop systems. The step from KDE 3 to KDE 4 is a big one which will give birth to many new technologies within KDE and allows to change a lot about the desktop in general. Thus, the interest in semantic KDE features is big in the KDE community which gives the Nepomuk-KDE project the chance to influence the direction of the KDE desktop a lot. To accomplish this, however, it is important to closely work with the KDE community and weight their opinion as important.

As already mentioned before the Nepomuk-KDE core parts, i.e. the middleware and KMetaData are scheduled to be included into the kdelibs. Once this move is complete, it is likely that the interest in the project will increase a lot and people will start including semantic features into their applications[49].

5.1.1 Akonadi

One major step in the KDE 4 development cycle is the Akonadi PIM data storage system which will take care of storing all PIM data in the KDE. This allows for an enhanced interoperability between PIM applications. It is important to closely work with the Akonadi team to early integrate semantic features into the Akonadi system or at least implement fast and reliable meta data extractors supporting data stored in Akonadi.

5.2 Nepomuk-KDE Development

After the move of the core Nepomuk-KDE components (middleware and KMetaData) into the kdelibs, it will be important to establish them and make sure people use them. Thus, the example tagging and annotation applications developed within Nepomuk-KDE (see section 4.5) need to be replaced by real applications and plug-ins that provide high usability for KDE.

KMail and the KDE-Pim components in general are the most important applications to be extended with semantic features. The first step will be to allow tagging of emails, contacts, and calendar entries. Having generic tagging support throughout the KDE desktop will already be a big leap forward.

Metadata that can be extracted from files such as the artist and title fields in music files or the author of a PDF file is already handled well but there is no standard that defines which meta data is defined for which type of data (except the scattered specifications of the file formats). Thus, one goal is to extend the Nepomuk Desktop Ontology in cooperation with the Nepomuk Ontology task force that it may serve as a basis for all metadata on the desktop. Then metadata of all types can be combined under one common API (preferably KMetaData) which allows application developer a very simple access to it and also provides the basis for very powerful search and inference algorithms.

To reach this goal, KMetaData has to be extended to handle read-only meta data and, more importantly, data from different sources. The kdelibs already include a plug-in system to extract metadata from files, namely KFileMetaInfo. This system has to be extended to support the Desktop Ontology and provide its information to KMetaData. There has already been discussion with the author of the Strigi indexing tool to merge the Strigi metadata extraction and

[49] In this first phase semantic features will mainly consist of generic tagging and annotation support.

caching with KDE's metadata system.

Another important issue once KDE has been extended by semantic features as described above is the "intelligent" usage of this new data. Desktop search is the first step but is very dumb since it indexes only to a depth of one. Semantic information, however, connects resources much more complicated then just 1-to-1. Resources can be related less closely but related nonetheless. For example a document sent in an email by a person tagged to be in the NEPOMUK group could also be relevant when searching for the term "NEPOMUK". Standard desktop search, however, will not pick it up since it is not directly related. The graph has to be traversed from the tag to the person to the email to the document to find the connection. Once links like this can be exploited in the desktop search or in applications KDE will be already very semantic.

The following list names the goals more specificly:

- Complete the move of the core Nepomuk-KDE components into the kdelibs.
- Implement tagging support in the KDE-Pim components (KMail, KOrganizer, KAddressBook, etc.).
- Implement easy and improved file tagging and annotation on the KDE Desktop.
- In cooperation with the Nepomuk Ontology task force extend the Nepomuk Desktop Ontology to handle all necessary types of metadata.
- Extend KMetaData to handle meta data from multiple sources.
- Implement "intelligent" metadata querying based on the Nepomuk storage services. A first step will be traversing the metadata graph in more depth.

The following section list all Nepomuk components and state the plans regarding their implementation and integration in Nepomuk-KDE.

5.2.1 Nepomuk Components

Middleware and Core Services

The currently existing implementation of the Nepomuk-KDE middleware and the core services were created during the development of the component specifications themselves. Thus, they do not match completely yet the specifications defined in the work packages WP2000 [2] and WP6000 [3]. One of the next steps is thus, to adapt the core services in Nepomuk-KDE to the exact specification of the WP2000 components, i.e. the PIMOService and the RDFRepository.

This will finally allow Nepomuk-KDE components to properly interact with the Java reference implementation developed in work package WP6000 and give birth to the Nepomuk middleware federation.

A full implementation of the results of work package WP2000 includes an adaption of the resource identification scheme for the KDE desktop. Since this is without a doubt one of the most new and advanced part of the Nepomuk middleware, an integration into the KDE will mean intense community work.

DataWrapper

A further implementation will mean realizing Strigi as a Nepomuk DataWrapper [2] which stores the indexed data in the Nepomuk RDFRepository. This will serve as a first "real-life" test platform for the performance of the future Nepomuk system and hopefully show the way into the future. Since the KDE

	community and even more so the Strigi development team have their own plans regarding desktop search, the future of the Nepomuk-KDE project will involve a lot of communication and negotiating between the Nepomuk standards which are intended to be realized in KDE and the current state of KDE. The Nepomuk project is a very academic project which means clean and mostly new solutions that have the luxury of starting from a fresh state. The Nepomuk-KDE project, however, needs to adapt existing KDE technologies and integrate them with Nepomuk ideas. The extent to which the Nepomuk standards can actually be integrated indeed into existing KDE components will be clarified in the second part of the project.
Local Search Service	The Nepomuk-KDE local search service will be implemented on top of Strigi, as introduced in the state of the art paragraph. Strigi will be extended to implement the Nepomuk local search component.
Rich Wiki	One of the more straight-forward metadata creation tools is the rich semantic wiki component as specified within the workpackage WP1000[1]. The WikiModel component designed by WP1000 is intended to be implemented within Nepomuk-KDE in form of a semantic wiki editor based on Kate[50]. In a first step, the editor will merely provide syntax highlighting and pass the created text to a semantic wiki parser complying with the specifications of the WikiModel component described in Nepomuk deliverable D1.1 [1]. In a later phase, the wiki editor is intended to provide more elaborate features, fulfilling the functional requirements expressed in the deliverable D1.1 [1].
IMapping Framework	Once specified by WP1000, the Nepomuk IMapping framework will be implemented and developed on top of the powerful graphical framework Qt4.
Personal Task Manager	Although there are no specific plans yet, it might be possible to build upon existing applications like KOrganizer or TaskJuggler[51] and extend them with Nepomuk functionalities as specified by the workpackage WP3000. This would make use of an existing user base and bring the Nepomuk task model approach directly to the user.
User Context Service	Creating a user context framework within KDE is a big task. The Nepomuk interfaces and services defined in work package WP2000 will have to be combined with either a plug-in system or a library that can be used by all KDE applications. It is without a doubt desirable to integrate as much of the user context service as possible directly into the kdelibs to automate a big part. It is, however, still unclear how this can be accomplished.
Distributed Index	Since a distributed index is quite independant from the graphical user interface and can probably be developed encapsulated within the Nepomuk-KDE components of KDE a realization will be a fairly straight-forward copy of the reference implementation from work package 4 [4].
Desktop Ranker	The simple desktop ranking functionality will be realized as part of the Strigi desktop search.
Advanced Recommendation Algorithms	The roadmap for this components will be defined once the first deliverable of WP5000 has been issued.
Detection and Labeling of Communities	The roadmap for this component will be defined once the first deliverable of WP5 has been issued.
Community Structure Analysis	The roadmap will be defined once the first deliverable of WP5000 has been issued.
Trust, Reputation and Spam Prevention	The roadmap will be defined once the first deliverable of WP5000 has been issued.

[50] Kate: `http://www.kate-editor.org/`. Kate project consists of KatePart, an advanced editor component which is used in numerous KDE applications, and Kate, a MDI text editor application. KatePart is part of kdelibs.

[51] TaskJuggler: `http://taskjuggler.org`

5.3 Integration of Nepomuk-KDE into Mandriva Linux

Mandriva Linux is a Linux distribution created by Mandriva, one of the Linux distribution leaders in the world, whose user base is ranked second as of December 2006 by the popular Linux web site DistroWatch.com[52]. Mandriva is directly involved into the Nepomuk project through its subsidiary Edge-IT, which is spearheading the Nepomuk dissemination workpackage (WP7000) and the Nepomuk Mandriva help-desk case study (WP11000).

The release cycle of the Mandriva Linux distribution is annual. However, intermediate releases are issued several times a year. The next release of the mainstream Mandriva community version is scheduled for the first semester of 2007. This upcoming version will include the current snapshot of the KDE4 environment, which will comprise some of the Nepomuk-KDE libraries, properly packaged, integrated and tested into the Mandriva Linux operating system. Since KDE is the default graphical environment of Mandriva Linux, it is expected that this release will set off a vibrant uptake across the community of users and developers.

5.4 Community Involvement

Once the Nepomuk desktop ontology has been established in KDE, it will be proposed on freedesktop.org as an open standard. That would be the next big step in unifying different desktop environments. Metadata could be handled the same way in KDE as in Gnome or even on Windows systems.

For the first quarter of 2007 a Nepomuk-internal workshop is planned presenting Nepomuk-KDE to the Nepomuk project partners. In January 2007 the Nepomuk-KDE project will be presented at the Solutions Linux fair in Paris.

[52] DistroWatch: `http://www.distrowatch.com`

6 Conclusion

The first months of the Nepomuk-KDE project can be regarded as very successful. A working implementation of the Nepomuk middleware including the core services has been realized and made public to the KDE community. First simple client applications have been implemented and used to prove the usefulness of the concept.

The high point of the project so far is, however, the upcoming integration of the Nepomuk-KDE middleware and KMetaData into the kdelibs. This will drastically improve the awareness of Nepomuk-KDE and Nepomuk in general. Hopefully it will also bring new developers to the effort and help speeding up the realization of a social semantic KDE desktop.

Although negotiations between the KDE community and the Nepomuk project are not always easy due to the differences in goals and work style, sound foundations have been laid down for a very fruitful collaboration between KDE and Nepomuk teams. KDE 4 is scheduled for mid 2007. It will include a first version of the Nepomuk-KDE achievements. The upcoming releases of KDE 4 are expected to realize a complete KDE implementation of the Nepomuk Social Semantic Desktop.

The importance of efforts like Nepomuk-KDE should be obvious. The implementation for a popular desktop like KDE will provide a wide acceptance of the Nepomuk project in general, including users and developers.

A Nepomuk Desktop Ontology

The metadata used throughout the Nepomuk-KDE project is based on the Nepomuk Desktop Ontology. As of this writing it has not yet been defined. It will however be based on NRL, the Nepomuk Representation Language which is an XML language building on top of RDF/S.

Currently the KMetaData library uses a dummy place-holder ontology which only defines tags and annotations (see listing 1).

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:nrl="http://semanticdesktop.org/ontology/nrl-20061204#"
  xmlns:nkde="http://nepomuk-kde.semanticdesktop.org/ontology/nkde-0.1#">
  <rdf:Description rdf:about="http://nepomuk-kde.semanticdesktop.org/ontology/
      nkde-0.1#Thing">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:comment>Thing is the base class of nearly everything. It represents
        all user-accessible resources. Thus, all Things can be annotated and
        tagged.</rdfs:comment>
  </rdf:Description>
  <rdf:Description rdf:about="http://nepomuk-kde.semanticdesktop.org/ontology/
      nkde-0.1#Tag">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:comment>A Tag can be assigned to any Thing. This allows simple
        grouping of resources.</rdfs:comment>
  </rdf:Description>
  <rdf:Description rdf:about="http://nepomuk-kde.semanticdesktop.org/ontology/
      nkde-0.1#hasTag">
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#
        Property"/>
    <rdfs:range rdf:resource="http://nepomuk-kde.semanticdesktop.org/ontology/
        nkde-0.1#Tag"/>
    <rdfs:domain rdf:resource="http://nepomuk-kde.semanticdesktop.org/ontology
        /nkde-0.1#Thing" />
  </rdf:Description>
  <rdf:Description rdf:about="http://nepomuk-kde.semanticdesktop.org/ontology/
      nkde-0.1#hasAnnotation">
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#
        Property"/>
    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
    <nrl:maxCardinality>1</nrl:maxCardinality>
    <rdfs:domain rdf:resource="http://nepomuk-kde.semanticdesktop.org/ontology
        /nkde-0.1#Thing" />
    <rdfs:comment>Everything can be annotated with a simple string.</
        rdfs:comment>
  </rdf:Description>
  <rdf:Description rdf:about="http://nepomuk-kde.semanticdesktop.org/ontology/
      nkde-0.1#hasName">
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#
        Property"/>
    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
    <nrl:cardinality>1</nrl:cardinality>
    <rdfs:domain rdf:resource="http://nepomuk-kde.semanticdesktop.org/ontology
        /nkde-0.1#Tag" />
    <rdfs:comment>A Tag is basicly a string value. The name of a Tag is all
        there is.</rdfs:comment>
  </rdf:Description>
  <rdf:Description rdf:about="http://nepomuk-kde.semanticdesktop.org/ontology/
      nkde-0.1#EMail">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="http://nepomuk-kde.semanticdesktop.org/
        ontology/nkde-0.1#Thing" />
  </rdf:Description>
  <rdf:Description rdf:about="http://nepomuk-kde.semanticdesktop.org/ontology/
      nkde-0.1#File">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="http://nepomuk-kde.semanticdesktop.org/
        ontology/nkde-0.1#Thing" />
  </rdf:Description>
  <rdf:Description rdf:about="http://nepomuk-kde.semanticdesktop.org/ontology/
```

```
      nkde-0.1#hasLocation">
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#
        Property"/>
    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
    <nrl:cardinality>1</nrl:cardinality>
    <rdfs:domain rdf:resource="http://nepomuk-kde.semanticdesktop.org/ontology
        /nkde-0.1#File" />
    <rdfs:comment>The location of a File is it's URL or path.</rdfs:comment>
  </rdf:Description>
</rdf:RDF>
```

Listing 1: The place-holder ontology used in KMetaData until the Nepomuk Desktop Ontology has been drafted

B Testing the Nepomuk-KDE components

This section provides a step-by-step guide on how to setup a KDE4 session and the Nepomuk-KDE tools in order to test the current state of development.

B.1 Preparations

Since the Nepomuk-KDE components are developed for the KDE4 platform the first step is to setup a complete running KDE4 session.

B.1.1 Running a KDE4 session

As of this writing, KDE4 is still in an early stage of development and, thus, unstable. It is recommended to create an entirely new user to run the KDE4 session. Let kde4dev be a newly created user account that will be used for all KDE4 related steps.

Environment Setup Login as user kde4dev and setup some environment variables (the best way is to put them in a script that is run at login time, for example `~/.bashrc`)

```
export QTDIR=$HOME/qt4
export PATH=$QTDIR/bin:$PATH
export LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH
export PKG_CONFIG_PATH=$QTDIR/lib:$PKG_CONFIG_PATH

export KDEDIR=$HOME/kde4/inst
export PATH=$KDEDIR/bin:$PATH
export LD_LIBRARY_PATH=$KDEDIR/lib:$LD_LIBRARY_PATH
export QT_PLUGIN_PATH=$KDEDIR/lib/kde4/plugins
export KDEDIRS=$KDEDIR

unset XDG_DATA_DIRS
# set it, to avoid seeing kde3 files from /usr
export XDG_DATA_DIRS=$KDEDIR/share
unset XDG_CONFIG_DIRS

export KDEHOME=$HOME/.kde4
export KDETMP=/tmp/$USER-kde4
mkdir -p $KDETMP
export KDEVARTMP=/var/tmp/$USER-kde4

function cmakekde { cmake -DCMAKE_INSTALL_PREFIX=$HOME/kde4/inst \
-DCMAKE_BUILD_TYPE=debugfull $@
}

export PS1="kde4 - $PS1"
```

Build QT 4.2 Retrieve the current QT4 source code from the KDE subversion trunk and build it (no installation necessary).

```
svn co svn://anonsvn.kde.org/home/kde/trunk/qt-copy ~/qt4
cd ~/qt4
./apply_patches
./configure -qt-gif -no-exceptions -debug -qdbus -opengl -fast -prefix \$QTDIR
make sub-src sub-tools
```

Build Kdelibs 4 Retrieve the current kdelibs 4 source code from the KDE subversion trunk and install it.

```
svn co svn://anonsvn.kde.org/home/kde/trunk/KDE/kdelibs ~/kde4/src/kdelibs
mkdir -p ~/kde4/build/kdelibs
cd ~/kde4/build/kdelibs
cmakekde ../src/kdelibs
make install
```

Build Kdebase 4 Retrieve the current kdebase 4 source code from the KDE subversion trunk and install it.

```
svn co svn://anonsvn.kde.org/home/kde/trunk/KDE/kdebase ~/kde4/src/kdebase
mkdir -p ~/kde4/build/kdebase
cd ~/kde4/build/kdebase
cmakekde ../src/kdebase
make install
```

Running KDE 4 in an XNest session It is best to run a complete KDE 4 session. Since it is highly unstable this can be done using Xnest.

As the normal user running the current X-Session start a new Xnest instance:

```
Xnest :1
```

Then from the kde4dev user account start the KDE4 session:

```
export DISPLAY=:1
startkde
```

Now KDE4 starts up in Xnest and can be fully used. In order to actually test the Nepomuk-KDE components follow the steps in the next section before starting KDE4.

B.1.2 Installation of Nepomuk-KDE

Once KDE4 is properly setup the actual Nepomuk-KDE components and their dependencies can be installed.

Installation of Soprano/QRDF Soprano is a QT4 wrapper library around the Redland RDF framework.

```
svn co svn://anonsvn.kde.org/home/kde/trunk/playground/base/qrdf ~/kde4/src/qrdf
mkdir -p ~/kde4/build/qrdf
cd ~/kde4/build/qrdf
cmakekde ../src/qrdf
make install
```

Installation of Strigi Strigi is a meta data indexer and desktop search engine which is supported by Nepomuk-KDE through a KMetaData plug-in (see section 4.4).

```
svn co svn://anonsvn.kde.org/home/kde/trunk/playground/base/strigi ~/kde4/src/strigi
mkdir -p ~/kde4/build/strigi
cd ~/kde4/build/strigi
cmakekde ../src/strigi
make install
```

Installation of the Nepomuk-KDE components The following steps install all Nepomuk-KDE components in the KDE4 setup. This includes the backbone with the core services, KMetaData, and the example applications.

```
svn co svn://anonsvn.kde.org/home/kde/trunk/playground/base/nepomuk-kde ~/kde4/src/nepomuk-kde
mkdir -p ~/kde4/build/nepomuk-kde
cd ~/kde4/build/nepomuk-kde
cmakekde ../src/nepomuk-kde
make install
```

B.2 Tagging a File

Section 4.5 shows how the simple tagger can be used to tag a file from within the Konqueror. Just use the context menu and choose "Tag file...". In the presented GUI new tags can be created and assigned.

B.3 Searching for Tagged Files

Strigi provides a very simplistic testing GUI (real GUI applications are not available for KDE4 at the moment of this writing). It can be started as `strigiclient`.

The Strigi client allows full control over the Strigi system. First the Strigi daemon has to be started by clicking the appropriate button. Then the folders to index (i.e. the folders that contain the files tagged earlier) have to be selected and then the indexing of the files has to be started. Once the Strigi daemon is idle again, it is possible to search for files using the information set via KMetaData before. By inserting tag:foo into the search bar all files tagged with tag foo are shown[53]

[53] Strigi does not search all possible meta data fields yet. Thus it is necessary to prefix the search term with *tag:*.

C Programming with KNepClient

A full reference API of KNepClient can be found on the Nepomuk-KDE wiki[54] or in the sources themselves.

C.1 Writing a Nepomuk-KDE client

A KNep client is an application, plug-in, or piece of code that uses Nepomuk services to perform certain tasks (but does not publish services itself). Writing a KNep client is quite easy. One first has to create an instance of `Registry`[55] which will then automatically connect to the local Service Registry to retrieve the list of available Nepomuk services.

```
Nepomuk::Backbone::Registry* reg = new Nepomuk::Backbone::Registry( someQOject );
```

To actually use a service, one has to know the service type URL[56]. As an example the RDF storage triple service as defined by Nepomuk has the type URL `http://nepomuk.semanticdesktop.org/services/storage/rdf/Triple`.Thus, to get a triple service one simply asks the Registry to retrieve a generic `Service` object:

```
QString typeUrl = "http://nepomuk.semanticdesktop.org/services/storage/rdf/Triple";
Service* tripleService = reg->discoverServiceByType( typeUrl );
```

`Service` provides a generic interface via the `Service::methodCall` method. In theory one could do all communication with the service through this simple interface. But since that would be way to complicated (one had to know the exact syntax of the service's API) KNep provides `ServiceWrapper` classes for the most common service types.

In the case of the triple service, one can simply create an instance of `TripleService` as a wrapper around the Service object:

```
Nepomuk::Backbone::Services::TripleService tripleServiceWrapper( tripleService );
```

Now the triple service can easily be used as if it were a local object.

C.2 Writing and Publishing a Nepomuk Service

Writing a Nepomuk-KDE service that provides a certain service type is very easy[57]. In order to create and publish a service in the local Nepomuk system one has to create an implementation of one of the `ServicePublisher` subclasses. For each service type there exists a related ServicePublisher subclass which defines the methods to implement in an abstract interface.

Again the example will be based on the RDF triple service. To create such a service a new class has to implemented which is derived from `TripleServicePublisher`.

```
class MyTripleService : public Nepomuk::Backbone::Services::TripleServicePublisher
{
  Q_OBJECT
```

[54] http://nepomuk-kde.semanticdesktop.org/xwiki/documentation/index.html

[55] The *Registry* class is intended to be used as a singleton and might later be changed to enforce that usage.

[56] In the future service type handling will be simplified by type URL mapping and the introduction of type abbreviations. In addition *Nepomuk::Backbone::Registry* already has convenience methods to access the core Nepomuk services without knowing the service type URI.

[57] Creating a new service type that is not already part of KNepClient is a little more work at the moment. In the future creating service types should be handled by a tool chain that parses WSDL service description files within the KNepClient framework. For now, however, this has to be done manually. The steps to perform are not documented here and the reader is encouraged to contact the Nepomuk-KDE development team for help.

```
  public:
   MyTripleService( const QString& uri );

  public Q_SLOTS:
   int addStatement( const QString& graphId, const RDF::Statement& statement );
   int removeStatement( const QString& graphId, const RDF::Statement& statement );

   [...]
};
```

To actually register the new service implementation with the local Nepomuk system, an instance of `Registry` is necessary. Then registering the new service just takes single call to `Registry::registerService`:

```
Nepomuk::Backbone::Registry* reg = new Nepomuk::Backbone::Registry( someQOject );
reg.registerService( new MyTripleservicePublisher() );
```

From that point on Nepomuk clients can see and use the new service.

D Programming with KMetaData

A complete API reference documentation of KMetaData can be found in the sources themselves. They can simply be generated via doxygen[58]:

```
doxygen kmetadata/Doxyfile
```

This will create the sub-folder `docs` which in turn contains the API reference in the sub-folder `html`.

In general there are two ways of using KMetaData.

- The preferred way: use the Resource subclasses as generated from The Nepomuk Desktop Ontology This is also the much simpler way since KMetaData takes care of all type casting and list handling automatically.
- Using Nepomuk::KMetaData::Resource directly. This is much harder since in this case the type names (i.e. their URIs as defined in The Nepomuk Desktop Ontology) have to be known. On the other hand it allows to use additional resource types not defined in the ontology and handle resources without knowing their type.

Since all resource classes are derived from `Resource` and only add additional methods both ways can be used interchangeably. Resource objects (and thus also all objects of classes derived from Resource) with the same URI share their data. Thus, if one is changed the other one is, too.

D.1 Using Resource Subclasses

Using Resource subclasses directly is very simple. All that is necessary to handle a resource is to know its type and its URI (the URI can vary a lot between resource types; The simplest example is certainly a local file: the URI is the path to the file).

To access or create meta data for a resource one simply creates an instance of the corresponding class and passes the resource URI to its constructor.

In case of a file this would look as follows.

```
Nepomuk::KMetaData::File f( "/home/foo/bar.txt" );
```

Now meta data can be read and set via the methods provided by `File` such as `setAnnotation`.

Each resource class also provides a static method which returns all existing instances of this type. This includes instances in the store as well as locally non-synced objects.

D.2 Using Resource Directly

Using the Nepomuk::KMetaData::Resource class directly forces one to learn a little more about the internals of KMetaData. Resource provides four methods to handle the properties of a resource (reminder: all Resource subclasses as generated from The Nepomuk Desktop Ontology are based one these methods):

- `Resource::getProperty`
- `Resource::setProperty`

[58] http://www.stack.nl/ dimitri/doxygen/

- `Resource::removeProperty`
- `Resource::allProperties`

Each property's value is represented by a `Variant` object which can contain another Resource or a literal (string, int, ...) or even a list of the former two. Other than with the Resource subclasses no automatic type conversion is performed.

In case of a property that can have multiple values (cardinality greater than 1) `Resource::setProperty` has to be called with a list to set more than one (the Resource subclasses simplify this by adding add methods in addition to the set method) and `Resource::getProperty` will also return a list (in both cases encapsulated in a `Variant` object).

When creating a Resource object there are two cases that are dealt with differently:

1. The resource does not exist yet, i.e. no information about it is stored. In this case KMetaData does not know the type of the resource and will fall back to http://www.w3.org/2000/01/rdf-schema#Resource.
2. If the resource already exists the type may be empty. It will then be read from the local meta data store (where it was saved before by KMetaData automatically).

As a rule of thumb one should always define the type when creating meta data and leave it empty when reading meta data.

When using the plain Nepomuk::KMetaData::Resource class one is completely free to choose the resource URIs, the type URIs, and the property URIs. However, to preserve compatibility with other applications one is encouraged to stick to those define in The Nepomuk Desktop Ontology.

D.3 KMetaData Resource Manager

KMetaData is designed so the user (the developer of a client application) does not have to care about loading or saving the data. Unless auto syncing is disabled via `ResourceManager::setAutoSync` meta data is automatically synced with the local Nepomuk meta data store. (Currently the auto-sync feature is only partially implemented. Data will be synced once the last instance of a resource is deleted.)

Although in normal operation it is sufficient to only work with `Resource` and its subclasses errors might occur. This is where the `ResourceManager` comes in: it provides the `init` method which can be called manually (the resource manager will be initialized automatically anyway) to check if the initialization was successful and KMetaData can be used. In addition it provides the `ResourceManager::error` signal which is emitted whenever an error occurs. Errors include failed syncing or loading of meta data.

E Examples

E.1 KMetaData File Class

See below for an example of a class generated by KMetaData representing a type from the Nepomuk Desktop Ontology. File extends over Thing by introducing one more property location. Since location is a literal string value it is automatically casted in the method calls (as compared to the usage of Variant in the generic setProperty and getProperty methods. In addition each generated class contains a static method to retrieve all defined resources of this type.

```
class File : public Thing
{
 public:
   File();
   File( const File& );
   File( const QString& uri );
   ~File();

   File& operator=( const File& );

   QString getLocation() const;

   void setLocation( const QString& value );

   static QList<File> allFiles();

 protected:
   File( const QString& uri, const QString& type );
};
```

E.2 Resource Annotation

Commenting a file with an annotation is as simple as

```
Nepomuk::KMetaData::File f( "/home/foo/bar.txt" );
f.setAnnotation( "This is quite a nice file and I like annotating it." );
```

E.3 Resource Tagging

Other than an annotation a tag is not a literal value but tags are also resources with a single property: their name which is the one presented to the user. Tagging a resource consists of two steps. First a new tag has to be created; then this tag has to be assigned to the resource. Since a tag is an artificial resource it has to have a unique URI. At the time of this writing no automatic unique URI generation has been implemented yet so the developer creating the tags has to come up with some arbitrary tag URI manually.

```
Nepomuk::KMetaData::Tag tag( "http://sometaguri" );
tag.setName( "Nepomuk" );

Nepomuk::KMetaData::File f( ‘‘/home/foo/bar.txt’’ );
f.addTag( tag );
```

This will create a new tag named `Nepomuk` and assign this tag to the file `/home/foo/bar.txt`.

E.4 GUI Interaction

`KMetaData::Ontology` provides information about the Nepomuk Desktop Ontology such as human readable representations of types and properties. This

allows to simply present all properties defined for a certain resource in a GUI element.

```
Nepomuk::KMetaData::Ontology* ont = Nepomuk::KMetaData::ResourceManager::instance()->ontology();
Nepomuk::KMetaData::Resource f( "/home/foo/bar.txt" );
QHash<QString, Variant> properties = f.allProperties();
QHashIterator<QString, Variant> it( properties );
while( it.hasNext() ) {
  it.next();
  kdDebug() << ont->propertyName( it.key() ) << ": " << it.value().toString() << endl;
}
```

References

[1] Malte Kiesel Max Volkel Heiko Haller Mikhail Sogrin Par Lannero Brian Davis Mikhail Kotelnikov, Alexander Polonsky. Nepomuk deliverable d1.1 - interactive semantic wikis. Technical report, Cognium Systems, DFKI, FZI, IBM, KTH, NUIG, 2006.

[2] Enrico Minack and Leo Sauermann. Nepomuk deliverable d2.1 - adapters, extractors, and knowledge structure services. Technical report, L3S, DFKI, 2006.

[3] Leo Sauermann Tudor Groza and Paul-Alexandru Chirita. Nepomuk deliverable d6.1 - first version backbone and connector infrastructure. Technical report, NUIG, DFKI and L3S, 2006.

[4] Renault John Vasilios Darlagiannis, Roman Schmidt and Ekaterini Ioannou. Nepomuk deliverable d4.1 - distributed search system - basic infrastructure. Technical report, EPFL and L3S, 2006.

[5] Scott Wheeler. Tenor: A contextual linkage framework for kde. Technical report, April 2005. `http://websvn.kde.org/*checkout*/trunk/playground/base/tenor/docs/tenor-architecture.pdf?rev=475778`.