Akonadi – Pimpin' your PIM

Volker Krause / Till Adam

Why?

- overcome limitations of KMail and KResources
- provide convenient, safe, fast access to PIM data and metadata to everything on the desktop
- make it fun again to hack on KDE PIM
- because PIMsters wanted something with a cool code name too

Design Overview

- client/server
- DBus for control traffic

  + IMAP-like data transfer

  + standard storage formats (ical,vcard,mime)
- toolkit and platform independent protocol
- NOT a groupware server

Design

- filesystem-like structure: collections and items
- type-agnostic
- multi-part items with persistent unique identifiers
- collections with arbitrary attributes (ACLs, unread count, ...)
- change notifications
- cache for remote items, based on cache policies, not a storage server per se

Design

- heavily multi-threaded and multi-process
- one process (agent) per backend server or storage endpoint (ical file, maildir folder)
- cache currently a database (MySQL), important special cases like mail could be optimized to use maildir as the cache format

How do I use it?

libakonadi

- fully asynchronous
- KJob based API
- change monitoring, including notification filtering
- self-updating models with support for D'n'D, copy/paste, etc.

How do I use it?

libakonadi

- type-specific libs for KABC, KMime, KCal providing specialized models, etc.
- extensible to new types via serializer plugins
- value-based API, polymorphic types supported via smart pointers (tr1::shared_ptr)

Show me!

Mail support in 10 minutes

- writing a serializer plugin
- writing an agent
- writing a specialized item model
- writing a Plasma data engine

The serializer plugin

```
void MailSerializer::deserialize( Akonadi::Item& item,
                                  const QString& part,
                                  QIODevice& data ) {
     KMime::Message::Ptr msg;
     if ( part == Item::PartBody ) {
         msg->setContent( data.readAll() );
         msg->parse();
     } else if ( part == Item::PartEnvelope ) {
         // ...
     }
     item.setPayload<KMime::Message::Ptr>( msg );
}
```

The serializer plugin

```
void MailSerializer::serialize( Akonadi::Item& item,
                                const QString& part,
                                QIODevice& data ) {
     KMime::Message::Ptr msg = item->payload();
     if ( part == Item::PartBody ) {
         msg->assemble();
         data.write( msg->encodedContent() );
     } else if ( part == Item::PartEnvelope ) {
         // ...
     }
}
```

The Agent: Collection Listing

```
void MaildirAgent::retrieveCollections {
     Akonadi::Collection::List list;
     // read maildir subfolders and add them to list
     collectionsRetrieved( list );
}
void MaildirAgent::synchronizeCollection(
                   const Akonadi::Collection & col) {
     // create a Akonadi::Item per mail (without data)
     // and store it using Akonadi::ItemAppendJob
}
```

The Agent: Getting the Data

```
bool MaildirAgent::requestItemDelivery(
     const Akonadi::DataReference &ref,
     const QStringList &parts, const QDBusMessage &msg )
{
     KMime::Message::Ptr mail;
     // get data, ref.remoteId() helps you to find it
     Akonadi::Item item( ref );
     item.setMimeType( "message/rfc822" );
     item.setPayload( mail );
     ItemStoreJob *job = new ItemStoreJob( item );
     return deliverItem( job, msg );
}
```

The Agent: Reacting on Changes

```
void itemAdded( const Akonadi::Item &item,
                const Akonadi::Collection &collection );
void itemChanged( const Akonadi::Item &item,
                  const QStringList &partIdentifiers  );
void itemRemoved( const Akonadi::DataReference &ref );
void collectionAdded( const Collection &collection );
void collectionChanged( const Collection &collection );
void collectionRemoved( int id, const QString &remoteId );
```

Specialized Item Model

```
QVariant MessageModel::data( const QModelIndex & index, int role )
  const {
  Akonadi::Item item = itemForIndex( index );
  KMime::Message::Ptr msg = item.payload();
  if ( role == Qt::DisplayRole ) {
    switch ( index.column() ) {
      case Subject:
        return msg->subject()->asUnicodeString();
      // ...
    }
  }
}
```

Plasma Data Engine

```
NewMailEngine::NewMailEngine(QObject* parent, const QStringList& args) :
  Plasma::DataEngine(parent) {
      Akonadi::Monitor *monitor = new Monitor( this );
      monitor->monitorMimeType( "message/rfc822" );
      monitor->addFetchPart( Item::PartEnvelope );
      connect( monitor, SIGNAL(itemAdded(Akonadi::Item,Akonadi::Collection)),
           SLOT(itemAdded(Akonadi::Item)) );
}
void NewMailEngine::itemAdded(const Akonadi::Item & item) {
      KMime::Message::Ptr msg = item.payload<Message::Ptr>();
      QString source = QString::number( item.reference().id() );
      setData( source, "Subject", msg->subject()->asUnicodeString() );
}
```

Who did the pretty thingie?

Thanks

Thomas Moenicke did the swell plasmoid thingie. He couldn't come, had to stay at home in Berlin and do more pretty things while the rest of KDAB has fun in Glasgow. Three cheers for Thomas!

Can I help?

Ideas

- filter agents
- integrating live search as virtual collections
- support for searches on the backend (LDAP, IMAP)
- (local) subscription / subscription profiles
- OpenSync agent
- cool plasmoids and workflow-centric apps
- ...

Can I help?

Resources

- #kontact
- pim.kde.org/akonadi
- kde-pim@kde.org