Semantic collection for Amarok

Abstract :

Nepomuk has been a great semantic framework in recent years with many applications like Dolphin using it for managing metadata associated with its files and resources. But Amarok, still doesn't make use of the existing framework which comes bundled with any KDE distro. So, the objective of the project is to develop a Nepomuk based collection backend for Amarok which is functionally equivalent to the existing embedded MySql backend. An earlier attempt to achieve this was made this in GSoC 2008, but it never went on to be implemented and bundled along with the Amarok package because of reliability and performance issues. This is another attempt to make use of the excellent Nepomuk framework in Amarok and make it more semantic.

Project :

The Nepomuk-KDE Semantic Desktop[1] project aims to become the central storage for meta data, the one from the files itself (e.g. size, author, id3-tags) collected by Strigi, the file indexer, user entered (ratings, tags, comments) and most interesting automatically generated data provided by the KDE applications (e.g. file source, relations between files and contacts or play counts). Nepomuk provides access to all these data to all applications to help them show helpful context information and it will allow to search through the data. Nepomuk itself stores this data as RDF graph in Virtuoso.

The projects aim to develop a Nepomuk based backend collection plugin which is equivalent and independent to the existing embedded MySql backend. It is a two way mechanism which allows indexed data of Nepomuk to be used by Amarok, and the metadata generated by Amarok (eg. play count, song rating) to be written back into the Nepomuk index.

The Nepomuk based collection backend will be able to work along with the MySql backend. It will be feature complete in itself so that even if the Nepomuk collection is turned off, it won't affect the normal functioning of Amarok.

Another key part of the project is developing a strigi analyzer which uses taglib. Taglib is the most preferred library for audio file metadata extraction and Strigi can leverage the usefulness of taglib and extract metadata.

Highlights :

- There would be no extra burden (scanning and indexing) on Amarok as Strigi would have already indexed the audio files and extracted meta data from the whole system. Amarok would have to just read the metadata from the index through Nepomuk.

- Even if the song is modified outside of Amarok, Nepomuk keeps a track of all the changes and hence Amarok need not bother about external changes to its associated files.

- This will result in a uniform rating and commenting system in KDE. Other applications can make use of these ratings in them.
  Eg, If a user has rated a song in Dolphin, the same will be reciprocated in Amarok since both the applications use Nepomuk. Amarok already has a rating system in place, just that we need to use Nepomuk to handle it.

Another usage scenario is audio CD burning. For example, K3b can use the metadata generated by Amarok and populate the artist, album fields, rating etc.

- The user can tag a song with a website or to another contact, say he downloaded a song from a particular website he can tag that website or if he borrowed the song from a friend of his, he can tag that contact. This helps him remember the source of his songs. The nco:contact and nfo:website can be used for contact and website respectively.

  If a user bought music using Amarok stores like Jamendo/Amazon, the metadata about the purchase can be collected as well. There will be many people waiting to utilize this data and develop applications/plugins on top of it.

- The Nepomuk collection backend can be turned off and the existing MySql backend can be used. My interactions with the mentor proved that, this is infact very easy to implement with the current Amarok architecture.

At the end of the day, the purpose of Nepomuk will be served better if it is used by other applications.

Technical details :

The core of the project is to implement a NepomukCollection and NepomukQueryMaker classes. The classes to handle the generated and indexed metadata will be needed as well, eg a handler to write data back to Nepomuk, to update the UI using the metadata from Nepomuk index etc.

The NepomukQueryMaker can be developed into an API which can be used by plugin developers to exploit the Nepomuk backend.

- The existing database schema will be followed so as to not break the existing applications and plugins. So, the propagation to a Nepomuk backend would be seamless.
  The current schema can be found here[2]

  The NMM ontology will be used along with other ontologies. More information on this can be found here[3]. A few examples are

Duration	nfo:duration
Album	nmm:musicAlbum
Artwork	nmm:artwork
Rating	nao:rating

  A few required ontologies (one for background artist) seem to be missing, we can create our own ontologies in such a scenario or else existing ontologies will be used.

- Another feature that was asked for by the mentors is a Strigi Taglib analyzer. Taglib extracts metadata very well and is currently used by Amarok. Strigi is based on streams whereas taglib works on files. We need to figure out a way to use Taglib in Strigi, and build a suitable Strigi analyzer. I'm not familiar with the intrinsics of Strigi so achieving this requires considerable research. I plan to use the initial community bonding period for this purpose.

- The code written during the 2008 GSoC attempt will be studied if it can be reused or not. Nepomuk has come a long way since 2008 and the previous code will be tested for its relevancy in the current scenario.

- The UI need not be changed much or tinkered (as the mentor suggested). The existing UI will remain the same and the Nepomuk backend will be completely isolated. The entire project will have no dependency on the UI and will not add new Qwidgets or UI elements. The main reason being that the user should be able to turn off the Nepomuk backend and keep using Amarok as it was before.

- At the start of the project, I'll research Gnome's Tracker and check how it uses Nepomuk. Or, if time permits a separate Tracker plugin could be implemented. Decisions will be taken after thorough research and interactions with the mentor and the Tracker team. This is important as Amarok is not platform or desktop environment specific anymore.

- The mentor has asked for unit tests for the collection backend in the final stages of the coding phase. I do not possess expert proficiency in writing unit tests, but after a quick brush up on the basics, will come up with tests to validate the project code, esp the Collection backend.

What next?

If this project is successfully completed as a plugin which can be turned on or off, then it will lead to development of new applications. Applications can use the indexed metadata to perform metadata based searches like ' song : Stairway to heaven , date : 21st February '. Similarly, the user should be able to retrieve the song he played on Christmas last year.

De-duplication :

This is a feature that was requested for in the Amarok forum, removal of duplicate songs from the disk and not just the playlist. This can be easily implemented using this plugin. Using the sha1 hash of each song resource, they can be compared and checked if they are duplicates of each other. A simple SPARQL query like this may suffice to find the duplicates.

```
QString( "select distinct ?u1 where { "
                 "?r1 a %1 . ?r2 a %1. ?r1 %2 ?h. ?r2 %2 ?h. "
                 "?r1 %3 ?u1. ?r2 %3 ?u2. filter(?r1!=?r2) . }order by ?h ")
         .arg( Soprano::Node::resourceToN3(Nepomuk::Vocabulary::NFO::Audio()))
         .arg( Soprano::Node::resourceToN3(Nepomuk::Vocabulary::NFO::hasHash()))
         .arg( Soprano::Node::resourceToN3(Nepomuk::Vocabulary::NIE::url()));
```

An application to search for songs(on the hard disk) using its lyrics (nid3:hassynchronizedText or equivalent) can be developed. This removes the need to use a web service to search for songs using lyrics.

Other use cases like CD burning etc have already been mentioned before.

Note : This is not the primary priority of the project and will only be implemented after the Nepomuk backend collection and other project goals are achieved. Extra features like this are only a indication of what can be done after completion of the project, so that it can serve as an example for other developers and motivate them to try it out and build applications on top of it.

Time Line :

My university exams start and get over by the end of May and I would be able to start coding full time, after the completion of my exams. I have no external constraints and can work on the project spending atleast 40 hours a week for the entire duration until the pencil down phase.

24th April to 7th May	Go through Amarok code base and its Collection abstractions. Brush up on Nepomuk. Review 2008 GSoC code for its present day relevance.
7th May to 31st May	Study how Strigi works, especially the streams. Check on Tracker. University exams being the reason for the long duration
4th June to 6th June	Decide on the changes to the db schema if needed. Decide on Tracker plugin. Discussion with mentor about the Strigi analyzer.
7th June to 22nd June	Coding on the Nepomuk collection plugin. Decide on what goes into the database and the right ontologies to be used. Have a working NepomukCollection at the end of this phase.
25th June to 30th June	Read up on Unit Tests and write tests for NepomukCollection
2nd July to 18th July	Implement the NepomukQueryMaker. A possible Amarok API to be developed in line which can be used by plugins after feasibility analysis. Mid term evaluation. Get feedback from mentor on overall project design.
19th July to 21st July	Unit Tests for NepomukQueryMaker
23th July to 8th August	Work on the Strigi – Taglib analyzer after inputs from the mentors. Test the analyzer.
9th August to 18th August	Testing the entire project along with documentation. Buffer stage, consult the mentor if changes or additions have to be made. Try implement a side effect application using the newly built collection backend. Maybe the de-duplication feature.
20th August to 24th August	Pencil down phase, discussion with the mentor over the work done, and on taking the project forward

Note : Based on the decision on implementing the Tracker plugin, the timeline will be suitably adjusted during the project term.

About Me :

My name is Phalgun Guduthur, a 21 year old final year under grad majoring in Computer Science in PESIT, Bangalore India. I have experience in C++, Qt and Web development (JS, PHP, Python). I have been an avid supporter of open source since 2years now. Have been itching to contribute to an organization with the stature and size of Amarok since then.

I attended a conference on KDE held in Bangalore, India last year called conf.kde.in where I was introduced to Nepomuk by Vishesh Handa himself. I have prior experience in open source with my college project which is hosted in KDE repos. Its a semantic resource browser called 'RepontiK' built on top of the Nepomuk framework. More details can be found here[4]

My experience in Nepomuk and love for Amarok makes it an ideal challenge for me. The interactions with the community and meeting interesting people is also a driving force. I always wanted Nepomuk to go mainstream and happy to help in achieving it.

The purpose of GSoC would not be served if I don't continue working for Amarok after the completion of this project. I look at this as a way to get a foothold in the community and keeping working for it and not just limit myself to this project.

Finally, I can assure you that I am capable of and want to take responsibility of code maintenance post GSoC and get people to use the plugin hopefully as their primary backend collection plugin.

Contact Details :

Phalgun Guduthur

irc nick : phalgun on #amarok and #nepomuk-kde
email : phalgun.guduthur@gmail.com
github : http://github.com/phalgun

time zone : UTC +530

External Links :

[1] http://nepomuk.kde.org/
[2] http://amarok.kde.org/wiki/Development/Database_Schema
[3] http://test.semanticdesktop.org/ontologies/nmm.html
[4] http://quickgit.kde.org/index.php?p=scratch/hegde/resourcebrowser.git&a=summary