1 Problems

1.1 Monitoring

Both Nepomuk Backup and Sync were created with the assumption that we would be able to log every change made in Nepomuk Repository. While that is still possible, it is a huge resource hog and not practical.

Plus, in systems where there aren't many nrl:DiscardableInstanceBase graphs, the memory occupied by logging every statement would consume large amounts of memory, which may even double the actual storage size.

And then there is question of how long should the logs be kept? Syncing can only be done as we have the logs.

1.2 Backup is not a subset of Syncing

It was assumed that backup is just a special case of syncing, while that is true. Syncing is more of a continous process, whereas restoration of a backup is a one time afair.

During backup restoration, the identification needs to be performed once after which the change log can be merged, and the identification results forgotten. In syncing, as it would be done many times, it would be benefitial if we somehow remembered the identification results, that way we would not be bothering the user each time when the identification fails.

1.3 The DMS and SyncLib overlap

The identification mechanism used in the SyncLib is based on the percentage of identification resources that are matched. The deafault being 80%. This approach does not work well when there are very few identification properties are even a single one does not match. In many cases the identification properties just did not exist on the system in which cases identification would fail.

The approach followed by the DMS, which is based on using better identification properties, and making sure that no identification property does not match. Existance is not a critera for matching.

The identification code of SyncLib uses manual categorization instead of relying on sparql queries and is therefore extremely slow. (Over 1 second per resource, in the simple cases) The DMS on the other hand only relies on Sparql queries, and is very fast.

The SyncLib identification code will need to be thrown and the DMS code will have to be adopted.

2 A better approach

Since monitoring is inherrently error prone, it would be better to not log anything, and rely only on the information provided by Nepomuk i.e. per statement and resource modification time.

The main disadvantage with this approach is that we have no conclusive way of knowing when statements have been deleted. For resources, we should modify the nao:lastModified whenever a resource is modified, and later which can check which all statements have been modified since this date.

But for Resources that are completely deleted, there is no simple way. One appraoch would be to mark deleted resources with a nxx:DeletedResource, and the other would be to log all the deleted resources.

2.1 Backup Files

- There is no need for explicitly storing the identifying properties in a different file. So, the changeLog and IdentificationSet can be combined.
- As per statement logging is not possible, the changeLog will just be a list of statements.
- The backup file format will have to be redesigned. It will consist of one file which contains all the changed statements along with a 'status' file which gives common metadata such as time of last backup.
- Instead of generating all the backup statements on each backup, only the statements that have changed since the last backup will be saved.
- Since some resources may have properties which have been deleted since the last backup, all the resources which have a nao:lastModified $>$ last backup time will be backed up entirely.
- This still doesn't cover the resources that have been completely removed. We need some mechanism to track deleted resources, and add those to the Backup File.

2.1.1 Algo

1. If first time - Get all statements whose graph type is not nrl:Ontology or nrl:DiscardableInstanceBase. Serialize them, and create a status file which contains simple metadata about the backup
   Eg - Automated Backup, backup date
   Compress the serialized file along with the status file.
2. If backup exists - get the backup time - dt.
3. Get all the resources whose nao:lastModified $>$ dt, and serialize the file.
4. Get all the graphs whose nao:created $>$ dt, and serialize them as well.
5. Create a file 'status2' with the backup metadata.
6. Compress the serialized file along with 'status2' in the original backup file.

2.2 Identification Mechanism

1. Check for properties which uniquely identify the resource such as nfo:hashValue or ISDN number for books.
2. For Files - Check the nie:url after translating it to the correct home folder.
3. Call DMS storeResources

2.3 Merging

- The ideal approach would be to combine the SyncLib's, and DMS's ResourceMerger, but that is possible cause the DMS uses the class and property tree heavily.
- While restoring a backup, it is assumed that the cardinality and domain/range is perfect as it is being imported from another nepomuk repository.
- No type checking will be performed and all the identified resources will be pushed.

2.4 Separation of logic and GUI

It serves no point when restoring a backup. No one is going to create their own restoration GUI, and even if they wanted to they can easily modify my code.

The separation of the backup creation code makes a lot of sense.

3 Synchronization

As synchronization is similar to backup, a sync file will follow the same format as a backup file.

3.1 Identifying different machines

We need some kind of ontology will allow us to uniquely identify a Nepomuk Repository, and possible annotate it with some identifier such "Desktop" or "Laptop".

With that kind of ontology in place, during synchronization we could save the results as -

```
<nepomuk:/res/A>

    a nfo:FileDataObject ;
    nxx:sameAs <nepomuk:/laptop/res/A> .
```

That way we would only have to identify the new resources.

3.2 Communication

How would the two machines communicate? Both would need each others last sync date (which can be stored in a separate ontology) and the generated sync files would need to be transferred.

If sync files are generated from both the machines, the identification will need to be performed on both the machines as well. This would result in the user having to resolve some resources manually on both some systems.

It would be better if the identification could be performed on one machine and then the changes synced.