A component based re-design

Why?

- The code is still a mess even though it has been converted from an ant to a maven project.
- The Harvester-core module is just a trashcan location for anything remotely harvester related.
- Unit tests are separated into their own modules making code coverage close to impossible.
- Impractical to use 1-2 weeks of running test cases which should be covered much better by unit testing. (every time anything major is released)
- Registering/deregistering JMS listeners just seems wrong and it is surprising it has worked so far. Nothing in the JMS specifiction suggests that this is even a good idea.

Overall solutions

- Isolate storage implementation completely. It is going to happen sooner or later anyway.
- Remove JMS as much as possible. (Except from legacy code like the bitarchive implementation)
- Split the different parts of NAS into separate modules with little or no dependencies.

Reasoning

- It is easier to build/test/deploy. With improved and localized unit testing you could most likely...
    - release much quicker without having to use 1-2 weeks testing.
    - release only the module(s) you changed.
- People who don not want to use NAS could instead just use the parts that fit their needs.
    - For example people could use the harvest controllers and nothing else.

Harvest Control Manager Component

- Remove the JMS dependency from the controller.
    - Instead use a REST interface or some other means of exposing a simple extendable API.
- Remove the notion of channels from the controller.
    - The management of organizing controllers into groups is left to the user of these APIs.
- Make the code independent of the rest of NAS so it can be used not only by NAS.
- Controllers should be deployed independently of the rest of NAS.
- Use a plugin architecture for core functionality. (Use classloaders)
    - configure harvester
    - build progress reports
    - build metadata files when the job is complete
    - upload data to persistant storage
- A controller is built for a specific harvester; H1, H3, API
- Extendable using custom commands that the plugins add to the controller. (Thinking beyond H3...)
- The API should include all required functions to control the harvest manager
    - Submit job.
    - Upload configuration files.
    - Upload additional files; indexes etc.
    - Start job.
    - Get progress/report.
    - Stop job.
    - Initiate metadata generation.
    - Initiate upload.
    - uploading new versions of the plugins.
    - uploading new versions of the harvester package. (h3 bundle)
- Offer base client implemention. (Used by a job manager/monitor)

The API should make it possible to redo certain operations which occasionally fail and require manual intervention.

Only when the worst happens should i be required for a person to fiddle with the server.

The existing code in the harvest control manager that must be migrated into one or more plugins.

- move files to h1/h3 folder.
- build metadata reports.
- build progress reports.
- upload data to the bitarchive

The plugins can form the base for other people adapting the code to suit their own needs.

The only state the manager should know about is it's harvest.

Everything else is up to the caller. This way you can use "channels" or not. And more importantly you can manage "channels" dynamically without having to reconfigure and redeploying harvest controll managers.

Job Scheduler Component

- No more JMS!
- Harvesters can be added dynamically either in the GUI or by periodic scans.
- Provide a simple service to talk to all harvest control managers and handle their state.
    - Idle.
    - Harvest workflow In progress.
    - Software update.
- Provide GUI configuration for managing "channels" without reconfiguring/redeploying.
- Expose an API for the most common operations.
    - Polling a database constantly is just BAD practice.

GUI

- Move from JSP to template driven servlets.
    - Ability to unit test almost everything instead of time consuming manual release tests.
    - Easier for institutions to customize or create their own interface/layout.

Wayback Indexing Server

- Is very very very slow at indexing using a single thread.
    - Should instead use an improve multithread batch system.
- CDX generation should be done on the harvest controller after metadata generation.
- Remove JMS.
- Add API.
- Maybe add some GUI to show status of indexing and other minor tasks.

Deploy package

- Should be deprecated as much as possible.
- Each component/server should include an assembly to build separate installation archives.
- Split the deploy code from the configuration code.

Restructuring tasks

- Cleanup H3 harvest control manager.
    - Remove JMS.
    - Split H3 management code and Netarkiv code apart.
- Provide abstract to manage harvest control managers through an API/REST/NIO.
- Rewrite Job Scheduler to use the new control interface.
- Cleanup harvester-core package into more appropriate modules.
- Cleanup indexing server.
- Cleanup deploy package.
- Isolate storage completely exposing only simple interfaces.

Component platform

- Applications should run on the same basic "framework".
- Interact through an API/REST/NIO to make it as independent as possible.
- Encapsulate the application in a classloader context that can be updated/restarted by the application through the API.
- Find or construct an appropriate "framework" using as few small 3rd party components as possible.

Prototyping

Since it is a daunting project to just refactor everything it would be best to just start with the harvest control manager and see how easily it can be turned into an independent component.

Depending of how this works more of the restructuring tasks can be done in an other that makes sense. However reworking the Job Scheduler for work with the new control manager component is probably the most logical step.

The order should also consider refactoring parts of the code that would eliminate long known bugs or other areas that have not been focused on. Such as manual workflows.