Concept Activation

Draft for 5.2, 5.3
Proposal for rewriting/improving existing activation.

Current status

Activation is realized as a push system and is divided in two parts. Sender part consists of the `Syndicator` and `Subscriber` where `Subscriber` is a description of the target system. `Syndicator` is a component responsible for collecting all pieces of information that need to be transmitted to activate content successfully. All such pieces are collected in the object called `ActivationContent`. The simplest way to initiate activation is using `ActivationCommand` that will call syndicator to create activation content and ensure such is published to each of the subscribers. Receiver part is realized as a `ReceiveFilter`. The filter is responsible for acting upon incoming activation request and storing the data in proper location and order after receiving all the data.

The two possible methods of activating content are supported:

- single (non-recursive) content activation
- recursive content activation (given content and all subcontent).

On top of the above 2 possible variations of `ActivationContent` can be used to perform the activation

- current content
- versioned content

The `Content` is in this context defined by the filtering `Rule` used collect all the subnodes that are still considered to be part of the content for any given activation.

Main Issues

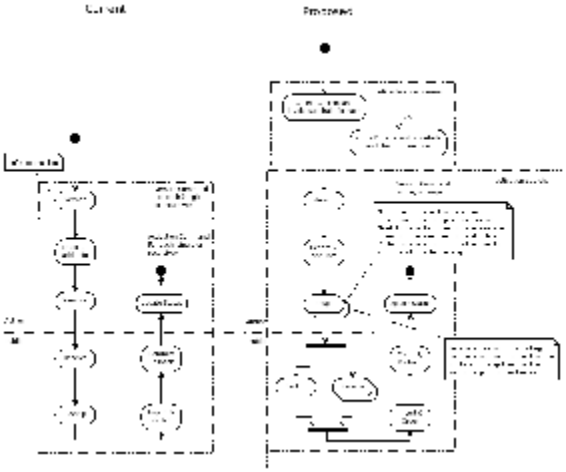
- non-uniformity of recursive activation. Currently, when recursively activating the content, activation of versioned content has to be treated differently since content is delivered as a list to the activation rather then tree such is during recursive activation of the non versioned content.
- compactness. Whole system is realized as one `Syndicator` and one `ActivationContent` with many if/else clauses and methods used only in some of the above mentioned cases. This creates for high number of possible combinaitons that are difficult to test separately and missing some is not immediatelly obvious. Also due to convoluted way of calls between the methods, changes in one of them often causes side effects in other scenarios that are difficult to predict.
- non-extendibility. Due to above mentioned it is not easy to provide additional syndicator implementations that would allow new activation scenarios such as for example activation of arbitrary lists or conditional recursive activation without resolving to adding even more methods to handle such cases.
- incomplete info transfer. The current activation data set doesn't contains always all the information necessary to place the content in proper place (contains only info about previous/next sibling, but not whole structure and not complete current hierarchy order incl. the last ops (e.g. unactivated moves on the author)
- incorrect information used when collecting data. The ordering info status transfered to the public is always the "current order" even when activating versioned content.
- monolithic transfer - currently it is not possible to speed up activation by transferring chunks of activated content in parallel.
- uncertainty. It is not possible to predict exact state of the content on public instance. e.g. content might have been bootstrapped there even though it was never activated, or content might have been deleted manually even though the activation status on author says that content have been activated. Nodes might have been moved(reordered) on public manually, should activation honour this or override?
- potential OutOfMemoryErrors and high GC load during acitvation of large data sets, because activation requests sent from author to public are not chunked and instead completely held in memory once

All currently open activation related issues

Somewhat categorized collection of features activation currently does (or in the future should be able to) take care of:

The issues with extensibility and testability of the current activation code stem from the fact that module is written in very monolithic way, with single implementation trying to deal with all possible use cases.
The source of performance issues is over simplification of the process by stacking the commands in a chain without taking into account the amounts of data each command needs to handle separately.

The diagram below lists the current state and possible future implementation. The elementary steps are more less the same, the main difference is always chaining the commands only at the level of single content.



Target

The new API/implementation should support (or in the very least allow for later implementation of):

- testability ... keep the code in small manageble, easy to understand chunks
- extendible API
    - single content act., recursive act., versioned content act., arbitrary list of content act. ... all those should be separate implementations of some interface
    - different types of subscribers ... subscriber should define the transport method as well as mapping on the target
    - possiblity to change the transport mechanisms (multipart, parallel chunked transfer, rss publishing?, ...)
- parallelization of content transfer to different subscribers (already exists)
- parallelization of the activation of multiple pieces of the content to single subscriber
- counter the uncertainty
    - negotiation of the status between A and P prior to activation?
    - generation and transfer of complete content map incl. results of ops such as move?
    - callbacks from public upon forced deletion/move/rename of the content?
    - for both activation and deactivaiton transmit always UUID and path ... to ensure nodes with same names, but created separately (e.g. config nodes created by install tasks) can be always handled properly.
    - info related to the ordering of the content need to be complete - i.e. provide info about all content in the author (even not activated one since it might exist on public anyway) and also provide the info about status (useful specially for modified (moved) not yet activated content that might influence the order negatively. It would be also possibly useful for recursive activations to double-check the order after all the activated content exists on public (or make this the only time when ordering is performed to speed up the activation) MAGNOLIA-2434@jira,MAGNOLIA-2731@jira
- usability
    - tracking of status of the activation per instance
    - tracking of progress of the activation (e.g. when content activation have been initiated but is somewhere in the workflow (approval, time based publishing) MAGNOLIA-2303@jira
    - tracking of progress of the activation via AdminCentral and not only via log files MAGNOLIA-1186@jira
    - ability to show activation status for the children recursively, or specific view to filter content based on the activation status MAGNOLIA-2043@jira
    - use public/private key or some other mechanism to authenticate during activation to avoid dependency password synchronization. The mechanism used should be able to cope with the fact that the password/key might change and change needs to be activated before it is applied. MAGNOLIA-1355@jira, MAGNOLIA-1644@jira
    - activation of deletion MAGNOLIA-2156@jira,MAGNOLIA-2251@jira,MAGNOLIA-2204@jira
    - hook for arbitrary feedback from public back to the author above the current possibility of transferring the error message MAGNOLIA-2089@jira

Possible solutions

Use of JMS to facilitate transport and ensure delivery even when some public instances are temporarily unavialable

While on first sight it seems that use of JMS would be a great idea, there are some shortcomings of such solution that need to be thought out first:

- ~~transport size. Most of the JMS servers are not well suited to transfer big amounts of data. While activation of pages is usually small enough to easily fit message envelope, transfer of big binaries will cause issues and requires prior testing of suitable JMS server.~~ ActiveMQ supports sending of large binary files - http://activemq.apache.org/can-i-send-really-large-files-over-activemq.html
- ~~delivery times. JMS works asynchronously by design. This is unfortunately in contradiction with requirement to deliver and make available content on all subscribed public instances at the same time. If content is not delivered to all currently on-line instances at the same time, some instances might already serve new content, while others would still serve the old one. This would either require special configuration (sticky sessions) of load balancer or would lead to inconsistencies in content ppl might see when accessing website from outside.~~ It is possible to send messages synchronously - http://activemq.apache.org/how-do-i-enable-asynchronous-sending.html
- ActiveMQ:
    - has its own implementaion of transactions (commit/rollback) - http://activemq.apache.org/how-do-transactions-work.html
    - supports reconnect to the broker
    - can be monitored for example with JMX
    - supports multiple network protocols http://activemq.apache.org/uri-protocols.html

I think its worth to explore possibilities of JMS to use it for activations. However there is a downside for this solution - customers will need to set up new server with some sort of a storage (disk, database) where messages will be persisted.

Jackrabbit JCR-RMI (http://jackrabbit.apache.org/jackrabbit-jcr-rmi.html)

Jackrabbit provides its own implementation of rmi to communicate with another repository remotely. It looked like a good idea to use this implementation, however there are some difficulties that would require quite a lot of effort to overcome

- Author needs to connect to the rmi service, credentials that works are admin/admin. These are defined in magnolia.properties. IIRC we do not recommend to change them so it means we would need to rewrite/add new security mechanism for the rmi connections.
- Current implementation of activation uses zip to compress size of transported content. Sending uncompressed xml files over rmi would have significant impact on activation speed. There we would need to have some wrapper that would sent zipped content over rmi.
- Re-implement commit/rollback mechanism in transactional activation
- According to the jackrabbit jcr-rmi implementors jcr-rmi performance is low

Imho this is not a way to go. It would be difficult to notify public instances when to rollback, commit, also public instances would have to report to author if activation was successful or not. We would also need to solve authentication to the remote repository.

Socket connection

Using of sockets would require to re-implement everything from the scratch. Positive thing is that we would have everything under control. Unfortunately there are these downsides:

- complete reimplementation of security, transport, commit/rollback mechanism, etc
- parallel activation - we would need to open new sockets for parallel activations
    - i believe customers would not like to open ports for that
- non-persistent activation storage mechanism
    - any public instance that is not responding would cause a rollback of activation
- sockets would be pretty much the same as the activation over HTTP

Clustered repository

Clustered repository for author and public instances is another idea how we could implement activation. However we would have same problem (as with JCR-RMI) - how to notify all author or public instances. Another thing that should concern us is repository corruption. This would definitively cause a lot of troubles to the activation process and it might happen that whole activation would broke up and nobody could activate anything.