22BlockImplementation

Migration path from current monolytic application to a decentralized block system

The goal of this evolving document is

1. providing a migration path from our current monotlic application and
2. to find out what we have to do on the Cocoon side (see BlockManager, component lookup, block protocol)

This is written without having it discussed with Pier - so possible classloading issues are not considered. His input is very appreciated 🙂

Component management in general

- provide a descriptor for each block
- every block has a root sitemap
- every block has a BlockManager
  - provides external access to all public components
  - provides internal access to all private components
  - performs component lookups: if the component is NOT locally available, the lookup is delegated to the BlockManager of the block that provides access to this block
  - components can be managed by different frameworks
    - Spring: we provide a special BeanFactory implementation that can lookup public components of other Blocks
    - ECM: service manager uses the BlockManager to perform components lookups

Sitemap & Source resolving

- a special "block" protocol that can lookup sources of other blocks (it's also aware of block inheritance --> possible usecase: fallback mechanisms)
- order of source resolving
  - the root sitemaps of blocks are implicitly mounted, this means that they match first --> if not, passthrough
- use of a sitemap components
  - core sitemap componet `<map:generate type="file"/>` or `<map:generate type="core:file"/>`
  - local component `<map:generate type="local:myComponent"/>`
  - a sitemap component declared by another block `<map:generate type="anotherComponent:yourComponent"/>`
- input modules could be declared by other blocks
  - use of a core input modul {request-param:xyz} or {core:request-param:xyz}
  - use of a local input modul or

    ```
    {local:myIM:myParam}
    ```

  - use of an input module of another block

    ```
    {otherBlock:otherIM:yourParam}
    ```

- source factories could be declared by other blocks
  - tbd
- link rewriting (one block does not now where the other block's URLs are mounted - implemented as transformer)

O/R-mapping

O/R-mapping tools have some kind of super factory that does all the persistence stuff (object creation, updating, deleting, ...). Used in combination with blocks this means that the metadata are spread over blocks and they are not at a single place.

As there might be dependencies between objects of different blocks, this super-factory has to be configured using all the metadata files of the different blocks. It should be possible to replace the base implementations of those super-factories by an own implementation that has access to the BlockManager that has access to the metadata files.

Another option would be that every block registers its o/r-mapping files at an super-factory-metadata-registry.

This way O/R-mapping relies on the "feature" that all classes put into the public JAR of it, can be accessed from other blocks.

Logging

Any implications on logging because of blocks?

Kernel implementation - or: How do he things described above fit into the current design of kernel?

Here input by Pier would be *very* helpful.

Backwards compatibility

- all existing components can be reused
- sitemap? (declarations of components located in blocks will change --> only affects root sitemaps)