Blocks

Blocks are reusable functional modules at the cocoon web application level.

Blocks were introduced in Cocoon to allow users to:

- easily deploy their content on cocoon without requiring operation downtime
- package functionality and services in modules that can be reused as-is
- easily extend existing modules
- create complex web applications by high-level composing of these functional modules
- depend on module behaviors, allowing for polymorphic module composition

Read the BlockIntroduction to get a picture of the concept.

Blocks Design

- BlocksDefinition - This document defines what a block is and how it works.
- BlockIdentification - This document defines the URI scheme used for block identification

---

Blocks Implementation Workspace

- BlockImplementation - This document contains the plan for the implementation of the block design.
- BlocksFSLayout - This is an example of the file system layout that will be used as a contract between cocoon and the block deployer.
- BlocksWiring - This is an example of the markup used by the wiring document (wiring.xml), created by the block deployer and used by cocoon to load the block wiring at initialization time.
- BlocksCob - This is an example of the markup used in the block descriptor document (block.xml), contained inside the block, that contains the metadata that is used by the block deployer to deploy the block, know its dependendies, the eventual behaviors implemented, the eventual block extended, required parameters, provided components and sitemap.

Implementation Phases

Phase 1: definition of the contract between the block manager inside cocoon and the standalone block deployer. These contracts include:

1. description of the BlocksFSLayout (**done**)
2. description of the BlocksWiring schema (**done**)
3. description of the BlocksCob schema (*in progress*)

Phase 2: definition and implementation of the block data model, with reading/writing capabilities

1. implementation of the block wiring data model
2. implementation of the xml -> data model parser
3. implementation of the data model -> xml serializer

Note: since the xml formats are not meant to be human editable, roundtripping of comments or formatting included in those xml files should not be a priority.

At this point, implementation can work parallel:

Phase 3 - cocoon side: implementation of block support.

1. implementation of the !BlockManager
2. implementation of the block: protocol handler
3. implementation of the link transformer
4. implementation of the reload watchdog

The link transformer has to be "block-aware" in order to identify where other blocks are mounted]

Note: during this phase, development can happen with an handwritten and extracted block wiring info and block descriptors.

Phase 3 - deployer side: definition of the interfaces between the components

1. the Locator interface
2. the Block services interfaces

Phase 4 - deployer side: implementation of a basic block deployer

1. implementation of the block services
2. implementation of a "file system"-based locator
3. implementation of a command-line user interface

Phase 5 - deployer side: implementation of a webservice block librarian

1. implementation of a REST-style web service locator
2. implementation of a cocoon block that implements block librarian capabilities

Reader Comments

"Block" has a connotation of "lump" – these things are "active" and should have an "active" name. See "Block" @ thesaurus.com, compared to for instance "Engine". I realise that "Block" has a meaning in electronics but it has other connotations in everyday English that are undesirable IMHO.
– Con

In electronics "block" is not really used. The smallest building blocks (hmm?) are called "components", which may be assembled to "modules", which may be used in "subsystems". I think ASIC designers uses "Block" at silicon level though.
– Niclas Hedhman

Blocks are not active nor passive. Just like electronic chips, they are useless by themselves, but provide functionality if properly connected with an external system. This is not the behavior of engines, for example, which are active by themselves. Calling them components would create conflicts. Blocks are used by some Avalon containers. There is a potential conflict here, but there is also potential overlap (it could be possible, in the future, to install an Avalon Block in Cocoon) so the use of the same name is justified.
– stefanomazzocchi

Have people considered using something like [HiveMind] as the infrastructure for supporting blocks, rather than 'rolling our own'? I don't see that we'd need to build much (if anything) on top of it, and it would be seriously elegant: installing a block is a case of dropping it in the 'lib' directory of your .war file. I'd be more than happy to assist in making this change, although I'll need to get back up-to-speed with cocoon first!!
– [PaulRussell]

Related technologies and projects

- Debian apt-get – *In the beginning there was the .tar.gz. Users had to compile each program that they wanted to use on their GNU/Linux systems...* Sounds familiar, doesn't it?
- Maven – maven uses a centralized repository for functional modules (jars in this case).
- jjar – JJar an attempt at making a CPAN-like service/infrastructure for the Java development community.
- Deployment on JBoss – interesting to compare with the requirements for Blocks.
- Avalon Blocks – both Merlin and Phoenix are Avalon containers that have the notion of functional blocks.