CocoonBestPractices

Cocoon Best Practises

Lessons learnt during the development of big projects.

Speaker : Jeremy Quinn of Luminas, *Venue : Cocoon GT2004*

Or : How to avoid my stupid mistakes

I am a developer for Luminas, a web development company in the UK, specialising in Open Source solutions to industry. Luminas is a member of the Orixo XML Business Alliance.

This presentation is about best practises for development of projects within Cocoon and is a distillation of techniques culled from the Cocoon mail lists and my Orixo colleagues.

You can download or view the presentation

Contents

The talk will be broken down into these sections.

Write Use Cases

- Describe your project
- Keep your use cases updated
- Find the happy balance

Use cases are a formalised way of describing how a piece of software will work from the point of view of specific ‘actors’, eg. Users, Admins, The System etc.

Luminas are in the process of preparing their in-house UseCaseML and sample XSLT for the community to use.

It is work in progress, so we hope others will get involved in improving it.

Describe your Project

- Developers need to know what to do
- Clients need to know what they will get
- Managers need to allocate resources

Writing use cases is the stage when you make sure that everyone knows what is supposed to be happening.

Use cases can be used to make sure everything that is being asked for is possible.

Everyone involved in a project needs to be able to flag up any issues involving their domain.

Use cases can form the basis of the functional contract between you and the client.

Use cases can tell you when you have finished !!!

Keep it updated

- Projects change
- Your use cases should reflect this
- Keep them versioned
- Publish them online for the client

As the project changes due to engineering or client constraints, your usecases should be updated to reflect the changes.

This allows both the client and the developers to keep in sync and make sure changes are properly described and possible to implement.

Keep your use cases in a versioning repository like your source code.

Code releases will typically be related to use case versions.

Find the balance

- Enough detail for it to be useful for the developer
- Written in language the client can understand

Finding the happy ground between descriptions the client can understand and a level of detail that is useful for the developer can be difficult. Use cases should not be considered a marketing document, but an engineering document. Luminas use them as the point of communication between developer and client. They need to be written in collaboration between both groups.

see attachment below

This is a slim sample from an ongoing project at Luminas.

You can see from this a description of a single use case, and how it relates to other use cases.

We are still learning how to use this tool effectively.

Testing

- Continuous testing
    - Cruisecontrol, Gump etc.
- Write unit tests
    - jUnit, ejbUnit, Cactus etc.
- Automate functional tests
    - Anteater, httpUnit, Cactus etc.

Products like Cruisecontrol or Gump can be used for continuous testing (Luminas do not actually do this yet).

Write unit tests for business objects. Tests can be run by Ant, Eclipse or Maven etc. Test suites include jUnit, Cactus, ejbUnit etc., Luminas currently use jUnit.

Tests should be conducted before the code is used from flowscript etc. Run the tests before committing code to reassure yourself that you are not breaking anything.

Use Anteater, httpUnit, Cactus etc. for automated functional tests. Maven can impose many of these best practises on you.

Sitemap Usage

- Design your URI request space
- Setup the top level
- Use Sub-Sitemaps
- Use Resources

The sitemap is a key and maybe unique Cocoon resource.

Learning to use it effectively is key to using Cocoon efficiently.

Design your URIs

- Make URIs memorable
- Principle of least surprise
- Use suffixes consistently
- Make sure / goes somewhere sensible
- No accented characters

Fine control of your URI request space is central to Cocoon.

Design your URL request-space early on.

A URI is a contract with the world.

Clean URIs are an important aspect of the overall good design of your site.

We use suffixes for static content, especially where we may need to provide that content in different formats like HTML, PDF, WAP etc. but we tend to avoid using suffixes within the webapp part of projects.

Output relative URLs so Development and Live can work within different URL contexts, as you will probably want to serve this behind a virtualhost via Apache mod_proxy.

Setup the top level

- Components
- Error handling
- Authentication regions
- Mounting sub-sitemaps
- Simple static pipelines

Your top level (project) sitemap should be set up carefully. Don't rely on the default components from the main Cocoon sitemap, these could change when you update Cocoon. Be specific.

Declare the Components you actually use and parameterise them according to how you need to use them in the project.

We declare authentication regions in the top-level Project sitemap, but keep authentication in it's own sitemap. Some recommend using the container for authentication.

If you have simple url patterns for simple static content, that is re-used in many URIs, the top-level Project Sitemap is a good place for these.

Use Sub-SiteMaps

- Keeps sitemaps small
- Break your URI space down into sub-sitemaps
- Separate functionality into subsitemaps

Keep sitemaps as small as reasonably possible, this makes debugging problems much easier, aids maintainability and makes them easier to understand.

The use of sub-sitemaps makes this possible. Keep similar bits of functionality together, so that pipelines can share resources. 12

Use Resources

- <map:resource>
    - Aids re-use
    - Aids readability
    - Aids management
- Use good naming schemes.

Read and understand the Wiki entry : "CleanerSiteMapsThroughResources"

Always use i18n

- Labels sent to the user
    - CForms
    - Messages from FlowScript
    - JX Template
- Localise dates and numbers
- Test UTF-8 cross-platform

Always use i18n for messages or labels sent to the user, regardless of whether you expect there to be translations to other languages.

Never embed user strings in flowscript etc. it makes maintenance a nightmare.

i18n dictionaries can be passed off to those with writing skills i.e. they should not be written by developers, remember what they say about developers not being able to write documentation !!!

We have experienced problems with different Server OSes outputting UTF-8 inconsistently, specifically the 'high' characters. Test thoroughly!!!

Relational Databases

- Use persistence frameworks
    - Apache ORB/OJB
    - Hibernate
    - Spring tools

If you are reading from and/or writing to SQL databases, particularly business objects, the use of persistence frameworks can assist you to separate your logic from the structure of your database tables.

Persistence

```
<hibernate-mapping package="org.project.bean">
        <class name="Image" table="image">
                <meta attribute="class-description">
                        This bean represents the basic information about an Image.
                </meta>
                <meta attribute="implements">Persistable</meta>
                <id name="id" column="id" type="long">
                        <generator class="sequence">
                                <param name="sequence">image_id_seq</param>
                        </generator>
                </id>
                <discriminator column="class" type="string" force="false"/>
                <property name="title" type="string" not-null="true"/>
                <property name="description" type="string"/>
                <many-to-one name="imagecategory" class="ImageCategory" column="image_category_id"/>
        . . .
</hibernate-mapping>
```

A sample mapping between a Bean and an SQL Table This is a snippet from a Hibernate mapping file showing how a Bean may be mapped to SQL Tables.

Persistence

Some sample flowscript to access an SQL Table as a Bean

```
function imageRecord() {
        var factory = cocoon.getComponent(PersistanceFactory.ROLE);
        var session = factory.createSession();
        try {
                var id = new Long(cocoon.parameters["imageid"]);
                var image = ImagePeer.load(session, id);
                if (image != null) {
                        cocoon.sendPage(cocoon.parameters["screen"], { image: image });
                } else {
                        cocoon.sendPage("screen/error", {message: "non.existant.record"});
                }
        } catch (e) {
                cocoon.log.error(e);
                cocoon.sendPage("screen/error", {message: e});
        } finally {
                session.close();
                cocoon.releaseComponent(factory);
        }
}
```

This is a simple function that loads a named entity from SQL as a Bean that is passed to JXTG for display.

Flowscript

- Explicitly pass parameters
- Write business logic in Java
- Flow > Actions > XSP

Explicitly pass parameters from your sitemap to flowscript, rather than asking for arbitrary parameters from request (etc.) in flowscript, as the sitemap will quickly show you what the input contract required by the flowscript should be.

It is possible to use Chained Input Modules to provide defaults for missing request params (eg. Wiki: WorkingWithLocales).

Write complex business logic in Java, call those methods from flowscript (though flowscript can be used to prototype those classes).

Prefer Flow over Actions, and Actions over XSP.

Development

- Use SCM tools
- Replicate the environment
- Use Bugzilla

Choose a development environment that suits your real needs.

Source Code

- Test before committing
- Commit early, commit often
- Don’t commit built material
- Don’t commit local customisation
- Read commit-mails

Use Source Code Management tools like CVS or Subversion etc.

Test before committing, to save holding up other people with your new bugs.

Don't hang around until you've finished the entire project, get discrete functioning units committed as soon as they work.

Always write proper comments in your commits.

Don't check-in generated build results, rebuild them on each build-run.

Don’t commit local.build.properties.

Provide commitmails and viewcvs.cgi on your repos.

Read the commit mails, to keep up with what others on your team are doing !!!

Replicate

- Choose a build pattern
- Make Ant scripts that build into a Cocoon distribution
- Make build scripts that strip your project out of Cocoon
- Allow developers to override defaults values

For a consistent development environment, you want to make a project that patches itself into Cocoon (and cleans itself out) in a predictable way, so that anyone doing work in it is not going to introduce bugs merely due to the way they have the project set up.

If you have databases in your project, have scripts that load a consistent test dataset into them.

Choose one of the build patterns in the Wiki: “YourCocoonBasedProject”, “YourCocoonBasedProjectAnt16”, “ProjectBuilding”.

The main criteria seem to be:
One or many developers.
One or many concurrent projects.

The combination of Ant, X Conf Patch Task and build.properties can be used very effectively for this. local.build.properties can be used for customisation.

Bugzilla

- Track bugs
- Discuss details
- Track change requests

Use tools like Bugzilla in your own projects.

You can use it for notifying the group of the addition and completion of bugs, tracking their progress and discussing their details.

“Bug” can seem like a very rude word to the hardworking developer .... I remember feeling “That is not a bug, it is the result of a poor specification !!”.

Remember, “Bugs” can include visual, behavioural, communicative and logical problems with all aspects of the project, not just the source code. Bugs can include enhancements and feature requests.

Tip: Warn developers to mind their language if the client is subscribed to Bugzilla as well 😉

Going Live

- Minimise log messages
- Disable dangerous features
- Hide Cocoon behind Apache
- Serve static assets via Apache

Turn off debug messages in logkit.xconf.

Disable dangerous components and features, including views, cocoon-reload, cocoon-status, cache-clearance etc.

Don't run cocoon in a publicly-visible way. Close the ports used by your Servlet engine. Run with a front-end proxy (see the Wiki: “ApacheModProxy”).

Get Apache to provide error pages if the Servlet goes down or is under maintenance. This way developers working directly from the Servlet, get to see verbose error messages, while users see something polite.

Write an Ant Task for automating the creation of a special version for going live.

Serve static resources (images, css, javascript, etc.).

Be a good citizen

- Read the manual
- Search the Wiki or Mail Lists
- Ask on the User’s
- List Write your solution in the Wiki
- Report errors in the documentation

If you don't know how to do something, read the documentation on the cocoon site. Search the Wiki and the mailing lists.

If you still cannot find a solution, ask on the users list.

If you can't find documentation for something and then later work out how to do it, be a good citizen and write it up in the Wiki.

If you find mistakes in the code or documentation, submit a patch or at least a bug report. 24

Last Words

- Be liberal in what you accept
- Be strict in what you send
- Separate concerns ruthlessly

When pulling in external content, pass it through Tidy if it is HTML, and clean it through XSLT in an intelligent way if it is XML, so that the mess made by others cannot break your processes.

When handling URIs, apply the “Process of least surprise”, when handling form inputs, provide sensible defaults for missing values where possible.

When sending pages, make sure you validate against the relevant standards. When you output HTML, validate against the WAI standards.

Don’t fight against the SoC that is embedded in Cocoon, embrace it, it is your friend!!! There is a very good reason Cocoon works like that. Even if you are a sole developer prototyping a project, separate!!! Sooner or later, other people will be sharing the work.