## 1. ABSTRACT

Content Packaging is the process in which content that belongs together is bundled into a zip file. A file describing the content (called a manifest) is added so the receiving side can understand what to do with it. For transporting educational content (e.g. e-learning courses) this is common practice, but it is also used elsewhere (e.g. Java `.jar` and `.war` files).
This presentation derives from practical experience building a Content Packaging application, for educational content, using the open source Cocoon framework.
Creating such a Content Package starts with thousands of XML source files and involves quite a lot of transforming, splitting and merging before the end zip file can be assembled. Cocoon proved itself to be a very good platform for operations like this.

## 2. CONTENT PACKAGING

It is a common problem: How do I transport large numbers of related files from one system to the other? How do I keep everything, including all the relations between the separate files, together? How do I make sure the receiving side knows what it gets and what to do with it?
An often used solution for this is packaging everything into a zip file. For example, Java has its .jar and .war files to transport code. Word processors use `.odt` or `.docx`.

In the educational world this solution is called “Content Packaging”. Content Packaging is used for instance to package complete e-learning courses. All kinds of content can be used:

- Texts, like instructions, help pages, etc. Often, of course, in some XML format.
- Questions in a (XML) format that allows automatic presentation, answer checking and feedback.
- Images, sounds and films (assets)
- Application software for presenting the content (often Flash players)

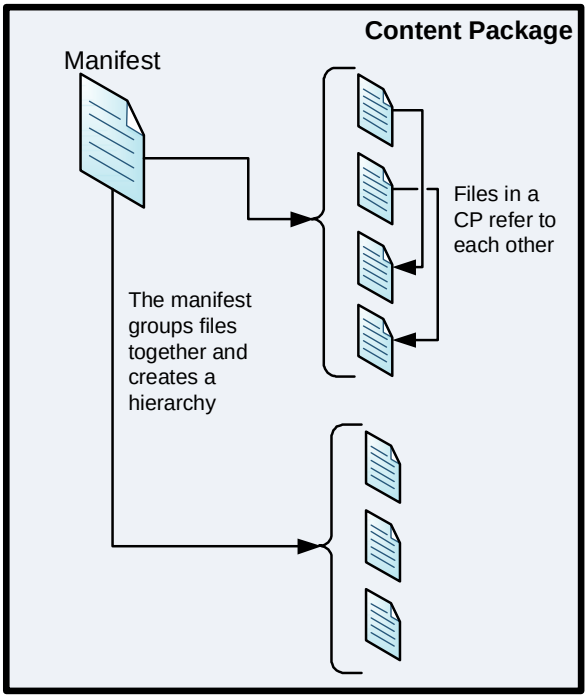
A plethora of more or less complex XML and other standards has evolved around this. Some examples are:

- Content Packaging itself (`http://www.imsglobal.org/content/packaging/`)
- Questions and tests (`http://www.imsglobal.org/question/`)
- Educational metadata (`http://www.imsglobal.org/metadata/`)
- Packaging complete courses (SCORM: `http://www.adlnet.gov` or Common Cartridge: `http://www.imsglobal.org/cc/`)

The net result is that it has become quite difficult to comply with all standards involved. An extra complicating factor is that, more often than not, the input for a content packaging application does not comply to any of the standards. For instance because it comes from a CMS that has its own ideas about XML formats. So before packaging, you usually need to do a lot of transformation, splitting and/or merging of the content.

Accompanying an educational Content Package is always a packing slip, called the manifest. A manifest (which is of course XML) describes which files in the package belong together (are a unit, for instance a single question) and describes the hierarchy. Very important is also that it contains standardized metadata. This allows a receiver to find out where this package belongs and what to with it.

All files in a Content Package have relations with each other. Questions need a player to present themselves, content references assets (images, films, etc.), units have a sequence, sequences are conditional (if you have this question right, you do not need to do that one), etc.
In other words: Files in a Content Package can reference each other but there are also relations on a higher level.




To package all this and comply with all standards and constraint is not exactly easy. Sources are in a wrong format, all references must be there, the package structure (directories, etc.) must be determined, references must be adjusted to this structure, a manifest build, etc. Quite a job that results in rather complicated software. Experience has shown that in a traditional environment (e.g. Java or .NET) this is harder than it may seem at first glance.

## 3. COCOON

Cocoon is an open source Java framework that enables you to process XML in an intuitive way (`cocoon.apache.org`). The fundaments of Cocoon are pipelines: Complex XML transformations, composed of several simple transformations (often XSLT). These pipelines can be combined into complex applications. The XML "streams" through the pipelines as SAX events, so its performance is much better than you might expect.

If this all sounds familiar to you: Cocoon is the predecessor of the XProc standard. I am however not sure whether there is a current XProc implementation that can do what I describe here. Something for future research…

Cocoon was originally developed to create complete websites. That is indeed possible but IMHO there are much better alternatives for doing this. Normal websites are easier to build with, for instance, PHP or ASP.NET. However, the developers of Cocoon created (unwittingly?) a fantastic platform for the manipulation and transformation of large XML volumes, even when the results are not web pages at all.

Cocoon allows you to describe your XML processing on higher, more abstract, level. Doing this in traditional programming languages suffers from an awful lot of low level tasks like opening and closing files, handles, writing intermediate results to disk, buffering, caching, etc. In Cocoon, you "only" have to describe your transformations: Exactly what you need for Content Packaging.

## 4. PACKAGING

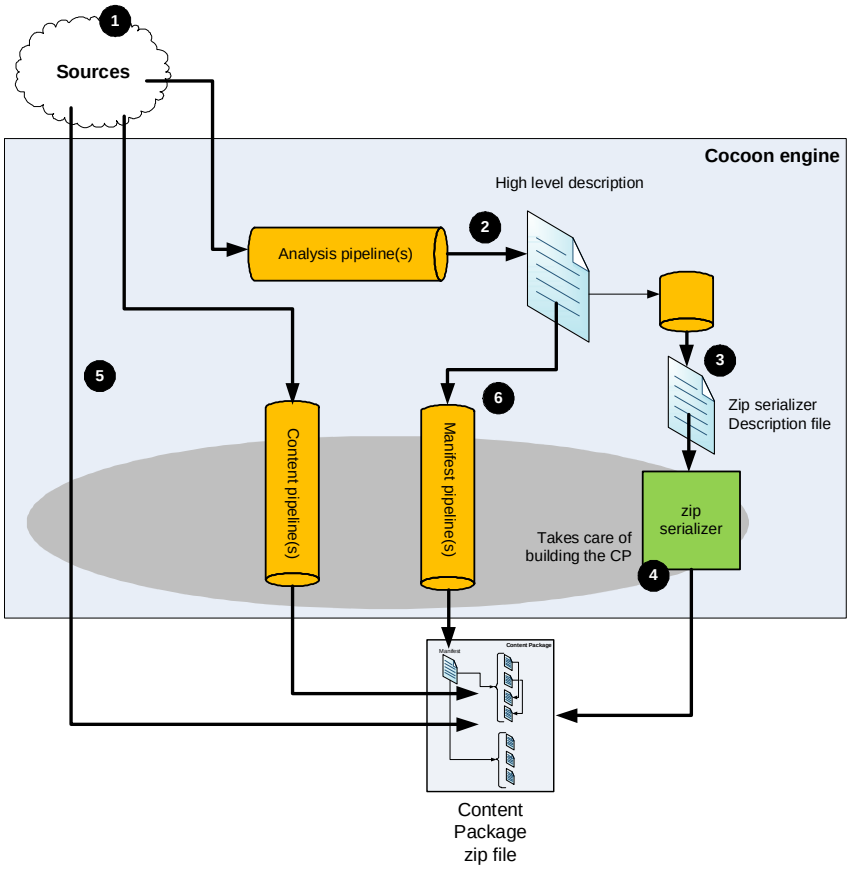
What steps are necessary to create a Content Package:

- You must of course be able to inspect and access the sources from which the package must be build. This could be anything: A CMS, a database, just files on disk, etc. Most common source types are XML documents and assets.
- Our next step is to analyze theses sources. What belongs together, which assets are referenced, which conversions to apply, what will become the metadata, etc. This analysis uses special description data but often also needs to inspect the content itself.

- The results of this analysis are used to assemble the Content Package. For every destination file in the Content Package it is now known how to create it, whether it is from a transformation or just a simple copy. Also its location in the internal directory structure of the Content Package is determined.
- Somewhere along the way we need to create the manifest. Because our analysis describes the Content Package in full, this can be done by transforming the analysis into the right format.
- The result must be delivered as a zip file, with the right name and in the right location.

## 5. USING COCOON

Cocoon's pipeline architecture makes this an excellent tool for the job:




1. Somewhere (lets not get into this) there are source files. Of course, Cocoon must be able to find and process these. It has several options for this, varying from files on disk to databases and webservices.
2. The next step is to analyze these sources into a high-level description of the Content Package. The pipeline for this (or more often: a whole series of interconnected pipelines) results in a XML document containing all the necessary information.
   This analysis document is internal and will never leave the environment "as-is" . Therefore the programmer can design a format for this completely suited to the task. Because the way Cocoon works, it will not be stored anywhere nor will it will be kept after the processing is done. It is fully internal. For debugging purposes it can of course be inspected.

Cocoon has an important component called the "zip serializer". A zip serializer allows you to assemble a zip file based on a description in a XML document. The source for a file in the zip can be anything: another file but also a Cocoon pipeline. This construction allows you apply the necessary transformations on the individual files when constructing the Content Package.

3. Our high level analysis is transformed, again using a pipeline, into an input file for the zip serializer. This is an exact description of the structure and contents of the resulting Content Package. It also describes exactly how to get or create the individual files.

4. This file is passed to the zip serializer.
5. Files that are the result of a conversion operation are created using specific Cocoon pipelines. Other files are copied directly.
6. There is also a pipeline for creating the manifest. This uses the analysis output as its source.

## 6. PRACTICAL EXPERIENCE

This article is based on the experiences developing a Cocoon based Content Packaging application. This application takes its input from several sources and can create about five different flavors of Content Packages. It contains approximately a hundred pipelines and more than 500 XSLT (V2.0) transformations. The resulting Content Packages vary in size from less than one Mb to, in some cases, more than one Gb! It is in production for over a year and frequently used.

Some observations developing this system:

- Using Cocoon for a system like this makes developing it a whole lot of easier. Of course you still have to do the hard work (analyzing, writing code, etc.). However, you can attack the problem on the right level and with the right tools. You are not distracted by trivialities like temporary files. Inserting an XSL style sheet somewhere is as easy as inserting a line of code.
- Using Cocoon is not easy to learn. Like many open source technologies, documentation is scarce, incomplete and sometimes obscure. There are some books but these are obsolete.
- Cocoon can handle large volumes of data easily. Content Packages of over one Gb were created without any problems. XML files of many megabytes were be handled and transformed without complaints.
- However, handling these volumes of data is not always fast. Creating bulky Content Packages can easily take more than an hour. But creating Content packages is (for my customer) not a real-time operation so, although the waiting is annoying, this is not considered a serious problem.
- It is possible to get large performance gains by using tricks like caching and tuning intermediate results. In one exceptional case we even made creating a large Content Package five times faster.

## 7. CONCLUSION

We tried to build a Content packaging application using several different technologies but it was not very successful: Too slow and too cumbersome to maintain. Discovering Cocoon made us very enthusiast: Finally we seemed to have found the right tool for the job. And this of course applies to all problems were large volumes of XML must be handled and/or zip files produced.