SeDiCI
SERVICIO DE DIFUSIÓN
DE LA CREACIÓN INTELECTUAL

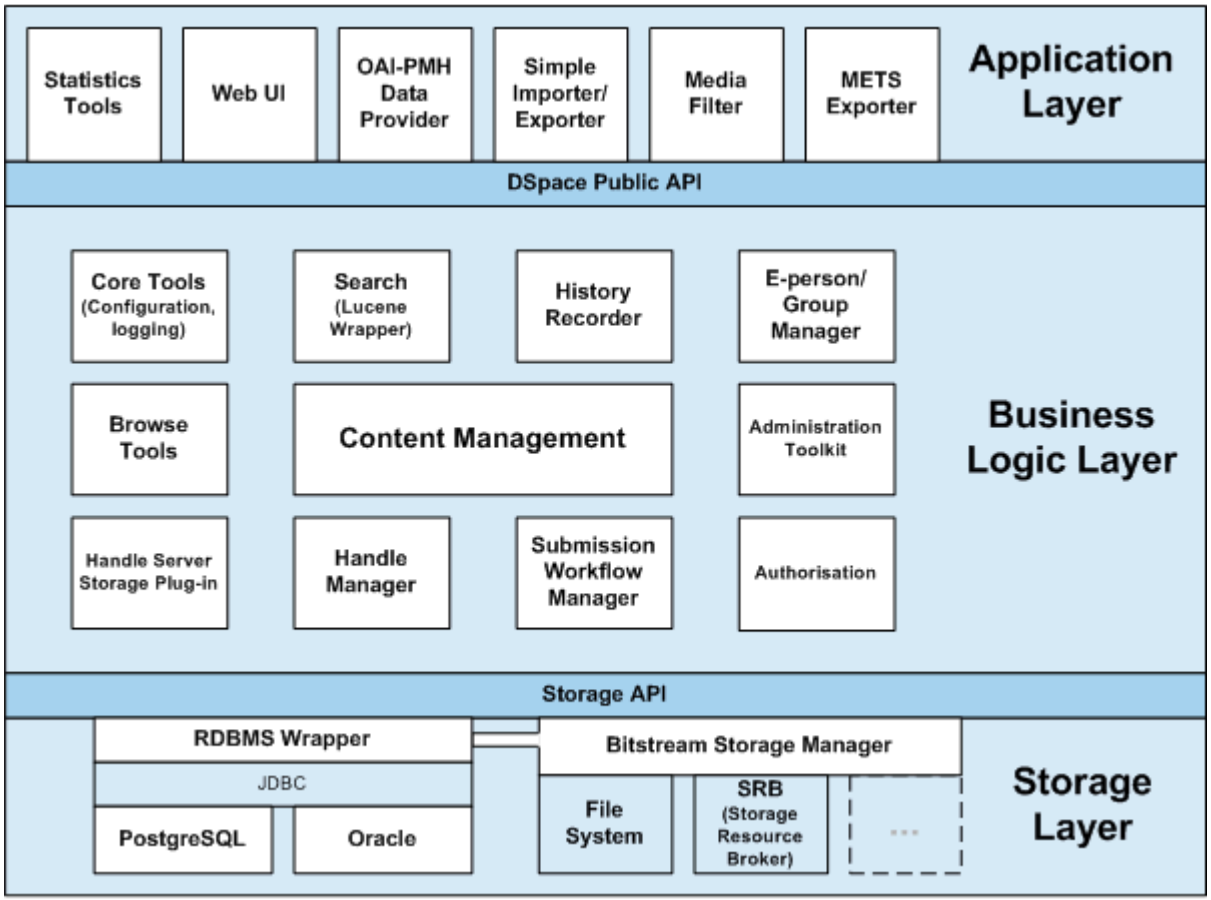
Aspectos técnicos de DSpace

Contenido

- Arquitectura
- Modelo de datos
- Estructura del proyecto
- Maven como gestor de dependencias
- Comando *dspace*
- Introducción a XMLUI
  - DRI, Cocoon, Temas, Aspectos

Arquitectura de DSpace

Arquitectura de DSpace



Fuente: https://wiki.duraspace.org/display/DSDOC18/Architecture

Arquitectura de DSpace

Cada capa ofrece servicios a la capa superior por medio de APIs, y utiliza los servicios de la capa inferior

El código fuente se organiza en paquetes que representan esta arquitectura en capas:

- org.dspace.app — Capa de aplicación
- org.dspace — Capa de lógica del negocio
- org.dspace.storage — Capa de almacenamiento

Arquitectura de DSpace
Capa de almacenamiento

Interacción con la base de datos

- Items y sus metadatos
- Personas y grupos
- Información de autorización
- Trabajos en curso (workflow)
- Indices de búsqueda y exploración

Arquitectura de DSpace
Capa de almacenamiento

Almacenamiento de bitstreams

- Local: el almacenamiento se realiza en el sistema de archivos local al servidor en el que funciona la aplicación

- Storage Resource Broker (SRB): permite tener un sistema de archivos distribuido

Arquitectura de DSpace
Capa de lógica de negocios

Ofrece

- Administración
- Búsqueda
- Exploración
- Gestión de usuarios y grupos
- Autorización
- Carga de documentos
- Workflow
- Handle manager
- Abstracción en Comunidades, Colecciones e Items

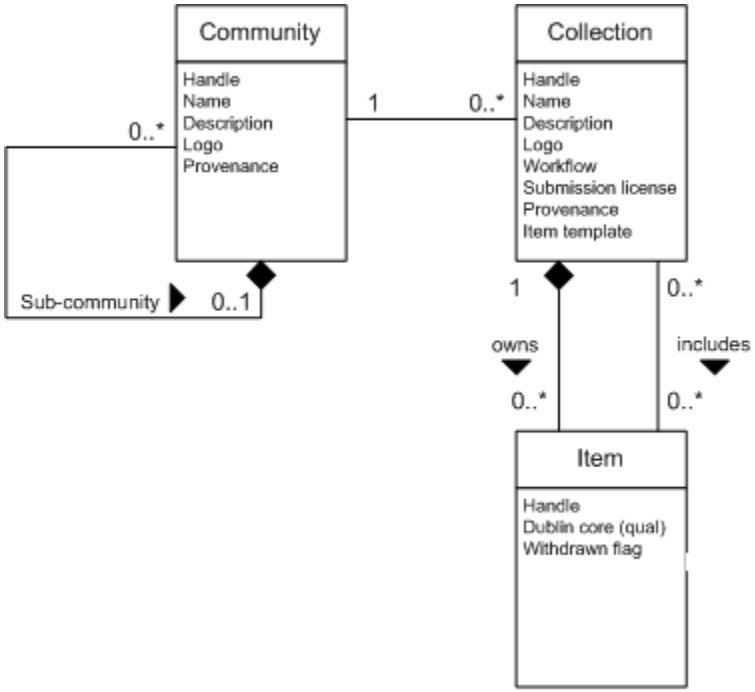
Arquitectura de DSpace
Capa de aplicación

Conjunto de módulos que permiten la interacción con el mundo exterior

- Aplicación web: JSPUI y XMLUI
- OAI-PMH Data Provider
- Estadísticas
- Importar/Exportar
- MediaFilter

Modelo de datos

Modelo de Datos
Organización de contenidos



Fuente: https://wiki.duraspace.org/display/DSDOC18/Functional+Overview

Modelo de datos
Organización de contenidos

Características:

- Las comunidades pueden contener sub-comunidades o colecciones, pero no ítems
- Las colecciones sólo pueden contener ítems
- Un ítem pertenece a una sola colección, pero puede estar asociado a otras colecciones

Modelo de datos
Organización del contenido

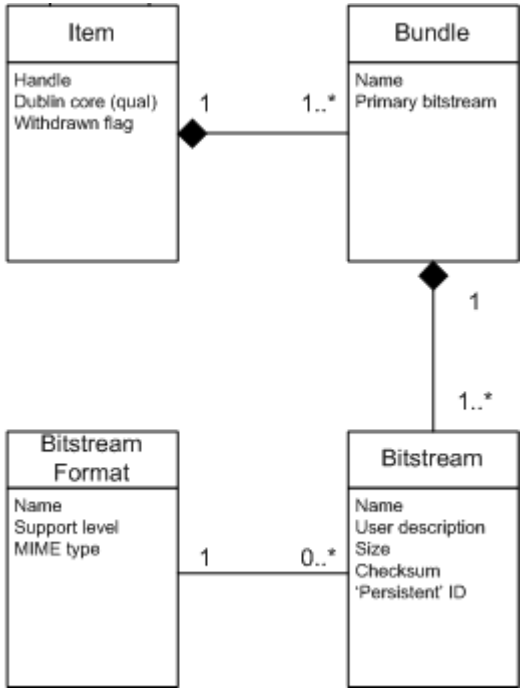
Ventajas:

- Permite establecer restricciones de acceso y modificación específicos para cada nivel
- Permite plantear un esquema navegacional de forma simple

Desventajas:

- Obliga a crear y mantener una estructura estática
- Tiende a generar estructuras redundantes

Modelo de datos Archivos



Fuente: https://wiki.duraspace.org/display/DSDOC18/Functional+Overview

Modelo de datos
Archivos

Los archivos se representan como Bitstreams

Los Bitstream contienen información de validación (checksum), descriptiva, y de preservación (formato y nivel de soporte del formato)

Modelo de datos
Archivos

Los archivos se agrupan en Bundles, según su naturaleza

- ORIGINAL: archivos originales subidos por el usuario
- TEXT: archivos de texto extraído de forma automática a partir de los archivos cargados en el Bundle ORIGINAL
- LICENSE: archivos de licencia asociados al ítem

Estructura del proyecto

Estructura del proyecto

Un proyecto principal (*dspace-parent*) con múltiples subproyectos, en varios niveles:

- Un subproyecto por módulo
- Un subproyecto especial para la generación del instalador: *dspace*

  - dspace-parent
  - dspace-xmlui
  - dspace-xmlui-api
  - dspace-xmlui-webapp

Estructura del proyecto

La relación entre proyectos se mantiene con Maven

- verticalmente entre proyectos y subproyectos, configurando *módulos*

- horizontalmente entre proyectos no emparentados, mediante dependendencias

Estructura del proyecto

Tres tipos de proyectos y subproyectos:

- Agrupamiento de módulos (no genera archivos)
- Librerías (archivos jar)
- Aplicaciones web (archivos war)

Se usan en:

- Directorio de librerías (usado en el classpath)
- Overlay de aplicaciones web

Maven

Gestión de dependencias

Maven

Maven es responsable de la compilación y empaquetado (generación de jar o war)

El ***Reactor*** de Maven es un componente que lista y ordena los proyectos a procesar.

Este orden de procesamiento se determina según las dependencias entre proyectos

Maven
POM

Cada proyecto define un archivo descriptor denominado POM (pom.xml)
En el POM (Project Object Model) se especifica toda la información asociada al proyecto:

- ***Descriptiva***: grupo, id, descripción, autores, etc
- ***Dependencias***: listado de proyectos y sus respectivas versiones
- **Objetivos**: configuración específica para cada etapa de procesamiento (compilación, pre-empaquetado, empaquetado, etc)

Maven
POM

También permite definir:

- **Plugins**: componentes independientes que agregan funciones sobre los proyectos
- **Repositorios**: ubicación de respositorios para descarga de dependencias
- **Perfiles**: permite establecer parámetros de configuración a demanda (según algún criterio de activación)
- **Módulos**: definición de subproyectos
- **Licencia** del proyecto

Maven
Herencia

Los POM heredan algunos elementos de configuración de un POM
Si no se define padre, implícitamente se hereda de ***Super POM***
Esto permite centralizar información como:

- dependencias
- desarrolladores
- plugins habilitados y su configuración
- recursos (define como tratarlos)

Maven
Ejemplo con XMLUI-Webapp

Identificación y descripción del proyecto

```
<groupId>org.dspace</groupId>
<artifactId>dspace-xmlui-webapp</artifactId>
<packaging>war</packaging>
<name>DSpace XML-UI (Manakin) :: Web Application Resources</name>
<url>http://projects.dspace.org/dspace-xmlui/dspace-xmlui-webapp</url>
<description>
    DSpace/Manakin XML-UI Based Web Application using the Cocoon and
    Wing frameworks
</description>
```

Maven
Ejemplo con XMLUI-Webapp

Definición del POM padre

```
<parent>
    <groupId>org.dspace</groupId>
    <artifactId>dspace-xmlui</artifactId>
    <version>1.8.2</version>
    <relativePath>..</relativePath>
</parent>
```

Maven
Ejemplo con XMLUI-Webapp

Listado de dependencias

```
<dependencies>
    <!-- DSpace XMLUI API -->
    <dependency>
        <groupId>org.dspace</groupId>
        <artifactId>dspace-xmlui-api</artifactId>
    </dependency>
</dependencies>
```

Maven
Compilación y empaquetado

Compilar y empaquetar: *mvn package*

- Se analiza el pom.xml ubicado en el directorio de trabajo y se agrega al ***Reactor***
- Si hay módulos definidos, se incluyen en la lista de proyectos del ***Reactor***
- Luego de revisados todos los subproyectos, el ***Reactor*** establece el orden de compilación basado en la definición de las dependencias

Maven
Compilación y empaquetado

- Se ejecuta el empaquetado sobre cada uno de los proyectos del ***Reactor***, en el orden establecido.
- Para cada proyecto se descargan todas las dependencias necesarias desde algún repositorio de Maven
- Para cada proyecto se crea un directorio ***target*** que contiene los archivos compilados y el empaquetado (archivos jar o war)

Maven
Overlay de proyectos

Hay ***overlay*** cuando una aplicación web (se empaqueta como *war*) tiene dependencia de otra aplicación web.

Overlay es la *"mezcla"* de los archivos y directorios entre dos aplicaciones web, respetando un orden de prioridades (configurable)

Maven
Overlay de proyectos

MyWebApp
- index.jsp
- image.png

Depende de

BaseWebApp
- index.jsp
- other-image.png

FinalWebApp
- index.jsp
- image.png
- other-image.png

Maven
Overlay de proyectos

Permite "extender" una aplicación web, creando o redefiniendo sólo los elementos de interés

Maven permite configurar explícitamente el orden en el que se realizará el overlay y qué recursos deben considerarse en el proceso

Comando *dspace*

Comando *dspace*

- Script shell (*#!/bin/sh*)
- Inicia una nueva instancia de la JVM
- Invoca el método *main()* de la clase
- *org.dspace.app.launcher.ScriptLauncher*
- Reenvía los parámetros de la línea de comandos a la clase ScriptLauncher
- Utiliza sus propios parámetros de tamaño de pila y PermGen
- Se incluye el directorio {dspace.dir}/lib en el classpath

Comando *dspace*
Ordenes disponibles

- El primer parámetro que se pasa es la órden que debe ejecutarse (ej.: curate, create-administrator, dsrun, etc.)
- Los siguientes parámetros dependen de la órden invocada
- Las órdenes se configuran desde el archivo launcher.xml, donde se especifican:
  - nombre de la orden
  - clase que debe invocarse

Comando *dspace*
Orden: curate

Ejecuta una "curation task" para realizar algún tipo de análisis o modificación sobre los ítems

Puede aplicarse sobre:

- Repositorio completo
- Una comunidad específica
- Una colección específica
- Un ítem específico

Comando *dspace*
Orden: curate

Ejemplos de curation tasks

- Verificación de links muertos
- Validaciones de integridad de datos
- Análisis de formatos de archivos usados
- Análisis de los archivos en busca de virus

Comando *dspace*
Orden: create-administrator

- Se usa para crear un usuario Administrador en el sistema
- Debe invocarse luego de la instalación para crear el primer usuario en el sistema (no puede haber ningún administrador hasta que no se halla ejecutado esta orden al menos una vez)

Comando dspace
Orden: dsrun

- Ejecuta una clase parametrizable en el momento de la invocación

- La clase a invocar debe definir un método main()

- Permite definir cualquier tipo de clase para luego ejecutarlas desde la línea de comandos

Comando dspace
Orden: embargo-lifter

- Revisa los ítems que tienen fecha de fin de embargo y *levanta el embargo*

- El metadato con la fecha de fin de embargo es configurable

- Debe ejecutarse con cierta frecuencia

Comando dspace
Ordenes para estadísticas

- Se utilizan para recopilar información estadística de acceso, descargas, etc.

- Se realiza un análisis de los logs de DSpace

- Existen múltiples comandos asociados: stat-general, stat-initial, stat-monthly, stat-report-general, stats-utils, etc

Comando dspace
Orden: update-discovery-index

- Actualiza el índice de Apache Solr

- Se utiliza cuando es necesario reindexar todo el repositorio

Introducción a XMLUI

DRI, cocoon, temas y aspectos

Introducción a XMLUI

XMLUI (aka Manakin) es la interfaz de usuario basada en Cocoon Framework

Modelo en capas:

- **Estilos**: look & feel (css e imágenes)
- **Temas**: XSLT para generar XHTML
- **Aspectos**: uso de clases java (o XSLT) para manipular el contenido de una página

Introducción a XMLUI

Modelo en capas

Permite personalizar la aplicación en distintos niveles según:

- Necesidad
- Capacidad técnica

En la mayoría de los casos solo se personaliza la capa de Estilos

Introducción a XMLUI

Tres componentes principales:

- Documentos DRI
- Aspectos
- Temas

Los Aspectos modifican el documento DRI, luego el Tema lo transforma al XHTML final

Introducción a XMLUI
Principales componentes

DRI Schema

- es una representación abstracta de una página del repositorio
- Contiene toda la información (en XML) necesaria para renderizar una página (estructura y metadatos)
- Es la "salida" de los aspectos (en la capa de Aspectos) y la "entrada" para el Tema (en la capa de Temas)

Introducción a XMLUI
Principales componentes

DRI Schema: estructura bien definida

- **BODY**: cuerpo de la página a renderizar. Los temas procesan el body para generar el XHTML final
- **OPTIONS**: opciones de menú disponibles. Permite que los aspectos agreguen opciones de menú. Parte del tema también procesa esta sección
- **META**: meta-información sobre la página
- **userMeta**: información sobre el usuario logueado (o anónimo)
- **pageMeta**: información sobre la página a renderizar (hojas de estilos, javascripts, variables de control)
- **repositoryMeta**: información sobre el repositorio

Introducción a XMLUI
Principales componentes

Aspectos

- Pueden ser clases Java o archivos XSLT
- Los aspectos son independientes entre si
- se usan para manipular el DRI
- Reciben un documento DRI como entrada y generan un documento DRI como salida
- Tienen acceso a la API de DSpace y otras dependencias
- Son el mecanismo pensado para agregar funcionalidad a la interfaz de usuario

Introducción a XMLUI
Principales componentes

Temas

- Responsable de transformar el DRI final a un XHTML para enviarlo al cliente
- Permiten cambiar la estructura de la interfaz de usuario
- Incluye los CSS y JavaScript necesarios para el look&feel deseado
- Se puede aplicar sobre todo el sitio o sobre una URL específica

Introducción a XMLUI
Configuración: xmlui.xconf

Configuración de aspectos

```
<aspect name="Administration" path="resource://aspects/Administrative/" />
```

Se indica:

- *name*: nombre asociado al aspecto
- *path*: ubicación del archivo sitemap.xmap del aspecto
- resource:// indica que se encuentra en el classpath

El orden en el que los aspectos aparecen en el archivo determina el orden de ejecución

Introducción a XMLUI
Configuración: xmlui.xconf

Configuración de temas

```
<theme name="Test Theme 1" handle="123456789/1" path="theme1/"/>
<theme name="Test Theme 2" regex="community-list" path="theme2/"/>
<theme name="Atmire Mirage Theme" regex=".*" path="Mirage/" />
```

- Los parámetros handle y regex permiten aplicar un tema específica sobre algunas URLs
- Cuando regex vale ".*" indica que se aplica a todas las URL
- Si más de un tema aplican, se considera el que aparece primero
- path indica el directorio del tema

Introducción a XMLUI
Cocoon

- Framework para la construcción de aplicaciones web
- Se basa en la ejecución componentes individuales organizados en secuencia (pipelines)
- Existen distintos tipos de **componentes** y su uso debe seguir ciertas reglas
- La configuración de los pipelines se realiza sobre XML (archivos **sitemap.xmap**)

Introducción a XMLUI
Cocoon

- En el root de la webapp está el root **sitemap.xmap**: el punto de entrada para todos los requests
- Cada sitemap.xmap tiene la capacidad de "montar" otros archivos sitemap.xmap. Esto permite la modularización
- Los archivos sitemaps definen los distintos posibles pipelines a ejecutar para un request determinado

Introducción a XMLUI Cocoon

- De ejecución diferida
  - Generator
  - Transformers
  - Serializers
- De ejecución inmediata
  - Matchers
  - Selectors
  - Readers
  - Actions

Introducción a XMLUI
Cocoon

Secuencia básica de procesamiento:

- Se **GENERA** un documento de base
- Se aplican **TRANSFORMACIONES** sobre el documento
- Se **SERIALIZA** el documento luego de todas las transformaciones

Esta secuencia presenta tres componentes básicos: **generators**, **transformers**, **serializers**

Introducción a XMLUI
Cocoon

Ejecución diferida

- Se encolan para ser procesados al finalizar el análisis del sitemap

Se valida que se respete el orden:

1. Un Generator
2. Múltiples Transformers
3. Un Serializer

Introducción a XMLUI
Cocoon

Generators:

- Generan un **documento base** (temporal) sobre el cual se trabajará en el resto del pipeline.
- Solo puede haber **UN** generator por pipieline.
- El caso más frecuente es la generación del documento **DRI vacío** (sólo contiene los elementos de nivel superior)

Introducción a XMLUI
Cocoon

Transformers:

- Modifican el DRI de un documento
- Los transformer se ejecutan según el orden de aparición en el pipeline
- Cada transformer realiza una "pequeña" modificación sobre el DRI
- Permite la modularización y separación de *concerns*

Introducción a XMLUI
Cocoon

Serializers:

- Hasta ahora todo el documento DRI temporal es en realidad una estructura en memoria
- El serializer genera una representación concreta.
- Serializa los objetos de memoria a una representación en texto
- La serialización más frecuente es la generación del documento DRI en XML

Introducción a XMLUI
Cocoon

Serializers:

- Es el último elemento de un pipeline

- El procesamiento del pipeline finaliza cuando se encuentra un serializer

- Todos los componentes que aparecen luego de un serializer son descartados

Introducción a XMLUI Cocoon

Ejecución inmediata

Se ejecutan al momento de encontrarse en el sitemap

- Modifican el curso de evaluación del sitemap

- Permiten incluir componentes al pipeline de forma dinámica, según distintos criterios

Introducción a XMLUI Cocoon

Matchers

- Aplican algún tipo de matching sobre algún parámetro
- Permiten decidir si una parte del sitemap debe procesarse u omitirse
- Si el Matcher determina que HAY matching, su contenido se procesa
- Pueden devolver datos para ser usados dentro del bloque del matcher
- Múltiples implementaciones

Introducción a XMLUI Cocoon

Matchers

ejemplo WildcardURIMatcher

```
<map:match pattern="feed/*/**">
    <map:generate type="DSpaceFeedGenerator">
      <map:parameter name="feedFormat" value="{1}"/>
      <map:parameter name="handle" value="{2}"/>
    </map:generate>
    <map:serialize type="xml"/>
</map:match>
```

Introducción a XMLUI
Cocoon

Selectors

- Permite incluir dentro del sitemap uno de entre varios grupos de instrucciones, según alguna condición
- Es análogo a una instrucción case de un lenguaje de programación
- Permite definir un caso por default
- Múltiples implementaciones

Introducción a XMLUI Cocoon

Selectors

```
<map:select type="request-parameter">
    <map:parameter name="parameter-name" value="format"/>
    <map:when test="rss">
        <map:serialize type="rss"/>
    </map:when>
    <map:when test="atom">
        <map:serialize type="atom"/>
    </map:when>
    <map:otherwise>
        <map:serialize type="html"/>
    </map:otherwise>
</map:select>
```

Introducción a XMLUI Cocoon

Readers

- Se utiliza para retornar archivos que no requieren procesamiento (imágenes, archivos binarios, etc.)
- Cumple los roles de generator, transformer y serializer
- Cuando se encuentra un Reader, se descarta todo lo encontrado hasta el momento y se finaliza el procesamiento del sitemap

Introducción a XMLUI Cocoon

Readers

```
<map:match pattern="bitstream/handle/*/*/**">
    <map:match type="request-parameter" pattern="sequence">
        <map:select type="browser">
          <map:when test="spider">
            <map:read type="BitstreamReader">
                <map:parameter name="userAgent" value="spider"/>
                <map:parameter name="handle" value="{../1}/{../2}"/>
                <map:parameter name="name" value="{../3}"/>
                <map:parameter name="sequence" value="{1}"/>
            </map:read>
          </map:when>
          .....
```

Introducción a XMLUI Cocoon

Actions

- Realizan acciones que indirectamente pueden afectar el comportamiento de otros componentes (por ejemplo, seteando alguna variable en la sesión)
- No modifican el DRI ni generan derivaciones en el procesamiento del sitemap
- Puede retornar valores para ser usados como parámetros en otros componentes

Introducción a XMLUI Cocoon

Actions

```
<map:act type="locale">
   <map:transform type="i18n">
      <map:parameter name="locale" value="{locale}"/>
   </map:transform>
</map:act>
```

Introducción a XMLUI Cocoon

Otros elementos útiles

mount: "monta" otro sitemap para su procesamiento. Permite dividir un sitemap muy grande en porciones más chicas, montando cada una según algún criterio (usando un matcher o un selector)

handle-errors: cuando se genera una excepción sin manejar en cualquier parte del código invocado como parte del procesamiento del pipeline, ésta es capturada por este elemento, permitiendo mostrar algún error amigable al usuario

Introducción a XMLUI
Cocoon

El uso de los componentes (matchers, selectors, actions, readers, generators, transformers y serializers) permite generar pipelines dinámicos en función de distintos parámetros

Listado de todos los componentes que Cocoon incluye en su distribución:
http://cocoon.apache.org/2.2/1347_1_1.html

Introducción a XMLUI
Procesamiento de un request

- El request es atendido y se comienza la evaluación del root sitemap (el punto de entrada para todos los requests)
- Si se trata de la descarga de algún documento, el request es procesado por un Reader en el root sitemap y termina el procesamiento
- Si es una página, el root sitemap monta el sitemap que gestiona los temas (/themes/themes.xmap)

Introducción a XMLUI
Procesamiento de un request

- Se determina qué tema debe aplicarse y monta el sitemap.xmap correspondiente al mismo
- La primer instrucción en este nuevo pipeline es <map:generate type="file" src="cocoon://DRI/{1}"/>
  El prefijo cocoon: indica que se trata de un request interno
- El procesamiento de este sitemap termina cuando se encuentra el serializer adecuado

Introducción a XMLUI
Procesamiento de un request

- Se comienza la ejecución del *pipeline primario* armado, el cual comienza con el **generate**
- Ese **generate** dispara el request interno, el cual es manejado nuevamente desde el root sitemap
- Dado que la URL de dicho request comienza con "DRI/", se monta el sitemap para la gestión de aspectos (aspects/aspects.xmap)

Introducción a XMLUI
Procesamiento de un request

- Se determinan los aspectos a ejecutar, en el orden en el que fueron configurados
- Cada aspecto en la lista se carga y se procesa su sitemap.xmap. Esto genera un pipeline independiente para cada aspecto
- Se ejecutan los pipelines de cada aspecto, con lo que se manipula el DRI
- Así finaliza la ejecución del generate inicial

Introducción a XMLUI
Procesamiento de un request

- Continúa la ejecución del pipeline primario, ahora con el documento DRI generado luego de la ejecución de todos los aspectos
- Se aplican los transformers correspondientes
  - inclusión de archivos css y javascripts
  - **ejecución del XSLT principal del tema, a fin de generar el XHTML final**
- Finalmente, se serializa a XHTML y se retorna el resultado al navegador del usuario