Indice

1. DESCRIZIONE DEL PROGETTO

Il progetto consiste nello studio delle funzionalità di base della framework Cocoon e nella creazione di un sito internet utilizzando questa nuova tecnologia.
Cocoon è una framework XML con la particolarità di essere basato sulla separazione dei concetti (contenuti, logica e stile) ed essendo stato progettato come un motore astratto, è in grado di connettersi a qualsiasi applicazione. Una configurazione centralizzata e una cache sofisticata permettono di creare, sviluppare e mantenere delle applicazioni basate sulla solida e flessibile tecnologia XML.

1.1. PERCHE' COCOON

Supponiamo di voler creare un sito web che abbia una versione in inglese, una in HTML3.2 per i palmari e una in WML per questa gamma di cellulari in rapida espansione. Nel caso si volessero apportare delle modifiche al sito, l'alberatura è destinata a raddoppiare e anche la mole fisica dei documenti ospitati dal server.
Le problematiche relative all'aggiornamento di questo sito sono riassumibili generalmente in questi due aspetti principali:

- l'aggiunta di un documento all'interno di un sito non si limita alla creazione dello stesso ma comporta modifiche in cascata all'interno di altri documenti: ad esempio le altre pagine della sezione, la sitemap, la home se si vuole dare maggiore visibilità al nuovo materiale.
- quanto più le modifiche si trovano alla base dell'alberatura, maggiori saranno le modifiche da apportare: l'aggiunta di una nuova sezione di primo livello richiederà probabilmente l'aggiunta del link relativo in ogni pagina del sito.

Grandi pubblicazioni web quali portali e siti multilingue, sono destinati quindi a diventare meno facilmente gestibili quanto più aumentano i contenuti, le lingue, le versioni.
Traducendo queste esperienze in ottica futura, in cui si prevede una utenza di lingua sempre più eterogenea e distribuita su di uno spettro di device molto più ampio appare chiaro come le metodologie di aggiornamento siano cruciali.

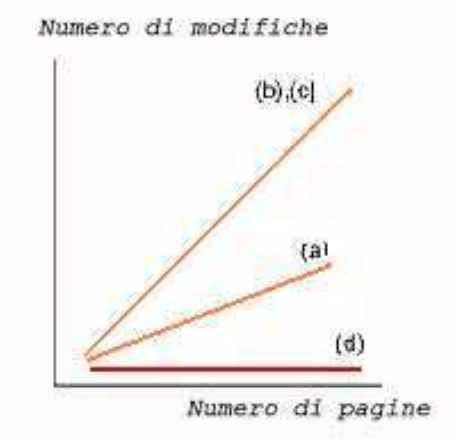
Per quantificare i vantaggi del ricorso a Cocoon nella gestione di un prodotto web pensiamo ad una realizzazione ipotetica di medie dimensioni, poniamo cinque sezioni contenenti ognuna altrettanti documenti oltre naturalmente l'homepage, la pagina dei credits e l'immancabile sitemap per un totale di ventotto documenti.

La tabella che segue riporta le operazioni di aggiornamento più comuni nella gestione di un sito e accanto i relativi costi in termini di numero di pagine create/modificate nel caso di una gestione "classica" dei contenuti oppure secondo l'approccio previsto da Cocoon.

Di seguito alla tabella troviamo un grafico qualitativo dei costi di aggiornamento al variare del numero complessivo di pagine del sito. Sugli assi non sono riportati valori di riferimento in quanto è la pendenza delle rette (ovvero la proporzionalità tra dimensioni del sito e numero di modifiche necessarie) la radice delle problematiche di aggiornamento.

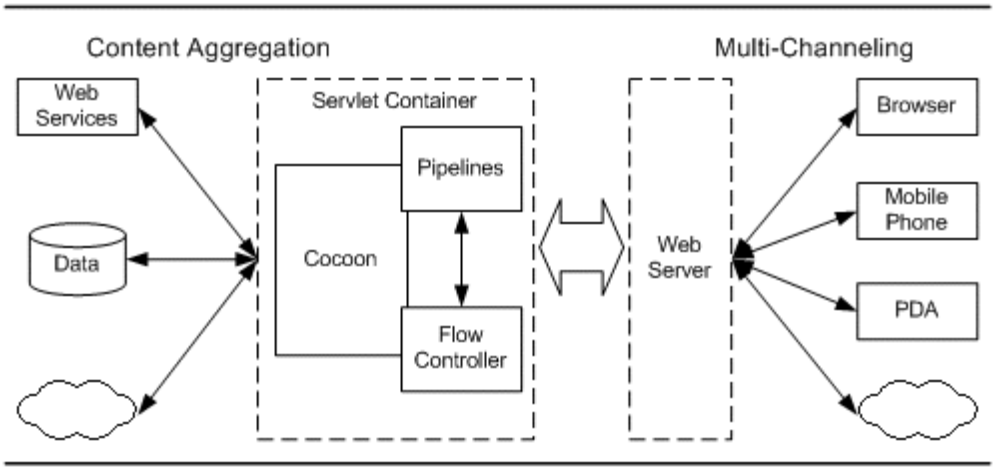
Operazione	Gestione classica	Gestione Cocoon
Creazione di una nuova pagina	**7** Produzione della pagina nel suo insieme (navigazione+contenuti),aggiunta del relativo link di tutte le pagine della sezione corrispondente e modifica della sitemap (a)	**2** Produzione dei soli contenuti e aggiunta di un elemento <pagina>...</pagina> in site.xml (d)
Creazione di una nuova sezione	**28** Aggiunta del link alla nuova sezione in tutte le pagine del sito (b)	**1** Aggiunta di un elemento <sezione>...</sezione> in site.xml (d)
Restyling generale del sito	**28** Modifica del codice html in tutte le pagine (c)	**1** Modifica delle sezioni di html in site.xsl (d)

Qualora si volessero realizzare versioni successive del sito per device differenti è possibile riutilizzare i contenuti e scrivere semplicemente un secondo file "sito-wml.xsl" referenziandolo nell'xml originale. In questo secondo template si dovrebbe spogliare i contenuti dei tag validi in HTML ma non in WML.



Appare evidente come i costi di aggiornamento della tabella precedente raddoppino nel caso di una gestione classica dei contenuti mentre rimangano invariati utilizzando il sistema di pubblicazione presentato.

La piattaforma presentata consente quindi di realizzare con un'unica coppia di file xml/xsl tutto il sito. Al "contenuto" in senso stretto di ogni pagina dovrà corrispondere comunque un file singolo o una aggregazione di contenuti (come mostrato nella figura sottostante), che tuttavia non andranno modificati in caso di aggiornamenti, essendo stati isolati da tutto ciò che è struttura, navigazione e, in generale, dalle parti dinamiche del prodotto.



Informazioni generali di Cocoon

- Dispone di un avanzato Control Flow: un flusso di pagine basato sulle *continuation* che semplifica la complessità del processo di request/response. Questo processo è separato da stili e contenuti.
- E' un software open source.
- Integra varie tecnologie.
- Può essere usato per creare applicazioni web dinamiche multi-channel acquisendo dati da diverse datasources e trasformandoli in diversi formati.
- Si possono creare contenuti statici separando stile e contenuti.
- Si possono creare applicazioni web avanzate grazie all'integrazione J2EE (sempre con la separazione tra dati, stili e logica
- Si può sviluppare un portale usando la framework Cocoon Portal
- Supporta multiple clients, layouts e languages (i18n) senza duplicazioni di codice
- Può essere integrato con una applicazione web esistente
- Può essere usato come base per l'Enterprise Application Integration (EAI)
- Può essere usato come base per un Content Management System (CMS) (per esempio Apache Lenya)
- Può essere usato per produrre contenuti visualizzabili in telefoni cellulari
- Può coesistere e cooperare con soluzioni J2EE esistenti (EJB, JMS, ...)
- Può essere integrato con un motore di ricerca (usando Lucene)
- E' integrato con Apache Axis (per la pubblicazione del proprio Webservice)
- Ha il supporto Java Mail
- Ha il supporto multilingue I18n
- Facilmente estendibile

1.2. CARATTERISTICHE SITO CREATO

Per iniziare lo studio della piattaforma Cocoon, si è deciso di realizzare un sito relativo ad un'associazione di macellai padovani (I Maestri del Gusto). L'obiettivo del sito è di far conoscere l'esistenza di questa associazione e di promuovere i loro prodotti, specialmente i pronti a cuocere.

Il target del sito è composto dai clienti delle varie macellerie che possono trovare suggerimenti su come preparare i prodotti acquistati ed avere nuove idee per gli acquisti futuri.

I contenuti principali sono:

- Storia della formazione della società
- Elenco soci e rispettive macellerie
- Sezione contenente le specialità del gruppo di macellai
- Sezione in cui i clienti possono inviare la loro ricetta
- Sezione in cui l'amministratore può cancellare o aggiungere una specialità

La pagina è formata da tre aree principali: un banner, un menù laterale e una sezione centrale per pubblicare i contenuti.
Il banner è composto dal logo dell'associazione sulla sinistra e da una componente grafica nella restante parte. Il menù laterale sinistro comprende quattro aree principali: Chi siamo, Specialità, Informazioni generali e Invia la tua ricetta. Quando si naviga all'interno di una di queste aree il menù collassa mostrando alcune sottovoci, che fanno parte di quella sezione, allineate però a destra. L'area centrale contenente le informazioni da pubblicare comprende il resto della pagina.
Tutte le informazioni sull'ampiezza delle aree, bordi, colori, stili, ecc… sono contenute nel foglio di stile "gusto.css":

```
/************************
MAESTRI DEL GUSTO - CSS
************************/
/*stili BODY*/
body { font-family:Georgia,Times,serif; font-size:0.63em;
background-color:#900; color:#333; margin:0px; padding:0px; line-
height:1.3; text-align:center; }
html, body { height:100%; width:100%; }
img { border:0px; }

/*stili testi*/
h1 { font-family:Georgia,Times,serif; font-weight:normal; font-
size:2.2em; padding:0px; margin:0px; padding-bottom:1.5em;
color:#900; padding-right:10px; padding-left:10px; }
h2 { font-size:1.3em; font-weight:bold; padding:0px; margin:0px;
padding-top:0.7em; padding-right:10px; padding-left:10px; }
p { font-size:1.1em; padding:0px; margin:0px; padding-
bottom:0.7em; padding-right:10px; padding-left:10px; }
a { text-decoration:underline; color:#333;}
a:hover { text-decoration:underline; color:#900}
ol { padding:0px; margin:0px;}
ul { list-style-type:none; padding:0px; margin:0px;}
li { padding-bottom:0.7em;}
```

```
/*stili layer*/
DIV.mainbox {position:relative; margin:auto; padding:0px;
width:726px; clear:left; background-color:#FDF7E8; text-
align:left; background-color:#FDF7E8; height:100%; min-
height:100%; height:auto !important; height:100%; }
DIV.leftbox { position:absolute; width:175px; clear:none;
margin:0px; padding:0px; background-color:#E1D8CB; }
DIV.contentbox { position:absolute; width:551px; float:right;
clear:none; margin:0px; padding:0px; padding-left:175px;
background-color:#FDF7E8; height:100%; min-height:100%;
height:auto !important; height:100%; }

/*stili vari*/
ul.menu1 { padding:0px; margin:0px; padding-bottom:0.7em; padding-
right:10px; padding-left:10px; text-align:right; line-height:1.0;
}
li.menu1 { list-style:none; padding-bottom:0.7em; }
a.menu1 { font-size:1.2em; text-decoration: none; color:#333; }
a.menu1:hover { color:#900; text-decoration: underline;}
p.foot { font-size:0.9em; padding-right:10px; padding-left:10px; }
a.foot { text-decoration: underline; }
a.foot:hover { text-decoration: underline; }
```

Ecco un esempio della grafica della pagina:

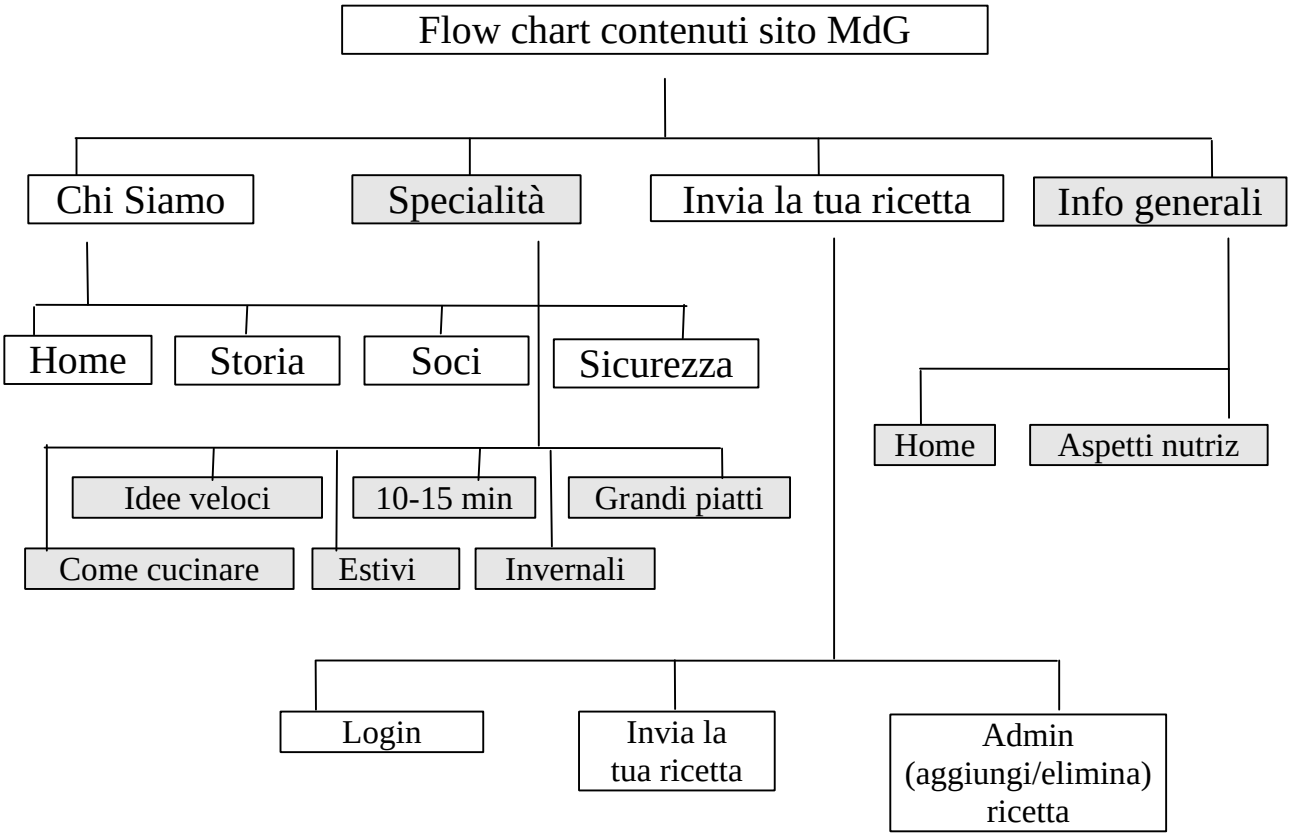
1.3. Struttura

Il sito è diviso in quattro aree principali. L'area *Chi Siamo* contiene informazioni relative alla fondazione, alla storia e ai componenti dell'associazione. Qui ogni socio ha a disposizione una pagina dedicata alle specialità del proprio negozio e le indicazioni necessarie per raggiungerlo.

L'area *Specialità* contiene cinque sottosezioni in cui sono divise tutte le specialità comuni a tutti i soci. Questa suddivisione è creata in base alla stagionalità della ricetta (Estiva o Invernale) e al tempo necessario per prepararla (Idee veloci, 10-15 minuti e Grandi piatti).

L'area *Informazioni* generali contiene le informazioni relative agli aspetti nutrizionali della carne ed eventi organizzati dall'associazione.

L'area *Invia la tua ricetta* contiene una sezione in cui l'utente può inserire una sua ricetta che verrà spedita via e-mail all'amministratore, il quale deciderà se pubblicarla o meno. Sempre in quest'area l'amministratore potrà accedere ad un'area protetta e aggiornare il database delle ricette.



2. PIATTAFORMA UTILIZZATA: **COCOON**

Cocoon è una framework per lo sviluppo di applicazioni web ideata con l'obiettivo di separare stili, logica e contenuti. La framework, completamente Open Source, è scritta in java, ma non è necessario saper programmare in java per lavorare con Cocoon. Sono invece necessarie alcune conoscenze sull'XML e sull'XSL per cominciare a costruire le prime applicazioni con Cocoon.

“Il progetto Cocoon ambisce a cambiare il modo in cui vengono create, rappresentate e servite le informazioni sul web. L'innovativo paradigma di Cocoon è basato sulla considerazione che contenuto, stile e logica di un documento sono create da più gruppi di lavoro. Cocoon punta ad una completa separazione dei tre livelli”. Con queste parole Stefano Mazzocchi, l’ideatore di Cocoon, descrive uno dei più interessanti progetti sviluppati dalla Apache Software Foundation (ASF) negli ultimi anni.

Cocoon è una architettura più che uno strumento ed è quindi sbagliato confrontarlo sia con linguaggi di programmazione (PHP, Perl), sia con soluzioni di generazione del contenuto (JSP, ASP, ColdFusion). Cocoon è scritto in Java per necessità tecnologiche: molte delle librerie XML erano scritte in Java quando il progetto partì nel 1998. Tuttavia, i concetti architetturali possono essere implementati in qualsiasi linguaggio di programmazione, cosa che è già stata fatta dal progetto AxKit (www.axkit.org) in Perl e dal progetto Eclypse in Pyton. Ad ogni modo, l'architettura di Cocoon è totalmente astratta dalla generazione del contenuto XML: è infatti possibile generare il contenuto XML usando una qualsiasi tecnologia web esistente e processarlo successivamente con Cocoon, senza dover scrivere nulla in Java.

La futura direzione del web publishing è aggregazione di contenuti (dai portali alle web application) e Cocoon non vuole introdurre nuove tecnologie dove è possibile utilizzare il know-how precedentemente esistente. Vuole solamente fornire una struttura flessibile ma al tempo stesso solida delle funzionalità tipiche di cui il publishing XML necessita.

Ne è un esempio la pipeline, elemento fondamentale delle applicazioni di Cocoon: con una serie di componenti e un potente mezzo per connetterli insieme, la maggior parte del lavoro può essere fatto senza dover scrivere codici complessi, ma sarà sufficiente utilizzare quello che esiste già, con un meccanismo simile al gioco dei Lego.

2.1 DALL'HTML ALL'XML

L'HTML (Hypertext Markup Language) è considerata la base del World Wide Web. Questo linguaggio consente infatti di creare in maniera standardizzata pagine di informazioni formattate in grado di raggiungere, tramite Internet, un numero di utenti in costante aumento. Insieme al protocollo HTTP (Hypertext Transport Protocol), l'HTML ha rivoluzionato il modo in cui le persone inviano e ricevono informazioni, ma lo scopo principale per cui è stato realizzato è la *visualizzazione* dei dati. Per questo motivo, l'HTML prende in considerazione soprattutto il modo in cui le informazioni vengono presentate e non il tipo o la struttura di tali informazioni.
Così, con l'HTML, si è creato uno strumento volto a definire come visualizzare una pagina, perdendo le informazioni relative all'analisi dei dati contenuti.

Esempio pagina HTML:

```
<html>
  <body>
   <p> Ciao, sono una pagina HTML.</p>
   <p align="center">Scritta da Beghin Daniele</p>
  </body>
 </html>
```

Il browser ricava da questa pagina le seguenti informazioni:

- Si tratta di una pagina HTML
- La pagina ha un body
- Nel body ci sono due paragrafi
- Il primo contiene la frase "Ciao, sono una pagina HTML."
- Il secondo invece dice: "Scritta da Beghin Daniele"

Supponiamo di chiedere a degli stranieri, che non capiscono l'italiano, informazioni riguardo chi ha scritto questa pagina.
Loro, esattamente come il web browser, non saranno in grado di rispondere (non sono in grado di effettuare un'analisi semantica). La sola cosa che il browser può fare è disegnare la pagina con quei contenuti nello schermo, in quanto questa è la cosa per cui è stato programmato. In altre parole, la capacita semantica del browser è limitata al visualizzare delle strutture con dei contenuti e pochi altri concetti (come i link).

2.2 L’analisi semantica

Supponiamo di ricevere questa pagina:

```
<pagina>
 <autore>sflkjoiuer</autore>
 <contenuto>
  <paragrafo>sofikdjflksj</paragrafo>
 </contenuto>
</pagina>
```

Ora alla domanda “Chi ha scritto questa pagina?” Tutti risponderanno "sflkjoiuer".
Se però poi riceviamo:

```
<dlkj>
 <ruijfl>sofikdjflksj</ruijfl>
 <wijlkjf>
   <oamkfkj>sflkjoiuer</oamkfkj>
 </wijlkjf>
</dlkj>
```

Alla domanda “Chi ha scritto la pagina?” Ora possiamo supporre che la struttura sia la stessa nei due documenti, ma come possiamo essere certi che ci sia la stessa informazione semantica? I due esempi precedenti sono entrambi documenti XML.
A questo punto potremmo chiederci che aiuto ci dà l’analisi semantica se, da quanto visto finora, invece di semplificare le cose le complica ulteriormente.

2.3. L’ XML

L’Extensible Markup Language (XML) è un metalinguaggio che permette di creare dei linguaggi personalizzati di markup. La necessità di espandere le capacità di HTML ha spinto i produttori di browser a introdurre nuovi marcatori nella sintassi, rendendola a tutti gli effetti proprietaria e non più standard. Da ciò segue che una pagina HTML che sfrutti marcatori proprietari non può essere visualizzata correttamente se non con il browser adatto, con le ovvie conseguenze che ne derivano.
L’XML è un linguaggio di markup aperto e basato su testo che fornisce informazioni di tipo strutturale e semantico relative ai dati veri e propri. Questi "dati sui dati", o *metadati*, offrono un contesto aggiuntivo all’applicazione che utilizza i dati e consente un nuovo livello di gestione e manipolazione delle informazioni basate su Web.

Su questo modello fu ideato SGML (Standard Generalized Markup Language), lo standard internazionale per la descrizione della struttura e del contenuto di documenti elettronici di qualsiasi tipo.
L'XML quindi permette a gruppi di persone, o ad organizzazioni, di creare il proprio linguaggio di markup specifico per il tipo di informazione che trattano; per molte applicazioni e per diversi settori.

SINTASSI

Le 3 regole principali sono:

1. Non è necessario che il nome del tag si riferisca ad una struttura definita.
   - L'informazione è strutturata con nomi scelti dal progettatore.

2. Ogni elemento deve essere composto da un tag di inizio e un tag di fine
   - I tag html
 oppure <img src="..."> diventano in xml

 o
</br> e o <img src="...">...</img>;

3. Gli elementi devono essere nidificati correttamente
   - ... è generalmente accettato in html mentre
    ... in xml non lo è e produrrà un errore;

Esempio:

Pagina html:

```
<html>
  <body>
   <h1> Apache Cocoon </h1><br>
   <p align="center">Written by Daniele</p>
   <img src="immagine.gif">
  </body>
 </html>
```

Pagina Xml:

```
<g4>
  <titolo> Apache Cocoon </titolo><br/>
  <contenuti>
    <testo> Written by Daniele </testo>
    <img src="imagine.gif"/>
  </contenuti>
</g4>
```

Apertura e chiusura dei tag

Nel codice HTML un elemento contiene in genere sia tag di apertura che di chiusura. A differenza dell'HTML, l'XML richiede che un tag di chiusura venga utilizzato per ogni elemento.
Si consideri ad esempio l'elemento HTML Paragraph che dovrebbe in genere includere un tag di apertura, il contenuto e un tag di chiusura come mostrato di seguito:

```
<P>Questo è un elemento HTML Paragraph.</P>
```

Non sempre viene utilizzato un tag di chiusura in questo contesto. Questo avviene perché l'HTML e il linguaggio di origine SGML consentono di omettere alcuni tag di chiusura senza invalidare il codice.
Poiché un paragrafo in HTML non può essere annidato all'interno di un altro paragrafo, l'elaboratore è in grado di leggere il tag di apertura del paragrafo e di presumere che indichi anche la fine del paragrafo precedente. Queste tecniche di minimizzazione non sono consentite nel linguaggio XML. Questa caratteristica costituisce la differenza sintattica più evidente tra i due linguaggi.

Il tag di elemento vuoto

Il linguaggio XML supporta un collegamento per elementi vuoti, il *tag di elemento vuoto*. Questo tag unisce i tag di apertura e di chiusura per un elemento senza alcun contenuto. Viene utilizzato un formato speciale: <NOMETAG/>. In questo caso la barra segue il nome del tag, il che non è possibile nel linguaggio HTML.

Attributi

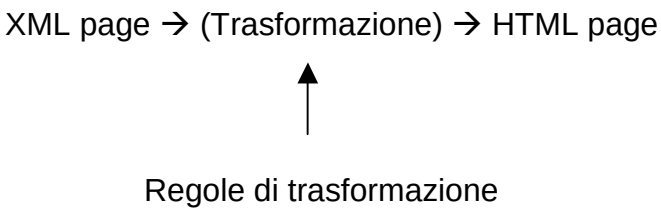
Gli *attributi* consentono di associare valori a un elemento senza che siano considerati parte del contenuto dell'elemento stesso. Ad esempio osserviamo un comune elemento HTML e l'utilizzo di un attributo:

```
<A HREF = "http://www.microsoft.com">Microsoft Home Page</A>
```

In questo caso l'elemento Anchor indicato dal tag <A> contiene un attributo denominato HREF. Il valore dell'attributo è http://www.microsoft.com. Mentre il valore non viene mai visualizzato all'utente, l'attributo contiene importanti informazioni relative all'elemento e fornisce la destinazione dell'ancoraggio. Questo formato del nome e del valore mostra il modo in cui sono utilizzati gli attributi nel linguaggio XML.

XML Transformations

Come può essere utile un linguaggio che può essere capito solo da alcuni browsers?
Bisognava creare delle specifiche di stile XSL che permettevano di "trasformare" una pagina XML in qualcos'altro che tutti i browser potevano elaborare:



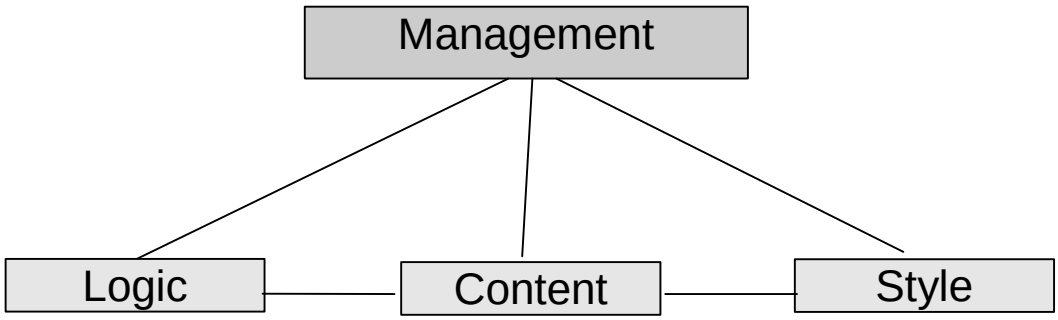
Questo ci permette di scrivere la pagina in XML, creare lo stile della pagina grazie alle regole di trasformazione e generare la pagina HTML. Apache Cocoon 1.0 faceva esattamente questo.

2.4. Il Modello Evolve

Essendo l'XML un metalinguaggio, questo significa che il software XML può lavorare su qualsiasi programma senza dover sapere di cosa si tratta. Se ad esempio un telefono cellulare richiede una pagina, Cocoon deve solo cambiare le regole di trasformazione e spedire la pagina WAP al telefono. Oppure se si richiede un documento PDF per stampare un documento, basterà cambiare le regole di trasformazione e Cocoon genererà il PDF richiesto, o il VRML, oppure il VoiceML, ecc. Tutto questo, ovviamente, senza cambiare la struttura di base definita nel documento XML.

Separazione dei Concetti (SoC)

Cocoon non fu il primo prodotto a sviluppare trasformazioni di XML. L'innovazione che lo distingue dagli altri va ricercata nella particolarità che è stato sviluppato basandosi sulla separazione dei concetti (SoC).
La SoC rende possibile separare le aree di lavoro ed assegnare a ciascuna area più persone che possono quindi specializzarsi in un determinato lavoro secondo le proprie abilità. Si può infatti affermare che l'unione di più persone con abilità comuni in diversi gruppi di lavoro, aumenta la produttività e riduce i costi di gestione se i gruppi hanno compiti ben divisi e complementari. Per un sistema di pubblicazione su web, il progetto Cocoon usa una *piramide di contratti* che descrive le quattro principali aree di lavoro:



Cocoon è ideato con lo scopo di isolare le quattro aree concettuali, togliendo il legame che c'era precedentemente tra stile e logica. Queste aree quindi vengono divise in due diversi files, assegnando a due diversi gruppi di lavoro questi compiti che saranno collegati dalle "pipes" (i contatti).

Esempio:

```
<page>
 <content>
  <para>Today is <dynamic:today/></para>
 </content>
</page>
```

In questo codice si può definire che il tag <dynamic:today/> stampi l'ora della giornata quando viene incluso in una pagina. L'area incaricata ad elaborare i contenuti non deve avere il know-how necessario per generare quel contenuto, per il semplice fatto che quello è compito di un'altra area. Quindi <dynamic:today/> è il "contenuto - logico" del documento. Secondo questa logica, la struttura della pagina è formata con l'unione tra il

lavoro di chi si occupa dei contenuti e dei disegnatori grafici, che devono definire le regole di trasformazione della struttura XML in un linguaggio in cui tutti i browser possano processarla (HTML, for example).

Le varie aree possono essere sviluppate completamente in parallelo, senza sovrapporre le risorse umane a disposizione. I costi decrescono in quanto I tempi si riducono e gli errori non si propagano nelle aree rendendo più facile la manutenzione dell'applicazione.
Ad esempio, possiamo dire ai disegnatori grafici di sviluppare uno "stile Natalizio" del nostro sito e, senza dire nulla agli altri gruppi, cambiare le regole di trasformazione la mattina di Natale. Con Cocoon questo è possibile modificando solo alcune righe di codice.

2.5. I Namespaces

Un XML Namespace è un prefisso usato per definire uno spazio di XML. Per evitare di creare ambiguità tra i tag, il namespace viene riferito ad un'URI.

Il Parser verifica se il documento è valido (se i Tag sono usati in modo corretto), e quando viene trovato un particolare comando, applica una serie di classi e genera degli eventi per formare una struttura.

Esempio dichiarazione ns

Il documento:

```
<?xml version="1.0"?>
<g4>
 <titolo>cocoon's slides</titolo>
  <slides>
    <intestazione>Apache Cocoon</intestazione>
  </slides>
 </g4>
```

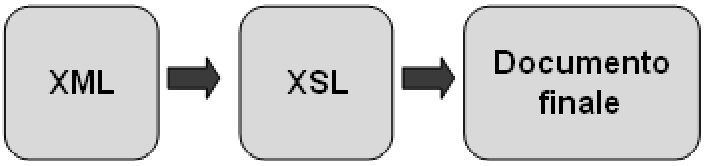
Puo diventare:

```
<?xml version="1.0"?>
<g4:esempio xmlns:g4="http://www.gruppo4.com/slide_ns">
<g4:gruppo4>
 <g4:titolo>cocoon's slides</g4:titolo>
  <g4:slides>
    <g4:intestazione>Apache Cocoon</g4:intestazione>
  </g4:slides>
 </g4:gruppo4>
```

2.6. L’XSLT

XSL, acronimo di **Extensible Stylesheet Language**, è un metalinguaggio che permette di definire le regole di trasformazione che descrivono come devono apparire i file codificati in formato XML.

Un foglio di stile XSLT è un documento XML valido e ben formato. Una trasformazione può essere così schematizzata:



Dato un documento XML d'origine e un foglio XSLT ad esso associato, un processore XSL produrrà un terzo documento. In quest'ultimo, i contenuti saranno presi dal primo documento, la struttura e le regole di presentazione dal secondo.

Struttura di base

La struttura minima contiene due parti:

- **il prologo**
- **l'elemento radice**

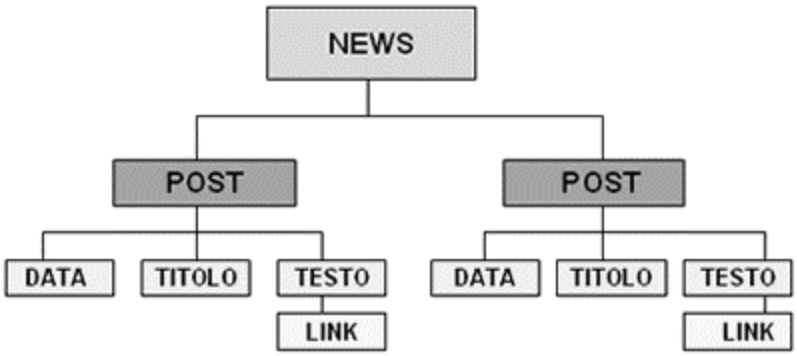
Nel prologo si specifica in genere solo la dichiarazione XML:

<?xml version="1.0">

Subito dopo il prologo è necessario dichiarare l'elemento radice. In un documento XSL l'elemento radice è `<xsl:stylesheet>`. Esso deve contenere un attributo che definisca la versione del linguaggio e almeno una dichiarazione di namespace. Quest'ultima ha come valore: `"http://www.w3.org/1999/XSL/Transform"`.
La struttura di base di un foglio XSLT si presenta dunque così:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
.................................
</xsl:stylesheet>
```

Tra il tag di apertura e quello di chiusura dell'elemento radice vanno definite tutte le regole di trasformazione.
Un foglio di stile XSLT vede un documento XML come un albero fatto di nodi. Nella figura sottostante è evidenziata graficamente la sua struttura:



`<news>` è l'elemento radice; `<post>` è un suo elemento figlio. `<data>`, `<titolo>` e `<testo>` sono sullo steso livello, ma tutti sono figli di `<post>`. L'elemento `<link>` è un nodo figlio di `<testo>`. Esso ha anche un attributo: `href`.

Un processore XSLT può trattare sette tipi di nodi presenti in un documento XML:

- **l'elemento radice**
- **gli attributi e i loro valori**
- **i commenti**
- **gli elementi**
- **i namespace**
- **le istruzioni di elaborazione**
- **il testo contenuto in un nodo**

In un foglio XSLT è necessario specificare delle regole per trasformare i singoli nodi, che vengono definite in un template. Con i template si indica al processore XSL di eseguire una determinata trasformazione quando incontra il nodo specificato.

Esempio:
Supponiamo di voler trasformare "news.xml" in XHTML e che l'elemento <testo> con il suo contenuto debba essere racchiuso in un paragrafo. In XSL creeremo questo template:

```
<xsl:template match="post">
<p><xsl:value-of select="testo"/></p>
</xsl:template>
```

Il processore percorrerà tutto l'albero del documento. Quando incontrerà l'elemento <post>, verificherà che <testo> sia presente come elemento figlio, trasformerà il testo in un paragrafo.

Rintracciare i nodi

Per individuare i singoli nodi si ricorre alla sintassi Xpath. Per capire come funziona Xpath si può fare riferimento al modo in cui si individuano file e cartelle in un file-system. Se scrivo: `C:\documenti\ritratto.jpg` significa che individuo:
un elemento radice (C:\)
una cartella al suo interno
un file all'interno della cartella
Se volessi specificare il path per l'elemento <testo> del file news.xml scriverei così in Xpath:

```
<xsl:template match="news/post/testo">
```

La sintassi Xpath è notevolmente più complessa e se ben usata consente di individuare e trasformare ogni aspetto di documenti XML, anche molto complessi.

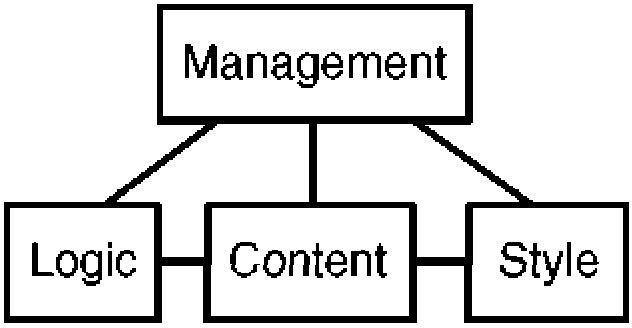
Elementi principali

Elenchiamo ora alcuni dei più importanti tag XSLT specificando per ciascuno la funzione:

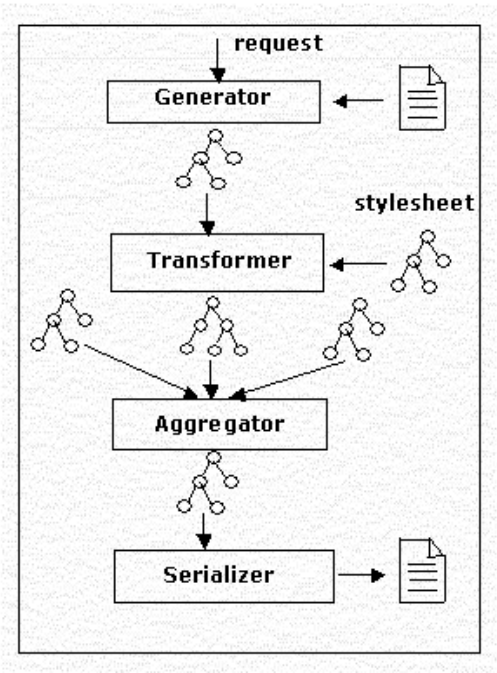
xsl:apply-templates	Applica le regole definite in un tamplate.
xsl:attribute	Crea un attributo per un elemento.
xsl:comment	Genera un commento nel documento finale
xsl:element	Crea un nuovo elemento
xsl:for-each	Stabilisce una struttura ciclica ripetendo un template tutte le volte che un dato nodo viene incontrato.
xsl:if	Usato per impostare semplici strutture condizionali.
xsl:include	Include un foglio XSL esterno
xsl:sort	Ordina un insieme di nodi in bse ai parametri forniti.
xsl:stylesheet	Elemento radice di un documento XSL
xsl:template	Genera un template.
xsl:text	Genera nuovo testo nel documento di output
xsl:value-of	Restituisce il valore di un elemento o di un attributo.
xsl:when	Usato per espressioni condizionali complesse

3. CAPIRE APACHE COCOON 2.1

Separare contenuti, logica, stile e gestione delle funzioni in un sito o servizio web. Questa è l'innovazione di base che Cocoon ci offre:



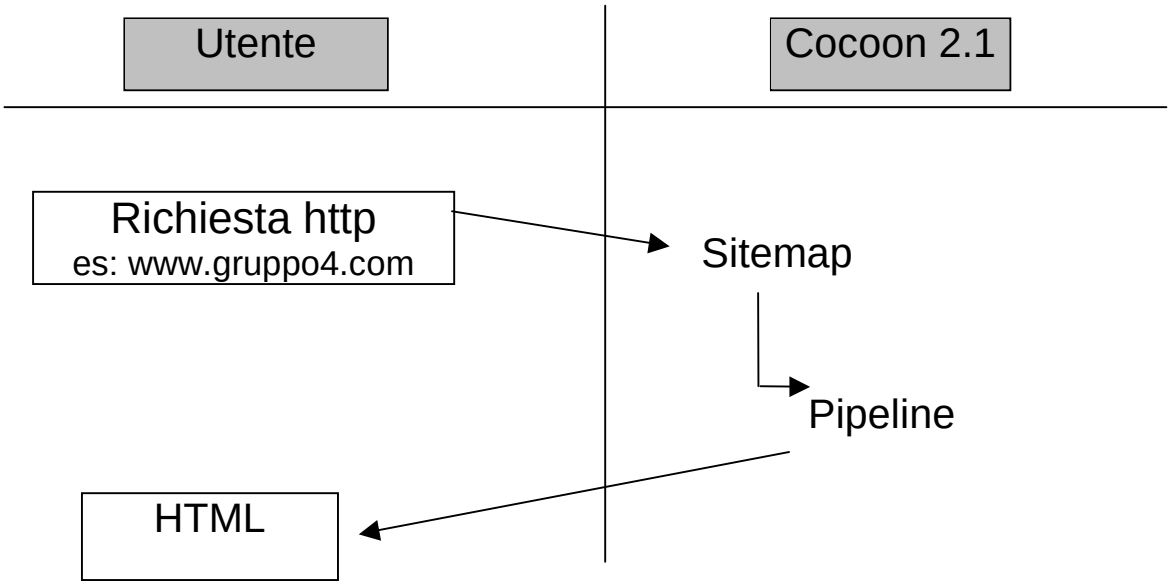
Quando Cocoon riceve una richiesta per processare un documento, genera un documento XML (che può contenere dati di sola lettura, di logica, ma anche una query, ecc) al quale poi verranno applicate delle trasformazioni definite in un altro file (stylesheet).
Il documento così formato verrà, se necessario, unito ad altri documenti e infine sarà formattato e reso disponibile nel formato richiesto.



Il concetto che sta alla base della framework Cocoon è lo sviluppo di applicazioni web basate sull'integrazione di singole componenti ognuna delle quali esegue un compito specifico. In modo tale da facilitare la realizzazione di operazioni tanto complesse quanto estremamente diversificate, gestendole modulatamente attraverso una serie di semplici operazioni.

3.1. SITEMAP

La Sitemap è l'elemento chiave di Cocoon: questo documento infatti contiene le informazioni su come devono essere elaborate tutte le richieste che riceve. La sua struttura è formata da una prima parte di componenti e infine dalle pipelines. La pipeline assembla tutti i componenti necessari per produrre l'output. I principali sono: un generator, zero o più transformer e un serializer. Questi componenti, quindi, vengono sviluppati separatamente e agganciati insieme all'interno della pipeline.



Attraverso una o più sitemap vengono definiti, utilizzando una sintassi dichiarativa XML, i seguenti aspetti:

- i componenti di Cocoon. Tra i quali troviamo i generator, i transformer ed i serializer, in cui vengono inoltre specificati vari parametri, come il nome che lo identifica e la classe Java che lo implementa (oltre a quelli descritti sopra, Cocoon offre altri componenti per effettuare operazioni di vario tipo e complessità);
- le pipeline gestite da Cocoon. Ognuna di queste corrisponde ad un determinato pattern ed è messa in moto in risposta ad una richiesta indirizzata ad un certo URI locale da parte del client; tale URI è detto *virtual-URI* poiché ad esso non corrisponde uno specifico file sul server da inviare quale risposta al client. Il **virtual-URI** infatti è utilizzato per selezionare una specifica pipeline che, tramite una sequenza di operazioni, (definite dalla catena di componenti di cui è composta) effettua le azioni richieste dal client e produce una adeguata risposta per il client stesso.

La semplicità della sitemap permette anche ai programmatori meno esperti di creare siti e applicazioni web formati da componenti di logica e documenti XML.
Questo documento è scritto usando il concetto di “eventi a cascata” (come nei documenti W3C CSS) e di linguaggio dichiarativo (come il linguaggio W3C XSLT).

La sitemap ha i seguenti obiettivi:

- Essere un linguaggio semplice.
- Non richiedere strumenti specifici per la sua creazione, pur essendo predisposta per questa possibilità futura.
- Non imporre limiti di grandezza al sito.
- Contenere tutte le informazioni usate da Cocoon per generare tutte le richieste che riceve.
- Contenere informazioni per operazioni sia dinamiche che statiche.
- Disporre di un potente mezzo di gestione per tutte le URI che vengono richieste.
- Disporre di tutte le funzionalità di base di web-serving (redirection, pagine di errore, autorizzazioni).
- Non limitare le caratteristiche di Cocoon.
- Elaborare le risorse con tutte le possibili variabili di stato, e non solo con l’URI (parametri http, variabili di stato, parametri del server, tempo, ecc...).
- Possedere la nozione di "semantic source" per essere compatibile con le future applicazioni.
- Essere abbastanza flessibile da permettere a Cocoon di creare una applicazione web intera e completa.

Il tag <map:sitemap>

Questo tag è l’elemento radice della pagina e contiene come attributo il namespace che indica al parser informazioni riguardanti la versione del linguaggio usato.
Invece di usare un attributo (del tipo versione="1.0") che potrebbe creare ambiguità, viene usato il namespace:

```
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
```

Infatti, mentre l’attributo “versione” può essere creato per una sitemap personale, si associa un’URI alla versione per identificarla univocamente.

All’interno di questo tag, troviamo una prima sezione composta da tutti i componenti, seguita da una sezione contenente le pipeline.

Le componenti sono:

```
  <map:components>
    <map:generators/>
    <map:transformers/>
    <map:serializers/>
    <map:readers/>
    <map:selectors/>
    <map:matchers/>
    <map:actions/>
    <map:pipes/>
  </map:components>
```

Le impostazioni definite in questi elementi verranno poi richiamate nelle pipeline, quando viene elaborata la richiesta. Se non sono specificate impostazioni particolari, si utilizzano le impostazioni di default.
Tutti i componenti hanno gli attributi “name” e ”src” in comune:

name
Identifica univocamente un componente che potrebbe essere richiamato in seguito dalla sezione delle pipeline.

src
Specifica la classe che implementa questo componente.

Parametri dei componenti
Tutti i componenti sono configurati con dei parametri definiti nei loro elementi figli quando vengono creati. Il seguente esempio mostra come specificare il parametro <use-request-parameter> per una componente della trasformazione XSLT:

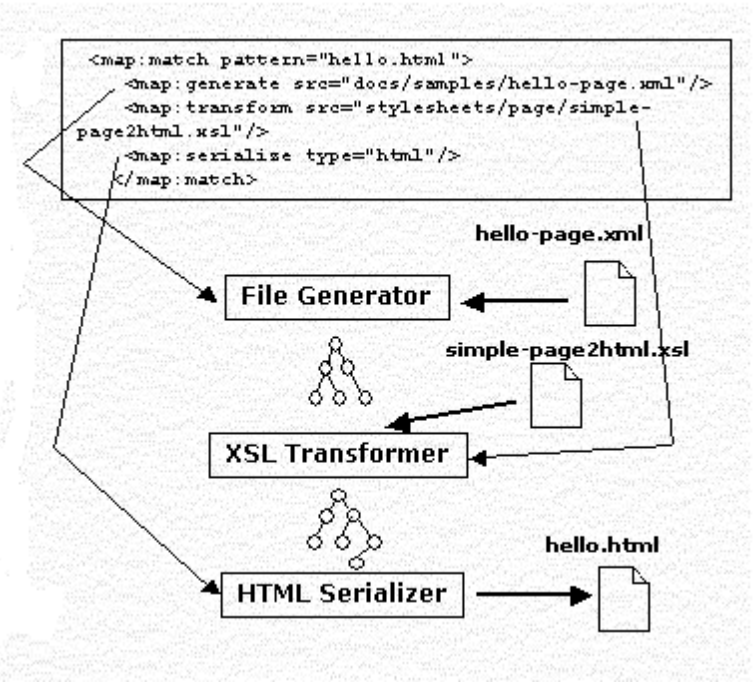
```
<map:components>
  <map:transformer name="xslt"
      src="org.apache.cocoon.transformation.TraxTransformer">
     <!-- Questo è un parametro per i transformer -->
     <use-request-parameters>false</use-request-parameters>
  </map:transformer>
</map:components>
```

3.2. Pipeline

La pipeline è una catena di processi che si occupa, a fronte di una richiesta, di formattare in output il risultato.

Cocoon è stato sviluppato attorno alla nozione di pipeline (letteralmente conduttura) che rende possibile la separazione dei concetti. Ogni applicazione web sviluppata utilizzando Cocoon è sostanzialmente composta da un certo numero di pipeline, ognuna delle quali è riferita ad un URL e corrisponde ad un determinato pattern. Ogni pipeline contiene una catena di elaborazioni da effettuare ogni volta che la pipeline stessa viene invocata (ad esempio da un client effettuando una richiesta indirizzata al particolare URL corrispondente). All'interno delle pipeline viene indicata una sequenza di componenti che all'attivazione della pipeline stessa eseguono elaborazioni particolari in successione. Vi sono tre grandi categorie di componenti disponibili in Cocoon: i generator, i transformer ed i serializer. Ogni pipeline è di solito composta dalla sequenza di un generator, uno o più transformer ed un serializer.

Il generator si occupa di prelevare l'informazione da una fonte e di immettere il risultato nella pipeline sotto forma di eventi SAX (tutto il processo di pipeline avviene manipolando l'XML). Il transformer filtra il dato grezzo preparandolo per essere presentato al richiedente. Il serializer infine si occupa di erogare il dato finale nella forma più consona al richiedente (HTML, WML, testo etc.). Naturalmente Cocoon fornisce una serie di componenti predisposte a coprire la maggior parte delle esigenze, ma nulla vieta di scrivere i propri componenti.



3.3. Matchers

```
<map:match pattern="slides">
      <map:generate src="xml/slides.xml"/>
      <map:transform src="stylesheets/slides2html.xsl"/>
      <map:serialize type="html"/>
    </map:match>
```

Il matcher è un componente chiave di Cocoon. Questo infatti permette di associare un virtual-URI ad una serie di istruzioni. I matchers della sitemap sono usati per determinare il flusso e l'ordine del processo di richiesta. Descrivono come si genera, elabora e presenta la richiesta del client. Possono inoltre essere usati per re-indirizzare le richieste ad altre pipelines o chiamare risorse di altre sitemap.

Le richieste vengono cercate a seconda dell'ordine in cui sono inserite nella pipelines. Le parti variabili del pattern (gli eventuali asterischi) verranno memorizzate per essere poi utilizzate in altre parti della sitemap .Quando un match viene trovato, inizia il processo della relativa pipeline.

Esempio:

```
<map:match pattern="body-faq.xml">
  <map:generate src="xdocs/faq.xml"/>
  <map:transform src="stylesheets/faq2document.xsl"/>
  <map:transform src="stylesheets/document2html.xsl"/>
  <map:serialize/>
</map:match>

<map:match pattern="body-**.xml">
  <map:generate src="xdocs/{1}.xml"/>
  <map:transform src="stylesheets/document2html.xsl"/>
  <map:serialize/>
</map:match>
```

I due esempi processano richieste URI per mezzo dei due matcher, che usano due diversi virtual URI: il primo definisce le azioni da applicare solamente alle richieste URI composte esattamente dalla stringa "*body-faq.xml*", mentre il secondo processa tutte le richieste che cominciano con "*body-*" e finiscono con"*.xml*".

Per esempio, la richiesta "body-cocoon.xml" sarà gestita dal secondo matcher.

Ordine

E’ importante capire che Cocoon è basato su un approccio di "first-match". Infatti quando Cocoon riceve una richiesta, verrà cercato il matcher appropriato da applicare nell’ordine in cui sono scritti nella sitemap. Quindi il primo match che ha successo, viene processato. Questo significa che i match con un pattern specifico devono essere scritti prima di quelli con un pattern generico. Se l’ordine dei due match dell’ultimo esempio fosse invertito, la richiesta "body-faq.xml" sarebbe stata presa dal match"body-**.xml" (e non dal match "body-faq.xml", come sarebbe stato corretto) perché appare prima.

Simboli riassuntivi (tokenization)

Un’altra importante caratteristica dei matchers è l’utilizzo di simboli che vengono memorizzati per un uso successivo. Questa rimane disponibile all’interno del match come un argomento numerico ed è anche chiamata “parte variabile”.
Sempre usando l’esempio precedente, consideriamo la richesta "body-index.xml" processata dal secondo map:match. La stringa "index" che in questo caso sostituisce il simbolo "**", è disponibile per un eventuale uso da parte degli elementi figli del matcher. Sempre in questo esempio, si può notare che la stringa viene successivamente richiamata nell’elemento <map:generate> usando la chiave {1}. Questa chiave è usata come un parametro per il generator che prima sostituirà "index" a {1}, e poi cercherà nella directory "xdocs” il file “index.html".

Espressioni regolari (regexp)

La maggior parte dei matchers in Cocoon sono costruiti usando le espressioni regolari. Imparare a maneggiare questi potenti sistemi per definire i pattern è molto importante perché permette di semplificare molto la struttura della sitemap.

Le espressioni regolari sono basate in un poche e semplici regole:

- Un asterisco ('*') unisce zero o più caratteri, finchè non si presenta il carattere '/' (che indica un separatore di path). Una stringa, come "/cocoon/docs/index.html", NON viene processata con successo con il pattern '/*/*.html' in quanto il primo asterisco è relativo alla sola stringa "cocoon". Un pattern corretto è '/*/*/*.html'.
- Una stringa contenente due asterischi ('**') unisce zero o più caratteri. A differenza dell'asterisco singolo, stavolta può essere incluso il separatore di path '/'. In questo caso la stringa "/cocoon/docs/index.html" sarà processata con successo dal match pattern '/**/*.html'. Quindi il doppio asterisco, includendo separatore del path, corrisponde alla stringa "cocoon/docs".
- Dato che con le espressioni regolari, il carattere backslash ('\') indica una sequenza di uscita, la stringa '*' corrisponde al carattere asterisco mentre il doppio backslash ('\\') corrisponde al carattere '\'.
- Quindi il pattern "**/a-*-is-born.html" processa le stringhe del tipo "documents/movies/a-*-is-born.html" o "un/path /molto /lungo /a-*-is-born.html". Il pattern ignorerà invece la stringa " un/path /molto /lungo/a-star-is-born.html".

3.4. Generator

```
<map:match pattern="slides" type="file">
  <map:generate src="xml/slides.xml"/>
  <map:transform src="stylesheets/slides2html.xsl"/>
  <map:serialize type="html"/>
</map:match>
```

Il generator è il componente Cocoon che avvia le elaborazioni di una pipeline. Sostanzialmente esso si interfaccia con una determinata risorsa (che può essere ad esempio un file XML, un database, un file di testo o anche il filesystem) e produce un documento XML seguendo un vocabolario ben determinato, contenente i dati reperiti accedendo alla risorsa stessa. Esistono differenti tipi di generator specifici a seconda delle informazioni che si vogliono ottenere e della particolare risorsa con la quale si deve interagire. Il generator di default è il “file generator”.
I dati reperiti attraverso il generator vengono passati in input al componente di Cocoon che segue immediatamente nella pipeline e che nella maggior parte dei casi è un transformer (anche se nulla vieta che sia direttamente un serializer).

Il documento XML gestito dal generator è specificato nell’attributo “src”:

```
<map:generate src="document.xml" type="file"/>
```

L’attributo type può essere omesso dato che si tratta del generator di default

Si possono inoltre utilizzare pathname assoluti nel caso si voglia generare un documento da un’altra pagina internet, ma bisogna accertarsi che questa sia scritta in XML o XHTML, altrimenti si verificherà un errore (a meno che non usi l’HTMLGenerator).

3.5. Transformer

```
<map:match pattern="slides">
  <map:generate src="xml/slides.xml"/>
  <map:transform src="stylesheets/slides2html.xsl"/>
  <map:serialize type="html"/>
</map:match>
```

Ha il compito di definire il documento (XSLT) che dovrà applicare le regole di formattazione ai dati contenuti nell'XML.
L' xslt transformer è configurabile: si possono infatti specificare una o più delle seguenti configurazioni:

- use-request-parameters: true|false – Fissando questo parametro su true rende disponibili tutti i parametri richiesti dall'XSLT stylesheet. Da notare che questo può creare problemi riguardanti la cachability dell'output generato da questo transformer. L'algoritmo di cache non controlla solo gli ultimi dati modificati, ma anche tutti i valori dei parametri richiesti. Questa proprietà è falsa di default. Se fissata su true i valori dei parametri richiesti saranno disponibili usando una variabile all'interno dell'xslt con il nome del parametro.
- use-browser-capabilities-db: true|false – Questa configurazione si usa solamente per I database forniti dal web e rende tutte le proprietà e le strutture contenute nel database disponibili nell'XSLT stylesheet. Di default questa proprietà è falsa.
- use-cookies: true|false – Questa configurazione forza il transformer a rendere tutti i cookies della richiesta disponibili nell'XSLT stylesheetas. Questa proprietà è falsa di default.

Esistono molti tipi di transformer a seconda delle elaborazioni che si devono effettuare. Tra questi sono largamente usati i transformer XSLT che sono dei documenti di stile (XSLT stylesheet) per mezzo dei quali sono definite specificate le regole da applicare al documento XML ricevuto in input. Vi sono anche dei transformer che si interfacciano con un database per effettuare delle query specificate opportunamente nel documento XML di input includendo i risultati nell'output, o ancora altri transformer che includono nell'output parti di documenti XML esterni.

L'xslt può includere anche un file css col comando:

```
<link rel="stylesheet" href="file.css" type="text/css"/>
```

che dovrà essere dichiarato anche nella sitemap.

SQL Transformer

Lo scopo dell'SQLTransformer è di tradurre il risultato di una query a un database in XML. L'XML che interroga il database avrà questa struttura:

```
<page>
 <title>Hello</title>
 <content>
  <para>This is my first Cocoon page filled with sql data!</para>
  <sql:execute-query xmlns:sql="http://apache.org/cocoon/SQL/2.0">
   <sql:query name="department">
     select id,name from department_table
   </sql:query>
  </sql:execute-query>
 </content>
</page>
```

L'attributo "name" identifica con un nome l'intero rowset corrispondente. Per invocare l'SQLTransformer bisogna aggiungere I seguenti parametri nella sitemap:

```
<map:transform type="sql">
  <map:parameter name="use-connection" value="personnel"/>
  <map:parameter name="show-nr-of-rows" value="true"/>
  <map:parameter name="clob-encoding" value="UTF-8"/>
</map:transform>
```

Il parametro use-connection definisce quale connessione (tra quelle definite nel file cocoon.xconf) viene usata nell'SQLTransformer per ottenere i dati. Il parametro show-nr-of-rows indica il numero di righe contenute nel resultset.

L'output XML apparirà così:

```
<page>
 <title>Hello</title>
 <content>
  <para>This is my first Cocoon page filled with sql data!</para>
  <sql:rowset nrofrows="2" name="department"
              xmlns:sql="http://apache.org/cocoon/SQL/2.0">
    <sql:row>
      <sql:id>1</sql:id>
      <sql:name>Programmers</sql:name>
    </sql:row>
    <sql:row>
      <sql:id>2</sql:id>
      <sql:name>Loungers</sql:name>
    </sql:row>
  </sql:rowset>
 </content>
</page>
```

3.6. Serializer

```
<map:match pattern="slides">
 <map:generate src="xml/slides.xml"/>
 <map:transform src="stylesheets/slides2html.xsl"/>
 <map:serialize type="html"/>
</map:match>
```

Il serializer è il punto finale di ogni matcher e rende disponibili i dati generati dalla richiesta ed elaborati dal transformer.

Come indica il nome stesso, questa componente serializza, ovvero trasforma in un flusso di dati binari (o in una successione di caratteri) il contenuto del documento XML che ricevono in input, al fine di poterlo spedire al client che ne aveva effettuato la richiesta.

Esistono serializers per generare file HTML, XML, PDF, VRML, WAP, ecc… Ovviamente si dovrà aver applicato le opportune regole di trasformazione nel transformer.

Il più semplice serializer è l'XML serializer che crea un file XML contenente i dati strutturati secondo logica il quale non necessita di un transformer, in quanto riceve l'input direttamente dal generator.

3.7. Control Flow

Cocoon dispone di un avanzato control flow, che consiste nel definire, ad ogni punto e in ogni momento dell'applicazione, l'ordine delle pagine che devono essere spedite al client.

Le tradizionali applicazioni web provano a creare un modello di control flow considerando l'applicazione come una "finite state machine" (FSM). In questo modello invece, l'applicazione è composta da più stati che si susseguono. Ogni input ricevuto fa cambiare lo stato dell'applicazione. Durante queste transizioni l'applicazione può generare diversi effetti come aggiornare gli oggetti in memoria o in un database, oppure rispedire al client browser una pagina web.

Questo modello è semplice per modeste applicazioni web, ma per grandi applicazioni, il numero degli stati e transizioni tra loro aumenta rendendo difficile ed impegnativo definire cosa succede nell'applicazione (il completo susseguirsi dei vari stati).

Usando un concetto di programmazione di alto livello chiamato *continuations*, Cocoon prova a risolvere questo problema rendendo modellabile il control flow dell'applicazione web come un normale programma.

Un diverso approccio

Di solito le applicazioni web sono essenzialmente applicazioni a eventi guidati che devono rispondere a richieste del client cambiando il loro stato interno e quindi generando la risposta. Il risultato è che perfino una semplice applicazione che necessita di tenere in memoria informazioni fornite dall'user in più di una pagina, deve mantenere in memoria le informazioni acquisite. Queste informazioni e il punto esatto in cui ritrova il programma che è processato sono contenuti nella *continuations*.
Ad esempio, supponiamo di voler scrivere la fuzione di una semplice calcolatrice, la quale riceve i numeri su cui operare e l'operazione da effettuare da pagine separate:

```
function calculator()
{
  var a, b, operator;
  cocoon.sendPageAndWait("getA.html");
  a = cocoon.request.get("a");
  cocoon.sendPageAndWait("getB.html");
  b = cocoon.request.get("b");
  cocoon.sendPageAndWait("getOperator.html");
  operator = cocoon.request.get("op");
  try {
    if (operator == "plus")
      cocoon.sendPage("result.html", {result: a + b});
    else if (operator == "minus")
      cocoon.sendPage("result.html", {result: a - b});
    else if (operator == "multiply")
      cocoon.sendPage("result.html", {result: a * b});
    else if (operator == "divide")
      cocoon.sendPage("result.html", {result: a / b});
    else
      cocoon.sendPage("invalidOperator.html", {operator: operator});
  }
  catch (exception) {
    cocoon.sendPage("error.html", {message: "Operation failed: " +
    exception.toString()});
  }
}
```

In questo esempio la funzione calculator è chiamata ad effettuare l'applicazione di calcolo. Noi vogliamo che la funzione *sendPageAndWait* sia una funzione particolare che prenda come argomento il file HTML spedito in risposta ed altri valori necessari per stabilire il posizionamento dinamico dei vari files. In altre parole vogliamo che la funzione *sendPageAndWait* spedisca la pagina di risposta e blocchi il processo in esecuzione fino a

quando l’utente rispedisce la richiesta al server. Quando la richiesta arriva, il processo verrà ripreso al punto in cui era stato lasciato, subito dopo la chiamata al *sendPageAndWait*.
Il flusso delle pagine nell’applicazione può essere descritto quindi come un normale programma. Usando questo approccio la propria applicazione web diventa un insieme di processi che, attraverso la generazione di pagine di risposta, passano da uno stato all’altro.
Lo svantaggio sta nel fatto che bisogna mantenere il processo in vita finché l’utente manda l’ultima pagina richiesta. Per evitare questo, particolari oggetti chiamati *continuations* hanno il compito di memorizzare lo stato in cui è l’applicazione e i valori di eventuali variabili locali o globali inseriti fino a quel punto. Grazie a questi oggetti è possibile riprendere l’applicazione esattamente nel punto in cui era stata salvata.

Cosa sono le continuation?

Una continuation è un oggetto che, ad un dato punto nel programma, salva in memoria lo stato del file system, includendo tutte le variabili locali e lo stato del registro. Inoltre in questo oggetto non si può solamente memorizzare l’esecuzione del programma, ma anche ristabilire l’esecuzione dell’applicazione in un dato punto. Questo significa che lo stack trace e il program counter del programma in esecuzione vengono salvati nella continuation. Le continuation che fanno parte di un flusso, sono collegate insieme in una struttura padre/figlio, formando una specie di struttura ad albero. Questo consente ad un utente di poter usare più volte il bottone back del browser senza problemi. Le continuations si auto-eliminano dopo un determinato periodo.

3.8. Aggregazione di documenti

L'aggregazione è creata dall'oggetto aggregator: durante l'esecuzione della pipeline, l'aggregator produce un unico contenuto XML formato dall'unione di una o più parti, ognuna delle quali definita da una sorgente XML indipendente. Il nome dell'elemento madre che contiene l'unione di tutti i contenuti XML è definito dal valore dell'attributo map:aggregate.
L'aggregator quindi "incolla" i vari pezzi di documenti per generare la pagina finale e pertanto viene definito al posto del generator. Un semplice aggregator è definito nell'esempio sottostante, in cui le parti unite tra loro sono figlie dell'elemento "contenuti-aggregati".

```
<map:aggregate element="contenuti-aggregati">
  <!-- inserire qui le parti da aggregare -->
</map:aggregate>
```

Definire un aggregator implica definire le varie parti interne che insieme formano il documento finale. Alcuni esempi di parti che si possono inserire sono:

- *http://foo/bar* per unire contenuti provenienti da un protocollo http.
- *cocoon:/current-sitmap-pipeline/foo/bar* per unire contenuti elaborate da qualche altra pipeline nella stessa sitemap.
- *resource:///class-path-context/foo/bar* per unire contenuti provenienti dal classpath.
- *jar:http://www.foo.com/bar/jar.jar!/foo/bar* per unire contenuti provenienti da un jar attraverso una connessione http.
- *file:///foo/bar* per unire contenuti del filesystem.
- *xmldb:<nome del driver>://your.xmldb.host/db/foo/bar* per unire contenuti provenienti da un database XML.

Definire una componente di un'aggregazione è semplice: basta specificare attraverso l'attributo *src* l'origine del documento XML che si vuole inserire.

Il seguente esempio mostra l'unione di tre parti: un banner, un menu e un contenuto avendo come elemento padre "index".

```
<map:match pattern="*/*">
 <map:aggregate element="index">
  <map:part src="cocoon:/banner" element="banners"/>
  <map:part src="cocoon:/menu_{1}" element="sx"/>
  <map:part src="cocoon:/mdg{1}-{2}" element="chiSiamo"/>
 </map:aggregate>
 <map:transform src="stylesheets/stile-base.xsl" />
 <map:serialize/>
</map:match>
```

Il documento finale (in assenza di fogli di stile) appare cosi:

```
<index >
 <banners >
  <!-- Banner -->
        ...
 </ banners >
 <sx>
  <!-- contenuto del menu -->
        ...
 </sx>
<chiSiamo>
  <!-- contenuto della pagina -->
        ...
 </chiSiamo>

</index>
```

3.9. Reader

Un reader è una particolare pipeline che non necessita di generator, transformer e serializer in quanto il suo compito è di rendere disponibili contenuti binari. In generale possiamo dire che i reader rappresentano i contenuti senza apportare alcuna elaborazione, un esempio è dato dalla visualizzazione di immagini.
Nella sitemap, ciascun reader è identificato con un unico nome e definito con una classe java, ogni ulteriore configurazione è specificata negli elementi figli.

3.10. XSP Logicsheet

Taglibs e Logicsheets

L’XSP logicsheet è una "tag library", ovvero un insieme di tag XML che possono essere usati in un programma XSP per inserire interi blocchi di codice all’interno del file.
Esistono vari "taglibs" predefiniti, ad esempio se l’URL è del tipo "http://myserver.org/index.xml?fruit=apple", usando il taglib request, si può ottenere il valore del parametro "fruit" (che è "apple"):

```
<request:get-parameter name="fruit"/>
```

E’ quindi importante capire che tutti i taglib sono definiti da un logicsheet, che è un particolare tipo di stylesheet XSL, il cui output è un file XSP.

Esempio: Hello World!

Cominciamo con il classico esempio ”Hello, World” estendendolo fino ad usare un semplice documento XSP. Tutti gli esempi di questa sezione produrranno un output HTML essenzialmente equivalente al seguente:

```
<html>
  <body>
    <h1>Hello, world!</h1>
  </body>
</html>
```

Esempio XML/XSL

Un semplice file XML (greeting.xml) sarà:

```
<?xml version="1.0"?>
<greeting>Hello, world!</greeting>
```

E lo stylesheet (greeting.xsl) che lo trasformerà al file HTML finale sarà:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
    <body>
      <h1>
        <xsl:value-of select="greeting"/>
      </h1>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

L'input XML ha un singolo elemento, <greeting>, il cui contenuto viene visualizzato in HTML. In seguito vedremo che elaborando il nostro esempio aggiungendo del codice java attraverso l'XSP, continueremo ad usare lo stesso stylesheet per formattare il nostro output finale. Quindi quest'input apparirà simile all'XML dell'ultimo esempio.

Esempio XSP

Come appena affermato inseriamo del codice Java all'input formando così da creare un semplice programma XSP. Il nuovo file creato (greeting2.xml) verrà elaborato con lo stesso XSL dell'esempio precedente:

```
<?xml version="1.0"?>
<xsp:page xmlns:xsp="http://apache.org/xsp">

  <xsp:logic>
   // Qui ci può essere del codice Java, queries JDBC, ecc.
    String msg = "Hello, world!";
  </xsp:logic>

  <greeting>
    <xsp:expr>msg</xsp:expr>
  </greeting>

</xsp:page>
```

ESEMPIO con l'XSP Logicsheet

Per separare completamente la logica dai contenuti si usano i logicsheet:

greeting3.xml:

```
<?xml version="1.0"?>
<?xml-logicsheet href="logicsheet.greeting.xsl"?>

<xsp:page xmlns:xsp="http://apache.org/xsp"
          xmlns:greeting="http://duke.edu/tutorial/greeting">
  <greeting>
    <greeting:hello-world/>
  </greeting>
</xsp:page>
```

E questo è il logicsheet (logicsheet.greeting.xsl) incaricato alla formattazione di greeting3.xml:

```
<?xml version="1.0"?>

<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xsp="http://apache.org/xsp"
                xmlns:greeting="http://duke.edu/tutorial/greeting">

<xsl:template match="xsp:page">
 <xsl:copy>
  <xsl:apply-templates select="@*"/>
  <xsl:apply-templates/>
 </xsl:copy>
</xsl:template>

<xsl:template match="greeting:hello-world">
  <!-- more complex XSLT is possible here as well -->
  <xsp:logic>
    // this could be arbitrarily complex Java code, JDBC queries, etc.
    String msg = "Hello, world!";
  </xsp:logic>
  <xsp:expr>msg</xsp:expr>
</xsl:template>

<!-- This template simply copies stuff that doesn't match other -->
<!-- templates and applies templates to any children.           -->
<xsl:template match="@*|node()" priority="-1">
 <xsl:copy>
  <xsl:apply-templates select="@*|node()"/>
 </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

Ci sono alcuni elementi da notare in questi file: per prima cosa informiamo il processore XSP che vogliamo usare un determinato XSL (per formattare) i dati con il seguente comando:

```
<?xml-logicsheet href="logicsheet.greeting.xsl"?>
```

Inoltre, nel logicsheet è definito un nuovo namespace (**greeting)** che viene dichiarato in entrambi i documenti con l'URI:

```
xmlns:greeting="http://duke.edu/tutorial/greeting"
```

Infatti i namespace di tutti i logicsheet usati devono essere dichiarati nell'elemento radice (`xsp:page`).

L'URI di riferimento è completamente arbitraria, ma deve essere esattamente la stessa di quello usato nel documento XSP che usa quel logicsheet.
Infine possiamo affermare che il logicsheet è molto simile all'XSL stylesheet. Il motivo per cui è chiamato logicsheet è che può essere applicato non solamente ad un file XML, ma più specificamente ad un file XSP ed il risultato finale sarà un altro file XSP.

Logicsheets

I logicsheets sono usati per tradurre *tag dinamici* in un linguaggio di markup. Per esempio il tag:

```
<util:time-of-day format="hh:mm:ss"/>
```

sarà trasformato dal logicsheet *util* nell'equivalente espressione XSP:

```
<xsp:expr>
  SimpleDateFormat.getInstance().format(new Date(), "hh:mm:ss")
</xsp:expr>
```

I logicsheets possono essere applicati in sequenza, così è possiblile produrre un tag dinamico che verrà in seguito processato da un altro logicsheet. Per esempio:

```
<util:time-of-day>
  <util:param name="format">
    <request:get-parameter name="time-format" default="hh:mm:ss"/>
  </util:param>
</util:time-of-day>
```

verrà trasformato in:

```
<xsp:expr>
  SimpleDateFormat.getInstance().format(
    new Date(),
    <request:get-parameter name="time-format" default="hh:mm:ss"/>
  )
</xsp:expr>
```

e il logicsheet che lo elaborerà lo trasformerà in:

```
<xsp:expr>
  SimpleDateFormat.getInstance().format(
    new Date(),
    XSPRequestHelper.getParameter("name", "hh:mm:ss")
  )
</xsp:expr>
```

3.11. Redirection

Un Redirector permette alla sitemap di passare una richiesta da un URI ad un altro anche se l'Uri non è gestito da Cocoon.
Per esempio, per reindirizzare dalla `page1.html` alla `page2.html`, possiamo usare:

```
  <map:match pattern="page1.html">
    <map:redirect-to uri="page2.html"/>
  </map:match>
```

3.12. Accesso al Database

Pubblicare contenuti dinamici o creare applicazioni web spesso implicano accedere ad un database. Apache Cocoon offre tre diversi metodi (con relativi vantaggi e svantaggi) per accedere ai database: le actions, i transformers e i logicsheets.

Database Actions

Le Actions sono codici eseguiti quando viene processata la pipeline, con un output dipendente da come la pipeline è assemblata. Per esempio, se l'operazione in un particolare database fallisce, verrà visualizzata una pagina di errore. Sono utilizzate soprattutto per stabilire il successo o il fallimento di una determinata operazione sui dati (inserimento, modifica e cancellazione).
Questa particolare tecnica, non necessita di conoscenze di linguaggi particolari di programmazione in quanto i meta-data, quando vengono letti, sono trasformati in un file XML.

SQL Transformer

Usare l'SQL transformer non richiede particolari operazioni di configurazione come l'ESQL logicsheet. Questa tecnica prevede la scrittura della query sql all'interno del file processato dal generator, ma ha lo svantaggio che il risultato non può essere gestito in quanto al momento dell'esecuzione la pipeline è già assemblata e la logica necessaria per gestire, ad esempio, l'operazione di errore, non è disponibile.
Quindi il transformer sql è usato solamente per le operazioni di lettura dei dati.

Esempio SQL Transformer:

```
<execute-query xmlns="http://apache.org/cocoon/SQL/2.0">
        <query name="mdgDB">
                select rid,nome,descrizione FROM ricette WHERE idmac='ID'
        </query>
</execute-query>
```

ESQL Logicsheet

La tecnologia ESQL è usata limitatamente all'interno dei documenti XSP, rendendo leggermente più complicate le operazioni compiute sul database in quanto si aggiunge il livello di complessità del documento XSP.
L'ESQL logicsheet è quindi un documento XSP che interroga il database con delle query sql e ne serializza i risultati in XML. L'ESQL ha il vantaggio di permettere la gestione dell'output di una query in qualsiasi modo grazie alle caratteristiche del documento di logica XSP.
Per esempio, permette di unire il file esql con un altro logicsheet ed offre una serie di funzionalità logiche pre-impostate grazie all'utilizzo dei parametri definiti nella taglib. La connessione al database avviene attraverso un connettore JDBC.

Connessione

La logica Esql può usare delle connection pools configurate nel file cocoon.xconf o configurare di volta in volta la connessione al database. Per evitare di configurare la connessione per ogni query useremo una pool connection.

Dopo aver scaricato i drivers JDBC e copiati nella directory `$COCOON/WEB-INF/lib` si può procedere alla configurazione dei file web.xml e cocoon.xconf

Configurazione file "`web.xml`":

```
<init-param>
     <param-name>load-class</param-name>
     <param-value>
     <!--  for mysql DataBase -->
          com.mysql.jdbc.Driver
      </param-value>
   </init-param>
```

Se si userà la logica ESQL, è necessaria la configurazione del file "cocoon.xconf":

```
<jdbc logger="core.datasources.mdgDB" name="mdgDB">
       <pool-controller max="10" min="5"/>
       <auto-commit>false</auto-commit>
       <dburl>jdbc:mysql://192.168.100.32:3306/mdcDB</dburl>
       <user>NOME</user>
       <password>*****</password>
     </jdbc>
```

Significato dei tag:

`<dburl>jdbc:mysql://some.database.server/databasename</dburl>` indica dove si trova il database a cui connettersi.

`<jdbc name="some-named-connection-pool">` Questo tag crea il pool della connessione del db. Il nome del pool è specificato dal valore del parametro "nome".

`<pool-controller min="minimo" max="massimo">` Determina il numero minimo e massimo di connessioni permesse al database. Per esempio: se ci sono meno connessioni del massimo consentito e le connessioni esistenti non sono disponibili, (perchè usate da qualche processo) alla richiesta di una connessione da parte di un alto processo, il pool crea una nuova connessione.

`<user>` Il nome necessario per connettersi al db.

`<password>` La password necessaria per connettersi al db.

Esempio

In questo esempio selezioniamo due colonne da una tabella. Da notare che il verificarsi di uno dei due tag “esql:results” e “esql:no-results” implica l’esclusione dell’altro nella struttura dell’albero XML. Questo esempio presuppone la connessione ad un datasource definito nel file cocoon.xconf:

```
<esql:connection>
  <esql:pool>connectionName</esql:pool>
  <esql:execute-query>
    <esql:query>SELECT mycolumn1,mycolumn2 FROM table</esql:query>
    <esql:results>
      <table>
        <esql:row-results>
          <tr>
             <td><esql:get-string column="mycolumn1"/></td>
             <td><esql:get-string column="mycolumn2"/></td>
          </tr>
        </esql:row-results>
      </table>
    </esql:results>
    <esql:no-results>
       <p>Sorry, no results!</p>
    </esql:no-results>
  </esql:execute-query>
</esql:connection>
```

Risultati

Il risultato di una query viene di solito gestito dal tag esql:results e i suoi tag figli. Se per esempio si dovesse verificare un errore o si volesse usare una query di aggiornamento, si possono utilizzare i numerosi tags previsti per questi casi speciali.
Ad esempio se ad una interrogazione non c’è nessun risultato, viene usato il tag esql:no-results.

Limitare il numero di righe in un risultato

Esql permette di visualizzare solo una parte dei risultati usando `esql:use-limit-clause`. Con questo tag infatti verranno visualizzate solo il numero di righe indicate. Questo è molto utile in caso di interrogazioni con numerosi risultati: Cocoon per evitare di

visualizzare la richiesta in una unica pagina avendo così una lunghissima lista, divide il risultato in gruppi di righe fissati dall'utente che verranno visualizzati in pagine diverse.
Se il tag `esql:use-limit-clause` è impostato su "auto", esql applica le regole di visualizzazione descritte in seguito.
I tag `esql:skip-rows` e `esql:max-rows` specificano rispettivamente quante righe devono essere saltate a partire dalla prima e il numero massimo di righe da visualizzare.
Il contenuto dei tag `esql:previous-results` e `esql:more-results` verrà visualizzato rispettivamente se sono state omesse righe e se ci sono altre pagine oltre a quella attuale.

Esempio

```
<esql:connection>
  <esql:pool>connectionName</esql:pool>
  <esql:execute-query>
    <esql:query>SELECT mycolumn1,mycolumn2 FROM table</esql:query>
    <esql:use-limit-clause>auto</esql:use-limit-clause>
    <esql:skip-rows><xsp:expr>skiprows</xsp:expr></esql:skip-rows>
    <esql:max-rows>10</esql:max-rows>
    <esql:results>
     <table>
      <esql:row-results>
      <esql:previous-results>Sono stati omessi risultati</esql:previous-results>
        <esql:more-results>ci sono altri risutati</esql:more-results>
          <tr>
             <td><esql:get-string column="mycolumn1"/></td>
             <td><esql:get-string column="mycolumn2"/></td>
          </tr>
        </esql:row-results>
     </table>
    </esql:results>
    <esql:error-results>An error occurred</esql:error-results>
    <esql:no-results>
       <p>Sorry, no results!</p>
    </esql:no-results>
  </esql:execute-query>
</esql:connection>
```

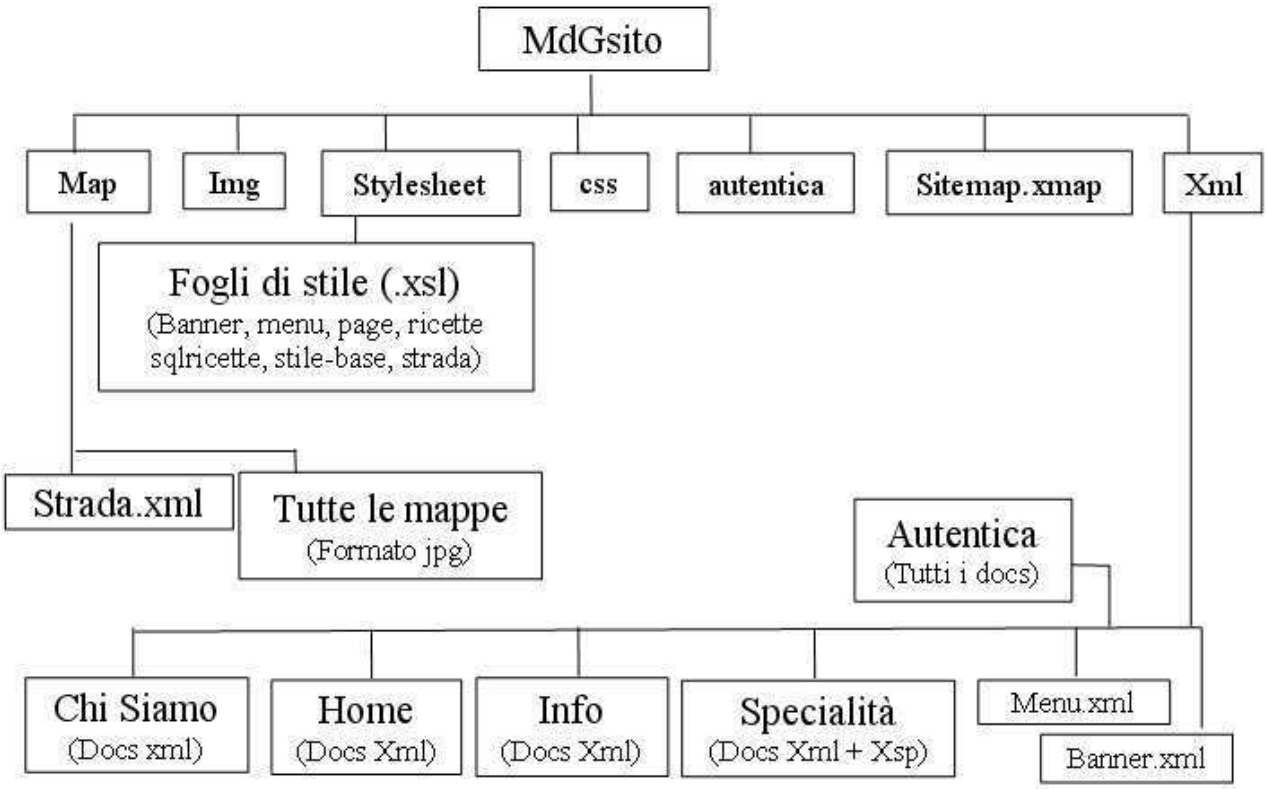
Tag Library

Tag	Descrizione
esql:row-results//esql:get-columns	Hanno come risultato un set di elementi il cui nome è il nome delle colonne. Questo elemento permette di creare una struttura ai dati, potendo definire alcuni elementi che contengono i dati del database
esql:row-results//esql:get-string	Definisce il contenuto di una data colonna sottoforma di stringa
esql:row-results//esql:get-date	Definisce il contenuto di una data colonna sottoforma di data. I formati supportati sono definiti nel file java.text.SimpleDateFormat.
esql:row-results//esql:get-time	Definisce il contenuto di una data colonna che contiene un dato temporale (time). I formati supportati sono definiti nel file java.text.SimpleDateFormat.
esql:row-results//esql:get-boolean	Definisce il contenuto di una data colonna che può assumere solamente valori veri o falsi
esql:row-results//esql:get-double	Definisce il contenuto di una data colonna che assume valori numerici double. Se il dato esiste, il suo valore sarà preso nel formato decimale definito nel file java.text.DecimalFormat.
esql:row-results//esql:get-float	Definisce il contenuto di una data colonna che assume valori numerici float. Se il dato esiste, il suo valore sarà preso nel formato decimale definito nel file java.text.DecimalFormat.
esql:row-results//esql:get-int	Definisce il contenuto di una data colonna che assume valori numerici integer
esql:row-results//esql:get-long	Definisce il contenuto di una data colonna che assume valori numerici long
esql:row-results//esql:get-short	Definisce il contenuto di una data colonna che assume valori numerici short
esql:row-results//esql:get-object	Definisce il contenuto di una data colonna che contiene un oggetto

esql:row-results//esql:get-array	Definisce il contenuto di una data colonna come un java.sql.Array. Questo è usato soprattutto per le liste, sets, etc.
esql:row-results//esql:get-struct	Definisce il contenuto di una colonna come una java.sql.Struct.
esql:row-results//esql:get-xml	Definisce il contenuto di una colonna come un testo xml. Se esiste un attributo root, il suo valore verrà mantenuto.
esql:results//esql:get-column-count	Definisce il numero delle colonne contenute nel resultset.
esql:row-results//esql:get-row-position|esql:results//esql:get-row-position	Definisce il la posizione della riga corrente nel resultset
esql:row-results//esql:get-column-label	Definisce l'etichetta di una data colonna. La colonna deve essere specificata con il suo numero.
esql:row-results//esql:get-column-type-name	Definisce il nome del tipo di una data colonna.
esql:row-results//esql:is-null	Permette di testare colonne nulle. Questo tag conterrà il valore "true" quando contiene un valore nullo.
esql:results/esql:get-metadata	Definisce il metadata associato con il corrente resultset
esql:results/esql:get-resultset	Definisce il corrente resultset
esql:group	Permette di raggruppare elementi di record consectivi con valori identici
@*|node()	E' usato per determinare a quale colonna appartiene una data colonna. Se per esempio l'attributi di una colonna esiste ed è un numero, il suo valore sarà interpretato come la posizione della colonna. Se il valore non è un numero sarà interpretato come il nome della colonna.

4. REALIZZAZIONE PROGETTO

FILE SYSTEM



Nel sito dei Maestri del gusto le pipeline principali sono:

```
<!-- ========================== Pipelines ================================== -->

<map:match pattern="*/*">
     <map:aggregate element="index">
       <map:part src="cocoon:/banner" element="banners"/>
       <map:part src="cocoon:/menu_{1}" element="sx"/>
       <map:part src="cocoon:/mdg{1}-{2}" element="chiSiamo"/>
     </map:aggregate>
     <map:transform src="stylesheets/stile-base.xsl" />
     <map:serialize/>
</map:match>
```

Dove il primo asterisco indica la sezione e il secondo il documento da aprire. In questa pipeline è definito l'aggregator delle tre aree principali: il banner, il menù e l'area centrale

per pubblicare i contenuti. Quando verrà processata, la pipeline unirà i risultati di tre diverse pipeline. Il risultato dell'aggregazione è trasformato con il file *stile-base.xsl* che includendo il foglio di stile (*gusto.css*) genera la pagina HTML.

Stile-base.xsl

```
<xsl:template match="index">
     <html>
      <head>
       <meta http-equiv="content-type"
          content="text/html;charset=iso-8859-1"/>
       <title>I Maestri del Gusto</title>
       <link rel="stylesheet" href="gusto.css" type="text/css"/>
      </head>
          <body>
               <div class="mainbox">
                     <div class="contentbox">
                         <xsl:apply-templates select="banners"/>
                         <xsl:apply-templates select="chiSiamo"/>
                     </div>
                     <div class="leftbox">
                         <xsl:apply-templates select="sx"/>
                         </div>
                    </div>
          </body>
     </html>
</xsl:template>
```

Banner

```
<map:match pattern="banner">
     <map:generate src="xml/banner.xml"/>
     <map:transform src="stylesheets/banner.xsl"/>
     <map:serialize/>
</map:match>
```

Questa pipeline richiama il documento *banner.xml* contenente un solo nodo <banner/>. Quando il foglio di stile *banner.xsl* processerà questo nodo, verrà rappresentata l'immagine scelta come banner.

Menù

```
<map:match pattern="menu_*">
     <map:generate src="xml/menu.xml"/>
     <map:transform src="stylesheets/menu.xsl">
      <map:parameter name="section" value="{1}"/>
     </map:transform>
      <map:parameter name="use-request-parameters" value="true"/>
     <map:serialize/>
</map:match>
```

Come si può notare, nella pipenine del menù viene passato al transformer il parametro "*section*" che indica in che sezione stiamo navigando. Questo serve per individuare la voce del menù da espandere con le proprie sottosezioni.

Menu.xsl

Questo file contiene le regole di visualizzazione del menù. L'area menù è formata dal logo nella parte superiore, dalle varie voci del menù nella parte centrale e da una componente grafica nella parte inferiore. Tutti i nomi delle sezioni principali vengono visualizzati, mentre, grazie al template `match="section"` che esegue un confronto tra la sezione in cui stiamo navigando e il nome delle varie sezioni, vengono visualizzati solo i sottomenù relativi alla sezione attuale.

```
<xsl:param name="section"/>

 <xsl:template match="sx">
         <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="docmenu">
     <IMG SRC="logo.jpg" alt="" width="175" height="155"
border="0"/>
         <xsl:apply-templates/>
     <IMG SRC="border.jpg" alt="" width="175" height="25"
     border="0"/>
  </xsl:template>

<xsl:template match="section">
     <xsl:apply-templates select="title"/>
      <xsl:if test="$section = @name">
       <ul class ="menu1"><xsl:apply-templates select="menu"/></ul>
      </xsl:if>
</xsl:template>
```

```
<xsl:template match="title">
     <p><a class="menu1" href="{@href}"> <xsl:apply-
     templates/></a></p>
</xsl:template>

<xsl:template match="menu">
     <li class="menu1">  <xsl:value-of select="@label"/> </li>
</xsl:template>
```

menu.xml

Questo documento contiene la struttura del menù:

```
</sx>
 <docmenu>
      <section name="chiSiamo">
            <title href="chiSiamo.home">Chi siamo</title>
            <menu label="Storia" href="chiSiamo.storia"/>
            <menu label="Elenco Associati" href="chiSiamo.soci"/>
            <menu label="Qualit&#224; e sicurezza" href="chiSiamo.sicurezza"/>
      </section>
      <section name="specialita">
       <title href="specialita.home_page">SpecialitÃ </title>
            <menu label="Piatti estivi"
            href="specialita.ricette?sez=1&amp;skip=0"/>
            <menu label="Piatti invernali"
            href="specialita.ricette?sez=2&amp;skip=0"/>
            <menu label="Pronti subito"
            href="specialita.ricette?sez=3&amp;skip=0"/>
            <menu label="Pronti in 20-30 mi"
            href="specialita.ricette?sez=4&amp;skip=0"/>
            <menu label="Grandi piatti"
            href="specialita.ricette?sez=5&amp;skip=0"/>
      </section>
      <section name="info">
            <title href="info.home">Informazioni Generali</title>
            <menu label="Aspetti Nutrizionali" href="info.aspettiNutrizionali"/>
      </section>
      <section name="ricetta">
            <title href="ricetta.invia">Invia la tua ricetta</title>

            <menu label="Area Autenticazioni" href="ricetta.login"
            target="_blank"/>
            <menu label="Statistiche" href="ricetta.login"/>
      </section>
 </docmenu>
</sx>
```

Questa è la pipeline che interroga il database usando la logica Esql e limitando il numero di righe visualizzate grazie al parametro “skip” in modo da non avere una lista troppo lunga di ricette. Gli altri parametri che vengono dichiarati sono rid e sez che indicano rispettivamente il numero identificativo della ricetta e la sezione a cui appartiene.

```
      <map:match pattern="mdgspecialita/**">
            <map:generate src="xml/specialita/ricette.xsp" type="serverpages">
              <map:parameter name="sez" value="{request-param:sez}"/>
              <map:parameter name="rid" value="{request-param:rid}"/>
               <map:parameter name="skip" value="{request-param:skip}"/>
             </map:generate>
              <map:parameter name="use-request-parameters" value="true"/>
            <map:transform src="context://mdgsito/stylesheets/ricettexsp.xsl">
                     <map:parameter name="skip" value="{request-param:skip}"/>
            </map:transform>
            <map:serialize type="html"/>
      </map:match>
```

La logica è definita nel documento XSP:

```
<text>
      <xsl:param name="sez"/>
      <xsl:param name="rid"/>
      <xsl:param name="skip"/>
            <titolo>Le nostre specialit&#224;</titolo>
 <esql:connection>

 <esql:pool>mdgDB</esql:pool>
            <esql:execute-query>
                  <esql:query>
                    select rid, nome, descrizione FROM ricette
                    WHERE tiporicetta=<esql:parameter><xsp-request:get-parameter
                    name="sez"/></esql:parameter>
            </esql:query>
      <esql:use-limit-clause>auto</esql:use-limit-clause>
    <esql:skip-rows><xsp-request:get-parameter name="skip"/></esql:skip-rows>
    <esql:max-rows>2</esql:max-rows>
            <esql:results>
             <rowset>
              <esql:row-results>
<esql:previous-results><be4>Pagina precedente</be4></esql:previous-results>
<esql:more-results><after>Pagina successiva</after></esql:more-results>
         <row>
      <nome><esql:get-string column="nome"/></nome>
      <descrizione><esql:get-string column="descrizione"/></descrizione>
      <rid><esql:get-string column="rid"/></rid>
      <sez><xsp-request:get-parameter name="sez"/></sez>
         </row>
            </esql:row-results>
             </rowset>
            </esql:results>
            <esql:error-results>Spiacenti, si &#232; verificato un
                  errore</esql:error-results>
            </esql:execute-query>
```

```
            <esql:execute-query>
             <esql:query>
             select nome, ingredienti, descrizione,tempo_cottura,tiporicetta,
             foto FROM ricette
             WHERE rid=<esql:parameter><xsp-request:get-parameter
             name="rid"/></esql:parameter>
            </esql:query>
          <esql:results>
            <rowset>
             <esql:row-results>
              <row1>
      <nome><esql:get-string column="nome"/></nome>
      <ingredienti><esql:get-string column="ingredienti"/></ingredienti>
      <descrizione><esql:get-string column="descrizione"/></descrizione>
      <tempo_cottura><esql:get-string column="tempo_cottura"/></tempo_cottura>
      <tiporicetta><esql:get-string column="tiporicetta"/></tiporicetta>
      <foto><esql:get-string column="foto"/></foto>
              </row1>
             </esql:row-results>
            </rowset>
          </esql:results>
        </esql:execute-query>
  </esql:connection>

</text>
</xsp:page>
```

La struttura degli output della sezione specialità è :

```
<text>
<xsl:param name="sez"/>
<xsl:param name="rid"/>
     <rowset>
          <row>
               <nome>Spadellata</nome>
               <descrizione>Mettere in padella....</descrizione>
               <rid>1</rid>
               <sez>1</sez>
          </row>

          <row>
               .....
          </row>

     </rowset>
</text>
```

In cui il tag “<row>” si ripete per il numero di righe che sono state trovate in quella sezione, mentre quando viene richiesta una specifica ricetta, verrà generata la struttura <row1> con un diverso stile di visualizzazione:

```
<text>
   <xsl:param name="sez"/>
   <xsl:param name="rid"/>
     <rowset>
      <row1>
          <nome>Spadellata</nome>
          <ingredienti>Pollo, Tacchino, Verdure
          sott'olio</ingredienti>
          <descrizione>Mettere in padella....</descrizione>
          <tempo_cottura>5</tempo_cottura>
      </row1>
     </rowset>
</text>
```

Anche per la pagina relativa alle specialità di ogni macelleria è stata usata la logica Esql e tutte le pagine vengono formattate con lo stesso stylesheet:

```
      <map:match pattern="mdgchiSiamo/Macelleria_*">
            <map:generate src="xml/chiSiamo/macelleria.xsp" type="serverpages">
              <map:parameter name="rid" value="{request-param:rid}"/>
              <map:parameter name="parameter_mac" value="request-param:{1}"/>
              <map:parameter name="use-request-parameters" value="true"/>
            </map:generate>
            <map:transform src="context://mdgsito/stylesheets/sqlRicetta.xsl">
              <map:parameter name="mac" value="{1}"/>
              <map:parameter name="clob-encoding" value="UTF-8"/>
              <map:parameter name="use-request-parameters" value="true"/>
            </map:transform>
            <map:serialize type="html"/>
      </map:match>
```

Il cui relativo documento xsp usato dal generator è:

```
<text>
      <xsl:param name="mac"/>
      <xsl:param name="rid"/>
 <esql:connection>
 <esql:pool>mdgDB</esql:pool>
            <esql:execute-query>
                  <esql:query>
                    select rid, nome, descrizione FROM ricette
                    WHERE macelleria=<esql:parameter><xsp-request:get-parameter
                    name="mac"/></esql:parameter>
            </esql:query>
            <esql:results>
             <rowset>
              <esql:row-results>
               <row>
                  <nome><esql:get-string column="nome"/></nome>
                  <descrizione><esql:get-string
                   column="descrizione"/></descrizione>
                  <rid><esql:get-string column="rid"/></rid>
                  <mac><xsp-request:get-parameter name="mac"/></mac>
               </row>
            </esql:row-results>
```

```
            </rowset>
           </esql:results>
           </esql:execute-query>

           <esql:execute-query>
            <esql:query>
            select nome, ingredienti, descrizione,tempo_cottura FROM ricette
            WHERE rid=<esql:parameter><xsp-request:get-parameter
            name="rid"/></esql:parameter>
           </esql:query>
         <esql:results>
           <rowset>
            <esql:row-results>
             <row1>
              <nome><esql:get-string column="nome"/></nome>
              <ingredienti><esql:get-string
              column="ingredienti"/></ingredienti>
              <descrizione><esql:get-string
              column="descrizione"/></descrizione>
              <tempo_cottura><esql:get-string
              column="tempo_cottura"/></tempo_cottura>
             </row1>
            </esql:row-results>
           </rowset>
          </esql:results>
        </esql:execute-query>
  </esql:connection>
</text>
</xsp:page>
```

Infine, queste pipeline servono a re-indirizzare la pagina nulla alla home-page e a visualizzare le immagini:

```
<map:match pattern="">
            <map:redirect-to uri="home.home"/>
      </map:match>

      <map:match pattern="gusto.css">
            <map:read src="css/gusto.css" mime-type="text/css"/>
      </map:match>

      <map:match pattern="*.gif">
            <map:read src="img/{1}.gif" mime-type="image/gif"/>
      </map:match>

      <map:match pattern="*.map">
            <map:read src="map/{1}.jpg" mime-type="image/gif"/>
      </map:match>

      <map:match pattern="*.jpg">
            <map:read src="img/{1}.jpg" mime-type="image/jpg"/>
      </map:match>
```

La pipeline:

```
<map:match pattern="mdgricetta/login">
      <map:mount uri-prefix="mdgricetta" check-reload="yes"
       src="autentica/sitemap.xmap"/>
</map:match>
```

delega la gestione della validazione dell'amministratore ad un'altra pipeline:

```
<!-- ================ -->
<!-- Pagina del login -->
<!-- ================ -->
<map:match pattern="login">
<!-- se si ha già fatto il login, questa action  reindirizza nel doc protetto -->
   <map:act type="auth-loggedIn">
     <map:parameter name="handler" value="demohandler"/>
     <map:redirect-to uri="autentica.protected"/>
   </map:act>
   <map:generate src="docs/login.xml"/>
   <map:transform src="stylesheets/simple-page2html.xsl"/>
   <map:transform type="encodeURL"/>
   <map:serialize/>
</map:match>


<!-- ============================ -->
<!-- verifica dell'autenticazione -->
<!-- ============================ -->
<map:match pattern="do-login">
   <map:act type="auth-login">
     <map:parameter name="handler" value="demohandler"/>
     <map:parameter name="parameter_name" value="{request-
       param:username}"/>
     <map:parameter name="parameter_psw" value="{request-param:psw}"/>
     <map:redirect-to uri="protected"/>
   </map:act>
<!-- in caso di errore, ritorna alla pagina di login -->
   <map:redirect-to uri="ricetta.login"/>
</map:match>

<!-- ========================== -->
<!-- Accesso alle aree protette -->
<!-- ========================== -->

<map:match pattern="protected">
   <map:act type="auth-protect">
     <map:parameter name="handler" value="demohandler"/>
     <map:generate src="docs/logicsheet1.xsp" type="serverpages"/>
   <map:transform type="session"/>
   <map:transform src="context://mdgsito/stylesheets/dynamic-
       page2html.xsl">
     <map:parameter name="servletPath" value="{request:servletPath}"/>
     <map:parameter name="sitemapURI" value="{request:sitemapURI}"/>
     <map:parameter name="contextPath" value="{request:contextPath}"/>
     <map:parameter name="file" value=".xsp"/>
   </map:transform>
   <map:transform type="encodeURL"/>
     <map:serialize/>
   </map:act>
```

```
<!-- in caso di errore, ritorna alla pagina di login -->
    <map:redirect-to uri="ricetta.login"/>
  </map:match>

  <map:match pattern="logicsheet2">
    <map:act type="auth-protect">
      <map:parameter name="handler" value="demohandler"/>
      <map:generate src="docs/logicsheet2.xsp" type="serverpages"/>
          <map:parameter name="use-request-parameters" value="true"/>
    <map:transform type="session"/>
    <map:transform src="context://mdgsito/stylesheets/dynamic-
        page2html.xsl">
      <map:parameter name="servletPath" value="{request:servletPath}"/>
      <map:parameter name="sitemapURI" value="{request:sitemapURI}"/>
      <map:parameter name="contextPath" value="{request:contextPath}"/>
      <map:parameter name="file" value=".xsp"/>
    </map:transform>
    <map:transform type="encodeURL"/>
      <map:serialize/>
    </map:act>
<!-- in caso di errore, ritorna alla pagina di login -->
    <map:redirect-to uri="ricetta.login"/>
  </map:match>

  <!-- ==================== -->
  <!-- Effettuare il Logout -->
  <!-- ==================== -->
  <map:match pattern="do-logout">
    <map:act type="auth-protect">
      <map:parameter name="handler" value="demohandler"/>

      <map:act type="auth-logout"/>
    </map:act>
    <map:redirect-to uri="ricetta.login"/>
  </map:match>
</map:pipeline>

  <!-- ========================== -->
  <!-- Effettuare l'autenticazione -->
  <!-- ========================== -->

<map:pipeline internal-only="true">
  <map:match pattern="authenticate">
    <map:generate src="docs/userlist.xml"/>
    <map:transform src="stylesheets/authenticate.xsl">
      <map:parameter name="use-request-parameters" value="true"/>
    </map:transform>
    <map:serialize type="xml"/>
  </map:match>
</map:pipeline>
```

Il documento xml contenente gli utenti autorizzati è:

```
<authentication>
  <users>
   <user>
    <name>Daniele</name>
    <psw>****</psw>
   </user>
  <user>
    <name>guest</name>
    <psw>****</psw>
   </user>
  </users>
</authentication>
```

L'autenticazione avviene con il file:

```
<xsl:param name="name"/>
<xsl:param name="psw"/>

<xsl:template match="authentication">
  <authentication>
      <xsl:apply-templates select="users"/>
  </authentication>
</xsl:template>

<xsl:template match="users">
    <xsl:apply-templates select="user"/>
</xsl:template>

<xsl:template match="user">
    <!-- Compare the name of the user -->
    <xsl:if test="normalize-space(name) = $name">
    <xsl:if test="psw=$psw">
     <!-- crea l'ID se esiste il nome -->
        <ID><xsl:value-of select="name"/></ID>
    </xsl:if>
    </xsl:if>
</xsl:template>
```

Il template "*user*" confronta nome e password scritte dall'utente con i valori presenti nel documento degli utenti autorizzati e, in caso di successo, il tag ID avrà il valore del nome e consentirà l'accesso all'area riservata.

BIBLIOGRAFIA

Michael Floyd, Costruire siti web con XML, *Tecniche Nuove*

Siti internet consultati:

http://cocooncenter.org/

http://cocoon.apache.org/

http://www.jugpadova.it

http://www.latoserver.it/

http://xml.apache.org/

http://www.developer.com/

http://webservices.xml.com/

http://www.oio.de/

http://www.w3schools.com

http://www.html.it

http://www.xml.com

http://digilander.libero.it

http://java.sun.com

http:// www.zvon.org

http://www.xmlfiles.com

http://www.w3.org

http://www.ibiblio.org

http://xmlfiles.com