Framework di sviluppo Java EE e il progetto Spago

/* bound to please: the Java Enterprise Wide Framework */

Testimonianza di:
Gianfranco Boccalon
gianfranco.boccalon@eng.it

Argomenti del Corso

- Introduzione al framework Spago: scopo, funzionalità, architettura
- Modulo di Dispatching
- Modulo di Presentation
- Modulo di accesso ai dati
- Servizi trasversali

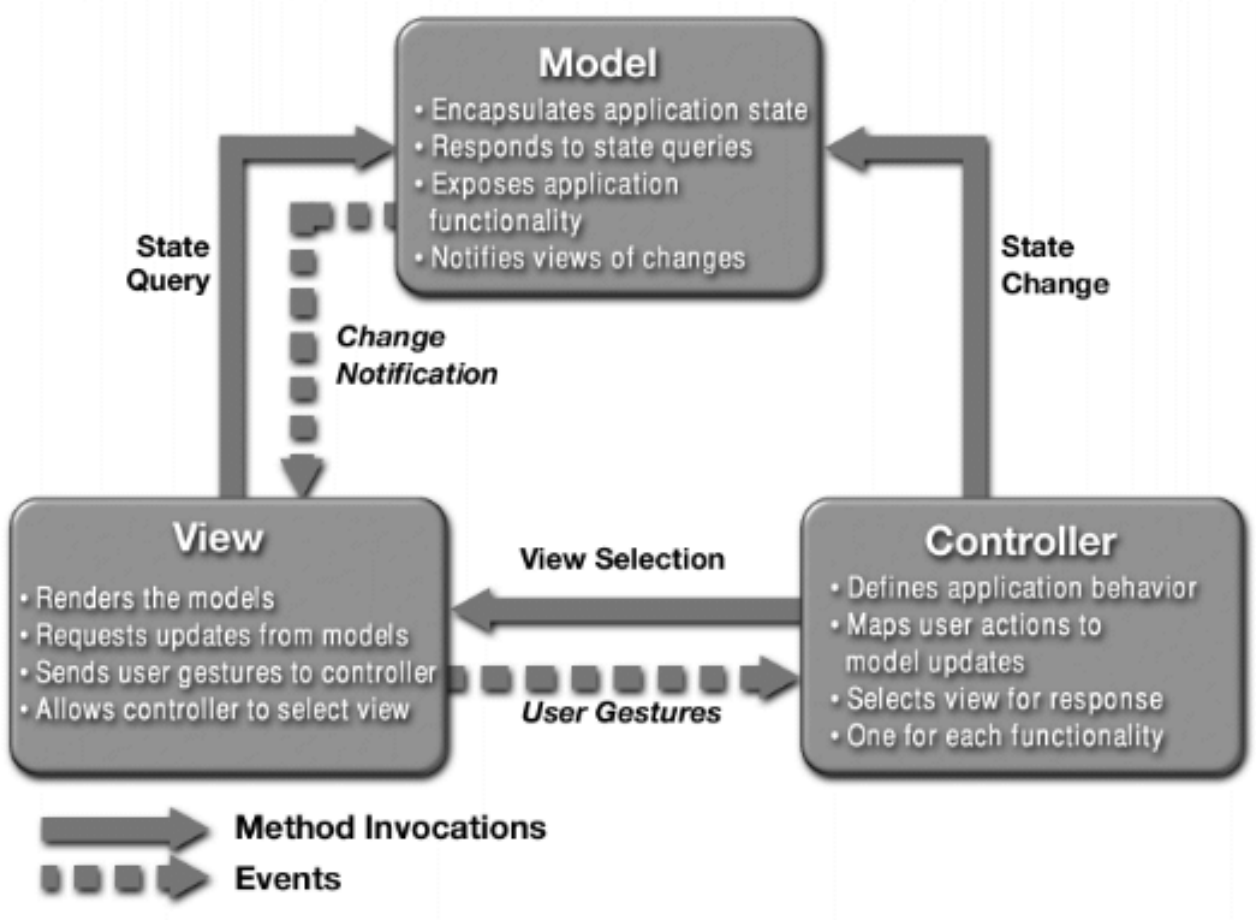
Architettura di riferimento

- Architetture MVC
  - Model-View-Controller è il design pattern architetturale raccomandato dalle linee guida di Sun Microsystems per implementare applicazioni interattive

- Architetture a servizi (SOA)
  - In cosa consistono, quando sono vantaggiose, come implementarle
  - Uno degli standard: SOAP

Architettura MVC

Tale modello comporta la separazione di ogni applicazione in 3 componenti principali



Obiettivi Architettura MVC

La separazione della logica di business dalla presentazione porta i seguenti vantaggi:

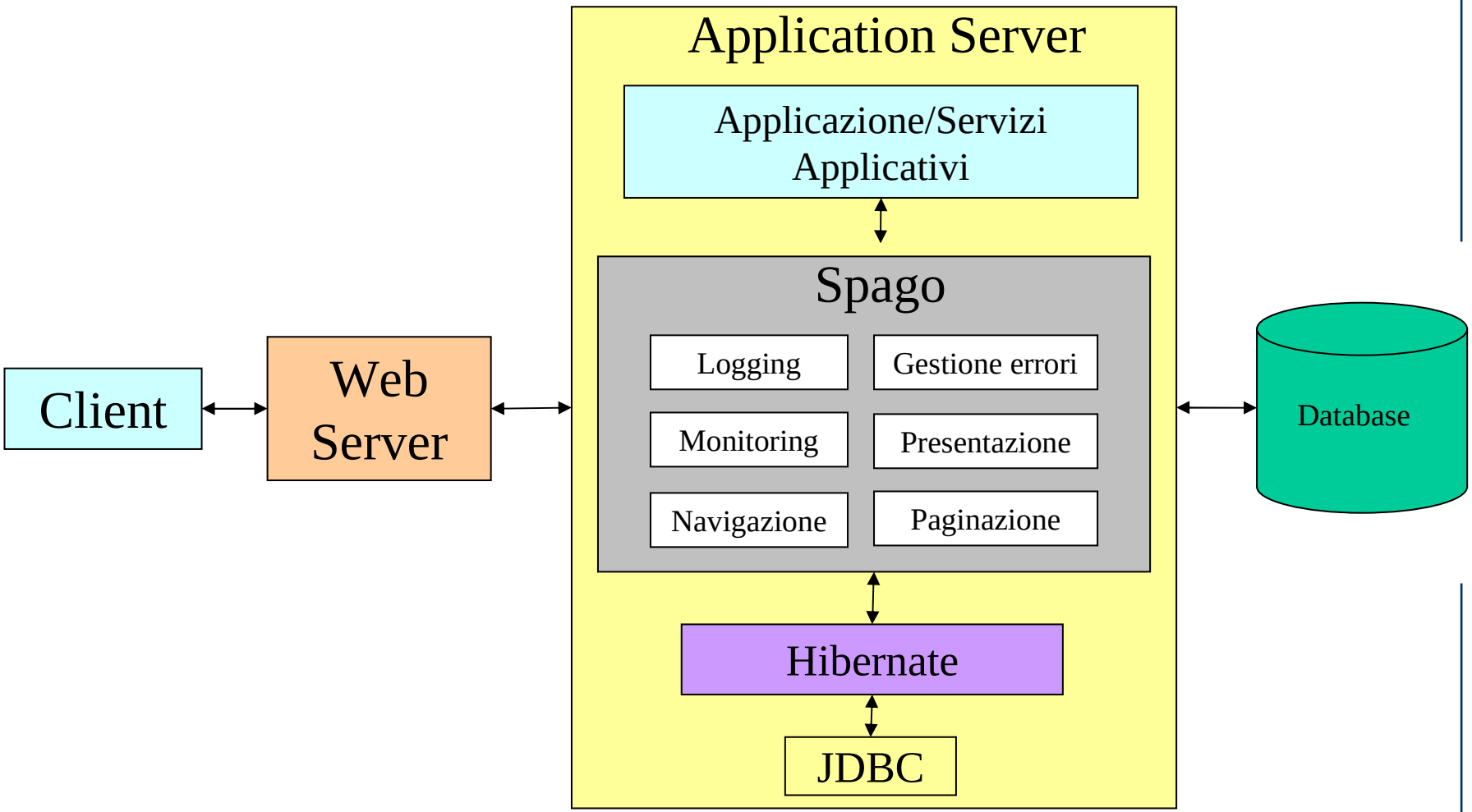
1. Minimizza gli impatti dei cambiamenti
2. Aumenta la manutenibilità
3. Indipendenza dal client e riuso del codice
4. Separazione dei ruoli degli sviluppatori

Una visione d'insieme

- Cos'è un framework
  - Framework infrastrutturali: **Spago**
  - Framework di persistenza: **Hibernate**
  - Framework di templating: **JET**

- Strumenti di sviluppo
  - **IDE**: Eclipse, WSAD
  - **Application Server**: Tomcat, WebSphere Application Server, JBoss Application Server
  - **Versioning**: SubVersion

Una visione d'insieme



- **COS'E' UN FRAMEWORK**

  Un framework è un'applicazione "semi-completa", riutilizzabile che può essere specializzata per produrre applicazioni specifiche

- **VANTAGGI**
  - ***Modularità***
  - ***Riusabilità***
  - ***Estensibilità***
  - ***Inversione del Controllo***

Spago

- **DEFINIZIONE:**

  SPAGO è un framework J2EE, sviluppato secondo il pattern architetturale **MVC** , che fornisce una soluzione per lo sviluppo di applicazioni “multicanale/multiprotocollo” e l’integrazione di servizi.

  Consente lo sviluppo di applicazioni Web, l’integrazione con infrastrutture esistenti e la pubblicazione di servizi su canali diversi.

Confronto con altri framework

Esistono molti framework Open Source per lo sviluppo di applicazioni Web, alcuni dei quali molto specializzati su determinate fasi dello sviluppo (disegno dell'interfaccia grafica, sviluppo della logica di business, etc).

JavaServerFaces

Il futuro di Spago

- L'evoluzione di Spago è inoltre indotta dai nuovi progetti che ne faranno uso (il primo è SpagoBI, una piattaforma per la Business Intelligence)

- Esiste una roadmap evolutiva di Spago sul sito

  http://spago.eng.it

Punti di forza di Spago

1. Multicanalità

2. Rende immediata la pubblicazione dei servizi tramite SOAP

3. Modalità di dispatching a moduli (modalità piu' complessa di quella ad action ma che consente grande flessibilità ed elevato potenziale di riutilizzo del codice)

Punti di forza di Spago

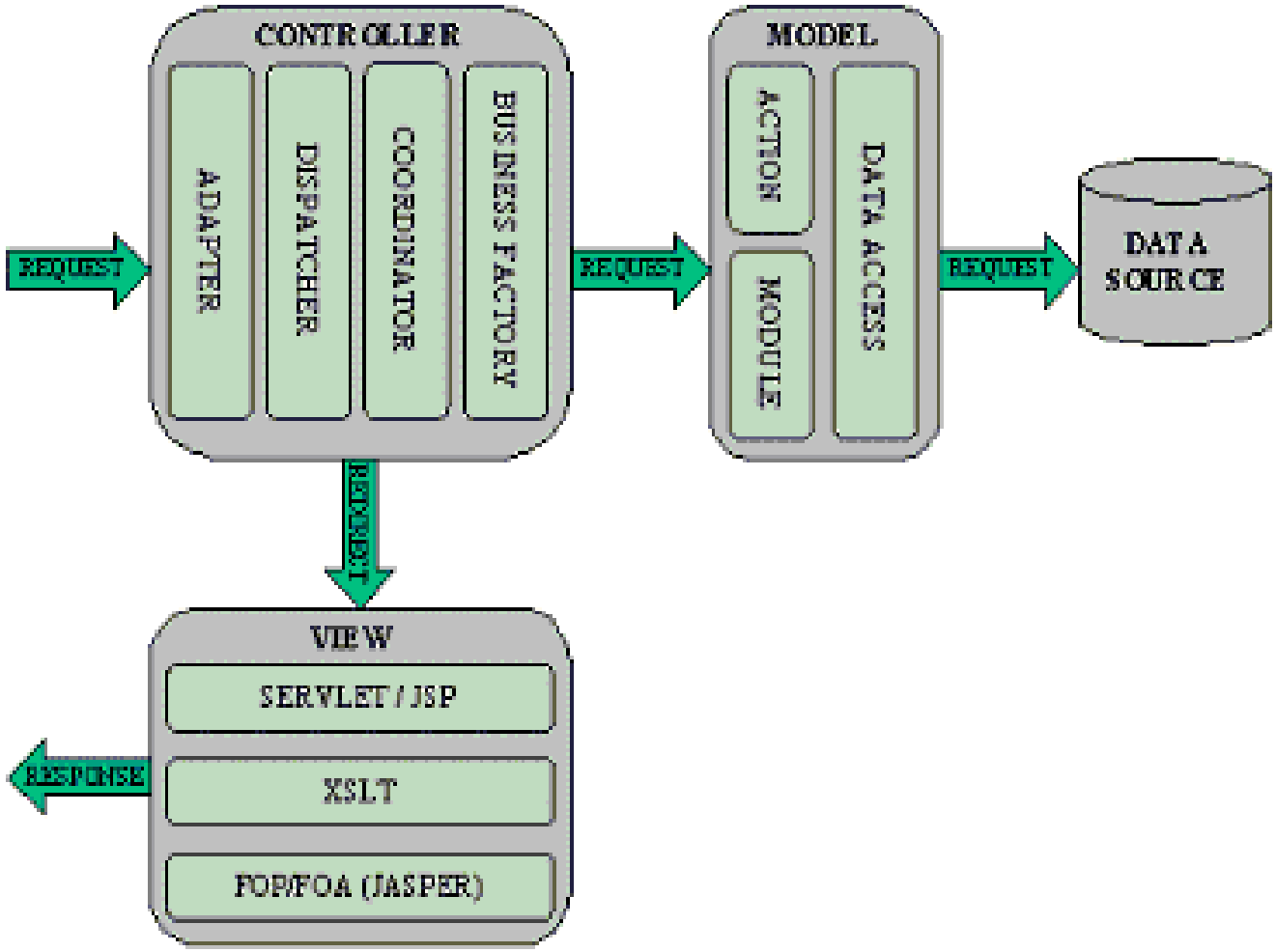
4. Gestore della navigazione: semplifica la gestione della navigazione

6. Paginazione: moduli di gestione della paginazione

7. XML : oggetti per la gestione efficiente di dati in formato XML

8. Validazione: servizio server side di validazione dei dati

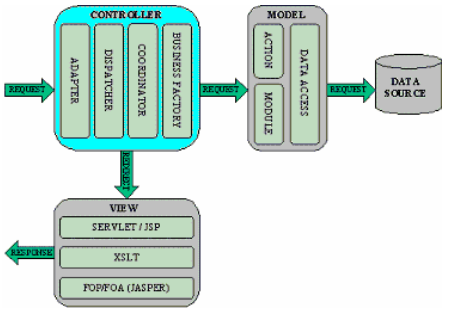
Architettura di Spago



Dispatching della richiesta (Controller)

Adapter

- Riceve i dati della richiesta da uno specifico canale
- Converte i dati delle richieste nel formato interno del framework (XML)
- Attiva il servizio;



Dispatcher

- Identifica la modalità di esecuzione della logica di business fra quelle supportate e ritorna il corretto coordinatore

Coordinator

- Coordina l'esecuzione della logica di business secondo una specifica modalità.

Pubblicazione multicanale (Adapter)

DEFINIZIONE:

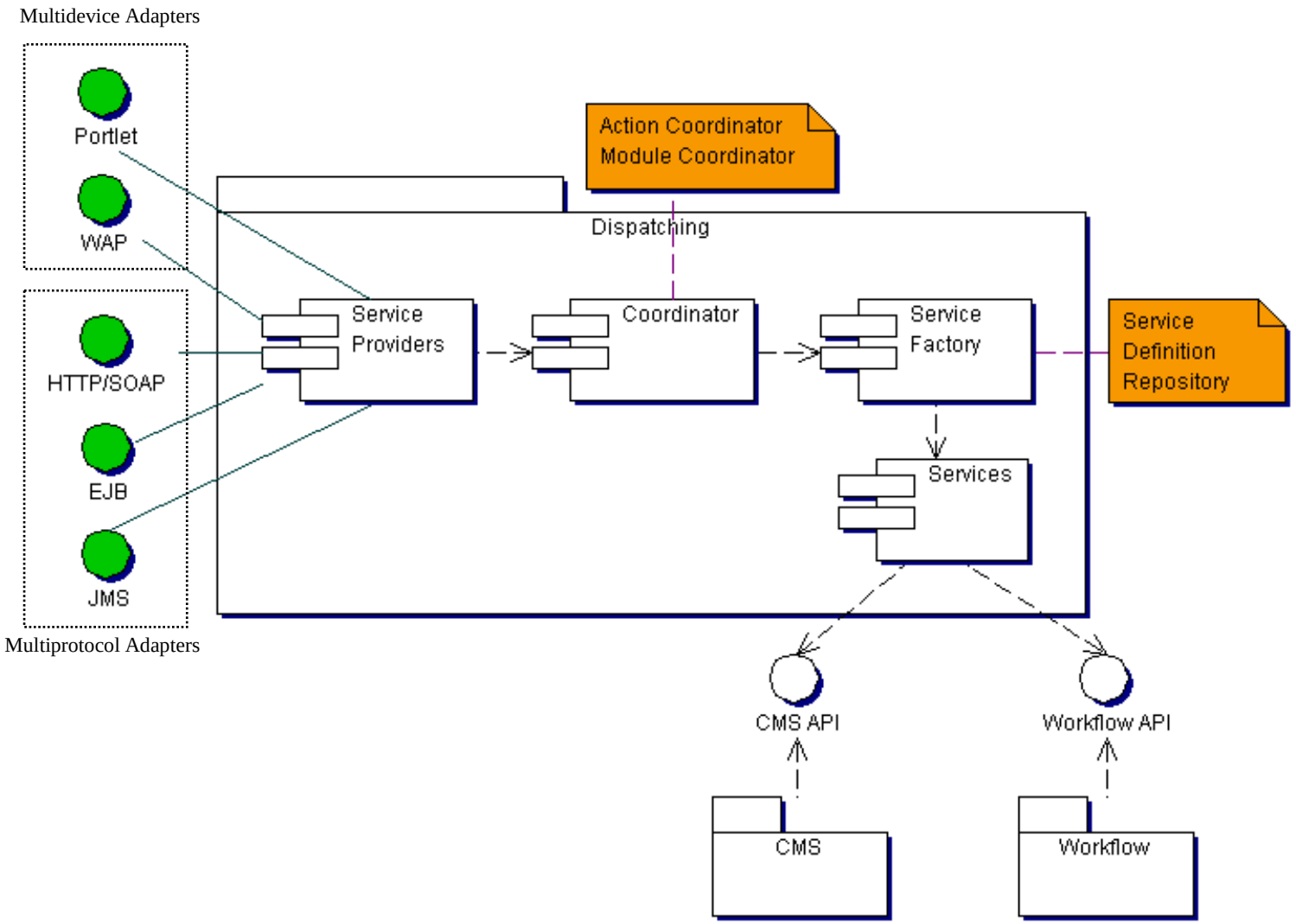
Per Multicanalità (in termini di device) si intende la capacità di un'applicazione di ricevere richieste attraverso più canali (quali Browser web, telefono cellulare WAP ecc.) e di presentare il contenuto della risposta in modo coerente con il canale attraverso il quale è stata effettuata la richiesta.

Pubblicazione multicanale (Adapter)

Un'applicazione J2EE, sviluppata con Spago, può erogare i suoi servizi su più canali (device) come HTTP e WAP, e attraverso più protocolli: HTTP, SOAP, EJB, JMS.

1. Le richieste possono provenire da più canali (web browser, mobile wap browser, etc)
2. La logica applicativa è indipendente dal contesto
3. La pubblicazione delle informazioni è coerente con il canale attraverso il quale è pervenuta la richiesta

Multicanalità - Schema



Architettura aperta

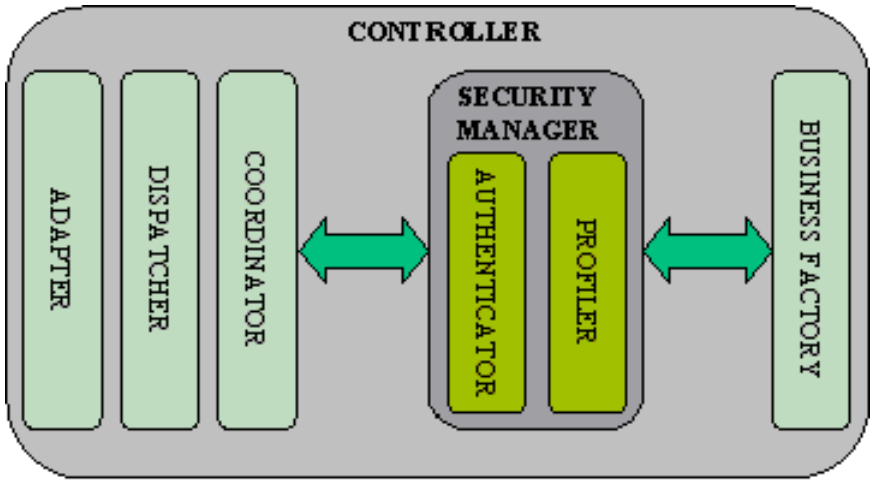
- L’architettura è aperta e permette di aggiungere nuovi canali oltre a quelli già implementati:

- Canali implementati
  - HTTP/WAP
  - SOAP
  - EJB
  - JMS
  - JBI
  - Portlet

Personalizzazioni del Framework

- Modulo “Security Manager”

Verifica i privilegi di esecuzione della logica applicativa associata ad un oggetto di business. La verifica delle autorizzazioni può essere abilitata su tutti gli oggetti di business o solo su alcuni di questi.



Strumenti di supporto (SourceBean)

- Alla base di tutto il framework c'è l'XML, tutti i sottosistemi di Spago comunicano tramite Flussi XML

- Il SourceBean è l'oggetto che in Spago rappresenta i flussi XML

- Ha capacità di navigazione con notazioni puntate.

Esempio: il SourceBean

```
<OUTER param3="value3">
    <INNER param1="value1" param2="value2">
    </INNER>
    <INNER param1="value3" param2="value4">
    </INNER>
</OUTER>
```

Una volta letto il frammento XML in un SourceBean è possibile, ad esempio, accedere a tutte le buste con chiave "INNER" tramite Statement

List inners = sourceBean.getAttributeAsList("INNER");

String param3 = (String) outer.getAttribute ("param3");

Modalità di dispatching

1. **Action**

Un singolo Oggetto di business per evadere la richiesta.

2. **Moduli**

Più oggetti di business che collaborano tra loro per evadere la richiesta.

E’ una modalità più complessa di quella ad Action, ma ha un alto potenziale di riuso dei moduli.

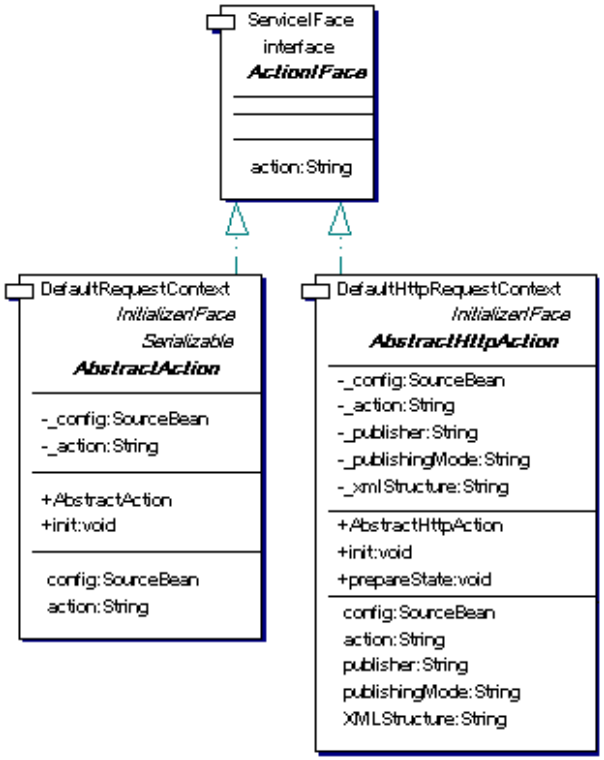
Modalità Action

1. Oggetti di business di questo tipo evadono per intero la richiesta del servizio
2. Esiste una corrispondenza univoca tra servizio e oggetto di business, mentre lo stesso oggetto di business può essere usato per evadere più richieste.
3. Ogni action è caratterizzata da uno scope che può essere:
   a) *Request*: istanzia nuova action ad ogni richiesta;
   b) *Session*: nella stessa session viene usata stessa istanza;
   c) *Application*: tutte le richieste indirizzate al medesimo container (JVM), viene usata stessa istanza;

Modalità Action

Esistono due modalità di implementazione:

1. Implementare ActionIFace
2. Estendere AbstractAction



In ogni caso è necessario Implementare il metodo "service":

*void **service** (SourceBean serviceRequest, SourceBean serviceResponse)*
throws Exception;

Metodo “service”

- La logica di business è implementata nel metodo

  void ***service*** *(SourceBean serviceRequest, SourceBean serviceResponse) throws Exception;*

- Tutte le action devono implementarlo

- Parametri del metodo:
  - *serviceRequest* per accedere ai parametri della richiesta
  - *serviceResponse* per valorizzare la risposta

Censimento Action

- Censimento di una action nel file “actions.xml”

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ACTIONS>
    <ACTION   name="LISTA_UTENTI_ACTION"
        class= "it.eng.spago.demo.action.ListaUtentiAction"
    scope="REQUEST">
            <CONFIG pool="afdemo" title="Lista Utenti" rows="2">
              ……..……..…..
            </CONFIG>
    </ACTION>
</ACTIONS>
```

- Nella sezione “config” possono essere censite informazioni utilizzabili in fase di inizializzazione della action. Ovvero nel motodo “init” della action

Modalità Moduli

1. Ogni module è un oggetto di business che collabora con altri per evadere la richiesta di un servizio

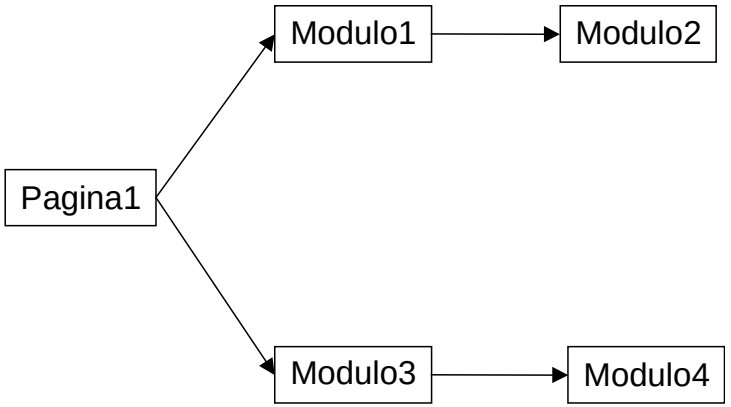
2. Tutti i module che collaborano tra di loro per lo stesso servizio formano un unità logica chiamata “*Page*”

3. La risposta del servizio è rappresentata dall’unione delle risposte di tutti i module

4. L’ordine e le condizioni sotto le quali i module vengono invocati è descritto tramite un “workflow”

Modalità Moduli

- La risposta ad una richiesta di servizio viene evasa da più moduli tra loro cooperanti e organizzati in un'unità logica chiamata PAGE.
- Una pagina è una composizione logica di moduli ovvero un grafo che specifica in che sequenza vanno invocati i moduli e con che parametri
- Il punto di inizio valutazione del grafo è la pagina



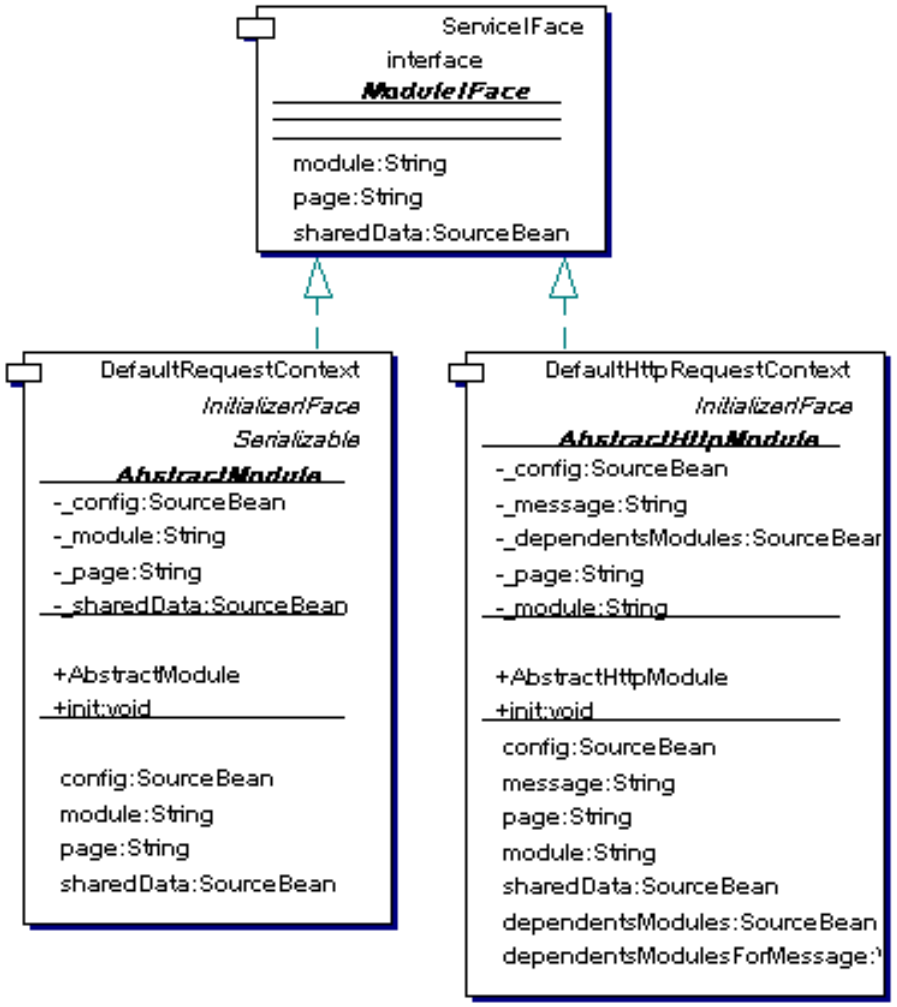
- I moduli vengono invocati nel seguente ordine: modulo1, modulo2, modulo3, modulo4.

Dispatching: Modalità Moduli

Esistono due modalità di implementazione :

1. Implementare ModuleIFace
2. Estendere AbstractModule

In ogni caso è necessario implementare il metodo “service”



Censimento Moduli

- Censimento dei moduli nel file “modules.xml”

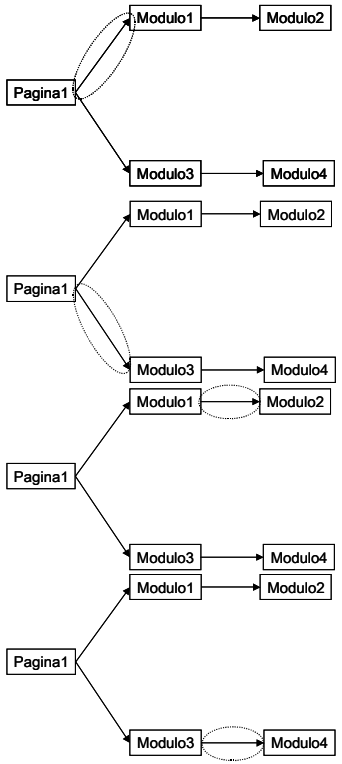
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<MODULES>
    <MODULE  name="ModuloCliente"
    class="it.eng.spago.demo.module.ModuloCliente"/>
    <MODULE  name="ModuloFornitore"
    class="it.eng.spago.demo.module.ModuloFornitore"/>
</MODULES>
```

- Non è necessario definire lo scope in quanto ereditano quello della pagina in cui sono inclusi

Censimento Pagine

La definizione della struttura della pagina è censita nel file pages.xml”

```
<PAGES><PAGE name="Pagina1" scope="SESSION">
    <MODULES>
      <MODULE name="Modulo1"/><MODULE name="Modulo2“ keep_response =“false”/>
      <MODULE name="Modulo3"/><MODULE name="Modulo4“ keep_istance=“true”/>
    </MODULES>
    <DEPENDENCIES>
      <DEPENDENCE source="Pagina1" target="Modulo1">
        <CONDITIONS/><CONSEQUENCES/>
      </DEPENDENCE>
      <DEPENDENCE source="Pagina1" target="Modulo3">
        <CONDITIONS/><CONSEQUENCES/>
      </DEPENDENCE>
       <DEPENDENCE source="Modulo1" target="Modulo2">
        <CONDITIONS/><CONSEQUENCES/>
      </DEPENDENCE>
      <DEPENDENCE source="Modulo3" target="Modulo4">
        <CONDITIONS/><CONSEQUENCES/>
      </DEPENDENCE>
    </DEPENDENCIES>
  </PAGE></PAGES>
```



Condizioni

- Ogni busta “<dependence>” rappresenta un arco

- La busta “<conditions>” consente di definire in quali casi un modulo deve essere invocato, tutte le “<conditions>” devono essere verificate affinché venga invocato un modulo

<CONDITIONS>
<PARAMETER name="nome parametro"
*scope="**USER** | **ADAPTER_REQUEST** | **SERVICE_REQUEST** | **SESSION** | **APPLICATION** | **ADAPTER_RESPONSE** | **SERVICE_RESPONSE** | **ERROR**"*
*value=" **AF_DEFINED** | **AF_NOT_DEFINED** | **...**"/>*
</CONDITIONS>

Conseguenze

- Sono i parametri aggiuntivi che vengono passati da un modulo al modulo target, consentono la cooperazione tra moduli

<CONSEQUENCES>

*<PARAMETER name="nome parametro" type="**ABSOLUTE** | **RELATIVE** scope=" **USER** | **ADAPTER_REQUEST** | **SERVICE_REQUEST** | **SESSION** | **APPLICATION** | **ADAPTER_RESPONSE** | **SERVICE_RESPONSE**"*
value="valore parametro"/>
</CONSEQUENCES>

- L'accesso ai parametri definiti nelle consequencies avviene tramite il SourceBean della richiesta

*void **service**(SourceBean serviceRequest, SourceBean serviceResponse) throws Exception {*
*String param = **serviceRequest.getAttribute**("user");…….*

Considerazioni

Vantaggi	Action	Moduli
Semplice da implementare	✓	
Flessibile e potente		✓
Consente riuso componenti		✓
Validazione dei dati	✓	✓

Logica di presentazione

- I contenuti prodotti dalla logica di business vengono rediretti al modulo di view che gestisce la pubblicazione

- La scelta della vista corretta in base al canale, ai parametri di risposta allo stato dell'applicazione è censita come parametro di configurazione

- Se non è stato configurato nessun publisher viene ritornata al client la rappresentazione XML dei contenuti

Presentazione

Il framework consente di scegliere tra diverse modalità di presentazione dei dati:

- XML/XSLT
- JSP
- Servlet
- Java
- Loop
- Nessuna

Non tutti le modalità sono adatte a gestire la pubblicazione per tutti i canali

Tabella modalità valide in base al canale

Channel	Type	Mode	Prog	resource
HTTP	NOTHING		0	
	SERVLET	FORWARD	0	Nome servlet
		SENDREDIRECT	0	Nome servlet
	JSP	FORWARD	0	Nome jsp
		SENDREDIRECT	0	Nome jsp
	XSL		“0,1,2,…”	Nome degli XSL da applicare
	LOOP		0	Parametri della nuova richiesta
WAP	NOTHING		0	
	SERVLET	FORWARD	0	Nome servlet
		SENDREDIRECT	0	Nome servlet
	JSP	FORWARD	0	Nome jsp
		SENDREDIRECT	0	Nome jsp
	XSL		“0,1,2,…”	Nome degli XSL da applicare
SOAP	NOTHING		0	
	XSL		“0,1,2,…”	Nome degli XSL da applicare
EJB	NOTHING		0	
	XSL		“0,1,2,…”	Nome degli XSL da applicare
JMS	NOTHING		0	
	XSL		“0,1,2,…”	Nome degli XSL da applicare

Configurazione

Per pubblicare la risposta, bisogna associare la action o la page, che implementa il servizio, al corrispondente publisher nel file *presentation.xml*

```
<PRESENTATION>
   <MAPPING
      business_name="LIST_USERS_ACTION"
      business_type="ACTION"
      publisher_name="ListUsersAction"/>
   ..............
<PRESENTATION>
```

Configurazione, continua

Gli oggetti di pubblicazione sono definiti nel file *publishers.xml*.

```
<PUBLISHERS>
   <PUBLISHER name="ListUsersAction">
      <RENDERING channel="HTTP" type="XSL" mode="">
         <RESOURCES>
            <ITEM prog="0" resource="
                  /WEB-INF/xsl/HTTPListUsersAction.xsl"/>
         </RESOURCES>
      </RENDERING>
      <RENDERING channel="WAP" type="XSL" mode="">
         <RESOURCES>
            <ITEM prog="0" resource="
                  /WEB-INF/xsl/WAPListUsersAction.xsl"/>
         </RESOURCES>
      </RENDERING>
   </PUBLISHER>
```

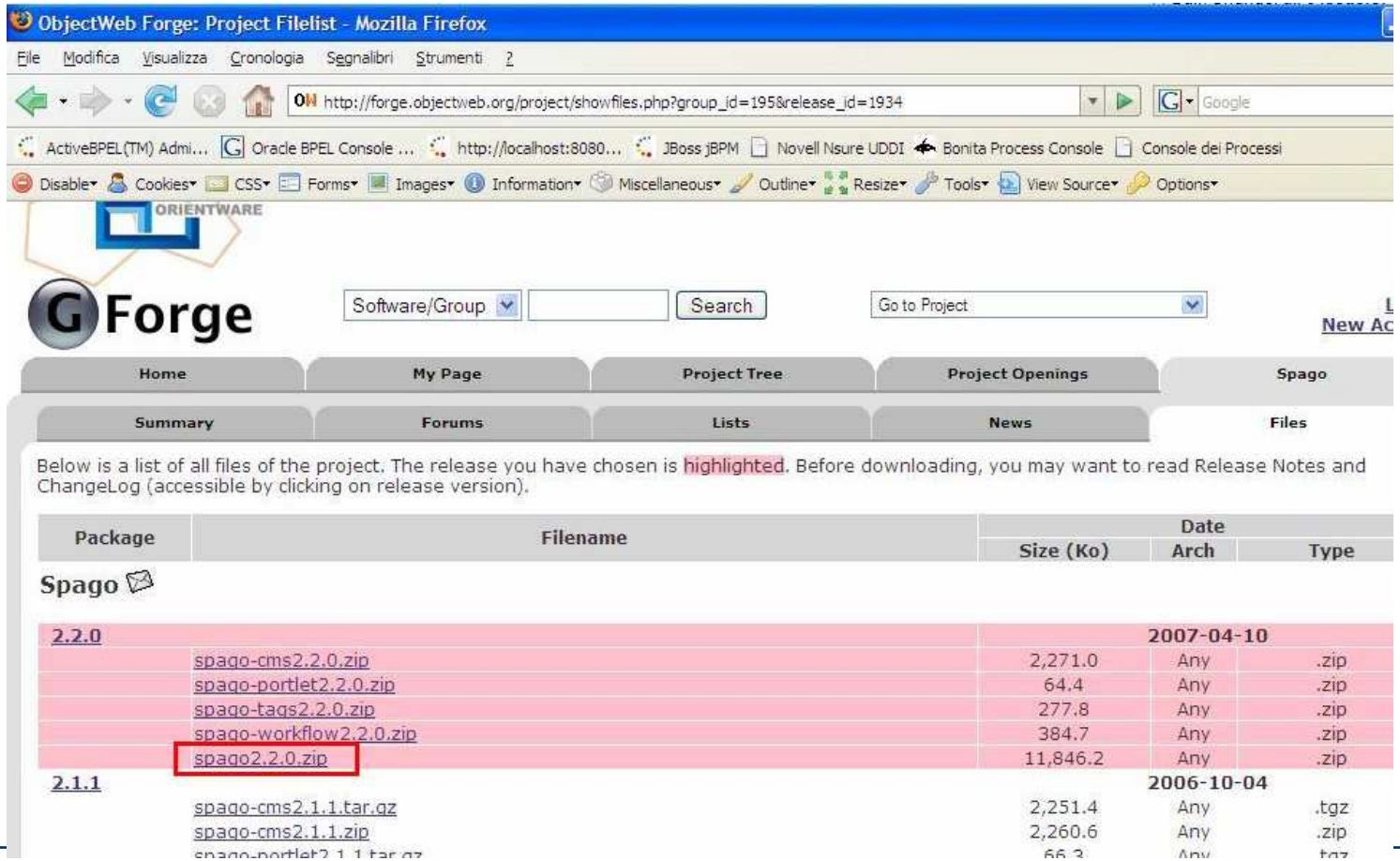
Esempio: prerequisiti

- JDK 1.5.x download from *http://java.sun.com*
- Tomcat 5.5.X download from **http://jakarta.apache.org/**
- Eclipse 3.2.0 download from **http://www.eclipse.org/**
- WTP 1.5 download from **http://www.eclipse.org/**
- Hsqldb 1.7.2 download from **http://hsqldb.sourceforge.net**

Cosa scaricare

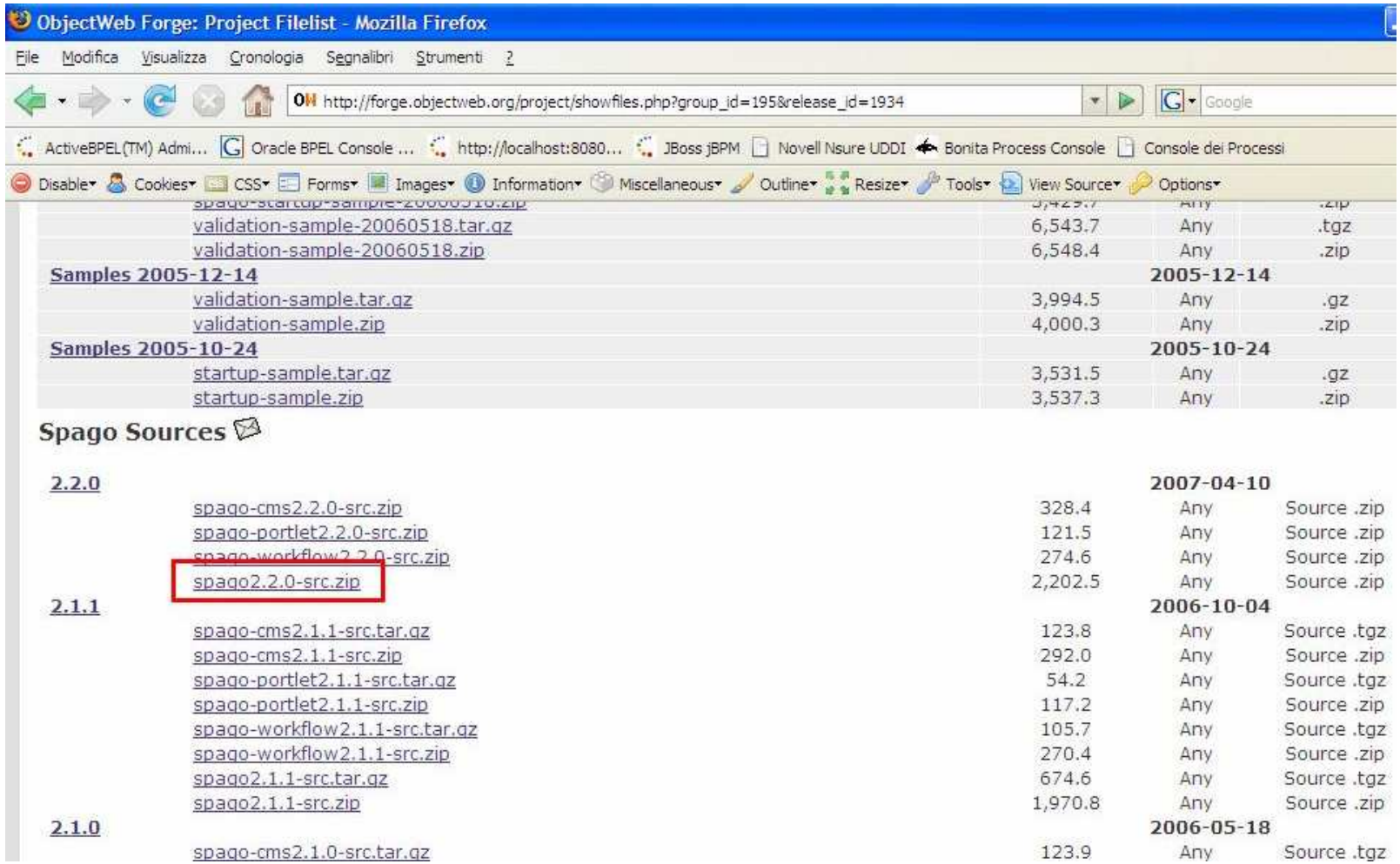
Dal sito di ObjectWeb (http://forge.objectweb.org) bisogna scaricare:

- La distribuzione binaria di Spago



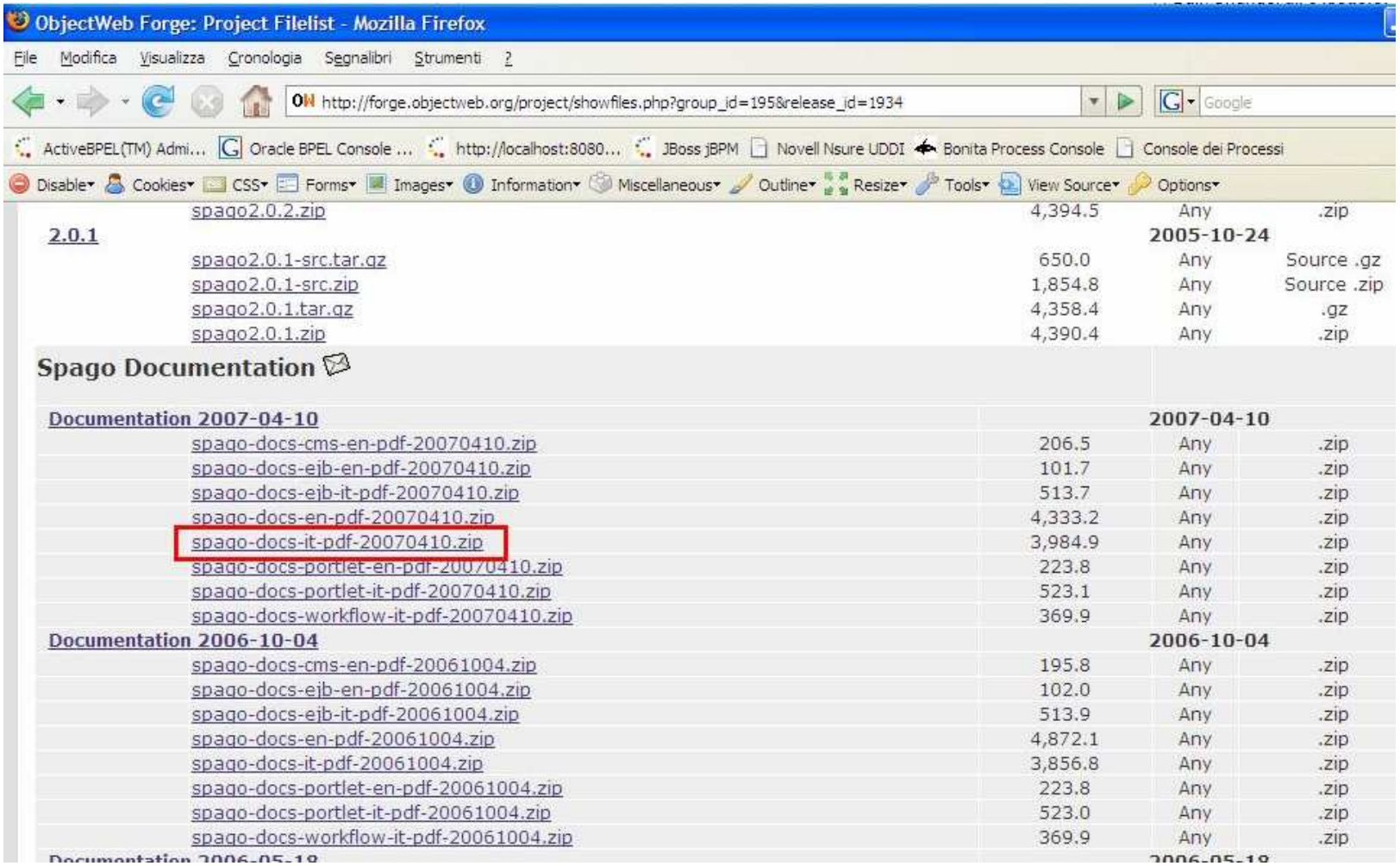
Cosa scaricare

- I sorgenti di Spago



Cosa scaricare

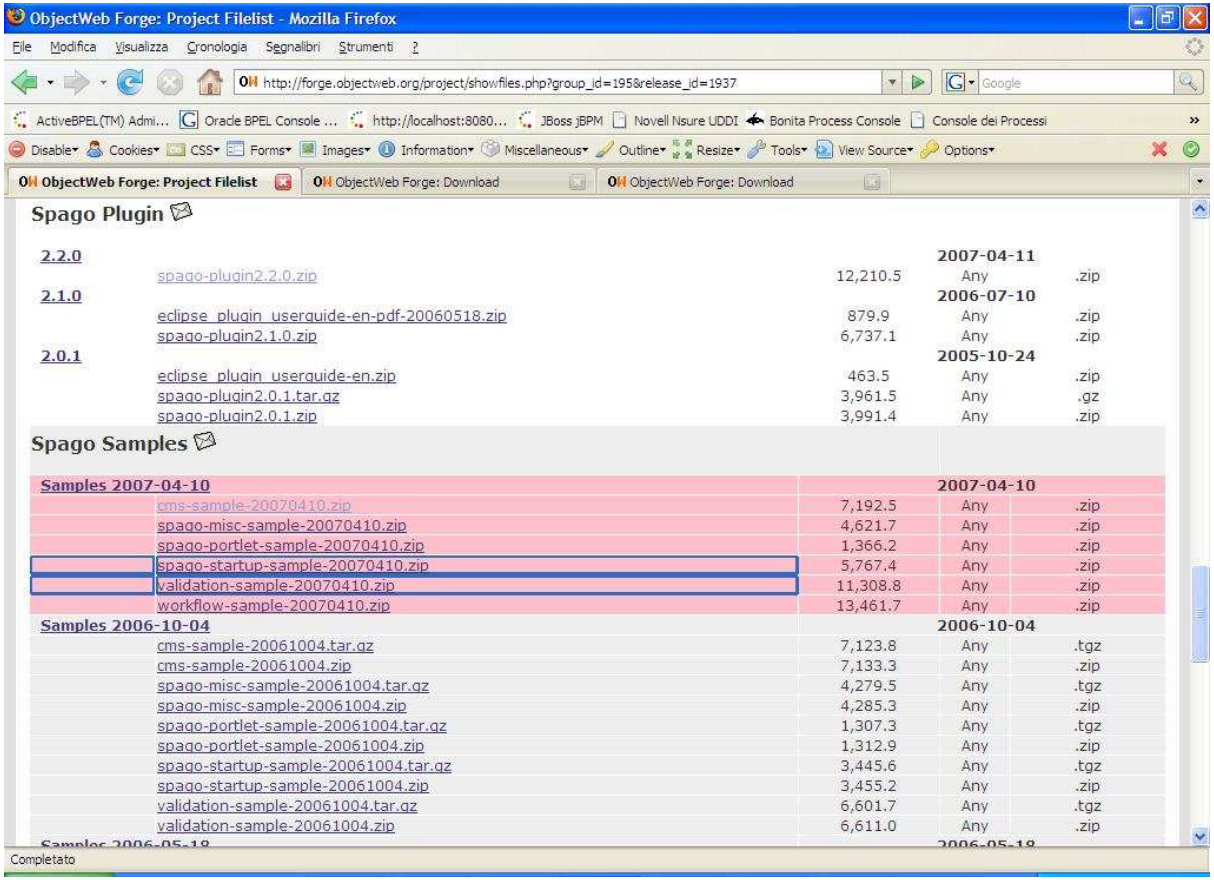
- La documentazione di Spago



Cosa scaricare

Opzionale:

- Il plugin per Eclipse
- Gli esempi



Esempio : Configurazione progetto

Step 1) Installare il plugin per Spago in Eclipse

Step 2) Avviare Eclipse

Step 3) Creare un nuovo "*Dynamic Web project*" chiamato "SpagoDemo"

Step 3) Dalla prospettiva "*Java*" invocare il comando "*Add Spago Nature*" sul progetto

Step 4) Invocare il comando "*Add Spago Tree*" sul progetto

Esempio: deploy

Step 1) Selezionare la propettiva "*Java*"

Step 2) Installare l'applicazione configurando Tomcat 5.5.X come server di default

Step 3) Avviare l'application server Tomcat appena configurato

Step 4) Aprire il browser e scrivere l'URL:

http://localhost:8080/SpagoDemoWeb/

Esempio: pubblicazione

Step 1) Creare la risorsa di pubblicazione (pagina JSP)

Step 2) Configurare il publisher nel file *publisher.xml*

Step 3) Associare l'action (o la page) con il publisher tramite il file *presentation.xml*

Step 4) Installare l'applicazione nel server

Modalità semplificata:

Step 1,2) Come I punti precedenti

Step 3) Configurare il nome del publisher direttamente nel file actions.xml

Servizi Trasversali

- Inizializzazione
- Tracing
- Gestione Errori
- Monitoring
- Gestione Configurazione
- Navigazione
- Gestione della sessione
- Paginazione
- Accesso ai dati

Inizializzazione

- Spago gestisce le attività di inizializzazione all'avvio dell'applicazione

- I processi di inizializzazione sono definiti nel file *initializer.xml*

```
<INITIALIZERS>
<INITIALIZER
class="it.eng.spago.dbaccess.DataAccessInitializer" config=""
/>
 </INITIALIZERS>
```

- Tutti gli inizializzatori devono implementare l'interfaccia *InitializerIFace*

Tracing

Consente la memorizzazione su file dell'attività applicativa, ad ogni messaggio viene associata una severity per gestire la registrazione dei soli messaggi il cui livello è definito nel file di configurazione "*Tracing.xml*".

Spago mette a disposizione 2 logger

1. DefaultLogger
2. Log4JLogger

DefaultLogger

Tale sistema consente di:

- Fare lo switch a un nuovo file a mezzanotte
- Specificare il livello minimo delle informazioni da tracciare
- Può tracciare sia stringhe che interi flussi XML

Log4JLogger

Tale sistema:

- Utilizza Log4J

Consente di:

- Registrare i log su destinazioni diverse
- Specificare il livello minimo delle informazioni da tracciare
- Può tracciare sia stringhe che interi flussi XML

Configurazione logger “Tracing.xml”

La configurazione del file di log viene effettuata tramite l’impostazione di parametri nel file “tracing.xml”

```
<TRACING>
<LOGGER name="DEFAULT_LOGGER"
    class="it.eng.spago.tracing.DefaultLogger">
    <CONFIG trace_min_log_severity="0" debug="true"
    trace_path="/demo/log/" trace_name="demo-" append="true"
    trace_thread_name="false" />
</LOGGER>
<LOGGER name=“Framework"
    class="it.eng.spago.tracing.DefaultLogger">
    <CONFIG trace_min_log_severity=“4" debug=“false"
    trace_path="/demo/log/" trace_name=" framework-"/>
</LOGGER>
<TRACING>
```

Come utilizzare i Tracer

```
TracerSingleton.log(Constants.NOME_MODULO,
                    TracerSingleton.DEBUG,
                     "AdapterHTTP::service: …..");
```

```
TracerSingleton.log(Constants.NOME_MODULO,
                    TracerSingleton.DEBUG,
                     "AdapterHTTP::service: …..“,
                      exception);
```

Gestione degli errori

Il framework fornisce uno stack di errori ognuno dei quali è caratterizzato da una severity e da una descrizione.
Una volta generato uno degli errori sopra descritti viene passato al gestore degli errori "**it.eng.spago.EMFErrorHandler**". Tale gestore mantiene la lista degli errori generati per la richiesta del servizio che può essere utilizzata dalla logica di presentazione.

```
<%@page extends
    "it.eng.spago.dispatching.httpchannel.AbstractHttpJspPage" %>
<%EMFErrorHandler errorHandler = getErrorHandler();
    Collection errors = errorHandler.getErrors();%>
```

Viene inoltre fornita una tagLib che accede allo stack degli errori

```
<%af:error>
```

Tipologie di errori previsti

Tipologie di errori gestite dal framework:

- **“it.eng.spago.error.EMFInternalError”**

  Viene usato per mappare errori di terze parti ad esempio una SQLException, (può contenere un oggetto, purché serializzabile)

- **“it.eng.spago.error.EMFUserError”**

  E’ un errore di tipo applicativo o un messaggio di natura informativa, contraddistinto da un codice identificativo, che consente la catalogazione multilingua della descrizione dell’errore.

EMFInternalError

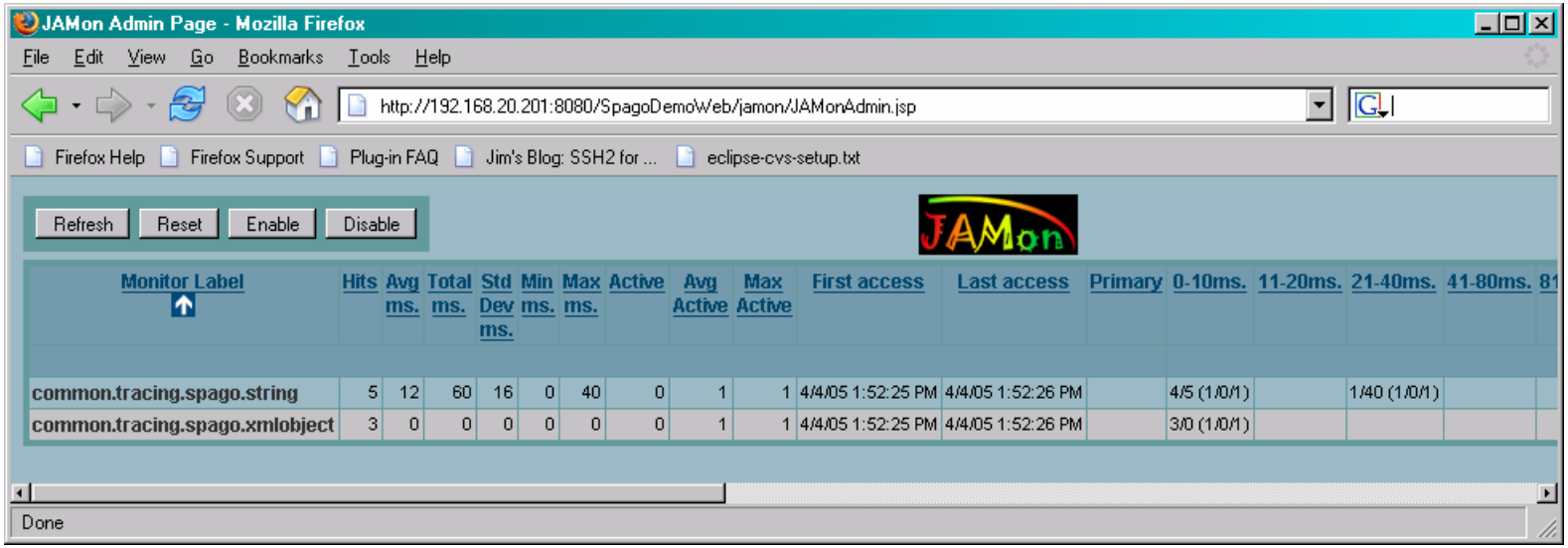
- Prodotto da componenti esterni al contesto
- Contraddistinti da una Severity, un messaggio, e una eccezione nativa
- Di solito non vengono pubblicati

EMFUserError

- Legati alla business logic o utilizzati per produrre messaggi
- Contraddistinti da una Severity, e un codice identificativo
- I messaggi sono definiti in un properties file
- Per messaggi dinamici si possono usare dei *PlaceHolder*

Monitoring: JAMon

Il prodotto Java Application Monitor "**JAMon**" (disponibile al sito http://www.jamonapi.com) è stato integrato all'interno del framework per eseguire il profiling delle attività di business di accesso ai dati e di presentazione.



Configurazione

Il Framework si basa su una serie di file di configurazione il cui elenco è censito nel file di configurazione indicato in web.xml

“web.xml”

….<init-param>
<param-name>AF_CONFIG_FILE</param-name>
<param-value>\WEB-INF\conf\spago**master.xml**</param-value>
</init-param> ….

“master.xml”

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<MASTER data_access_configuration_file_path="/WEB-
    INF/conf/spago/data_access.xml">
    <CONFIGURATOR path="/WEB-INF/conf/spago/common.xml" />
    <CONFIGURATOR path="/WEB-INF/conf/spago/dispatchers.xml" />
    ……
</MASTER>
```

Caricamento Configurazione

- La classe “ConfigSingleton” si occupa di caricare il file **master.xml** e tutti i file il cui path relativo sia in esso indicato.
- Nel caso in cui sia necessario un nuovo file di configurazione è sufficiente censirlo nel file “master.xml”
- ConfigSingleton è un SourceBean, quindi permette di accedere a tutte le informazioni presenti nel file.
- All’utente è trasparente il fatto che la configurazione sia memorizzata su più file

Esempio File di configurazione

actions.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ACTIONS>
   <ACTION name="LISTA_UTENTI_ACTION“
        class="it.eng.spago.demo.action.ListaUtentiAction"
        scope="REQUEST">
        <CONFIG>
        </CONFIG>
     </ACTION>
</ACTIONS>
```

Accesso ai dati di configurazione

Tramite lo stesso configuratore è possibile accedere ad informazioni presenti sui due diversi file riportati nella slide precedente:

```
ConfigSingleton config = ConfigSingleton.getInstance();
SourceBean actionBean =
    (SourceBean)config.getFilteredSourceBeanAttribute
    ("ACTIONS.ACTION","NAME","LISTA_UTENTI_ACTION");
String actionClassName = (String) actionBean.getAttribute("class");
```

Navigazione

- Il framework mette a disposizione un'oggetto "Navigatore" che memorizza in uno stack le richieste ed è in grado di rispondere alla richiesta di un servizio di tipo Back

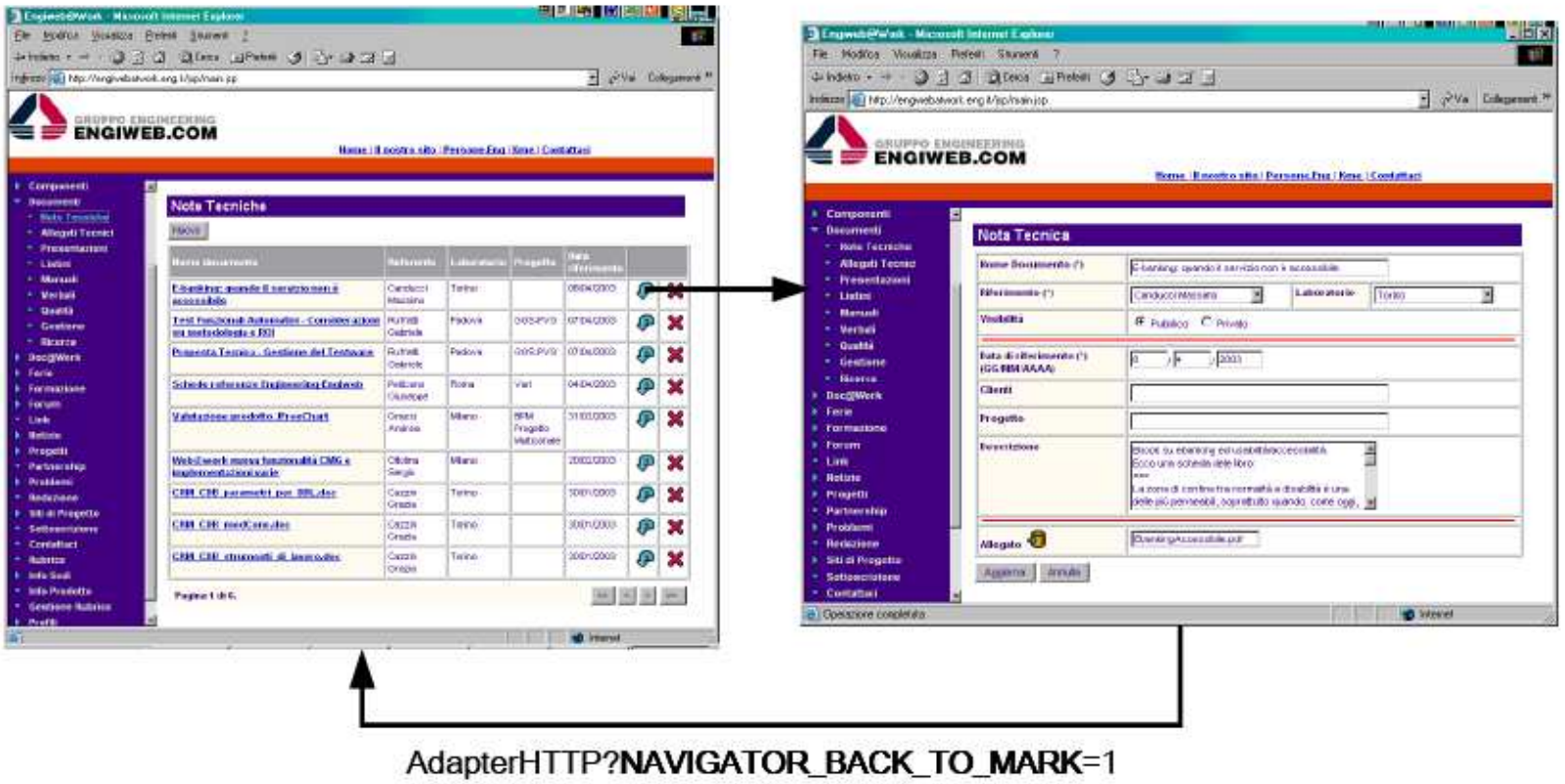
- Viene attivato impostando nell'url della richiesta determinate variabili
  - NAVIGATOR_BACK_TO_MARK
  - NAVIGATOR_BACK_TO_SERVICE …

- Fornisce inoltre due custom tag per la visualizzazione di una toolbar di navigazione:
  - It.eng.spago.tags.DefaultNavigationToolbarTag
  - It.eng.spago.tags.ComboNavigationToolbarTag

Navigazione

Considerate il seguente esempio:

Navigator.**signAction**(getRequestContainer());



Gestione Sessione

- SessionContainer: Oggetto che consente l'accesso ai dati presenti in sessione
- Es. recupero SessionContainer e informazioni in esso contenute

```
public void service(SourceBean request, SourceBean response) {
    RequestContainer requestContainer = getRequestContainer();
    SessionContainer sessionContainer =
    requestContainer.getSessionContainer();
    String userId = (String)sessionContainer.getAttribute("userId");
}
```

- L'oggetto SessionContainer implementa Serializable quindi la sessione può essere persistita su Data Base.
- L'oggetto SessionContainer può essere azzerato, tuttavia esiste un Container immune dai reset "PermanentContainer"

Multi-lingua

- Spago fornisce il supporto per la gestione multilingua delle interfacce utente

- Un catalogo per ciascuna lingua è definito nei properties file con il nome:

  messages_languagecode_countrycode.properties

  Esempi: *messages_en_US.properties*

  messages_it_IT.properties

Messages.properties

- Ogni messaggio è contraddistinto da un codice alfanumerico.

  EMPTY_LIST=List empty

  10002=Delete successfully

- Il codice è utilizzato per istanziare l'oggetto *EMFUserError*

  *EMFUserError(String severity, String **code**)*

Come utilizzare un linguaggio specifico

Vanno definiti nel *permanentContainer* della sessione i seguenti attributi:

- AF_LANGUAGE
- AF_COUNTRY

Esempio:

*permanentContainer.setAttribute("**AF_LANGUAGE**", "it");*

*permanentContainer.setAttribute("**AF_COUNTRY**", "IT");*

Accesso ai dati

- Il modulo di accesso ai dati fornito da Spago, aggiunge inoltre nuove funzionalità rispetto a quelle già fornite dal driver JDBC

- Tali funzionalità sono indipendenti dalla base dati. Questo consente di minimizzare lo sforzo nel caso in cui si renda necessario la migrazione ad una base dati diversa

- I risultati si possono trasformare in XML

Accesso ai dati: Funzionalità

1. Implementa lo standard SQL'92
2. Possibilità di utilizzare più database nella stessa applicazione
3. Indipendenza dal database
4. Indipendenza dal driver JDBC
5. Gestione di pool nativi e gestiti

Accesso ai dati: Funzionalità

6. Tutte le operazioni SQL effettuate all'esterno di una transazione sono autocommittanti

7. Gestione manuale o automatica delle transazioni

8. Conversione automatica degli oggetti di tipo Date o Timestamp in String e viceversa

Gestione delle connessioni

Spago mette a disposizione la classe "*DataConnectionManager*" che consente di:

- Creare e configurare pool;
- Fornire connessioni sui vari tipi di pool gestiti;

```
DataConnection dcDefault =
    DataConnectionManager.getInstance().getConnection();
DataConnection dcOracle =
    DataConnectionManager.getInstance().getConnection("oracle");
DataConnection dcDB2=
    DataConnectionManager.getInstance().getConnection("DB2");
……
dcDefault.close();
dcOracle.close();
dcDB2.close();
```

Configurazione Pool

I pool di connessioni sono censiti nel file “*data_access.xml*”

```
<DATA-ACCESS>
  <DATE-FORMAT format="DD-MM-YYYY"/>
  <TIMESTAMP-FORMAT format="DD-MM-YYYY hh:mm:ss"/>
  <CONNECTION-POOL connectionPoolName="demo"
    connectionPoolFactoryClass="it.eng.spago.dbaccess.factory.OracleConnectionPoolDat
    aSourceFactory" connectionPoolType="native">
    <CONNECTION-POOL-PARAMETER parameterName="connectionString"
    parameterValue="jdbc:oracle:thin:@xxx.xxx.xxx.xxx:xxxx:XXX" parameterType=""/>
    <CONNECTION-POOL-PARAMETER parameterName="jdbcDriver"
    parameterValue="oracle.jdbc.OracleDriver" parameterType=""/>
……..……..
</CONNECTION-POOL>
  <CONNECTION-MANAGER>
    <REGISTER-POOL registeredPoolName="demo"/>
  </CONNECTION-MANAGER>
</DATA-ACCESS>
```

Censimento Statement

Gli statements possono essere censiti nel file "statements.xml". L'accesso a tale file è possibile tramite la classe "SQLStatements"

String statement = SQLStatements.getStatement("LISTA_UTENTI");

Ecco un esempio di tale file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<statements>
    <statement>
        name="LISTA_UTENTI"
        query="SELECT userid,nome,cognome FROM utenti ORDER BY
  userid"
    < /statement>
</statements>
```

Esecuzione di comandi SQL

```
//
// You've already obtained the DataConnection dc Object
//
String statement = SQLStatements.getStatement("USERS_LIST");
SQLCommand cmdSelect = dc.createSelectCommand(statement);
List inputParameter = new ArrayList();
parameters.add(dataConnection.createDataField(
                                "column_name",
                                Types.VARCHAR,
                                paramStr));
DataResult dr = cmdSelect.execute(inputParameters);
```

Lettura del risultato

```
dataResult = sqlCommand.execute(parameters);
scrollableDataResult = (ScrollableDataResult)
                                        dataResult.getDataObject();
ArrayList codiciTipoModuli = new ArrayList();
while (scrollableDataResult.hasRows())
{
    DataRow dataRow = scrollableDataResult.getDataRow();
    codiciTipoModuli.add(
        dataRow.getColumn(S_CODTIPMOD).getStringValue());
} // while (scrollableDataResult.hasRows())
```

Validazione Dati

Spago fornisce un componente per la validazione server-side dei dati dalle seguenti caratteristiche:

- Opzionale
- Configurabile (sia per i moduli che per le action)
- Bloccante o non Bloccante

Tipi previsti dalla Validazione

1. Validazione Automatica dei tipi più comuni

2. E’ possibile implementare delle classi Java che effettuano delle validazioni custom

3. Template di validazione: è possibile creare dei template dei tipi di validazione

Template di Validazione

1. Definizione del template:

```
<FIELD-VALIDATORS>
    <FIELD-VALIDATOR fieldType="DATE1"
    fieldValidatorClass="it.eng.spago.validation.fieldvalidators.DateValid
    ator">
        <CONFIG maxLength="30">
            <DATE-FORMAT dateFormat="dd/MM/yyyy"/>
        </CONFIG>
    </FIELD-VALIDATOR>
</FIELD-VALIDATORS>
```

Template di Validazione

2. Utilizzo del template:

```
<SERVICE name="TEST_VALIDATION" type="ACTION">
<VALIDATION blocking="true" validators="">
    <FIELDS>
        <FIELD name="ABC" maxLength="30"  type="DATE1"
    aliasAfterValidation="Data2"/>
        <FIELD name="DEF" maxLength="30"  type="DATE1"
    aliasAfterValidation="Data2"/>
    </FIELDS>
</VALIDATION>
</SERVICE>
```

Validazione Automatica Configurazione

```
<VALIDATIONS>
 <SERVICE name="insertEmployee" type="ACTION">
  <VALIDATION blocking="true" validators="">
   <CONDITIONS>
        <PARAMETER name="field1" scope="SERVICE_REQUEST"
                value="xxx" />
   </CONDITIONS>

   <FIELDS>
      <FIELD name="field1" maxLength="10" mandatory="true" type="8" />
      <FIELD name="field2" maxLength="5" mandatory="false" type="6" />
   </FIELDS>
  </VALIDATION>
 </SERVICE>
 ……..
</VALIDATIONS>
```

Validatori Java

- Permettono di effettuare validazioni complesse
- Implementano l'interfaccia *AbstractValidator*:
  - ***check****(SourceBean request, EMFErrorHandler errorHandler)*

    Controlli formali
  - ***validate****(SourceBean request,*

    EMFErrorHandler errorHandler)

    Controlli incrociati

Messaggi di Validazione

I messaggi degli errori di validazione sono personalizzabili in diversi modi:

1. E’ possibile cambiare i messaggi per tutti gli errori
2. E’ possibile modificare i messaggi d’errore solo per un specifico servizio
3. E’ possibile modificare i messaggi d’errore per uno specifico campo

Paginazione

L'oggetto "Paginatore" ha la responsabilità di paginare collezioni di oggetti, delegando al programmatore la responsabilità di recuperarli

Spago fornisce 2 implementazioni del paginatore :

1. **One-Shot**
2. **with Cache**

One-shot

- L’intera collezione di elementi da visualizzare è reperita in un’unica soluzione e mantenuta in memoria
- La paginazione viene effettuata solo a livello di presentazione
- Soluzione semplice
- Possibili problemi di occupazione di memoria e di tempi di esecuzione

Con Cache

- La paginazione viene effettuata anche a livello di recupero dei dati, gestendo una “finestra” sui dati che vengono visualizzati
- I dati possono essere reperiti da qualunque datasource
- Più complessa da utilizzare
- Non ha problemi di tempi di esecuzione
- Più efficiente

Generazione Automatica di liste e dettagli

- Spago implementa un meccanismo automatico per costruire una lista paginata
- Gli sviluppatori devono solo configurare alcuni file XML
- Le operazioni di rendering possono essere delegate sia ad alcuni tag di Spago che a tag implementati dall'utente
- Si possono utilizzare sia i moduli che le action

Integrazione con sistemi di sicurezza

1. Spago fornisce le interfacce per l'integrazioni di sistemi esterni di profilatura
2. Spago fornisce un'implementazione di esempio basata su file di configurazione XML

In tutti i casi, la seguente interfaccia deve essere implementata:

*it.eng.spago.security.**IEngUserProfile***

IEngUserProfile

L'interfaccia *IEngUserProfile* definisce i metodi per accedere alle informazioni dell'utente:

- Identificatore utente
- Attributi utente
- Ruoli utente e funzionalità
- Diritti di esecuzione di un'action o un modulo

Processo di Login

L'oggetto che implementa l'interfaccia *IEngUserProfile*, deve essere memorizzato nel *PermanentContainer* (in sessione) con uno specifico nome di attributo:

permanentContainer.setAttribute(

IEngUserProfile.ENG_USER_PROFILE, *userProfile);*

Funzionamento del modulo di Sicurezza

Spago verifica le autorizzazioni dell'utente ad ogni richiesta di servizio.

Se l'utente non ha i diritti di esecuzione sul servizio:

- Action: redirezione al publisher SECURITY_ERROR_PUBLISHER
- Moduli: il modulo non viene eseguito ma l'esecuzione della pagina prosegue

Riferimenti

- **http://spago.objectweb.org**
- **http://spago.eng.it**
- **Mailing list: spago-dev@objectweb.org**
- **My address: gianfranco.boccalon@eng.it**

Esercitazioni

Le esercitazioni consistono nel contribuire, tramite invio di codice, a estensioni di Spago o correzioni di errori conosciuti.

Tutte le richieste di miglioramento e di correzione sono censite nel tracking system (JIRA) di Spago, a cui si accede previa registrazione (libera).

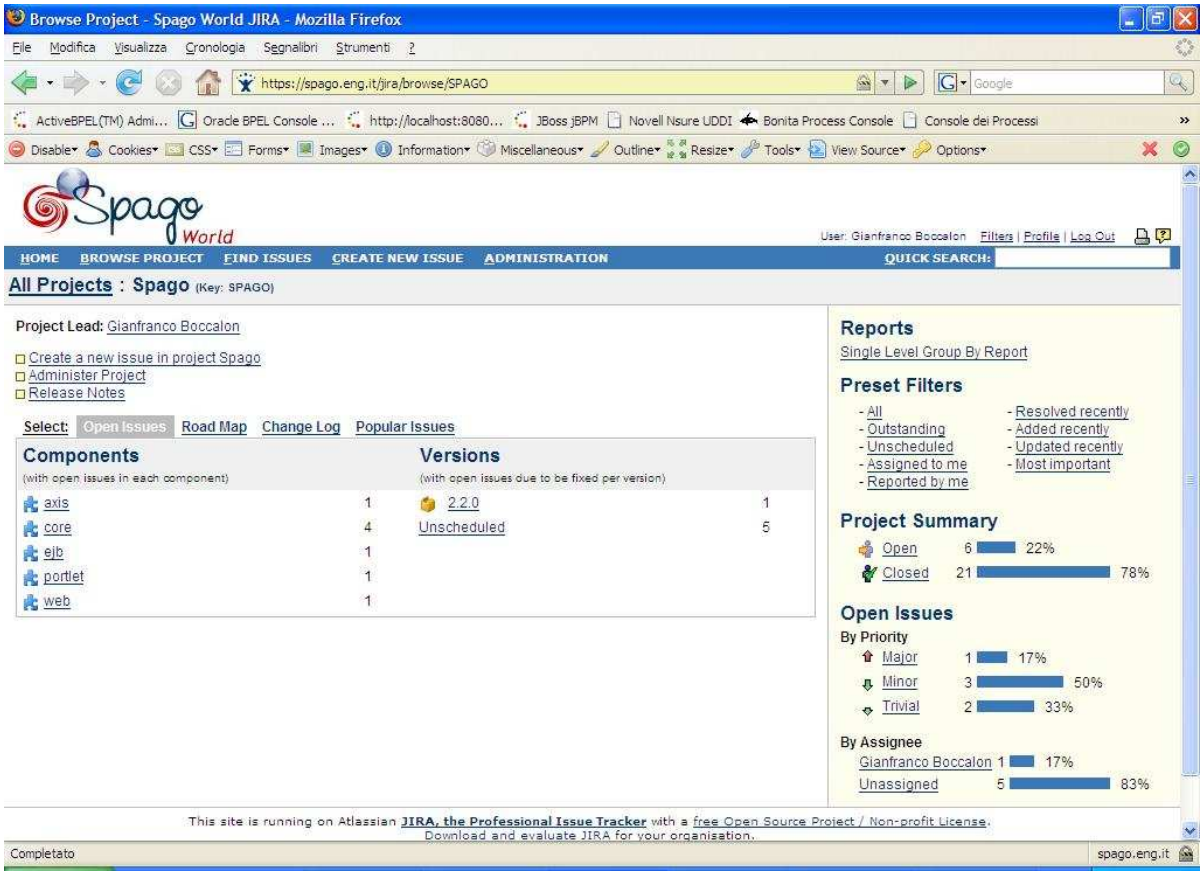
Tracking System

Il tracking system di Spago è accessibile all'URL http://spago.eng.it, sotto la voce Community->Tracker.

Il sito è accessibile dopo aver effettuato il login.

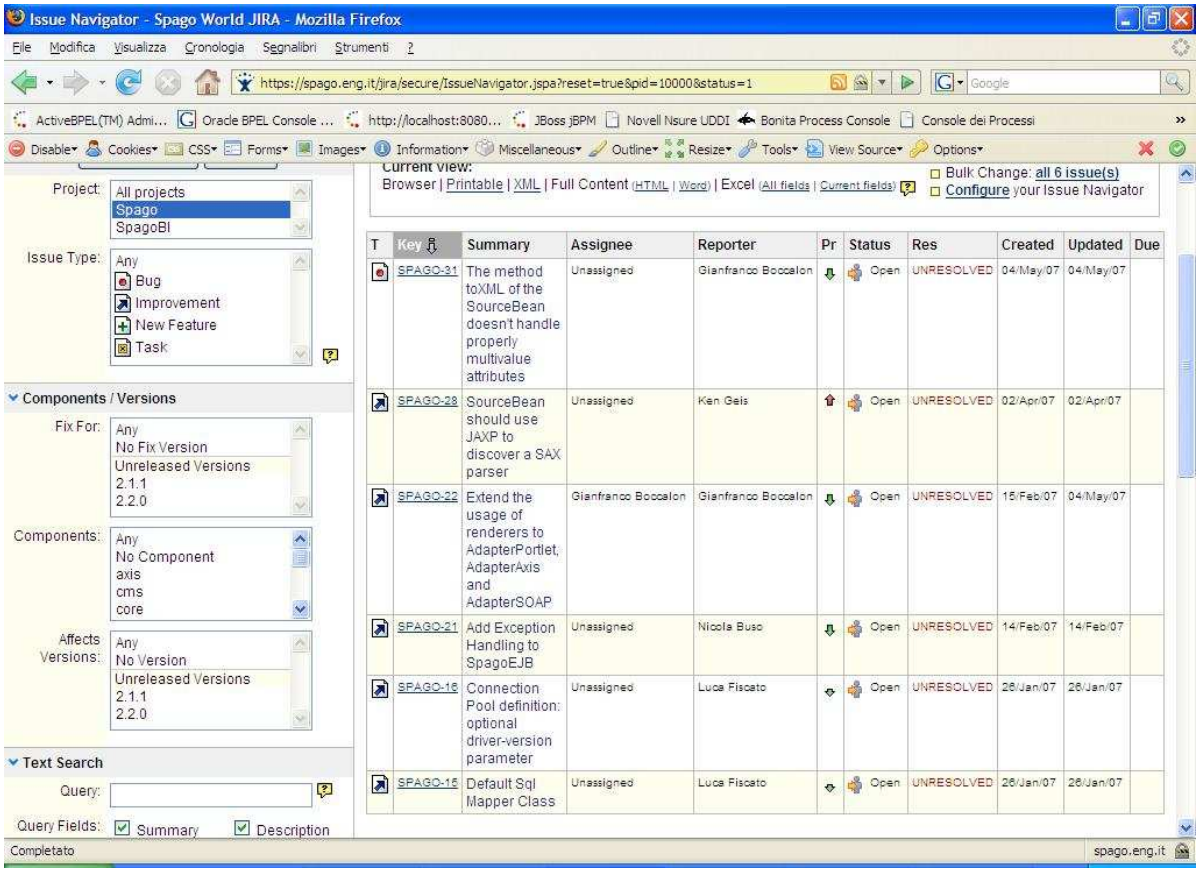
Tracking System

Nel tracking system sono censite tutte le richieste di miglioramento o di correzione. Non tutte le richieste sono oggetto delle esercitazioni.



Esercitazioni

Le possibili esercitazioni sono censite sotto la voce Project Summary->Open, il che significa che sono chiari i requisiti ma non è ancora stata assegnata l'attività.



Esercitazioni

Le possibili esercitazioni sono le seguenti:

• SPAGO-31: The method toXML of the SourceBean doesn't handle properly multivalue attributes (molto complessa)

• SPAGO-28: SourceBean should use JAXP to discover a SAX parser (mediamente complessa)

• SPAGO-15: Default Sql Mapper Class (mediamente complessa)

• SPAGO-16: Connection Pool definition: optional driver-version parameter (molto semplice)

Esercitazioni

Deve essere inviata la patch contenente il codice della contribuzione: la patch è un file di testo creato automaticamente che contiene le differenze tra il codice “ufficiale” e il codice contenente la contribuzione.
Viene creata con gli strumenti dell’ambiente di sviluppo.

I commenti nella contribuzione devono essere in inglese.

Se possibile deve essere mantenuta la retrocompatibilità della contribuzione con il codice esistente sviluppato con Spago.