

Spago Workflow Guide

Redatto da Andrea Zoppello

Scopo del Documento	3
Informazioni sulla versione	3
1 Introduzione	4
2 Concetti Preliminari	4
3 Architettura	6
4 Spago Workflow API – Utilizzo e Casi d’uso	7
4.1 Inizializzazione del Motore di Workflow, Connessione ed autenticazione.	7
4.2 Amministrazione delle definizioni di processo	8
4.3 Gestione dei processi	10
4.4 Gestione delle attività	11
5 Componenti di WorkList e Generazione Automatica di Form	13
6 Implementazione di SpagoWorkflowAPI su Shark workflow engine	15

Scopo del Documento

In questo documento vengono presentati i componenti di Spago per l'interfacciamento con i motori di workflow.

Informazioni sulla versione

Versione/Release n° :	1.0	Data Versione/Release :	11/01/2006
Descrizione modifiche:	Versione iniziale		

1 Introduzione

La crescente complessità ed eterogeneità dei sistemi informativi e delle applicazioni ad essi collegate, rende necessario una maggiore attenzione sulle problematiche di “integrazione” e di definizione di processi di business complessi che coinvolgano diverse applicazioni e diverse tecnologie anche diverse tra di loro.

Per queste ragioni in spago è stato sviluppato un componente che permette di integrare nel framework un “sistema di workflow”.

In questo documento verranno riassunti brevemente alcuni concetti preliminari sui workflow, e in seguito vedremo quali sono i componenti sviluppati su spago per introdurre i concetti del workflow.

2 Concetti Preliminari

Una buona definizione di **workflow** può essere la seguente [Mokabyte]:

“Si può definire un Workflow come l’automazione di un processo di business, in tutto o solo in parte, durante il quale i documenti, le informazioni o i compiti sono passati da un partecipante a un altro per compiere una determinata azione secondo quanto specificato da un insieme di regole procedurali ben definite. Un workflow costituisce quindi un flusso di lavoro composto da un insieme di attività correlate tra loro attraverso diverse tipologie di relazioni.”

In termini formali possiamo quindi dire che un workflow è la definizione di un processo di business in termini di attività, che coinvolge degli attori (persone, gruppi di persone, sistemi esterni).

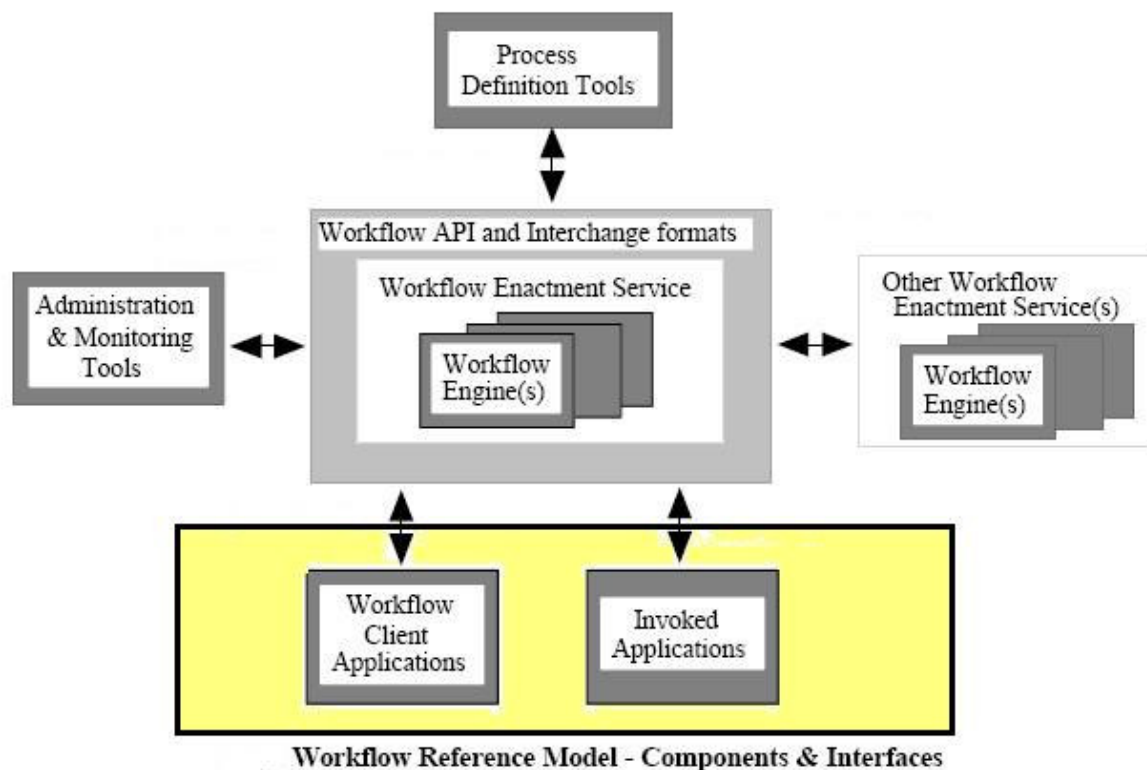
Per definire in termini precisi un workflow è quindi necessario:

- Identificare le **attività** e assegnare queste attività agli attori.
- Descrivere il **flusso del processo**, in termini di **attività** e di **transizioni** (la transizione definisce come avviene il passaggio da una attività ad un'altra).
- Identificare le informazioni che rappresentano lo **stato del processo** (o Contesto del processo).

Una volta definito precisamente il workflow possiamo definire un **Workflow Management System (WFMS)** un sistema automatico che mi permetta di:

- Creare Modificare e Cancellare Workflow su di un repository.
- Istanziare ed eseguire processi (descritti dai workflow) su di un motore (Workflow Engine).
- Permetta di monitorare lo stato dei processi in esecuzione e di intervenire attraverso opportuni tools.

Il modello di workflow definito dal WfMC (Workflow Management Coalition) è riassunto dalla figura che segue:



Al centro si trova il WFMS mentre sui quattro lati si trovano essenzialmente le tipologie di applicazioni sistemi che interagiscono con esso e che possiamo riassumere nelle seguenti categorie:

1. Applicazioni per la definizione dei processi
2. Applicazioni di amministrazione e monitoring dei processi in esecuzione
3. Altri sistemi di workflow
4. Applicazioni Client che utilizzano le api degli engine per implementare le logiche applicative.

Lo schema appena descritto è uno schema assolutamente generale. Su questo argomento esistono diversi standard che i prodotti commerciali e/o open source possono o meno supportare.

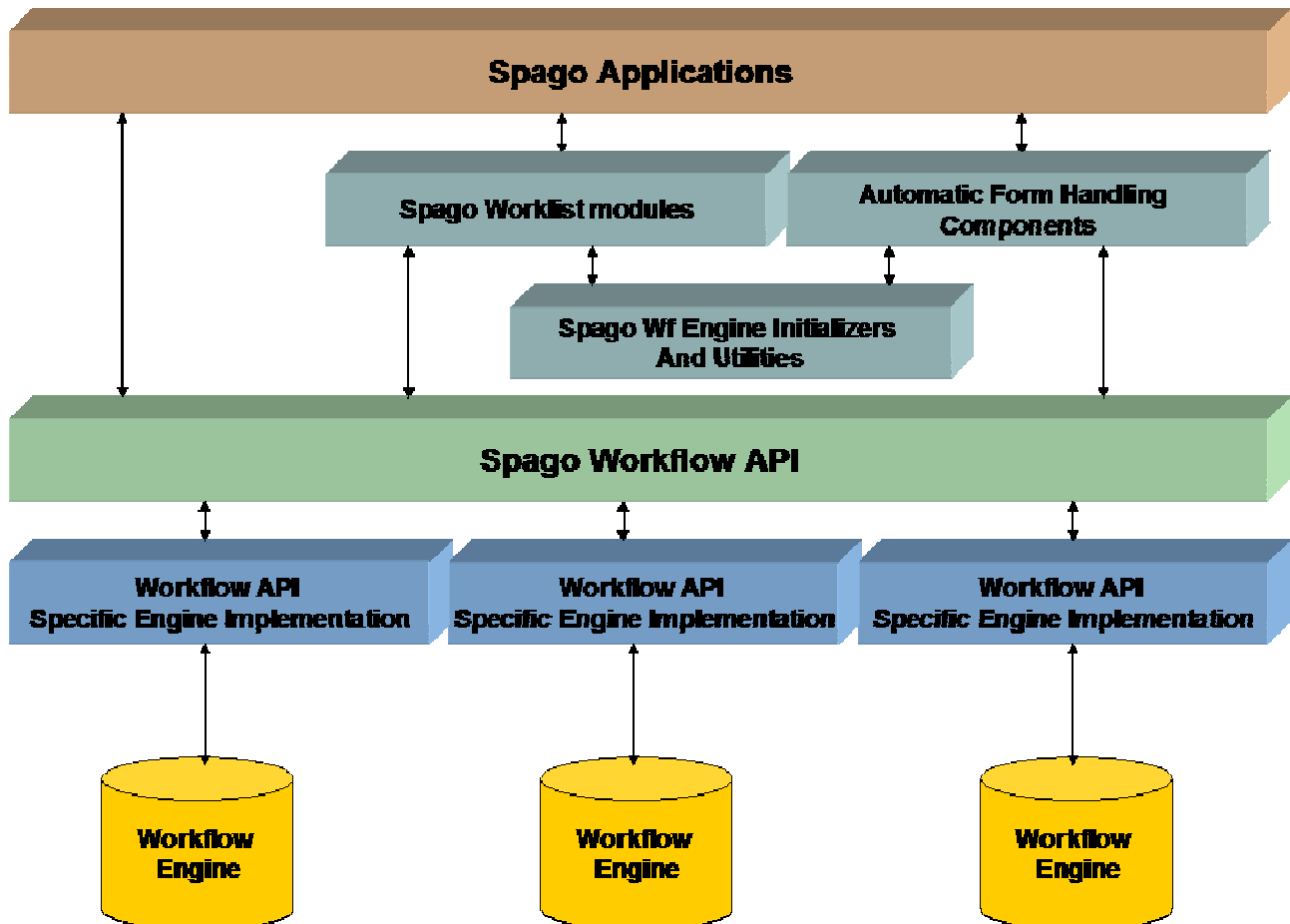
In un tale contesto i componenti sviluppato per spago rientrano nella categoria delle applicazioni che sono client di un workflow, prevedono quindi:

- Un insieme di API (Per la connessione al sistema di workflow, l'autenticazione, e la gestione dell'intero ciclo di vita del processo dalla sua creazione, fino alla gestione dello stato etc.. etc..)
- La realizzazione di alcuni componenti basati su Spago, per la realizzazione di applicazioni WEB che abbiano la necessità di fornire agli utenti delle funzionalità di workflow.

- La realizzazione di componenti che sulla base di informazioni associate alle attività di un workflow siano in grado di presentare in modo automatico dei form web all'utente.

3 Architettura

La figura seguente riassume, schematicamente i componenti di spago per l'integrazione di motori di workflow:



In particolare si hanno i seguenti componenti:

- **SpagoWorkflowAPI** : E' un layer di interfacce tra spago e i diversi motori di workflow, le applicazioni e i componenti spago che necessitano di interfacciarsi verso un motore di workflow utilizzano esclusivamente le interfacce presenti in questo layer senza utilizzare api di un particolare motore di workflow. Questo permette di realizzare in spago delle applicazioni che siano totalmente indipendenti dallo specifico motore di workflow.
- **Workflow API Specific Engine Implementations**: Sono dei layer di software che implementano le api di workflow di spago per interfacciare le varie implementazioni dei motori di workflow. Attualmente è presente l'implementazione per il motore di workflow open source shark (<http://shark.objectweb.org>).
- **SpagoWorkflowEngineUtilities And Initializers**: Componenti spago per l'inizializzazione che provvedono a rendere disponibili alle applicazioni ed ai componenti spago gli oggetti per poter effettuare le varie operazioni sul motore.

- **SpagoWorkList Modules:** Moduli di spago per la realizzazione di funzionalità di tipo worklist, in cui l'utente abbia a disposizione le attività pendenti per la quale è un possibile candidato all'esecuzione. Questi componenti di spago forniscono oltre ai moduli i servizi applicativi per l'accettazione, il rifiuto e il completamento delle attività.
- **Automatic Form Handling Modules:** Componenti di spago per l'associazione di particolari pagine alle attività del workflow sulla base degli attributi dell'attività. Questi componenti permettono anche la generazione di pagine con form descritti in un file di configurazione XML.

4 Spago Workflow API – Utilizzo e Casi d'uso

In questo paragrafo descriveremo l'utilizzo di Spago Workflow Api andando a descrivere i casi d'uso di più comune utilizzo.

4.1 Inizializzazione del Motore di Workflow, Connessione ed autenticazione.

Per poter utilizzare la api di workflow di spago la prima operazione da fare, e quella di creare un'istanza di un oggetto di tipo **it.eng.spago.workflow.api.IWorkflowEngine**. Per creare gli oggetti di questo tipo viene utilizzata una factory che implementa l'interfaccia **it.eng.spago.workflow.api.IWorkflowEngineFactory**.

Senza addentrarci in dettagli implementativi il nome della factory fornita da una specifica implementazione per un particolare motore può facilmente essere messo in una file di configurazione ed essere istanziato via reflection.

Quando si utilizzano i componenti di workflow in una applicazione web per esempio, il nome della factory viene scritto nel file di configurazione **workflow.xml** che ha la struttura seguente:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<WORKFLOW>
  <ENGINE name="shark01" factory="it.eng.spago.workflow.shark.impl.SharkWorkflowEngineFactoryImpl">
    <LOAD-PACKAGE packageFile="SanitaBO.xpd1"/>
  </ENGINE>
</WORKFLOW>
```

Questo file di configurazione viene utilizzato dall'initializer di spago che istanzia la factory, istanzia il motore e lo mette a disposizione nell'application container. Per quanto riguarda il resto del paragrafo supponiamo di istanziare la factory attraverso la reflection.

Una volta ottenuta l'istanza dell'engine è necessario effettuare ottenere la connessione ed effettuare l'autenticazione.

Il codice che segue dimostra il caso d'uso appena descritto:

```
package it.eng.spago.workflow.usecase;

import it.eng.spago.workflow.api.IWorkflowConnection;
import it.eng.spago.workflow.api.IWorkflowEngine;
import it.eng.spago.workflow.api.IWorkflowEngineFactory;

public class InitializeEngineAndConnect extends AbstractUseCase {

    public static void main(String[] args) {
        try{
            //Obtain the factory
            IWorkflowEngineFactory wfEngineFactory = getWorkflowEngineFactory();

            // Create IWorkflowEngine instance
            IWorkflowEngine wfEngine = wfEngineFactory.getWorkflowEngine();

            // Obtain IWorkflow Connection
            IWorkflowConnection wfConnection = wfEngine.getWorkflowConnection();
            wfConnection.open("andrea", "andrea");

            // do some activities with connection

            // Close Connection
            wfConnection.close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

4.2 Amministrazione delle definizioni di processo

Prima di poter istanziare dei processi, è necessario caricare nel sistema di workflow, le definizioni dei processi. In Spago queste operazioni vengono svolte utilizzando la classe IWorkflow Engine come di seguito, le definizioni di processo sono raggruppate in unità logiche chiamate package. Un package può contenere una o più definizioni di processi.

```
package it.eng.spago.workflow.usecase;

import java.util.List;
import it.eng.spago.workflow.api.IWorkflowConnection;
import it.eng.spago.workflow.api.IWorkflowEngine;
import it.eng.spago.workflow.api.IWorkflowEngineFactory;
import it.eng.spago.workflow.api.IWorkflowPackage;

public class PackageManagement extends AbstractUseCase {
```



```
public static void main(String[] args) {
    try{
        //Obtain the factory
        IWorkflowEngineFactory wfEngineFactory = getWorkflowEngineFactory();

        // Create IWorkflowEngine instance
        IWorkflowEngine wfEngine = wfEngineFactory.getWorkflowEngine();

        // Obtain IWorkflow Connection
        IWorkflowConnection wfConnection = wfEngine.getWorkflowConnection();
        wfConnection.open("andrea", "andrea");

        //List all package definitions in process definition repository
        //Pay attention the process definition in the repository are NOT LOADED
        //To do this we must call the loadPackageDefinition api

        List avPkgList = wfEngine.getAvailablePackages();
        for(int i=0; i < avPkgList.size(); i++){
            System.out.println(
                "The package " +
                "["+(IWorkflowPackage)avPkgList.get(i)).getPackageID()+"] -- " +
                "is availbale on the process definition repository");
        }

        //List all package laoded in the engine
        List lPkgList = wfEngine.getLoadedPackges();
        for(int i=0; i < lPkgList.size(); i++){
            System.out.println(
                "The package " +
                "["+(IWorkflowPackage)lPkgList.get(i)).getPackageID()+"] -- " +
                "is loaded on the engine");
        }

        // Load a Package
        wfEngine.loadPackageDefinition("SanitaBO.xpdl");

        IWorkflowPackage wfPackage = wfEngine.getWorkflowPackage("SanitaBO");

        // Unload a package from the engine
        wfEngine.unloadPackageDefinition(wfPackage);

        // Close Connection
        wfConnection.close();
    }catch(Exception e){
        e.printStackTrace();
    }
}
```

4.3 Gestione dei processi

Una volta caricate le definizioni dei processi è possibile istanziare, far partire e terminare i processi, queste attività vengono effettuate utilizzando l'interfaccia ***it.eng.spago.workflow.IWorkflowProcess***. Il codice che segue dimostra come questo avviene utilizzando le api di spago. Il codice illustra le funzionalità fornite da spago per la gestione dei processi.

```
package it.eng.spago.workflow.usecase;
import it.eng.spago.workflow.api.IWorkflowConnection;
import it.eng.spago.workflow.api.IWorkflowEngine;
import it.eng.spago.workflow.api.IWorkflowEngineFactory;
import it.eng.spago.workflow.api.IWorkflowProcess;

import java.util.Map;

public class ProcessManagement extends AbstractUseCase {

    public static void main(String[] args) {

        try{
            //Obtain the factory
            IWorkflowEngineFactory wfEngineFactory = getWorkflowEngineFactory();

            // Create IWorkflowEngine instance
            IWorkflowEngine wfEngine = wfEngineFactory.getWorkflowEngine();

            // Obtain IWorkflow Connection
            IWorkflowConnection wfConnection = wfEngine.getWorkflowConnection();
            wfConnection.open("andrea", "andrea");

            // Create a Process Instance
            IWorkflowProcess aWfProcess =
wfConnection.createProcess("SanitaBO", "Simulazione");

            // Start the Process
            aWfProcess.start();

            //Print Process state
            System.out.println(" PROCESS STATE [" + aWfProcess.getState() + " ]");

            // Get process Context
            Map context = aWfProcess.getContext();

            //Abort process before execution ends
            aWfProcess.abort();

            // Close Connection
```

```
        wfConnection.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

4.4 Gestione delle attività

La funzionalità più importante per un'applicazione client di un sistema di workflow, è la gestione delle attività legate ai processi. E' necessario fornire delle api per:

- Determinare le attività che possono essere prese in carico dagli utenti (Assignment List, Worklist)
- Prendere in carico (o accettare) un'attività. Il fatto che un utente sia un possibile candidato all'esecuzione di un attività non significa che vi è l'accettazione automatica da parte dell'utente. In un sistema di workflow spesso gli attori identificano gruppi di persone attraverso il concetto di ruolo, quindi per un'attività non esiste un solo utente abilitato ma possono essercene diversi. Una volta che l'attività viene presa in carico da un utente, non è comunque più completabile dagli altri utenti a meno che l'utente che ce l'ha in carico non la renda di nuovo disponibile senza completarla.
- Rifiutare un'attività (o renderla di nuovo disponibile)
- Completare un'attività una volta che la si è presa in carico.
- Ricavare eventuali attributi legati all'attività. Questo è molto importante se si vuole associare all'attività del workflow, qualche operazione legata alla logica applicativa.

Per poter svolgere le attività sopra descritte si utilizza l'interfaccia ***it.eng.spago.workflow.IWorkflowAssignment***, come in precedenza illustriamo le funzionalità descritte attraverso un esempio:

```
package it.eng.spago.workflow.usecase;

import it.eng.spago.workflow.api.IWorkflowAssignment;
import it.eng.spago.workflow.api.IWorkflowConnection;
import it.eng.spago.workflow.api.IWorkflowEngine;
import it.eng.spago.workflow.api.IWorkflowEngineFactory;
import it.eng.spago.workflow.api.IWorkflowProcess;

import java.util.List;
import java.util.Map;

public class ActivityManagement extends AbstractUseCase {

    public static void main(String[] args) {

        try{

            //Obtain the factory
```

```
IWorkflowEngineFactory wfEngineFactory = getWorkflowEngineFactory();

// Create IWorkflowEngine instance
IWorkflowEngine wfEngine = wfEngineFactory.getWorkflowEngine();

// Obtain IWorkflow Connection
IWorkflowConnection wfConnection = wfEngine.getWorkflowConnection();
wfConnection.open("andrea", "andrea");

IWorkflowConnection wfConnectionAndrea = wfEngine.getWorkflowConnection();
wfConnectionAndrea.open("andrea", "andrea");

List l = wfConnectionAndrea.getAssignments();

System.out.print("ANDREA HAS ["+l.size()+"] ACTIVITIES'");
for (int i = 0; i < l.size(); i++){
    System.out.println(l.get(i));
}

// ACCEPT AN ACTIVITY
IWorkflowAssignment aAssignment = ((IWorkflowAssignment)l.get(0));

//PRINT SOME INFORMATION ABOUT THE ACTIVITY
System.out.println("ActivityKey [" + aAssignment.getActivityKey()+"]");
System.out.println("ActivityName [" + aAssignment.getActivityName()+"]");
System.out.println("ActivityDescription [" +
aAssignment.getActivityDescription()+"]");
System.out.println("ActivityState [" + aAssignment.getActivityState()+"]");

// VERIFY IF THE ACTIVITY IS ACCEPTED
System.out.println("Activity Accepted Status [" + aAssignment.isAccepted() + "]");

System.out.println(" Accepting the activity");
// ACCEPT THE ACTIVITY
aAssignment.accept();

System.out.println(" Cannot complete reject it");
// REJECTING
aAssignment.reject();

// REACCEPT
aAssignment.accept();

// VERIFY IF THE ACTIVITY HAS A MAPPED FORM
String mappedform = (String)aAssignment.getMappedForm();

if (mappedform != null){
```

```
        System.out.println(" This activity has a mapped form associated named  
["+mappedform+"]");  
    }  
  
    // COMPLETE ACTIVITY  
    aAssignment.complete();  
    wfConnection.close();  
  
    }catch(Exception e){  
        e.printStackTrace();  
    }  
}  
}
```

5 Componenti di WorkList e Generazione Automatica di Form

Nel precedente caso d'uso abbiamo visto in dettaglio la api che spago offre per l'interfacciamento verso i motori di workflow. In questo paragrafo invece andiamo ad analizzare in dettaglio i componenti di worklist di spago che mettono a disposizione una serie di facilities già pronte per l'integrazione di funzionalità di workflow all'interno di applicazioni web based.

In linea di massima i requisiti di un'applicazione di worklist sono i seguenti:

1. In seguito ad un'autenticazione all'utente deve venire presentata una lista di attività pendenti sulle istanze dei processi, per cui è un possibile candidato all'esecuzione.
2. Da questa lista l'utente deve poter accettare e completare le attività. Se il completamento dell'attività richiede dei passaggi successivi quali il completamento di un form, l'accettazione dell'attività deve automaticamente predisporre il form necessario al completamento dell'attività.
3. All'utente deve essere possibile in ogni caso controllare e modificare lo stato relativo alle attività del processo.

In dettaglio i componenti di Worklist di Spago sono composti da:

1. ***it.eng.spago.workflow.worklist.Worklist.initializer.WfInitializer***: è un'initializer di spago che legge il file di configurazione workflow.xml, istanzia il motore di workflow e lo mette a disposizione nell'application container con chiave "WfEngine".
2. ***it.eng.spago.workflow.worklist.action.WorklistAction***: è un'action di lista di spago che si occupa di preparare la lista degli assignment per la presentazione. Il risultato deve essere presentato su di una pagina attraverso il tag ***it.eng.spago.workflow.worklist.tags.ListTag***.
3. ***it.eng.spago.workflow.worklist.action.AcceptActivityAction***: L'action che viene invocata dalla pagina della worklist quando l'utente accetta un'attività. Questa action ha la logica per determinare se una determinata attività è

associata ad un particolare form. Se l'attività deve essere associata ad un form autogenerato o ad un publisher di spago l'attributo "TargetForm" viene messo nella service response. L'action deve essere associata al publisher:

4. **it.eng.spago.workflow.worklist.publisher.AcceptActivityPublisher** che in base alla presenza nella service response dell'attributo "TargetForm" determina se ripubblicare la worklist, oppure renderizzare o un form autogenerato o un publisher consistente in una pagina custom. Se l'attributo "TargetForm" è presente nella service response il valore viene utilizzato per cercare nel file **forms.xml** che cosa deve essere pubblicato.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<FORMS>
  <!-- CUSTOM PUBLISHER THIS WILL RESULT IN PUBLISH OF -->
  <FORM formIdentifier="JSPCustomPublishre"
        formType="CUSTOM" formCustomPublisher="JSPDetailPublisher"/>

  <!-- AUTO GENERATED FORM THIS WILL RESULT in publisher "AUTO_GENERATE_FORM_PUBLISHER"
  -->
  <FORM formIdentifier="FormDatiEnteFlusso" formType="AUTO">

    <FIELDSET legend="Dati Ente Inviante Flusso" >

      <FIELD name="ACTION_NAME"
            type="hidden" value="COMPLETE_OR_REJECT_ACTIVITY_ACTION"/>

      <FIELD name="ActivityKey" type="hidden"
            value="$_SERVICE_REQUEST.ActivityKey"/>

      <FIELD name="codiceEnteInvianteFlusso"
            type="select" label="Codice Ente" labelType="pre">
        <FIELD-OPTION optionLabel="Ospedale Pubblico"
                      optionValue="OSP_PUB"/>
        <FIELD-OPTION optionLabel="Struttura Privata"
                      optionValue="OSP_PRIV"/>
        <FIELD-OPTION optionLabel="Casa di Riposo"
                      optionValue="CASA_RIP"/>
        <FIELD-OPTION optionLabel="Azienda Sanitaria Locale"
                      optionValue="ASL"/>
        <FIELD-OPTION optionLabel="Day Hospital"
                      optionValue="DAY"/>
      </FIELD>

      <FIELD name="nomeEnteInvianteFlusso"
            type="text" label="Nome Ente" labelType="pre"/>

      <FIELD name="indirizzoEnteInvianteFlusso"
            type="text" label="Indirizzo Ente" labelType="pre"/>

    </FIELDSET>

    <ONSUBMIT name="CompleteActivity" value="Complete Activity"/>

    <ONSUBMIT name="CancelActivity" value="Cancel Activity"/>

  </FORM>
</FORMS>
```

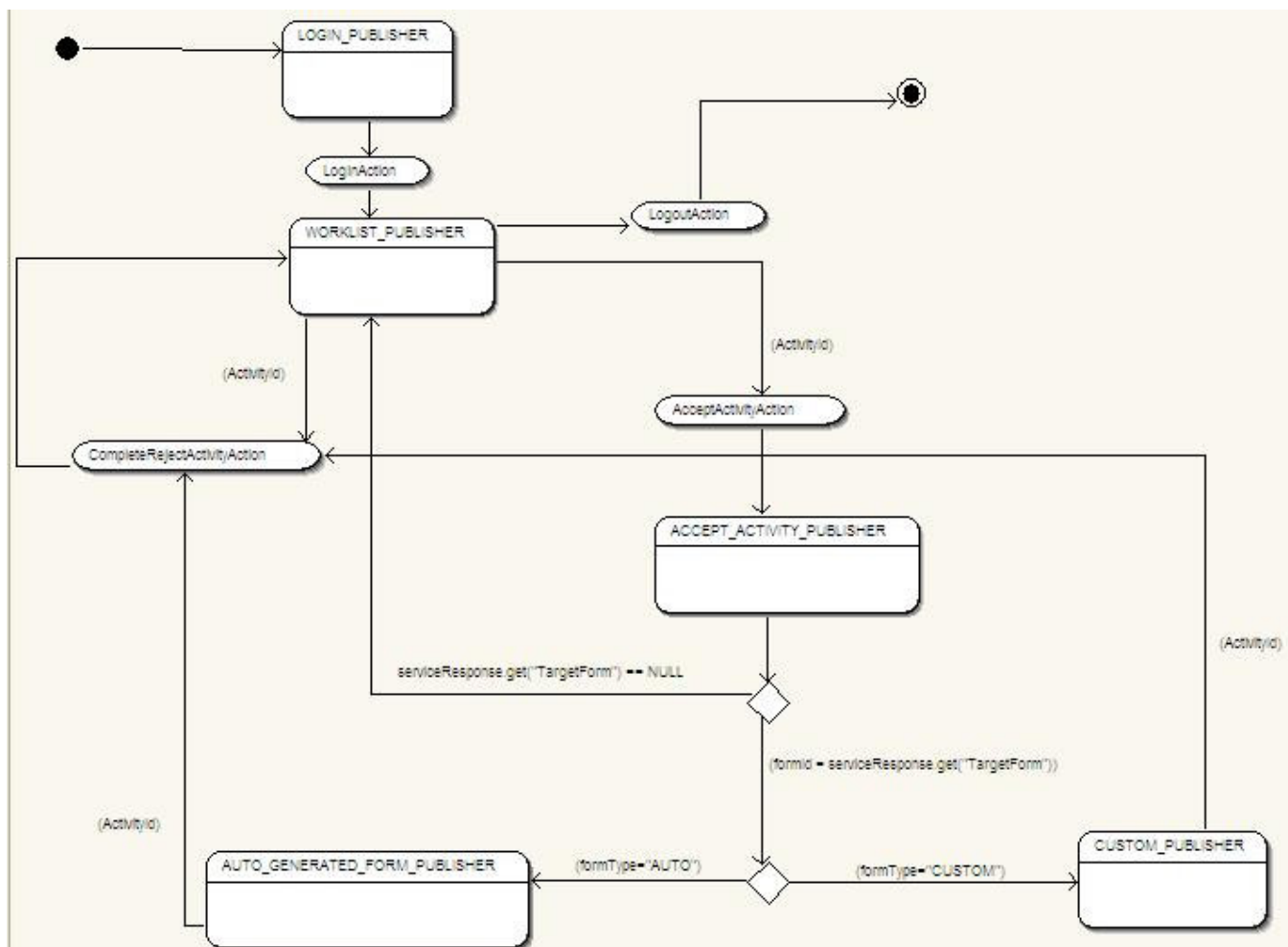
Se il valore dell'attributo "TargetForm" in sessione determina una busta **<FORM>** con l'attributo **formType** valorizzato con **"CUSTOM"** la pubblicazione viene semplicemente reindirizzata al publisher determinato dal valore dell'attributo **"formCustomPublisher"**.

Se invece il valore dell'attributo "TargetForm" in sessione determina una busta **<FORM>** con l'attributo **formType** valorizzato con **"CUSTOM"** la pubblicazione viene sempre reindirizzata sul publisher

"AUTO_GENERATE_FORM_PUBLISHER" che è un publisher che effettua il rendering sulla JSP autoForm.jsp che

attraverso il tag libraries ***it.eng.spago.workflow.worklist.tag.FormTag*** e le informazioni descritte nel file forms.xml genera automaticamente la pagina JSP.

5. ***it.eng.spago.workflow.worklist.action.CompleteRejectActivityAction***: Questa action viene invocata direttamente dalla worklist, o dalla pagina che deve essere completata per il completamento dell'attività.



6 Implementazione di SpagoWorkflowAPI su Shark workflow engine

Shark (<http://shark.objectweb.org>) è un prodotto open source presente da diverso tempo sul mercato in grado di gestire workflow descritti nel linguaggio standard XPDL che mette a disposizione i seguenti componenti:

1. Un motore di workflow, utilizzabile sia come motore embedded nelle applicazioni, sia come workflow server a cui è possibile accedere attraverso il protocollo CORBA.

2. Una utility di amministrazione (Swing) attraverso il quale è possibile:
 - a. Gestire le definizioni dei processi (package)
 - b. Istanziare nuovi processi
 - c. Monitorare l'esecuzione dei processi ed eventualmente intervenire manualmente
 - d. Gestire le utenze e la mappatura verso gli attori definiti nel workflow
3. L'integrazione con un prodotto di disegno visuale di workflow denominato JaWE (<http://jawe.objectweb.org>) attraverso la quale è possibile disegnare workflow aderenti allo standard XPDL.

Per il momento l'implementazione fornita con spago permette di utilizzare Shark solo come motore di workflow Embedded nell'applicazione.

Per utilizzare shark all'interno di spago sono quindi necessari i seguenti passi:

1. Creare il database di Shark attraverso gli script presenti nel prodotto (Vedere la documentazione di Shark)
2. Disegnare i workflow con JaWE e metterli a disposizione nel repository di processi di Shark
SHARK_HOME\repository\external\.
3. Utilizzare la console di amministrazione di shark per caricare le definizioni dei processi, nel motore. Quando un package (definizione di uno o più processi) viene caricato nel motore gli attori definiti nel workflow sono visibili nell'engine ed è quindi possibile rimappare gli utenti del motore sui rispettivi attori.
4. Riappare gli utenti del motore sugli attori attraverso la console di amministrazione di Shark
5. Copiare tutte le librerie di shark e le dipendenze nel classpath dell'applicazione (Vedere la documentazione di Shark)
6. Copiare le librerie **spago-workflow-api.jar**, **spago-worklist** e **spago-shark.jar** nel classpath dell'applicazione
7. Predisporre il file **shark.conf** nel classpath dell'applicazione (Vedere la documentazione di Shark)
8. Utilizzare la factory denominata **it.eng.spago.workflow.shark.impl.SharkWorkflowEngineFactoryImpl**.

