Programmazione Distribuita II

A.A. 2011/12

Assignment n. 6 – Parte 2

Tutto il materiale necessario per questa esercitazione si trova nel file .zip dal quale questo documento è stato estratto. Per lo svolgimento dell'esercitazione si consiglia di estrarre tutto l'archivio .zip in una cartella vuota, all'interno della quale si lavorerà.

Questa seconda parte dell'assignment consiste nei seguenti 3 punti:

A. Implementare il servizio progettato nella prima parte di questo assignment e scrivere una classe dotata di `main` che serva per pubblicare il servizio tramite la classe `Endpoint` di JAX-WS. Il servizio deve essere pubblicato all'URL `http://localhost:8181/WorkflowService` lanciando il main e deve fornire il WSDL progettato nella parte 1 dell'assignment. Più precisamente, all'URL specificato deve essere disponibile il servizio che permette di creare i processi. Se sono necessari altri URL, questi possono essere scelti liberamente ma usando solo le porte immediatamente successive (8182, 8183, ...).

Tutte le classi sviluppate devono essere poste nel package `it.polito.pd2.WF.sol6.service` e memorizzate nella cartella *`[root]/src/it/polito/pd2/WF/sol6/service/`* dove, come al solito, `[root]` è la radice dove l'archivio è stato estratto. Oltre alle classi Java, è necessario scrivere uno script di ant e salvarlo nel file `[root]/sol_build.xml`. Questo script deve avere un target `run.server` lanciando il quale vengono svolte automaticamente tutte le operazioni di compilazione (compresa la generazione automatica di codice) e viene pubblicato il servizio. Importante: lo script deve definire *`basedir="."`* e tutti i path a cui fa riferimento devono stare sotto *`${basedir}`* ed essere definiti in modo relativo rispetto a *`${basedir}`*. Il file WSDL del servizio deve essere posto sotto *`${basedir}/wsdl`*.

Il servizio deve poter essere utilizzato da più client contemporaneamente in modo affidabile. La gestione della persistenza invece non è richiesta.

Il servizio deve ottenere le informazioni sui workflow istanziando il generatore pseudocasuale come nell'assignment 2. A questo scopo, lo script `sol_build.xml` deve fare in modo che vengano impostate le proprietà necessarie per l'avvio del generatore di dati, come nell'assignment 2 e 4. All'avvio, il servizio deve partire da uno stato nel quale non ci sono processi ma solo workflow.

B. Implementare un client, analogo a quello dell'assignment 5, che acceda in lettura a tutte le informazioni su workflow e processi e le renda disponibili tramite le interfacce e le classi astratte definite nel package *`it.polito.pd2.WF`*, restituendo i dati ottenuti dal servizio. Come per l'assignment 5, il client deve assumere la forma di una libreria, scritta interamente nel package *`it.polito.pd2.WF.sol6.client1`* e i suoi sorgenti devono essere memorizzati in *`[root]/src/it/polito/pd2/WF/sol6/client1`*. Come per l'assignment 5, la libreria include una classe factory *`WorkflowMonitorFactoryImpl`*, che estende la factory astratta *`it.polito.pd2.WF.WorkflowMonitorFactory`* e, tramite il metodo *`newWorkflowMonitor()`*, crea un'istanza della classe della soluzione che implementa l'interfaccia *`it.polito.pd2.WF.WorkflowMonitor`*. Analogamente, per questo client valgono le specifiche date per l'assignment 5 (tranne per il fatto che deve restituire anche le informazioni sui processi) e l'URL effettivo da usare sarà memorizzato nella proprietà di sistema *`it.polito.pd2.WF.sol6.URL`*.

Lo script `sol_build.xml` deve includere anche un target con nome *`compile-client-1`* per la compilazione del client, compresa la generazione degli artifacts necessari. Tutti i file `.class` devono essere salvati nella cartella `${basedir}/build` che viene inclusa nel classpath per l'esecuzione dei test.

C. Implementare un altro client per il servizio, che permetta all'utente di eseguire le operazioni di cui ai punti 2, 3 e 4 della prima parte dell'assignment 6 (creazione processi, presa in carico e terminazione di azioni) tramite un semplice sistema di menu realizzati tramite la console. All'avvio, il client legge come parametro sulla linea di comando due stringhe che rappresentano il nome e il ruolo iniziale dell'attore per conto del quale saranno eseguiti i comandi. In tutti i menu, la selezione di una voce avviene tramite

l'introduzione di un numero intero compreso tra 0 e n-1 dove n è il numero delle voci del menu. Per facilitare la conformità alle specifiche si consiglia di usare per l'input dei comandi da tastiera la classe `it.polito.pd2.WF.lab6.InlutConsole` fornita come sorgente.

Il menu principale, da visualizzare all'inizio, deve avere le seguenti voci:

0 termina
1 crea processo
2 prendi azione in carico
3 termina azione
4 assumi ruolo

Ognuna delle voci 1-4, quando selezionata, porta ad un altro menu corrispondente. Per esempio, se viene selezionata la voce 1 verrà presentato un menu che elenca i nomi dei workflow noti, in ordine alfabetico crescente. Selezionato il workflow, viene creato un processo di quel workflow e poi viene nuovamente presentato il menu iniziale.
Se invece viene selezionata la voce 4, viene presentato un elenco di tutti i ruoli che intervengono nei workflow noti, in ordine alfabetico crescente. Selezionato il ruolo, questo viene assunto da quel momento in avanti per le prossime operazioni eseguite (il nome dell'attore invece rimane sempre lo stesso).
Se infine vengono selezionate le voci 2 o 3, viene presentato prima un menu che elenca tutti i processi e poi uno che elenca le azioni che in quel momento possono essere prese in carico o terminate in base al ruolo attuale nel processo selezionato. Per la presa in carico, una volta selezionata l'azione si ritorna al menu iniziale, mentre per la terminazione, nel caso in cui l'azione selezionata sia una Simple Action vi è un ultimo menu che permette di selezionare le azioni future da abilitare prima di tornare al menu principale. In questo ultimo, la voce 0 significa che la selezione è terminata mentre le voci successive sono le possibili azioni future selezionabili.

Ai fini dell'esame nel primo appello d'esame è richiesto di implementare solo le voci 0 e 1 del menu principale e il menu di selezione del workflow per la creazione del processo.

Il client deve essere scritto interamente nel package *`it.polito.pd2.WF.sol6.client2`* e tutti i suoi sorgenti devono essere memorizzati in *`[root]/src/it/polito/pd2/WF/sol6/client2`*. La classe contenente il main deve chiamarsi `Client2`.

Anche per questo client lo script `sol_build.xml` deve includere un target per la compilazione con nome *`compile-client-2`* che generi anche degli artifacts necessari, e tutti i file `.class` devono essere salvati nella cartella `${basedir}/build` che viene inclusa nel classpath per l'esecuzione dei test.

Verifica della correttezza

Prima di sottomettere la soluzione dell'esercizio siete tenuti a verificarne il corretto funzionamento e l'aderenza alle specifiche date. Perché sia accettabile ai fini dell'esame, la soluzione deve superare almeno i test automatici obbligatori. Si noti che i test automatici controllano solo alcuni dei requisiti funzionali! In particolare essi controllano che

1. il file sol_build.xml sia corretto sintatticamente e il server venga pubblicato correttamente
2. i dati estratti dalla libreria di cui al punto B. siano congruenti con i comandi eseguiti tramite il client di cui al punto C. Questo controllo viene eseguito inviando una serie di comandi di test.

Altri controlli e valutazioni sul codice verranno fatti in sede d'esame (quindi superare tutti i test non garantisce necessariamente che la valutazione sarà ottima).
Il file *.zip* fornito per questo assignment include un insieme di test come quelli che verranno fati girare sul sistema di sottomissione. Per lanciarli si può usare il comando

```
ant run-tests -Dseed=XXX -Dtestcase=Y -Dtestfile=ZZZ
```

che provvede anche a lanciare i target di compilazione dei client in `sol_build.xml`. Oltre a seed e testcase, che hanno lo stesso significato come nei precedenti assignments, si può specificare un file di input che fornisce una sequenza di comandi per il client 2. Un esempio di tale file (usato nel caso non si specifichi questa opzione) è contenuto nella cartella `testcases`. Si noti che subito prima di lanciare questo test va

lanciato il server, che deve trovarsi nello stato iniziale.

Formato per la sottomissione della soluzione

Occorre sottomettere un unico file *.zip*, contenente tutti i file prodotti. Il file *.zip* può essere prodotto con il seguente comando (dalla cartella *[root]*):

```
$ jar -cf lab6.zip sol_build.xml src/it/polito/pd2/WF/sol6/service/*.java
src/it/polito/pd2/WF/sol6/client1/*.java
src/it/polito/pd2/WF/sol6/client2/*.java
wsdl/*
```

Si noti che il file .zip non deve includere i file generati automaticamente.