Anno accademico 2006/2007

Università di Roma "La Sapienza"
Facoltà di Ingegneria
Corso di Laurea Specialistica in Ingegneria Informatica

Tesina di Seminari di Ingegneria del software

Composizione automatica di servizi: l'approccio ASTRO

Autore:
Vanda Piacentini

Indice

Introduzione

Questa tesina si colloca nell'ampio mondo dei *Web Services*. Secondo la definizione data dal World Wide Web Consortium (W3C) un **Web Service** (**servizio web**) è un sistema software progettato per supportare l'interoperabilità tra diversi elaboratori su di una medesima rete. Caratteristica fondamentale di un Web Service è quella di offrire un'interfaccia software (descritta in un formato automaticamente elaborabile quale, ad esempio, il Web Services Description Language) utilizzando la quale altri sistemi possono interagire con il Web Service stesso attivando le operazioni descritte nell'interfaccia tramite appositi "messaggi" inclusi in una "busta" SOAP. Tali messaggi sono, solitamente, trasportati tramite il protocollo HTTP e formattati secondo lo standard XML. Proprio grazie all'utilizzo di standard basati su XML, tramite un'architettura basata sui Web Service (chiamata, con terminologia inglese, *Service oriented Architecture - SOA*), applicazioni software scritte in diversi linguaggi di programmazione e implementate su diverse piattaforme hardware possono quindi essere utilizzate, tramite le interfacce che queste "espongono" pubblicamente e mediante l'utilizzo delle funzioni che sono in grado di effettuare (i "servizi" che mettono a disposizione) per lo scambio di informazioni e l'effettuazione di operazioni complesse (quali, ad esempio, la realizzazione di *processi di business* che coinvolgono più aree di una medesima azienda) sia su reti aziendali come anche su Internet. La ragione principale per la creazione e l'utilizzo di Web Service è il "disaccoppiamento" che l'interfaccia standard esposta dal Web Service rende possibile fra il sistema utente ed il Web Service stesso: modifiche ad una o all'altra delle applicazioni possono essere attuate in maniera "trasparente" all'interfaccia tra i due sistemi; tale flessibilità consente la creazione di sistemi software complessi costituiti da componenti svincolati l'uno dall'altro e consente una forte riusabilità di codice ed applicazioni già sviluppate.

Lo scopo della tesina, svolta per il corso di Seminari di Ingegneria del software, è lo studio del tool realizzato dal Progetto ASTRO per la composizione automatica di servizi e lo studio di una demo esistente basata su un semplice esempio di Web Services: VTA, che possiamo definire un'agenzia di viaggi virtuale. In realtà si è provato anche a realizzare un esempio di demo sulla base di quella già esistente, ma a causa della mancanza di documentazione e tutorial e soprattutto delle numerosi limitazioni del tool, non si è riusciti a giungere ad una demo funzionante.
Nel primo capitolo si parlerà in linea generale del problema della Web service composition. Nel secondo capitolo si inizierà a parlare del Progetto ASTRO e di come viene affrontata la composizione automatica di servizi e quindi del tool sviluppato. Nel terzo capitolo si spiegherà l'installazione del tool, chiamato Astro suite o Astro toolset. Nel quarto capitolo si analizza la demo

VTA nell'intero processo di composizione. Nel quinto capitolo si accennerà all'esempio di demo che si è provato a realizzare.

Capitolo 1 - Il problema della Web service composition

1.1 WS Composition

L'ampia diffusione di applicazioni service - oriented e in particolare dei web services, all'interno di numerose organizzazioni, ha introdotto nel campo della ricerca tematiche interessanti su come poter sfruttare i componenti di business già esistenti, in modo da poter costruire a partire da questi e in modo automatico, nuovi servizi composti di valore aggiunto per il cliente. La composizione di servizi viene attualmente affrontata in modo manuale: il cliente specifica i suoi requisiti e il progettista si occupa di concepire un nuovo processo di business che invochi adeguatamente le componenti applicative esistenti. Come si può ben intuire, tale compito si presenta abbastanza laborioso e non privo di difficoltà. Ciò a cui si vuole arrivare al contrario, è un procedimento efficiente, affidabile e di facile uso che permetta, a partire da specifici requisiti, di comporre in modo automatico web service e in generale frammenti di applicazioni.

La Service Composition è una metodologia che ha per fine l'implementazione di Composite Service, un Web Service che offre servizi da una sua interfaccia come qualsiasi altro Web Service, sebbene dal punto di vista implementativo i servizi offerti siano il risultato di un'opportuna interazione con altri Web Services, indipendenti tra loro e non pensati a priori per cooperare in un Web Service comune.

Nella Web Service Composition si vuole trovare, dati dei requisiti opportunamente espressi, un piano d'esecuzione in cui siano indicati quali Web Services invocare, in che ordine farlo e come gestire condizioni di errore e imprevisti.

In generale gli strumenti necessari per risolvere il problema sono:

- un linguaggio per la *rappresentazione comportamentale* di Web Services, in modo da modellare efficacemente il loro flow d'esecuzione e le funzionalità che offrono;
- una *logica di composizione*, ovvero un procedimento generale che partendo dai requisiti (Business Requirements) e dai Web Services di partenza (Component Services) realizzi il Composite Service finale in una qualche forma eseguibile; questo punto rappresenta il cuore di un approccio per la WS Composition, una sorta di algoritmo di base;
- un *ambiente di sviluppo*, possibilmente ricco di componenti GUI, che aiuti il progettista a creare il servizio ad alto livello, automatizzando la metodologia definita dalla logica di composizione;
- un *composition engine* per eseguire e monitorare le istanze della composizione trovate.

Gli approcci che cercano di affrontare il problema della Web service composition, mirano alla realizzazione pratica del workflow descritto dalla seguente figura:

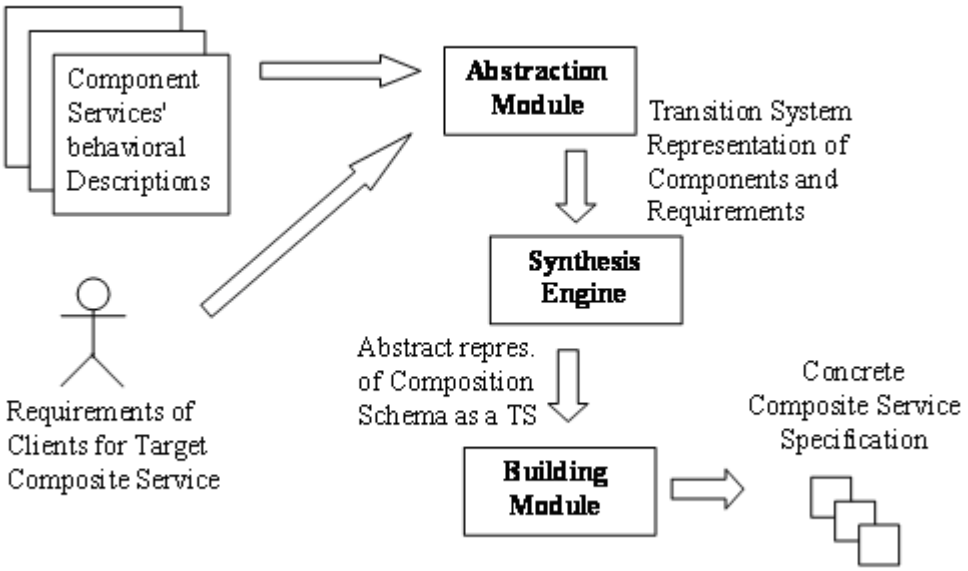


Figura 1: *The general WS composition workflow*

1.2 Terminologia

Il termine *conversazione* rappresenta un'interazione con un Web Service consistente nell'esecuzione sequenziale di più operazioni, in un particolare ordine.

Il termine *coreografia* indica un piano per la coordinazione di più conversazioni, volta ad un preciso scopo d'insieme.

Con il termine *sintesi* di un Composite Service si intende la costruzione delle specifiche necessarie all'esecuzione del servizio a partire da requirements ben definiti; tali specifiche sono conosciute come *Composition Schema*.

Il termine *orchestrazione* indica la gestione runtime dell'esecuzione del Composite Service.

Capitolo 2 - Il progetto Astro

2.1 Panoramica del progetto

Il Progetto Astro è un'iniziativa di ricerca congiunta riguardo l'integrazione di Web Services, sia intra- che inter-organizzazione, promossa dall'Università di Trento e l'ITC-IRST, il Centro di Ricerca Scientifica e Tecnologica della Fondazione Bruno Kessler.

Il suo scopo principale è favorire l'adozione worldwide di Web Services compositi prestando attenzione a metriche fondamentali quali efficacia, flessibilità, facilità d'uso, basso costo ed efficienza temporale.

Astro vuole fornire:

- un framework generale per la composizione automatica di servizi;
- dei tools concreti per la realizzazione del framework, utilizzanti una larga serie di tecnologie affermate;
- supporto software per l'intero ciclo di vita delle applicazioni, dalle prime fasi di design fino al monitoraggio e verifica a runtime;
- evitare di delegare allo sviluppatore dei compiti noiosi, complessi ed error-prone, in modo da permettergli di concentrarsi in modo trasparente e user-centered sulla logica dell'applicazione ad un alto livello di astrazione.

I tools dovrebbero essere capaci di analizzare i processi in dettaglio e di scoprire i problemi sia a livello di design sia a livello di run-time, e fornire soluzioni alternative. Le attività di ricerca sono strutturate secondo i seguenti settori:

- *Business Requirements*: questo settore mira allo sviluppo di un framework per rappresentare efficacemente la definizione di strategie, obiettivi e business requirements aziendali, con particolare riguardo anche alle interazioni tra differenti business processes;
- *Service Synthesis*: il settore della sintesi offre un modello per ottenere dei servizi compositi in maniera generale ed efficiente, nonché supportata da una teoria di fondo che garantisce la correttezza e l'affidabilità dei risultati; questo ramo ha una controparte pratica incarnata dai tools eseguibili per la composizione di Web services;
- *Service Verification*: il tool offre anche strumenti di supporto per controllare se i requirements definiti sono violati dal servizio risultante ottenuto;
- *Service Monitoring*: il corrispettivo a runtime della service verification;

- *Semantics*: parte degli sforzi di ricerca sono volti all'adozione di supporto per integrare semantic web services, rendendo il tools interoperabile con OWL-S e WSMO.

2.2 WS Composition in ASTRO

Nell'approccio ASTRO gli input al problema sono costituiti da un set di Component Services, espressi come Abstract BPEL Processes, e da una specifica di Composition Requirements come EAGLE formula, e si vuole generare automaticamente un nuovo servizio W, il target composite service, che utilizza i component services esistenti e soddisfa i nostri composition requirements.
Inoltre si hanno le seguenti assunzioni:

- essere in un dominio *asincrono*: ogni web service evolve indipendentemente e con velocità imprevedibile, sincronizzandosi con gli altri tramite scambio di messaggi ed evidentemente, in implementazioni reali, vengono impiegati dei buffer che consentono di non perdere i messaggi che non possono essere immediatamente processati;
- i component services offrono *osservabilità parziale*, cioè non espongono le loro operazioni interne ma solamente le interazioni con l'esterno;
- i composition requirements devono essere espressi come *extended goals*, in grado di catturare condizioni esistenti sui percorsi dell'intero piano.

La composizione viene modellata come un problema di pianificazione, basata sull'approccio "Planning as Model Checking", concepite per poter lavorare anche in domini non-deterministici, con condizioni di parziale osservabilità ed "extended goals". Il risultato finale è un piano di esecuzione nel quale, a partire da alcune condizioni iniziali, esso specifica l'insieme di azioni da eseguire per raggiungere il goal.

La rappresentazione comportamentale dei servizi è basata su STSs che distinguono *azioni di input, di output ed interne(*t-transitions).

L'interfaccia pubblica di invocazione del servizio viene specificata con WSDL, mentre per codificare una descrizione "comportamentale" viene utilizzato il linguaggio BPEL4WS che permette di modellare ad alto livello le interazioni che si verificano in un singolo web service (per esempio invio e ricezione di messaggi).

La figura seguente mostra come sia possibile giungere dagli input definiti sopra, al Concrete BPEL Process eseguibile che implementa il Composite Service desiderato.

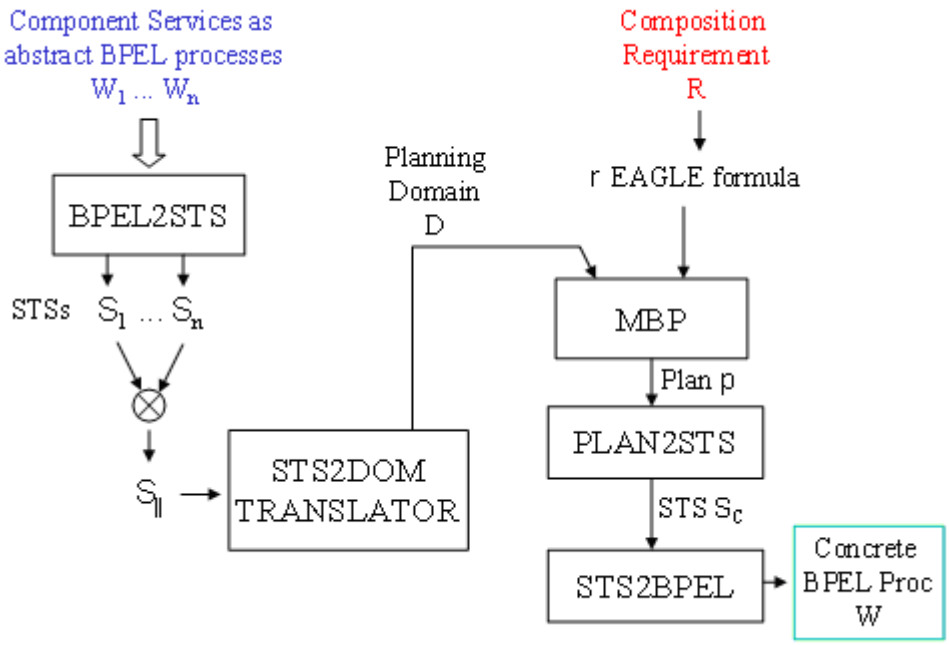


Figura 2: *The Astro composition workflow*

Da un'analisi generale della figura possiamo vedere che si parte da due input: un set di component services espressi come abstract BPEL processes $W_1 \ldots W_n$, che descrivono il comportamento "visibile dall'esterno" dei component services; e i composition requirements espressi tramite una formula EAGLE r. Tramite un modulo software BPEL2STS, i processi BPEL sono trasformati in rappresentazioni STS e si ottengono così gli STS $S_1 \ldots S_n$. Questi vengono manipolati per creare un nuovo STS $S_{||}$, definito come prodotto parallelo dei $S_1 \ldots S_n$. Intuitivamente possiamo pensare al Prodotto Parallelo come ad un STS che combina tutte le possibili evoluzioni dei Web Services componenti. A questo punto viene creato un dominio D che, insieme alla formula r, viene utilizzato per individuare un piano p, da cui in seguito si ricava l'STS del mediatore (una forma di orchestratore), che si occuperà di inviare/ricevere le invocazioni riguardanti i servizi componenti. L'ultima fase prevede la traduzione dell'orchestratore in un processo BPEL concreto che può essere eseguito su un ambiente runtime, come ad esempio Active BPEL.

2.2.1 Basi teoriche della composizione in Astro

Ora presentiamo alcuni concetti teorici che sono dietro questo processo di composizione.
Lo scopo finale è la realizzazione del Composite Service a partire dai nostri requirements, che si concretizza nel trovare un STS S_C, che soddisfa particolari proprietà: esso deve "muoversi" all'interno di $S_{||}$ per "controllare" i component services, allo stesso tempo rispettando il goal r ed evitando di porsi in stati pericolosi, come ad esempio un deadlock.

Si definiscono quindi due attori, entrambi STS: S_C e S, facendo in modo che il primo controlli il secondo, cioè S_C è il *Controller* e S il *Sistema Controllato*. La nozione di controllo deriva principalmente dalla corrispondenza input-output delle azioni: l'input del Controller è l'output dell'STS controllato, ovvero il controllato fornisce all'output le informazioni generate, mentre l'output del Controller è l'input del controllato, ovvero il Controller "istruisce" l'STS controllato sulle prossime azioni.

Le assunzioni di sistema asincrono in cui ci poniamo pongono di fronte a noi una difficoltà: non tutti gli STS controllers per un dato STS sono adatti ai nostri scopi: vorremmo evitare deadlocks, più precisamente vorremmo che ogniqualvolta il Controller "invii" un messaggio in output all'STS controllato, questo sia pronto a ricevere tale messaggio. Dobbiamo pertanto definire un sottoinsieme dei Controllers possibili, tale che l'STS controllato possa ricevere gli input forniti, eventualmente dopo una catena, arbitrariamente lunga ma finita, di t-transitions. Si introduce così il deadlock-free controller, cioè esiste una input transition raggiungibile dallo stato presente per accettare l'input lanciato dal sistema controllato.

Per soddisfare il composition goal r, abbiamo bisogno di esplorare tutte le possibili esecuzioni del Sistema Controllato e le proprietà soddisfatte in tali esecuzioni. Non possiamo fare ciò sotto ipotesi di osservabilità parziale (il Controller non ha piena osservabilità sul Prodotto Parallelo Controllato). Ci portiamo quindi al *Belief-Level*, ovvero consideriamo set di stati ugualmente plausibili date le nostre conoscenze, che evolvono tramite *external transitions* includendo nel nuovo Belief State stati raggiungibili tramite *t-closure* (set di stati raggiungibili da transizioni interne).

Quindi possiamo definire formalmente il problema della composizione in Astro con la seguente definizione.

Definizione: Astro Composition Problem

Siano S_1, ..., S_n un insieme di STSs, e r un composition requirement.

Il problema di composizione per S_1, ..., S_n e r è il problema di trovare un Controller S_C che è deadlock-free e tale che $S_B \models r$, dove S_B è il Belief-Level System dell'STS $S_C \triangleright (S_1 \| ... \| S_n)$.

2.2.2 Rappresentazione dei Component Services come STS

Un metodo efficace per rappresentare Web services, consiste nell'utilizzare State Transistion System (STS), un tipo di macchine a stati finiti. In generale infatti un web service può essere caratterizzato dalle operazioni (atomiche) che esso espone all'esterno, inserite opportunamente in particolari sequenze di esecuzione (conversazioni); negli STS le operazioni sono rappresentate dalle transizioni, mentre gli stati codificano le condizioni in cui i web services si trovano.

Gli STS definiti in Astro distinguono possibili stati, e i cambiamenti tra stati avvengono attraverso azioni, le quali possono essere classificate in azioni di input (ricezione di messaggi), azioni di output (invio di messaggi) e t-transitions, ovvero azioni di evoluzione interna e non visibile alle entità esterne.

Di seguito viene riportata una ridefinizione di STS secondo il progetto Astro:

Definizione: Astro State-Transition Sytem (STS)

Un Transition System S è una tupla $< S, S^0, I, O, R, L >$, dove:

- S è l'insieme finito degli stati;
- S^0, sottoinsieme di S, è l'insieme di stati iniziali;
- I è l'insieme finito di *input actions* (cioè ricezione di messaggi);
- O è l'insieme finito di *output actions* (cioè invio di messaggi);
- R è la relazione di transizione da $S \times (I \cup O \cup \{t\}) \rightarrow S$;
- $L: S \rightarrow 2^{Prop}$ è una funzione di etichettatura.

Sostanzialmente quindi, uno STS rappresenta il servizio come un sistema che può trovarsi in uno di diversi stati possibili (alcuni marcati come iniziali altri come finali in cui il servizio può terminare) e che può transitare in altri stati per mezzo di azioni. Tali azioni possono essere di input, output (invio e ricezione di messaggi a/da altri web services) o interne, ovvero il sistema evolve senza produrre output e indipendentemente dalla ricezione di input (t-transitions). La relazione di transizione spiega invece come passare da uno stato all'altro, al verificarsi delle azioni appena descritte. Infine, la funzione di etichettatura associa ad ogni stato l'insieme delle proprietà valide in quel determinato stato.

Vengono effettuate alcune assunzioni sulla modellazione in STS di Component Services: l'assenza di loops infiniti su t-actions e l'impossibilità che uno stato abbia origine sia da input che da output transitions.

Inoltre il modulo di traduzione, BPEL2STS non supporta tutti i costrutti BPEL, ad esempio nell'ultima versione 3.4 dell'Astro suite i costrutti "Scope" e "Fault" non sono supportati; tuttavia il range di operatori attualmente disponibili permette un certo livello di complessità.

Ricordiamo infine come lo stato di un STS dipenda dalle sue variabili interne, così come le transizioni definite da R dipendono da queste stesse variabili; perché il file .smv che incarna l'STS dei Component Services sia trattabile, vengono definiti ranges finiti per le variabili in gioco.

2.2.3 Nozione di BPEL

L'acronimo BPEL sta per Business Process Execution Language, ed è un linguaggio appositamente creato per la definizione ed esecuzione di processi i cui passi di esecuzione possono rappresentare invocazioni a Web Services. Il linguaggio è basato su XML (ovvero un file BPEL è a tutti gli effetti un file xml con dei costrutti particolari e processabile da tools appositi) ed è il fulcro centrale su cui orbita la realizzazione dell'intero Astro toolset: sia parte dei composition requirements, sia l'output eseguibile finale sono file BPEL.

Un'importante distinzione tra i files BPEL è quella tra *Abstract* e *Concrete* files. Entrambi descrivono un (composite) Web service attraverso xml, ma mentre un Abstract process definisce solo il comportamento visibile "dall'esterno" dello scambio di messaggi tra web services, un Concrete process è un file a tutti gli effetti eseguibile, dettaglia anche le evoluzioni interne dei servizi, cioè le t-transitions, e può concretizzarsi in un processo deployable su un BPEL engine e monitorabile.

I costrutti BPEL più importanti ed usati dal tool Astro sono:

- un processo abstract che definisce un set di messaggi scambiati tra web services, senza però definire la internal business logic;
- un processo concrete che definisce la business logic di un servizio servendosi di *attività* constituenti, *partners* coinvolti nel servizio, *message exchange* necessario e procedure di *exception handling*.

Per quanto riguarda le attività BPEL, esse possono essere primitive o strutturate.

Le attività primitive principali includono:

- *invoke*: per invocare un Web Service;
- *receive* e *reply*: per ricevere messaggi da una sorgente esterna o inviarli verso l'esterno;
- *wait* ed *empty*: per rimanere inattivi, rispettivamente per un certo periodo di tempo e indefinitamente;
- *assign*: per assegnare valori alle variabili interne che costituiscono lo stato del Web service;
- *throw*: per lanciare eccezioni.

Le attività strutturate includono:

- *sequence*: per eseguire una catena di azioni sequenziali;
- *switch*: simile al noto costrutto informatico, effettua una singola decisione basandosi su una variabile;
- *pick*: per "ascoltare" i cambiamenti su un dato set di eventi; non appena accade un certo evento, viene scelta ed eseguita una certa azione (legato al non-determinismo);

- *while*: per il tradizionale ciclo di iterazioni;
- *flow*: per gestire esecuzioni parallele (Astro non lo usa nella versione corrente 3.4).

2.2.4 Nozione di EAGLE

I requisiti in ASTRO vengono descritti nel linguaggio EAGLE che rispetto a logiche temporali come CTL e LTL permette di specificare un tipo di soddisfacibilità "best-effort"; è possibile indicare un obiettivo principale (main goal) e alcune condizioni che devono invece verificarsi in caso di fallimento (exception handling). Il linguaggio fornisce costrutti per esprimere condizioni che il sistema deve garantire di raggiungere o mantenere, o in alternativa effettuare solo un tentativo, prevedendo delle proprietà da soddisfare in caso di fallimento.
Le formule EAGLE sono utilizzate per esprimere business goals del target composite service; esse consistono in blocchi contenenti formule proposizionali che intuitivamente definiscono dei particolari stati che il sistema può raggiungere; ogni blocco è associato ad un particolare operatore che definisce la funzione stessa di quel blocco all'interno del sistema.

Capitolo 3 - L'installazione del Toolset

Per questa tesina è stata presa in considerazione la versione 3.4 dell'Astro toolset. Esso è formato da numerosi componenti software, alcuni sviluppati interamente dal team Astro, altri sono programmi di terze parti con le quali il toolset interagisce. La documentazione che accompagna il toolset e gli esempi di demo, è praticamente *inesistente*.

Di seguito viene riportato un elenco dei vari componenti, accompagnato da brevi spiegazioni sul loro ruolo e dalle rispettive modalità di installazione.

3.1 Java 1.5.x

La JVM è un componente essenziale del toolset, poiché molte parti di esso sono scritte in Java e producono/usano files Java.

Quindi è necessario verificare se la Java Runtime Enviroment versione 1.5 è già installata sul proprio sistema attraverso l'esecuzione della seguente linea di comando:

```
java –version
```

Se la JVM non è installata allora può essere scaricata dal sito della Sun, installata e settate le seguenti variabili d'ambiente:

```
JAVA_HOME = <J2SE_INSTALL_DIR>
PATH = <J2SE_INSTALL_DIR>\bin
```

dove `<J2SE_INSTALL_DIR>` è la directory in cui è stata installata la JVM.

3.2 Eclipse IDE 3.2.2

L'ambiente di sviluppo Eclipse è stato scelto dal team Astro come la colonna portante di tutta la sezione grafica del toolset; molte componenti della Suite sono state sviluppate come Eclipse plugins. Se non si ha già installato Eclipse sul proprio sistema, bisogna provvedere a scaricarlo. L'installazione consiste semplicemente nello "scompattare" il file in una folder a piacere. Si consiglia di creare una folder in C:\ o al massimo in C:\Programmi, perché percorsi troppo lunghi potrebbero "dar fastidio" all'esecuzione del toolset.

3.3 Tomcat server 5.5.x

Il server Tomcat è utilizzato per il deployment e running dei processi BPEL che incarnano i Web Services offerti. Consiglio di installare la versione 5.5.20, perché è quella che ho utilizzato per analizzare la demo.

Anche l'installazione del Tomcat consiste semplicemente nello "scompattare" il file in una folder a piacere. Dopodichè bisogna settare alcuni parametri, in particolare:

- ✓ creare una variabile d'ambiente chiamata CATALINA_HOME e assegnarle il valore <TOMCAT_INSTALL_DIR>, che rappresenta la directory in cui si è deciso di installare il server web;
- ✓ la demo ASTRO si aspetta che Tomcat giri sulla porta 50000. Quindi bisogna verificare se nel file "server.xml" all'interno della folder "conf" della directory di Tomcat, esiste un 'Connector' su questa porta. Se non esiste aggiungere il seguente pezzo di codice

```
<Connector port="50000" maxHttpHeaderSize="8192"
      maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
      enableLookups="false" redirectPort="8443" acceptCount="100"
      connectionTimeout="20000" disableUploadTimeout="true" />
```

subito dopo la seguente linea di commento

```
<!-- You should set jvmRoute to support load-balancing via AJP ie :
    <Engine name="Standalone" defaultHost="localhost" jvmRoute="jvm1">
    -->
```

3.4 ActiveBPEL Engine 2.0

L'ActiveBPEL engine è un prodotto freeware che permette di eseguire deployment e running di processi BPEL su un application server. Può essere scaricato dal sito dell'active-endpoints http://www.active-endpoints.com. La sua installazione è particolarmente semplice, poiché consiste nello "scompattare" il file ed eseguire `install.bat` (per sistema Windows) contenuto all'interno della folder. Tale esecuzione comporta la copia del contenuto della folder 'lib' in $CATALINA_HOME/shared/lib e crea la directory $CATALINA_HOME/bpr, dove i processi BPEL .bpr sono archiviati e deployed.

Il deployment di un BPEL process avviene eseguendo un packaging del file .bpel, wsdl relativi e files di deployment .xml e .pdd in un file .bpr, che viene copiato nella directory /bpr del server e

rilevato automaticamente e deployed quando il server è running. Il deployed process può essere monitorato via Web browser alla seguente URL http://localhost:50000/BpelAdminExt ed invocato da programmi client appositi per l'invocazione di Web Services.

3.5 Graphical Editing Framework & Graphical Modeling Framework Eclipse plugins

Questi due insiemi di plugins per Eclipse sono necessari per far funzionare l'aspetto visuale delle applicazioni Astro. Vengono installati con la classica procedura di installazione dei plugin di eclipse.

3.6 Astro wsToolset 1.8.0

Scaricabile dal sito del progetto http://www.astroproject.org. Per installare il Toolset basta eseguire la seguente linea di comando:

```
java -jar wsToolset-1.8.0_installer.jar
```

Durante l'installazione sarà chiesto di indicare la folder in cui installare il Toolset. Il pathname della folder di installazione non deve contenere spazi bianchi, e sarà creata se non esiste. Per completare l'installazione è necessario configurare l'applicazione settando alcune variabili d'ambiente:

- ✓ creare la variabile d'ambiente `wsTranslator_HOME` e assegnargli il valore `<WSTOOLSET_INSTALL_DIR>\tools\wsTranslator;`
- ✓ aggiornare la variabile d'ambiente PATH al seguente valore

  ```
  PATH=<WSTOOLSET_INSTALL_DIR>\tools\wsTranslator\bin;
  <WSTOOLSET_INSTALL_DIR>\tools\synTools\bin;
  <WSTOOLSET_INSTALL_DIR>\tools\NuSMV\bin
  ```

  dove `<WSTOOLSET_INSTALL_DIR>` corrisponde alla folder di installazione che è stata scelta appunto durante l'installazione.

Il package wsToolset è composto da quattro programmi necessari a vari stadi della composizione, attivabili da linea di comando:

- ➢ *Astro wsToolset 1.8.0 - wsTranslator 0.14.0*: il programma wsTranslator è l'importantissimo primo modulo adibito alle traduzioni dei files di "coreografia" (.chor) in vari formati di STS, ad esempio files .smv o Spin, per poi realizzare il prodotto parallelo dei Component Services e preparare il terreno per il planning via Model Checking;

- *Astro wsToolset 1.8.0 - synTools 0.13.1*: il package synTools contiene due programmi, *wmon* e *wsynth*: il primo è adibito al monitoring dei processi BPEL, e quindi alla generazione del codice Java che controlla a runtime il verificarsi di eventi d'interesse e fa rapporto all'utente nelle schermate di monitoring dei processi (accessibili via browser); la seconda applicazione, wsynth, è la responsabile del vero e proprio processo di sintesi che restituisce il file concrete BPEL eseguibile e con un certo livello di ottimizzazione per realizzare il composite service obiettivo;
- *Astro wsToolset 1.8.0 NuSMV 2.2.5* - prodotto da terze parti, NuSMV è essenziale per eseguire operazioni di model checking su STSs, il che è al cuore dell'approccio Astro.

3.7 Astro wsMonitor 1.6.0

Scaricabile dal sito del progetto http://www.astroproject.org. Per installare il wsMonitor basta eseguire la seguente linea di comando:

```
java -jar wsMonitor-1.6.0_installer.jar
```

Durante l'installazione verrà chiesto di indicare la folder in cui installare il wsMonitor: il wsMonitor deve essere installato nella folder CATALINA_HOME dell'engine ActiveBpel che si intende usare.

Si è gia accennato a come l'installazione del BPEL Engine permetta di utilizzare una schermata accessibile via browser per monitorare i deployed BPEL processes; questa applicazione Astro è un'estensione a Tomcat che aggiunge funzionalità ulteriori di monitoring online all'interfaccia dell'Engine; del codice Java viene pre-generato ed eseguito a runtime dal wsMonitor per cercare situazioni insolite o di errore e fare rapporto all'utente via browser. È quindi sostanzialmente un'estensione dell'interfaccia offerta dal BPEL engine per il monitoraggio dei processi.

3.8 Astro wsRequirement 0.2.0 Eclipse plugin

Questa plugin permette di integrare i files di input necessari alla composizione (Abstract BPEL processes per component and target services, EAGLE requirements) creando in output un unico file xml con estensione .chor (file "di coreografia" che caratterizza completamente il problema) da dare in pasto a wsTranslator e wSynth per il processo di composizione. Oltre alla creazione del file .chor, la plugin di Eclipse permette anche di analizzare le sue proprietà (come i component services, i "main" e "recovery" goals, gli interessi nel monitoring ecc) tramite un'efficace GUI.

3.9 Astro wsChainManager 2.4.0 Eclipse plugin

La plugin permette di eseguire, a partire da un file .chor, vari servizi di composizione automatica, verification offline e preparazione per l'online monitoring, attivabili tramite la pressione di un singolo tasto. Il nome deriva dallo stile di esecuzione: le varie funzionalità sono delle *catene* di chiamate ai vari componenti dell'Astro Suite.

3.10 Astro wsAnimator 0.0.7 Eclipse plugin

La plugin è dedicata alla simulazione dei composite services impiegando lo stile grafico di ActiveWebFlow/ActiveBPEL Designer per mandare in esecuzione diverse tipologie di scenari di simulazione; usa files grafici con estensioni .adf. Questa parte però è ancora in fase di ingegnerizzazione e quindi attualmente non è banale creare file .adf per la simulazione dei processi.

3.11 Astro wsUseCases 1.0.0 Eclipse plugin

Questa plugin è semplicemente un insieme di folders che rappresentano due esempi di demo di composizione, chiamati VOS e VTA, rispettivamente dedicati a scenari “classici” come l'acquisto User-Store-Bank e la prenotazione User-Hotel-Flight; i folders contengono tutto l'occorrente per testare tutte le funzionalità offerte. È importante studiarne la struttura per capire il funzionamento del Toolset.

3.12 ActiveBpel Designer

Per creare i file abstract BPEL ho utilizzato ActiveBpel Designer 2.0; non è l’ultima versione ma è stato necessario utilizzare questa perché i tool di Astro allo stato attuale supportano WS-BPEL 1.1. Inoltre attualmente la loro roadmap non prevede attività legate al supporto WS-BPEL 2.0. Il designer è un prodotto freeware, scaricabile dal sito dell'Active-Endpoints http://www.active-endpoints.com, ha documentazione e tutorial.
In realtà per creare file .bpel e relativi file .wsdl, si possono utilizzare altri tool, ma bisogna fare attenzione che il .bpel/.wsdl che ne esce sia compatibile con il sottoinsieme di bpel wsdl definito nei file .xsd che sono utilizzati dal traduttore (wsTranslator).

Capitolo 4 – La demo VTA

4.1 Scenario di funzionamento

Il cuore di questa tesina è stato cercare di analizzare la demo VTA già esistente per poi provare a realizzare una nuova demo funzionante, ma come detto nell'introduzione ciò non è stato possibile a causa di numerosi problemi.

Per capire meglio la struttura della demo VTA basta analizzare la seguente figura:

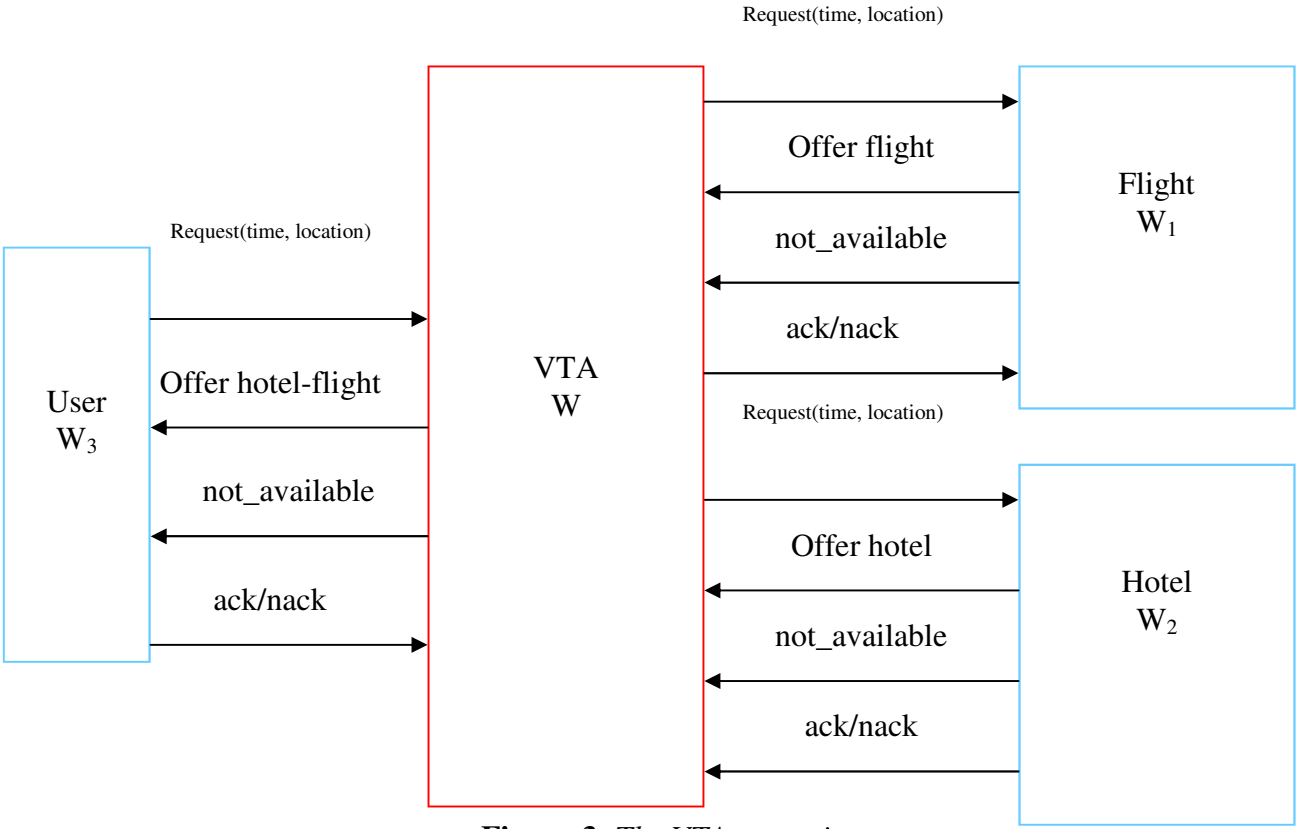


Figura 3: *The VTA scenario*

I component services sono: User, Flight e Hotel. VTA è il target composite service che funge da mediatore tra W_1 e W_2, W_3.

Lo User prepara il messaggio di richiesta contenente il periodo e il luogo del viaggio.

VTA, ricevendo la richiesta dallo User, a sua volta prepara un messaggio di richiesta per il service Flight e un messaggio di richiesta per il service Hotel.

Flight verifica la disponibilità del volo per il periodo e il luogo indicati nella richiesta e può inviare a VTA l'offerta o un not_available. Hotel verifica la disponibilità dell'hotel per il periodo e il luogo indicati nella richiesta e può inviare a VTA l'offerta o un not_available. VTA se riceve l'offerta da Flight e Hotel, la invia allo User: se lo User l'accetta, allora VTA invia un ack a Flight e Hotel,

altrimenti un nack; oppure VTA può ricevere un not_available da Flight o da Hotel o da entrambi e di conseguenza invia un not_available allo User.

4.2 Dagli STS agli abstract BPEL

Il primo passo per la realizzazione di una demo, è partire dalla rappresentazione in STS dei servizi W_1, W_2 e W_3 e tradurli in file BPEL astratti. Presentiamo i tre component services e il target composite service della demo VTA.

4.2.1 Flight component service

L'STS del Flight component service è il seguente:

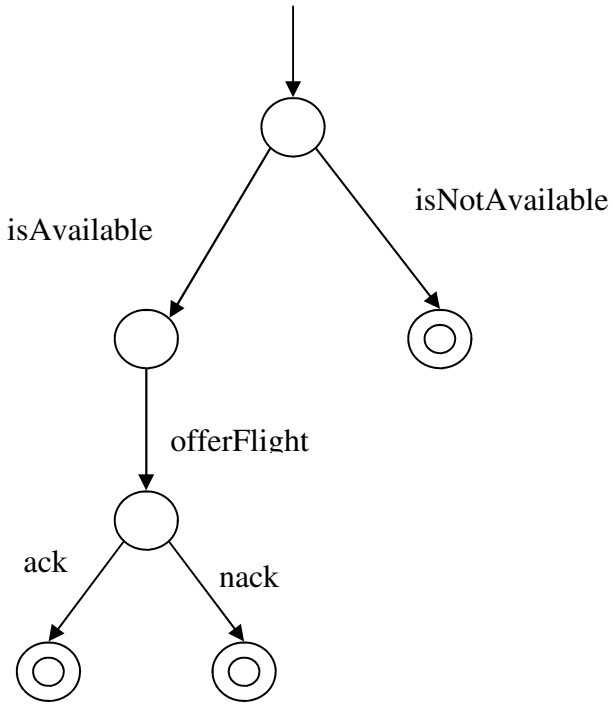


Figura 4: *STS del Flight component service*

Si parte da uno stato iniziale in cui si riceve la richiesta; viene verificata la disponibilità del volo per il periodo e il luogo indicati nella richiesta: se non c'è disponibilità si invia un not_available e si termina passando quindi in uno stato finale. Altrimenti si prepara l'offerta e tramite la transizione "offerFlight" si passa in un nuovo stato di attesa: se è stata accettata la richiesta si riceve un ack altrimenti un nack; in entrambi i casi si termina e si passa in uno stato finale.

Entrando più nel dettaglio possiamo dire che nello stato iniziale si riceve una richiesta, quindi la prima attività da inserire nel file BPEL è *Receive*. Da questa partono le due transizioni e quindi è necessario inserire come successiva attività lo *Switch* con un *Case condition* corrispondente alla

transizione “isAvailable” e un *otherwise* corrispondente alla transizione “isNotAvailable”, con la quale si prepara il messaggio not_available per l’utente, tramite le attività *Assign* e *Invoke*, e si termina con un’attività *Empty*. La transizione “isAvailable” porta in uno stato in cui vengono invocate le t-transitions del Web service, quindi è necessario assegnare i valori delle variabili contenute nel messaggio ricevuto alle variabili interne del Web service, cioè si inserisce l’attività *Assign*. Successivamente l’invocazione del web service avviene tramite l’attività *Invoke*. Inviata l’offerta, si rimane in attesa di un ack/nack, quindi è necessario inserire l’attività *Pick* con due attività *onMessage*, una relativa all’operazione di ack, l’altra all’operazione di nack. Quindi il file .bpel risultante è il seguente:

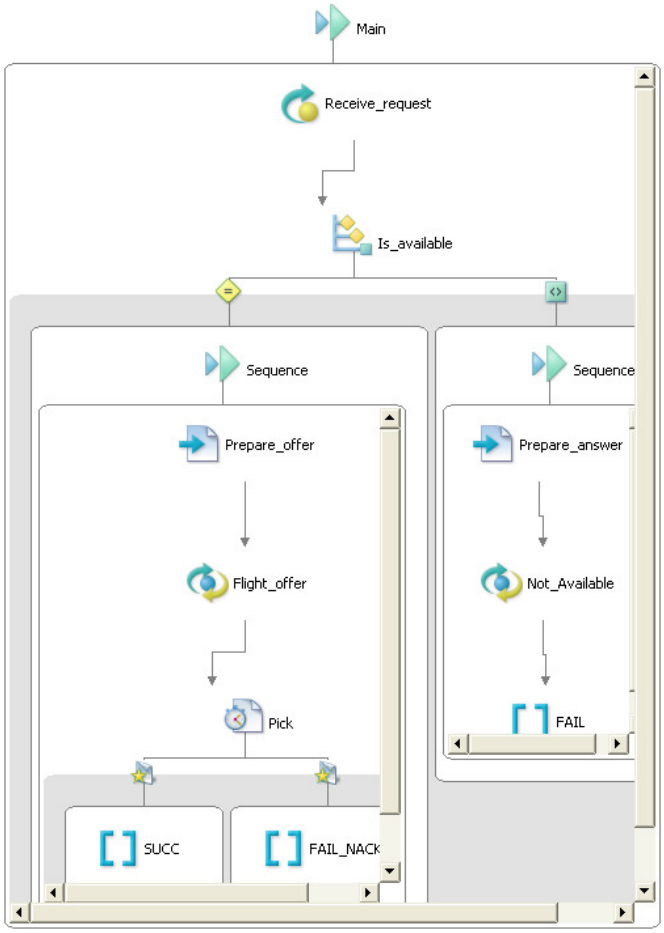


Figura 5: *File Flight_ABS.bpel*

4.2.2 Hotel component service

L'STS del Hotel component service è praticamente simile a quello del Flight ed è il seguente:

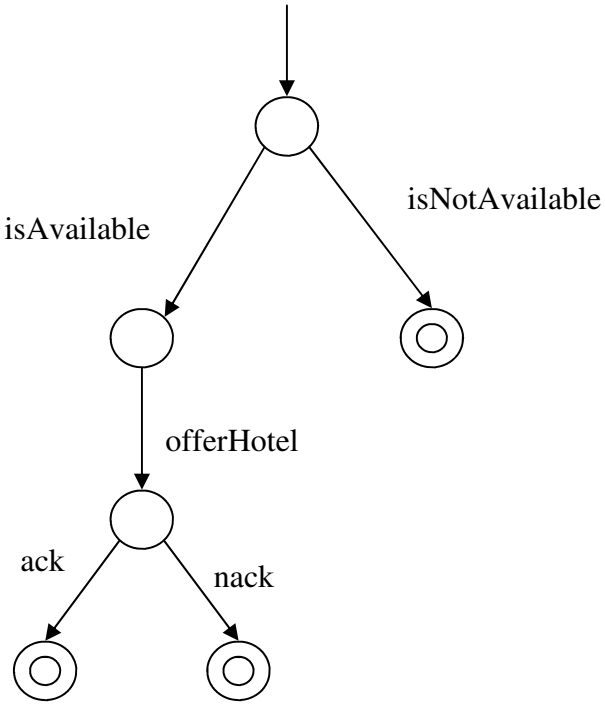


Figura 6: *STS del Hotel component service*

Anche in questo caso si parte da uno stato iniziale in cui si riceve la richiesta; viene verificata la disponibilità dell'hotel per il periodo e il luogo indicati nella richiesta: se non c'è disponibilità si invia un not_available e si termina passando quindi in uno stato finale. Altrimenti si prepara l'offerta e tramite la transizione "offerHotel" si passa in un nuovo stato di attesa: se è stata accettata la richiesta si riceve un ack altrimenti un nack; in entrambi i casi si termina e si passa in uno stato finale.

Entrando più nel dettaglio possiamo dire che nello stato iniziale si riceve una richiesta, quindi la prima attività da inserire nel file BPEL è *Receive*. Da questa partono le due transizioni e quindi è necessario inserire come successiva attività lo *Switch* con un *Case condition* corrispondente alla transizione "isAvailable" e un *otherwise* corrispondente alla transizione "isNotAvailable", con la quale si prepara il messaggio not_available per l'utente, tramite le attività *Assign* e *Invoke*, e si termina con un'attività *Empty*. La transizione "isAvailable" porta in uno stato in cui vengono invocate le t-transitions del Web service, quindi è necessario assegnare i valori delle variabili contenute nel messaggio ricevuto alle variabili interne del Web service, cioè si inserisce l'attività *Assign*. Successivamente l'invocazione del web service avviene tramite l'attività *Invoke*. Inviata l'offerta, si rimane in attesa di un ack/nack, quindi è necessario inserire l'attività *Pick* con due

attività *onMessage*, una relativa all'operazione di ack, l'altra all'operazione di nack. Quindi il file .bpel risultante è il seguente:

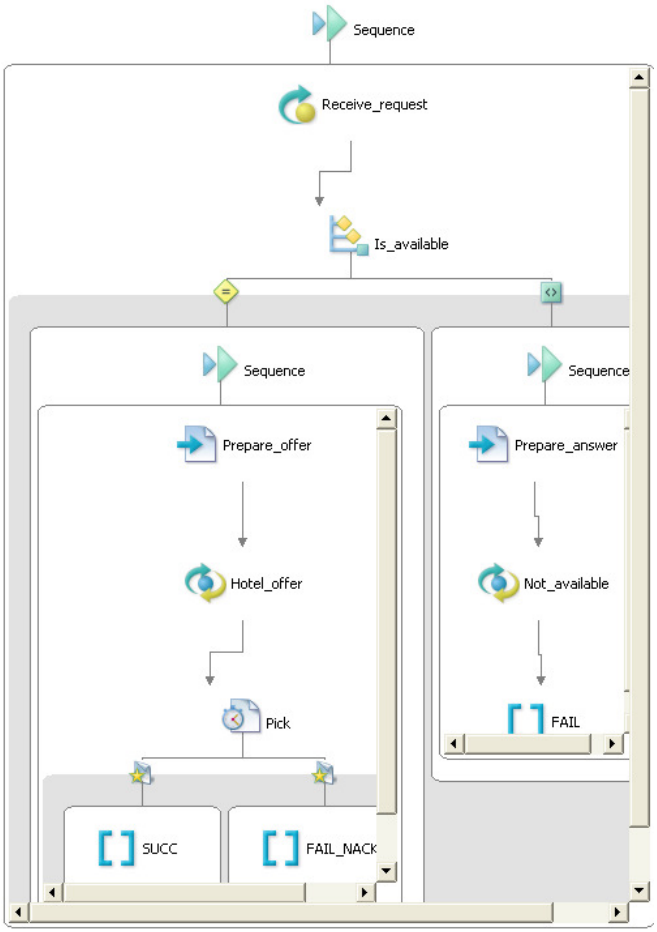


Figura 7: *File Hotel_ABS.bpel*

4.2.3 User component service

L'STS dello User component service è il seguente:

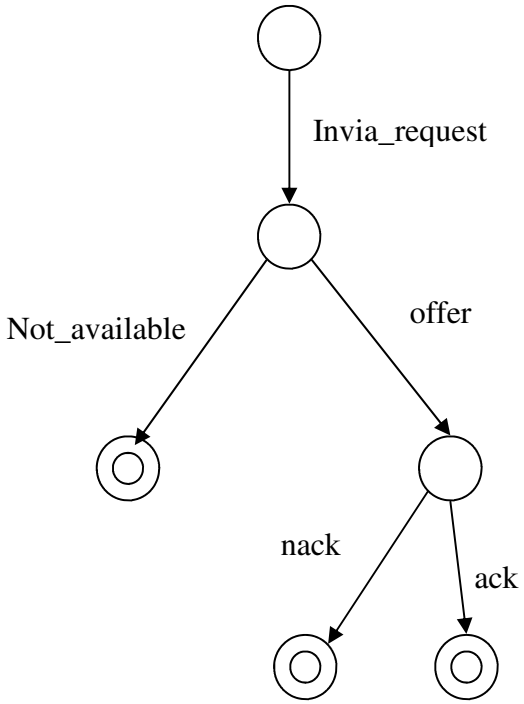


Figura 8: *STS dello User component service*

Lo User parte da uno stato iniziale in cui prepara la richiesta specificando il periodo e la locazione del viaggio e la invia con la transizione "Invia_request", passando in nuovo stato di attesa. Se si riceve un not_available, allora si termina in uno stato finale; se si riceve un'offerta, lo User decide se accettare o no e quindi invia un ack o un nack rispettivamente, passando in entrambi i casi in uno stato finale.

Entrando nel dettaglio possiamo dire che nello stato iniziale si deve preparare la richiesta, quindi tramite l'attività *Assign* si assegnano i valori alle variabili che vengono date in input al Web service invocato tramite l'attività *Invoke*. Inviata la richiesta si rimane in attesa, perciò si inserisce un'attività *Pick* con due attività *onMessage*, corrispondenti alle transizioni "Not_available" e "offer". Nel primo caso si termina in un stato finale rappresentato da un'attività *Empty*. Nel secondo caso partono due transizioni, quindi si inserisce un'attività *Switch* con un *Case condition* in cui viene accettata l'offerta e tramite le attività *Assign* e *Invoke* si invia un ack; e un *otherwise* in cui viene rifiutata l'offerta e sempre tramite le attività *Assign* e *Invoke* si invia invece un nack. In entrambi i casi si termina con un'attività *Empty*. Quindi il file .bpel risultante è il seguente:

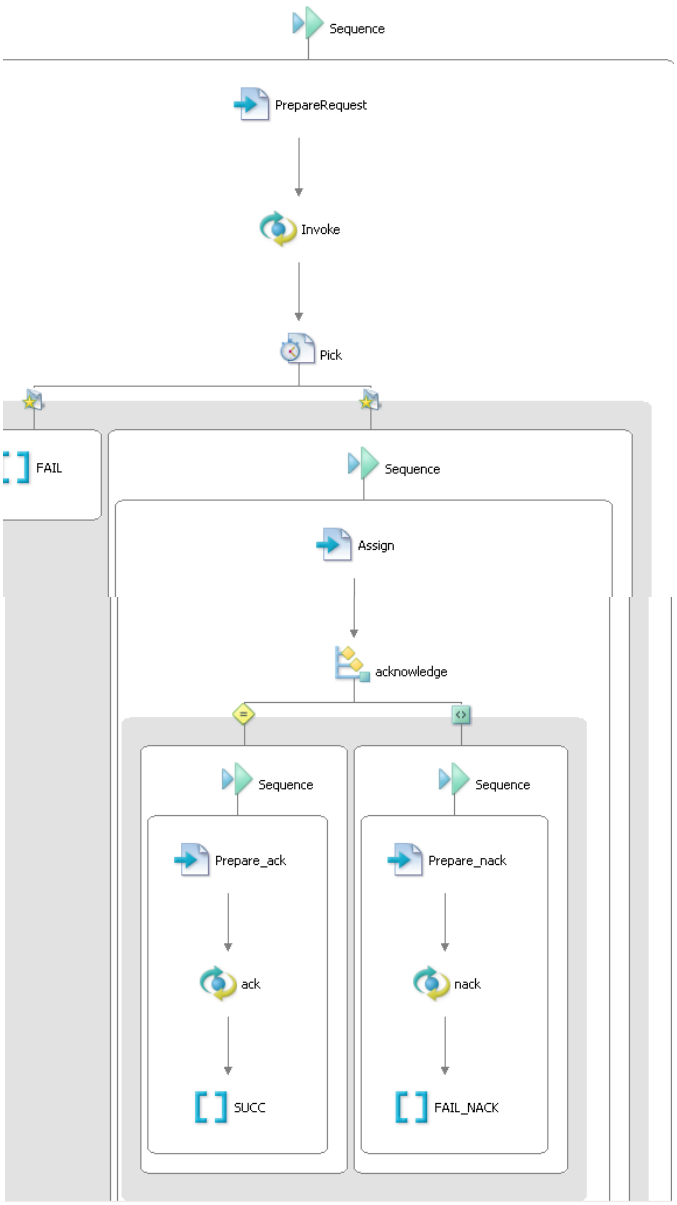


Figura 9: *File User_ABS.bpel*

4.2.4 VTA Composite service

L'STS del VTA Composite service è il seguente:

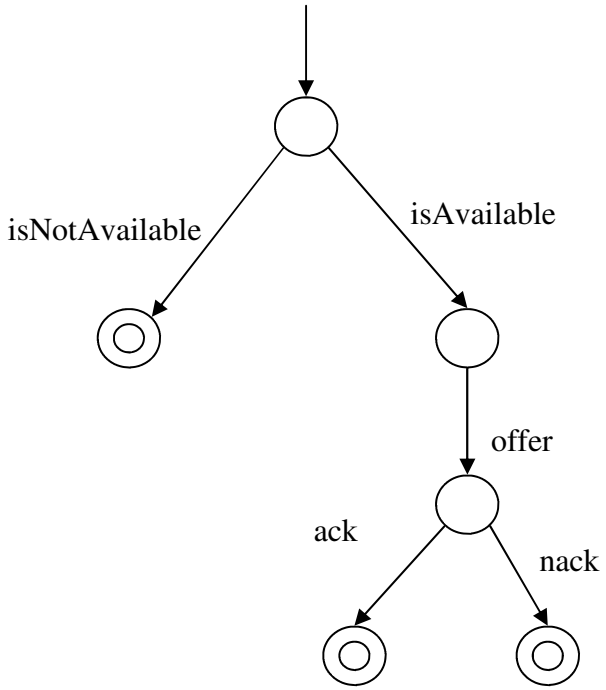


Figura 10: *STS del VTA composite service*

Si parte in uno stato iniziale in cui si riceve la richiesta dallo User e si verifica la disponibilità: se non è disponibile si prepara un messaggio di not_available e si termina in uno stato finale; altrimenti si prepara l'offerta allo User e tramite la transizione "offer" si passa in un nuovo stato di attesa: se è stata accettata la richiesta si riceve un ack altrimenti un nack; in entrambi i casi si termina e si passa in uno stato finale.

Entrando più nel dettaglio possiamo dire che nello stato iniziale si riceve una richiesta, quindi la prima attività da inserire nel file BPEL è *Receive*. Da questa partono le due transizioni e quindi è necessario inserire come successiva attività lo *Switch* con un *Case condition* corrispondente alla transizione "isNotAvailable" con la quale si prepara il messaggio not_available per l'utente, tramite le attività *Assign* e *Invoke*, e si termina con un'attività *Empty*; e un *otherwise* corrispondente alla transizione "isAvailable". Questa porta in uno stato in cui vengono invocate le t-transitions del Web service, quindi è necessario assegnare i valori delle variabili contenute nel messaggio ricevuto alle variabili interne del Web service, cioè si inserisce l'attività *Assign*. Successivamente l'invocazione del web service avviene tramite l'attività *Invoke*. Inviata l'offerta, si rimane in attesa di un ack/nack, quindi è necessario inserire l'attività *Pick* con due attività *onMessage*, una relativa all'operazione di ack, l'altra all'operazione di nack. Quindi il file .bpel risultante è il seguente:

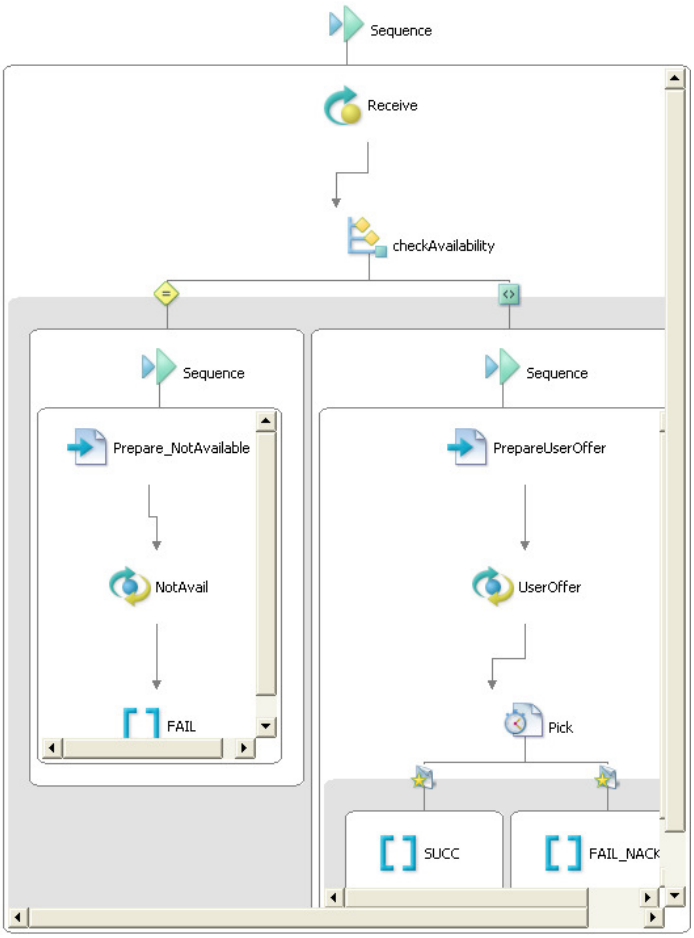


Figura 11: *File VTA_ABS.bpel*

4.3 Creazione del file .chor

Una volta creati i file abstract .bpel e i rispettivi .wsdl, ci si posiziona in eclipse e si caricano i progetti nel workspace tramite il task File→Import → Astro suite wsUsesCases. A questo punto i progetti sono visibili nella view Navigator a sinistra.

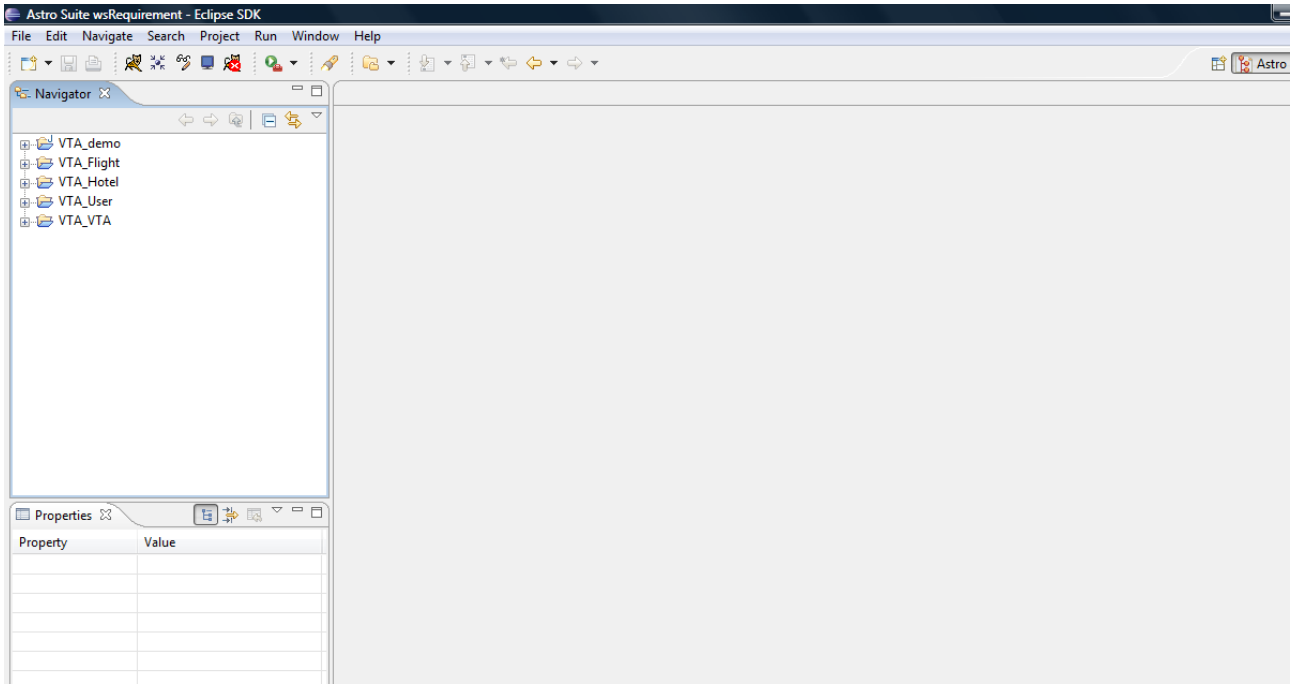


Figura 12: *View Navigator*

Questa demo è già fornita completa del file .chor, però di seguito descriviamo lo stesso come viene creato questo file.

Tutti gli input sono riuniti insieme tramite un Wizard sviluppato da Astro per la creazione di files di coreografia (.chor files) grazie alla plugin wsRequirement. La realizzazione del file .chor si struttura in 5 passi:

- ✓ Definizione del nome del file .chor e della directory di appartenenza;
- ✓ Specifica dei file .bpel e relativi .wsdl da comporre;
- ✓ Specifica dei Process references for Composition;
- ✓ Specifica dei Process references for Verification;
- ✓ Specifica dei Process references for Monitoring.

Di seguito vengono mostrate delle immagini a titolo di esempio, non riferibili alla demo VTA, corrispondenti ai 5 passi della creazione.

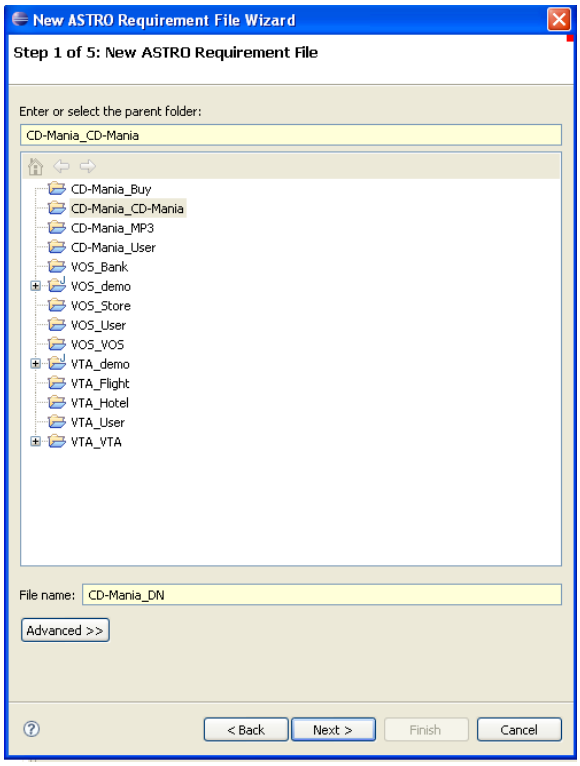


Figura 13*: Step 1*

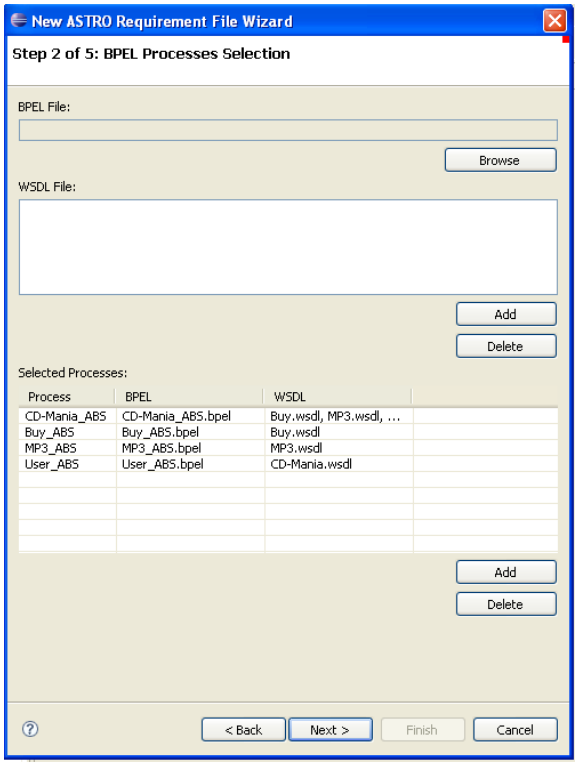


Figura 14*: Step 2*

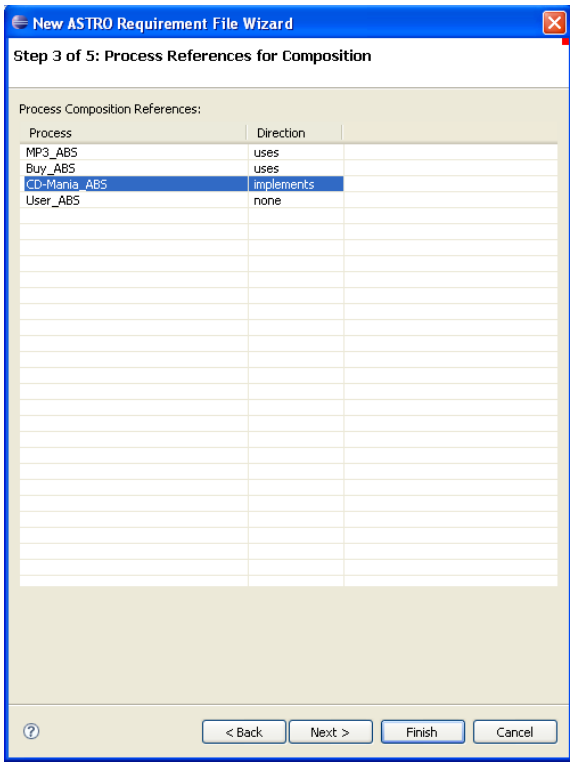


Figura 15*: Step 3*

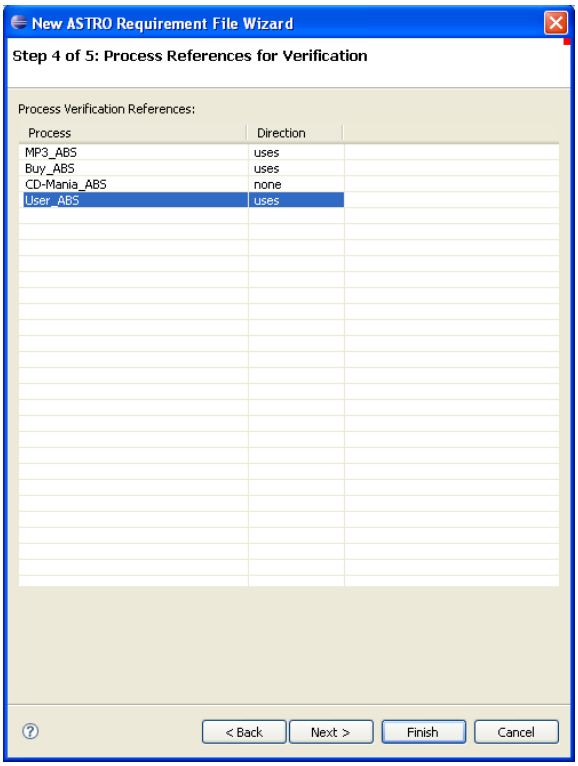


Figura 16*: Step 4*

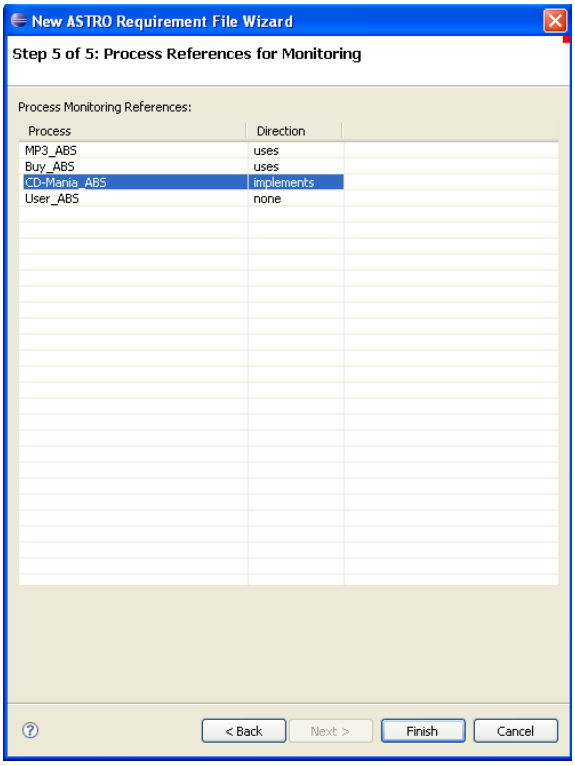


Figura 17*: Step 5*

Il file VTA_DN.chor include le descrizioni dei processi, il main goal, i recovery goals (che non devono avere molti vincoli, altrimenti il traduttore farà fatica a trovare un piano), le proprietà d'interesse da monitorare, quelle da verificare, ed altro.

Per quanto riguarda la definizione del goal, esso è composto da una parte di definizione del flusso di controllo (control flow) e da una parte di definizione del flusso dei dati (data flow). Entrambe queste parti sono definibili tramite l'utilizzo di wsRequirement. La parte di controllo viene definita tramite una semplice stringa mentre la parte di dati viene definita in modo grafico con un grafo che si chiama datanet. I file .datanet sono dei files di servizio del wsRequirment utilizzati per memorizzare la definizione della componente di data flow del goal. Da un punto di vista interno la parte di control flow del goal viene utilizzata come un normale goal di pianificazione che definisce gli stati finali del "main goal" e del "recovery goal" mentre la parte di dataflow va a modellare un insieme di macchine a stati che arricchiscono il dominio di planning.

La plugin wsRequirement consente anche di ispezionare il file coreografico tramite delle views:

- **Process definition**→contiene la definizione dei processi facente parte della composizione;
- **Composition ControlFlow**→contiene la specifica dei main e recovery goal, che è possibile anche modificare;
- **Composition DataFlow**→contiene il grafo datanet, che può essere creato con le apposite Palette;

- **Monitor**→contiene le proprietà da monitorare on-line, possono essere aggiunte o eliminate altre proprietà;
- **Verify**→contiene le proprietà da verificare off-line, possono essere aggiunte o eliminate altre proprietà;
- **XML**→contiene il codice xml del file .chor.

Di seguito vengono mostrate le views del file coreografico.

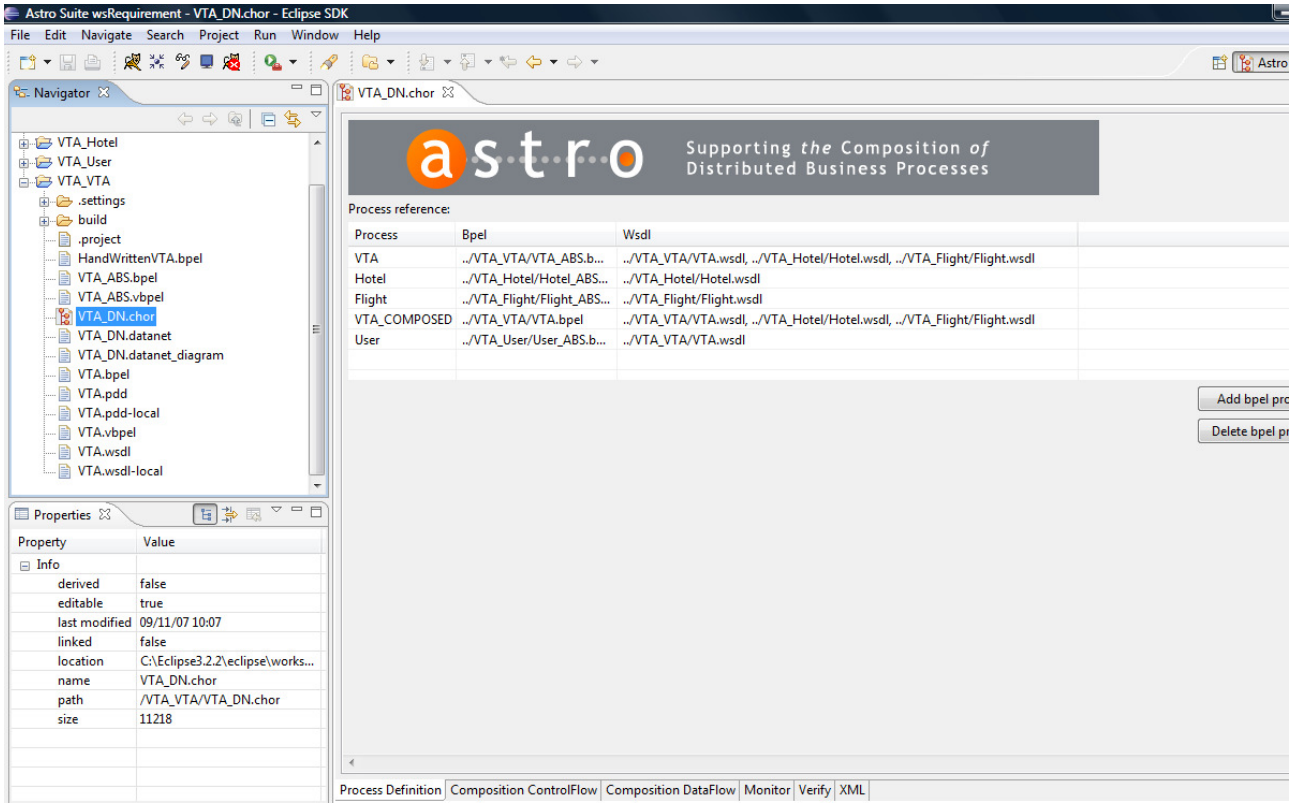


Figura 18: *View Process definition*

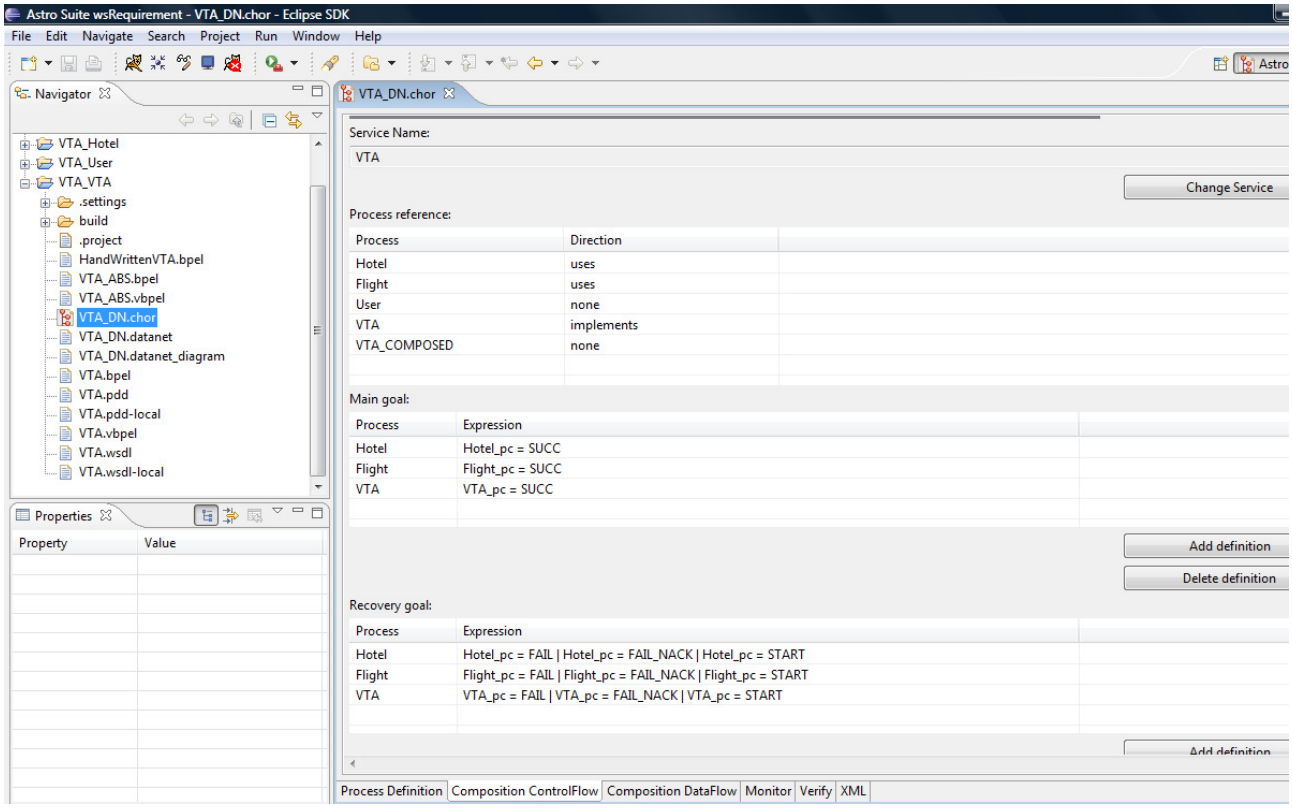


Figura 19: *View Composition ControlFlow*

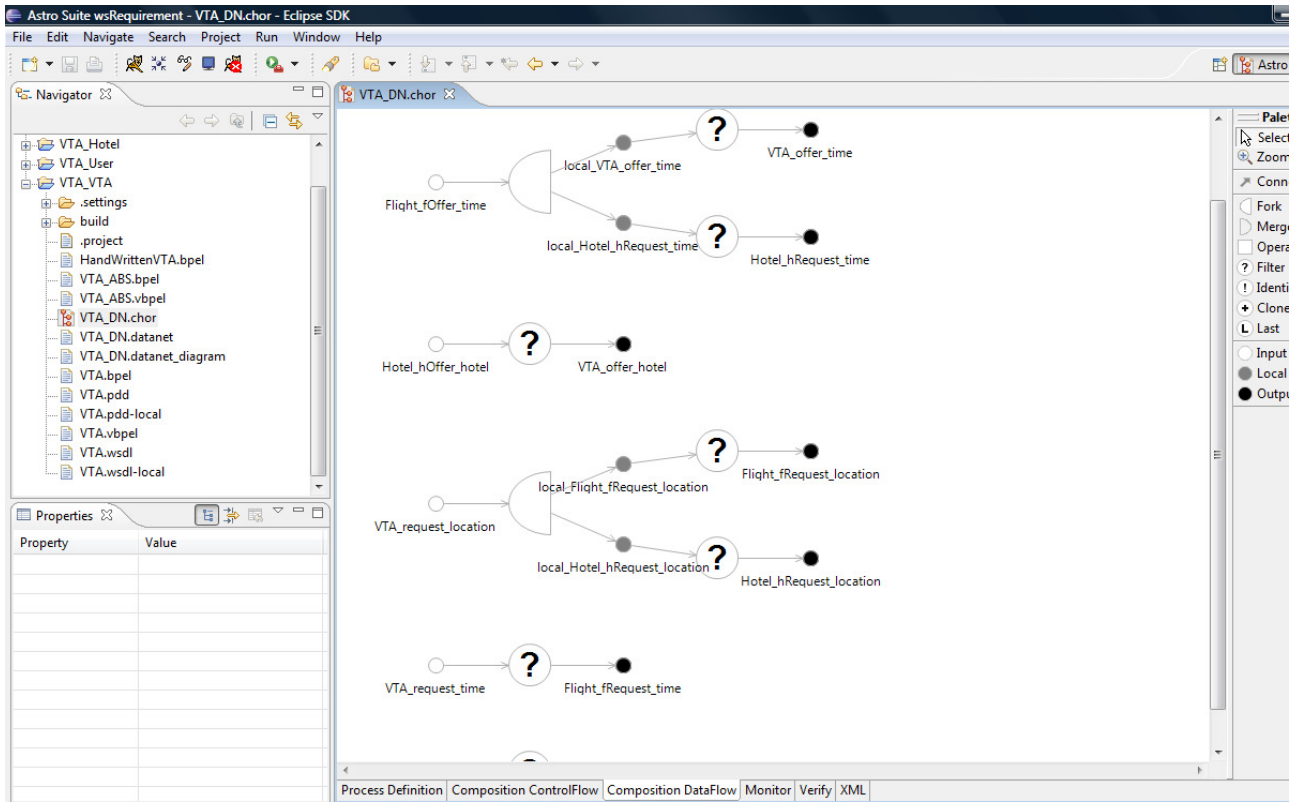


Figura 20: *View Composition DataFlow*

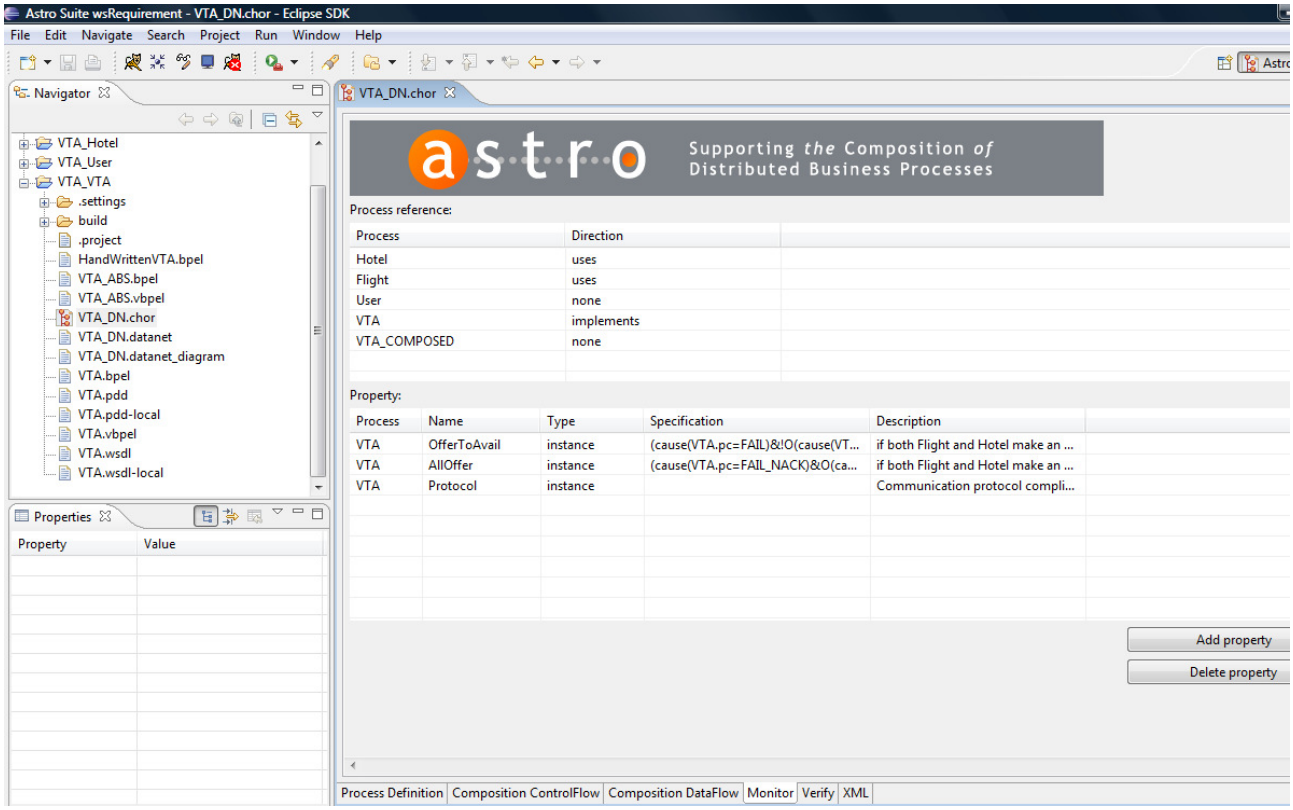


Figura 21: *View Monitor*

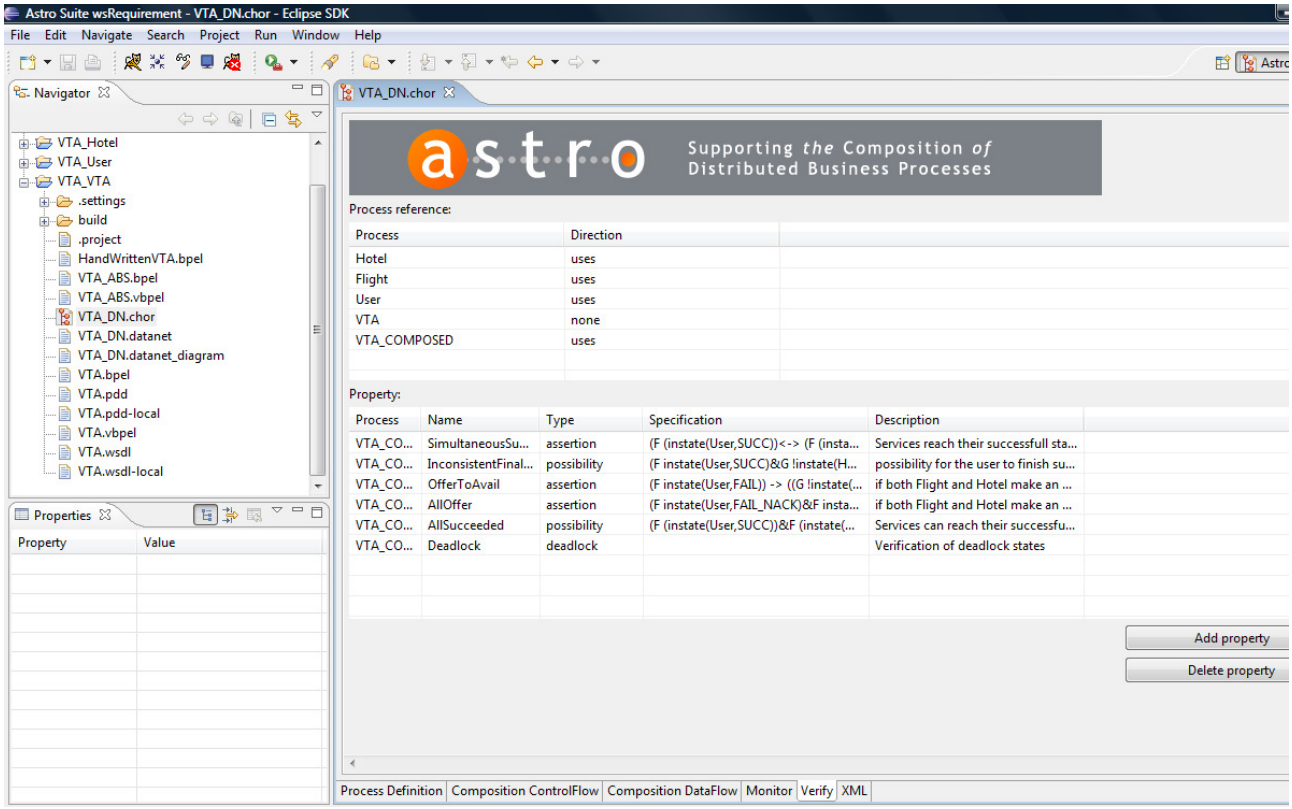


Figura 22: *View Verify*

4.4 La composizione automatica

Con l'installazione del wsToolset, vengono inseriti nella toolbar di eclipse 5 bottoni rispettivamente per: avviare Tomcat, iniziare la catena di Process Composition, avviare la Process Verification (offline), preparare le procedure di Process Monitoring, effettuare lo shutdown di Tomcat.

A questo punto, creato il file VTA_DN.chor (nel nostro caso viene già fornito), si può procedere con la composizione tramite wsChainManager: si deve avviare il Tomcat Server tramite toolbar, ed una volta che l'inizializzazione di Tomcat e del BPEL Engine è completata, si seleziona il file VTA_DN.chor (e tale azione rende attivi i tasti per composizione, monitoring e verification) e si invoca la funzionalità di Service Composition, che conduce ad una checklist di steps da affrontare.

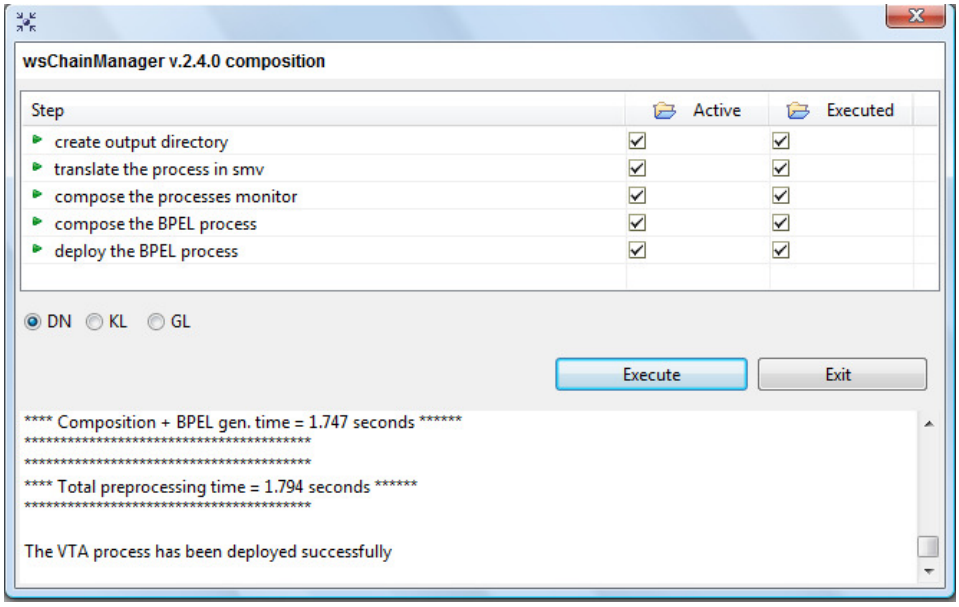


Figura 23: *wsChainManager nella Process Composition*

Nella finestra del wsChainManager possiamo distinguere la checklist delle operazioni in alto, e la message box in basso, che riporta dati sull'esecuzione, eventuali problemi e tempo impiegato, dimensioni delle strutture dati in gioco ecc.

Ciò che il wsChainManager fa è costruire il dominio D dal prodotto parallelo e risolvere il problema di planning via Model Checking, utilizzando i programmi NuSMV, wsTranslator, wSynth.

L'output finale della composizione è il file Concrete BPEL VTA.bpel automaticamente deployed su ActiveBPEL Engine, e può essere monitorato e testato via browser alla seguente URL: http://localhost:50000/BpelAdmin.

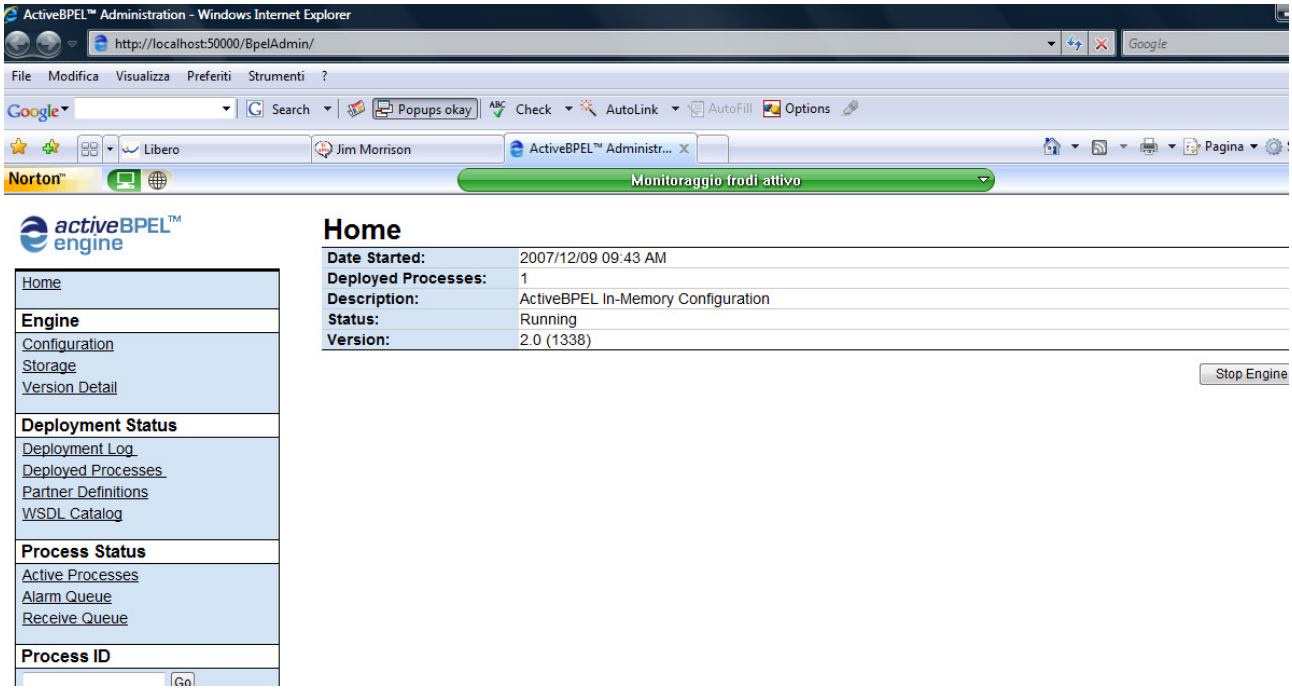


Figura 24: *Home di ActiveBpel Engine*

4.5 Process Verification

L'Astro Suite offre funzionalità per verificare proprietà del modello costruito (ovvero il file .chor) semplicemente a partire dal file di coreografia.

Si seleziona il file VTA_DN.chor e cliccando sul bottone  si avvia, tramite wsChainManager, la process verification. Trattandosi di una procedura offline, possiamo ricevere risposta immediatamente, tramite una schermata Web apposita:

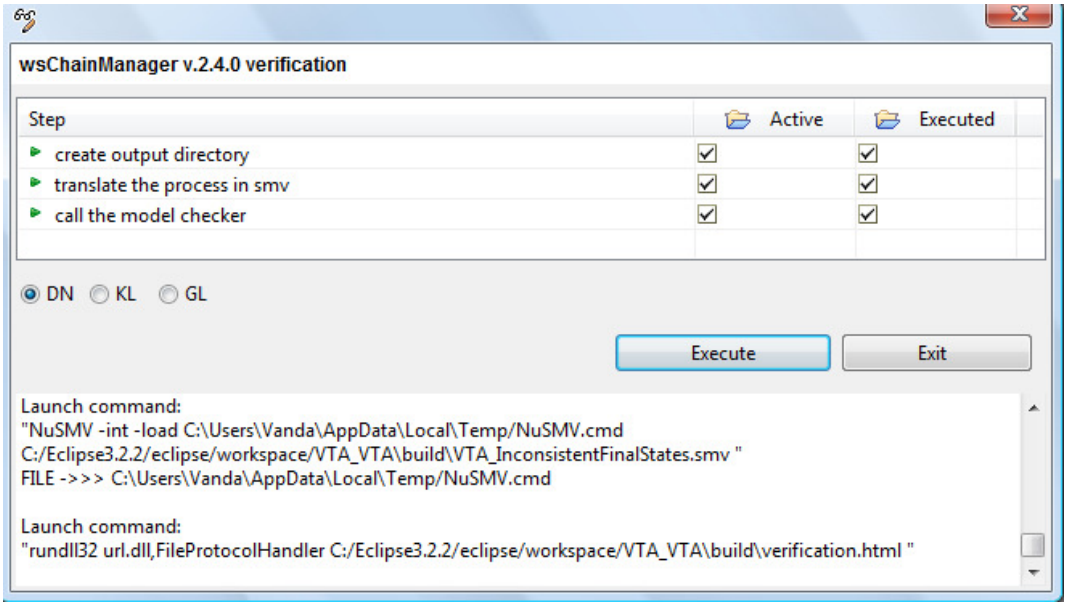
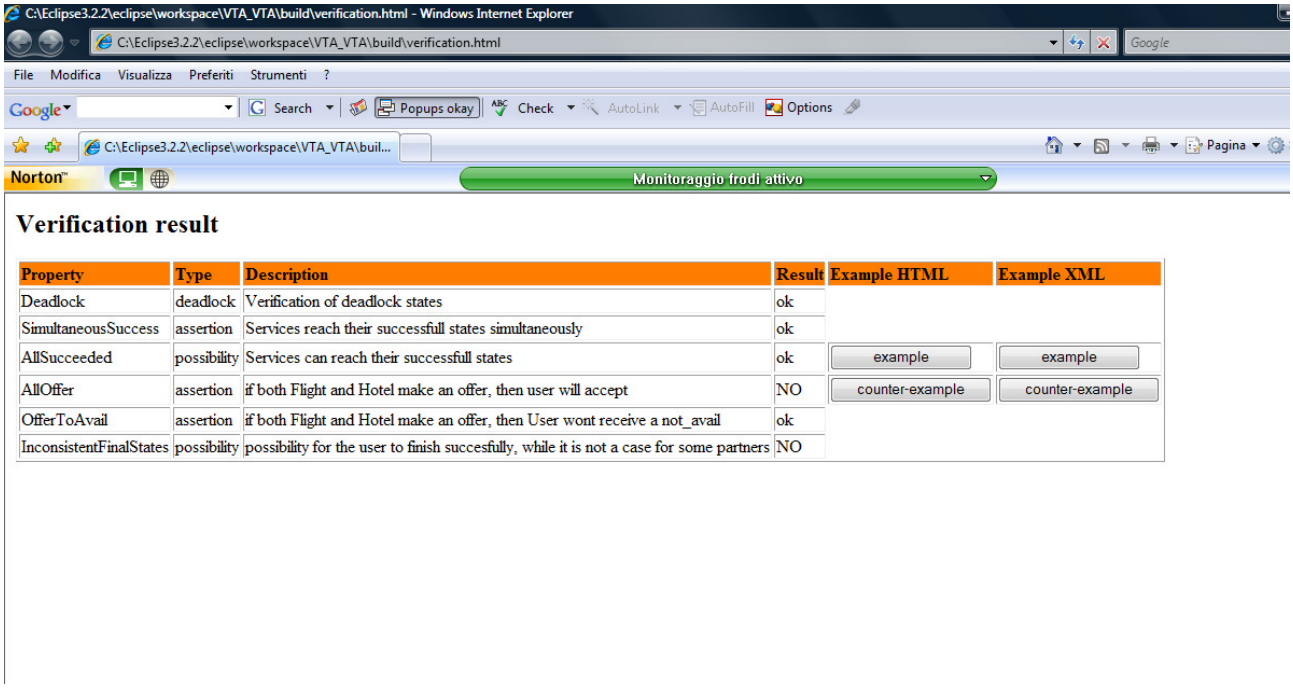


Figura 25: *wsChainManager per la Process Verification*



Verification result

Property	Type	Description	Result	Example HTML	Example XML
Deadlock	deadlock	Verification of deadlock states	ok		
SimultaneousSuccess	assertion	Services reach their successfull states simultaneously	ok		
AllSucceeded	possibility	Services can reach their successfull states	ok	example	example
AllOffer	assertion	if both Flight and Hotel make an offer, then user will accept	NO	counter-example	counter-example
OfferToAvail	assertion	if both Flight and Hotel make an offer, then User wont receive a not_avail	ok		
InconsistentFinalStates	possibility	possibility for the user to finish succesfully, while it is not a case for some partners	NO		

Figura 26: *Schermata web della verification*

Selezionando esempi e controesempi forniti, ci viene mostrato uno scenario ad alto livello stile UML Sequence Diagram:

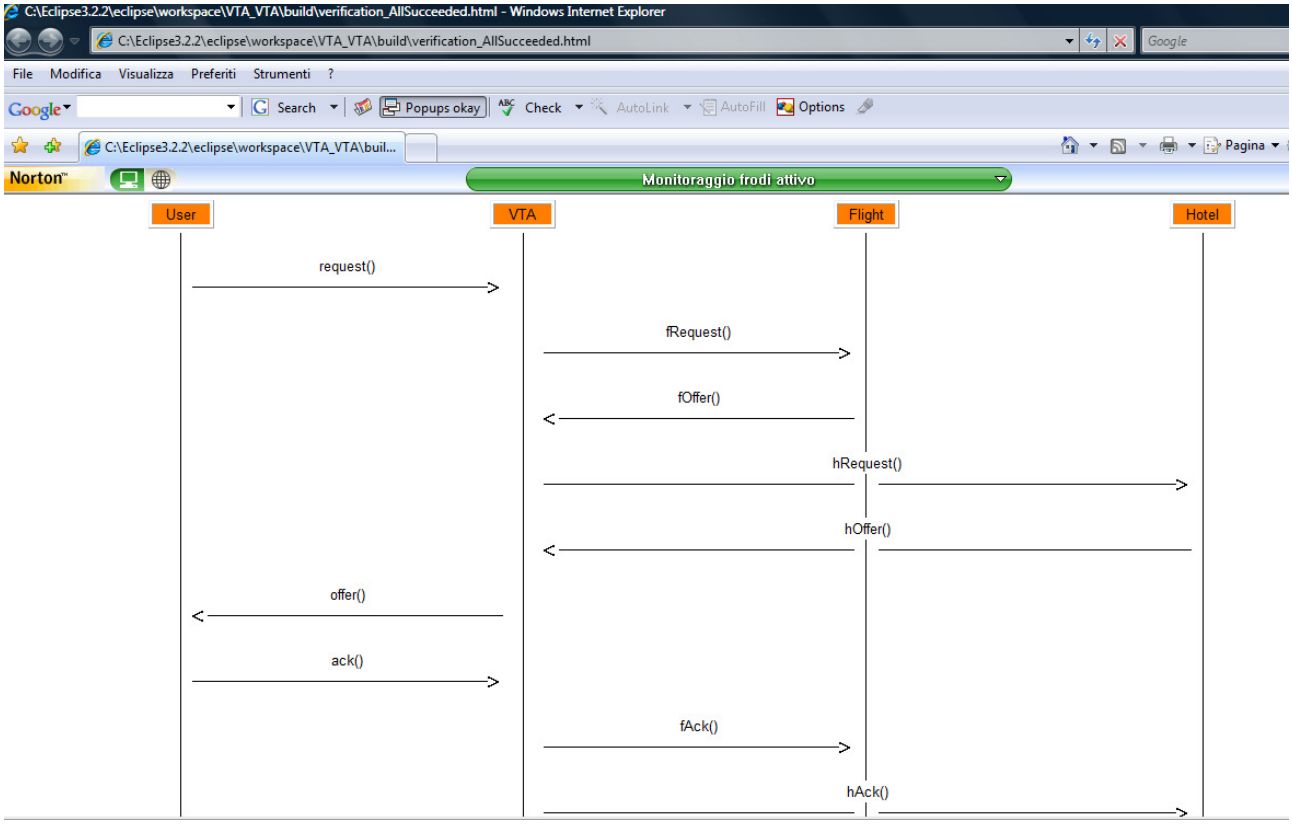


Figura 27: *Scenario di un esempio andato a buon fine*

4.6 Process Monitoring

La funzionalità di Monitoring è il corrispettivo online della Verification, e vuole fornire rapporti all'utente su stati anormali dell'esecuzioni di processi BPEL in esecuzione.

Tramite la consueta schermata wsChainManager vengono creati automaticamente dei files Java (i cui obiettivi sono definiti ancora una volta nel file .chor), che vengono messi in ascolto su esecuzioni del processo d'interesse per monitorarlo e fornire informazioni all'utente.

Sempre selezionando il file VTA_DN.chor e cliccando sul bottone  si avvia il process monitoring:

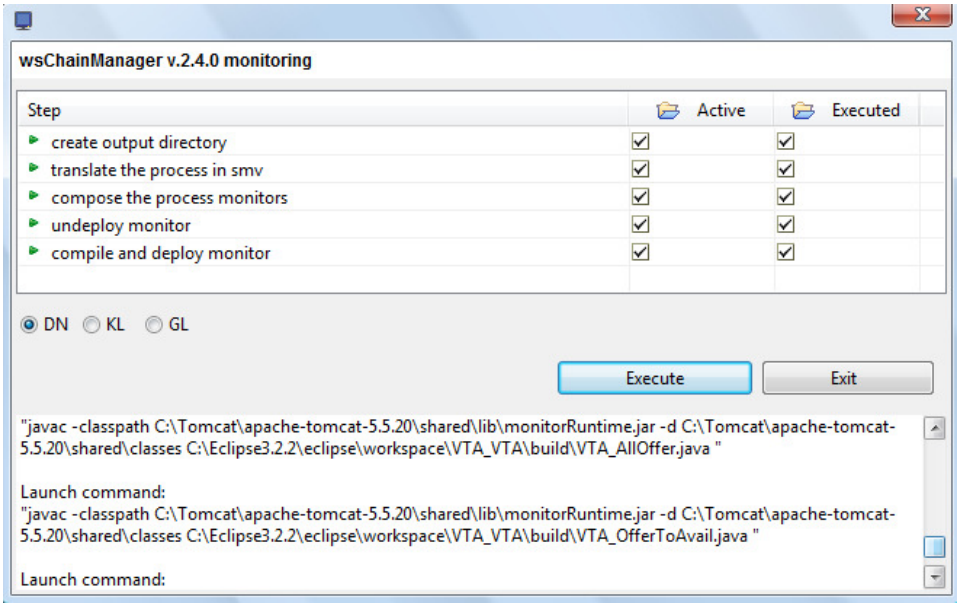


Figura 28: *wsChainManager per la Process Monitoring*

Di seguito viene mostrato uno dei file .java generati:

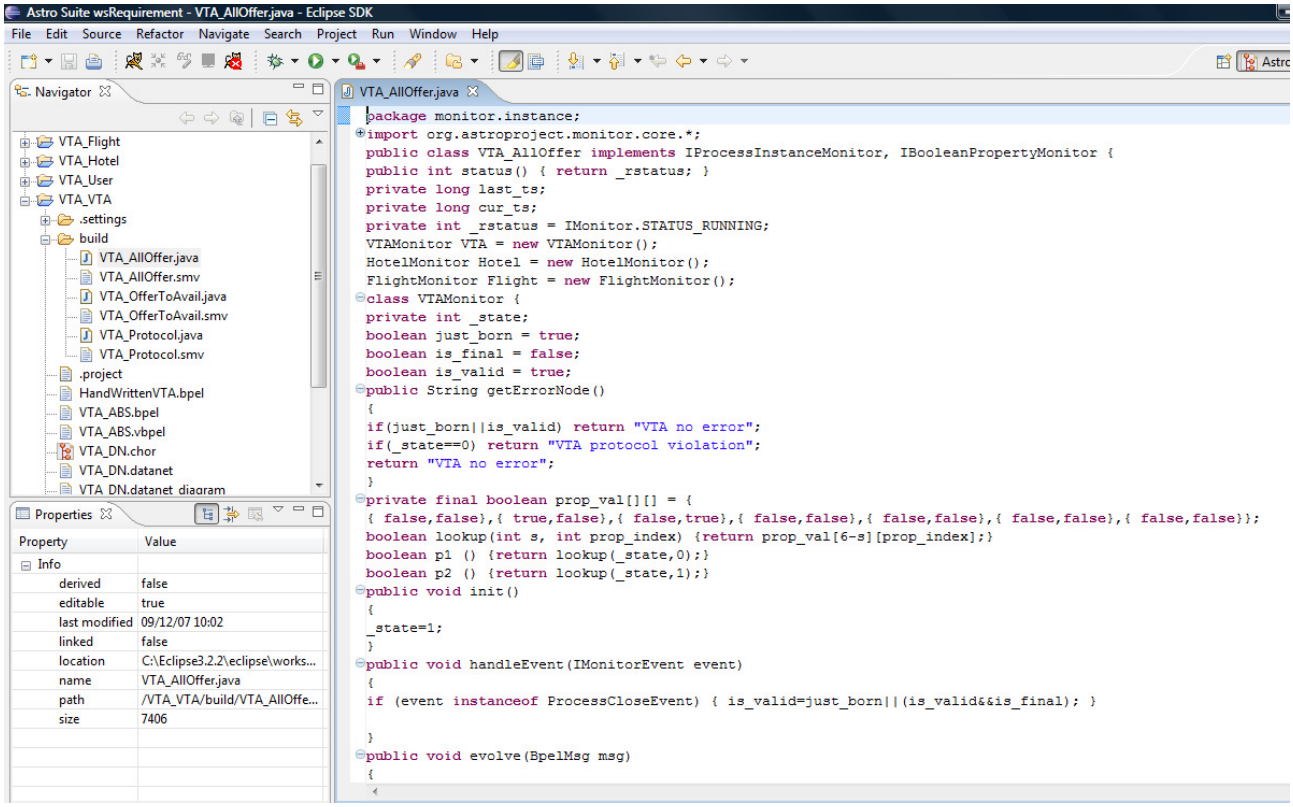


Figura 29: *File VTA_AllOffer.java generato dal process monitoring*

4.7 Process execution simulation

Tramite la plugin wsAnimator sviluppata dal team Astro ed utilizzando un formato file particolare .adf, è possibile mandare in esecuzione diversi scenari pre-programmati. Nella view Navigator si aprono i file Flight.adf, Hotel.adf e User.adf e per iniziare la simulazione si possono scegliere una delle quattro modalità di esecuzione:

- standard: è una simulazione interattiva, in cui noi gestiamo le varie scelte;
- nominal: è una simulazione che avviene in modo automatico e termina con successo;
- fail 1: è una simulazione che avviene in modo automatico e termina con un insuccesso del Flight;
- fail 2: è una simulazione che avviene in modo automatico e termina con un insuccesso di Hotel;

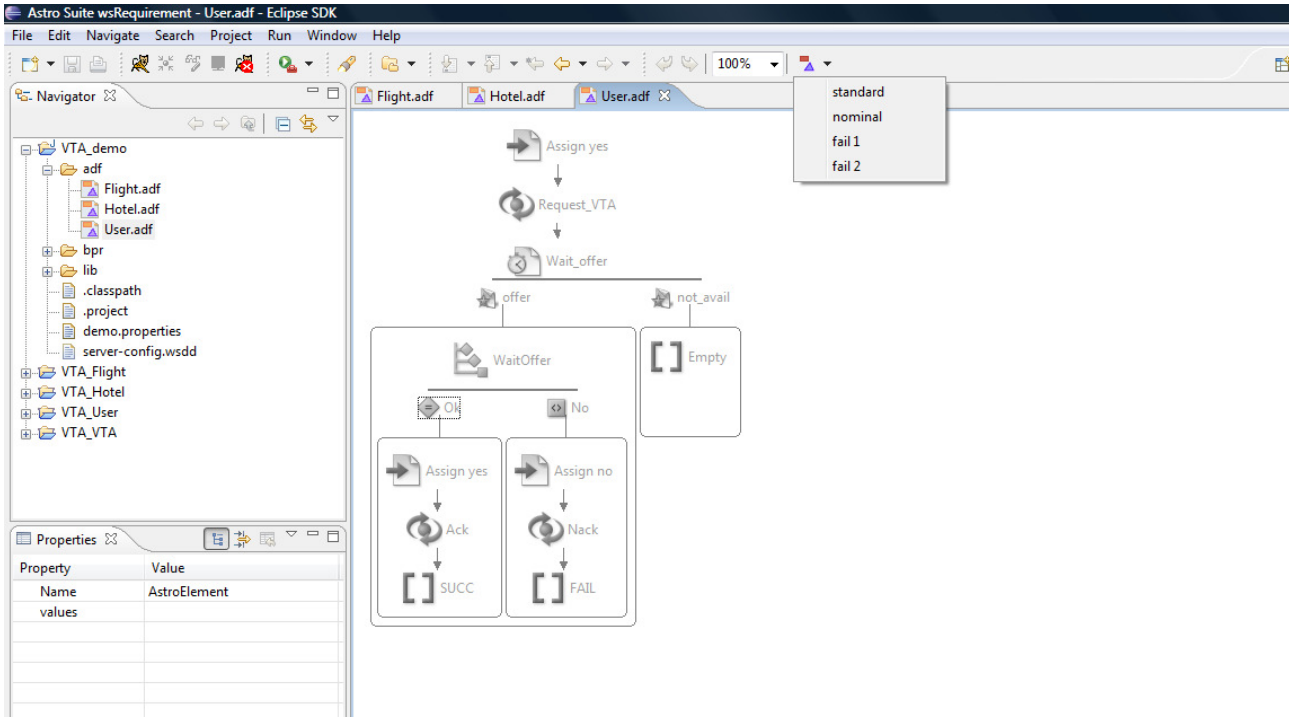


Figura 30: *File .adf*

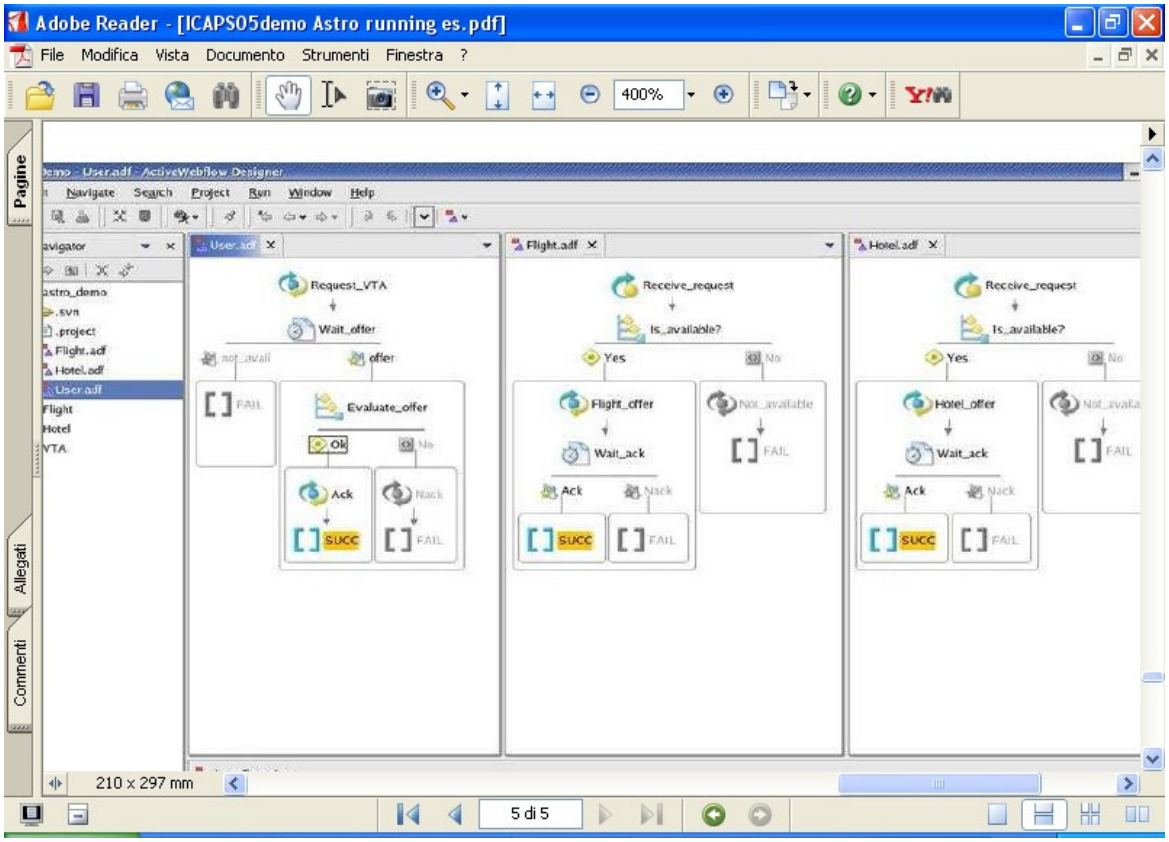


Figura 31: *Process execution simulation*

Attualmente, però, questa parte è ancora in fase di ingegnerizzazione.

Capitolo 5 – Provare a realizzare una demo

Dopo aver analizzato la struttura della demo VTA, ho provato a creare una propria demo che ho chiamato CD-Mania. Per darle un significato reale possiamo immaginarla una vendita di CD musicali, in cui l'utente richiede il CD inserendo il titolo oppure l'autore e inserisce il denaro. Se il CD è disponibile e l'utente ha inserito una moneta da 20 o da 10, allora gli viene fornito il CD, altrimenti gli viene comunicato un not_available. L'utente ha comunque la possibilità di accettare o rifiutare il CD. Uno scenario di funzionamento è mostrato nella seguente figura:

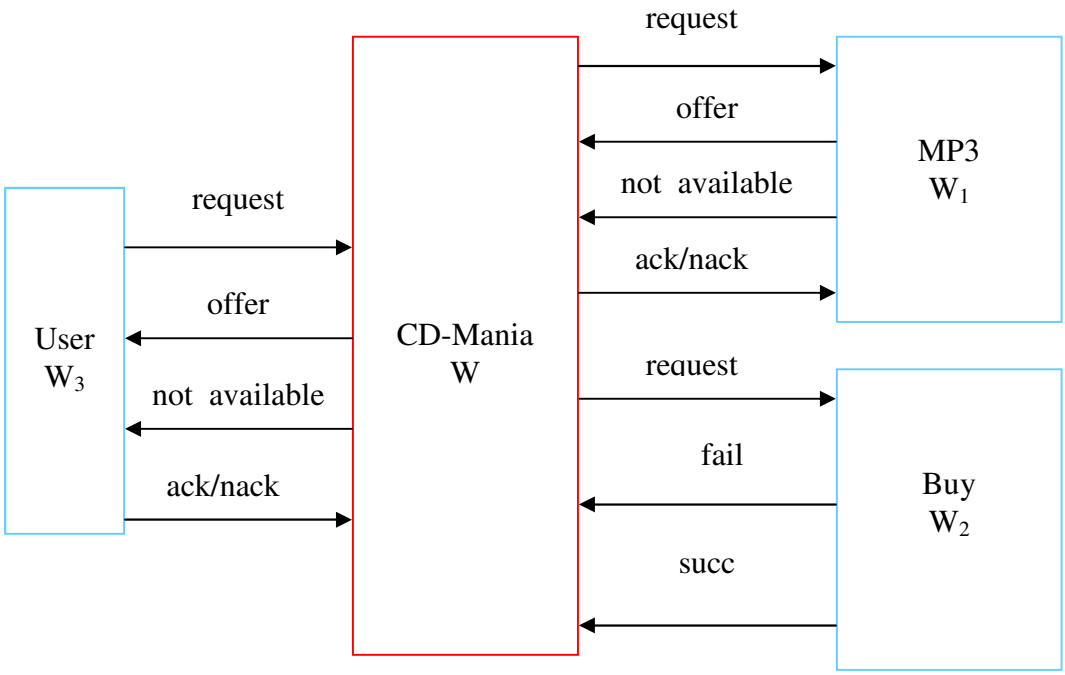


Figura 32: *Scenario di funzionamento della demo CD-Mania*

5.1 Dagli STS agli abstract BPEL

Il primo passo per la realizzazione della demo, è stato partire dalla rappresentazione in STS dei servizi W_1, W_2 e W_3 e tradurli in file BPEL astratti. Ho scelto tre servizi e li ho chiamati Buy, MP3 e User. Di seguito vengono forniti la struttura degli STS e dei corrispondenti file BPEL.

5.1.1 Buy component service

L'STS del Buy component service è il seguente:

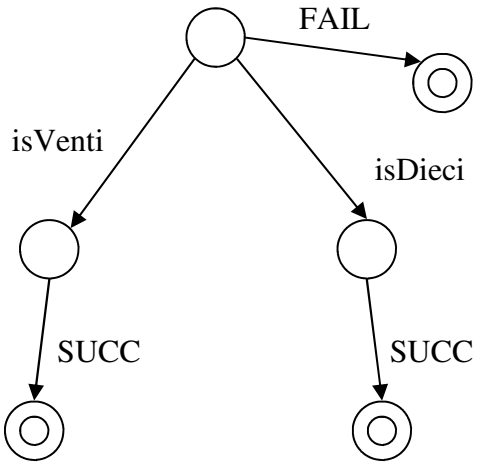


Figura 33: *STS del Buy component service*

Si parte da uno stato iniziale in cui si riceve la richiesta; viene verificato se la moneta inserita è da 20 o da 10, altrimenti si termina con un insuccesso e si passa in uno stato finale.

Entrando più nel dettaglio possiamo dire che nello stato iniziale si riceve una richiesta, quindi la prima attività da inserire nel file BPEL è *Receive*. Da questa partono le tre transizioni e quindi è necessario inserire come successiva attività lo *Switch* con due *Case condition* corrispondenti alle transizioni "isVenti" e "isDieci" e un *otherwise* che porta in uno stato finale di insuccesso rappresentato nel file BPEL da un'attività *Empty*. Le transizioni "isVenti" e "isDieci" portano in due stati in cui vengono invocate le t-transitions del Web service, quindi è necessario assegnare i valori delle variabili contenute nel messaggio ricevuto alle variabili interne del Web service, cioè si inserisce l'attività *Assign*. Successivamente l'invocazione del web service avviene tramite l'attività *Invoke*. In entrambi i case si passa in uno stato finale di successo, rappresentato sempre da una attività *Empty*. Quindi il file .bpel risultante è il seguente:

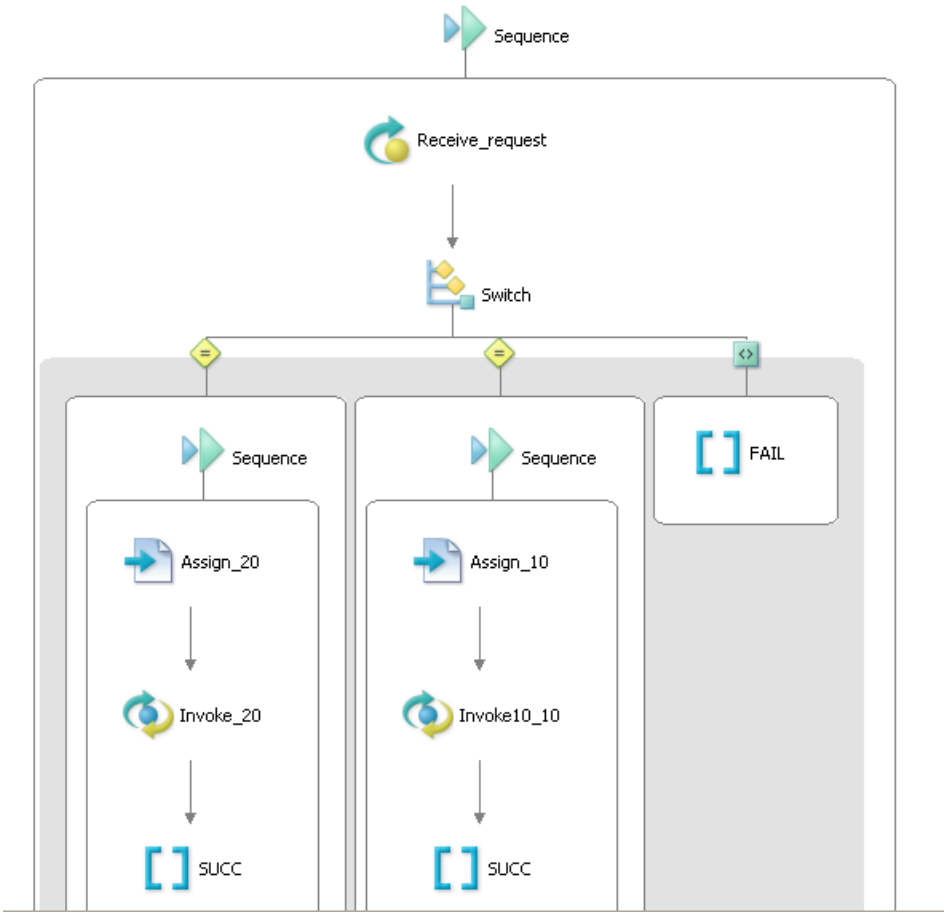


Figura 34: *File Buy_ABS.bpel*

5.1.2 MP3 component service

L'STS del MP3 component service è il seguente:

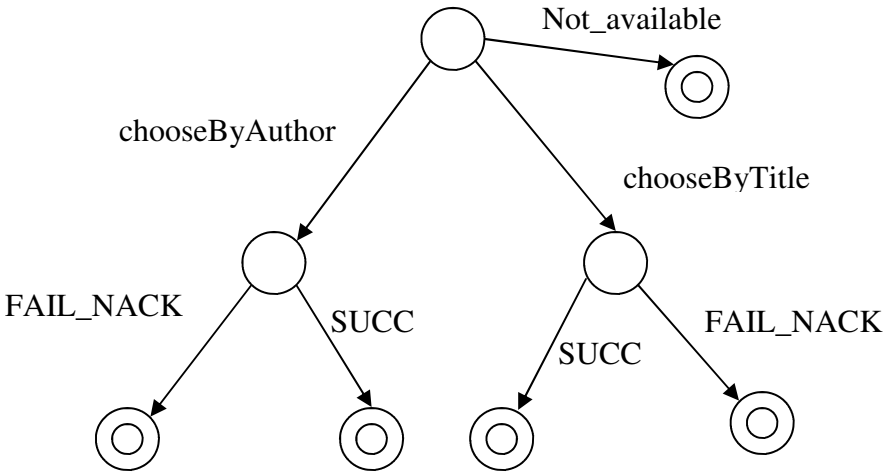


Figura 35: *STS del MP3 component service*

Anche in questo caso si parte da uno stato iniziale in cui si riceve la richiesta; viene verificato se la stringa rappresentante il CD corrisponde al nome dell'autore o al titolo del CD stesso, altrimenti si restituisce un not_available e si termina (si passa quindi in uno stato finale). Le due transizioni "chooseByAuthor" e "chooseByTitle" portano in due stati distinti in cui vengono invocate le t-transitions del Web-service per effettuare la ricerca del CD rispettivamente per autore e per titolo. Si presenta l'offerta (cioè il nome del CD) e si rimane in attesa di una risposta: se lo User accetta allora si termina con successo (si passa nello stato finale tramite SUCC), altrimenti si termina con un insuccesso (si passa nello stato finale tramite FAIL_NACK).

Entrando più nel dettaglio possiamo dire che nello stato iniziale si riceve una richiesta, quindi la prima attività da inserire nel file BPEL è *Receive*. Da questa partono le tre transizioni e quindi è necessario inserire come successiva attività lo *Switch* con due *Case condition* corrispondenti alle transizioni "chooseByAuthor" e "chooseByTitle" e un *otherwise* che porta in uno stato finale di insuccesso rappresentato nel file BPEL da un'attività *Empty*. Come detto prima le transizioni "chooseByAuthor" e "chooseByTitle" portano in due stati in cui vengono invocate le t-transitions del Web service, quindi è necessario assegnare i valori delle variabili contenute nel messaggio ricevuto alle variabili interne del Web service, cioè si inserisce l'attività *Assign*. Successivamente l'invocazione del web service avviene tramite l'attività *Invoke*. In entrambi i case si rimane in attesa di un ack/nack dello User e quindi viene inserita l'attività *Pick* con due attività *onMessage*, cioè si rimane in attesa di un messaggio, che potrebbe essere un nack e quindi si termina con un'attività *Empty* FAIL_NACK, oppure un messaggio ack e quindi si termina con un'attività *Empty* SUCC. Il file .bpel risultante è il seguente:

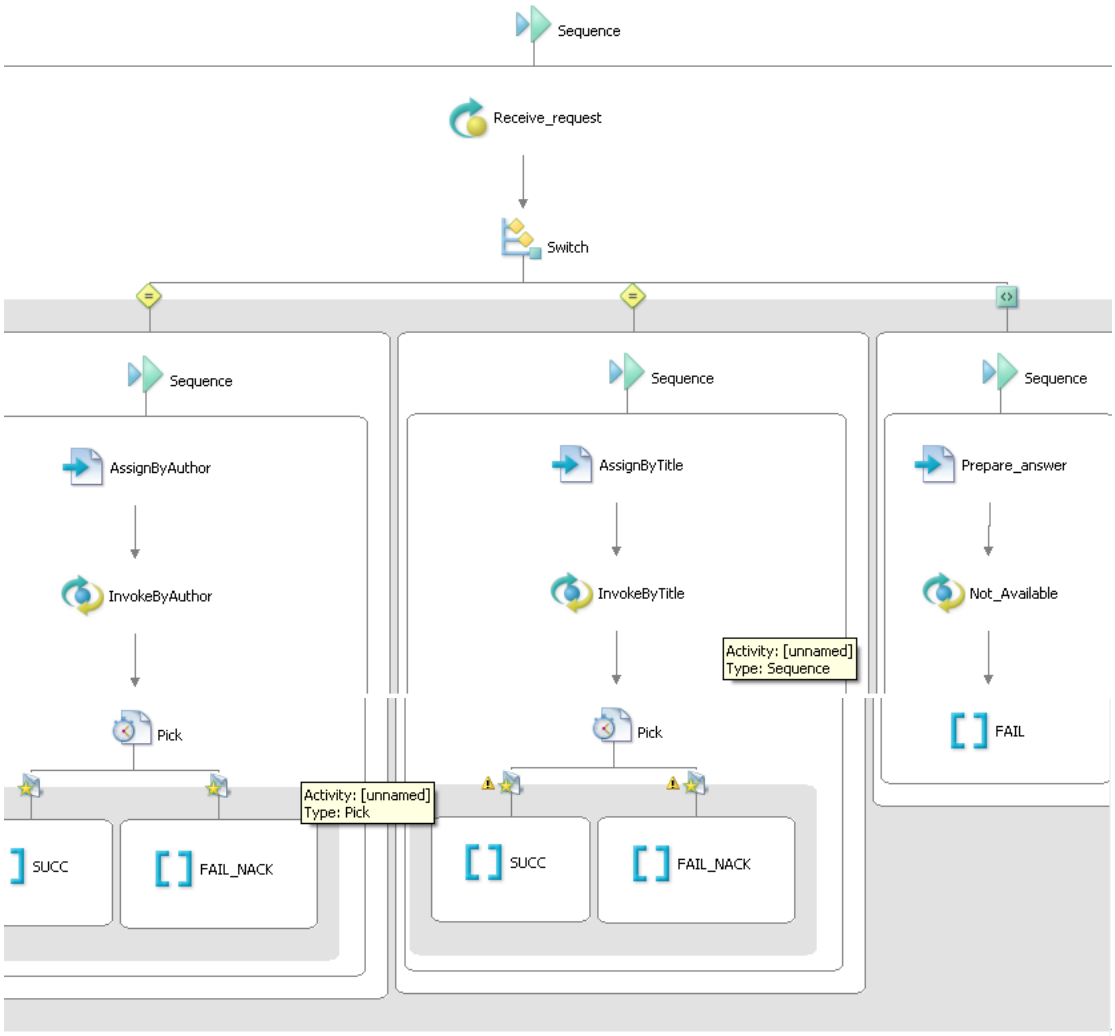


Figura 36: *File MP3_ABS.bpel*

5.1.3 User component service

Il servizio User è analogo a quello della demo VTA, cambia solo il contenuto della richiesta, in cui stavolta abbiamo la specifica del CD (tramite autore o titolo) e della moneta. Omettiamo quindi sia STS che file .bpel.

5.2 Creazione del file .chor

Una volta creati i file abstract .bpel e i rispettivi .wsdl, ci si posiziona in eclipse e si caricano i progetti nel workspace tramite il task File→Import. A questo punto i progetti sono visibili nella view Navigator a sinistra.

Tramite il Wizard per la creazione dei file .chor, ho creato il file CD-Mania_DN.chor. Di seguito vengono mostrati i 5 passi per la creazione del suddetto file:

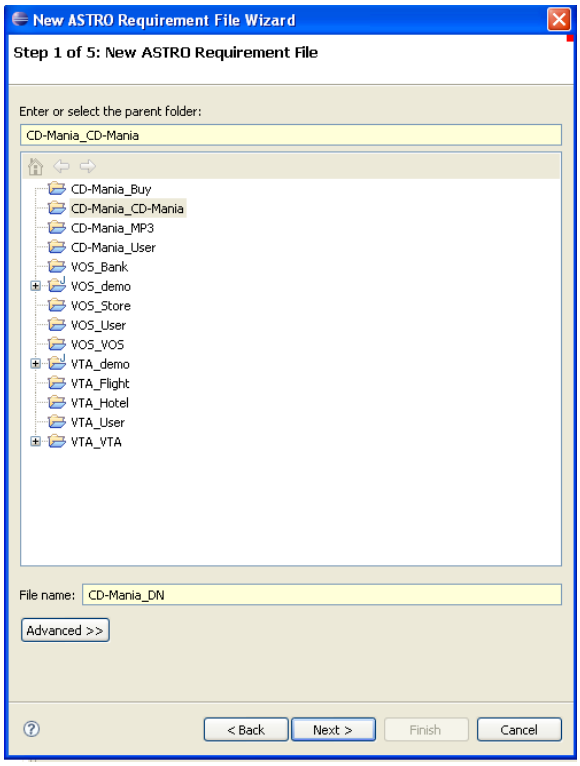


Figura 37: *Step 1*

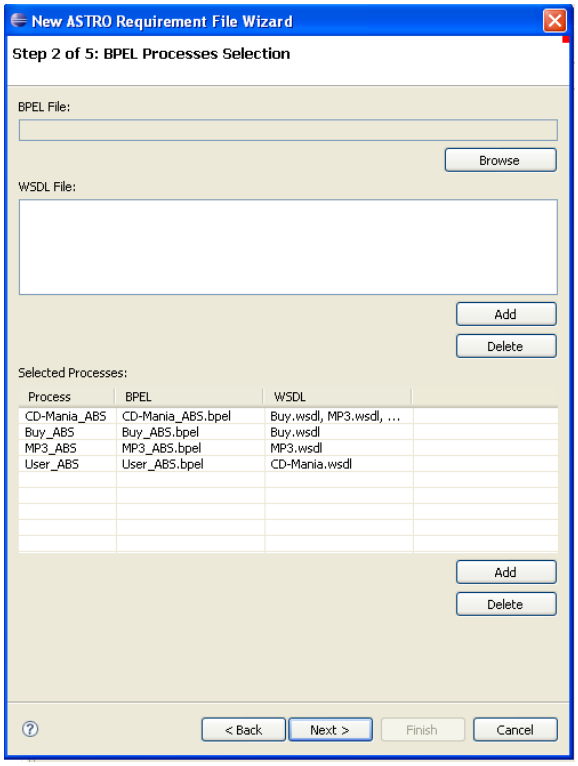


Figura 38: *Step 2*

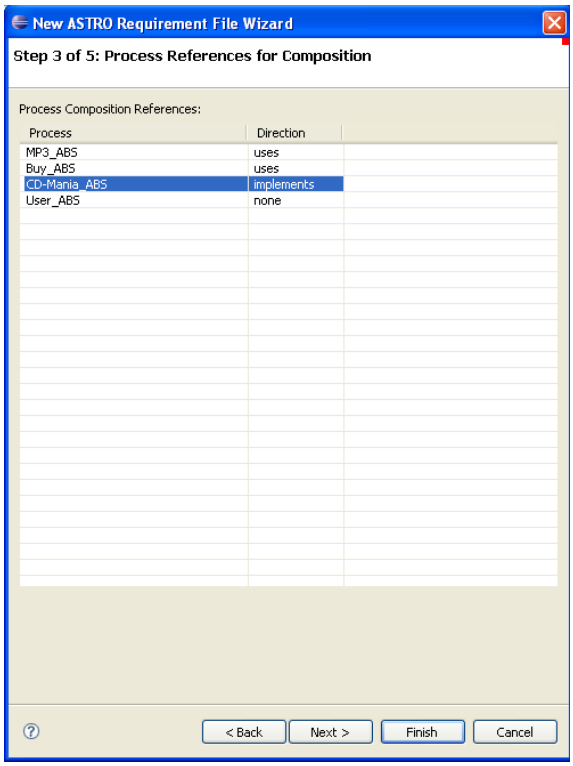


Figura 39: *Step 3*

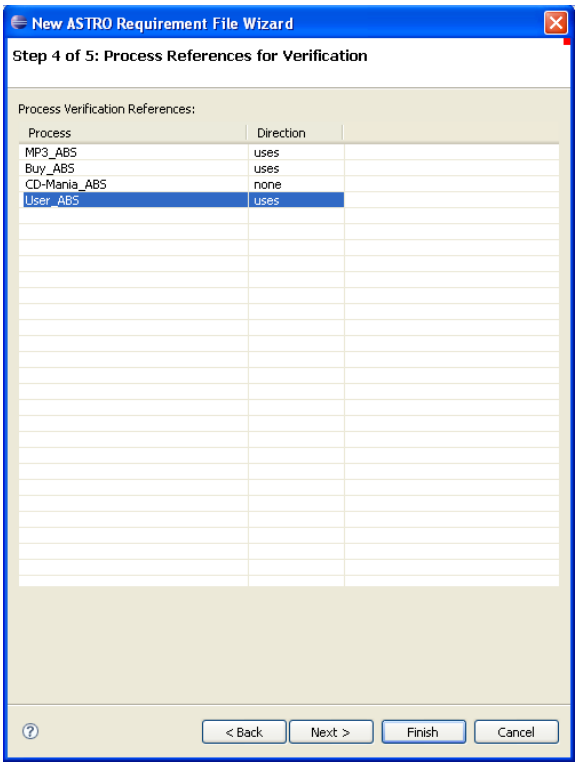


Figura 40: *Step 4*

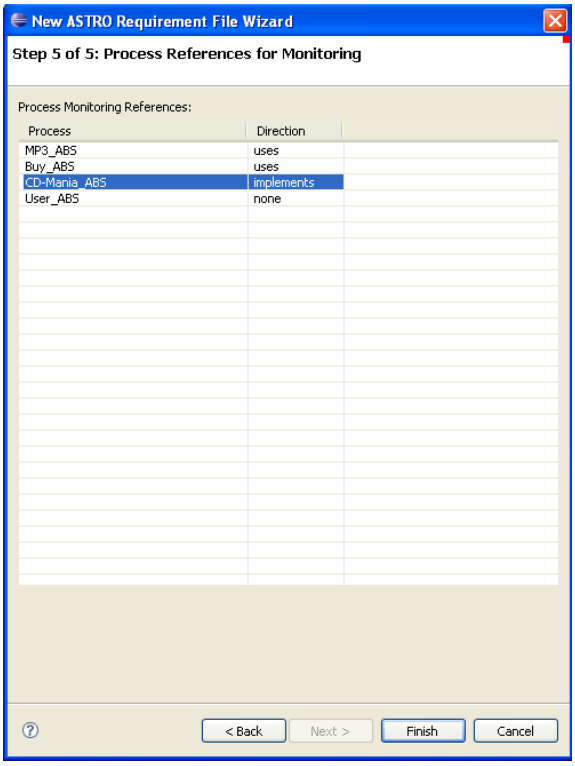


Figura 41: *Step 5*

Nella seguente figura viene mostrata la view Process definition del file CD-Mania_DN.chor, in cui vengono mostrati i processi.

Figura 42: *View Process definition del file CD-Mania_DN.chor*

Nella seguente figura viene mostrata la view Composition ControlFlow del file CD-Mania_DN.chor, in cui sono specificati i main e recovery goal:

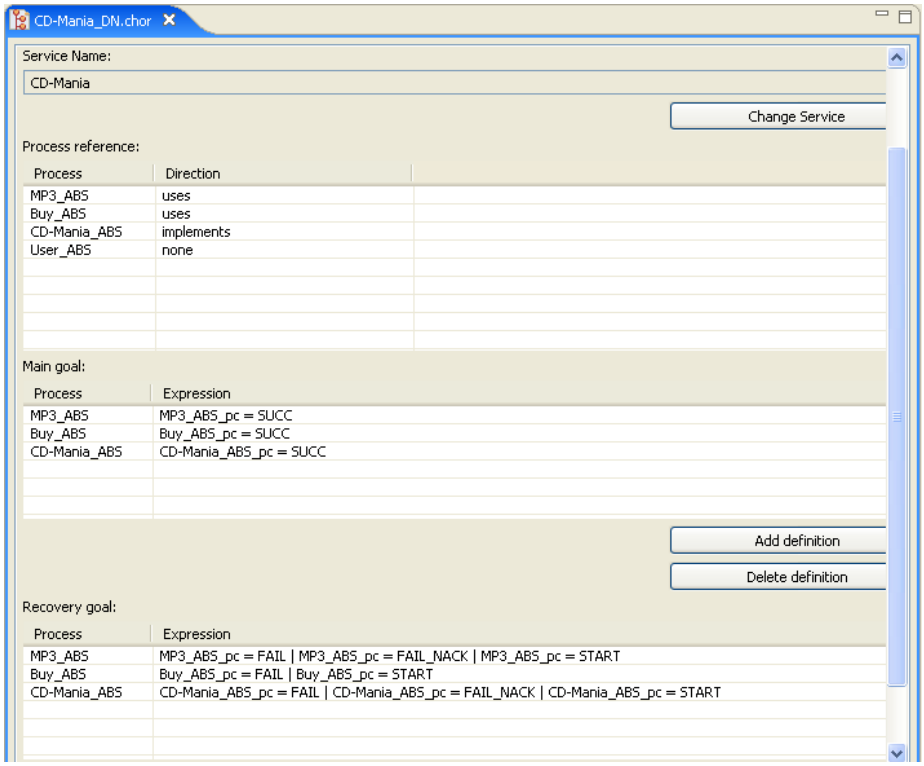


Figura 43: *View Composition ControlFlow del file CD-Mania_DN.chor*

5.3 Perché non funzionante

Nel creare i file .bpel ho incontrato molti problemi, dovuti alle limitazioni sui tipi di variabili e di costrutti supportati dal traduttore Astro.

Per esempio avevo inserito in un file BPEL un link e questo automaticamente mi inseriva anche un'attività *Flow* che è tra quelle non supportate dal tool; però dal messaggio di errore avevo capito che dipendesse dall'aver specificato in maniera errata il dataflow, perché io non sapevo che con il link si fosse aggiunta anche l'attività Flow (in quanto i file .bpel vengono creati tramite palette, non si scrive codice vero e proprio), solo dopo aver ispezionato in maniera più approfondita il codice xml del file, sono riuscita a capire e correggere l'errore.

Poi avevo creato dei messaggi con dei campi di tipo intero ed eseguivo operazioni matematiche; invece tutto ciò non è supportato e quindi ho dovuto modificare i file .bpel (il che richiede molto tempo).

Poi avevo utilizzato un tipo di operazione di assegnamento per la variabile "offer" sintatticamente non corretta e solo grazie all'aiuto del team support di ASTRO sono riuscita a risolvere questo

problema, perché la composizione restituiva un messaggio di errore che era fuorviante e quindi non si riusciva a capire il perché non venisse accettato il messaggio "offer", nonostante fosse stato correttamente dichiarato.
Un altro problema che ho dovuto risolvere è stato quello di aver definito dei recovery goal troppo vincolanti e questo impediva di trovare un piano.
L'ultimo problema incontrato è stato relativo alla semantica dei file .bpel, ma questo non sono riuscita tuttora a risolvere, perciò la demo non è funzionante.
Infatti avere un errore di semantica significa che il processo di composizione avviene, ma non produce l'output atteso, cioè il file concrete .bpel viene creato, ma al suo interno ci sono solo i tag degli elementi che dovrebbero essere inseriti ma non la lista degli elementi definiti; per esempio i tag delle variabili non contenevano la lista delle variabili definite, di conseguenza il processo non veniva deployed su engine. Nonostante questo, però, viene comunque creato il package del file .bpel, wsdl relativi e files di deployment .xml e .pdd in un file .bpr in Tomcat.

Purtroppo il tempo impiegato per affrontare tutti questi problemi è stato notevole, per via di capire dove era il problema e come risolverlo, e questo ha impedito di continuare a lavorare su questa demo.

Conclusioni

Il tool presenta ancora numerose limitazioni:

- allo stato attuale supporta WS-BPEL 1.1 e la roadmap non prevede attività legate al supporto WS-BPEL 2.0;
- Il tipo "xsd:positiveInteger" non è gestito al momento e deve essere sostituito dal tipo "xsd:string". Ovvero non è possibile ragionare sui numeri al momento ma solo sulle stringhe e su pochi altri tipi come ad esempio i booleani e gli enumerativi;
- Manca di robustezza, infatti quando si provano vari scenari di simulazione dei file .adf, dopo due – tre volte la simulazione rimane bloccata.

Nel corso della realizzazione di questa tesina ho incontrato numerosi problemi, molti dei quali li ho già specificati nell'ultimo paragrafo del quinto capitolo e vengono di seguito riassunti.

Problemi incontrati:

- Installazione del tool → dovuta a baghi sulla definizione delle variabili d'ambiente e sui nomi delle directory. In particolare la variabile path doveva essere impostata aggiungendo la directory \bin al percorso <WSTOOLSET_INSTALL_DIR>\tools\synTools, mentre all'interno della folder <WSTOOLSET_INSTALL_DIR>\tools bisogna aggiungere una copia della folder "wsTranslator-0.14.0" rinominandola "wsTranslator-$WSTRANSLATOR-VERSION", per ovviare a un bago che ancora non è stato eliminato;
- Realizzazione dei file bpel → dovuta alla non conoscenza della versione del linguaggio, dei tipi di variabili e dei costrutti sintattici supportati.
- Problemi relativi alla semantica dei file abstract .bpel della demo che si è provato a realizzare, e attualmente ancora non si è capito il motivo di questi errori.

Bibliografia

- Alonso, Casati, Kuno, Machiraju; 2004; "Web Services - Concepts, Architectures and Applications"
- The ASTRO Project Website; http://www.astroproject.org
- The BPEL4WS Specification, v1.1;
  http://dev2dev.bea.com/technologies/webservices/BPEL4WS.jsp
- The ActiveBPEL Engine, http://www.activebpel.org
- The ActiveBPEL Designer v2.0 User's Guide (http://www.active-endpoints.com)
- Il Progetto ASTRO nella Web Service Composition: analisi e confronto con il Roman Approach – Alessandro Pagliaro
- Composizione automatica di servizi: l'approccio ASTRO e il Roman Model a confronto – Alessandro Dionisi
- M. Trainotti, M. Pistore, G. Calabrese, G. Zacco, G. Lucchese, F. Barbon, P. Bertoli, P. Traverso - ASTRO: Supporting Composition and Execution of Web Services