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

Composizione automatica di servizi: l'approccio ASTRO

Corso di Seminari di Ingegneria del software
Proff. G. De Giacomo, M. Mecella, R. Rosati

Autore
Piacentini Vanda

Anno Accademico 2006/2007

Sommario

- Breve introduzione al problema della composizione automatica di servizi
- Il progetto ASTRO & Web Service Composition problem
- L'ASTRO suite per la composizione automatica
- La demo VTA

Il problema della composizione automatica

Costruire nuovi servizi componendo quelli esistenti

Requisiti per affrontare il problema:

- un linguaggio per la rappresentazione comportamentale dei Component e Target Services
- una metodologia di composizione, fondata su solide basi teoriche
- un ambiente di sviluppo che permetta l'esecuzione automatica di composizioni
- un composition engine per testing, monitoring, verification

Il problema della composizione automatica (cont.)

- Focus sul *comportamento* dei Web services in gioco
- Rappresentazione tramite Transition System (FSM)

- Nodi come "stati stabili" dell'esecuzione
- Archi come transizioni tra stati
- Differenza tra *external actions* e *internal actions* (anche chiamate t-transitions)

L’approccio ASTRO

Il progetto ASTRO

Il Progetto ASTRO è un'iniziativa di ricerca congiunta riguardo l'integrazione di Web services, promossa dall'Università di Trento e l'ITC-IRST

Obiettivi:

- Fornire un framework per la composizione automatica di servizi
- Fornire dei tools per implementare il framework
- Fornire la possibilità di gestire l'intero ciclo di vita delle applicazioni, dalle prime fasi di design fino al monitoraggio e verifica a runtime
- Automatizzare tasks noiosi ed error-prone

La composizione automatica in ASTRO

La composizione viene modellata come un problema di pianificazione

Requirements:

- Component e Target Services come Abstract BPEL Processes
- Business Requirement Goals come EAGLE Formula

Presupposti dell'Approccio:

- Ambiente *asincrono*
- Osservabilità *parziale* dei servizi
- *Extended* Business Goals

La composizione automatica in ASTRO(cont.)

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

Definizione di Astro State-Transition Sytem:
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 spedizione di messaggi);
- R è la relazione di transizione da S x (I U O U {t}) → S;
- L è funzione di labeling, e associa ad ugni stato un set di proprietà soddisfatte dallo stato. Formalmente, detto *Prop* l'insieme delle proprietà, L: S → 2^{Prop}

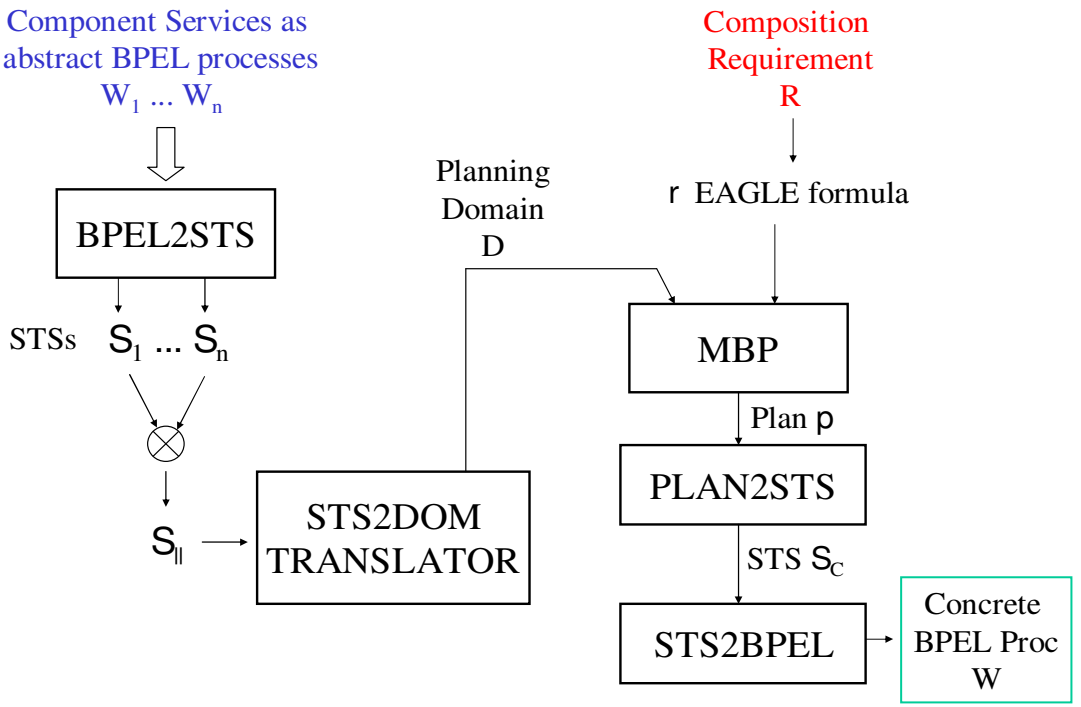
Assunzioni sugli STS:

- Assenza di loops infiniti sulle t-actions

L'interfaccia pubblica di invocazione del servizio viene specificata con WSDL

Una descrizione "comportamentale" viene codificata utilizzando un linguaggio per esprimere macchine a stati finiti: **BPEL4WS**

The Astro Composition Workflow



- Costruzione del prodotto parallelo $S_{||}$ a partire dagli STS dei servizi esistenti $S_1,\ldots,S_n$
- Creazione del dominio D, relativo a $S_{||}$
- A partire dallo stato iniziale *s,* il dominio D e i requisiti r, si genera il piano p
- Dal piano si ricava il controller S_c per $S_{||}$.
- Tramite il modulo STS2BPEL il controller S_c viene trasformato in file concrete .bpel

Il processo di composizione
Basi teoriche(1)

Step I – Prodotto parallelo:

I Component Services e il Target Service vengono trasformati in STSs tramite l'opportuno modulo BPEL2STS. Quindi viene calcolato il loro Prodotto Parallelo $S_{||}$.

Definizione: Prodotto parallelo tra due STS S_1 e S_2

Siano $s_1 = < S_1, S_1^0, I_1, O_1, R_1, L_1 >$ e $s_2 = < S_2, S_2^0, I_2, O_2, R_2, L_2 >$ due STS tali che $(I_1 \cup O_1) \cap (I_2 \cup O_2) = \emptyset$.

Il Prodotto Parallelo $s_1 || s_2$ tra s_1 e s_2 è definito come:

$$s_1 || s_2 = < S_1 \times S_2 , S_1^0 \times S_2^0 , I_1 \cup I_2 , O_1 \cup O_2 , R_1 || R_2 , L_1 || L_2 >$$

dove:

$< (s_1, s_2), a, (s_1', s_2) >$ appartiene a $(R_1 || R_2)$ se $< s_1, a, s_1' >$ appartiene a R_1;

$< (s_1, s_2), a, (s_1, s_2') >$ appartiene a $(R_1 || R_2)$ se $< s_2, a, s_2' >$ appartiene a R_2;

e inoltre $(L_1 || L_2) (s_1, s_2) = L_1 (s_1) \cup L_2 (s_2)$.

Rappresenta tutte le possibili evoluzioni concorrenti dei servizi, senza nessun controllo o interazione con il servizio che sarà generato.

Il processo di composizione
Basi teoriche(2)

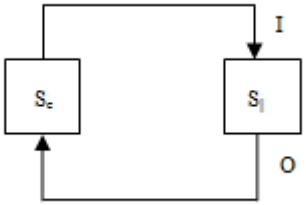
Step II – Sistema controllato:

La realizzazione del Composite Service a partire dai nostri requirements si concretizza nel trovare un STS S_C, che soddisfa particolari proprietà e che deve essere un CONTROLLER per $S_{||}$ (un STS che ne controlla l'esecuzione).

Step III – Deadlock-free controller:

Ogni volta che S_c effettua una transizione di output $S_{||}$ deve essere in grado di accettarla, e viceversa.

In caso contrario si incorre in una situazione di deadlock.



Il processo di composizione
Basi teoriche(3)

Step IV – Belief-Level:

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) → *Piena osservabilità*

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 |= r , dove s_B è il Belief-Level System dell'STS s_C |> (s_1 || ...|| s_n).

L'ASTRO Toolset

Struttura dell'ASTRO Toolset

Versione 3.4

È formato da numerosi componenti software, alcuni sviluppati interamente dal team Astro, altri sono programmi di terze parti con le quali il toolset interagisce:

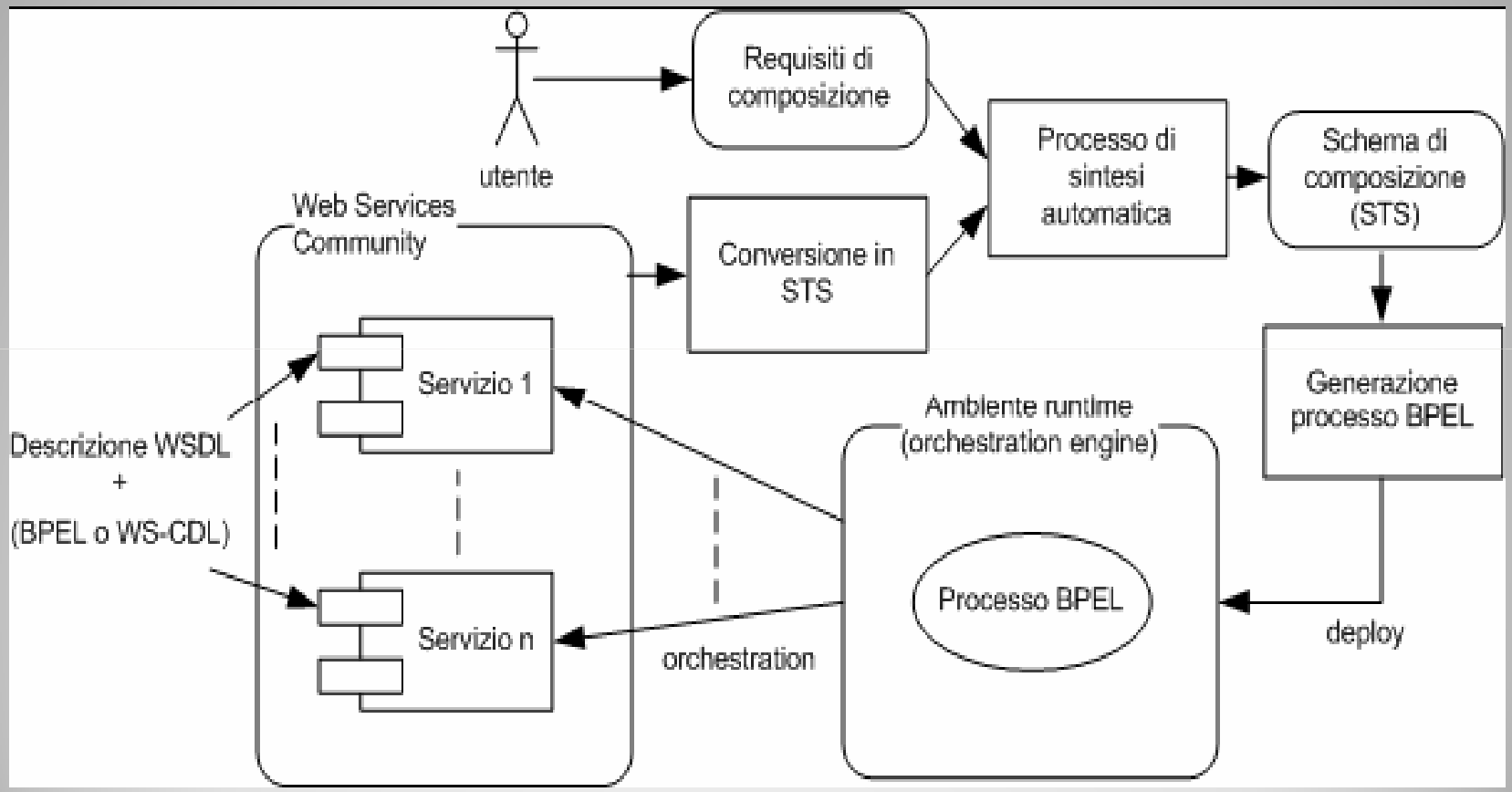
- Java 1.5.x
- Eclipse IDE 3.2.2
- Tomcat server 5.5.x
- ActiveBPEL Engine 2.0
- Graphical Editing Framework & Graphical Modeling Framework Eclipse plugins
- ActiveBPEL Designer 2.0
- Astro wsToolset 1.8.0
- Astro wsMonitor 1.6.0
- Astro wsRequirement 0.2.0 Eclipse plugin
- Astro wsChainManager 2.4.0 Eclipse plugin
- Astro wsAnimator 0.0.7 Eclipse plugin
- Astro wsUseCases 1.0.0 Eclipse plugin

Struttura dell'ASTRO Toolset (cont.)

Il **wsToolset** contiene:

- il programma wsTranslator 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, generando quindi il dominio D
- il package synTools contenente 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 ricava il piano p che soddisfa il goal r su dominio D e restituisce il file concrete BPEL eseguibile
- il programma NuSMV essenziale per eseguire operazioni di model checking su STSs

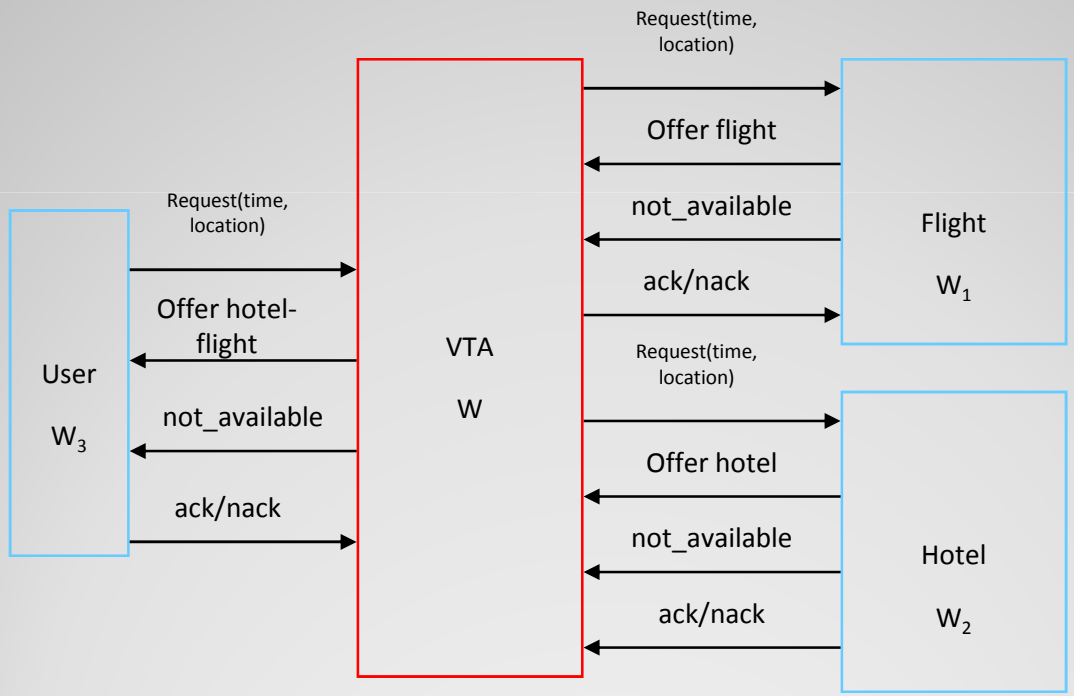
Il processo di composizione automatica



La demo VTA

Struttura della demo

VTA è una delle due demo fornite dal progetto ASTRO. Rappresenta un'agenzia di viaggi virtuale, tramite la quale si possono prenotare il volo e l'hotel, fornendo il periodo e il luogo. È costituita da tre servizi: User, Flight e Hotel.



Il processo di composizione automatica

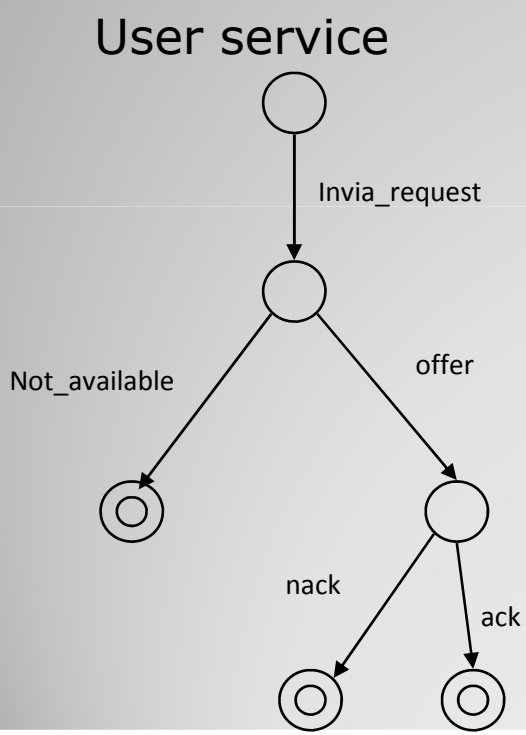
File abstract BPEL

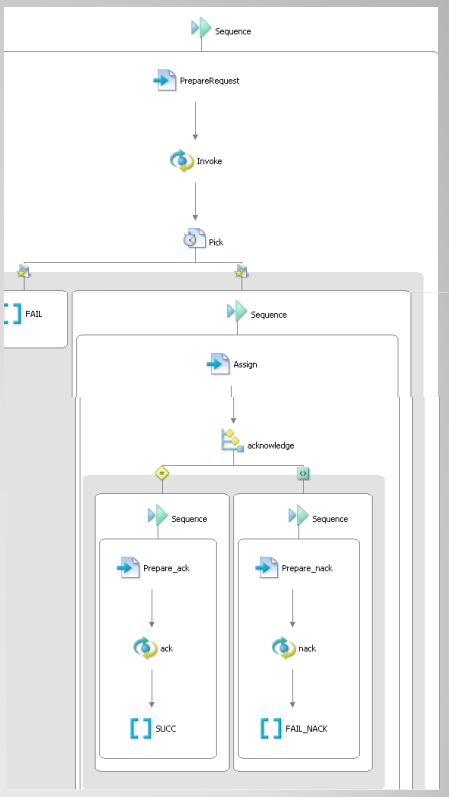
Tramite l'ActiveBpel Designer vengono creati gli Abstract BPEL processes e le interfacce WSDL, che rappresentano i Component Web services

Si parte con l'analisi dell'STS del web service e si traduce in file bpel.

VTA
Servizio User

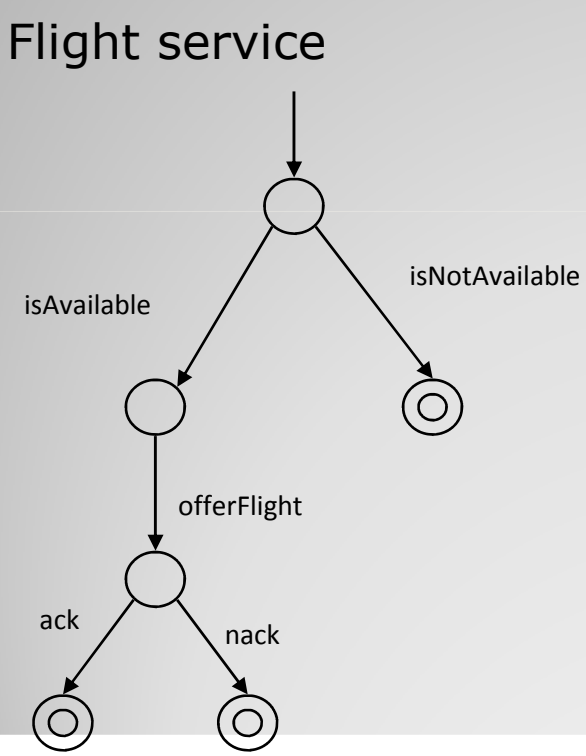
Corrispondenza tra STS e file BPEL

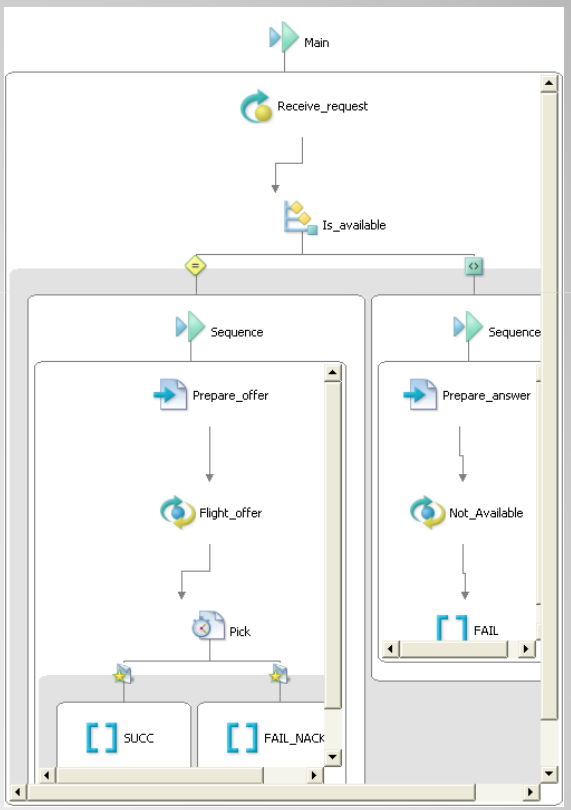




VTA
Servizio Flight

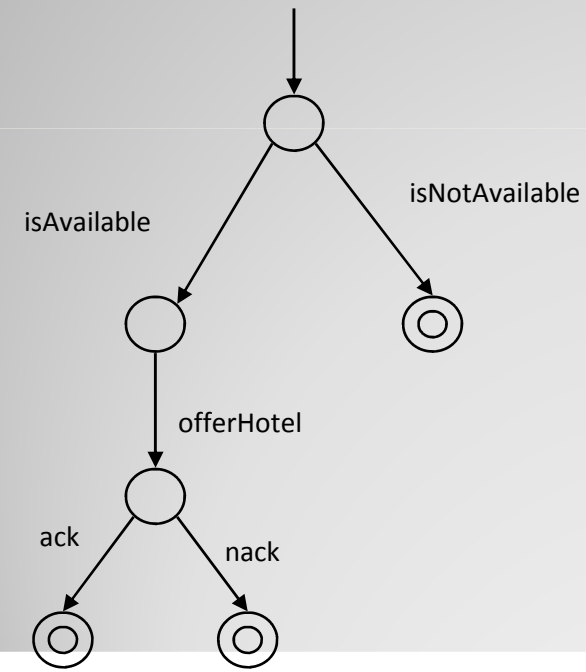
Corrispondenza tra STS e file BPEL

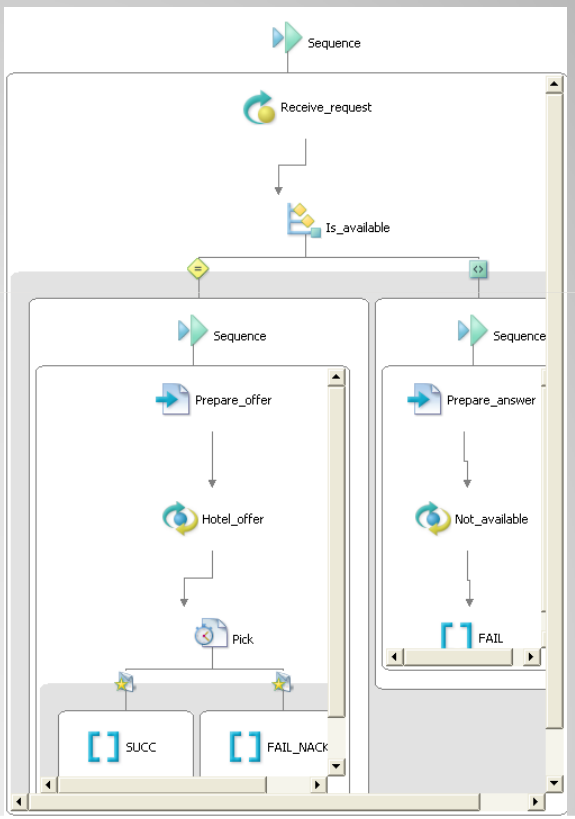




VTA
Servizio Hotel

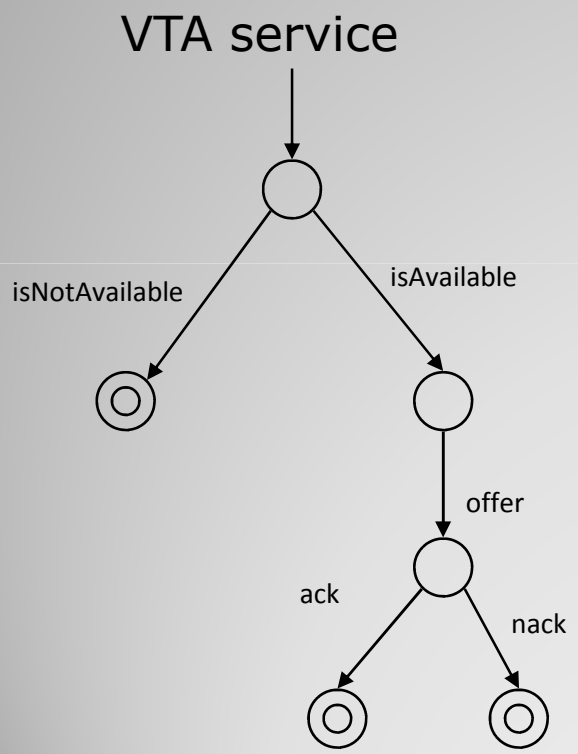
Corrispondenza tra STS e file BPEL

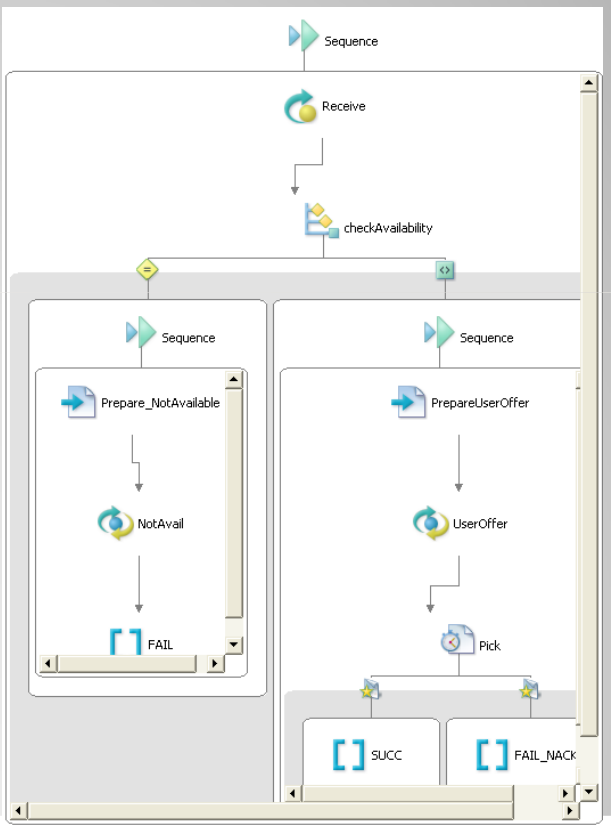




VTA
Servizio VTA

Corrispondenza tra STS e file BPEL





Il processo di composizione automatica

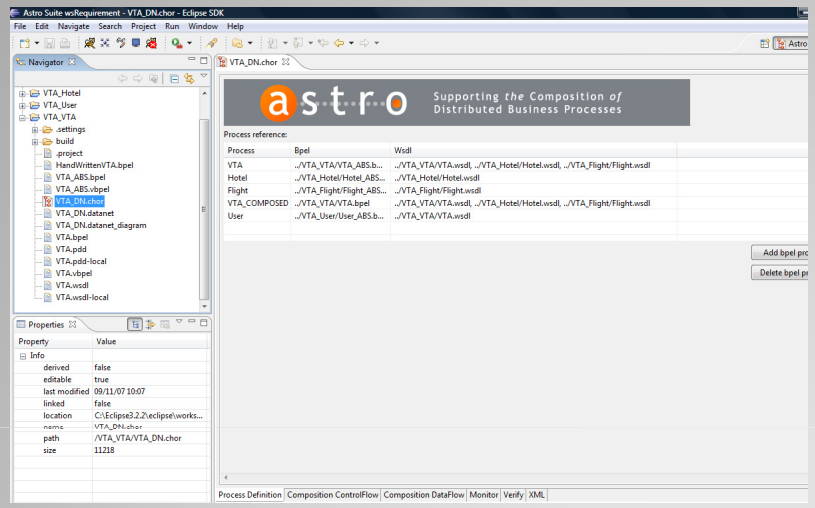
File VTA_DN.chor

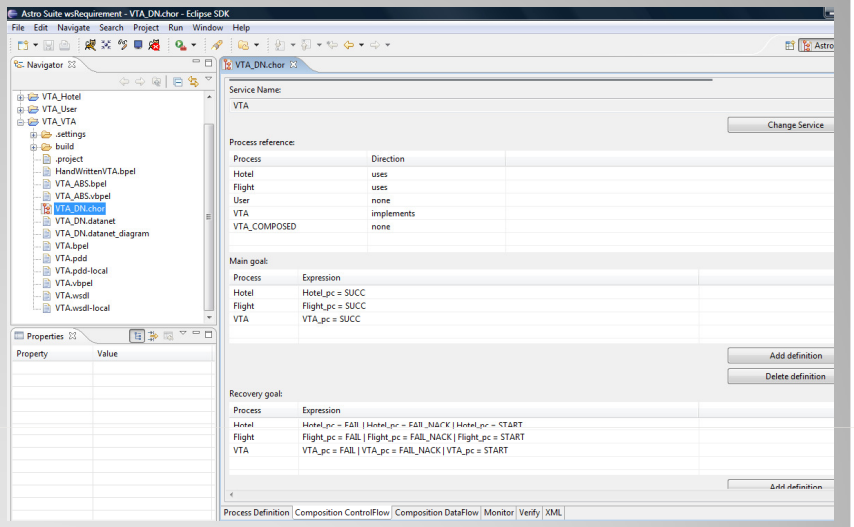
La plugin wsRequirement realizzata dal team Astro permette, tramite un Wizard, la creazione di un file XML "di coreografia" (con estensione .chor) che racchiude l'intero problema (component service, target service, requirements, proprietà da essere monitorate e verificate) ed è ispezionabile ed editabile tramite un'efficace GUI.

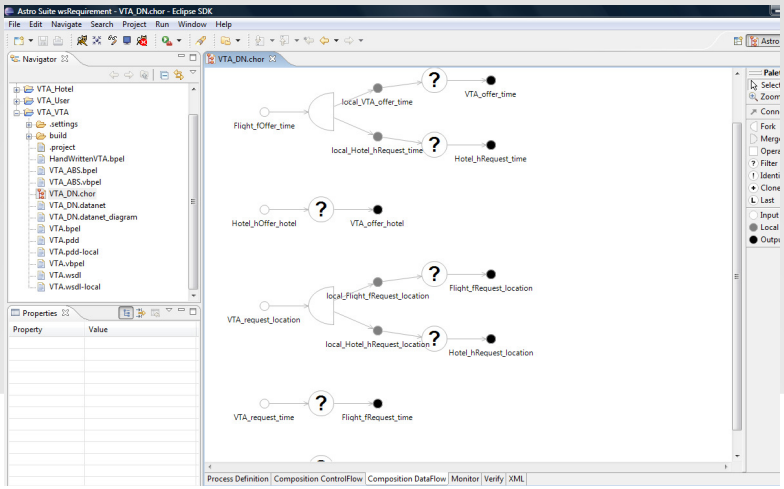
Step per la creazione:

- 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.

File VTA_DN.chor







Il processo di composizione automatica

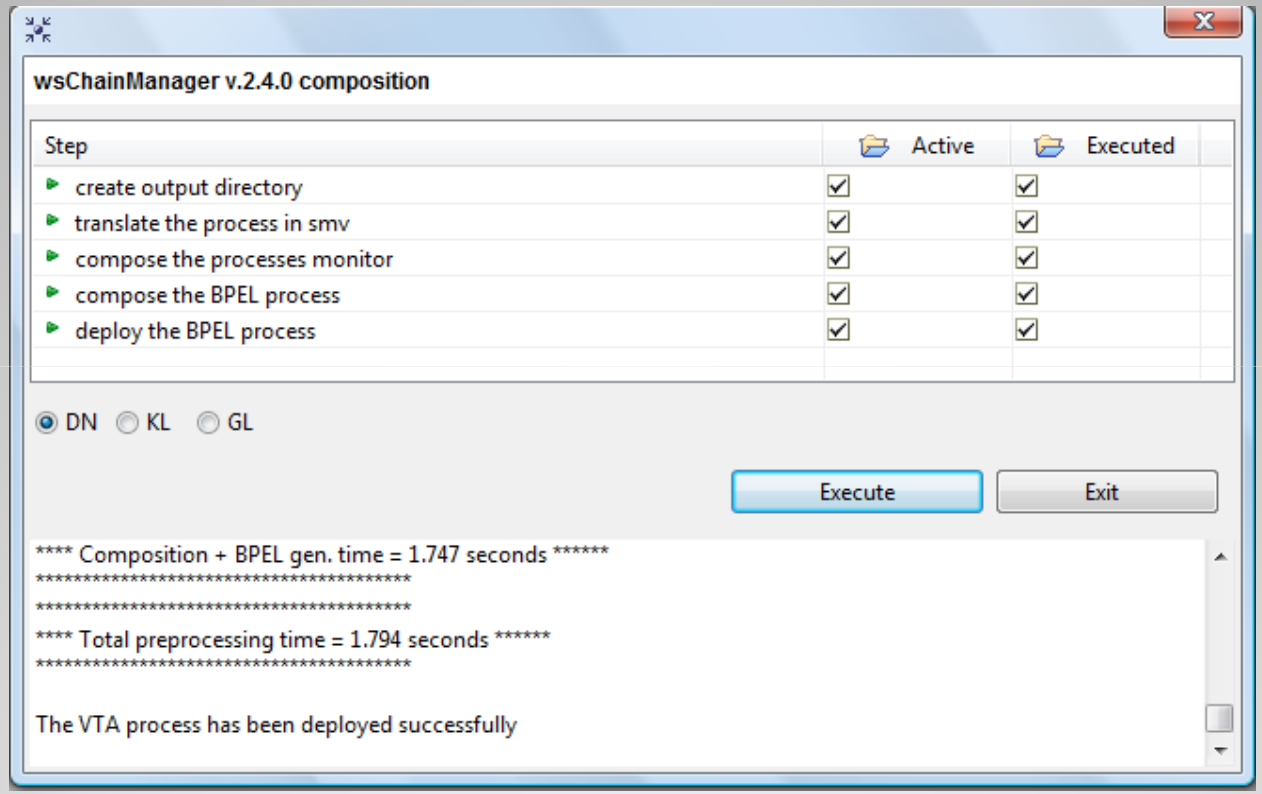
Process composition

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 .chor, 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 .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.

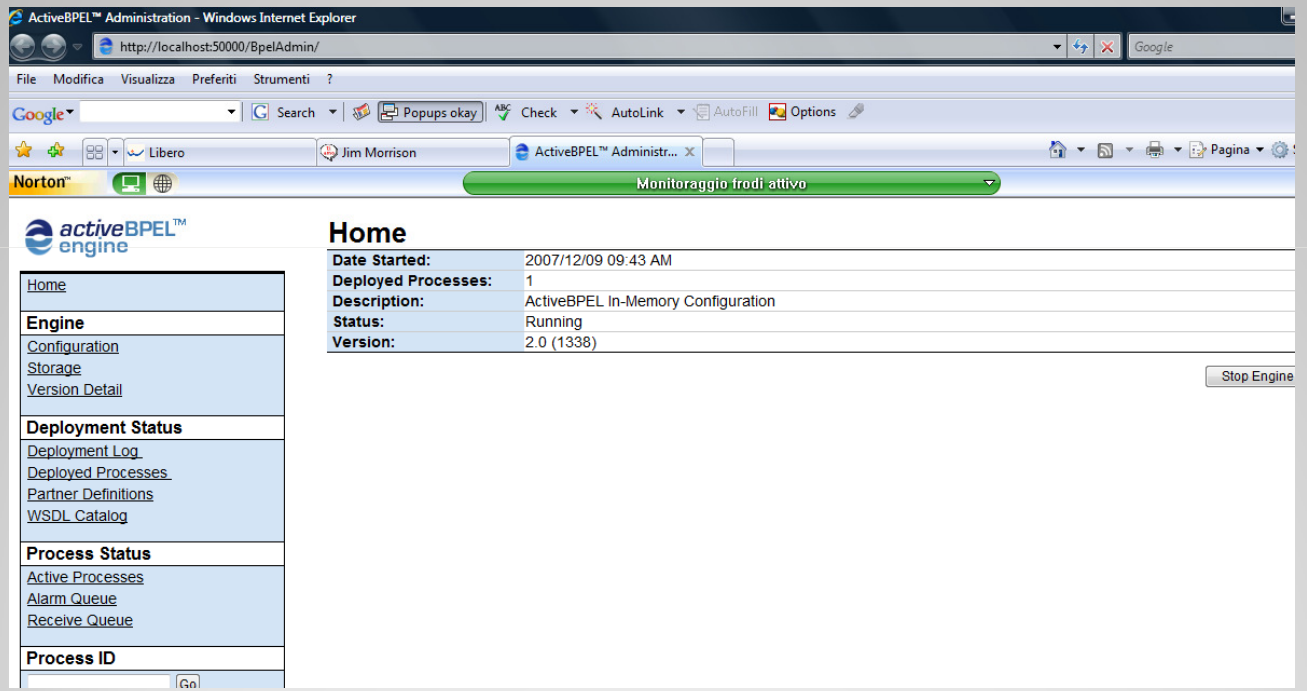
Ciò che il wsChainManager fa è costruire il dominio D dal prodotto parallelo e risolvere il problema di planning via Model Checking, utilizzando i programmi NuSMV, wsTranslator, wSynth.

Il processo di composizione automatica
Process composition - wsChainManager



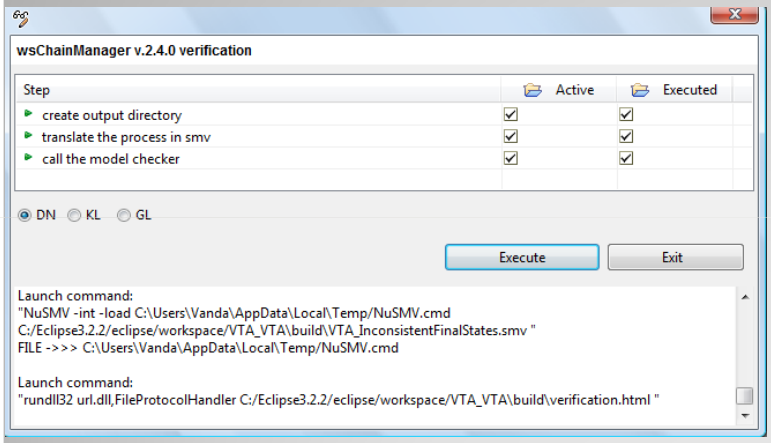
Il processo di composizione automatica
Process composition – activeBpel engine

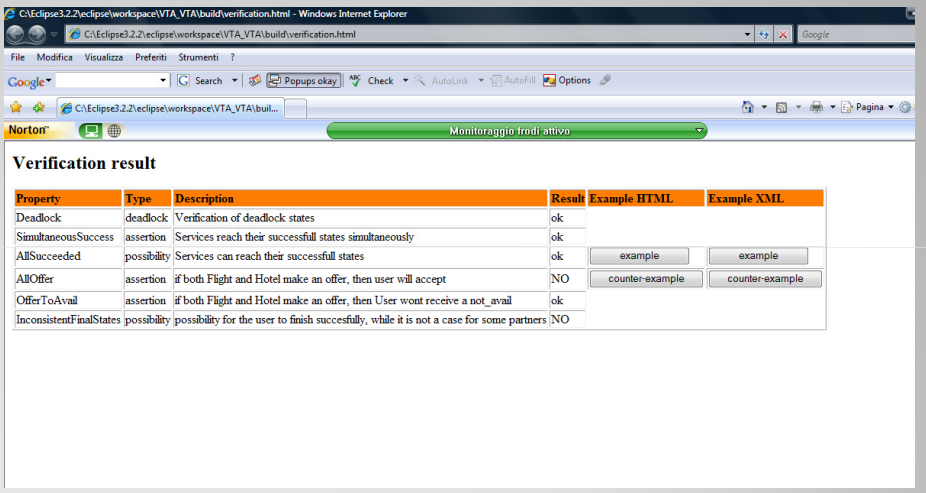
L'output finale della composizione è il file Concrete BPEL automaticamente deployed su ActiveBPEL Engine, e può essere monitorato e testato via browser alla seguente URL: http://localhost:50000/BpelAdmin

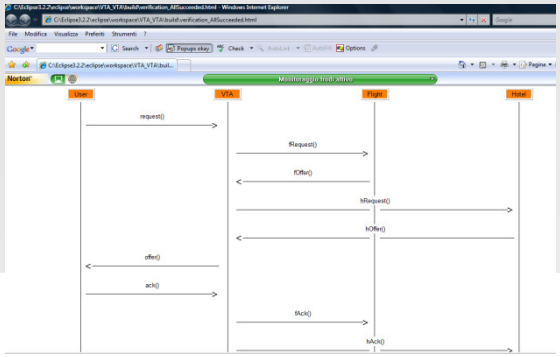


Process Verification

Sempre selezionando il file VTA_DN.chor e cliccando sul bottone  si avvia la process verification, una procedura offline che controlla determinate proprietà e fa rapporto all'utente, anche fornendo contro-esempi stile UML Sequence Diagram.

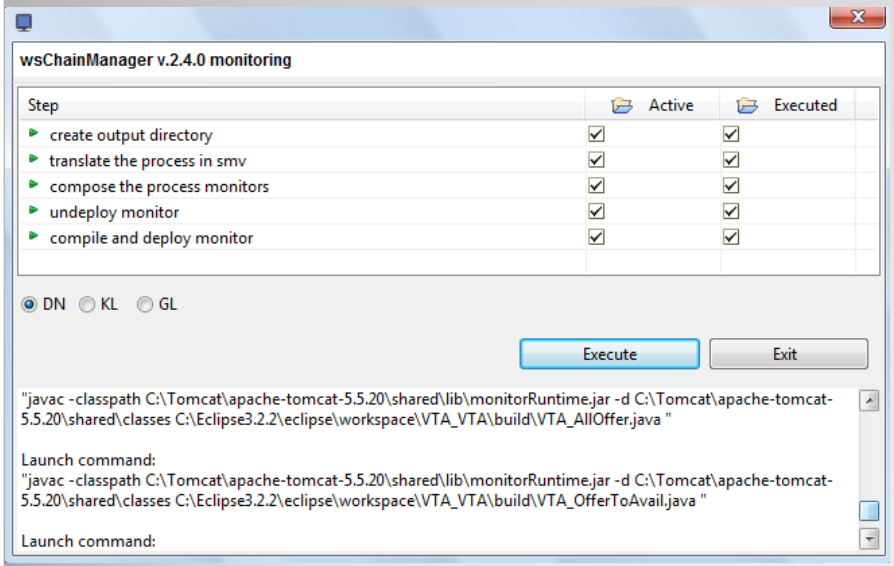


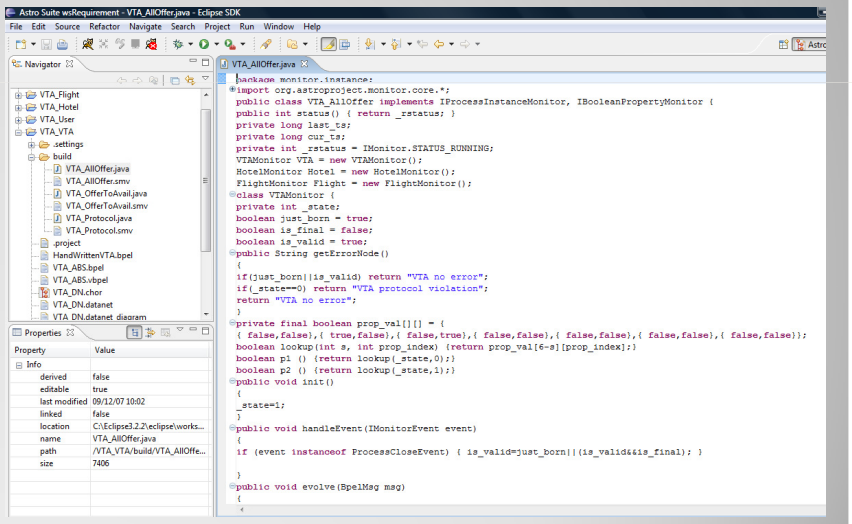




Process Monitoring

Sempre selezionando il file .chor e cliccando sul bottone  si avvia il process monitoring: una procedura online in cui viene eseguito del Java code generato automaticamente per monitorare l'esecuzione del processo che rappresenta il composite service e fornire reports su situazioni di interesse all'utente.

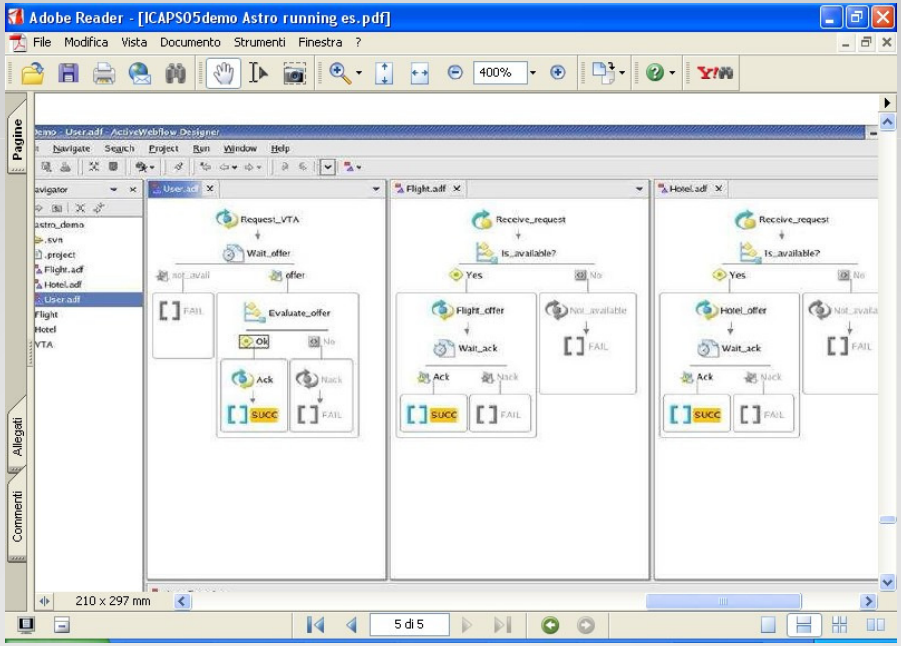




Process execution simulation

Tramite la plugin wsAnimator sviluppata dal team Astro ed utilizzando un formato file particolare .adf, è possibile mandare in esecuzione diversi scenari pre-programmati.

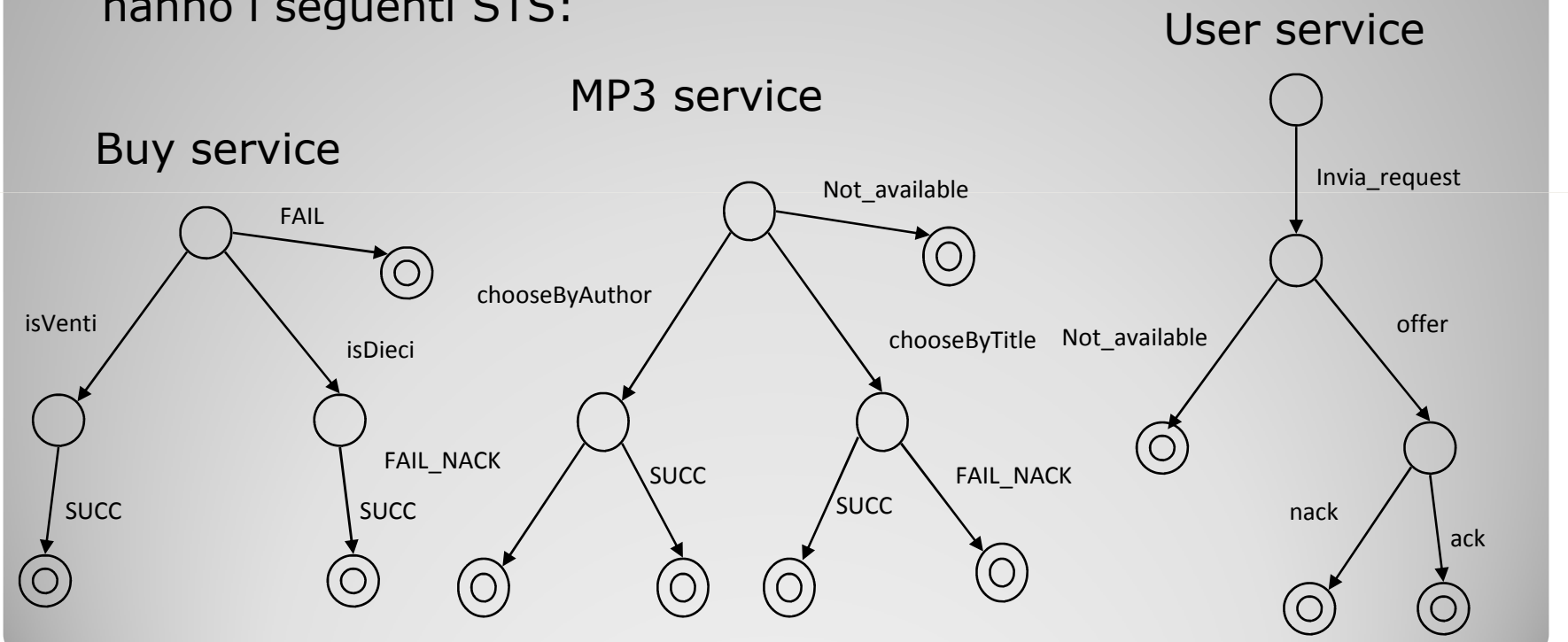
Attualmente, però, questa parte è ancora in fase di ingegnerizzazione.



Provare a realizzare una demo

Demo CD-Mania

Sulla base dell'esempio mostrato, ho provato a realizzare una demo, partendo da tre servizi che ho chiamato Buy, MP3 e User, che hanno i seguenti STS:

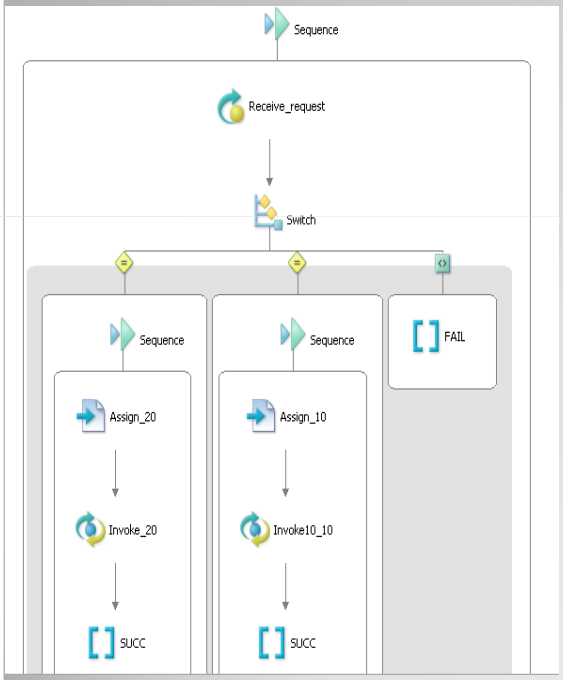


Demo CD-Mania

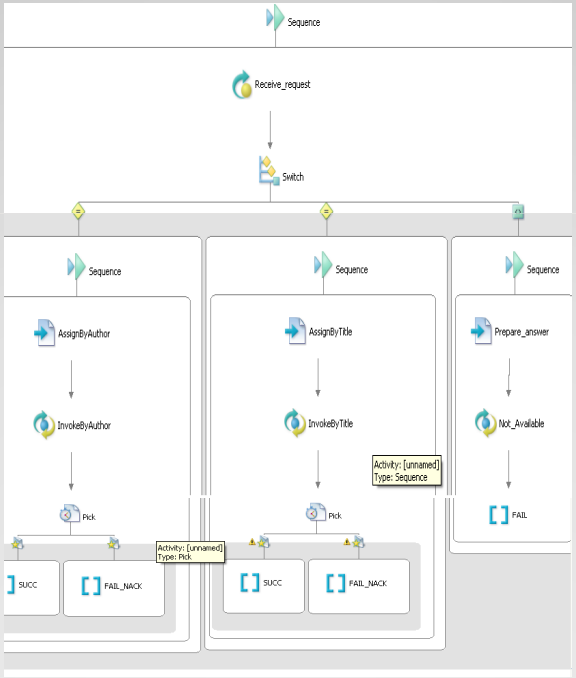
Per darle un significato reale possiamo immaginarla una vendita di CD musicali, in cui l'utente richiede il CD inserendo il titolo oppure l'autore e inserisce il denaro. Se il CD è disponibile e l'utente ha inserito una moneta da 20 o da 10, allora gli viene fornito il CD, altrimenti gli viene comunicato un not_available. L'utente ha comunque la possibilità di accettare o rifiutare il CD.

Demo CD-Mania
File .bpel

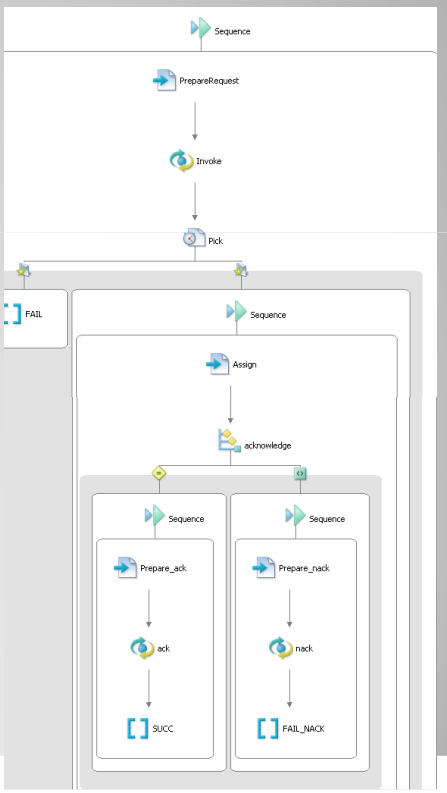
Buy service



MP3 service



User service



Conclusioni

Purtroppo il prototipo di demo CD-Mania non è funzionante a causa di problemi legati alla semantica, in quanto il processo di composizione veniva concluso ma non in maniera corretta, perché il file concrete .bpel conteneva solo i tag relativi a tutti gli oggetti, ma al loro interno non racchiudevano gli elementi effettivi; di conseguenza non avveniva il deploy del processo sull'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.

Il tool presenta ancora numerose limitazioni:

- allo stato attuale supporta WS-BPEL 1.1 e la roadmap non prevede attivita' 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.

Problemi incontrati:

- Installazione del tool → dovuta a baghi sulla definizione delle variabili d'ambiente e sui nomi delle directory
- Realizzazione dei file bpel → dovuta alla non conoscenza della versione del linguaggio, dei tipi di variabili e dei costrutti sintattici supportati:
  - Attività flow
  - Operazione di Assign
  - Definizione dei "recovery goal" troppo vincolante → impossibile trovare un piano
  - Semantica dei processi