Il Progetto ASTRO nella WS Composition: Analisi e Confronto con il Roman Approach

Tesina per il corso di Seminari
di Ingegneria del Software
Anno 2006-07
Docenti: proff. G.DeGiacomo,
M.Mecella, R.Rosati

Autore: Alessandro Pagliaro

Overview della Relazione

- Introduzione al problema della WS Composition
- Analisi dell'Approccio ASTRO
- Richiami sul Roman Approach
- Analisi di Confronto tra le due metodologie
- Esame del Toolset sviluppato dal Progetto ASTRO

Il Problema della WS Composition

- Allettanti possibilità offerte dai Composite Services
- Necessità di alta astrazione, focus sulla business logic

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 esecuzione automatica di composizioni e deleghi al Mw le operazioni low-level

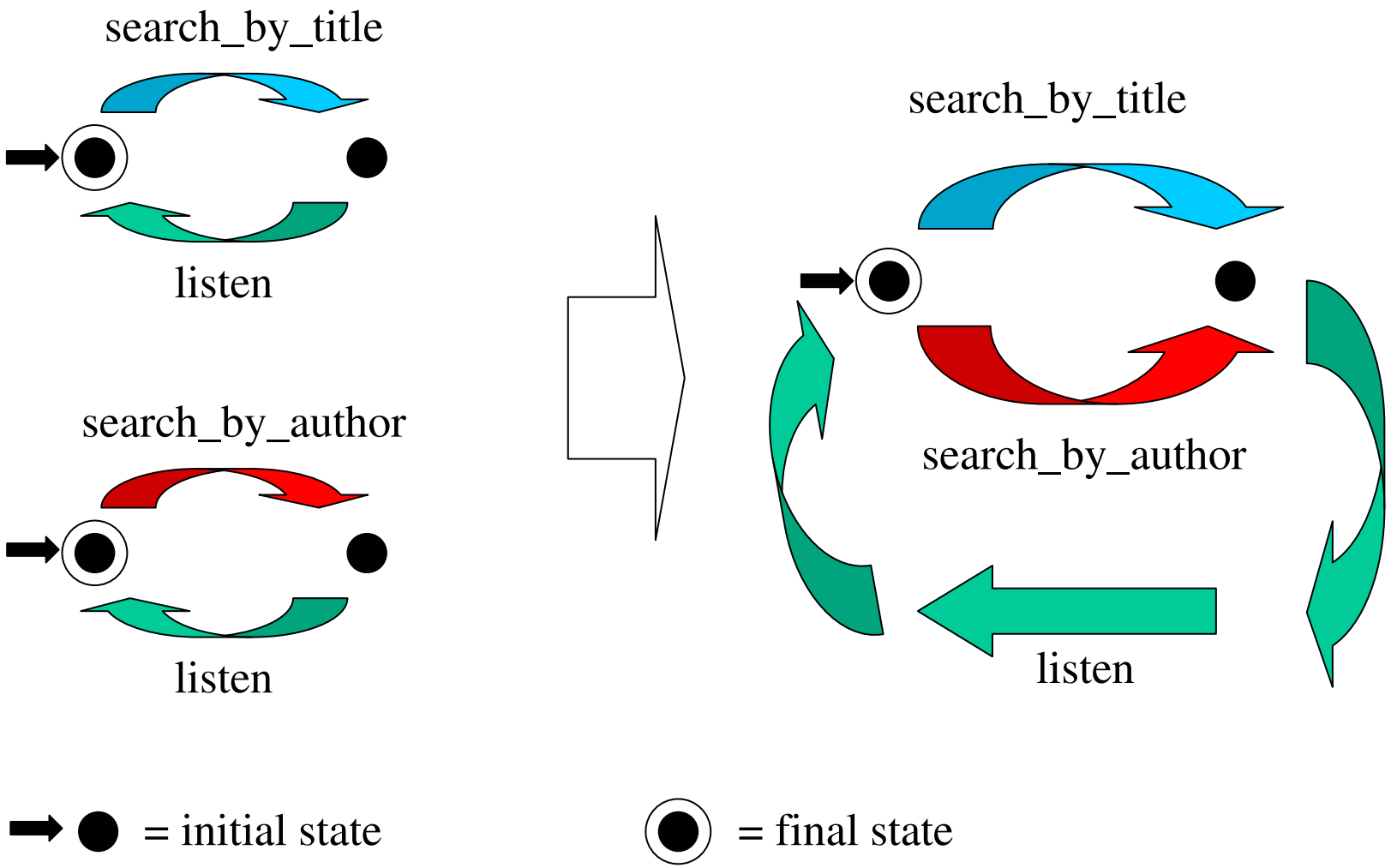
➢ un composition engine per testing, monitoring, verification

WS Behavioral Representation

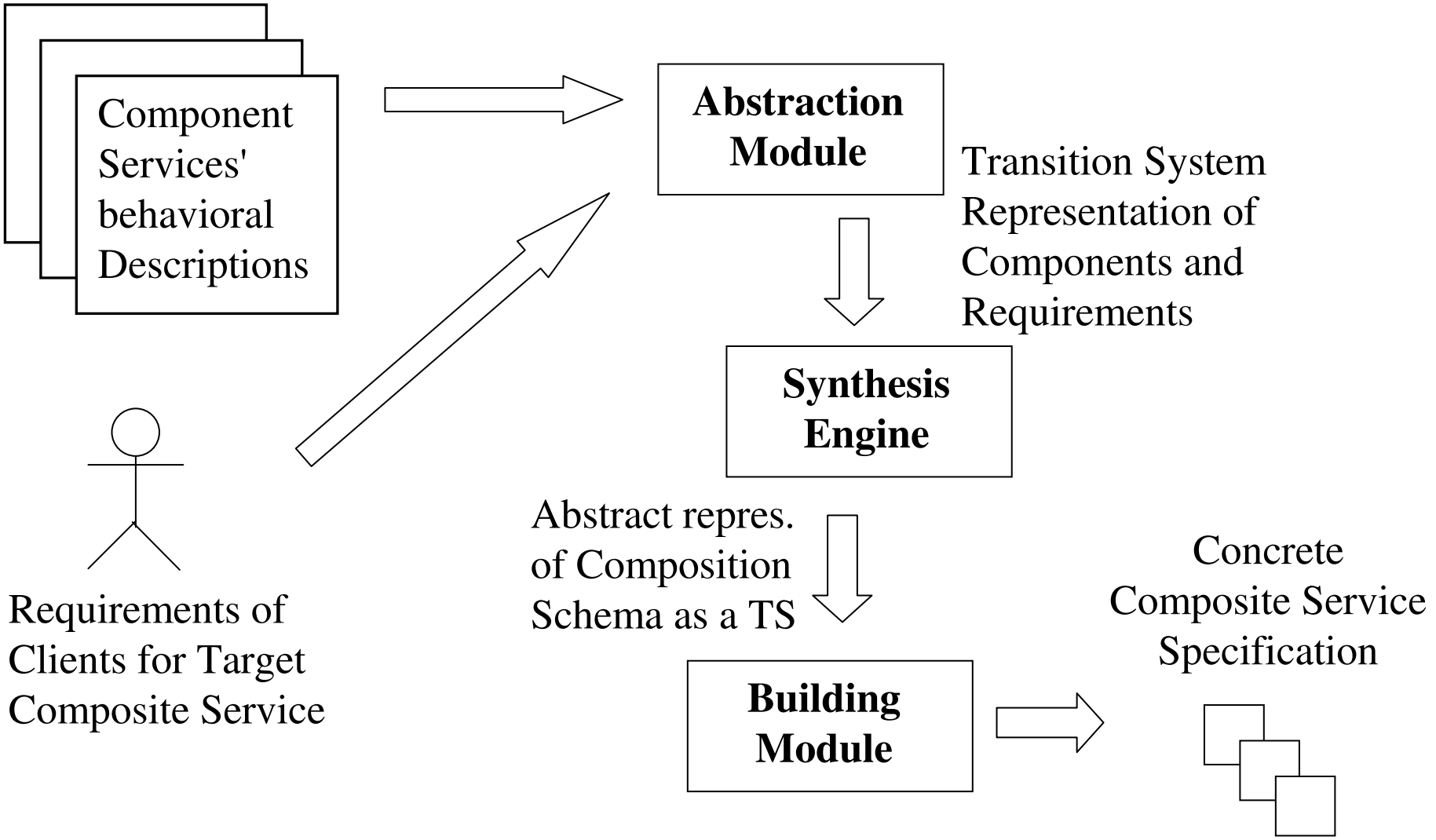
- Focus sul *comportamento* degli e-services in gioco
- Popolare descrizione usata in entrambi gli approcci in esame (sebbene con delle differenze): Transition System (FSM)

- Nodi come "stati stabili" dell'esecuzione;
- Archi come transizioni tra stati;
- Differenza tra *external actions* e *internal actions* (anche chiamate τ-transitions)

Component and Composite Services as TSs



The General WS Composition Workflow



L'Approccio ASTRO

Obiettivi dell'Effort ASTRO

- Fornire un framework per automatic service composition

- Fornire dei tools per implementare il framework, grande enfasi sulla traduzione nel concreto

- Fornire la possibilità di gestire l'intero lifecycle

- Automatizzare tasks noiosi ed error-prone

- Efficienza, ease-of-use, standards affermati

Macro-Aree del Project

- Modellazione di Business Requirements
- Automatic Service Synthesis
- Offline Service Verification
- Online Service Monitoring
- Semantics Support (still WIP)

Requirements in ASTRO

Essenzialmente due generi di requirements:

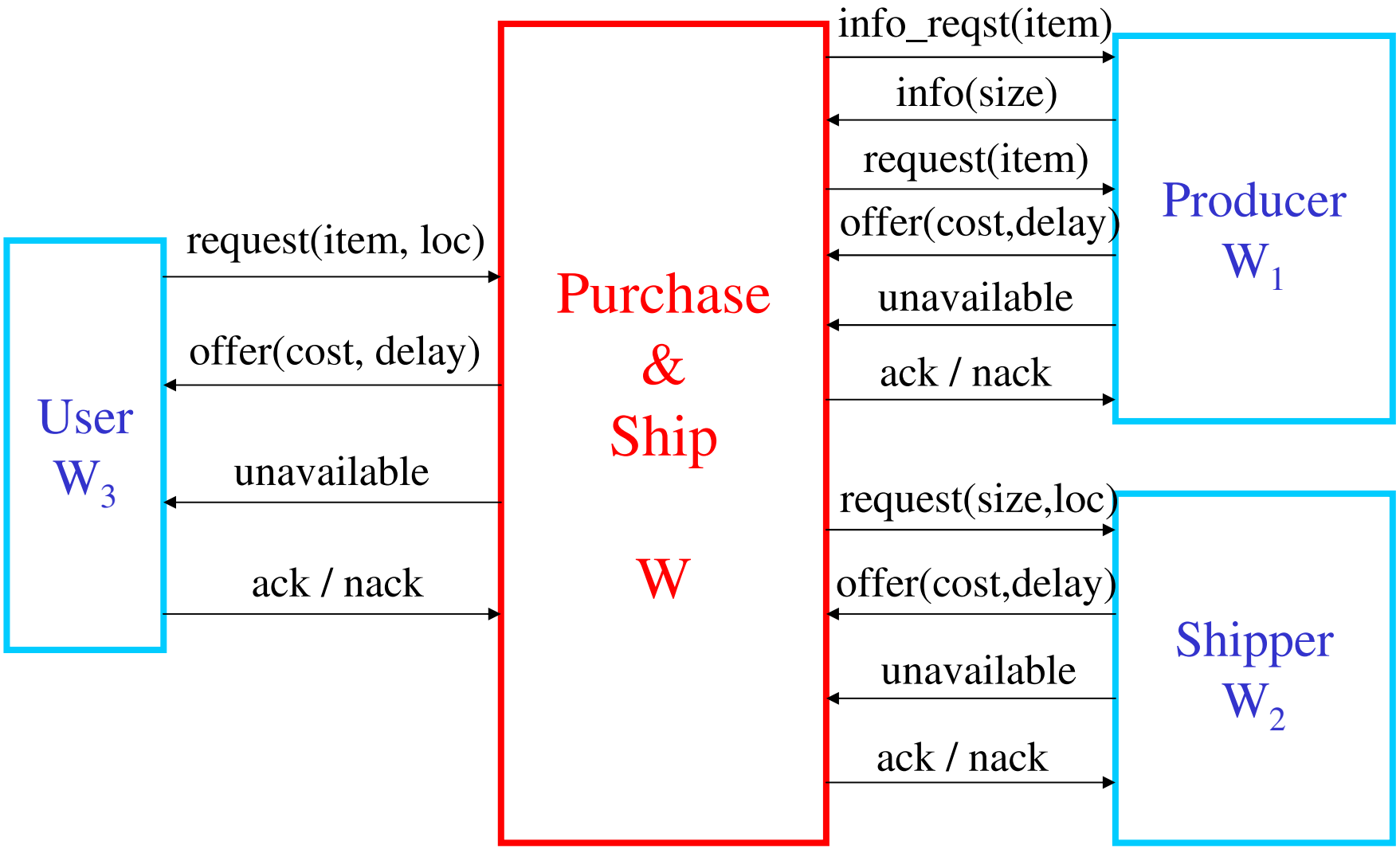
- Component e Target Services come Abstract BPEL Proc.
- Business Requirement Goals come EAGLE Formula

Presupposti dell'Approccio:

- Ambiente ASINCRONO
- Osservabilità PARZIALE dei servizi
- EXTENDED Business Goals

La rappresentazione comportamentale dei servizi è basata su STSs che distinguono *azioni di input, di output ed interne*, ad associano ad ogni stato un insieme di *proprietà soddisfatte*.

The Purchase and Ship Example Scenario



Business Goals & EAGLE

- Distinzione tra "Main Goal" e "Recovery Goals";
- NON un problema di reachability esprimibile in CTL;
- Presenza di condizioni di "forza" differente (Try vs Do);
- Presenza di situazioni "preferibili" (non modellabili con un semplice OR);

Esempio:

1. Try to sell items at home;
2. Upon failure, not a single commit must be done

Traduzione in EAGLE dell'esempio

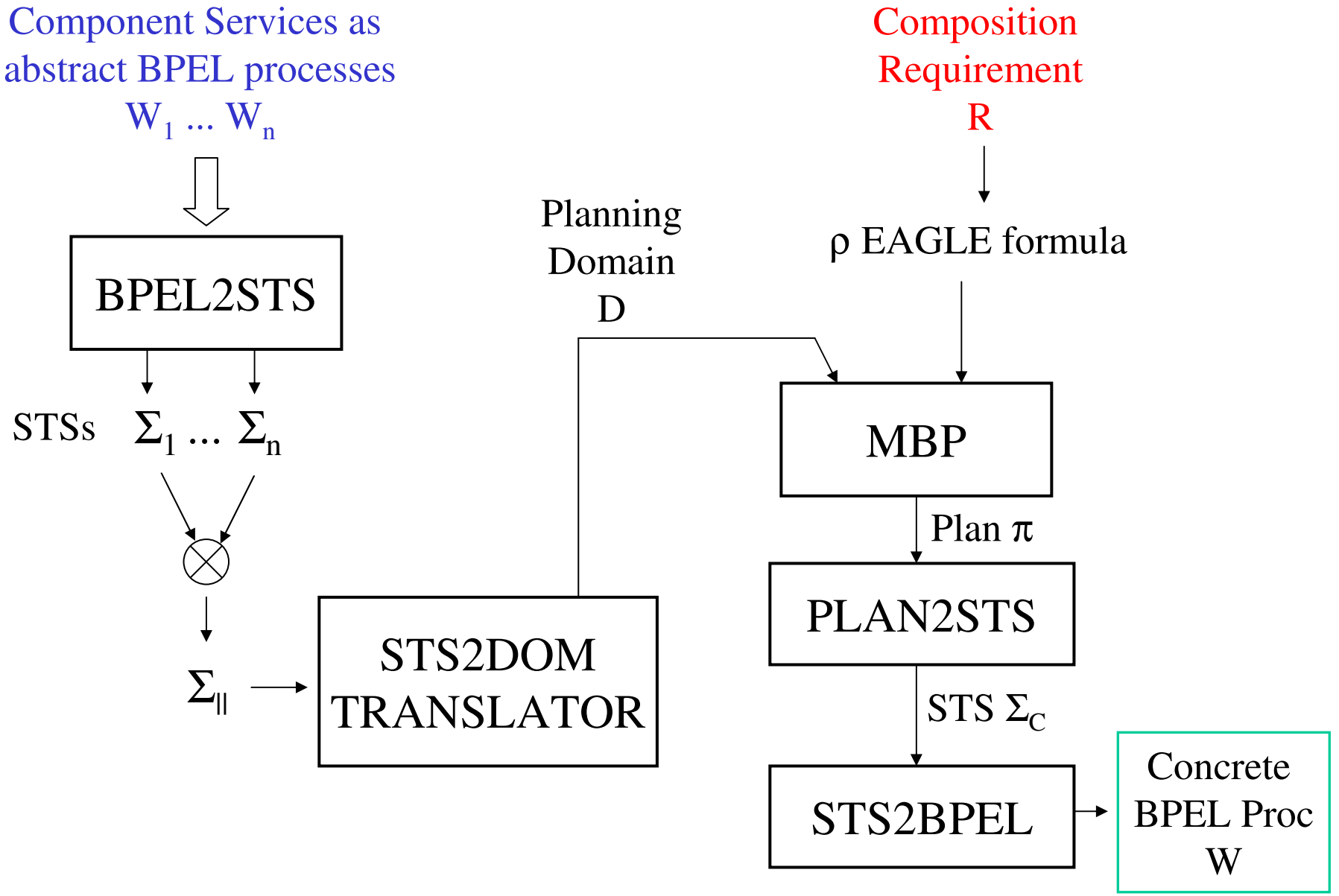
TryReach

user.pc = success && producer.pc = success && shipper.pc = success
&&
user.offer_delay = add_delay(producer.offer_delay + shipper.offer_delay)
&&
user.offer_cost = add_ cost (producer.offer_ cost + shipper.offer_ cost)

Fail DoReach

user.pc = failure && producer.pc = failure && shipper.pc = failure

The Astro Composition Workflow



Il Processo di Composizione

In sintesi: innanzitutto vogliamo costruire un Planning Domain D a partire dal Parallel Product dei Component Services + il Target Service: una sorta di combinazione di tutte le esecuzioni concorrenti dei servizi in gioco.

Dobbiamo passare al Belief-Level per ottenere Full Observability.

Risolviamo un problema di planning via Symbolic Model Checking, cercando un piano π che soddisfi il goal ρ nel dominio D. Da tale piano sintetizziamo un deadlock-free controller STS che è il nostro Composite Service.

Step I - Parallel Product

I Component Services e il Target Service vengono trasformati in STSs. Quindi viene calcolato il loro Prodotto Parallelo $\Sigma_{||}$:

Definizione: Parallel Product tra due STSs S1 e S2

Siano $\Sigma_1 = < S_1, S_1^0, I_1, O_1, R_1, L_1 >$ e $\Sigma_2 = < S_2, S_2^0, I_2, O_2, R_2, L_2 >$ due STSs tali che (I1 U O1) ∩ (I2 U O2) = insieme vuoto.

Il Prodotto Parallelo $\Sigma_1 \,||\, \Sigma_2$ tra Σ_1 e Σ_2 è definito come:

$$\Sigma_1 \,||\, \Sigma_2 = < S_1 \times S_2 , S_1^0 \times S_2^0 , I_1 \cup I_2 , O_1 \cup O_2 , R_1 \,||\, R_2 , L_1 \,||\, L_2 >$$

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

Step II - Controlled System

Il nostro scopo è trovare un STS Σ_C orchestratore per il Prodotto Parallelo, sotto certe condizioni. Σ_C deve essere un CONTROLLER per $\Sigma_{||}$.

Definizione: Controlled System

Siano $\Sigma = < S, S_0, I, O, R, L >$ e $\Sigma_C = < S_C, S_C^0, O, I, R_C, L_0 >$ due STSs tali che la funzione di labeling di Σ_C sia nulla per ogni stato s_C, cioè $L_0 (s_C) =$ insieme vuoto per ogni s_C in Σ_C.

Il nuovo STS $\Sigma_C \mid> \Sigma$ descrive il comportamento di Σ controllato da Σ_C, ed è definito come:

$$\Sigma_C \mid> \Sigma = < S_C \times S , S_C^0 \times S^0 , I , O , R_C \mid> R , L >$$

dove:

$< (s_C, s), \tau, (s_C', s') >$ appartiene a $(R_C \mid> R)$ se $< s_C, \tau, s_C' >$ appartiene a R_C;

$< (s_C, s), \tau, (s_C , s') >$ appartiene a $(R_C \mid> R)$ se $< s , \tau, s ' >$ appartiene a R;

$< (s_C, s), a, (s_C', s') >$ appartiene a $(R_C \mid> R)$, con a diverso da τ, se $< s_C, a, s_C' >$ appartiene a R_C e inoltre $< s , a, s ' >$ appartiene a R;

Step III - Deadlock-free Controller

Non tutti i controllers possibili soddisfano le nostre esigenze. Vorremmo che il sistema controllato possa sempre essere in grado di ricevere messaggi dal controller.

Definizione: Deadlock-Free Controller w.r.t the controlled STS

Siano $\Sigma = < S, S^0, I, O, R, L >$ e $\Sigma_C = < S_C, S_C^0, O, I, R_C, L_0 >$ due STSs tali Σ_C è un controller per Σ.

Σ_C è detto "deadlock-free per Σ" se per ogni stato (s_C, s) in S_C x S raggiungibile dagli stati iniziali del Controlled System $\Sigma_C \mid> \Sigma$, sono soddisfatte le seguenti proprietà:

1) se in R compare una transizione del tipo $< s, a, s' >$ con 'a' azione di *output*, allora esiste uno stato s_C' appartenente alla τ-closure(s_C) tale che in R_C compaia la transizione $< s_C', a, s_C'' >$ per qualche s_C'' appartenente a S_C;

2) se in R_C compare una transizione del tipo $< s_C, a, s_C' >$ con 'a' azione di *input*, allora esiste uno stato s' appartenente alla τ-closure(s) tale che in R compaia la transizione $< s', a, s'' >$ per qualche s'' appartenente ad S;

Step IV - Belief State

Per soddisfare il composition goal ρ, 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 Partial Observability (il Controller non ha piena osservabilità sul Prodotto Parallelo Controllato).

Ci portiamo quindi al *Belief-Level*, ovvero consideriamo sets di stati ugualmente plausibili date le nostre conoscenze, che evolvono tramite *external transitions* includendo nel nuovo Belief State stati raggiungibili tramite τ-*closure*.

Belief Evolution

Definizione: Belief Evolution
Sia Σ un STS e B un Belief , con B sottoinsieme di S.
Definiamo la *Belief Evolution* di B a causa dell'azione 'a' come un nuovo Belief B' = Evolve(B, a) etale che:

> Evolve(B, a) = { s' | exists s appartenente alla t-closure(B) con
> < s, a, s' > appartenente ad R }

Le assunzioni di ambiente asincrono e t-transitions complicano il concetto di "proprietà soddisfatta da un Belief State".
Diciamo che *un Belief soddisfa una proprietà p se tutti i suoi stati la soddisfano oppure, se uno stato s non la soddisfa allora esiste uno stato s' appartenente alla τ-closure(s) che la soddisfa.*

Step V - Belief-Level System

Ci prepariamo al planning considerando un STS che caratterizza il Controlled System e i cui stati sono Belief-States

Definizione: Belief-Level System

Sia $\Sigma = < S, S^0, I, O, R, L >$ un STS. Il corrispondente Belief-Level STS $\Sigma_B = < S_B, S_B^0, I, O, R_B, L_B >$ è definito nel seguente modo:

1) S_B è l'insieme di Beliefs di S raggiungibile dall'insieme di Beliefs iniziali S_B^0;

2) $S_B^0 = \{ S^0 \}$;

3) se Evolve(B, a) = B', con B' diverso dall'insieme vuoto, per qualche azione a di input o ouput, allora $< B, a, B' >$ appartiene a R_B;

4) $L_B (B) = \{$ p appartenenti a *Prop* $|$ $B \models_\Sigma p \}$.

The Astro Composition Problem

Vantaggi dei Belief-Level System:

✓ Un unico initial state;

✓ FULLY Observable;

✓ Per ogni coppia <B,a> esiste al massimo un B' tale che <B, a, B'> è in R_B.

Definizione: Astro Composition Problem

Siano $\Sigma_1, ..., \Sigma_n$ un insieme di STSs, e ρ un composition requirement.
Il problema di composizione per $\Sigma_1, ..., \Sigma_n$ e ρ è il problema di trovare un Controller Σ_C che è deadlock-free e tale che $\Sigma_B \models \rho$, dove Σ_B è il Belief-Level System dell'STS $\Sigma_C \triangleright (\Sigma_1 \| ... \| \Sigma_n)$.

Planning Preparations: the Domain

Per applicare algoritmi di planning e risolvere il problema via Symbolic Model Checking, costruiamo un Dominio D.

Tale dominio è un altro STS < S, S^0, A, T, L > costruito a partire dal Belief-Level System del Parallel Product dei servizi coinvolti; uno stato del dominio D è formato da una coppia < belief-state, last-transition-output >; un carattere speciale * è usato per caratterizzare output-action transitions e casi in cui non c'è stato output nell'ultima transizione.

Domains, Goals and Plans

Vogliamo trovare un piano π per il dominio D che soddisfi un goal g.

π è una tupla $< C, c^0, \alpha, \varepsilon >$, un insieme di contesti di esecuzione e funzioni di evoluzione per azioni e contesti; eseguire π su D significa considerare le possibili evoluzioni di configurazioni, coppie <contesto, stato>.

Siamo interessati a piani eseguibili, da cui possiamo ricavare un STS Σ_π deadlock-free in modo che il sistema controllato $\Sigma_\pi \mid> \Sigma_{\|}$ soddisfi il goal g.

A Solution through Planning

Lemma: Controller/Plan Executability
Sia Σ un deadlock-free STS, Σ_B il suo Belief-Level System e D il corrispondente planning domain. Sia inoltre π un piano per D, e Σ_π l'STS corrispondente a π.
Si ha che se π è eseguibile su D, allora Σ_π è eseguibile su Σ.

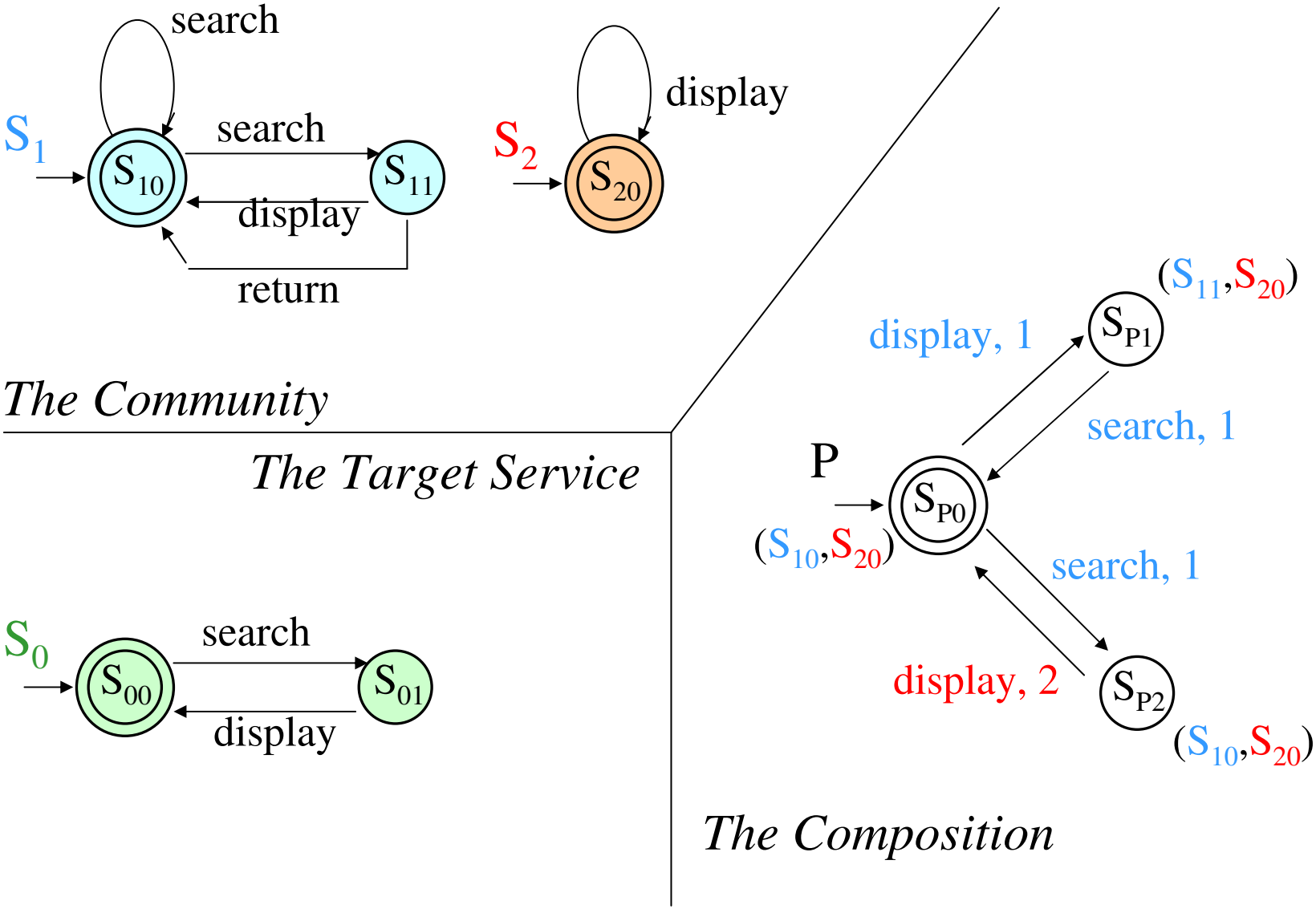
Lemma: Controller/Plan Equivalence
Sia Σ un deadlock-free STS, Σ_B il suo Belief-Level System e D il corrispondente planning domain. Sia inoltre π un piano per D, e Σ_π l'STS corrispondente a π.
Si ha che se π è soluzione su D per il goal g, allora è vero che $(\Sigma_\pi \mid> \Sigma) \mid= g$.
Inoltre, se esiste un STS Σ_C tale che $(\Sigma_C \mid> \Sigma) \mid= g$, allora esiste un piano π che è soluzione per il goal g su D.

Richiami sul Roman Model

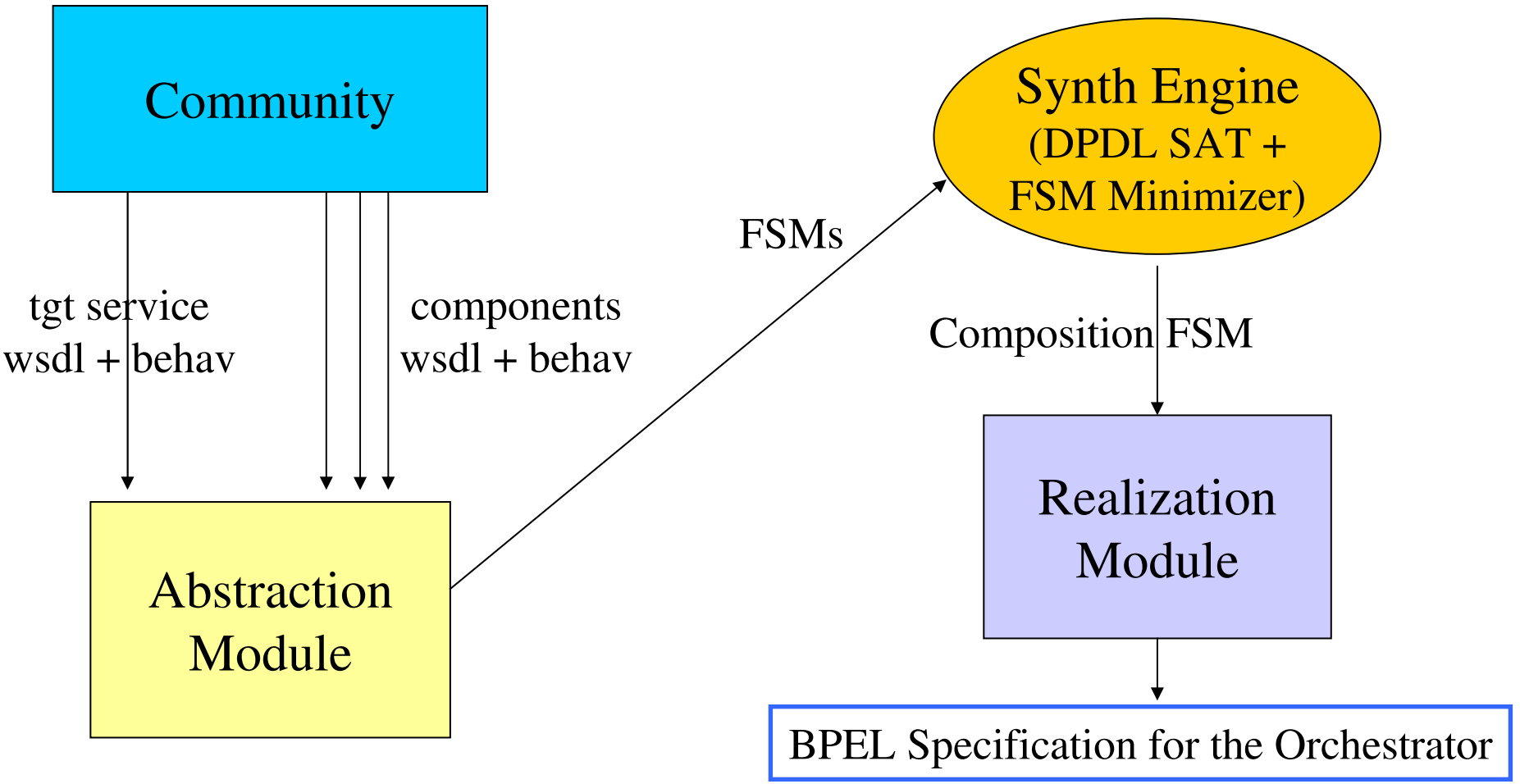
Roman Model basics

- Services' "exported behaviors";
- The Community as a "common understanding of a shared set of actions";
- TSs as tuples $< \Sigma, S, S^0, \delta, F >$
- Possibilità di "delegare" parte delle proprie azioni ad altri membri;
- Full Observability dal parte dell'orchestrator tramite runtime querying;
- Deterministic Target Service assembled via "building blocks";
- Problem Reduction to DPDL SAT, use of small model property to solve the problem in EXPTIME.

An Example of Roman Model Community



The Roman Model Composition Workflow



Confronto tra i due Approcci

Comparisons I

Il Problema di Composizione

Entrambi gli approcci hanno un taglio riconducibile all'approccio Client-Tailored, in quanto subordinano la costruzione del Composite Service ad un target service specificato dall'utente.

L'atteggiamento "teoria vs pratica" delle due metodologie sembra essere alquanto differente: le solide basi teoriche e gli obiettivi ambiziosi del Roman Model da un lato, la dichiarata massima attenzione all'obiettivo di fornire un tool concreto del Progetto Astro dall'altro.

Comparisons II

Componenti, Requisiti, Architettura di Fondo

Modellazione per entrambi dei servizi via TSs, seppur con diversi "flavours". Senza dubbio la differenza più evidente è la presenza della struttura Community nel Roman Model, con tutti i benefici e le assunzioni che essa comporta, mentre in Astro non si hanno ipotesi di architetture pre-esistenti. Conseguenze: *full* vs *partial* observability, benefici indotti dall'architettura da bilanciare con gli oneri che essa comporta, alfabeti di composizione del tgt service leggermente differenti tra loro ("building bricks" vs "composite interface" approaches).

Comparisons III

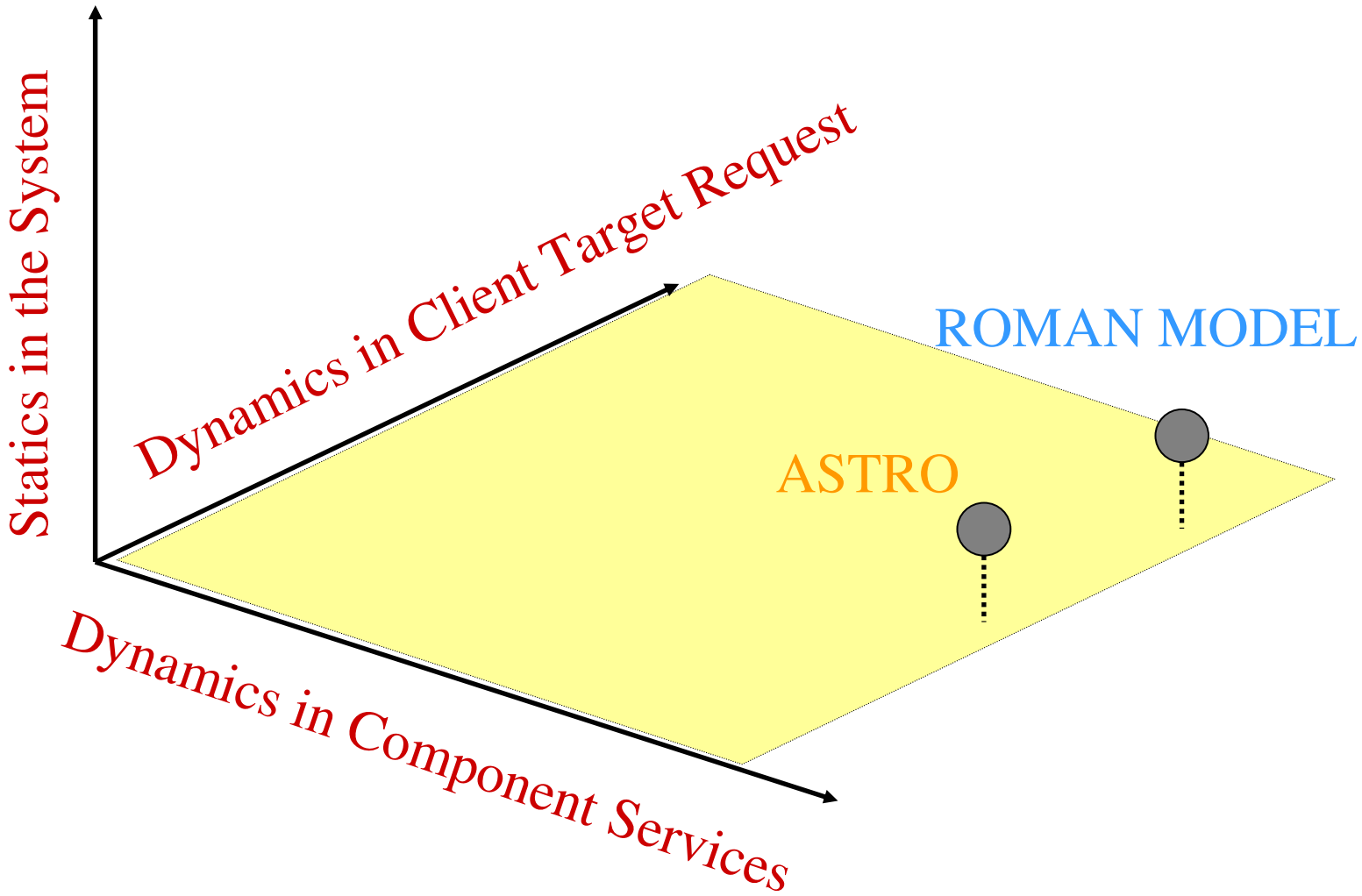
Astrazione e Sintesi

Astro Abstraction: internal/external transitions, state properties
Roman Abstaction: Community actions, possible final states
Synthesis: DPDL SAT vs Planning via Model Checking

Risultato Finale della Composizione

Ci sono delle differenze: Astro vuole fornire un processo eseguibile che concilii l'esecuzione dei Component Services, un nuovo servizio, definito dall'utente ma "statico".
Il Roman approach porta all'estremo l'atteggiamento Client-tailored avendo come scopo la costruzione di un internal schema per l'orchestrator, per usare differenti fragments di servizi secondo il "whim" dell'utente a runtime.

Our Approaches in the "WS Composition 3D Space"



L'Astro Suite toolset v3.4

A cooperative collection of modules

- Java and Eclipse environment;
- Tomcat application server;
- ActiveBPEL Engine;

ASTRO wsToolset:

- wsTranslator
- synTools
- NuSMV
- wsMonitor
- wsRequirements Eclipse plugin
- wsChainManager Eclipse plugin
- wsAnimator Eclipse plugin, wsUseCases

Synthesis Procedure

Component Services e Target Service sono espressi come Abstract BPEL Processes + WSDL files.

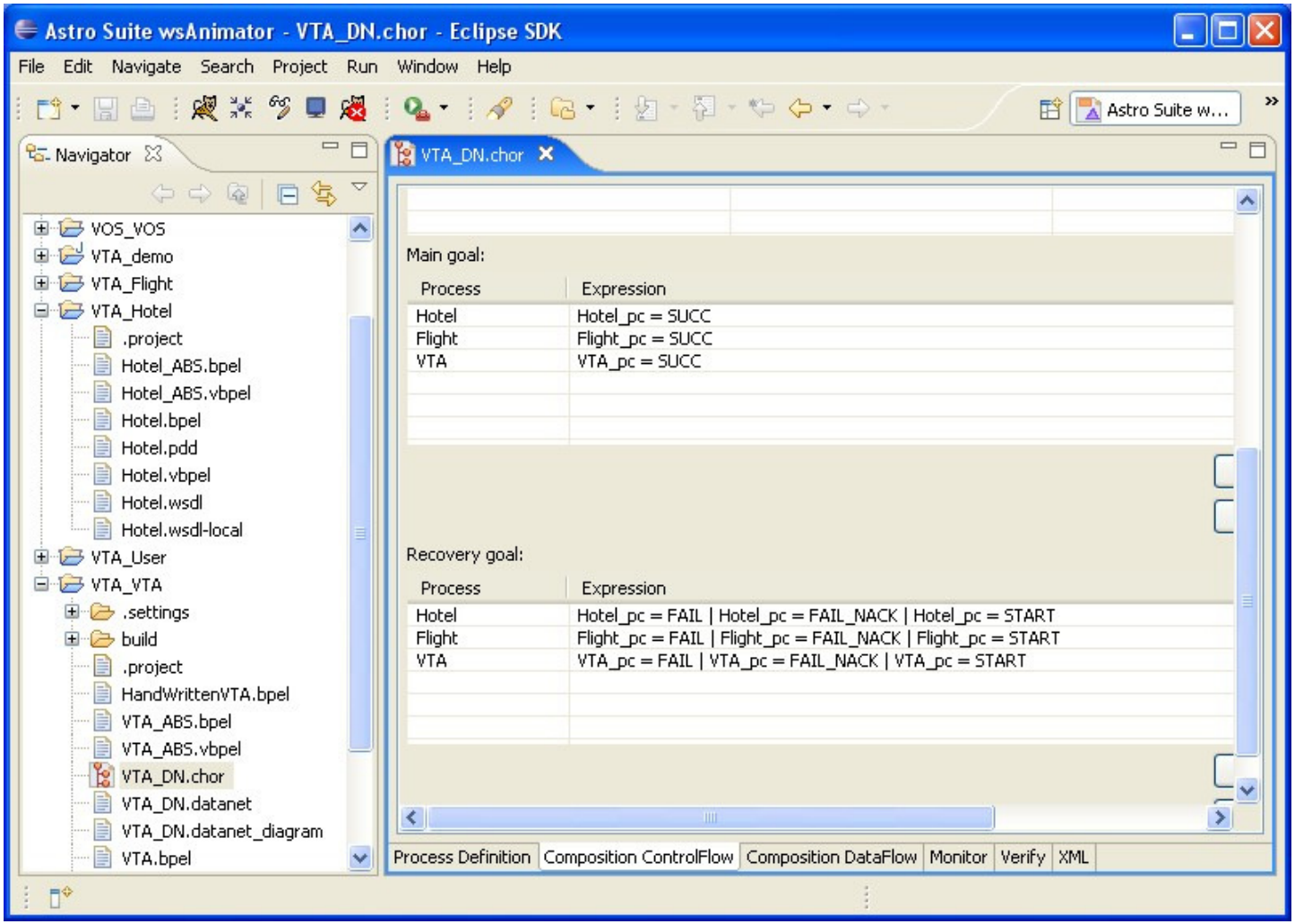
Un Wizard permette la creazione di un file XML "di coreografia" (.chor extension) che racchiude l'intero problema (component, target, requirements, properties to be monitored and verified) ed è ispezionabile ed editable.
Una serie di chiamate ai vari software modules componenti il toolset permette la realizzazione pratica dell'approccio teorico discusso in precedenza.
Il concrete BPEL Process ottenuto è automaticamente deployed su BPEL Engine, e può essere monitorato e testato.

Verification and Monitoring

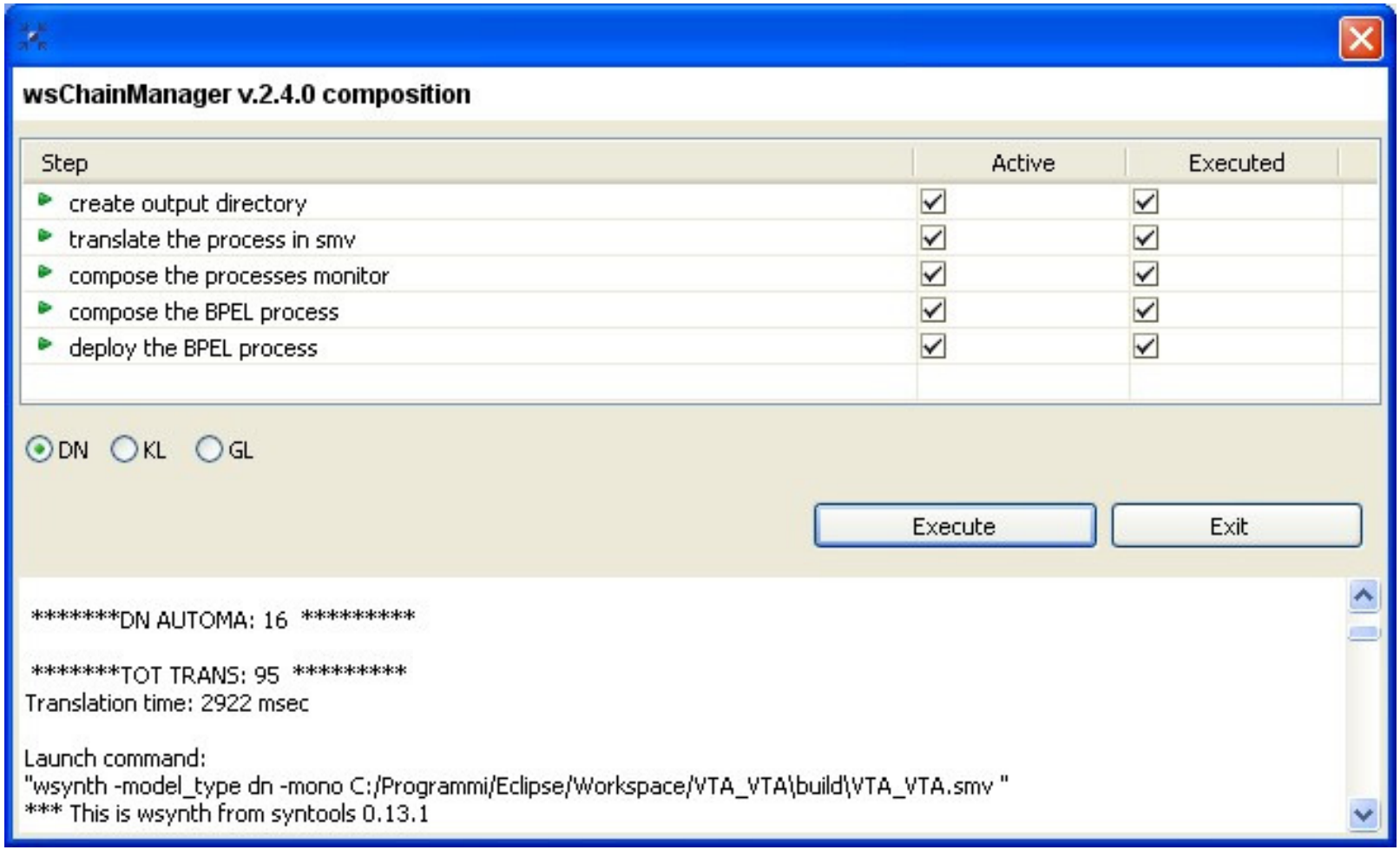
Per Verification si intende una procedura offline che a partire dal file .chor controlla determinate proprietà (sulla scia dell'uso di Metodi Formali nell'Ingegneria del Software) e fa rapporto all'utente, anche fornendo controesempi stile UML Sequence Diagram.

Il Monitoring è una procedura online in cui viene eseguito del Java code generato automaticamente a partire dal file .chor per monitorare il running process del composite service e fornire reports su situazioni di interesse all'utente.

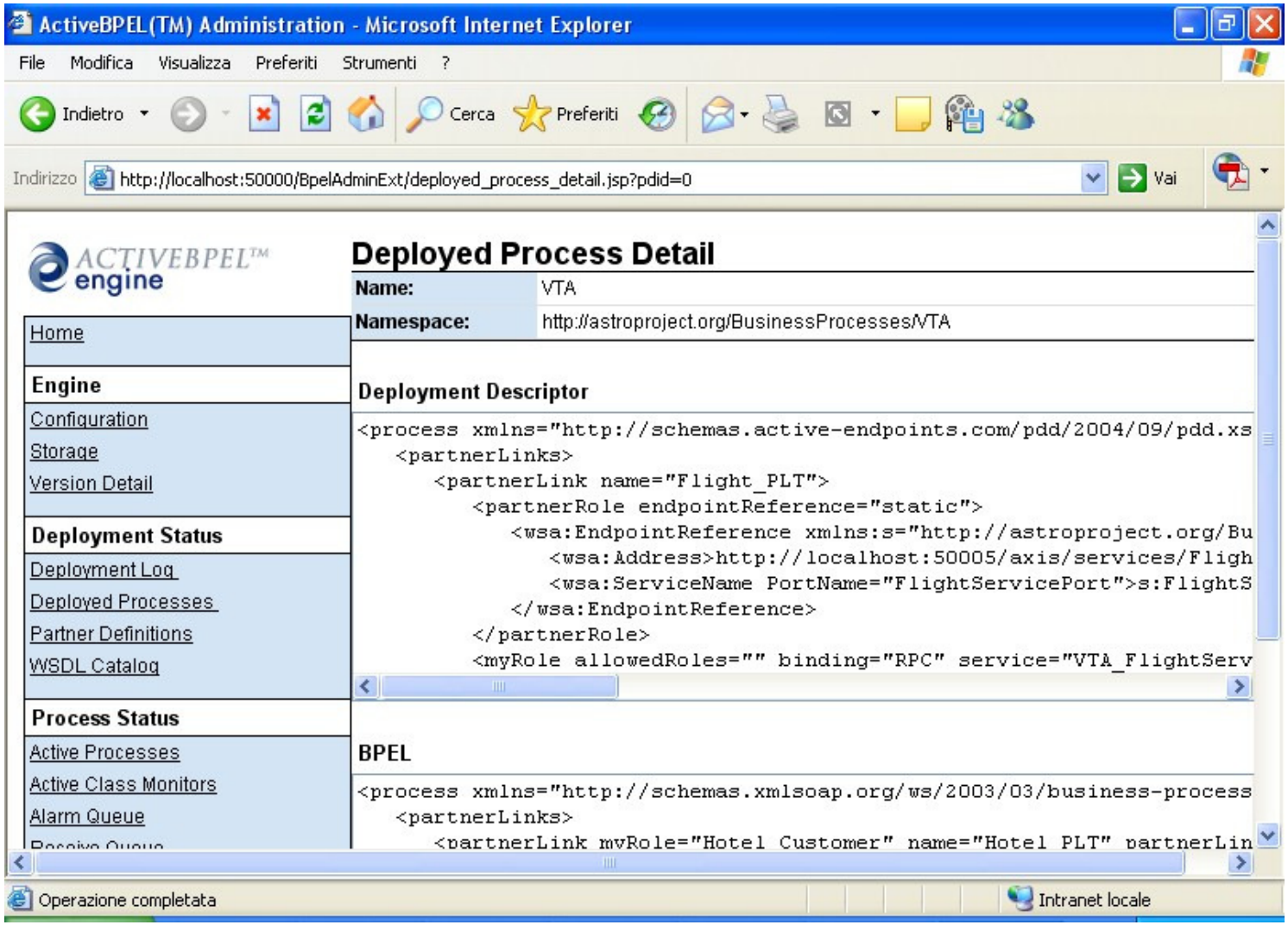
Composition Requirements



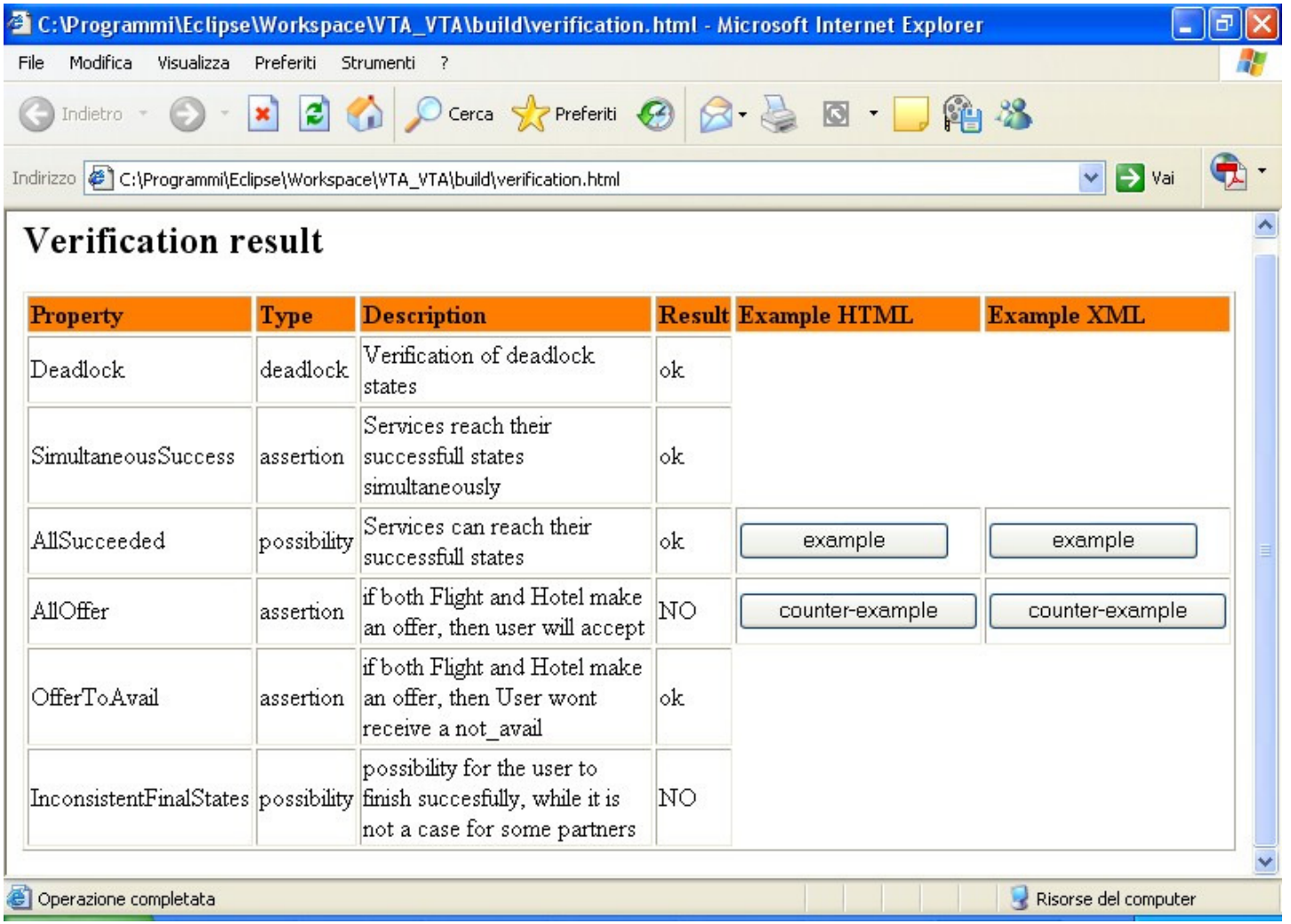
ChainManager



Deployed Process



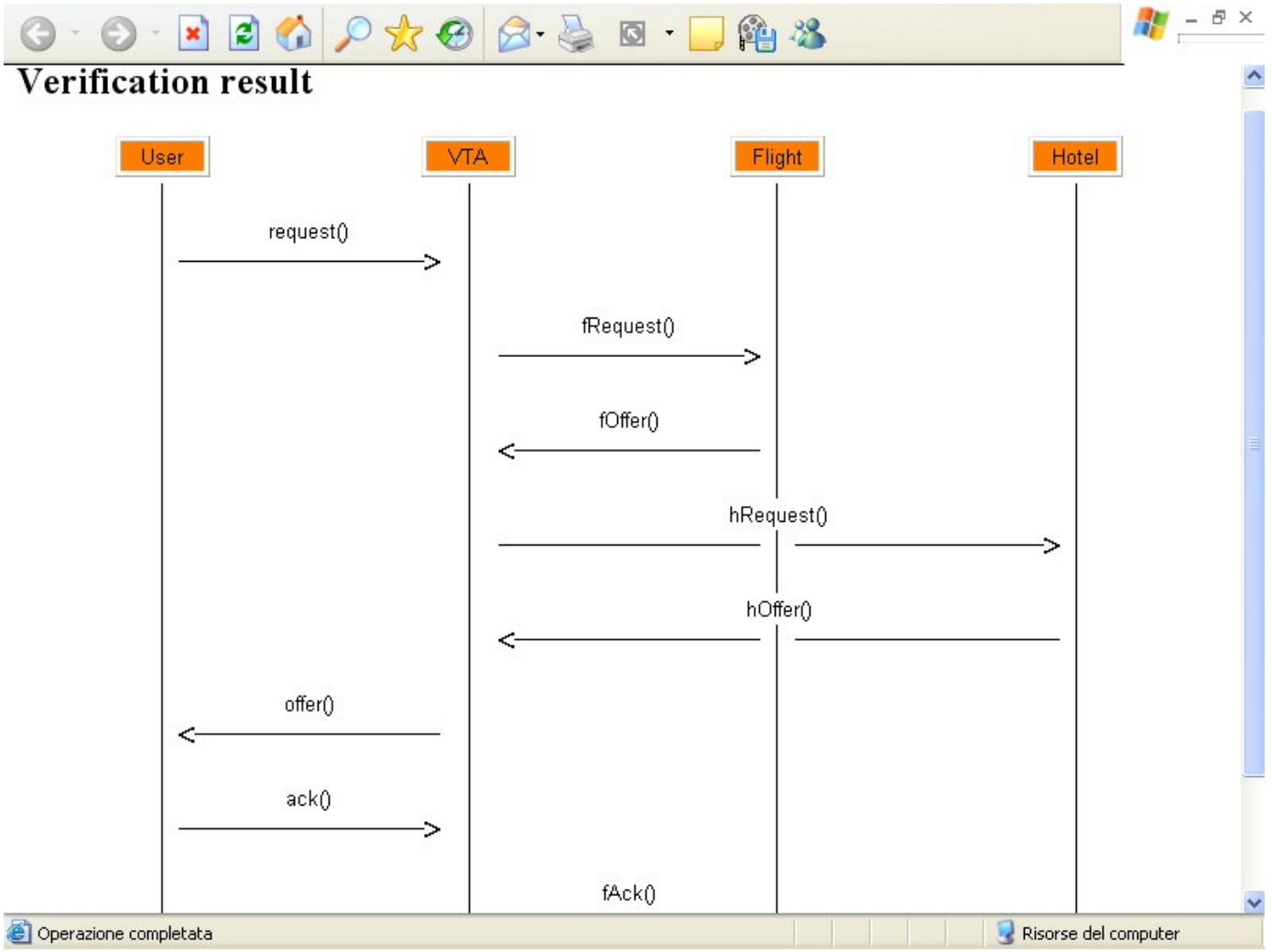
Offline Verification



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		

Verification Instance Found



Grazie per l'attenzione