Università degli Studi di Roma "La Sapienza"

Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria Informatica

Composizione automatica di servizi: l'approccio ASTRO e il Roman Model a confronto

Corso di "Seminari di Ingegneria Del Software"

Prof. Giuseppe De Giacomo

Autore

Alessandro Dionisi

Anno Accademico 2005/2006

Sommario

- Introduzione al problema

Come gli approcci affrontano le diverse fasi:

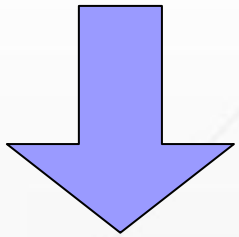
- Rappresentazione dei servizi
- Specifica dei requisiti
- Processo di sintesi automatica

- Conclusioni

Introduzione (1)

Ampia diffusione dei web services

- Sempre più organizzazioni espongono frammenti delle proprie applicazioni
- È facile trovare on-line “comunità” di web services
  - Es. http://xmethods.org/, eBay.com

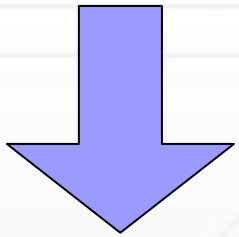


Costruire nuovi servizi, componendo quelli esistenti

Introduzione (2)

Attualmente la composizione avviene manualmente

- Il cliente specifica i suoi requisiti
- Il progettista concepisce un processo BPEL4WS che invochi adeguatamente i servizi esistenti



Come rendere automatica questa procedura?

- Numerosi progetti di ricerca al riguardo, soprattutto nell'ambito dell'intelligenza artificiale

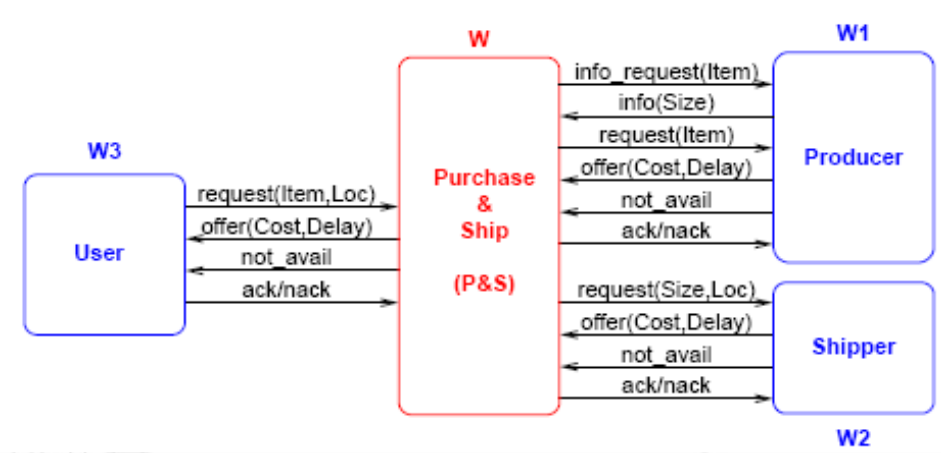
Obiettivi del lavoro

Confrontare due possibili soluzioni

- Il progetto ASTRO
  - Ideato nell'ambito di un progetto comune tra ITC-IRST e il DIT dell'Università di Trento
- Il Roman Model
  - Concepito da docenti e ricercatori del nostro dipartimento
- Simili in alcuni aspetti
  - E' comunque difficile trovare dei problemi risolubili in entrambi gli approcci

Scenario di composizione (ASTRO)

Nel seguito verrà considerato il seguente esempio:



- Si desidera fornire il servizio Purchase & Ship (W), a partire dai servizi già esistenti Producer (W1), Shipper (W2) e User (W3)
- Si vuole che i servizi esistenti, incluso quello che rappresenta il client, cooperino per soddisfare determinati requisiti

Rappresentazione dei servizi (1)

Visione comune dei servizi esistenti

- Oltre all'interfaccia è necessario modellare il comportamento (conversazioni supportate)
  - L'interfaccia di invocazione è definita in WSDL, mentre per la descrizione comportamentale si impiegano processi astratti BPEL4WS (ASTRO) o WS-CDL (Roman Model)
- Rappresentazione compatta per mezzo di State Transition Systems (STS)
  - È una tupla $\langle S, S^o, I, O, R, L \rangle$
  - Può trovarsi in uno *stato*
  - Passa da uno stato al successivo inviando/ricevendo *messaggi* o per transizioni interne (τ-transizioni)

Rappresentazione dei servizi (2)

Ciò che caratterizza il servizio è l'insieme di azioni con cui esso contribuisce nella comunità di servizi

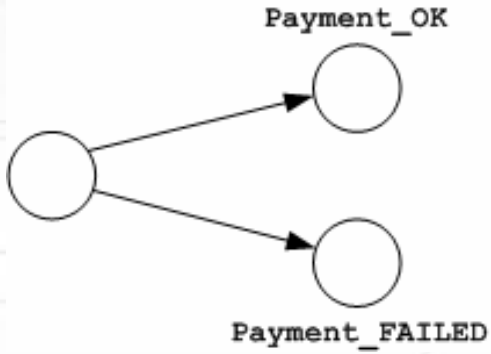
- Astraendo rispetto alle descrizioni in WSDL si può ottenere un *alfabeto* di azioni (atomiche)
- E' importante che tutti i partecipanti abbiano la stessa comprensione dell'alfabeto

In entrambi gli approcci siamo di fronte ad un mapping Local-as-View

- I servizi sono mappati sulla *community ontology* come STS, utilizzando l'alfabeto della community

Non-determinismo e osservabilità

I servizi esistenti possono avere comportamenti non-deterministici (es. servizio carta di credito)



- Questo porta a *parziale controllabilità* del sistema
- Osservando lo stato prima e dopo la transizione l'orchestratore può conoscere quale è stata scelta

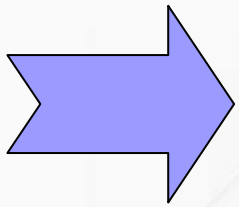
Piena osservabilità

Specifica dei requisiti (ASTRO)

Gli obiettivi della composizione sono espressi come *extended goals* tramite una formula EaGLe

- E' possibile specificare condizioni su interi percorsi del piano piuttosto che raggiungibilità di stati
- Possibilità di indicare preferenze sui percorsi e condizioni che invece devono verificarsi in caso di fallimenti del sistema (eccezioni)

try to "sell items at home";
upon failure,
do "never a single commit".

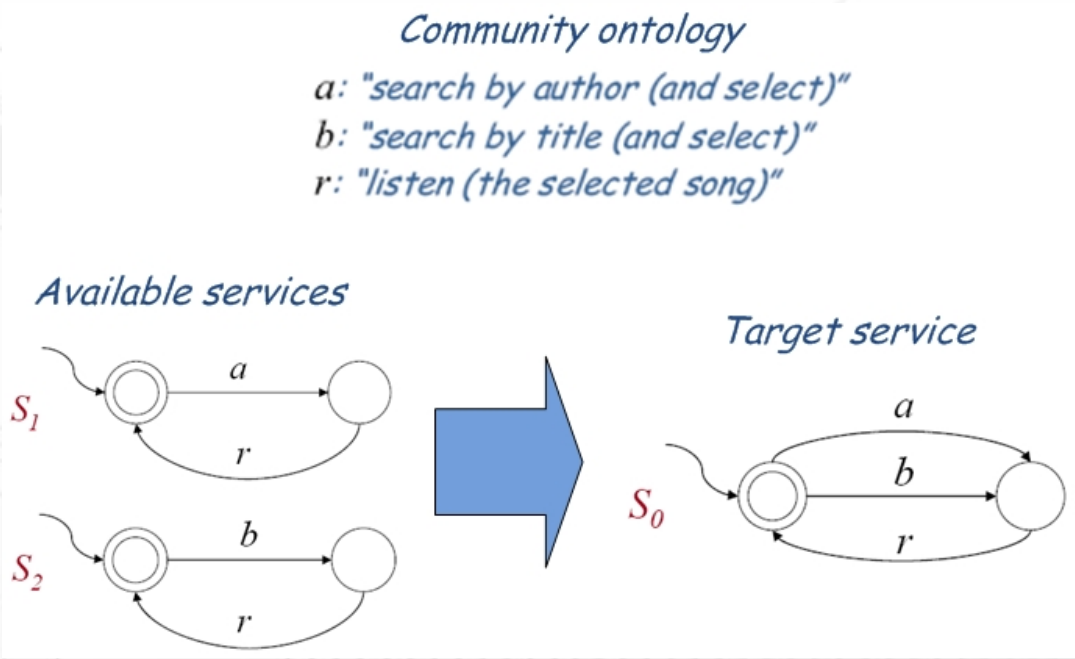


TryReach
`user.pc=SUCC∧producer.pc=SUCC∧shipper.pc=SUCC∧`
`user.offer_delay=add_delay(producer.offer_delay,`
`shipper.offer_delay)∧user.offer_cost=add_cost(`
`producer.offer_cost)`
Fail DoReach
`user.pc=FAIL∧producer.pc=FAIL∧shipper.pc=FAIL`

Specifica dei requisiti (Roman Model)

Si definisce un apposito STS deterministico, che codifica il comportamento desiderato dall'utente

- Vengono utilizzate solo le azioni disponibili nella *community ontology*

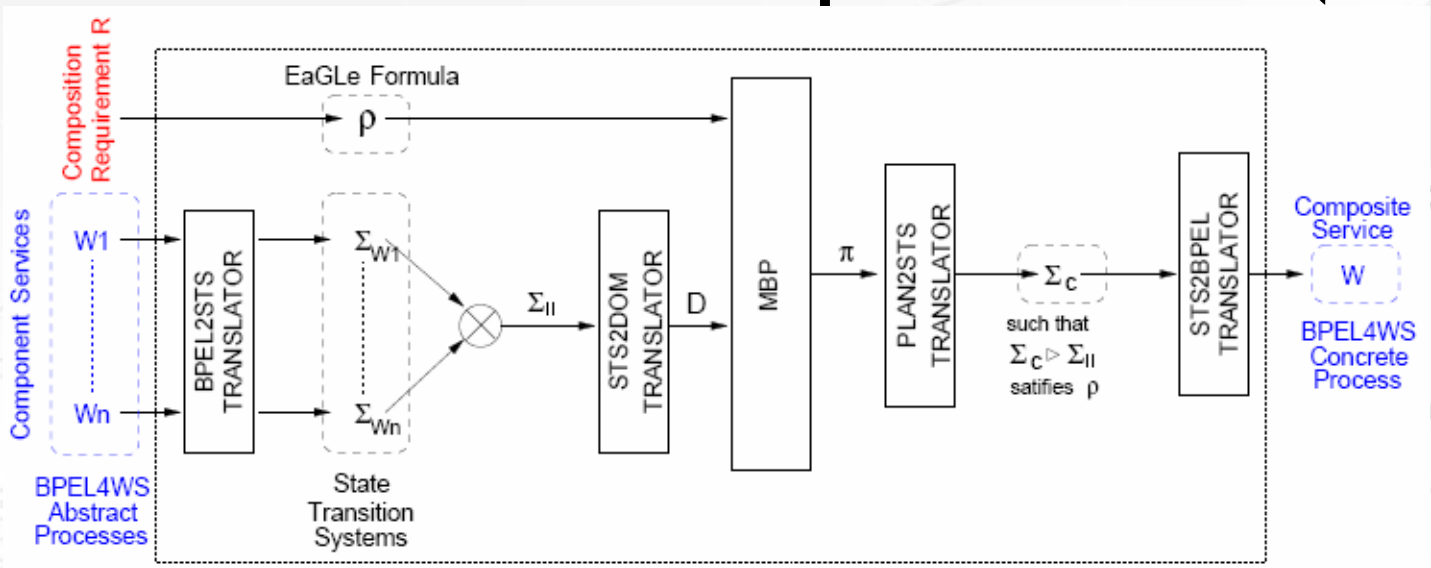


Il processo di sintesi automatica

Obiettivo finale: sintesi automatica di un "programma" per l'orchestratore

- ASTRO: impostazione di un problema di planning
  - Si ottiene un controller per i servizi esistenti
  - Una volta sintetizzato il piano, il client non può influire sul flusso di esecuzione (processo batch)
- Roman Model: soddisfacimento di una formula DPDL
  - Una volta trovato un modello della formula, essa contiene i "comandi" per l'orchestratore
  - Ciò che viene creato è un vero e proprio servizio che può essere condizionato dalle decisioni del client

Formalizzazione del problema (ASTRO)



- Costruzione del prodotto parallelo $\Sigma_{||}$ a partire dagli STS dei servizi esistenti $\Sigma_{W_1} \ldots \Sigma_{W_n}$
- Creazione del dominio $\mathcal{D}$, relativo a $\Sigma_{||}$
- A partire dallo stato iniziale s, il dominio $\mathcal{D}$ e i requisiti ρ, si genera il piano π
- Dal piano si ricava il controller Σ_c per $\Sigma_{||}$

Prodotto parallelo (1)

Dati due STS $\Sigma_1 = \langle S_1, S_1^0, I_1, O_1, R_1, L_1 \rangle$ ed $\Sigma_2 = \langle S_2, S_2^0, I_2, O_2, R_2, L_2 \rangle$ il prodotto parallelo $\Sigma_1 \| \Sigma_2$ è definito come:

$$\Sigma_{\|} = \Sigma_1 \| \Sigma_2 = \langle S_1 \times S_2, S_1^0 \times S_2^0, I_1 \cup I_2, O_1 \cup O_2, R_1 \| R_2, L_1 \| L_2 \rangle$$

dove:

$$\langle (s_1, s_2), a, (s_1', s_2) \rangle \in (R_1 \| R_2) \text{ se } \langle s_1, a, s_1' \rangle \in R_1;$$

$$\langle (s_1, s_2), a, (s_1, s_2') \rangle \in (R_1 \| R_2) \text{ se } \langle s_2, a, s_2' \rangle \in R_2;$$

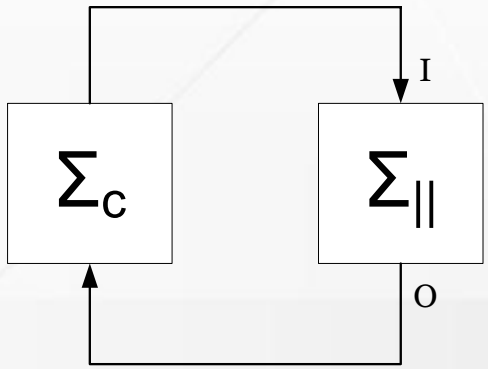
e:

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

Prodotto parallelo (2)

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

- Il problema di composizione automatica consiste nel costruire un *mediatore* Σ_c che riesca a controllare $\Sigma_{||}$ assolvendo i requisiti ρ
- Ovvero $\Sigma_c \triangleright \Sigma_{||}$ (ovvero Σ_c controlla $\Sigma_{||}$)



Sistema controllato (1)

Siano $\Sigma = \langle S, S^0, I, O, R, L \rangle$ e $\Sigma_c = \langle S_c, S_c^0, O, I, R_c, L_\varnothing \rangle$ due STS, dove $L_\varnothing(s_c) = \varnothing$ per ogni $s_c \in S_c$. Lo state transition system $\Sigma_c \triangleright \Sigma$ che descrive Σ quando controllato da Σ_c risulta:

$$\Sigma_c \triangleright \Sigma = \langle S_c \times S, S_c^0 \times S^0, I, O, R_c \triangleright R, L \rangle$$

dove:

$$\langle (s_c, s), \tau, (s_c', s) \rangle \in (R_c \triangleright R) \text{ se } \langle s_c, \tau, s_c' \rangle \in R_c;$$

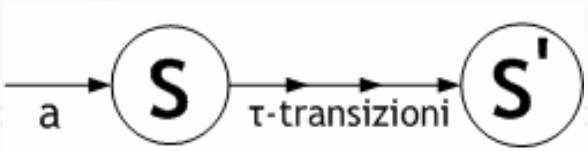
$$\langle (s_c, s), \tau, (s_c, s') \rangle \in (R_c \triangleright R) \text{ se } \langle s, \tau, s' \rangle \in R;$$

$$\langle (s_c, s), a, (s_c', s') \rangle \in (R_c \triangleright R), \text{ con a} \neq \tau, \text{ se}$$

$$\langle s_c, a, s_c' \rangle \in R_c \text{ e } \langle s, a, s' \rangle \in R$$

Sistema controllato (2)

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



- In caso contrario si incorre in una situazione di deadlock
- Definiamo il concetto di τ-chiusura
  - τ-chiusura(s) – l'insieme di stati raggiungibili da s attraverso una sequenza di τ-transizioni
  - τ-chiusura(T) – l'unione delle τ-chiusura(t) per ogni $t \in T$, dove $T \subseteq S$

Verso la piena osservabilità (1)

Per stabilire se $\Sigma_c \triangleright \Sigma_{||}$ soddisfa i requisiti ρ occorre esaminarne le varie configurazioni

- Se ad es. **`ρ=DoReach p`** tutte le esecuzioni di $\Sigma_c \triangleright \Sigma_{||}$ alla fine raggiungono una configurazione in cui vale **`p`**
- $\Sigma_{||}$ è parzialmente osservabile da Σ_c
  - Ad un determinato istante non è possibile conoscere esattamente lo stato dei servizi $\Sigma_{W_1} \ldots \Sigma_{W_n}$
  - Causata da
    - Presenza del non-determinismo
    - Impossibile sapere quando una τ-transizione viene eseguita

Verso la piena osservabilità (2)

In un certo istante di esecuzione abbiamo possiamo trovarci in un insieme “plausibile” di stati

- Rappresentato dalla *belief* del sistema
  - Viene aggiornata quando si eseguono transizioni osservabili dall’esterno (input/output)
  - Se $B \subseteq S$ è la corrente belief, osservando $a \in I \cup O$, essa evolve in $B' = Evolve(B,a)$ dove:

$$Evolve(B,a) = \left\{ s' : \exists s \in \text{τ-chiusura(B)}. \left\langle s,a,s' \right\rangle \in R \right\}$$

- Una proprietà p viene soddisfatta se per ogni $s \in B$, in esso vale p (ovvero $p \in L(s)$)

$$B \models_{\Sigma} p$$

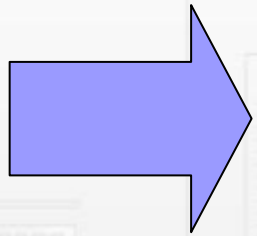
Sistema belief level di uno STS Σ

Indicato con $\Sigma_B = \langle S_B, S_B^0, I, O, R_B, L_B \rangle$

È un particolare STS così definito:

- Gli stati S_B sono le belief raggiungibili da S^0
- Le transizioni R_B descrivono l'evoluzione delle belief
- In L_B troviamo tutte le proprietà p tali che $B \models_\Sigma p$

È ora possibile definire formalmente il problema di composizione



Problema di composizione

A partire dai requisiti ρ e dai servizi componenti $\Sigma_{W_1} \ldots \Sigma_{W_n}$ trovare un controller Σ_c deadlock-free e tale che $\Sigma_B \models \rho$ dove Σ_B è lo STS belief-level associato a $\Sigma_c \triangleright (\Sigma_{W_1} \| \ldots \| \Sigma_{W_n})$

- Per sintetizzare Σ_c viene impostato un problema di pianificazione tramite symbolic model checking
  - Si ragiona nello spazio delle belief (piena osservabilità)
- A partire dal prodotto parallelo $(\Sigma_{W_1} \| \ldots \| \Sigma_{W_n})$ viene ricavato il dominio $\mathcal{D}$ per il planner MBP
- Viene generato un piano (condizionale) π dai cui viene ricavato Σ_c

Corrispondenza composizione/pianificazione (1)

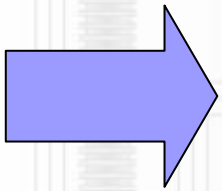
Teorema

- Se π è soluzione per $\mathcal{D}$ e goal ρ, e Σ_π è lo STS relativo a π, allora vale $\Sigma_\pi \triangleright \Sigma_{||} \models \rho$
- Inoltre, se esiste qualche Σ_c tale che $\Sigma_c \triangleright \Sigma_{||} \models \rho$, allora esiste qualche π che è soluzione per il dominio $\mathcal{D}$ e goal ρ (e quindi vale $\Sigma_\pi \triangleright \Sigma_{||} \models \rho$)

Corrispondenza composizione/pianificazione (2)

L'algoritmo specificato è:

- Sound: se esiste una composizione dei servizi (quindi esiste un piano π su $\mathcal{D}$ per il goal ρ) allora viene restituito un controller tale che $\Sigma_\pi \triangleright \Sigma_{||} \models \rho$
- Complete: ogni volta che esso trova l'esistenza della composizione (e quindi un controller che soddisfa il goal) allora quella è la decisione corretta, dato che esiste un piano π su $\mathcal{D}$ che soddisfa ρ



È possibile utilizzare algoritmi di pianificazione per domini pienamente osservabili ed extended goals per effettuare composizione automatica di servizi

Confronto con il Roman Model

- Modellazione del comportamento dei servizi
  - External schema – client
    - Insieme delle azioni atomiche con cui costruire conversazioni
  - Internal schema – sistema
    - Specifica delle responsabilità di esecuzione di ogni azione
  - Rappresentazione con *execution tree*
    - Infiniti e *bisimili* agli STS di partenza, ottenuti per unfolding
    - Indica tutte le possibili esecuzioni del servizio
- Obiettivo della composizione
  - Dato l'external E^{ext} schema del servizio E sintetizzare un internal schema E^{int} che sia composizione di E rispetto alla community di servizi C
    - Una volta sintetizzato esso indica quale servizio esegue una determinata azione

Soluzione della composizione

- Riduzione al problema di soddisfacibilità in DPDL
  - Viene costruita una formula Φ che codifica i servizi esistenti, il target e altre condizioni aggiuntive (polinomiale)
  - Se si trova un modello in cui Φ è soddisfatta allora si ottiene anche l'estensione di $moved_i$
    - Ovvero le istruzioni per coordinare per l'orchestratore
- Teorema - *La formula Φ è soddisfacibile se e solo se esiste la composizione del servizio E rispetto a $E_{1...n}$.*
- Risultati
  - La verifica dell'esistenza della composizione restituisce (se esiste) un internal schema rappresentabile come MFSM
    - Si sfruttano alcune proprietà della logica DPDL
  - Richiede EXPTIME

Conclusioni

Aspetti in comune:

- Operazioni black-box: le operazioni esposte sono descritte senza dettagli implementativi e viste in modo atomico
- Comportamento modellato come transition systems
  - Vengono descritte le possibili conversazioni
- Mapping LAV: servizi mappati utilizzando il linguaggio della community
- Composizione basata sul modello di workflow: processo orchestrato da un coordinatore centrale o mediatore

In particolare ASTRO:

- Più orientato all'orchestrazione che alla sintesi
- Non indicato in contesti molto dinamici, in cui il client influenza particolarmente il flusso di esecuzione
- Gestione di condizioni eccezionali
- Range limitati sui tipi

Riferimenti

- *Bertoli P., Pistore M. and Traverso P. Automated Composition of Web Services by Planning in Asynchronous Domains. In Proc. of ICAPS'05.*
- *Berardi D., Calvanese D.,De Giacomo G., Lenzerini M., and Mecella M. Automated Composition of e-Services that Exports their Behavior. In Proc. of ICSOC'03.*
- *Berardi D., De Giacomo G. and Mecella M. Automatic Service Composition based on Behavioral Descriptions. In Proc. of IJCIS'05.*
- *Berardi D., Calvanese D.,De Giacomo G., Lenzerini M., and Mecella M. Automatic Web Service Composition: Service-tailored vs. Client-tailored Approaches. In Proc. of AISC'06.*
- *Andrews T., Curbera F., Dolakia H., Goland J., Klein J., Leymann F., Liu K., Roller D., Smith D., Thatte S., Trickovic I. e Weeravarana S. Business Process Execution Language for Web Services (ver. 1.1). Microsoft & IBM. 2003.*
- *Berardi D., Calvanese D., De Giacomo G. and Mecella M. Composition of Services with Nondeterministic Observable Behavior. In Proc. of ICSOC'05.*
- *Dal Lago U., Pistore M. and Traverso P. Planning with a Language for Extended Goals. ITC-IRST. 2002.*
- *Pistore M. and Traverso P. Planning as Model Checking for Extended Goals in Non-Deterministic Domains. ITC-IRST. 2001.*
- *Bertoli P. Pistore M. and Roveri M. Planning as Model Checking. http://sra.itc.it/tools/mbp/AIPS02-TUT-We1_slides.pdf. AIPS'02 Tutorial.*
- *Pistore M. , Barbon F., Bertoli P., Shaparau P. and Traverso P. Planning and Monitoring Web Service Composition. In Proc of ICAPS'04.*
- *Berardi D., Calvanese D., De Giacomo G., Lenzerini M. and Mecella M. Synthesis of Underspecified Composite e-Services based on Automated Reasoning. In Proc. of ICSOC'04.*
- *De Giacomo G. Transition Systems and Service Composition. Seminari di Ingegneria del Software 2005/2006. http://www.dis.uniroma1.it/~degiacom/didattica/semingsoft/materiale/3-servizi/2-TS&ComposizioneServizi-2up.pdf.*
- *J. Koehler and B. Srivastava. Web Service Composition – Current Solutions and Open Problems. In Proc. Of ICAPS'03.*
- *D. Burdett, N. Kavantzas and G. Ritzinger. Web Services Choreography Description Language (WS-CDL) 1.0. http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427. W3C Working Draft, 2004.*