Programming multi-agent systems in AgentSpeak using Jason

Communication and interaction (chapter 6)

MARCO PALMA

Come avviene la comunicazione

- Gli agenti si scambiano messaggi
- Jason, rispetto agli altri, interpreta messaggi automaticamente, usando una semantica definita internamente
- Sta a chi programma fare in modo che gli agenti eseguano le azioni appropriate

Java with Jason

Struttura di un messaggio

- La struttura classica di un messaggio è la seguente:
  - (sender, illoc_force, content)

- sender:
  - Nome che identifica chi ha inviato il messaggio all'interno del sistema
- illoc_force:
  - Spesso viene chiamata anche «performative», dice l'intenzione di chi ha inviato il messaggio
- content:
  - È una variabile che varia a seconda della illoc_force

Performative

- I messaggi vengono inviati usando un'azione interna predefinita, la cui struttura generale è la seguente:
  - .send(receiver, illocutionary_force, propositional_content)
- Receiver: non necessariamente unico, può essere anche una lista di agenti che devono ricevere lo stesso messaggio
- Propositional_content: può rappresentare diverse cose; spesso un trigger o un piano
- Illocutionary_force: sono le «performatives» disponibili

Alcune performatives disponibili

- tell: S vuole che R creda che il letterale nel messaggio risulti vero
- achieve: S chiede ad R di provare a raggiungere uno stato dove il letterale nel messaggio risulta vero
- askOne: S vuole sapere se il contenuto del messaggio è vero per R
- askAll: S vuole tutte le risposte di R ad una domanda
- tellHow: S informa R di un piano

- .broadcast(illocutionary_force, propositional_content); azione interna per inviare messaggi broadcast a tutti gli agenti registrati in un sistema

Altre Performatives

- untell: S vuole che R non creda che il letterale nel messaggio è vero
- Unachieve: S chiede ad R di rimuovere un goal per raggiungere uno stato dove il letterale del messaggio risulta vero
- untellHow: S chiede ad R di ignorare un certo piano (ad esempio di rimuoverlo dalla propria libreria)
- askHow: S vuole tutti i piani di R siano rilevanti per il *triggering event* nel contenuto del messaggio

Semantica dei messaggi

- Esempio:
  - S→ .send(r, tell, open(left_door))
  - R→ (s, tell, open(left_door))

  - Il messaggio è recuperato dalla *mailbox* di R e viene passato come argomento di SoccAcc (socially acceptable)
  - Il metodo viene sovrascritto per ogni agente in modo da specificare chi altri può inviare questo tipo di messaggio
    - Case false: il messaggio è semplicemente scartato
    - Case true: il messaggio viene processato da R
- S può inviare diversi tipi di messaggi: scambio di informazioni, delegazione di *goal*, ricerca di informazioni e *know-how related*

Scambio di informazioni

- .send(r, tell, open(left_door))
  - Cosa accade quando si invia questo messaggio?
    - open(left_door) sarà aggiunta alla *belief* di R
    - R crederà che left_door è aperta in quanto gli è stato detto da S (tell)
    - Alla *belief* sarà aggiunta, precisamente, open(left_door) [source(s)]

Cycle	*s* actions	*r* belief base	*r* events
1	.send(r,*tell*, open(left_door))		
2		open(left_door) [source(s)]	+open(left_door) [source(s)]
3	.send(r,*untell*, open(left_door))	open(left_door) [source(s)]	
4			-open(left_door) [source(s)]

- Altre forme di messaggi:
  - .send([r1,r2], tell, open(left_door))
  - .send(r, tell, [open(left_door), open(right_door)])
  - .send(r, untell, open(left_door))
    - Qui open(left_door) sarà tolta dalla *belief* di R

- Se nella *belief* di R c'è open(left_door) [source(t), source(s)], soltanto source(s) sarà tolto dalla sua *belief*, rimarrà comunque open(left_door)[source(t)]

Delegare *goal*

- .send(r, achieve, open(left_door))
  - Viene creato l'evento <+!open(left_door)[source(s)], T> e viene aggiunto all'insieme di eventi di R
  - In particolare, se l'agente ha dei «piani» rilevanti per raggiungere lo stato dove «the door is open», e tra questi ce n'è uno applicabile quando l'evento è selezionato, allora quel piano diventa una *intention*

Cycle	*s* actions	*r* intentions	*r* events
1	.send(r, *achieve*, open(left_door))		
2			+!open(left_door) [source(s)]
3		!open(left_door) [source(s)]	
4	.send(r, *unachieve*, open(left_door))	!open(left_door) [source(s)]	
5			

- .send(r, unachieve, open(left_door))
  - In questo caso, non vengono generati piani, ma il goal viene solo rimosso, diventando subito una *intention*

Delegare goal

- In particolare, nel caso di un *unachieve goal*, saranno rimosse tutte le istanze di quel *goal*
- Oppure, in alternativa, vengono rimosse le istanze delegate da chi invia stesso.
- Jason, a tal proposito, utilizza delle annotazioni, chiamate «source annotation»

Ricerca di informazioni

- .send(r, askOne, open(Door), Reply)
  - S aspetterà finché R non manda un messaggio di risposta; ad esempio un messaggio con content: open(left_door), se questa si trova nella *belief* di R; se non c'è l'agente può tentare di eseguire un *goal* per ottenerla tramite un suo piano
  - Se R trova il messaggio, questo viene messo in Reply e inviato a S, il quale aspetterà la risposta e solo dopo eseguirà l'azione relativa
  - In particolare, la variabile Door non viene instanziata, Reply si
    - Se Reply= open(door) allora verrebbe instanziata (ridondante)
  - Se R non riesce a trovare una risposta al messaggio nella sua *belief*, ad s viene inviato Reply=false
  - Reply sarà «instanziata» a seconda del tipo di risposta o come singola risposta o una lista di risposte, a seconda della performative

Ricerca di informazioni

- La send può anche fallire:
  - Se il messaggio inviato è: .send(r, askOne, open(Door), false), allora la send fallisce nel caso in cui R creda che la porta sia aperta
  - Risponderà, infatti, con open(door) e questo non va bene per Jason, avendo inserito *false* nel messaggio di invio
  - Chi riceve usa un test goal per controllare se nella sua belief c'è il content del messaggio, quindi viene generato un evento +?open(door) e se possiede un piano rilevante e applicabile questo sarà eseguito e poi inviato all'agente che lo ha chiesto

Ricerca di informazioni

- Gli ask messages sono sincroni
- Un agente potrebbe diventare inattivo mentre aspetta la risposta (dipende dalle *intentions* che ha). Se ne ha alcune che non dipendono dalla risposta, continua ad eseguirle
- Se dovesse diventare inattivo, tecnicamente non è un problema. L'unico problema potrebbe essere che l'agente non porti a termine una delle sue *intentions*
- Per risolvere questo problema, gli *ask messages* possono avere un quinto parametro nella *send*, ovvero il *timeout* in millisecondi
- L'agente, quindi, aspetterà questo *timeout*. Se si va oltre, la variabile Reply sarà instanziata con valore *timeout*, così il programmatore sa che non c'è stata risposta

Ricerca di informazioni

r belief base
open(left_door)
open(right_door)

Cycle	*s* actions / unifier / events	*r* actions
1	.send(r, *askOne*, open(Door), Reply)	
2		.send(s, tell, open(left_door))
3	Reply ↦ open(left_door)	
4	.send(r, *askOne*, closed(Door), Reply)	
5		.send(s, tell, false)
6	Reply ↦ false	
7	.send(r, *askOne*, open(Door))	
8		.send(s, tell, open(left_door))
9	+open(left_door)[source(s)]	
10	.send(r, *askAll*, open(D), Reply)	
11		.send(s, tell, [open(left_door), open(right_door)])
12	Reply ↦ [open(left_door), open(right_door)]	
13	.send(r, *askAll*, open(D))	
14		.send(s, tell, [open(left_door), open(right_door)])
15	+open(left_door)[source(s)] +open(right_door)[source(s)]	

Know-how related

- .send(r, tellHow, «@pOD +!open(door) : not locked(door) <- turn_handle(door); push(door); ?open(door)»)
  - La stringa nel messaggio sarà trasformata in un *AgentSpeak plan* e aggiunta alla libreria di r
  - La stringa può essere ottenuta dalla libreria dei piani, tramite un'azione interna *.plan_label*
    - *.plan_label* unisce il piano con una stringa (id) che identifica il piano stesso
- .send(r, tellHow, [«+!open(door): locked(door) <- unlock(door); !!open(door)», «+!open(door): not locked(door) <- turn_handle(door); push(door); ?open(door)»])
  - E' possibile passare più piani, come in questo caso, ognuno di essi sarà aggiunto alla libreria di r

AgentSpeak plans for handling communication

- Jason include piani alla fine delle varie librerie, quando un agente inizia la sua esecuzione.
- Esistono diversi piani per le varie *performatives* e in generale le diverse situazioni
- Questo approccio di interpretare messaggi usando piani permette in qualche modo di dare priorità alle richieste di comunicazione. Si sfruttano infatti le funzioni di selezione di eventi/intentions
- Essendo quindi questi piani presenti nelle librerie, gli utenti faranno un override di queste funzioni

FINE