TuCSoN4JADE

di Sangiorgi Luca
n. 681281

Abstract. Lo scopo di questo elaborato è di descrivere come è stato progettato il modello di integrazione dei due principali approcci di coordinazione di MAS, l'approccio subjective coordination utilizzato dalla tecnologia ad agenti JADE e l'approccio objective coordination utilizzato dalla tecnologia ad agenti TuCSoN. In particolare si mostra come integrare JADE behaviour model con TuCSoN coordination model.

1. Introduzione

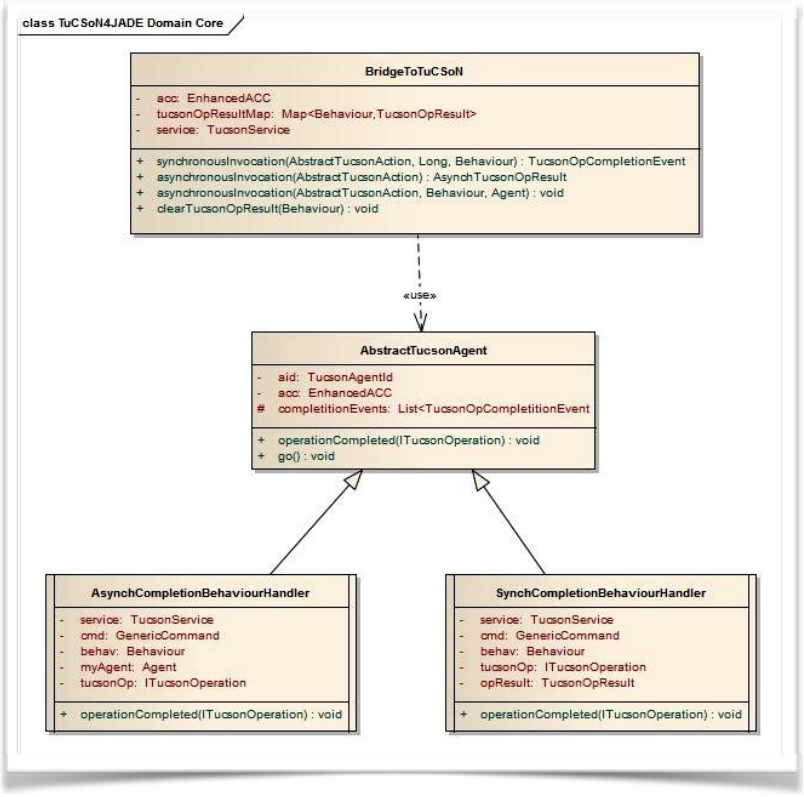
La realizzazione di un modello di integrazione tra approcci *subjective coordiation* e *objectvie coordination* è sorto dal fatto che questi due approcci sono essenzialmente complementari tra loro. Nel primo, il concetto di coordinazione è visto come un'azione individuale svolta da un'entità per raggiungere un proprio obbiettivo in un contesto multi-component system, nel secondo invece il coordinamento è una sorta di attività normativa svolta da qualche componente ben definito del sistema (diverso dalle entità coordinate) per conto del progettista del sistema.

In questo elaborato si descrive la realizzazione dell'integrazione tra il *behaviour model* di JADE[1] e il *coordination model* di TuCSoN[2], in particolare si è sfruttata l'infrastruttura JADE sulla quale si è integrato il *coordination model* di TuCSoN.

2. Integrazione Tucson4Jade

Data la natura del behaviour model di JADE, dove ogni agente è gestito da un singolo thread, e la natura del coordination model di TuCSoN, con il quale si possono eseguire operazioni di coordinazione in modalità sia sincrona che asincrona [3], è stato progettato un modello di esecuzione di operazioni di coordinazione sospensive in modo tale che queste non implichino il "blocco" del flusso di esecuzione dell'unico thread dell'agente JADE.

Per poter integrare "TuCSoN in JADE" si è, come prima cosa, implementato TuCSoN come servizio di JADE, più precisamente come *kernelservice*[4]. Tramite il *TucsonHelper*, classe che estende la classe *ServiceHelper* di JADE, è possibile effettuare alcune operazioni sul servizio TuCSoN come quella che permette di ottenere il riferimento all'oggetto di classe *BridgeToTucson,* la classe fulcro di questa integrazione TuCSoN4JADE. Questa permette di collegare il "mondo" JADE con il mondo "TuCSoN", infatti gestisce l'invocazione delle operazioni di coordinazione in modalità sia sincrona che asincrona.



Oltre alla possibilità di invocare le operazioni di coordinazione in una delle due modalità disponibili, che saranno descritte in dettaglio in seguito, è disponibile

anche un metodo *clearTucsonResul()* che è stato introdotto per "pulire" le strutture condivise utilizzate riguardante il behaviour passato come parametro. Questo metodo può essere utilizzato dal programmatore come strumento per liberare della memoria una volta che il comportamento che ha invocato delle operazioni di coordinazione in modalità sincrona termina ma anche per poter eseguire operazioni di coordinazione in modalità sincrona dentro dei cicli o loop.
Dal diagramma inoltre si può notare come per eseguire le operazioni di coordinazione, vengono utilizzate le classi *SynchCompletionBehaviourHandler* e *AsynchCompletionBehaviourHandler* che rappresentano gli agenti TuCSoN delegati a gestire ed eseguire le operazioni di coordinazione.

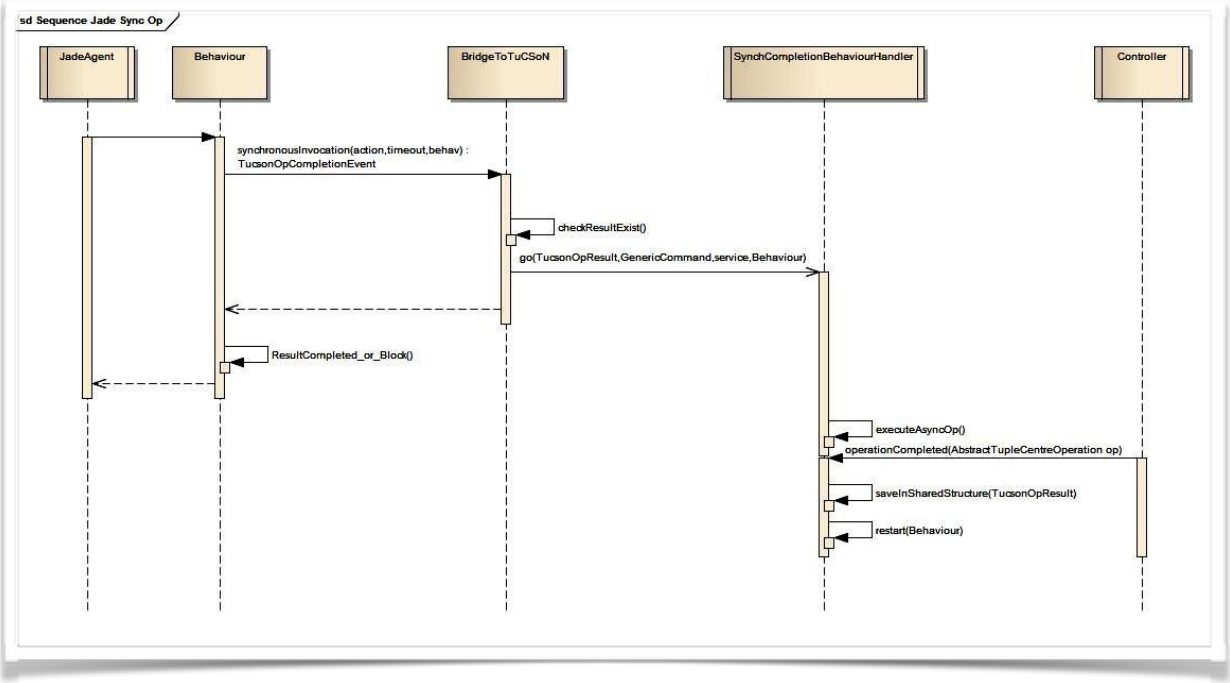
2.1.Operazioni di coordinazione con semantica sospensiva

2.1.1.Modalità sincrona

Il problema principale delle operazioni di coordinazione sospensive in modalità sincrona, quindi bloccanti, in JADE, è che queste oltre a sospendere il comportamento chiamante, bloccano di conseguenza l'intero agente, essendo questo gestito da un unico thread.
Per poter superare questo ostacolo è stato progettato un meccanismo che permette di "sospendere" il behaviour chiamante e "riprenderlo" al momento in cui questa operazione sia stata completata, senza bloccare lo scheduling dei comportamenti dell'agente.
In pratica si utilizza un meccanismo che sospende il comportamento se il risultato non è pronto, mettendo questo behaviour nella *Waiting Queue* tramite il metodo *block()*, per poi riattivarlo, quando il risultato sarà pronto, tramite un altro thread esterno alla struttura Jade con la chiamata *behaviour.restart()*.
Quest'ultima chiamata fa ripartire dall'inizio il comportamento, per cui per utilizzare queste operazioni di coordinazione sospensive si deve adottare la stessa metodologia di programmazione introdotta in JADE, come per esempio l'esecuzione dell'operazione receive() con template, ovvero se l'operazione ha successo si prosegue altrimenti si blocca il comportamento. In questo modo si mantiene la stessa metodologia utilizzata da un programmatore JADE rendendola di immediata comprensione per programmatori del settore.

Per poter recuperare il risultato dell'operazione, dato che questa è eseguita da un differente thread, viene utilizzata una struttura condivisa *TucsonOpResult* contenente il risultato della operazione eseguita che viene memorizzata in una lista contenuta in una HashMap, con chiave il comportamento che ha chiamato l'operazione. In questo modo ogni volta che un comportamento esegue un'operazione di coordinazione, viene controllato di "nascosto" dal programmatore se questa è già stata eseguita, magari causa un *restart()* del behaviour, e quindi viene restituito immediatamente il risultato.
Nel caso in cui il risultato non sia presente nella struttura appena citata, il comportamento chiamante verrà sospeso, con la chiamata *block()*, e il thread incaricato esegue l'operazione di coordinazione sospensiva in modalità asincrona. Per fare questa chiamata asincrona si crea un agente TuCSoN della classe *SynchCompletionBehaviourHandler* a cui si delega la gestione della chiamata e del suo risultato, i dettagli del'implementazione della chiamata asincrona si vedranno in seguito, l'unica nota importante da evidenziare è che l'agente TuCSoN tramite l'observer pattern è in grado di sapere quando l'operazione è completata e quindì, dopo aver memorizzato il risultato nella struttura condivisa dedicata, riavvia il behaviour "bloccato", spostando il suo riferimento dalla *Waiting Queue* alla *Ready Queue*.
In questo modo non si blocca l'unico thread che gestisce l'agente JADE e i suoi behaviour.
Inoltre dato che il comportamento sospeso tramite il metodo *block()* si "riattiva" anche nel momento in cui l'agente riceve un qualsiasi messaggio, c'è un controllo aggiuntivo durante l'invocazione dell'operazione di coordinazione, ovvero si controlla se è stato "svegliato" dal thread incaricato o da un messaggio ricevuto dall'agente, nel secondo caso sospende nuovamente il comportamento perchè il risultato non è ancora pronto.

Di seguito è presente il modello del flusso durante l'esecuzione di una operazione con semantica sospensiva in modo sincrono.



La classe *BridgeToTucson* è la più importante dato che fa il collegamento tra il "mondo" JADE e il "mondo" TuCSoN, incaricata di fare tutti i controlli, mapping e gestire le strutture dati condivise.

Come si può notare sono presenti diversi thread attivati per la corretta esecuzione dell'operazione. Il thread della classe "JadeAgent" è quello che si occupa dello scheduling e dell'esecuzione dei comportamenti di Jade. Il thread dell'entità *SynchCompletionBehaviourHandler* è il thread dell'agente TuCSoN che esegue l'operazione di coordinazione richiesta, infine l'entità attiva *Controller* è un thread che viene creato direttamente dalla infrastruttura TuCSoN ed è l'incaricato a ricevere e gestire i risultati delle operazioni di coordinazione dal centro di tuple.

Il “patter” di invocazione delle operazioni di coordinazione sospensive in modalità sincrona che il programmatore dovrebbe seguire è il seguente:

```
// Behaviour action() method body
...
final TucsonOpCompletionEvent res =             // final ACLMessage res =
  bridge.synchronousInvocation(op, null, this); //   myAgent.receive();
if (res != null) { // operation completion ready
    ...
    // do something
    ...
    bridge.clearTucsonOpResult(this);
} else { // operation still pending
    block();
}
...
```

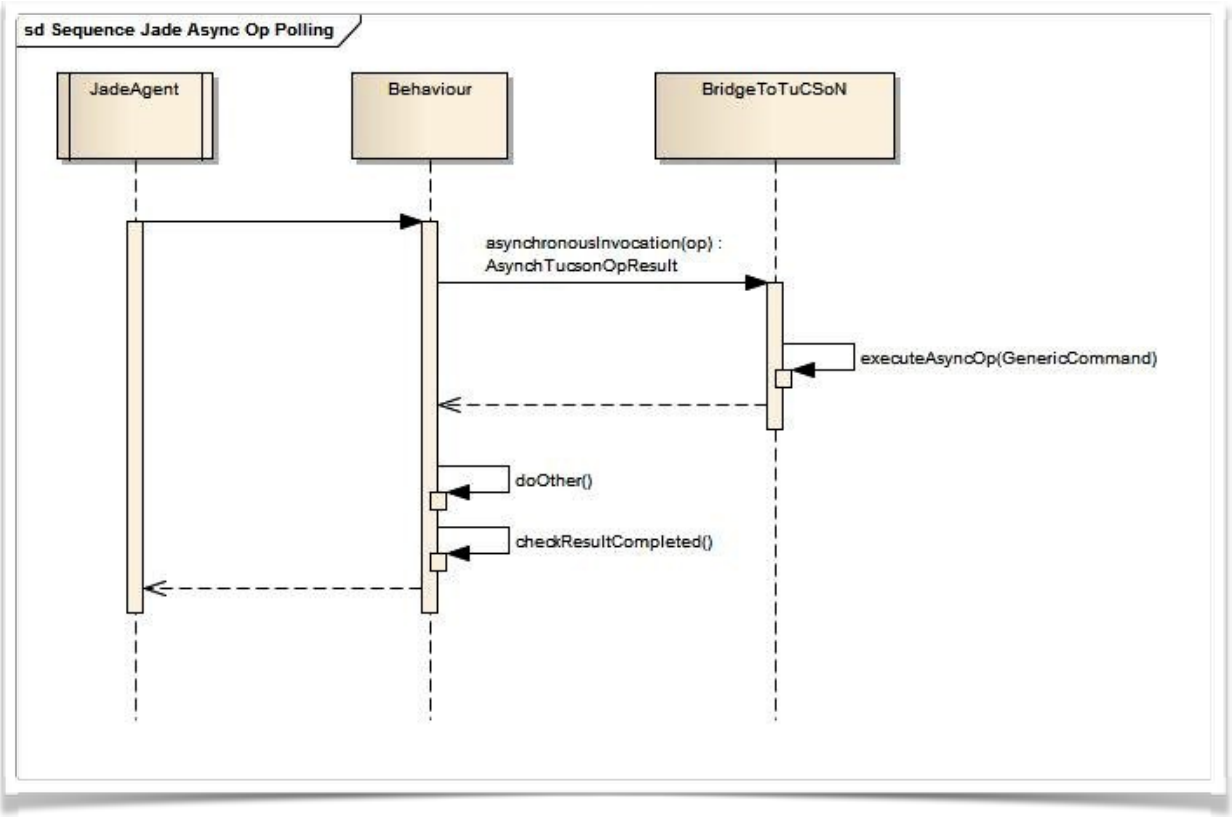
Come si può notare è del tutto identico al patter di ricezione dei messaggi di JADE come indicato nella Jade Programmers Guide [5].

2.1.Modalità asincrona

Per quanto riguarda le operazioni di coordinazione con semantica sospensiva in modalità asincrona, dato che non si blocca il normale flusso di controllo di JADE, come accade invece per la modalità sincrona, non bisogna preoccuparsi di sospendere e riprendere un behaviour ma il problema principale è il recupero del risultato dell’operazione. Per questo si è pensato di adottare due differenti soluzioni.

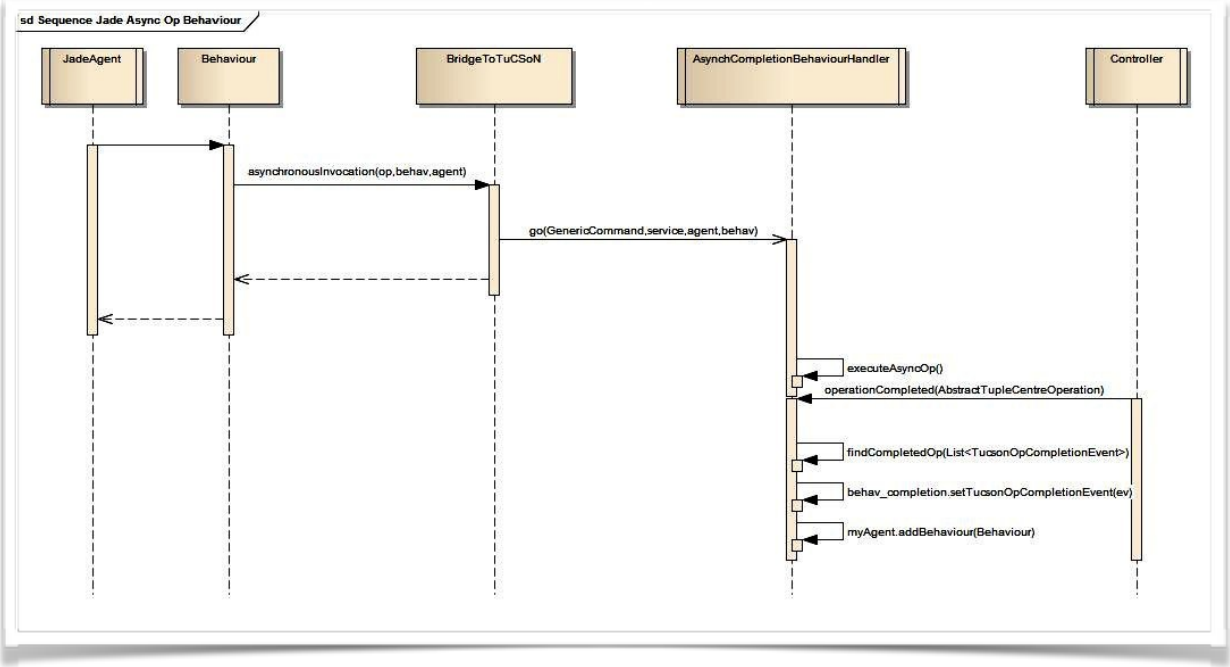
La prima idea si focalizza sul dare la massima libertà al programmatore, ovvero dopo aver eseguito la chiamata dell’operazione asincrona, è il programmatore stesso che controllerà se la risposta è arrivata o meno. Questa funzionalità è stata introdotta grazie alla possibilità di poter “osservare” lo stato della coda delle operazioni completate, infatti è stata introdotta in TuCSoN la funzionalità che permette di ottenere sia la coda delle operazioni completate sia quelle pendenti. Queste strutture devono essere accedute sempre in modo mutuamente esclusivo. In questo modo si danno tutti gli strumenti al programmatore per adottare al meglio il comportamento di “polling”, ovvero “sospendere” quello che si sta facendo per verificare se il risultato è pronto.

Nel diagramma seguente è possibile osservare il flusso di controllo per le operazioni di coordinazione con semantica asincrona nella versione “polling”.



Dal diagramma di flusso è possibile notare come il behaviour subito dopo aver eseguito l’operazione, cedendo il flusso di controllo al *BridgeToTucson*, riottiene subito il controllo dove, dopo aver eseguito del codice non legato all’operazione appena fatta, può controllare se il risultato è pronto o meno e agire di conseguenza.
Per facilitare la programmazione ed evitare errori di accesso a risorse condivise, è stata ideata la classe *AsynchTucsonOpResult* (Java Future-like object) la quale memorizza il riferimento alla coda delle operazioni completate e rende disponibile un metodo che cerca l’operazione tramite il suo id nella coda suddetta in modo mutuamente esclusivo evitando così corse critiche o errori del programmatore.
Questo oggetto di classe *AsynchTucsonOpResult* viene creato dal *BridgeToTucson* e restituito come parametro di ritorno alla chiamata dell’operazione asincrona effettuata dal behaviour.

La seconda soluzione per eseguire operazioni di coordinazione in modalità asincrona si basa sull'idea di dare al programmatore un meccanismo che automaticamente aggiunge un behaviour, predefinito dal programmatore, appena l'operazione è completata.
Per fare questo il programmatore deve definire, prima di eseguire l'operazione, un behaviour che rispetta il contratto *IAsynchCompletionBehaviour*, il quale richiede l'implementazione del metodo *setTucsonOpCompletionEvent()*.
Questo behavior viene poi passato come parametro nella chiamata dell'operazione, e sarà l'*AsynchCompletionBehaviourHandler* (agente TuCSoN) che una volta notificato del completamento dell'operazione controlla se il comportamento rispetta il contratto suddetto e aggiunge il comportamento all'agente, anche questo passato durante la chiamata dell'operazione.
I passaggi principali del flusso di controllo sono visibili nel diagramma di flusso seguente.



Si può notare come i thread presenti sono, come nella modalità sincrona, il thread incaricato a gestire il ciclo di vita dell'agente Jade, il thread dell'agente TuCSoN e il thread Controller, creato all'interno dell'infrastruttura TuCSoN.

Per quanto riguarda l'agente TuCSoN utilizzato nella modalità sincrona, che effettua l'operazione di coordinazione sospensiva in modalità asincrona, esegue la chiamata indicando come listener (pattern Observer) se stesso, e quindì una volta che l'operazione è completata viene eseguito il metodo *operationCompleted()* dove, come già detto sopra, viene gestito il risultato e risvegliato il behaviour.

3. Conclusioni

Uno dei fattori di forza di questo elaborato è la realizzazione di un modello di integrazione tra due approcci di coordinazione differenti, subjective e objective coordination, che oltre al processo di integrazione in sè, quindì la possibilità di utilizzare o uno o l'altro approccio nella stessa infrastruttura, si è focalizzato sulla possibilità di utilizzare entrambi gli approcci contemporaneamente preservando l'autonomia degli agenti coordinanti.

Riferimenti

[1] Bellifemine, F.L., Poggi, A., Rimassa, G.: JADE–a FIPA-compliant agent framework. In: 4th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-99), London, UK, The Practical Application Company Ltd. (19–21 April 1999) 97–108
[2] Omicini, A., Zambonelli, F.: Coordination for Internet application development. Autonomous Agents and Multi-Agent Systems 2(3) (September 1999) 251–269
[3] Gelernter, D.: Generative communication in Linda. ACM Transactions on Pro-
gramming Languages and Systems 7(1) (January 1985) 80–112
[4] Dellarocca N.: Coordination as a service in JADE
http://apice.unibo.it/xwiki/bin/view/Theses/DellaroccaJadecoordination2012
[5] Bellifemine, F., Caire, G., Trucco, T., Rimassa, G.: Jade programmer's guide. Jade version 3 (2002)