Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Nuove tecniche per il testing dei sistemi ad agenti

M. Cossentino

Rapporto Tecnico N.:16
RT-ICAR-PA-05-16 **dicembre 2005**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni(ICAR)
– Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sede di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sede di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Nuove tecniche per il testing dei sistemi ad agenti

M. Cossentino[1]

Rapporto Tecnico N.:16 ***Data:***
RT-ICAR-PA-05-16 **dicembre 2005**

---

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Palermo Viale delle Scienze edificio 11 90128 Palermo
[2] Università degli Studi di Palermo Dipartimento di Ingegneria Informatica Viale delle Scienze 90128 Palermo

Indice

1. Introduzione

In questo lavoro si descriverà l'attività di ricerca svolta dall'ing. M. Cossentino presso il Georgia Tech Institute of Technology (Atlanta, GA) sotto la supervisione del prof. A. Orso. Il tema affrontato è quello del testing dei sistemi ad agenti. In particolare l'obiettivo che ci si è posti è quello di fornire un completamento alla metodologia di progettazione di sistemi multi agente PASSI (ideata nel 2001 dall'ing. Cossentino in collaborazione con il prof. Potts del Georgia Tech) [22][23]. Tale metodologia fin dall'inizio prevedeva un test a livello di singolo agente (paragonabile ad uno unit test) e quindi un test della società di agenti (paragonabile ad un test d'integrazione) ma i dettagli dello svolgimento di questi test non erano stati definiti.

Durante le 3 settimane di permanenza al Georgia Tech si è svolto un approfondito studio bibliografico e si è delineata la soluzione al problema posto.

Nel seguito si chiariranno gli aspetti fondamentali del percorso di ricerca che si è definito e che si sta seguendo. Gli aspetti fondamentali prevedono la soluzione del problema nella sua formulazione iniziale mediante l'utilizzo di approcci provenienti dalla letteratura per quanto riguarda sia il test del singolo agente che quello sociale. Gli approcci selezionati sono riportati nel capitolo 2.

Più interessante dal punto di vista scientifico si è invece considerata la estensione della iniziale formulazione del problema al trattamento del test di società che si formano dinamicamente durante l'esecuzione del sistema. Il paradigma sociale che si è scelto di studiare è quello olonico; di esso si fornirà una ampia descrizione nel capitolo 3.

Il capitolo successivo (4) illustra lo studio preliminare che si è compiuto al fine di identificare i possibili difetti di queste strutture sociali e le strategie di test che potrebbero essere messe in atto. Tale attività è comunque ancora in atto in quanto prove sperimentali servono per supportare le iniziali ipotesi e queste sono ancora in fase di sviluppo.

Infine nel capitolo 5 si descrive il piano di lavoro che si è previsto di svolgere e che è ben più ampio di quanto fosse possibile svolgere durante il soggiorno ad Atlanta.

2. Stato dell'arte

Una analisi completa dello stato dell'arte del testing è al di fuori degli scopi di questo lavoro e quindi per ulteriori approfondimenti si rimanda ad articoli quali ad esempio [16] oppure gli articoli relativi al progetto "Testing UML Design" [18]. Nel seguito si riporteranno soltanto due approcci che si considerano particolarmente significativi per la loro possibile applicazione nel problema in esame. Il primo approccio [17] illustra un metodo che si ritiene

utile per il testing funzionale (a partire dai casi d'uso e diagrammi a stati) e si basa sulla generazione di sistemi eseguibili da modelli UML; il secondo [19] invece esplora il testing di specifiche basate su diagrammi di stato con particolare attenzione per l'*integration testing*.

2.1 Testing di sistemi eseguibili generati da modelli UML

Il metodo che si illustrerà [17] è relativo al testing di forme eseguibili dei modelli UML ed incorpora l'uso di un criterio di adeguatezza del test che è basato sugli elementi del modello UML in forma di diagrammi delle classi e delle interazioni. I diagrammi delle classi sono usati per determinare la configurazione degli oggetti su cui i test sono condotti, mentre i diagrammi delle interazioni sono usati per determinare la sequenza di messaggi che devono essere testati.

L'efficacia di un test è basata sulla capacità di coprire ed eseguire i comportamenti modellati. Analogamente al test eseguito sui programmi, in cui i criteri basati sulla copertura dei blocchi costruttivi sono usati per determinare l'adeguatezza del test, tali criteri possono essere definiti basandosi sulla copertura degli elementi del progetto UML. I blocchi costruttivi fondamentali considerati, sono elementi appartenenti ai diagrammi strutturali e statici (diagramma delle classi) e ai diagrammi comportamentali (diagrammi di stato e di interazione) entrambi dedotti da una rappresentazione funzionale del sistema.

2.1.1 Contesto di riferimento

Il test delle forme eseguibili dei modelli, è analogo al test di programmi e riguarda: la creazione di test case, l'esecuzione dell'artefatto usando i test case, e l'analisi dei risultati del test per determinare la correttezza del comportamento testato. L'adeguatezza del test case è misurata mediante criteri che definiscono le proprietà da coprire. I criteri sono basati sui blocchi costruttivi del software che è sotto test. Per esempio, le espressioni e le ramificazioni sono blocchi costruttivi per il codice. Gli attributi delle classi e le associazioni sono blocchi costruttivi per i progetti che riguardano modelli orientati agli oggetti.

I criteri di test, aiutano a definire gli obiettivi del test da perseguire. Le considerazioni sui costi e sulle risorse disponibili spesso, determinano la scelta di un criterio, rispetto ad un altro. Tali criteri, possono anche essere utilizzati al fine di stabilire quando il test può considerarsi concluso: il test può essere fermato quando i test che soddisfano tutti i criteri stabiliti hanno portato ad un successo.

L’approccio per definire un criterio di test per i modelli in UML è basato sulla divisione in categorie e il partizionamento (*category-partition testing*), approccio sviluppato per il codice. Tale approccio, consiste in:

1. identificare indipendentemente le unità funzionali (*functional unit*) testabili;
2. raggruppare in categorie gli *input* per ciascuna unità funzionale;
3. dividere le categorie di input in classi equivalenti.

Offutt e Irvine [20] hanno dimostrato che la tecnica di categorizzazione e partizionamento è in grado di rilevare difetti che riguardano funzioni implicite, ereditarietà, inizializzazioni e incapsulazioni quando essa è applicata al software OO.

Il test di un modello del sistema riguarda l’esecuzione del comportamento modellato, partendo da una configurazione specifica (object structure) ed usando una sequenza di segnali che danno inizio a tale comportamento. Durante l’esecuzione, la configurazione di partenza può cambiare in quanto possono essere stati aggiunti o eliminati oggetti e link, oppure in quanto sono stati cambiati dei valori negli attributi degli oggetti.

Il passo iniziale consiste nella modellazione delle funzionalità del sistema in termini di casi d’uso; da questi ultimi si deducono: 1) la definizione della struttura del sistema (diagrammi delle classi) e 2) il comportamento dinamico (diagrammi di interazione).

2.1.2 Definizione degli ingressi da testare

Per testare un progetto, bisogna prima creare un test set, composto da diversi test case. In questo approccio, il test case è una tupla che presenta la seguente forma:

<< *sequenza_dei_segnali, configurazione_di_partenza, prefix* >>

Una configurazione è una struttura di oggetti che soddisfa i vincoli espressi nel diagramma delle classi. Essa include: (1) gli oggetti delle classi e (2) il valore di ciascun attributo in ciascun oggetto della configurazione. La *configurazione_di_partenza* è la configurazione da cui il test parte. Il *prefix*, è una sequenza di segnali che possono essere usati per portare il sistema da una configurazione iniziale verso quella di partenza. Una volta che il sistema è nella scelta configurazione di partenza, una sequenza di segnali è applicata per eseguire il test.

Per determinare se un test ha avuto successo o no, vi è bisogno di un oracolo. In questo approccio, le pre e post condizioni dei casi d’uso associati al comportamento sotto test sono

usate per determinare il successo o il fallimento di un test: se la configurazione di partenza soddisfa le pre-condizioni del caso d'uso, allora alla fine del test, la configurazione finale deve soddisfare le post-condizioni del caso d'uso.

2.1.3 Criteri di adeguatezza del test

L'adeguatezza dell'esecuzione dei test di un modello, può essere espressa in termini della copertura degli elementi del modello stesso. Nel seguito verranno presentai alcuni criteri di test basati sulla copertura degli elementi dei diagrammi delle classi e dei diagrammi di interazione.

2.1.3.1 Criterio DCD (Design Class Diagram)

Il criterio DCD, determina le configurazioni che un test comportamentale deve coprire affinchè esso possa essere definito *adeguato*. Per esempio, un criterio può specificare le configurazioni valide (strutture degli oggetti) che devono essere realizzate durante l'esecuzione di un modello del sistema. Il malfunzionamento (*failure*) nel raggiungere una configurazione valida può essere il risultato di un inadeguato test set o di una inconsistenza nel modello del sistema che impedisce il raggiungimento della configurazione stessa.

Il criterio DCD, è basato su una forma di vincoli presenti nei diagrammi. In un DCD, i vincoli possono essere espressi come molteplicità di associazioni finali, generalizzazioni e dichiarazioni OCL (Object Constraint Language).

E' possibile definire i seguenti tre criteri DCD:

1. **Criterio AEM (Association-end Multiplicity):** specifica quante istanze di una classe alla fine opposta del link di associazione possono essere associate con una singola istanza alla fine dell'associazione. Dato un test set T e il modello del sistema SM, T deve permettere la creazione di ciascuna coppia di molteplicità rappresentativa in SM.
2. **Criterio GN (Generalization Criterion):** definisce l'insieme rappresentativo di tipi di specializzazioni, che devono essere create da una superclasse del DCD durante un test del modello del sistema. Dato un test set T e il modello del sistema SM, T deve permettere la creazione di ogni specializzazione definita in una relazione di generalizzazione.

I test che soddisfano tale criterio sono capaci di rilevare i fault che possono presentarsi a causa della violazione del principio di sostituibilità. In particolare, esso afferma che, gli stati di una istanza di una sottoclasse possono essere usati in qualsiasi luogo in cui è attesa una istanza della sua superclasse. I test che soddisfano il criterio GN possono scoprire le violazioni di sostituibilità testando i comportamenti in cui sono previsti oggetti delle superclassi usando le specializzazioni delle superclassi stesse anziché le loro istanze.

3. **Criterio CA (Class Attribute Criterion):** dato un test set T e il modello di un sistema SM ed una classe C, T deve permettere la creazione di un insieme di combinazioni rappresentative dei valori degli attributi per ciascuna istanza della classe. I valori degli attributi possono restringere il comportamento di un oggetto, ad esempio, particolari valori degli attributi possono fare in modo da restringere il modo in cui un oggetto risponde ai segnali. Così, lo spazio dei valori degli attributi fornisce un'altra possibilità per sviluppare criteri di test. Lo spazio dei valori di un attributo può essere ristretto utilizzando i vincoli presenti nell' OCL. La parte critica del criterio consiste nel definire le combinazioni dei valori rappresentativi degli attributi. Questo può essere fatto utilizzando i tre passi seguenti:

   a) Usando il metodo *category-partition* (divisione in categorie e partizionamento) per creare un insieme di valori rappresentativi per ciascun attributo;
   b) Prendendo il prodotto cartesiano di ciascun valore dell'insieme per creare un insieme aggregato di tuple rappresentative dei valori degli attributi per ciascuna classe;
   c) Identificando insiemi aggregati validi e non.

Due dei criteri (AEM e CA) sono espressi in termini dei valori rappresentativi. Per stabilire l'insieme di tali valori, si utilizza una forma di *category-partition testing* adattato ai diagrammi UML. Usando questo metodo, il dominio dei valori è partizionato in classi equivalenti e un valore di ciascuna classe è selezionato per far parte dell'insieme dei valori rappresentativi.

Le partizioni possono essere determinate usando il partizionamento basato sulle conoscenze del dominio, o il partizionamento di default (valori minimi, massimi e non limitati).

2.1.3.2 Criterio per i diagrammi di interazione

E' possibile identificare quattro tipi di criteri di copertura per i diagrammi di interazione:

1. Criterio Cond (Condition Coverage): alcuni messaggi nel diagramma delle collaborazioni possono essere eseguiti solo se si verificano delle specifiche condizioni. Un insieme di test adeguato, dovrebbe testare tutti i possibili rami basati su una certa condizione. Tale criterio quindi, si applica soltanto alle condizioni indicate sui diagrammi di collaborazione. Un insieme test che soddisfa tale criterio deve includere i test cases che rendono tale condizione vera e falsa.
2. Criterio FP (Full Predicate Coverage): una condizione può essere formata da una o più clausole connesse fra loro da un operatore booleano (and, or, ecc.). Un adeguato test set dovrebbe assicurare che ciascuna clausola, in ogni condizione, assuma sia il valore TRUE che FALSE, mentre tutte le altre clausole nella condizione assumono valori tali che il valore della condizione è uguale a quello della clausola esaminata. Questo assicura che ciascuna clausola in una condizione è testata separatamente.
3. Criterio EML (Each Message on Link): assicura che tutti i messaggi fra due oggetti avvengono durante il test. Infatti, i segnali di ciascun messaggio nel link di connessione fra due oggetti in un diagramma delle collaborazioni, dovrebbero verificarsi almeno una volta in un test di adeguatezza.
4. Criterio AMP (All Message Path): assicura che tutti i percorsi dei messaggi in un diagramma delle collaborazioni siano usati. Il percorso di un messaggio (*message path*) è una sequenza di messaggi.

I criteri DCD e quello basato sul diagramma delle collaborazioni, possono essere usati per identificare gli obiettivi del test. Per esempio, il primo può essere usato per definire un obiettivo di test che stabilisce i percorsi specifici da esercitare. Il criterio *Cond* può essere usato per identificare gli obiettivi di test che stabiliscono i valori per le condizioni specifiche.

2.1.4 Metodo di test

I metodi di test per i progetti UML differiscono a seconda del criterio di test utilizzato. Per illustrare i principi basilari e mettere in rilievo alcuni problemi che bisogna risolvere, si assume di utilizzare: il criterio del diagramma delle classi e il criterio AMP (per il diagramma delle collaborazioni).

Gli obiettivi del test, derivati dal criterio del diagramma delle classi, definiscono un insieme di configurazioni obiettivo S.

Questo insieme può essere diviso in: configurazioni obiettivo che rappresentano le configurazioni iniziali del sistema (S_0), e quelle che necessitano che un prefix sia eseguito (S_1).

Il criterio AMP permette di testare gli obiettivi in forma di un insieme di percorsi $P = \{P_1, \ldots, P_k\}$ nella progettazione del diagramma delle collaborazioni. Questi percorsi rappresentano una sequenza di messaggi. Sia PI la configurazione iniziale del sistema e l'insieme che richiede un prefix sia PP. Il processo di testing candidato si sviluppa come segue:

1. determinare la configurazione obiettivo S_0 e S_1 , a partire obiettivi del test, per il diagramma delle classi;
2. determinare i percorsi PP e PI come sequenza di messaggi dal diagramma delle collaborazioni;
3. applicare $p_i \in PI$, per i = 1,…, k_{PI} a $s_j \in S_0$ se la pre-condizione per p_i viene riscontrata in s_j . E' da notare che i percorsi possono essere applicati a più di un $s_j \in S_0$;
4. determinare se qualche configurazione intermedia durante l'esecuzione di uno di questi test case verifica le pre-condizioni per i percorsi $pp_i \in PP$. In tal caso, usare lo stato iniziale e gli stimoli che raggiungono quello stato come prefix per l'applicazione di questi percorsi pp_i . Questo riduce i percorsi non coperti ad un insieme PP' ;
5. applicare e misurare la copertura: Lasciare Sc come insieme di copertura della configurazione obiettivo. Qualsiasi configurazione obiettivo non coperta deve appartenere a S_1 . Gli stati non coperti sono $S_1' = S_1 \setminus \{ s \in S_1 \mid s \in S_c \}$;
6. determinare i prefix per le configurazioni che incontrano le pre-condizioni per i percorsi non coperti in PP' ;
7. misurare la copertura e ridurre S_1' per mezzo di qualsiasi configurazione coperta nel precedente passo;
8. determinare i prefix e eseguirli per i rimanenti stati in S_1 .

2.2 Test dei componenti e d'integrazione mediante diagrammi di stato

Nell'approccio che si presenterà adesso [19] vengono usati i diagrammi a stati come base per generare black-box test, che gli sviluppatori possono usare per unit test o test di integrazione. Se alcuni componenti non sono stati ancora sviluppati o se il codice sorgente non è disponibile, allora potrebbe essere possibile derivare un diagramma a stati astratto per uno o ciascuno dei sottosistemi del componente.

Il metodo proposto utilizza il diagramma a stati dell'UML poiché esso consente di descrivere facilmente il comportamento dinamico di un componente.

Gli stati e le transizioni definiscono tutti i possibili stati e cambiamenti di stato, che un oggetto può effettuare durante il suo ciclo vita. I cambiamenti di stato avvengono come reazioni ad eventi ricevuti dalle interfacce dell'oggetto. Le azioni corrispondono a chiamate ai metodi interni o esterni.

Poiché l'UML ancor oggi non fornisce un meccanismo adeguato per descrivere la comunicazione fra due componenti, si possono usare dei concetti propri del CSP (Communicating Sequential Processes).

2.2.1 Semantica della comunicazione tra componenti

La semantica di comunicazione che l'approccio adotta rispecchia da vicino il modo in cui i componenti COM/DCOM e CORBA interagiscono nei sistemi attuali. Come è noto, i componenti permettono sia comunicazioni sincrone che asincrone, ma in questo metodo di test gli autori si focalizzano soltanto su quelle sincrone.

Si distinguono due tipi di comunicazione sincrona: il primo, il modello dell'evento condiviso, dovrebbe diffondere un singolo evento a diversi componenti, ognuno dei quali è in attesa e tutti reagiscono all'evento contemporaneamente. Il secondo tipo di comunicazione, invece, è chiamato punto a punto, poiché un componente può mandare un singolo evento solo ad un altro componente, e solo questi due componenti risultano essere sincronizzati. Il componente che invia l'evento blocca la propria esecuzione, finchè il componente ricevente non avrà ricevuto l'evento.

2.2.2 Etichettatura delle transizioni

Per dimostrare la connessione esplicita fra i componenti ed associare le operazioni sulle interfacce con gli eventi all'interno dei diagrammi a stati, si usano nelle transizioni delle convenzioni usate nel CSP per le operazioni di comunicazione. Innanzitutto deve essere

assegnato un nome alla connessione fra 2 stati comunicanti; tale nome è formato da un prefisso per gli eventi entranti e un suffisso per gli eventi uscenti. Un'etichetta di transizione potrebbe essere così definita:

_timing?timeout ^_txport!data0

Questa etichetta potrebbe essere interpretata come la ricezione di un evento *timeout* dalla connessione *timing* seguito dall'invio di un evento *data0* mandato attraverso la connessione *txport*. Gli eventi di trigger, anche noti come eventi riceventi, sono identificati da un simbolo "?" usato come separatore, mentre gli eventi di invio sono identificati dal simbolo "^" e da un simbolo di separazione "!".

Le transizioni possono contenere molteplici eventi di invio e ricezione. Eventi di ricezione molteplici possono essere specificati all'interno dell'etichetta della transizione separandoli con un simbolo "+". Molteplici eventi di invio invece possono essere specificati separandoli con il simbolo ":".

2.2.3 Definizione del modello di comportamento globale

Si descriveranno adesso quali sono i passi per costruire un modello di comportamento globale, questo includerà anche un meccanismo che permette agli sviluppatori di raggruppare dei componenti in sottosistemi, ottenendo così una riduzione della dimensione del modello globale. Questo al fine di permettere una adeguata scalabilità del metodo.

L'approccio permette agli sviluppatori di specificare il sottosistema di componenti oppure le interfacce (qualora non vi fosse alcuna definizione di sottosistema) da testare; tutti i componenti modellati e le interfacce specificate vengono considerati come parte del modello globale.

2.2.3.1 Diagrammi a stati del modello

I diagrammi a stati verranno considerati come delle macchine a stati finiti, ossia delle macchine che reagiscono agli input in forma di eventi ricevuti e che producono output in forma di eventi spediti. Tali macchine definiscono un grafo diretto con nodi (che rappresentano gli stati) e archi (che rappresentano le transizioni). Esse hanno uno stato iniziale e possibilmente numerosi stati finali; le transizioni di stato vengono descritte da una funzione.

Una macchina a stati finiti per la comunicazione è definita come

$A = (S, M, T, \delta, s0, F)$

dove:

S è il set degli stati, unico per la macchina a stati;

$M \subset S$ sono stati intermedi;

T, composto da lettere, invece tiene conto del nome della connessione, del nome dell'evento e del tipo di transizione. Il tipo della transizione type $\in$ {INT, COMM, SEND, RECEIVE};

$\delta: S \times T \rightarrow S$ è una transizione che esprime la transizione fra gli stati;

$s0 \in S$ è lo stato iniziale;

$F \subset S$ è un set di stati finali.

Lo stato iniziale fornisce un punto di partenza per una descrizione comportamentale, mentre quelli finali esprimono possibili punti di arrivo per l'esecuzione di un componente; lo stato iniziale e gli stati finali sono stati regolari.

T, come abbiamo detto, contiene il nome dell'evento o il nome della connessione; possibili tipi di transizioni possono essere SEND, RECEIVE, le quali in particolare rappresentano eventi esterni mandati o ricevuti da un'interfaccia esterna alla macchina a stati del componente. Tali transizioni inoltre rappresentano i comportamenti esterni del componente.

Un transizione interna invece può essere vista come una *ε-transition* (transizione vuota) di una macchina a stati finiti. Essa non ha un comportamento osservabile, ma rappresenta soltanto operazioni interne arbitrarie. Transizioni di COMMunication sono esempi di transizioni interne che rappresentano l'interazione fra due macchine a stati. Quando componendo macchine a stati, si hanno coppie di transizioni SEND e RECEIVE con uguale connessione e con nomi degli eventi uguali, tali coppie vengono fuse per formare la transizione di comunicazione (COMMunication).

La definizione di una macchina a stati permette transizioni che contengono singole azioni. Ogni azione espressa da una transizione viene interpretata come una azione atomica. Se più azioni sono raggruppate insieme senza la possibilità di interruzione, allora gli stati tra le transizioni possono essere marcati come stati intermedi. Gli stati intermedi ($M \in S$) sono stati introdotti per raggruppare logicamente sottostrutture degli stati e delle transizioni. La semantica degli stati intermedi fornisce un meccanismo di descrizione comportamentale simile a quello dei microstep. Le azioni atomiche vengono separate in molteplici steps consecutivi, i microstep, i quali vengono eseguiti sempre tutti in una volta. Questi microstep sono le transizioni di uscita degli stati intermedi. Questa tecnica permette di convertire il diagramma a stati dell'UML in una rappresentazione interna; il risultato è un set di macchine a stati normalizzate.

2.2.3.1.1 Macchina a stati composta

Una macchina a stati composta può essere vista come il prodotto di più macchine a stati. Essa stessa è una macchina a stati con il comportamento dinamico dei suoi costituenti, e quindi può produrre output se stimolata da eventi. La sua struttura può essere definita come segue:

Siano A = (S1, M1, T1, δ1, s01, sf1) e B = (S2, M2, T2,δ2, s02, sf2) due macchine a stati e sia inoltre $S1 \cap S2 = \varnothing$. La macchina a stati composta C= A#B allora sarà così definita:

A#B = (S', M', T', δ', s0', F')

S' = S1 x S2

$M' \subset (M1 \times S2) \cup (S1 \times M2)$

$T' \subset T12 \cup TCOMM$

T12 = (T1 ∪ T2) \ {SEND, RECEIVE con connessioni tra A e B}

TCOMM = {COMM per mecciare gli eventi da T1 e T2}

δ': S' x T' → S'

dove δ' è generato da δ1 e δ2 con lo schema di composizione della state machine:

$s0' = (s01, s02) \in S'$.

$F' = \{(s1, s2) \in S' \mid s1 \in F1 \wedge s2 \in F2\}$

Per esempio uno stato globale per A#B è definito come una tupla (s_1,s_2) dove s_1 è uno stato di A e s_2 è uno stato di B. I due stati vengono detti stati delle parti (*part states*).

2.2.3.1.2 Metodo di Composizione

L'idea base per comporre due macchine a stati è quella di creare la macchina a stati prodotto applicando regole di moltiplicazione generiche per stati e transizioni, ciò produrrebbe però un sovraccarico, perché verrebbero prodotti molti stati irraggiungibili e tali stati dovrebbero essere rimossi successivamente, ciò si tradurrebbe in un calcolo computazionalmente oneroso.

E' invece possibile applicare una composizione incrementale che riduce l'ordine dell'algoritmo. Si parte dallo stato globale iniziale e ogni stato della macchina virtuale composta viene valutato solo una volta, inoltre gli stati e le transizioni che risultano ridondanti, rispetto ad osservazioni esterne, vengono rimossi. Usando regole euristiche è possibile scoprire le ridondanze e ridurre la dimensione della macchina a stati composta prima di effettuare il passo successivo.

I sottosistemi definiti vengono processati, indipendentemente, in maniera sequenziale, e per ogni sottosistema viene applicato l'algoritmo di composizione. L'ordine dei passi di

composizione determina la complessità e l'ordine di grandezza dell'algoritmo, il caso peggiore che si può presentare è quello di due componenti che non hanno interazioni, in questo caso il numero massimo di stati è dato dal prodotto delle due state machine (macchine a stati). Dunque risulta importante selezionare nel passo successivo di composizione, il componente più qualificato, per cui il requisito minimo per il componente selezionato è quello di avere una interfaccia comune con gli altri componenti. Si tenta quindi di selezionare la state machine con il più alto numero di punti di interazione e relazioni di comunicazione. Una opportuna norma di selezione riguarda il rapporto fra la possibili transizioni di comunicazione sul numero totale delle transizioni, in tal modo il componente con la più alta percentualità fornirà l'interfaccia più vasta.

Per ogni combinazione di transizioni di uscita dei part states (stati delle parti) si compila una tabella di decisione che è usata per calcolare le nuove transizioni per la state machine composta.

Se una nuova transizione produce uno stato globale che non è parte della struttura esistente della state machine composta, allora esso viene aggiunto in una lista di stati non marcati (*unmarked list*). La transizione viene inoltre aggiunta al modello globale. L'algoritmo termina quando non vi sono più stati non contrassegnati, ciò significa che ogni stato globale raggiungibile è stato inserito nel modello ed in seguito processato. Questo metodo è basato sullo schema di composizione presentato in [21].

Infine, in questo approccio, si è supposto che l'ordine di esecuzione di tutte le azioni del componente risulti sequenziale. Questo è importante perché si tenta di costruire un modello globale per creare test case che dipendono da particolari flussi di eventi e di azioni, ossia si cerca di creare test case lineari e sequenziali per un dato sottosistema.

L'algoritmo in genere ha una complessità lineare riguardo al numero degli stati.

2.2.4 Generazione ed esecuzione del test

Si mostrerà adesso come questo modello può essere usato come base per la generazione automatica del test e la sua esecuzione durante lo unit e lo integration test.

Dopo aver codificato ogni componente, gli sviluppatori effettuano uno unit test per assicurarsi che ogni componente implementi il modello e sia pronto per essere integrato nel sistema complessivo. Questo tipo di test, svolto sul componente isolato rispetto agli altri, conta pesantemente sul modello e l'implementazione di test stub e test drive.

Dopo aver effettuato lo unit test, viene svolto l'integration test, il quale ci assicura che tutte la componenti si interfaccino ed interagiscano correttamente con gli altri componenti. Questo tipo di testing è comunemente chiamato bottom-up integration testing.

2.2.4.1 Generazione del test

L'approccio proposto dagli autori in [19] mira a generare un test set che assicuri la conformità fra le specifiche del modello e le implementazioni risultanti, inoltre esso assume che l'implementazione si comporti in modo deterministico ed esternamente controllabile, altrimenti i test case generati potrebbero produrre dei risultati non corretti.

2.2.4.1.1 Metodo di partizione per categoria

Una categoria o una partizione è definita specificando tutte le scelte che essa rappresenta. Le scelte possono riferirsi a valori di dati, a riferimenti ad altre categorie o partizioni, o ad una combinazione di queste.

Un modello di test TSL (Test Specification Language) viene creato dal modello di comportamentale globale, mappando gli stati e le transizioni di quest'ultimo con le categorie o le partizioni TSL. Gli stati sono le classi equivalenti e sono perciò rappresentati da partizioni, ogni transizione dallo stato è rappresentata come un scelta della categoria/partizione. Solo le partizioni vengono usate per la definizione delle classi equivalenti.

2.2.4.1.2 Procedura di generazione

Un grafo diretto ricorsivo viene costruito usando una categoria/partizione radice; esso contiene tutti i differenti percorsi relativi alle scelte per pianificare la selezione dei dati. Questo grafo può contenere cicli a seconda delle scelte fatte in fase di definizione ed è equivalente al grafo della macchina a stati globale.

Un test frame, cioè un test case, è una istanza della categoria (o partizione) iniziale, ossia una possibile percorso dalla radice alle foglie raggiungibili dell'albero.

Un'istanza di una categoria o di una partizione è una selezione random di una scelta, dal possibile set di scelte definite per quella categoria/partizione. Nel caso delle categorie, la stessa scelta viene selezionata per ogni istanza di un test frame, e ciò riduce i possibili rami del grafo. Con le partizioni una nuova scelta viene selezionata in maniera random, per ogni nuova istanza.

Il contesto di un test case consiste in tutti i dati associati con i vertici lungo un percorso del grafo.

2.2.4.1.3 Requisiti di copertura

Il linguaggio TSL fornisce due tipi di requisiti di copertura:

1. I requisiti generativi che controllano quali test case sono istanziati. Se non sono stati definiti requisiti di test generativi non vengono creati test frame.

2. I requisiti di vincolo che costringono ad omettere certi test case generati.

Un apposito tool (TDE prodotto dalla Siemens) può essere usato per creare i test case che soddisfano tutti i requisiti di copertura. Le sequenze di ingresso per il sottosistema sono equivalenti ai percorsi all'interno del modello di comportamento globale che rappresenta il sottosistema, a partire dallo stato iniziale. Le transizioni di ricezione stimolano i sottosistemi, mentre le transizioni di invio definiscono le uscite risultanti che possono essere osservate da un tool di esecuzione del test.

Per lo unit test l'obiettivo è che tutte le transizioni all'interno dello Statechart siano attraversate almeno una volta, mentre l'obiettivo degli integration test è che queste transizioni che coinvolgono l'interazione fra i componenti vengano esercitate.

2.2.4.2 Esecuzione del Test

I test case generati possono essere mappati con dei modelli di programmazione COM/CORBA. Un test case consiste in una sequenza di eventi di invio e ricezione come le seguenti:

**SEND _tuser.msg( );*

**RECEIVE _txport.data0( );*

Lo scopo degli eventi di invio è quello di stimolare gli oggetti sotto test, mentre gli eventi di ricezione rappresentano un responso proveniente dall'oggetto sotto test.

L'esecuzione del test case coinvolgendo gli eventi RECEIVE, non solo richiede un confronto dei parametri di uscita e dei valori di ritorno con i valori desiderati, ma valuta anche i modelli degli eventi. Tali modelli specificano quali eventi sono attesi in risposta a particolari stimoli, per far ciò occorre monitorare gli oggetti sink associati con i test cases per vedere se i metodi sink richiesti vengono invocati (vedasi Figura 1).

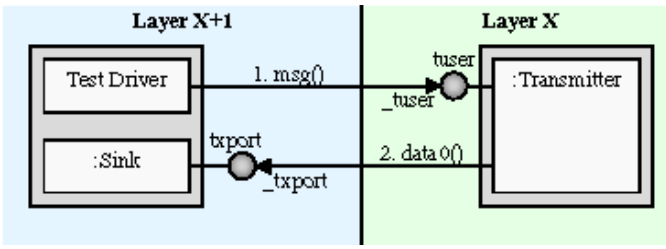

Figura 1. Interazione con l'oggetto sotto test

Completata la presentazione degli approcci che si potrebbero usare per il testing a livello di singolo agente, si passerà nel capitolo successivo ad introdurre i concetti necessari alla trattazione del testing del livello sociale.

3. Agenti e società di agenti

Le due maggiori astrazioni riconducibili ad un sistema multi-agente sono: l'agente singolo e il gruppo [2].

Una delle possibili definizioni di agente è quella fornita da Jennings e Wooldridge [3]: un agente è un modulo software che vive in un ambiente nel quale agisce ed è in grado di effettuare scelte autonome e flessibili allo scopo di perseguire i propri obiettivi. A prescindere dalla definizione, come fa notare Odell, agli agenti si riconoscono due importanti caratteristiche che li differenziano dagli oggetti: l'autonomia e l'interattività. L'agente singolo è autonomo ed è orientato al task che deve portare a termine [4].

Il gruppo si basa invece su concetti quali l'organizzazione, la squadra e le relazioni tra gli agenti; ovvero si basa e trae spunto da nozioni sociologiche.

In questi anni sono stati fatti tanti tentativi per stabilire e attribuire il giusto significato sociologico di organizzazione in riferimento ad un sistema multi-agente.

Secondo Gasser [5], un'organizzazione è un'astrazione che considera gli agenti come una collezione d'attività nella parte di un determinato ruolo. Ciò significa che un'organizzazione esiste realmente in maniera indiretta negli impegni e nelle aspettative dei suoi membri. Questa affermazione non prevede una struttura complessa architetturale, ma un concetto che esemplifica la complessità di un gruppo di agenti con una astrazione [6].

Werner [7], afferma che una struttura sociale può avere regole e norme implicite e codificate, che definiscono gli stati intenzionali degli agenti membri come anche dei ruoli dell'organizzazione. Un gruppo sociale è definito mediante una 5-tupla costituita da: un linguaggio, un gruppo di agenti, una struttura sociale, una distribuzione di ruoli, una definizione di ambienti.

Per So e Durfee [8], "un'organizzazione è un concetto quale un impegno a lungo termine fatto da agenti, in relazione ad un particolare criterio, per gestire in comune un task cooperativo". Sotto quest'aspetto, un'organizzazione ha cinque componenti principali:

- L'insieme di task e sub-task che devono essere portati a termine;
- L'insieme di agenti che partecipano all'organizzazione;
- L'assegnamento dei task e dei sub-task che devono essere portati a termine, agli agenti che partecipano all'organizzazione;

- Una struttura di workflow che stabilisce come i task e i sub-task devono essere distribuiti tra gli agenti e come i risultati finali e parziali devono essere sintetizzati;
- In maniera opzionale, un insieme di risorse distinte dagli agenti e un insieme di vincoli sulla possibilità di utilizzo che hanno gli agenti dell'organizzazione su queste risorse.

La specifica di organizzazioni nei sistemi ad agenti

Il parallelo tra lo studio sociologico delle organizzazioni e lo studio ingegneristico dei sistemi multi-agente risulta opportuno quando si interpretano gli agenti come entità autonome operanti in sistemi organizzati. Per descrivere le norme che regolano le varie forme di organizzazione si proporrà nel paragrafo seguente l'uso della grammatica ADICO, un formalismo linguistico per la descrizione degli aspetti di un'organizzazione e di cui si riporta a seguire una breve introduzione.

3.1.1 Grammatica ADICO

Modellare organizzazioni richiede l'identificazione delle caratteristiche basilari delle stesse e la loro successiva implementazione in forme algoritmiche. La letteratura offre un formalismo che può essere adottato come mezzo di mediazione tra il mondo reale delle società e il mondo dei modelli informatici; questo formalismo prende il nome di grammatica ADICO.

La grammatica ADICO è stata proposta nel 1995 da Crawford e Ostrom [9] per facilitare l'analisi delle organizzazioni e delle cooperazioni nella teoria dei giochi e dei comportamenti. Tali autori inquadrano le organizzazioni come "azioni umane regolari e durevoli nel tempo in situazioni strutturate da ruoli, norme, e strategie condivise, come anche dal mondo fisico".

Secondo questo punto di vista, le caratteristiche e i principi base discriminanti per l'identificazione delle organizzazioni possono essere espressi mediante asserzioni linguistiche, ognuna delle quali può essere attribuita ad una delle categorie menzionate prima: ruoli, norme e strategie condivise. Un'importante proprietà di questo formalismo linguistico è che ogni accordo implicito e tacito può essere espresso utilizzando questo schema.

La grammatica ADICO offre una via per decomporre queste asserzioni linguistiche in cinque componenti: ATTRIBUTES, DEONTIC, AIM, CONDITIONS, e OR ELSE. L'acronimo ADICO sta proprio ad indicare le lettere iniziali di queste componenti. Il significato di ognuno di questi è il seguente:

- ***A***TTRIBUTES: gli attributi contengono tutti i valori della variabile che tiene in considerazione il livello dei partecipanti, e che distingue a chi è applicata l'asserzione (per esempio maggiorenni, di sesso femminile, laureati, prima esperienza, impiegato).
- ***D***EONTIC: questo componente distingue fra i tre verbi modali usati nella logica deontologica: permesso (P), obbligo (O) e divieto (F).
- A***I***M: questo componente descrive azioni o risultati specifici circa le specifiche deontologiche.
- ***C***ONDITIONS: questo componente identifica quelle variabili che definiscono quando (*when*), dove (*where*), come (*how*) e cosa (*what*) limitano un obiettivo (*AIM*).
- ***O***R ELSE: questo componente identifica quelle variabili che definiscono le sanzioni imposte per il mancato rispetto delle specifiche di un determinato ruolo.

Tutte le strategie condivise possono essere modellate come segue:
[ATTRIBUTES] [AIM] [CONDITIONS] (AIC)

Tutte le norme possono essere modellate come segue:
[ATTRIBUTES] [DEONTIC] [AIM] [CONDITIONS] (ADIC)

Tutti i ruoli possono essere modellati come segue:
[ATTRIBUTES] [DEONTIC] [AIM] [CONDITIONS] [OR ELSE] (ADICO)

Da questa descrizione possiamo asserire che le norme sussumono le strategie condivise e sono sussunte dai ruoli.

Crawford e Ostrom mostrano un esempio di come la descrizione linguistica ADICO modella un comportamento cooperativo di un gruppo di persone che è di seguito riportato:

"Tutti gli abitanti del villaggio non devono lasciare che i propri animali calpestino i canali di irrigazione dei campi, altrimenti ai proprietari del bestiame sarà combinata una multa".

A	Tutti gli abitanti del villaggio
D	F
I	Calpestare i canali d'irrigazione tramite il loro bestiame.
C	Sempre

O	Multa

La grammatica ADICO nella modellazione delle strutture organizzative degli agenti

Il primo e più semplice caso di struttura organizzativa è quello in cui gli agenti agiscono individualmente. Questa forma d'interazione, mostrata in Figura 2, prevede che gli agenti fornitori agiscano in maniera autonoma. La delegazione completa delle richieste in entrata verso altri agenti è vietata per escludere problemi di cicli di delega infiniti. Per ragioni di efficienza del sistema è inoltre prevista la delega di parti del servizio richiesto; quindi ad ogni agente è permesso di contrattare solo per il servizio completo che offre e non per parti di questo.

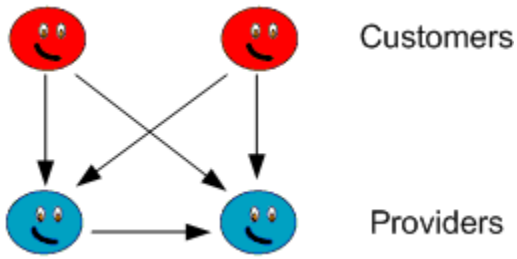

Figura 2. Assegnazione dei task nello scenario di agenti liberi

Considerando questa forma d'interazione secondo la grammatica ADICO si ottiene:

Tutti gli agenti fornitori agiscono da agenti singoli, possono ricevere cfp (call for proposal) da altri fornitori, ma non possono delegare il task ricevuto per intero.

A	Tutti gli agenti fornitori agiscono come agenti singoli
D	Permesso
I	Delegare parte del compito ricevuto da altri agenti mediante attuazione di una singola asta
C	L'ordine delegato non deve contenere tutte le specifiche dell'ordine ricevuto.

Complessità via via crescente presenteranno altre strutture organizzative come la virtual enterprise, la cooperativa, la strategic network fino ad arrivare al gruppo di agenti in cui tutti i

servizi di un agente che diventa membro di un gruppo vengono assimilati dalla organizzazione (vedasi Figura 3).

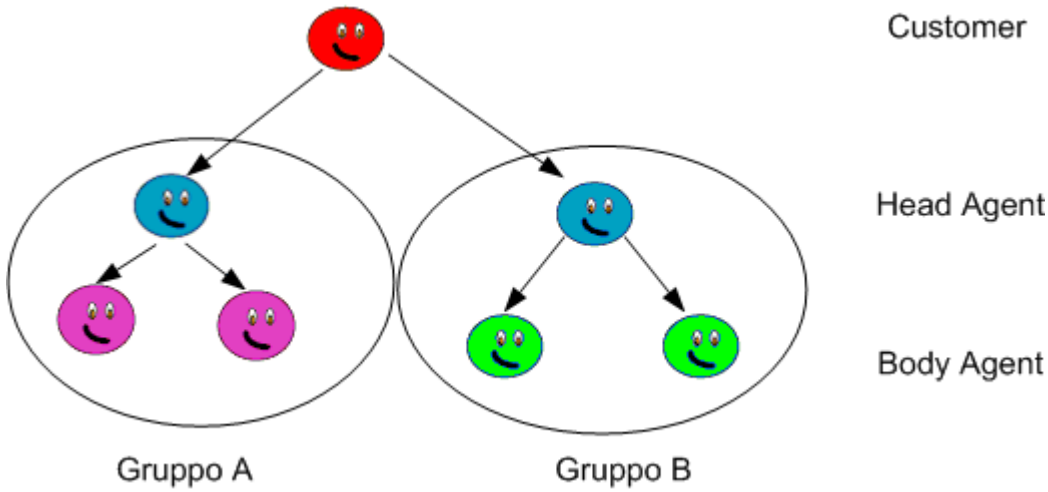

Figura 3. Assegnazione dei task nei gruppi

Ad un agente che è membro di un gruppo non è permesso essere membro di qualche altra organizzazione. Tutte le richieste di servizio sono rimandate ai mittenti con le stesse modalità già menzionate. La comunicazione all'interno del gruppo è fatta tramite authority e il rappresentante dell'organizzazione conosce lo stato dello scheduling delle risorse di ogni membro. La principale differenza con la strategic network sta nel fatto che il protocollo di comunicazione non prevede una fase di notifica; questo aspetto è una logica conseguenza al divieto di appartenenza a più gruppi. Inoltre la distribuzione del profitto avviene secondo dei compensi definiti in fase di creazione del gruppo che non sono mutabili in relazione ai profitti di un determinato ordine.

Specificando questa forma di interazione con la grammatica ADICO, si ottiene:

I membri di un gruppo non possono essere membri di un'altra organizzazione.

A	I membri di un gruppo
D	Vietato
I	essere membri di un'altra organizzazione proprie risorse disponibili

Gli agenti processando un ordine come gruppo nel ruolo di corpi devono rifiutare l'ordine

A	Gli agenti processando un ordine come gruppo nel ruolo di corpi
D	Obbligo
I	Rifiutare l'ordine, inviando l'indirizzo del rappresentante nel messaggio di rifiuto

Gli agenti corpo devono accettare ordini dal loro agente rappresentante

A	Tutti gli agenti nel ruolo di corpo
D	Obbligo
I	accettare ordini dal loro agente rappresentante

Una variazione del gruppo sarebbe la corporazione in cui tutte le risorse degli agenti membri vengono assorbite da uno che agisce da testa. Dopo questa fase, gli agenti membri, anche se presenti, non sono più visibili dal sistema; l'agente testa agisce come un singolo agente.

3.2 Organizzazioni dinamiche

Secondo Parunak [10][11], l'ultimo stadio dell'evoluzione del paradigma ad agenti è che "i progettisti del software identificano gli agenti necessari ai fini dell'applicazione finale ed essi stessi si organizzano per portare a termine le funzionalità preventivate" .

Il termine self-organization è stato introdotto nel 1947 da Ross Ashby, esperto nel campo della psichiatria e della cibernetica; tuttavia ancora oggi non esiste una definizione ufficialmente accettata dal mondo scientifico di cosa sia un sistema auto-organizzante. Una delle ragioni è dovuta sicuramente al fatto che la comprensione intuitiva del termine porta a pensare che sia notevolmente difficile formalizzare un sistema con capacità di organizzazione interne non influenzabili da direzioni, manipolazioni e controlli esterni senza cadere in contraddizione. Per esempio, le società d'insetti come le colonie di formiche, sono ampiamente considerate sistemi auto-organizzanti, ma il loro comportamento e la loro struttura sono chiaramente dipendenti da fattori esterni; le colonie si adattano all'ambiente dove vivono per massimizzare le proprie chances di sopravvivenza.

Una definizione che risulta appropriata per sistemi come questi, è espressa in termini di obiettivi e performances da Klir [12]:

“Un sistema auto-organizzante è un sistema che perfeziona le sue prestazioni nel corso del tempo componendo i suoi migliori elementi per realizzare i propri obiettivi”.

Una caratteristica che è comunemente associata ai sistemi auto-organizzanti è la condizione emergente delle sue capacità. Con questo termine si intende la formazione di modelli coerenti che emergono dall’interazione fra i componenti di un sistema. Questi modelli possono essere abbastanza complessi e vantaggiosi per il sistema, anche nei casi in cui ogni singolo componente gioca un ruolo semplice. Il comportamento dell’intero sistema è percepibile dall’esterno in maniera top-down, mentre in realtà ha origine in modalità bottom-up mediante semplici interazioni. I sistemi di insetti, come le colonie di formiche, sono i prototipi naturali di questi sistemi; ogni singola formica ha un vero e proprio piano, che l’intera colonia sa adattare ad un ampio range di circostanze ambientali.

La caratteristica del comportamento emergente è spesso desiderabile quando devono essere progettati sistemi complessi; il motivo è che risulta molto più facile progettare e trovare soluzioni per semplici unità e creare un sistema di interazione per queste unità piuttosto che agire secondo le classiche metodologie top-down. Altre due proprietà desiderabili in questi sistemi sono l’homeostasis (la capacità del sistema di ritornare ad uno stato stabile dopo un disturbo), e l’homeorhesis (la capacità di un sistema di mettere in atto movimenti di ristrutturazione per adeguare la propria organizzazione ai cambiamenti che incontra durante il proprio sviluppo) [11].

Nel paragrafo successivo si introdurranno un tipo di strutture auto-organizzanti che saranno di particolare interesse per lo studio che ci si propone di fare: gli oloni

Organizzazioni dinamiche e ricorsive: gli oloni

Il termine holon è stato coniato da Artur Koestler, basandolo sulla parola greca “holos” che significa “intero” ed il suffisso “-on” col significato “parte di”. Secondo Koestler un olone è un sistema stabile, composto da diversi sottosistemi detti oloni; esso è allo stesso tempo un intero sistema ed una parte di un sistema più grande [13]. Un esempio biologico è rappresentato dal corpo umano che consiste di organi i quali a loro volta sono composti da tessuti che a loro volta sono composti da cellule che possono essere ulteriormente decomposte. Inoltre, l'essere umano fa parte di una famiglia ed una società di uomini. Nessuno di questi elementi può essere considerato e compreso integralmente senza considerare i suoi componenti o senza considerare la struttura di cui è esso stesso un componente.

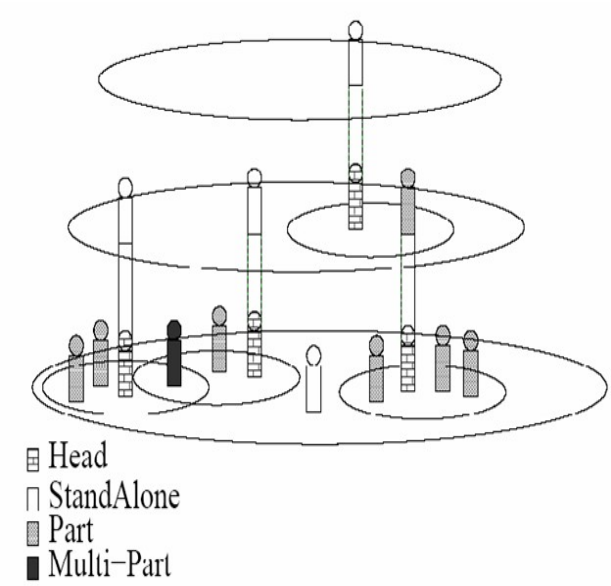

Figura 4. Gerarchia di oloni o olarchia

Come mostrato in Figura 4, ogni struttura olonica o olarchia ha un mediatore (o *Head*) che rappresentata l'organizzazione all'esterno come un'unica entità. La scelta del mediatore può essere realizzata selezionando uno dei membri già esistenti nel sistema come rappresentante dell'olone avvalendosi per esempio, di una procedura basata sul criterio dell'elezione del rappresentante (*voting*). E' anche possibile creare esplicitamente entità nuove per rappresentare l'olone durante la sua vita. In ambo i casi, i rappresentanti (e soltanto questi) manifestano gli obiettivi comuni dell'olone e negoziano secondo questi obiettivi sia con le altre entità presenti nell'ambiente (*StandAlone*) e sia con gli stessi membri dell'olone (*Part*). E' anche prevista la possibilità che un'entità sia parte di più organizzazioni (*Multi-Part*). I rappresentanti sono chiamati testa o mediatore dell'olone, gli altri partecipanti costituiscono corpo.

La forza vincolante che tiene insieme la testa e il corpo dell'olone può essere considerata come una forza emergente dall'insieme di impegni, *commitments*, stabiliti tra di essi.

I due ruoli dell'olone richiedono il bilancio di due forze contrastanti: la forza di coesione contro quella di disunione. Questo bilancio è riscontrabile ad ogni livello risolutivo del sistema olonico, in quanto tale struttura prevede una reiterazione dell'organizzazione distribuita nei vari livelli. Gli oloni interagiscono secondo un preciso ordine gerarchico. Per esempio, in materia amministrativa, due impiegati di dipartimenti diversi comunicano fra loro attraverso i rispettivi responsabili.

Le proprietà presentate dagli oloni sono la ragione per cui oggi questo termine è riscontrabile in diversi campi della tecnologia e dell'economia.

3.2.1 MAS e Oloni

Un sistema multi-agente (MAS) è formato da un insieme di agenti che in relazione alle proprie azioni e alla propria percezione del dominio si definiscono autonomi. Lo sviluppo di sistemi multi-agente robusti e scalabili richiede la progettazione di agenti autonomi che siano in grado di completare i loro obiettivi in un ambiente incerto e dinamico. Per tale motivo, questi agenti devono essere capaci di intraprendere interazioni sociali di alto livello e operare all'interno di strutture flessibili e organizzate. Questo tipo di strutture presuppone la coordinazione anticipata, che ha come conseguenza più rilevante la limitazione degli agenti nei comportamenti di comunicazione. Gli agenti che agiscono in tali strutture incapsulano la complessità dei sottosistemi di cui sono formati semplificando la rappresentazione e la progettazione.

La gerarchia olonica, applicata ai sistemi multi-agente offre la terminologia e la teoria necessarie per la realizzazione di interessanti organizzazioni dinamiche di agenti [14]. Al paradigma della programmazione ad agenti sono aggiunti concetti quali la ricorsione e la modularità, peculiari della definizione di olone. In un sistema olonico, un agente può mostrarsi come una sola entità al mondo esterno ed essere composto da molti agenti ed al contrario, molti agenti possono decidere di congiungersi in una struttura organizzata per formare un super-agente e agire nel sistema come una singola entità. Per la realizzazione di un sistema di questo tipo, è necessaria la definizione delle relazioni *intra-holonic* (*inter-organization*) e *inter-holonic* (*intra-organization*).

In un sistema olonico un'entità centrale accetta un task, lo suddivide in vari sub-task che distribuisce agli agenti di cui è formato. Questi agenti, che possono essere interpretati come *problem-solvers*, producono soluzioni per i sub-task che hanno ricevuto e inviano le soluzioni al risolutore centrale il quale integra le soluzioni ricevute in una soluzione complessiva per il task originale. Questo approccio è chiaramente molto più strutturato del paradigma della soluzione puramente emergente tipica dei MAS.

La trattazione presentata in questo capitolo ha permesso di definire la struttura sociale che si prenderà come riferimento per lo studio proposto (quella olonica) e la formalizzazione necessaria per rappresentarla (grammatica ADICO). Nel capitolo successivo si introdurranno i difetti che ci si aspetta di poter trovare in tali strutture sociali e un possibile approccio di test.

4. Difetti e test delle organizzazioni sociali

Verranno adesso presentati i diversi aspetti di un MAS che si ritengono rilevanti dal punti di vista del testing delle organizzazioni sociali.

Come già discusso nei capitoli precedenti, una struttura sociale (cioè l'insieme di regole sociali) specifica la posizione che un agente occupa in una organizzazione. In strutture sociali di tipo gerarchico, spesso ad una specifica posizione dell'agente corrispondono precisi privilegi e/o responsabilità.

Ad esempio, il coordinatore di un gruppo di robot può decidere la strategia da seguire per lo svolgimento di un compito collaborativo, gli altri devono seguire la strategia scelta e svolgere il compito loro assegnato.

Le società possono essere composte sia al momento della progettazione (design-time) che dinamicamente durante l'esecuzione del programma (run-time). In questo caso le strutture vengono definite a design-time mediante le regole che le caratterizzano (regole di ingresso, di comportamento e goal comuni). Un interessante lavoro in questa direzione è stato proposto da Zambonelli et al. in [24].

E' importante notare che in quanto seguirà ci si dedicherà allo studio dei problemi di testing 'sociale' degli agenti, tralasciando il testing del comportamento del singolo agente visto come 'asocial problem solver' (definizione proposta da Newell in [25]). Questo perché ci si è esclusivamente rivolti allo studio delle interazioni dell'agente con gli altri, ai servizi che esso offre/richiede, alla conoscenza che ha/acquisisce/scambia (tramite comunicazioni dirette o indirette).

4.1 Guasti oggetto di studio

Nel seguito si presenteranno i due guasti che si sono identificati come possibili nelle strutture sociali di interesse e che saranno il centro del lavoro futuro.

4.1.1 Posizione sociale

Un guasto degno di studio riguarda la appartenenza di un agente alla organizzazione sbagliata ('posizione sociale errata'). Questo guasto si può manifestare a partire da un errore nel modello della struttura sociale (cioè delle regole sociali di cui al già citato articolo di Zambonelli et al. [24]) oppure da una errata interpretazione da parte dell'agente di un corretto modello sociale (cioè delle regole che lo definiscono).

Come conseguenza si ha che l'agente si colloca nel gruppo sociale sbagliato per i suoi obiettivi (i goal che gli sono stati imposti a design time dal progettista o che si è prefisso

durante l'esecuzione); questo lo porterà a dei comportamenti non proficui e/o evidentemente in contrasto con quelli da lui attesi.

4.1.2 Violazione di regole sociali

Nella realizzazione di un MAS si suppone che ogni singolo agente rispetti le regole che governano l'organizzazione cui appartiene. Il caso più banale è quello di un agente che nel corso di un'asta eviterà di effettuare un rilancio per un ammontare di denaro superiore a quello che gli è consentito spendere o di cui dispone per il pagamento.

Quando un agente non rispetta le regole di un'organizzazione, il guasto relativo si definirà 'Violazione delle regole dell'organizzazione'.

Possiamo identificare un caso interessante di violazione delle regole dell'organizzazione: un agente potrebbe violare le regole di un gruppo semplicemente perché esso crede di trovarsi in un altro (ci si riconduce quindi al caso precedente) oppure perché osserva le regole di un altro gruppo.

Ecco un esempio di questa situazione: un agente appartiene al gruppo A ma si comporta come prescritto per un agente del gruppo B. In questo caso l'agente è correttamente programmato per comportarsi in B ma la sua condizione di affilamento ai gruppi è errata e lui si è posizionato in A.

In questo caso i possibili guasti sono due: l'agente di è collocato nel gruppo sbagliato ('posizione sociale errata', ricadiamo nel caso precedente) oppure l'agente si è collocato nel gruppo sociale corretto ma è sbagliato il suo comportamento ('Violazione delle regole dell'organizzazione', il caso che si sta studiando).

4.2 Il testing delle strutture sociali

Il testing della organizzazione sociale di un agente è molto interessante ma difficoltoso. La verifica del corretto comportamento sociale di ogni singolo agente è fondamentale per poter pensare a strutture auto-organizzanti che basino il raggiungimento del loro obiettivo sul comportamento che naturalmente emerge tra gli agenti.

E' però alquanto complesso studiare questi aspetti perché essi sono strettamente legati a stati mentali dell'agente. Il fatto che un agente decida di 'entrare' in una organizzazione o 'creda' di essere collocato in un'altra non è rilevabile immediatamente da una osservazione dell'agente stesso.

Questa considerazione può essere estesa alla osservabilità in generale del corretto comportamento dell'agente: non tutti gli aspetti del comportamento dell'agente sono

osservabili (cioè rilevabili da un monitoraggio delle sue azioni nel mondo in cui esiste). Certamente sono osservabili le manifestazioni sociali ma non lo sono ad esempio le azioni di aggiornamento della sua conoscenza che egli compirà a seguito di alcuni eventi nel mondo esterno (o meglio della sua percezione di essi eventualmente anche tramite relazioni sociali).

I guasti relativi alla organizzazione sociale del sistema saranno quindi correlati a caratteristiche interne di ogni singolo agente come lo stato della sua conoscenza (insieme di *belief*, quello che l'agente crede sia vero quali ad esempio l'appartenere ad una certa organizzazione) , i suoi desideri (definiti in letteratura come *desire*, insieme di goal che l'agente vorrebbe raggiungere ma non persegue con le sue azioni/piani) e intenzioni (*intention*, insieme di goal che l'agente si è prefisso di perseguire).

L'osservazione di questo tipo di guasti può quindi essere solo indiretta (possiamo osservare il comportamento dell'agente e da esso dedurre i suoi stati mentali).

4.3 Il Test delle strutture sociali in PASSI

La metodologia PASSI è stata inizialmente concepita per la progettazione di società di agenti di tipo peer-to-peer. Soltanto in seguito è stata aggiunta la possibilità di modellare sistemi olonici di tipo dinamico. In quest'ultimo approccio, le regole dell'olone sono specificate al momento della progettazione ma l'olone stesso si aggrega durante l'esecuzione.

La natura dinamica dei difetti che potrebbero verificarsi in tale contesto rendono questo studio particolarmente interessante.

Come oracolo per la predizione del corretto comportamento si potrebbe naturalmente usare la specifica in termini di grammatica ADICO dell'olone e della organizzazione sociale

4.4 Completezza del test

Per assicurare la completezza del test sarà necessario provvedere alla verifica del rispetto di tutte le regole sociali per ognuna delle organizzazioni progettate:

Le attività previste dal frammento PASSI per la definizione degli oloni sono:

1. Identificazione dei ruoli dell'olone
2. Determinazione della forma di organizzazione
3. Definizione della forma di organizzazione
4. Definizione del servizio (o dei servizi)
5. Definizione di parametri e soglie
6. Raffinamento diagramma COD

7. Raffinamento task di comunicazione

Nell'attività 2 si scelgono la forma o le forme di organizzazione che gli agenti possono formare in relazioni alle condizioni del dominio. Come detto, la verifica di tutte le organizzazioni definite assicura la copertura completa.

Per ogni organizzazione bisognerà poi verificare che siano rispettate le regole sociali (regole ADICO) in essa definite. Tali regole sono definite nell'attività 3 insieme a: ontologia delle organizzazioni, protocolli di comunicazione e modello dinamico degli agenti coinvolti nell'organizzazione.

La corretta applicazione dei protocolli di comunicazione va anch'essa verificata in quanto possibile fonte di difetti ma essa rientra in già noti approcci di letteratura sulla verifica di sistemi a stati e quindi si tralascerà in questo studio.

5. Piano di lavoro proposto

Il piano che ci si propone di svolgere è articolato in 3 fasi:

1) creazione dei casi di studio da usare per sperimentare l'approccio di test. Saranno necessari più casi di studio per poter testare differenti strutture sociali. Si partirà da sistemi sviluppati con PASSI o da nuovi esempi e si provvederà ad aggiungere nel loro progetto alcune strutture oloniche. Queste saranno l'oggetto del testing.
2) La implementazione dinamica delle strutture oloniche passa per l'utilizzo di appositi protocolli di interazione tra agenti che sarà necessario codificare in quanto non naturalmente supportati dalla piattaforma Jade che verrà utilizzata.
3) Verranno quindi applicati i criteri definiti nei capitoli precedenti per sperimentare la metodologia di test. In particolare essa includerà almeno:
    a. Un test funzionale basato sui casi d'uso; si potrebbe seguire un approccio simile a quello riportato in [15]. Questo tipo di test assicura che ogni agente svolga correttamente le funzionalità che gli sono state affidate. Si prevede anche l'utilizzo di strumenti per il regression testing specificatamente concepiti per l'uso con la piattaforma JADE; a tal proposito è già disponibile il tool che è descritto in [1] (articolo cui si è direttamente contribuito).
    b. Un test delle interazioni tra gli agenti (comunicazioni). Principalmente esso mirerà alla verifica della corretta esecuzione della macchina a stati che descrive il protocollo di interazione adottato dagli agenti. Anche in questo caso si ricorrerà ad approcci provenienti dalla letteratura. Ovviamente anche gli altri

aspetti della comunicazione (ontologia e linguaggio) meritano studio ma al momento verranno tralasciati.

c. Un test delle strutture sociali secondo quanto prima descritto.

Questa combinazione di test assicura che il livello del singolo agente venga coperto dal punto 3.a. Mentre la verifica del livello sociale del sistema multi-agente viene affidato al punto 3.b per quanto riguarda la interazione tra gli agenti (comunicazioni) ed al punto 3.c per la correttezza delle organizzazioni sociali.

Ovviamente altri aspetti potrebbero essere valutati tra questi:

- Errori nell'ontologia che descrive il dominio (i problemi potrebbero derivare da una errata modellazione del dominio, da una errata codifica dell'ontologia, da differenze nelle ontologie riferite da agenti diversi);
- Conoscenza (istanza della ontologia) di ogni agente (nell'istanziare la sua conoscenza in termini ontologici l'agente potrebbe soffrire di malfunzionamenti);
- Aspetti non funzionali (ad esempio la sicurezza oppure i tempi di risposta dei singoli agenti);
- Disponibilità/fruizione (anche in termini di diritti) delle risorse;
- Servizi offerti dagli agenti (sarebbero rilevanti aspetti come la descrizione del servizio oppure la sua effettiva disponibilità);
- Ruoli ricoperti dagli agenti (non necessariamente nell'afferire ad una organizzazione sociale); poiché il ruolo è una astrazione del comportamento dell'agente, errori nei ruoli potrebbero non essere coperti dal test funzionale

Questi aspetti si rimandano ad una fase successiva di lavoro non perché inessenziali ma perché valutazioni empiriche svolte sui sistemi già sviluppati li evidenziano come poco frequenti (ad esempio quelli relativi ad ontologia e conoscenza) oppure di minore interesse scientifico rispetto a quelli studiati in questa prima fase (è il caso dei malfunzionamenti che possono ricadere in studi fatti in ambito object-oriented).

6. Riferimenti bibliografici

[1] Caire G., Cossentino M., Negri A., Poggi A, and Turci P. (2004) Multi-agent Systems Implementation and Testing, In Proc. of From Agent Theory to Agent Implementation - Fourth International Symposium (AT2AI-4), Vienna, Austria.

[2] Jarillo, J. C. – (1988), “On Strategic Networks”, Strategic Management Journal, 9:31 – 41.

[3] Jennings, N.R. - Wooldridge, M. – (2000) “Agent-Oriented Software Engineering", In Bradshaw, J. – Handbook of Agent Technology, Ed. AAAI/MIT Press.

[4] Odell, J. – (2002), “Objects and Agents Compared”, Journal of Object Computing, Vol 1, Number 1, May, 2002.

[5] Gasser, L. – (1991), “Social conceptions of knowledge and action: DAI foundations and open systems semantics”, In Artificial Intelligence, Special Volume Foundations of Artificial Intelligence. Elsevier Science Publishers B. V. Vol. 47 (1-3) pp. 107-138.

[6] Gasser, L. – (1995), “Computational Organization Research”, In Proceedings of the First International Conference on Multi-Agent Systems ICMAS-95, San Francisco, AAAI Press/The MIT Press.

[7] Werner, E. – (1987), “Cooperating Agents: A unified Theory of Communication and Social structure”, In Distributed Artificial Intelligence Vol. II. Huhns, M. & Gasser, L. (Eds). Part I: Societies of Agents. Morgan Kaufmann Publishers Inc.

[8] So, Y. – Durfee, E. – (1996), “Designing Tree-Structured Organizations for Computational Agents”, In Computational and Mathematical Organization Theory 2:3 (1996): 219-246. Kluwer Academic Pub.

[9] Crawford, S.E.S. – Ostrom, E. – (1995), “A grammar of institutions“, American Political Science Review, 89 (3):582-599.

[10] Parunak, H.V.D. – (1995), “Manufacturing experience with the contract net”, In Huhns, M., editor, Distributed Artificial Intelligence, pages 285-310. Pitman, London.

[11] Parunak, H.V.D. – Brueckner, S. – (2002), “Co-x: Defining what agents do together”, AAMAS 2002 Workshop on Teamwork and Coalition Formation.

[12] Klir, G. – (1991), “Facets of Systems Science“, Plenum Press.

[13] Koestler, A. – (1967), “The Ghost in the Machine”, Hutchinson & Co, London, 1967.

[14] Schillo, M. – Fischer, K. – (2003), “Holonic multiagent systems”, Zeitschrift fur Kunstliche Intelligenz, 17 (4), 54–55.

[15] Sudipto Ghosh, Robert France, Conrad Braganza, Nilesh Kawane. Test Adequacy Assessment for UML Design Model Testing. International Symposium on Software Reliability Engineering (ISSRE 2003), Denver, USA, November 17-20, 2003.

[16] B. Hailpern and P. Santhanam. Software debugging, testing, and verification. IBM Systems Journal. Volume 41, Number 1, 2002.

[17] S. Ghosh, R. B. France, C. Braganza, N. Kawane, A. Andrews and O. Pilskalns. Test Adequacy Assessment for UML Design Model Testing, International Symposium on Software Reliability Engineering, ISSRE 2003, Denver, CA,USA, November 17-20, 2003.

[18] Testing UML Design. Online at: http://www.cs.colostate.edu/ ~trungdt/uml_testing/ uml_testing.html?publications#summary

[19] Jean Hartmann, Claudio Imoberdorf, Michael Meisinger. UML-Based integration testing. In Proc. of the 2000 ACM SIGSOFT international symposium on Software testing and analysis. Portland, Oregon, United States. Pp: 60 - 70

[20] A. J. Offutt and A. Irvine. Testing Object-Oriented Software Using the Category-Partition Method. In Proceedings of the 17th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS USA) 1995, pages 293-304, Santa Barbara, California, August 1995.

[21] Sabnani K. K., Lapone Aleta M., Uyar M. Ümit: An Algorithmic Procedure for Checking Safety Properties of Protocols. IEEE Transactions on Communications, Vol. 37, No. 9, Sept. 1989.

[22] M. Cossentino, C. Potts. A CASE tool supported methodology for the design of multi-agent systems. The 2002 International Conference on Software Engineering Research and Practice (SERP'02) - June 24 - 27, 2002 - Las Vegas (NV), USA.

[23] M. Cossentino. From Requirements to Code with the PASSI Methodology. In Agent-Oriented Methodologies, B. Henderson-Sellers and P. Giorgini (Editors). Idea Group Inc., Hershey, PA, USA. 2005.

[24] F. Zambonelli, N. Jennings, M. Wooldridge. Organizational Rules as an Abstraction for the Analysis and Design of Multi-agent Systems. Journal of Knowledge and Software Engineering, Vol. 11, No. 3, 2001.

[25] Newell, A. The knowledge level, Artificial Intelligence, 18 (1982) 87–127.