
ANNO ACCADEMICO 2024/2025

Agenti Intelligenti

Teoria

Altair's Notes



**UNIVERSITÀ
DI TORINO**

DIPARTIMENTO DI INFORMATICA

CAPITOLO 1	INTRODUZIONE	PAGINA 5
1.1	Perché Agenti Intelligenti? Agenti e Sistemi Multiagente — 6 • Approccio Interdisciplinare — 7	5
1.2	Cosa si Intende per Agenti Intelligenti? Test di Turing — 9 • Agenti Intelligenti — 10	8
1.3	Cenni alla Computazione Paradigma Orientato agli Agenti e Architetture per Sistemi ad Agenti — 15	12
CAPITOLO 2	LOGICA PER AGENTI	PAGINA 18
2.1	Richiami di Logica Classica Semantica — 19 • Sistemi Deduttivi e Logica del Primordine — 19	19
2.2	La Logica Modale Inadeguatezza della Logica Classica — 20 • Introduzione alla Logica Modale — 20	20
2.3	Sistemi Modali Accenni di Logica Epistemica — 25 • Assiomi per Knowledge e Belief — 25 • Ragionare sul Tempo — 26 • Ragionare sulle Azioni — 28	25
CAPITOLO 3	AGENTI RAZIONALI E ARCHITETTURE	PAGINA 30
3.1	Credenze, Desideri e Intenzioni Ragionamento su Azioni — 31 • Modellazione Logica di Agenti BDI — 32 • Un'Architettura per agenti BDI — 33 • Riconsiderazione delle Intenzioni — 38	30
3.2	Linguaggi e Architetture per Agenti Procedural Reasoning System — 39 • Agent-Oriented Programming — 41 • Agenti Reattivi — 41 • Architetture Ibride — 43	39
CAPITOLO 4	SISTEMI MULTIAGENTE	PAGINA 46
4.1	Agent Communication Languages Atti Comunicativi — 48 • KQML e FIPA — 48	47
4.2	Semantica Sociale per la Comunicazione Un Buon ACL — 52	51
4.3	Protocolli di Interazione Protocolli a commitment — 54 • Information Protocols — 54 • Modelli di Coordinazione — 57	53
CAPITOLO 5	JADE E JASON	PAGINA 59
5.1	JADE FIPA — 59 • Introduzione a JADE — 60 • Comunicazione tra Agenti — 63 • Protocolli di Interazione — 65	59

5.2 JASON

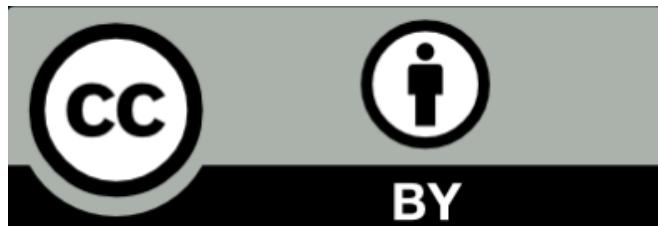
66

Introduzione — 66 • AgentSpeak(L) — 66 • Semantica Operazionale — 68 • JASON: Un'Implementazione di AgentSpeak(L) — 70

Premessa

Licenza

Questi appunti sono rilasciati sotto licenza Creative Commons Attribuzione 4.0 Internazionale (per maggiori informazioni consultare il link: <https://creativecommons.org/licenses/by/4.0/>).



Formato utilizzato

Box di "Concetto sbagliato":

Concetto sbagliato 0.1: Testo del concetto sbagliato

Testo contenente il concetto giusto.

Box di "Corollario":

Corollario 0.0.1 Nome del corollario

Testo del corollario. Per corollario si intende una definizione minore, legata a un'altra definizione.

Box di "Definizione":

Definizione 0.0.1: Nome delle definizioni

Testo della definizione.

Box di "Domanda":

Domanda 0.1

Testo della domanda. Le domande sono spesso utilizzate per far riflettere sulle definizioni o sui concetti.

Box di "Esempio":

Esempio 0.0.1 (Nome dell'esempio)

Testo dell'esempio. Gli esempi sono tratti dalle slides del corso.

Box di "Note":

Note:-

Testo della nota. Le note sono spesso utilizzate per chiarire concetti o per dare informazioni aggiuntive.

Box di "Osservazioni":

Osservazioni 0.0.1

Testo delle osservazioni. Le osservazioni sono spesso utilizzate per chiarire concetti o per dare informazioni aggiuntive. A differenza delle note le osservazioni sono più specifiche.

1

Introduzione

1.1 Perché Agenti Intelligenti?

La storia della computazione è stata definita da 5 trends:

- *Ubiquità.*
- *Interconnessione.*
- *Intelligenza.*
- *Delega.*
- *Human-orientation.*

Ubiquità:

- La continua riduzione dei costi di elaborazione ha permesso di introdurre capacità di elaborazione in luoghi e dispositivi che una volta sarebbero stati antieconomici.
- Ma mano che la capacità di elaborazione si diffonde, elementi sofisticati e intelligenti diventano onnipresenti.

Interconnessione:

- I sistemi informatici, attualmente, non esistono più da soli, ma sono collegati in rete in grandi sistemi distribuiti.
- Per esempio internet.
- Alcuni ricercatori stanno proponendo modelli teorici che ritraggono l'informatica principalmente come un processo di interazione.

Intelligenza e Delega:

- La complessità dei compiti che siamo capaci di automatizzare e delegare ai computer è cresciuta costantemente.
- Viene dato più controllo ai computer:
 - *Fly-by wire aircraft.*
 - *Fly-by-wire cars.*

Human-orientation:

- CI si sposta da una vista orientata alla macchina e alla programmazione verso una visione a metafore e concetti più vicine alla comprensione umana.
- I programmati concettualizzano e realizzano software in termini di livello superiore, più orientato all'uomo, *astrazioni*.

Note:-

Delega e intelligenza implicano la necessità di costruire sistemi informatici in grado di agire efficacemente.

Questo implica:

- La capacità di agire dei sistemi informatici *in modo indipendente*¹.
- La capacità dei sistemi informatici di agire in un modo che *rappresentino nel migliore dei modi i nostri interessi* durante l'interazione con altri esseri umani o sistemi.
- Interconnessione e distribuzione portano a sistemi che possono *cooperare* e *raggiungere accordi, agreements* o *competere* con altri sistemi che hanno interessi diversi.
- Tutto ciò porta alla nascita di un nuovo campo dell'informatica: *sistemi multiagente*.

1.1.1 Agenti e Sistemi Multiagente

Definizione 1.1.1: Agente

Un agente è un sistema di compilazione, un sistema informatico, capace di agire in modo indipendente per conto di un utente o proprietario (capendo cosa deve essere fatto per soddisfare gli obiettivi di progettazione, piuttosto che essere costantemente informato).

Definizione 1.1.2: Sistema Multiagente

Un sistema multiagente (MAS) consiste in una serie di agenti, che interagiscono l'uno con l'altro. Nel caso più generale, gli agenti agiscono per conto di utenti con differenti obiettivi e motivazioni.

Note:-

per interagire con successo, richiedono la capacità di *cooperare, coordinarsi* e *negoziare* tra loro.

Un sistema multiagente risponde a queste domande:

- Come può emergere la cooperazione in società composte da agenti *self-interested*?
- Quali tipi di linguaggi possono essere utilizzati dagli agenti per comunicare?
- Come possono gli agenti *self-interested* riconoscere conflitti? E come possono raggiungere accordi?
- Come possono gli agenti autonomi coordinare le proprie attività in modo da raggiungere cooperativamente gli obiettivi?

Ma soprattutto:

- Come possiamo creare agenti capaci di agire in modo indipendente, autonomo, in modo che possano svolgere con successo i compiti che gli deleghiamo?
- Come possiamo creare agenti capaci di interagire con altri agenti con il fine di portare a termine con successo i compiti delegati, soprattutto quando non si può presumere che gli altri agenti condividano gli stessi interessi e obiettivi?

¹Un oggetto fa qualcosa perché la deve fare, un agente fa qualcosa perché la vuole fare. Hey non avevo idea di essere diventat un oggetto quando mi sono iscritt all'università.

Osservazioni 1.1.1

- Il primo caso è un problema di progettazione/design di un agente.
- Il secondo caso è un problema di progettazione/design di società di agenti.

1.1.2 Approccio Interdisciplinare

Il campo dei sistemi multiagente è influenzato e ispirato da molti altri campi:

- Economia.
- Filosofia.
- Teoria dei giochi.
- Logica.
- Ecologia.
- Scienze sociali.

Note:-

Questo è sia un punto di forza (perché ci sono molte idee diverse) sia un punto di debolezza (perché ci sono molte idee diverse lol).

Domanda 1.1

Non è solo AI?

- Non abbiamo bisogno di risolvere tutti i problemi di intelligenza artificiale per costruire agenti utili.
- L'AI classica ignorava gli aspetti sociali dell'agire.

Domanda 1.2

Non è solo economia/teoria dei giochi?

- Nella misura in cui la teoria dei giochi fornisce descrizione di concetti, non sempre ci dice come calcolare soluzioni.
- Alcune assunzioni in economia/teoria dei giochi potrebbero non essere valide o utili nella costruzione di agenti artificiali.

Domanda 1.3

Non sono solo scienze sociali?

- Possiamo trarre spunti dallo studio di società umane, ma non ci sono particolari motivi per credere che le società artificiali siano costruite allo stesso modo².
- Si trae ispirazione, ma non si assume a prescindere.

²Effettivamente sarebbe raccapriccante se gli agenti cominciassero a uccidersi a vicenda per questioni triviali.

1.2 Cosa si Intende per Agenti Intelligenti?

Ricordiamo che un agente è un sistema di computazione capace di agire *autonomamente* in un qualche *ambiente* con il fine di raggiungere gli *obiettivi* per cui è progettato.

• **Note:-**

La caratteristica principale degli agenti e la loro autonomia.



Figure 1.1: Agente che interagisce con l'ambiente.

• **Note:-**

In figura 1.1 il fatto che l'agente e l'ambiente sono disegnati con due rettangoli identici non è casuale: sono due elementi alla pari.

Esempio 1.2.1 (Agenti)

Un termostato è un agente autonomo

Percepisce la temperatura
↓
Decide cosa fare



Accende/spegne il riscaldamento o noi fa nulla

Un aspirapolvere è un agente autonomo



Autonomia e controllabilità:

- Al di fuori della comunità AI, gli agenti autonomi intelligenti sono percepiti come entità consapevoli di sé, incontrollabili, la cui autonomia emerge come proprietà "extra-programma".
- Un agente può prendere iniziative che non sono incluse fin dall'inizio nel suo programma.

1.2.1 Test di Turing

Il test di Turing:

- Una persona o un computer sono nascosti a un investigatore (fig. 1.2).

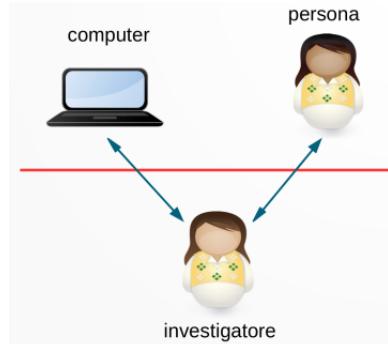


Figure 1.2: Rappresentazione del test di Turing.

- Interazione:

- L’investigatore pone domande scritte.
- L’entità nascosta fornisce risposte scritte.
- L’investigatore deve capire se dall’altra parte c’è un computer o una persona. Se all’altra parte c’è un computer e l’investigatore non riesce a distinguerlo allora il computer si può dire *intelligente*.
- Basta produrre gli output attesi per dire che vi è comprensione? (fig. 1.3)



Figure 1.3: Scambio di messaggi.

Esperimento di J. Searle (la stanza cinese):

- Un computer, programmato per rispondere con certi ideogrammi cinesi ad altri ideogrammi cinesi ricevuti in input (fig. 1.4). L’interlocutore umano che parla in cinese non vede il computer che è chiuso in una stanza. Cosa penserà di chi è nella stanza?

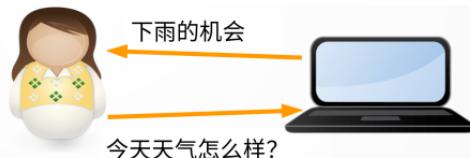


Figure 1.4: Rappresentazione dell’esperimento di Searle.

- Il computer parla cinese? Lo capisce?
- Una persona chiusa in una stanza ha istruzioni per rispondere con certi ideogrammi cinesi in risposta ad altri ideogrammi cinesi (fig. 1.5).

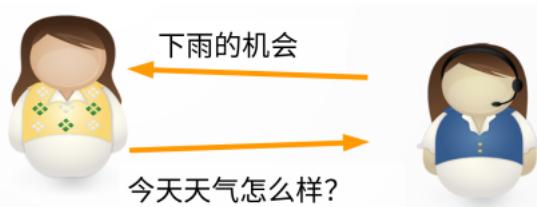


Figure 1.5: Esperimento con una persona.

- Scrivere un programma per un computer non è di per sé una condizione sufficiente per implicare l'intenzionalità.

Note:-

Searle ha scelto il cinese perché voleva considerare una lingua difficile da capire. Per me è skill issue.

AI:

- *AI forte*: è possibile riprodurre l'intelligenza umana? Compresa la consapevolezza di sé, l'essere senziente, etc.
- *AI debole*: esistono modi automatici per risolvere problemi che a un essere umano richiedono intelligenza? Task-oriented, studio del pensiero e del comportamento umano.

1.2.2 Agenti Intelligenti

Agenti triviali (non interessanti):

- Termostato.
- UNIX deamon³

Un agente intelligente deve eseguire azioni in modo *flessibile*:

- *Reattivo*.
- *Proattivo*.
- *Sociale*.

Osservazioni 1.2.1

- Se l'ambiente di un software è statico/fisso non è necessario preoccuparsi di esso (e. g. un compilatore, un package manager).
- Nel mondo reale le cose cambiano, le informazioni sono incomplete.
- È necessario considerare la possibilità che l'esecuzione di azioni (istruzioni) non abbia successo, è necessario chiedersi se valga la pena eseguire una certa azione.

Definizione 1.2.1: Agente Reattivo

Un agente reattivo è un sistema che mantiene una costante interazione con l'ambiente e risponde ai cambiamenti che occorrono in esso (in tempo perché la risposta sia utile).

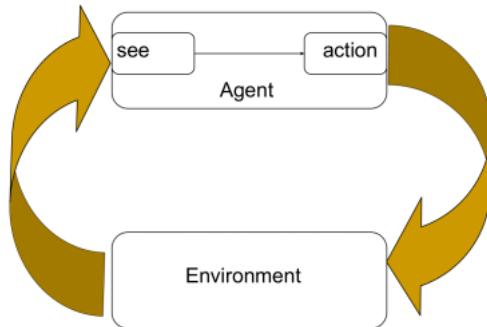


Figure 1.6: Un agente reattivo.

Note:-

Può essere visto come un if then.

Osservazioni 1.2.2

- Reagire a un ambiente è facile, però in generale, si vuole che gli agenti "facciano cose per noi".
- *Goal directed behaviour*: comportamento guidato dagli obiettivi.
- *Proattività*: generare e tentare di raggiungere gli obiettivi, non solo guidati dagli eventi, ossia *prendere l'iniziativa*.
- Riconoscere le opportunità.

Definizione 1.2.2: Agente Proattivo

Lo stato dell'agente contiene due tipi di conoscenza:

- L'agente necessita di informazioni su come l'ambiente evolve.
- L'agente necessita di informazioni su come le proprie azioni impattano sull'ambiente.

L'agente necessita di informazioni sull'*obiettivo* (goal) in modo che l'agente possa agire per raggiungerlo. Nel farlo l'agente deve essere in grado di lavorare su *piani*.

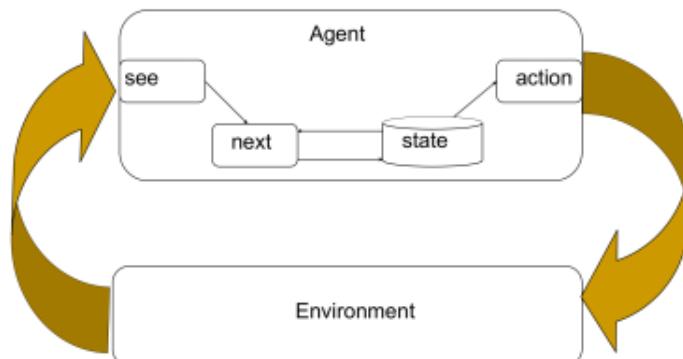


Figure 1.7: Agente proattivo.

³Come si permette di dire che il daemon UNIX non è interessante. Probabilmente non ha mai speso il pomeriggio a riaggiustarlo dopo aver rotto un arch install >:(

Reattivo vs. Proattivo:

- Si desidera che gli agenti siano reattivi e che rispondano ai cambiamenti nel tempo.
- Si desidera che gli agenti lavorino in modo sistematico verso obiettivi di lungo termine.
- Queste due considerazioni possono essere in conflitto.
- Progettare un agente che bilanci reattività e proattività è ancora un problema aperto.

Osservazioni 1.2.3

- Il mondo reale è spesso un ambiente multiagente, per cui non si possono raggiungere i propri obiettivi senza l'aiuto degli altri.
- Alcuni obiettivi possono essere raggiunti solo con la collaborazione di altri.
- allo stesso modo i sistemi sono immersi in una rete di computer.

Definizione 1.2.3: Abilità Sociale

Per abilità sociale di un agente intelligente si intende l'abilità di interagire con altri agenti (anche umani) attraverso un qualche tipo di *linguaggio di comunicazione* (*agent-communication language*, *ACL*) e cooperare con essi.

Domanda 1.4

Agente intelligente o generico programma?

Definizione 1.2.4: Is it an Agent or just a Program?

Un agente autonomo è un sistema situato in un ambiente che può percepire e agire su di esso nel tempo con il fine di perseguire una propria agenda e così facendo influire nelle successive percezioni.

Note:-

Secondo questa definizione sia gli esseri umani che i termostati sono agenti autonomi.

1.3 Cenni alla Computazione

L'ingegneria del software aspira a produrre software di qualità (tramite la *modularizzazione*):

- Correttezza.
- Robustezza.
- Estensibilità.
- Riusabilità.

Note:-

Che sono le stesse cose del software *suckless* che è 100% FOOS, ma ovviamente il prof non ne parla.

Meyer individua, nei linguaggi di programmazione, tre forze (fig. 1.8):

- Processo: la CPU fisica, un processo o un thread.
- Azione: operazioni che costituiscono la computazione.
- Oggetto: le strutture dati a cui si applica l'azione.

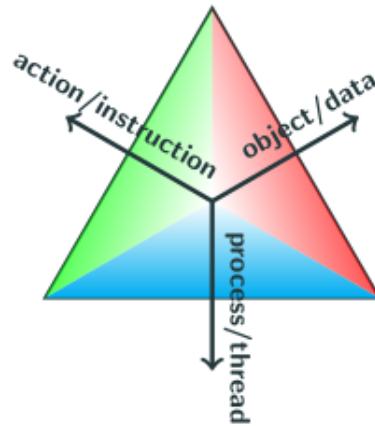
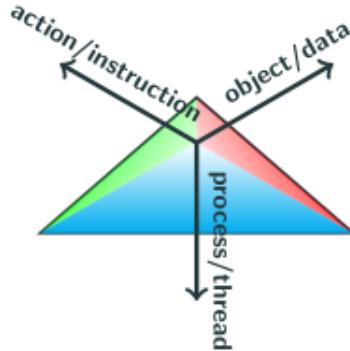


Figure 1.8: Il triangolo di Meyer.

Decomposizione funzionale (a dispetto del nome è l'approccio del paradigma imperativo):

- PRO:
 - Semplice e intuitivo: si costruisce un sistema con la decomposizione in passi.
 - Orientato agli algoritmi: utile quando si ha un solo top goal.
- CONTRO:
 - Difficile da mantenere⁴: difficile incorporare nuovi goal.
 - Difficilmente scalabile in presenza di dati condivisi e processi concorrenti.



Puts at the center the process force

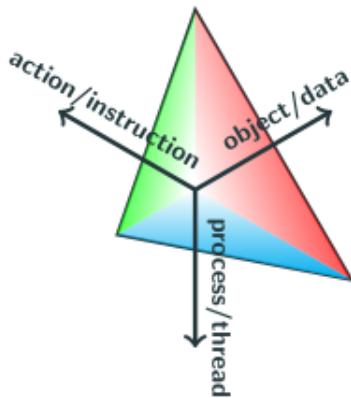
Figure 1.9: Decomposizione funzionale.

Decomposizione orientata agli oggetti:

- PRO:
 - Gli oggetti hanno una vita loro, indipendentemente dal processo che li usa.
 - Operazioni sui dati: fornisce azioni per lavorare su essi.

⁴Skill issue.

- GLi oggetti sono modelli.
- Contro:
 - Gli oggetti sono passivi: un processo esterno prende la decisione su quale azione invocare su un oggetto.
 - Non c'è differenza tra uso di un oggetto e gestione di un oggetto.
 - Non c'è supporto concettuale alle specifiche di un task.



Puts at the center the object force

Figure 1.10: Decomposizione orientata agli oggetti.

Modello ad attori e oggetti attivi:

- Gli attori hanno il proprio thread.
- Il modello degli attori non prende in considerazione il problema della coordinazione (sebbene siano state previste estensioni).
- Secondo le forze di Meyer:
 - Supporta la gestione dei dati e degli oggetti.
 - Non supporta la modularizzazione usando gli oggetti stessi.

Processi di Business:

- Crea una rappresentazione esplicita delle attività di un'organizzazione.
- Descrive come un insieme di attività collegate conducano a risultati precisi e misurabili in risposta a un evento esterno.
- Si mette enfasi sulla forza processo (visione attività-centrica).
- Ha gli stessi limiti della decomposizione funzionale.

Gestione dei processi artefatto-centrica:

- Ci si sposta da una visione attività-centrica a una visione dato-centrica.
- Business artifact (BA).

1.3.1 Paradigma Orientato agli Agenti e Architetture per Sistemi ad Agenti

Definizione 1.3.1: Paradigma Orientato agli Agenti

Il paradigma orientato agli agenti si basa su due astrazioni:

- Agente (forza processo).
- Ambiente (forza azione/oggetto).

Agenti vs. Oggetti:

- Gli oggetti non hanno controllo sul proprio comportamento.
- Gli oggetti non hanno un comportamento flessibile.
- Gli oggetti sono single-threaded.

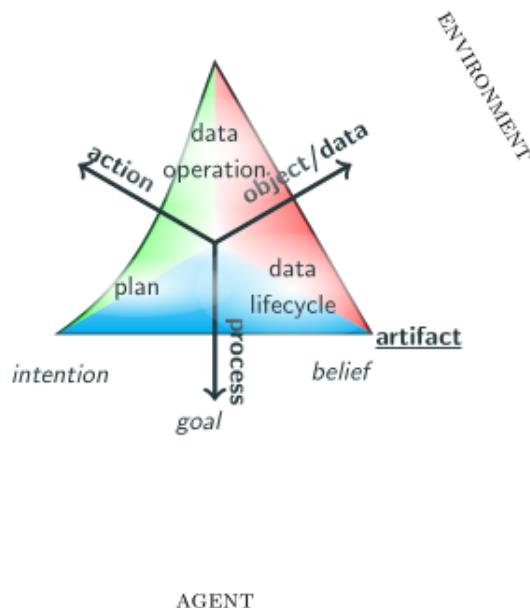


Figure 1.11: Forze nel paradigma Agent-oriented.

Domanda 1.5

Come realizzare agenti intelligenti? Come costruire agenti che abbiano le caratteristiche di autonomia, di reattività, di proattività e di abilità sociali fin qui citate?

- 1956-1985: la maggior parte dei progetti di architetture per agenti si basano sul *ragionamento simbolico* sviluppato per IA. Utilizzo del ragionamento logico per capire cosa fare.
- 1985-presente: *Reactive agents movements*.
- 1990-presente: *architetture ibride* che combinano il meglio delle architetture basate sul ragionamento logico e quelle reattive. (e. g. Jason).

Note:-

Il classico approccio per costruire agenti intelligenti è quello di vederli come casi particolari di sistemi basati sulla conoscenza (paradigma *symbolic AI*).

L'architettura di un agente deliberativo:

- Contiene una esplicita rappresentazione (*modello simbolico*) dell'ambiente.
- Prende decisioni attraverso un *ragionamento simbolico*.

Però ci sono due problemi da affrontare:

- *Il problema della trasduzione*: il problema della traduzione del mondo reale, dell'ambiente, in una descrizione simbolica accurata, in tempo perché possa essere utile.
- *Il problema della rappresentazione/ragionamento*: il problema di come rappresentare simbolicamente le informazioni su entità e processi complessi del mondo reale, e come fare in modo che gli agenti ragionano con queste informazioni in tempo perché i risultati possano essere utili.

Osservazioni 1.3.1

- Entrambi i problemi non sembrano facilmente risolvibili, servono alternative.
- Il problema risiede nella *complessità* degli algoritmi di manipolazione simbolica. La maggior parte degli algoritmi rilevanti sono *altamente intrattabili*.

2

Logica per Agenti

La logica è stata sviluppata nel corso di molti secoli da matematici e filosofi per modellare il "ragionamento corretto" degli esseri umani e ha avuto un ruolo importante nello sviluppo dell'intelligenza artificiale per realizzare agenti e sistemi multiagente.

La logica fornisce strumenti formali per:

- *La rappresentazione della conoscenza:*
 - Un linguaggio formale con una semantica precisa.
 - Rappresentazione dichiarativa della conoscenza.
- *Ragionamento automatico:*
 - Refole di inferenza.

Note:-

Queste proprietà della logica hanno diverse qualità: per esempio possono spiegare in modo preciso i passi di ragionamento di un agente.

Ruoli della logica per gli agenti:

- La logica può essere utilizzata da un agente intelligente per *rappresentare la conoscenza* e per *ragionare*. Dato che la conoscenza è espressa in un linguaggio formale, gli agenti possono usare metodi formali per derivare altra conoscenza.
- La logica può servire per *specificare il comportamento* di un agente intelligente. In questo caso la logica può essere usata per *verificare* che l'agente si comporti come specificato, anche se l'agente non fa uso della logica nel suo funzionamento.

La logica classica:

- Normalmente, quando si parla di logica in ambito AI si intende la *logica classica*, ossia la *logica proposizionale* o la *logica del primordine*.
- Però la necessità di modellare concetti diversi e le esigenze di efficienza hanno portato l'AI all'uso di logiche diverse da quelle classiche e anche alla definizione di nuove logica, per esempio le logiche non monotone.
- Nell'ambito degli agenti e dei sistemi multiagente si preferisce utilizzare la *logica modale*.

2.1 Richiami di Logica Classica

Definizione 2.1.1: Logica Classica

Le formule sono costituite da proposizioni atomiche appartenenti a un insieme P e da connettivi logici secondo la seguente formulazione, con $p \in P, \phi, \psi$ sono formule:

- p .
- $\phi \vee \psi$.
- $\neg\phi$.

Note:-

Quelli presentati nella definizione sono solo alcuni dei connettivi (ci sono anche and, nand, nor, xor, implica, etc.).

2.1.1 Semantica

Definizione 2.1.2: Semantica della Logica Proposizionale

La semantica definisce la verità delle formule rispetto a ogni modello. Nella logica proposizionale, un modello assegna un valore di verità (true o false) a ogni simbolo proposizionale.

Osservazioni 2.1.1

- Se $|P| = n$ ci sono n modelli (tavole di verità).
- Un modello può essere rappresentato come un insieme $M \subseteq P$ che contiene tutte le proposizioni atomiche che sono vere nel modello. Quelle che non appartengono a M sono false.

Definizione 2.1.3: Soddisfacibilità

La semantica definisce una relazione di soddisfacibilità $M \models \phi$ di una formula ϕ in un modello M (interpretazione):

- $M \models p$ se e solo se $p \in M$.
- $M \models \phi \vee \psi$ se e solo se $M \models \phi$ o $M \models \psi$.
- $M \models \neg\phi$ se e solo se $M \not\models \phi$.

Osservazioni 2.1.2

- Una formula è *soddisfacibile* se è solo se c'è *qualche* modello che la soddisfa.
- Una formula è *valida* se e solo se è soddisfatta da *ogni modello* (tautologia).

2.1.2 Sistemi Deduttivi e Logica del Primordine

Definizione 2.1.4: Modus Ponens

$$\frac{\alpha \Rightarrow \beta \quad \alpha}{\beta}$$

Note:-

L'applicazione di una sequenza di regole di inferenza porta a una derivazione $KB \models \alpha$. Ovviamente ciò che si deriva da un insieme KB è vero in tutti i modelli di KB .

Definizione 2.1.5: Deduzione

Data una base di conoscenza (insieme di formule) da KB , una formula α segue logicamente da KB :

$$KB \models \alpha$$

se e solo se, in ogni modello in cui KB è vera, anche α lo è.

Corollario 2.1.1 Teorema di Deduzione

Date due formule α e β ,

$$\alpha \models \beta$$

se e solo se la formula

$$\alpha \Rightarrow \beta$$

è valida.

Note:-

La logica classica del primordine estende la logica proposizionale con quantificatori universali ed esistenziali.

2.2 La Logica Modale

2.2.1 Inadeguatezza della Logica Classica

Gli agenti sono descritti come *sistemi intenzionali* attribuendo loro *stati mentali*. Supponiamo di voler formulare, con la logica, la seguente frase: *John crede che Superman voli*. In logica classica questo potrebbe essere espresso come: *Bel(John, vola(Superman))*, dove *Bel* è un predicato.

Questa formulazione non funziona per almeno due ragioni:

- *Ragione sintattica*: le formule della logica classica hanno la seguente struttura *Predicato(Term, ..., Term)*, però il secondo argomento di *Bel* è una formula e non un termine come richiesto.
- *Ragione semantica*: gli operatori intenzionali come *Bel* sono *referentially opaque*, ossia creano contesti chiusi in cui non è possibile sostituire una formula con una equivalente come in logica classica.

2.2.2 Introduzione alla Logica Modale

La semantica della *logica modale* è formulata in termini di *mondi possibili*. Ogni mondo rappresenta una situazione considerata possibile dall'agente. Ciò che è vero in tutti i mondi si può dire sia creduto dall'agente.

- Da un punto di vista filosofico, la proprietà fondamentale della logica modale è che tutto ciò che è vero in tutti i mondi è *necessariamente* vero. Ciò che è vero in qualche mondo è *possibile*.
- La logica modale è stata sviluppata per distinguere tra *verità necessarie* e *verità contingenti*.

Definizione 2.2.1: Logica Modale

La logica modale proposizionale estende la logica classica proposizionale con due operatori modali: \Box (verità necessaria) e \Diamond (verità contingente/possibile). Le formule sono:

- p
- $\varphi \vee \psi$
- $\neg\varphi$
- $\Box\varphi$
- $\Diamond\varphi$

Dove $p \in P$ e φ e ψ sono formule della logica.

Osservazioni 2.2.1

- I due operatori modali sono uno il duale dell'altro: $\Box\varphi \equiv \neg\Diamond\neg\varphi$ (se è necessario che φ sia vera allora, indipendentemente da cosa sia, non è possibile che φ sia falsa) e $\Diamond\varphi \equiv \neg\Box\neg\varphi$ (se è possibile che φ sia vera allora non è vero che è falsa in tutti i mondi, poiché in *almeno* un mondo è vera).
- Volendo si potrebbe minimizzare l'uso della sintassi utilizzando solo uno dei due operatori.

Definizione 2.2.2: Semantica della Logica Proposizionale

- Un *modello* della logica modale è dato come un insieme di mondi possibili.
- Ogni *mondo* è costituito da un insieme di proposizioni che sono considerate vere in quel mondo.
- La *struttura* del modello è descritta da una *relazione binaria di accessibilità* che collega coppie di mondi.

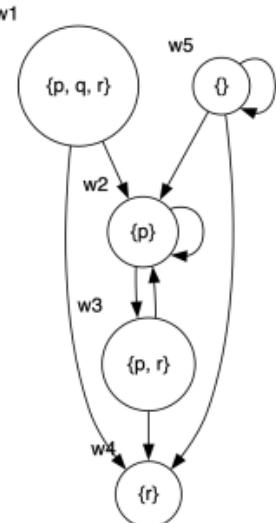
Corollario 2.2.1 Modello

Un modello M è una tripla $\langle W, R, L \rangle$ dove:

- W è un insieme di mondi.
- $R \subseteq W \times W$ è una relazione di accessibilità tra mondi.
- $L : W \rightarrow 2^P$ dà l'insieme di proposizioni vere in ogni mondo in W .

Esempio 2.2.1 (Modello)

- $W = \{w_1, w_2, w_3, w_4, w_5\}$
- $R = \{(w_1, w_2), (w_1, w_4), (w_2, w_1), (w_2, w_2), (w_2, w_3), (w_3, w_2), (w_3, w_4), (w_5, w_2), (w_5, w_4), (w_5, w_5)\}$
- $L = \{(w_1, \{p, q, r\}), (w_2, \{p\}), (w_3, \{p, r\}), (w_4, \{r\}), (w_5, \emptyset)\}$



- Da notare che R può collegare un mondo a sé stesso con un loop.

Definizione 2.2.3: Soddisfacibilità

La soddisfacibilità di una formula φ è definita rispetto a un modello M e a un mondo w di questo modello con la notazione $M \models_w \varphi$:

- $M \models_w p$ se e solo se $p \in L(w)$
- $M \models_w \varphi \vee \psi$ se e solo se $M \models_w \varphi$ or $M \models_w \psi$
- $M \models_w \neg\varphi$ se e solo se $M \not\models_w \varphi$
- $M \models_w \Box\varphi$ se e solo se $\forall w' (R(w, w') \Rightarrow M \models_{w'} \varphi)$
- $M \models_w \Diamond\varphi$ se e solo se $\exists w' (R(w, w') \Rightarrow M \models_{w'} \varphi)$

Corollario 2.2.2 Validità

Una formula φ è valida se è vera in tutte le coppie modello/mondo. La nozione di validità è analoga a quella di tautologia (qualcosa che è sempre vero) nella logica proposizionale classica.

Note:-

I modelli della logica modale sono spesso chiamati *Kripke Structures*. La coppia $\langle W, R \rangle$ è chiamata *Kripke frame*.

Proprietà fondamentali della logica modale:

- Assioma K*¹: $\Box(\varphi \Rightarrow \psi) \leftrightarrow (\Box\varphi \Rightarrow \Box\psi)$.
- Necessitation*, regola di inferenza: se φ è valida allora $\Box\varphi$ è valida.

Note:-

L'assioma K è proprio delle logiche modali "normali". Esistono altre logiche modali che non l'hanno.

¹Sempre in onore di Kripke.

Distribuitività:

- La logica modale distribuisce su AND: $\Box(\varphi \wedge \psi) \equiv (\Box\varphi \wedge \Box\psi)$.
- La logica modale non distribuisce su OR: $\Box(\varphi \vee \psi) \not\equiv (\Box\varphi \vee \Box\psi)$.

Domanda 2.1

Ma cosa c'entra questa logica con gli agenti?

- Si dà agli operatori modali un significato diverso da quello standard (necessario o possibile) per modellare una proprietà degli agenti, senza modificare la semantica dei mondi possibili.
- Si attribuisce all'operatore \Box il significato di *credere* (belief), ergo $\Box\varphi$ significa che l'agente crede che φ sia vero.
- Per chiarezza l'operatore \Box è chiamato B (ossia belief, credenza). All'operatore duale non viene attribuito un nome.

Principali logiche modali per agenti:

- *Logica epistemica* (conoscenza e credenza):

- $K_a\varphi$ l'agente a sa che φ è vero.
- $B_a\varphi$ l'agente a crede che φ sia vero.

- *Logiche Belief-Desire-Intention*:

- $B_a\varphi$ l'agente a crede che φ sia vero.
- $D_a\varphi$ l'agente a desidera φ .
- $I_a\varphi$ l'agente a ha l'intenzione φ .

- *Logiche deontologiche*:

- $O\varphi$ è obbligatorio che φ .
- $P\varphi$ è permesso che φ .

- *Logica temporale* (lineare):

- $X\varphi$, φ sarà vero al prossimo istante.
- $G\varphi$, φ sarà sempre vero.
- $F\varphi$, φ prima o poi diventerà vero.
- $\psi \cup \varphi$, φ è vero fino a quando φ diventa vero.

- *Logica dinamica* (delle azioni):

- $[\pi]\varphi$ dopo l'esecuzione del programma π , φ è vero.
- π è un'azione complessa (programma) ottenuta combinando azioni elementari.

Osservazioni 2.2.2

- Se \Box rappresenta la conoscenza, si vuole che la logica avesse la proprietà che tutto ciò che è conosciuto è vero. Questo può essere espresso aggiungendo l'assioma:

$$\Box\varphi \Rightarrow \varphi$$

- La formula $\Box\varphi \Rightarrow \varphi$ non vale in tutti i modelli.
- Si è può esprimere che le conoscenze dell'agente non devono essere contraddittorie, aggiungendo

l'assioma:

$$\Box\varphi \Rightarrow \neg\Box\neg\varphi$$

Proprietà dei frame:

- *Reflexive* ($\Box\varphi \Rightarrow \varphi$):

$$\forall w \in W . (w, w) \in R$$

- *Serial* ($\Box\varphi \Rightarrow \Diamond\varphi$):

$$\forall w \in W . \exists w' \in W . (w, w') \in R$$

- *Transitive* ($\Box\varphi \Rightarrow \Box\Box\varphi$):

$$\forall w, w', w'' \in W . ((w, w') \in R \wedge (w', w'') \in R) \Rightarrow (w, w'') \in R$$

- *Euclidean* ($\Diamond\varphi \Rightarrow \Box\Diamond\varphi$):

$$\forall w, w', w'' \in W . ((w, w') \in R \wedge (w, w'') \in R) \Rightarrow (w', w'') \in R$$

Note:-

A ciascuno di queste proprietà è stato dato un nome, in ordine: T, D, 4 e 5. Combinando queste proprietà si ottengono 11 sistemi modali (sarebbero 16, ma alcuni sono equivalenti).

Ad alcuni sistemi "notevoli" è stato dato un nome:

- KT è T.
- KT4 è S4.
- KD45 è weak-S5.
- KTD45 è S5.

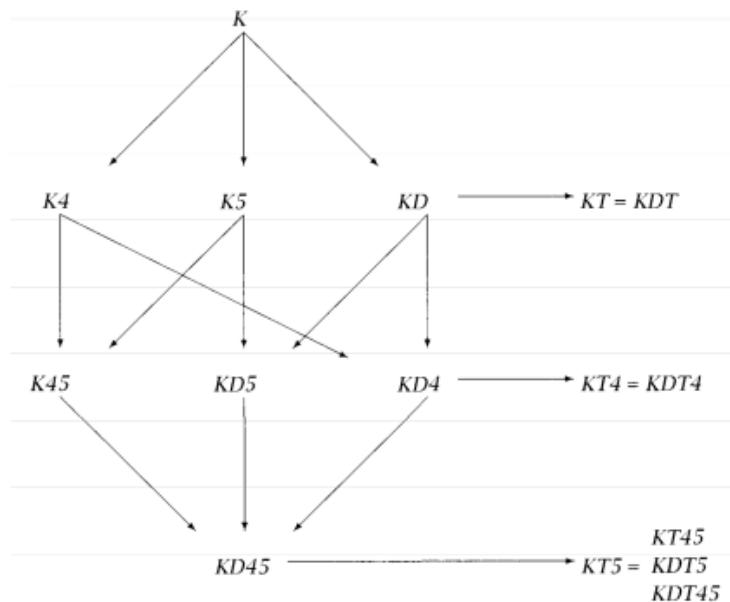


Figure 2.1: Gli 11 sistemi modali.

2.3 Sistemi Modali

2.3.1 Accenni di Logica Epistemica

Definizione 2.3.1: Logica Epistemica

La logica epistemica è la logica della conoscenza e delle credenze. Si introducono i due operatori modali K_a e B_a per rappresentare rispettivamente la conoscenza e le credenze di un agente.

- **Assioma K:** $K_a(\varphi \Rightarrow \psi) \leftrightarrow (K_a\varphi \Rightarrow K_a\psi)$.
- **Necessitation:** se φ è valida, allora $K_a\varphi$.

Il problema dell'omniscienza logica:

- Si assume che ψ sia una conseguenza logica di φ . Allora $\varphi \Rightarrow \psi$ deve essere valida. Secondo la necessitation questa formula deve essere conosciuta dall'agente: $K_a(\varphi \Rightarrow \psi)$.
- Questo significa che l'agente deve conoscere tutte le tautologie (che sono infinite).
- Inoltre se l'agente conosce φ , per l'assioma K, deve conoscere ψ . Questo significa che la conoscenza dell'agente è chiusa rispetto alla conseguenza logica.
- In altre parole se l'agente conosce una cosa e questa cosa ne implica un'altra allora l'agente deve conoscere anche quell'altra cosa.
- In questo corso si ignora completamente il problema.



Figure 2.2: Agente: "Throughout Heaven and Earth, I alone am the omniscient one".

2.3.2 Assiomi per Knowledge e Belief

Si possono realizzare dei sistemi modali aggiungendo nuovi assiomi alla logica modale standard:

- **Assioma D (serialità):** la conoscenza di un agente è non contraddittoria, $K\varphi \Rightarrow \neg K\neg\varphi$, se l'agente conosce φ allora non conosce $\neg\varphi$.
- **Assioma T (riflessività):** ciò che l'agente conosce è vero, $K\varphi \Rightarrow \varphi$, è accettabile per la conoscenza (non vogliamo che l'agente conosca qualcosa che è falso) ma non per le credenze (si accetta che l'agente crede vero qualcosa che è falso).
- **Assioma 4 (transitività):** $K \Rightarrow KK\varphi$.
- **Assioma 5:** $\neg K\varphi \Rightarrow K(\neg K\varphi)$.

Note:-

Gli assiomi di introspezione (4 e 5) implicano che l'agente ha conoscenza perfetta su quello che sa e quello che non sa.

Esempio 2.3.1 (Muddy Children)

- Consideriamo un esempio classico: dopo aver giocato nel fango, due bambini sanno che almeno uno di loro ha del fango sulla fronte (in realtà tutti e due lo hanno). Di solito i bambini sono almeno tre, ma per semplicità consideriamo il caso con due.
- Ogni bambino può vedere la fronte dell'altro, ma non la propria.
- Inizialmente il bambino B dice:

"Non so se ho la fronte infangata"

- Successivamente il bambino A dice:

"Io so di avere la fronte infangata"

- Indichiamo con K_A e K_B le conoscenze del bambino A e di B.

Quindi:

1. $K_A(\neg\text{muddy}(A) \Rightarrow K_B(\neg\text{muddy}(A)))$
2. $K_A(K_B(\neg\text{muddy}(A) \Rightarrow \text{muddy}(B)))$
3. $K_A(\neg K_B(\text{muddy}(B)))$
4. $\neg\text{muddy}(A) \Rightarrow K_B\neg\text{muddy}(A)$, da (1) e l'assioma della conoscenza ($K\varphi \Rightarrow \varphi$) (ciò che è conosciuto è vero)
5. $K_B(\neg\text{muddy}(A) \Rightarrow \text{muddy}(B))$, da (2) e l'assioma della conoscenza
6. $K_B\neg\text{muddy}(A) \Rightarrow K_B\text{muddy}(B)$, da (5) e l'assioma K ($K(\varphi \Rightarrow \psi) \Rightarrow (K\varphi \Rightarrow K\psi)$)
7. $\neg\text{muddy}(A) \Rightarrow K_B\text{muddy}(B)$, da (4) e (6) per transitività (se $\alpha \Rightarrow \beta$ e $\beta \Rightarrow \gamma$, allora $\alpha \Rightarrow \gamma$)
8. $\neg K_B\text{muddy}(B) \Rightarrow \text{muddy}(A)$, contrapposto di (7) ($\alpha \Rightarrow \beta \equiv \neg\beta \Rightarrow \neg\alpha$)
9. $K_A(\neg K_B\text{muddy}(B) \Rightarrow \text{muddy}(A))$, da (8) per necessitation (derivazione: (1) e (2) significano che A conosce le formule (4) e (5). Dalle formule (4) e (5) si deriva la formula (8), quindi (4) e (5) implicano (8) per il teorema di deduzione. La regola dell'omniscienza logica implica che A conosce la formula (8))
10. $K_A\neg K_B\text{muddy}(B) \Rightarrow K_A\text{muddy}(A)$, da (9) e l'assioma K

2.3.3 Ragionare sul Tempo

La logica temporale è la logica modale del tempo. Ci sono molte varianti della logica temporale. In questo corso si tratterà solo il caso in cui il tempo:

- Ha un istante iniziale.
- È infinito nel futuro.

Definizione 2.3.2: Linear Temporal Logic (LTL)

La struttura del tempo è un insieme totalmente ordinato di istanti di tempo. Un modello M è una struttura lineare, spesso chiamata *Kripke structure*, definita come:

$$M = \langle S, x, L \rangle$$

Dove:

- S è un insieme di stati.
- $x : \mathbb{N} \rightarrow S$ è una sequenza infinita di stati, con \mathbb{N} che rappresenta l'insieme dei numeri naturali.
- $L : S \rightarrow 2^P$ assegna a ogni stato l'insieme delle proposizioni vere in quello stato.

Corollario 2.3.1 Formule

Le formule della Propositional Linear Temporal Logic (PLTL) sono definite come segue:

- p , con $p \in P$
- $\alpha \vee \beta$
- $\neg\alpha$
- $X\alpha$
- $\alpha U \beta$

Dove $\alpha, \beta \in \text{PLTL}$.

Semantica informale degli operatori modali:

- $X\alpha$ ("nexttime α ") significa che α è vero nel prossimo istante di tempo.
- $\alpha U \beta$ (" α until β ") è vero al tempo t se e solo se β è vero in un futuro istante t_0 e α è vero in tutti gli istanti fra t e t_0 .

Operatori modali derivati:

- $F\alpha \equiv \text{true} U \alpha$ ("prima o poi α ").
- $G\alpha \equiv \neg F \neg \alpha$ ("sempre α ").

In Computing Tree Logic (CTL*) è possibile formulare due tipi di formule:

- **Path formulas:** riguardano i cammini infiniti della struttura temporale ad albero e sono simili alle formule della Logica Temporale Lineare (LTL).
- **State formulas:** riguardano tutti i cammini infiniti uscenti da uno stato.

Domanda 2.2

Quali sono gli usi delle logiche temporali:

- Sono usate per verificare la correttezza di sistemi concorrenti (e. g. multiagente) con la tecnica del *model checking*.
- Dato che il model checking affronta problemi ad alta complessità si utilizza una logica CTL (che restringe la sintassi della CTL*).

Definizione 2.3.3: Model Checking

Il model checking è un metodo per la verifica di proprietà di agenti e di sistemi multiagente (o, in generale, sistemi concorrenti).

Note:-

Le proprietà da verificare sono basate su logiche temporali e sulla relazione tra i modelli delle logiche temporali e degli automi a stati finiti che ne descrivono le computazioni.

2.3.4 Ragionare sulle Azioni

Il ragionamento sulle azioni si basa sulla logica classica:

- *Situazioni*: stato del mondo a qualche istante nel tempo.
- *Fluenti*: proposizioni il cui valore varia da una situazione all'altra.
- *Azioni*: causano un cambiamento nello stato del mondo.

Definizione 2.3.4: Azione

Ogni azione è descritta da due assiomi:

- Un'assioma di possibilità: che dice quando è possibile eseguire l'azione.
- Un'assioma di effetto: cosa accade quando un'azione possibile è eseguita.

Definizione 2.3.5: Frame Problem

Un'azione influenza solo un numero limitato di fluenti. Bisogna trovare un modo per descrivere il fatto che tutto il resto non cambi.

3

Agenti Razionali e Architetture

3.1 Credenze, Desideri e Intenzioni

Definizione 3.1.1: Sistema Intenzionale

Un sistema intenzionale è un sistema il cui comportamento può essere previsto dal metodo di attribuzione di credenze, di desideri e di acume razionale.

Note:-

Per McCarthy, in alcuni casi si possono attribuire posizioni intenzionali a sistemi informatici.

Osservazioni 3.1.1

- Più si sa di un sistema meno si fa affidamento a spiegazioni animistiche.
- Con sistemi complessi una spiegazione meccanica potrebbe non essere praticabile.

Definizione 3.1.2: Intenzioni e Astrazioni

Più i sistemi di calcolo diventano complessi più si ha bisogno di astrazioni e metafore per spiegare il loro funzionamento. Le posizioni intenzionali sono una tale astrazione.

Note:-

Le nozioni intenzionali sono quindi astrazioni che forniscono un modo semplice e familiare per descrivere, spiegare e prevedere il comportamento di sistemi complessi.

I più importanti sviluppi sono basati su astrazioni:

- Procedure e funzioni.
- Tipi di dati astratti (bloat).
- Oggetti.
- Agenti intelligenti e sistemi intenzionali.

3.1.1 Ragionamento su Azioni

Definizione 3.1.3: Practical Reasoning

Il practical reasoning è il ragionamento sulle azioni, sul processo di capire cosa fare.

Note:-

Si distingue da theoretical reasoning che riguarda le credenze.

Il practical reasoning consiste in:

- *Deliberazione*: decidere quale stato di cose vogliamo raggiungere.
- *Pianificazione* (means-ends reasoning/planning): decidere come raggiungere questi stati di cose.

Note:-

Dopo aver ottenuto un piano un agente tenterà di eseguirlo.

Osservazioni 3.1.2

- Il calcolo è una risorsa preziosa per gli agenti, un agente deve controllare il suo ragionamento.
- Gli agenti non possono deliberare a tempo indeterminato, a un certo punto devono smettere di deliberare e, dopo aver scelto uno stato di cose, impegnarsi per raggiungerlo.

Corollario 3.1.1 Intenzioni

Ci riferiamo allo stato di cose che un agente ha scelto e al quale si impegna come le sue intenzioni.

Note:-

Il ruolo delle intenzioni è quello di *favorire la proattività*, ossia portare all'azione.

Conseguenze:

- Persistenza delle azioni: se si adotta un'intenzione allora si deve insistere su essa e tentare di realizzarla. Se la motivazione cessa di esistere è razionale abbandonare l'intenzione.
- Le intenzioni vincolano il futuro practical reasoning.

Intenzioni nel practical reasoning:

1. Ci si aspetta che gli agenti trovino i modi per realizzare le loro intenzioni.
2. Le intenzioni forniscono un *filtro* per l'adozione di altre intenzioni, che non devono entrare in conflitto.
3. Gli agenti monitorano il successo delle loro intenzioni e sono inclini a *riprovare* se i loro tentativi falliscono.
4. Gli agenti credono che le loro intenzioni siano possibili¹.
5. Gli agenti non credono che non realizzeranno le loro intenzioni².
6. Nelle giuste circostanze gli agenti credono che realizzeranno le loro intenzioni.
7. Gli agenti non devono necessariamente avere intenzioni su tutti gli effetti collaterali delle loro intenzioni.

¹Gli agenti sono come Naruto che vuole diventare Hokage.

²Gli agenti sono come Naruto che vuole diventare Hokage pt. 2.

Osservazioni 3.1.3

- Il problema di avere l'intenzione di realizzare qualcosa credendo che non sia possibile è noto come *intention-belief inconsistency* e non è razionale. (punto 5)
- Il problema di avere l'intenzione di realizzare qualcosa senza credere che la realizzaranno è noto come *intention-belief incompleteness* ed è una proprietà accettabile per gli agenti razionali. (punto 6)
- Il problema al punto 7 è noto come *package deal*.

3.1.2 Modellazione Logica di Agenti BDI

- La logica riguarda principalmente il *rational balance* relativo a beliefs, goals, plans, intentions, commitments e azioni di agenti autonomi.
- L'obiettivo principale è di esplorare le reazioni che le intenzioni hanno nel mantenere questo bilanciamento.
- Questo deve essere analizzato con *beliefs*, *desires* e *intentions* (BDI).

La logica di Cohen e Levesque ha 4 operatori modali principali:

- $(BEL i \phi)$: l'agente i crede ϕ .
- $(GOAL i \phi)$: l'agente i ha il goal (desire) ϕ .
- $(HAPPENS \alpha)$: l'azione α occorre al passo successivo.
- $(DONE \alpha)$ l'azione α è appenaoccorsa.

Note:-

Un'azione α può essere un evento primitivo oppure complesso. L'intenzione non è un operatore primitivo, ma può essere derivato.

Successivamente viene proposta un'altra logica, da Rao e Georgeff (BDI):

- Ci sono tre modalità: *BEL*, *GOAL* e *INTEND*.
- I mondi sono strutture temporali branching time.

È possibile descrivere commitments diversi:

- Agente *blindly committed* (fanatical): mantiene le sue intenzioni finché non arriva a credere di averle soddisfatte.

$$INTEND(AF \phi) \Rightarrow A(INTEND(AF \phi) \cup BEL(\phi))$$

- Agente *Single-minded committed*: mantiene le sue intenzioni finché crede che queste possano essere realizzabili.

$$INTEND(AF \phi) \Rightarrow A(INTEND(AF \phi) \cup (BEL(\phi) \vee \neg BEL(EF \phi)))$$

- Agente *open-minded*: mantiene le sue intenzioni finché queste intenzioni sono ancora i suoi goals.

$$INTEND(AF \phi) \Rightarrow A(INTEND(AF \phi) \cup (BEL(\phi) \vee \neg GOAL(EF \phi)))$$

3.1.3 Un'Architettura per agenti BDI

Agent Control Loop Version 1:

```
while true do
    observe the world;
    update internal world model;
    deliberate about what
        intention to achieve next;
    use means-ends reasoning to
        get a plan for the intention;
    execute the plan
```

Figure 3.1: Una prima versione di un agente che utilizza il practical reasoning.

Nella prima versione:

- I processi di deliberation e means-end reasoning non sono immediati, hanno un costo nel tempo.
- Supponiamo che l'agente inizi a deliberare all'istante t_0 , inizi il means-end reasoning all'istante t_1 e che cominci l'esecuzione del piano all'istante t_2 :
 - Il tempo per deliberare è $t_d = t_1 - t_0$.
 - Il tempo per il means-end reasoning è $t_{me} = t_2 - t_1$.
 - Supponiamo inoltre che la deliberazione sia ottimale, cioè se si seleziona un'intenzione da raggiungere essa sia la migliore per l'agente.
 - Al tempo t_1 l'agente ha selezionato l'intenzione da raggiungere che sarebbe stata ottimale se fosse stata raggiunta all'istante t_0 .
 - A meno che t_d sia molto piccolo c'è il rischio che l'intenzione non sia più ottimale (calculative rationality).

L'agente avrà un comportamento complessivo ottimale nelle seguenti circostanze:

- Quando il tempo impiegato per i processi di deliberazione di means-end è estremamente piccolo.
- Quando il mondo è *statico* mentre l'agente effettua i suoi processi.
- Quando un'intenzione ottimale se raggiunta al tempo t_0 (il momento in cui si osserva il mondo) è garantita rimanere ottimale fino al tempo t_2 (il tempo in cui l'agente ha determinato le azioni per raggiungere le intenzioni).

Nella seconda versione:

- B : credenze.
- I : intenzioni.
- $brf()$: revisione delle credenze.
- $deliberate()$: deliberazione delle intenzioni.
- $plan()$: pianificazione.

Agent Control Loop Version 2:

```

/* initial beliefs */  $B := B_0;$ 
while true do
    get next percept  $\rho$ ;
     $B := \text{brf}(B, \rho);$ 
     $I := \text{deliberate}(B);$ 
     $\pi := \text{plan}(B, I);$ 
     $\text{execute}(\pi)$ 
```

Figure 3.2: Seconda versione di un agente che utilizza il practical reasoning.

- La pianificazione è la progettazione di una sequenza di azioni che consentirà di raggiungere l'obiettivo desiderato.
- Dati:
 - Una rappresentazione dell'obiettivo/intenzione da raggiungere.
 - Una rappresentazione delle azioni che può eseguire.
 - Una rappresentazione dell'ambiente.
- Si tratta di automatica programming.

Domanda 3.1

Come un agente delibera?

- *Option generation*: inizia cercando di capire quali sono le opzioni a sua disposizione sulla base delle proprie informazioni e credenze.
- *Filtering*: sceglie tra le opzioni possibili e si impegna verso di esse.

Note:-

Le opzioni scelte sono le intenzioni.

Agent Control Loop Version 3:

```

 $B := B_0;$ 
 $I := I_0;$ 
while true do
    get next percept  $\rho$ ;
     $B := \text{brf}(B, \rho);$ 
     $D := \text{options}(B, I);$ 
     $I := \text{filter}(B, D, I);$ 
     $\pi := \text{plan}(B, I);$ 
     $\text{execute}(\pi)$ 
```

Figure 3.3: Terza versione di un agente che utilizza il practical reasoning.

Nella terza versione:

- *option()*: l'agente genera un insieme di possibili alternative. Rappresenta la generazione di opzioni tramite una funzione.
- *filter()*: l'agente sceglie tra le alternative che competono e si impegna a raggiungerle.
- Quando un'opzione è restituita da filter diciamo che l'agente ha fatto un *commitment* verso tale opzione.
- Il commitment implica persistenza temporale, per cui un'intenzione adottata non dovrebbe essere immediatamente abbandonata.

Definizione 3.1.4: Commitment Strategies

Il meccanismo che un agente usa per determinare quando e come un'intenzione possa essere abbandonata.

Note:-

I tipi di strategie sono gli stessi dei tipi di agente (blind, single-minded, open-minded).

Agent Control Loop Version 4:

```

 $B := B_0;$ 
 $I := I_0;$ 
while true do
    get next percept  $\rho$ ;
     $B := brf(B, \rho)$ ;
     $D := options(B, I)$ ;
     $I := filter(B, D, I)$ ;
     $\pi := plan(B, I)$ ;
    while not empty( $\pi$ ) do
         $\alpha := hd(\pi)$ ;
        execute( $\alpha$ );
         $\pi := tail(\pi)$ ;
        get next percept  $\rho$ ;
         $B := brf(B, \rho)$ ;
        if not sound( $\pi, I, B$ ) then
             $\pi := plan(B, I)$ 

```

Figure 3.4: Quarta versione di un agente che utilizza il practical reasoning.

Nella quarta versione:

- Si ripianifica se qualcosa va storto.
- È ancora presente overcommitment rispetto alle intenzioni: non si ferma a valutare se le sue intenzioni siano più o meno adeguate (blind).

Agent Control Loop Version 5:

```

 $B := B_0;$ 
 $I := I_0;$ 
while true do
    get next percept  $\rho$ ;
     $B := brf(B, \rho)$ ;
     $D := options(B, I)$ ;
     $I := filter(B, D, I)$ ;
     $\pi := plan(B, I)$ ;
    while not empty( $\pi$ ) or succeeded( $I, B$ ) or
        impossible( $I, B$ ) do
             $\alpha := hd(\pi)$ ;
            execute( $\alpha$ );
             $\pi := tail(\pi)$ ;
            get next percept  $\rho$ ;
             $B := brf(B, \rho)$ ;
            if not sound( $\pi, I, B$ ) then
                 $\pi := plan(B, I)$ 

```

Figure 3.5: Quinta versione di un agente che utilizza il practical reasoning.

Nella quinta versione:

- Ci si ferma per determinare se le intenzioni hanno avuto successo o se sono diventate impossibili da soddisfare (single-minded).
- L'agente può riconsiderare le sue intenzioni ogni volta che il controllo è sul ciclo esterno, ossia quando:
 - Ha completamente eseguito un piano per raggiungere le sue intenzioni correnti.
 - Ritiene di aver raggiunto le sue attuali intenzioni.
 - Crede che le sue attuali intenzioni non siano più possibili.
- Questo limita il modo in cui consente a un agente di riconsiderare le sue intenzioni.

Agent Control Loop Version 6:

```

 $B := B_0;$ 
 $I := I_0;$ 
while true do
    get next percept  $\rho$ ;
     $B := brf(B, \rho)$ ;
     $D := options(B, I)$ ;
     $I := filter(B, D, I)$ ;
     $\pi := plan(B, I)$ ;
    while not empty( $\pi$ ) or succeeded( $I, B$ ) or
        impossible( $I, B$ ) do
             $\alpha := hd(\pi)$ ;
            execute( $\alpha$ );
             $\pi := tail(\pi)$ ;
            get next percept  $\rho$ ;
             $B := brf(B, \rho)$ ;
             $D := options(B, I)$ ;
             $I := filter(B, D, I)$ ;
            if not sound( $\pi, I, B$ ) then
                 $\pi := plan(B, I)$ 

```

Figure 3.6: Sesta versione di un agente che utilizza il practical reasoning.

Nella sesta versione:

- Riconsidera le intenzioni dopo l'esecuzione di ogni azione (open-minded).
- Tuttavia la riconsiderazione delle intenzioni costa:
 - Un agente che non si ferma abbastanza spesso continuerà a tentare di raggiungere le sue intenzioni anche dopo che sarà chiaro che non possono essere raggiunte.
 - Un agente che riconsidera costantemente dedica poco tempo a raggiungere le sue intenzioni.



Figure 3.7: L'agente che riconsidera costantemente be like.

3.1.4 Riconsiderazione delle Intenzioni

Agent Control Loop Version 7:

```

 $B := B_0;$ 
 $I := I_0;$ 
while true do
    get next percept  $\rho$ ;
     $B := brf(B, \rho);$ 
     $D := options(B, I);$ 
     $I := filter(B, D, I);$ 
     $\pi := plan(B, I);$ 
    while not empty( $\pi$ ) or succeeded( $I, B$ ) or
        impossible( $I, B$ ) do
             $\alpha := hd(\pi);$ 
            execute( $\alpha$ );
             $\pi := tail(\pi);$ 
            get next percept  $\rho$ ;
             $B := brf(B, \rho);$ 
            if reconsider( $I, B$ ) then
                 $D := options(B, I);$ 
                 $I := filter(B, D, I)$ 
            if not sound( $\pi, I, B$ ) then
                 $\pi := plan(B, I)$ 

```

Figure 3.8: Settima versione di un agente che utilizza il practical reasoning.

Nella settima versione:

- Incorpora una componente di controllo esplicita *meta-livello* che decide se eseguire o meno la riconsiderazione.

Situation number	Chose to deliberate?	Changed intentions?	Would have changed intentions?	<i>reconsider(...)</i> optimal?
1	No	—	No	Yes
2	No	—	Yes	No
3	Yes	No	—	No
4	Yes	Yes	—	Yes

Figure 3.9: Componente meta-livello.

Note:-

Il problema è che la *reconsider()* è una funzione oracolo, per cui va implementata mediante euristiche e studi specifici sul dominio. Il costo di questa funzione dovrebbe essere molto inferiore al costo del processo deliberativo stesso.

Kinny e Georgeff hanno investigato ciò in maniera sperimentale:

- Due tipi di agenti:
 - *Bold agents*: non si fermano mai a riconsiderare le intenzioni.
 - *Cautious agents*: riconsiderano le intenzioni dopo ogni azione.

- Considerazioni:
 - Se l'ambiente non cambia rapidamente gli agenti bold funzionano meglio.
 - Se l'ambiente cambia costantemente gli agenti cautious funzionano meglio.

3.2 Linguaggi e Architetture per Agenti

3.2.1 Procedural Reasoning System

Definizione 3.2.1: PRS

Il procedural reasoning system è stata una delle prime architetture per agenti che abbia fatto uso del paradigma BDI. È stata usata in applicazioni multi-agenti.

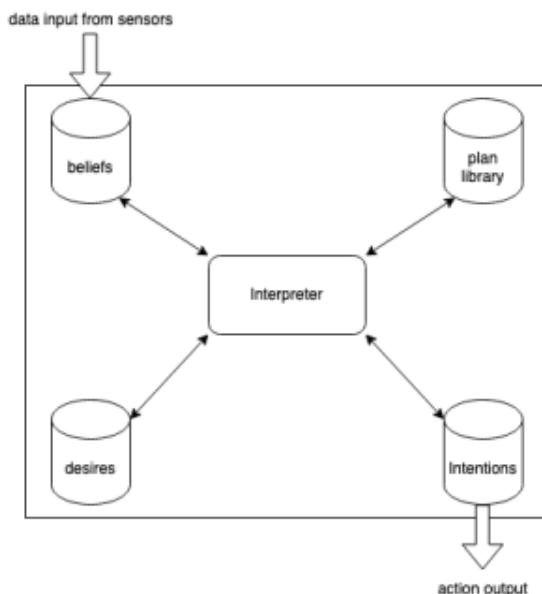


Figure 3.10: Procedural Reasoning System.

Osservazioni 3.2.1

- Gli input del sistema sono *eventi*, ricevuti attraverso una *coda di eventi*.
- Gli eventi possono essere:
 - *Esterne*: percepiti dall'ambiente.
 - *Interni*: come aggiunta o cancellazione di belief o goal.
- Gli output sono azioni che sono eseguite dall'interprete.
- I belief sono rappresentati come fatti del Prolog (atomi della logica del primordine).
- In PRS gli agenti non pianificano, ma fanno riferimento a una libreria di piani predefinita dal programmatore.

I componenti di un piano sono:

- *Nome*.
- *Condizioni di invocazione*.
- *Precondizioni*: le condizioni che devono valere prima di avviare l'esecuzione del piano.
- *Body*: una sequenza di espressioni:
 - Azioni atomiche.
 - Sottogoals.

Note:-

Per raggiungere un dato goal, l'agente formula l'intenzione di raggiungere l'obiettivo, cioè sceglie un piano applicabile. Questo piano diventa un'intenzione, ed è aggiunto alla intention structure.

I piani in PRS:

- I goal in PRS possono essere complessi e contenere a loro volta dei goal.
- I piani possono contenere disgiunzioni di goal, loops, etc.
- Un agente ha un insieme di piani, alcuni belief iniziali riguardo il mondo e un goal iniziale.
- I belief sono rappresentati Prolog-like.

Ragionamento:

- Per raggiungere un dato goal, l'agente formula l'intenzione di raggiungerlo (sceglie un piano applicabile). Il piano diventa un'intenzione ed è aggiunto alla intention structure.
- A ogni passo del loop principale, l'interprete sceglie un piano (parzialmente eseguito) nella intention structure e ne esegue un passo.
- Se ci sono molte opzioni disponibili l'interprete può scegliere quella con la massima utilità o entrare in un ragionamento di meta-livello.

Control loop:

1. Aggiorna belief e goal secondo gli eventi della *event queue*.
2. I cambiamenti dei goal e dei belief fanno scattare i diversi piani.
3. Uno o più piani applicabili sono scelti e messi nella intention structure.
4. Si sceglie una intenzione (task) dalla intention structure.
5. Si esegue un passo di quel task:
 - Esecuzione di un'*azione primitiva*.
 - Scelta di un nuovo subgoal.

Stack delle intenzioni:

- Quando un agente inizia la sua esecuzione, il goal iniziale è inserito nell'intention stack.
- Lo stack contiene tutti i goal in attesa di essere soddisfatti.
- L'agente cerca tra i suoi piani per vedere quali hanno un goal sulla cima dello stack delle intenzioni come la loro post-condizione.
- Tra questi piani alcuni avranno le loro precondizioni soddisfatte dall'insieme di belief correnti.
- L'insieme dei piani che permettono di raggiungere il goal e hanno le precondizioni soddisfatte diventano l'insieme delle possibili opzioni dell'agente.

Osservazioni 3.2.2

- Il processo di selezione tra i diversi piani è la *deliberazione*.
- La deliberazione è ottenuta mediante l'utilizzo di un piano metà meta-livello.
- Effettuata la scelta la sua esecuzione può portare ulteriori goal nello stack.
- Le azioni atomiche potranno essere eseguite direttamente dall'agente.
- Se un piano fallisce, l'agente può selezionare un nuovo piano tra quelli candidati.
- L'esecuzione dei task può essere *interleaved* (supporta il multithread).
- Le intenzioni (task) sono solitamente implementate con uno stack di frames.

Definizione 3.2.2: AgentSpeak(L)

Il linguaggio AgentSpeak(L) è stato proposto come linguaggio di programmazione per agenti (BDI) e consente di essere scritto e interpretato in modo simile a quello dei programmi logici basati su clausole di Horn (Prolog).

Note:-

Si propone di colmare il gap tra specifica teorica e implementazione di un agente BDI. Ha dato origine a JASON.

3.2.2 Agent-Oriented Programming

L'articolo Agent-Oriented programming di Shoham propone un nuovo paradigma di programmazione che promuove una visione sociale della computazione in cui più agenti interagiscono l'uno con l'altro. L'articolo presenta:

- Un linguaggio formale per descrivere stati mentali.
- Un linguaggio di programmazione per definire agenti.

Caratteristiche:

- Ci sono due categorie mentali principali: *belief* e *obligation*. Un'ulteriore categoria, che non è un costrutto mentale, è *capability*.
- *Decision* è una obligation a se stessi.
- Le formule fanno esplicitamente riferimento al tempo. Viene adottato un semplice linguaggio basato sugli istanti di tempo.
- Le azioni non sono distinte dai fatti: la presenza di un'azione è rappresentata dal fatto corrispondente.

Note:-

Il linguaggio di Shoham, Agent0, non ha avuto molta rilevanza, contrariamente al nome Agent-Oriented Programming.

3.2.3 Agenti Reattivi**Note:-**

Un linguaggio per agenti reattivi è CLIPS, visto in "Intelligenza Artificiale e Laboratorio".

Tesi di Brooks:

- Un comportamento intelligente può essere generato *senza rappresentazione esplicita* del tipo di ciò che è proposto dall'IA simbolica.
- Un comportamento intelligente può essere generato *senza ragionamento esplicito* del tipo di ciò che è proposto dall'IA simbolica.
- L'intelligenza è una proprietà *emergente* di certi sistemi complessi.

Due idee chiave:

- *Collocazione e personificazione*: l'intelligenza reale è situata nel mondo, non in sistemi senza sostanza come i "theorem provers" o i "sistemi esperti"³.
- *Intelligenza e apparenza*: il comportamento intelligente è generato come un risultato di un'interazione con l'ambiente.

Definizione 3.2.3: Subsumption Architecture

Una Subsumption Architecture è una gerarchia di task che eseguono behaviours.

Corollario 3.2.1 Behaviour

Una behaviour è una struttura a regole molto semplici.

Subsumption Architecture:

- Ogni behaviours compete con altri per esercitare il controllo sugli agenti.
- Gli strati più bassi rappresentano tipi di behaviours più primitivi e hanno precedenza su strati più in alto nella gerarchia.
- I sistemi risultanti sono estremamente semplici in termini della quantità di computazione che fanno.
- Alcuni robot eseguono task che sarebbero sconvolti se eseguiti da sistemi simbolici di AI.

Selezione delle azioni:

- La scelta di un'azione è realizzata mediante insiemi di behaviours (coppie (c, a)):
 - c : insieme di *condizioni*.
 - a : un'azione.
- Un behaviour scatta quando la funzione *see* restituisce una percezione p tale che $p \in c$.
- Associato a un insieme di behaviour rules c'è una *relazione di inibition*.
- Leggiamo $b_1 < b_2$ come " b_1 inibisce b_2 ", cioè b_1 è più basso nella gerarchia di b_2 , quindi ha priorità.

Definizione 3.2.4: Steels Mars Explorer

Steels propose un sistema di esplorazione per Marte usando la Subsumption Architecture. La soluzione fa uso di due meccanismi:

- Un campo gradiente per trovare la direzione verso la base.
- "Briciole radioattive" per permettere la comunicazione tra agenti.

³AKA touch some grass.

Regole non-cooperative:

1. Evitare gli ostacoli.
2. I campioni portati dagli agenti sono depositati alla base.
3. Gli agenti che portano i campioni ritornano alla base.
4. Gli agenti raccolgono i campioni che trovano.
5. Un agente che non sa cosa fare esplora a caso.

Osservazioni 3.2.3

- Si vuole favorire la collaborazione.
- Un agente crea un sentiero di briciole radioattive ogni volta che trova un campione di roccia.
- Il sentiero sarà creato quando un agente riporta le rocce alla base.
- Mentre un agente segue un sentiero verso la sorgente di rocce, raccoglie le briciole rendendo il sentiero più sottile.
- Dopo che un po' di agenti avranno seguito il sentiero senza trovare campioni il sentiero sarà scomparso.

Viene rimossa la regola tre e aggiunte:

6. Se un agente sta portando dei campioni e non è nella base, lascia cadere 2 briciole e va su verso il gradiente.
7. Se un agente percepisce delle briciole ne prende una e va giù verso il gradiente.

Note:-

L'ordine dei behaviours è 1, 2, 6, 4, 7, 5.

Limiti degli agenti reattivi:

- Agenti senza modello devono avere sufficienti informazioni locali.
- Se le decisioni sono locali gli agenti hanno una visione a breve termine.
- Difficili da realizzare.
- È difficile ingegnerizzare agenti specifici.
- È difficile definire agenti con un gran numero di behaviours.

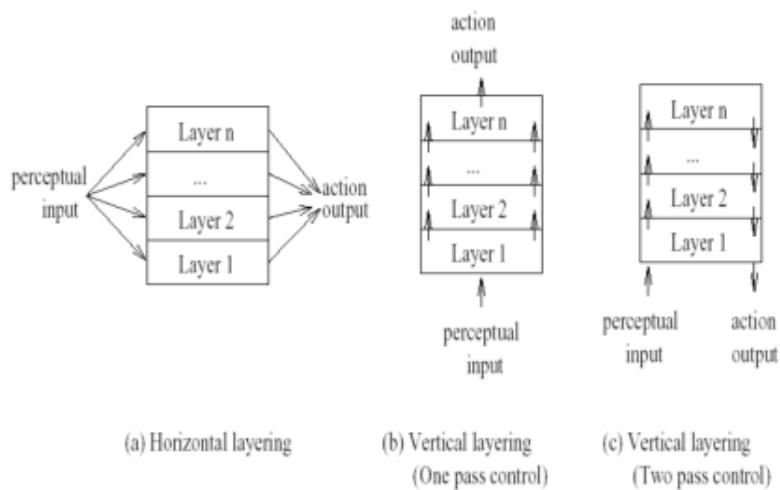
3.2.4 Architetture Ibride

- Molti ricercatori hanno sostenuto che né un approccio completamente deliberativo né un approccio completamente reattivo sono adatti a costruire agenti.
- Ci sono stati suggerimenti di utilizzo di sistemi ibridi.
- Un approccio è quello di costruire un agente con due o più sottosistemi:
 - Uno *deliberativo* contenente un modello simbolico del mondo, che sviluppa piani e prende decisioni nel modo proposto dall'IA simbolica.
 - Uno *reattivo* capace di reagire a eventi senza ragionamenti complessi.

Modelli ibridi:

- Solitamente viene data precedenza ai componenti reattivi.
- Questo tipo di struttura porta a un'architettura a strati.
- I sottosistemi di controllo di un agente sono organizzati in una gerarchia, con livelli più alti che trattano l'informazione a crescenti livelli di astrazione.
- Un problema è il tipo di schema di controllo in cui inserire i sottosistemi.
- Sono stati proposti:
 - *Horizontal layering*: tutti gli strati sono direttamente connessi all'input e all'output.
 - *Vertical layering*: input dei sensori e output delle azioni sono gestiti al massimo da uno strato ciascuno.

m possible actions suggested by each layer, *n* layers



- (a) m^n interactions: Introduces bottleneck in central control system
- (b) and (c) $m^2(n - 1)$ interactions: Not fault tolerant to layer failure

Figure 3.11: Schema delle architetture ibride.

Note:-

Esempi di architetture ibride sono TOURINGMACHINES e Stanley.

4

Sistemi Multiagente

Domanda 4.1

Perché sistemi distribuiti di agenti?

- Le soluzioni centralizzate sono spesso impraticabili perché i sistemi e i dati coinvolti appartengono a organizzazioni indipendenti.
- Le informazioni coinvolte sono distribuite e risiedono in sistemi informativi di grosse dimensioni e complessi sotto diversi punti di vista:
 - Distribuiti geograficamente.
 - Composti da molte parti indipendenti.
 - Contenuti di grandi dimensioni.
 - Coprono una porzione maggiore del dominio considerato.

Ci sono quattro tecniche principali per affrontare la **dimensione** e la **complessità** di un sistema informativo:

- Modularità.
- Distribuzione.
- Astrazione.
- Intelligenza.

Note:-

L'uso di moduli distribuiti Intelligenti combinati a queste tecniche produce un approccio di **intelligenza artificiale distribuita** (DAI). Gli agenti fanno parte di questo approccio.

Definizione 4.0.1: Sistemi Autonomi di Agenti

Per lo sviluppo di soluzioni globali e distribuite gli agenti necessitano di essere eseguiti in maniera autonoma e sviluppati indipendentemente. Gli agenti coordinano, cooperano e, eventualmente, competono per la soluzione di problemi, condividendo capacità e lavorando in parallelo. Questo porta ai sistemi Multiagente.

Note:-

Sistemi autonomi rappresentano soluzioni modulari e riutilizzabili.

Domanda 4.2

Perché comunicare?

- Gli agenti operano in ambienti che contengono altri agenti.
- Un sistema multiagente è un sistema che contiene agenti che interagiscono tra di loro attraverso la *comunicazione*.
- Hanno le seguenti caratteristiche:
 - Forniscono un'infrastruttura specifica per la comunicazione e l'utilizzo di protocolli di interazione.
 - Sono progettati per essere *aperti* e *distribuiti*.
 - Gli agenti ospitati sono *autonomi*, *self-interested* o *cooperativi*.
- Le infrastrutture includono:
 - *Protocolli di comunicazione*: per permettere agli agenti di comprendere i messaggi scambiati.
 - *Protocolli di interazione*: per permettere agli agenti di svolgere conversazioni, ossia scambi strutturati di messaggi.
- Gli agenti coordinano le loro azioni e i loro comportamenti. Quest'abilità è parte della percezione (ricevere messaggi) e parte delle azioni (inviare messaggi).
- La *coordinazione* è una proprietà di un sistema di agenti che eseguono attività in un ambiente condiviso.
- La *cooperazione* è coordinazione tra agenti non antagonisti.
- La *negoziazione* è coordinazione tra agenti antagonisti, competitivi o self-interested.

Un protocollo di comunicazione potrebbe includere i seguenti tipi di messaggio:

- Proposta di esecuzione di un'azione.
- Accettare una proposta.
- Rifiutare una proposta.
- Ritirare una proposta.
- Esprimere disaccordo rispetto a una proposta.
- Proporre una differente esecuzione.

4.1 Agent Communication Languages

Note:-

Gli oggetti lo fanno perché devono. Gli agenti lo fanno perché vogliono.

4.1.1 Atti Comunicativi

Definizione 4.1.1: Azioni Comunicative

Gli agenti non possono forzare altri agenti a eseguire qualche azione (come nel paradigma object-oriented). Possono eseguire *azioni comunicative* al fine di influenzare altri agenti in modo opportuno.

- Il linguaggio parlato umano è usato come modello per la comunicazione tra agenti.
- Austin e Searle introducono la teoria nota come *speech act theory* che tratta la comunicazione come azioni (requesting, informing, replying, etc.).
- Gli atti comunicativi sono spiegati in termini di intenzioni degli agenti, con riferimento a beliefs, desires, intentions e altre modalità.
- Un atto comunicativo ha tre aspetti:
 - *Locuzione*: l'atto fisico compiuto da chi parla, la produzione di enunciati grammaticali.
 - *Illocuzione*: il significato inteso dell'enunciato da parte del parlante, quello che il parlante desidera esprimere.
 - *Perlocuzione*: l'azione che risulta dalla locuzione.

Note:-

La speech act theory usa il termine *performativa* per indicare la forza illocutoria dell'enunciato.

Condizioni per il successo (Austin's felicity conditions):

- Deve esistere una procedura convenzionale accettata per la performativa, le circostanze e le persone coinvolte devono essere come specificato nella procedura.
- La procedura deve essere eseguita correttamente e completamente.
- L'atto deve essere sincero e la comprensione deve essere completa.

Un Agent Communication Language (ACL):

- Fornisce agli agenti i mezzi per scambiarsi informazioni e conoscenza.
- È solitamente a un livello a un livello più alto rispetto gli strumenti di comunicazione per i sistemi distribuiti, come remote procedure call o method invocation.
- Tratta con proposizioni, regole, azioni.
- Un messaggio descrive uno stato desiderato attraverso un linguaggio dichiarativo.

4.1.2 KQML e FIPA

Definizione 4.1.2: Knowledge Sharing Effort

Il Knowledge Sharing Effort (KSE) fu avviato dal DARPA intorno al 1990. L'obiettivo è sviluppare tecniche, metodologie e strumenti software per condivisione e riutilizzo della conoscenza.

Note:-

La condivisione della conoscenza richiede la comunicazione che a sua volta richiede un linguaggio.

Sviluppo le seguenti componenti:

- **KQML:** un linguaggio di interazione di alto livello.
- **KIF:** un linguaggio logico basato sulla logica del primordine, per esprimere proprietà di oggetti in una base di conoscenza.
- **Ontolingua:** un linguaggio per definire ontologie condivise, permettendo di dare significati uniformi su applicazioni diverse agli stessi concetti.

È utilizzato per dichiarare:

- Proprietà di cose.
- Relazioni tra cose.
- Proprietà generali.

Definizione 4.1.3: Knowledge Query and Manipulation Language (KQML)

KQML è un linguaggio di comunicazione di alto livello e indipendente da:

- Meccanismo di trasporto.
- Il linguaggio in cui è espresso il contenuto.
- L'ontologia utilizzata.

Osservazioni 4.1.1

- Un messaggio KQML specifica il tipo di messaggio (performativa).
- KQML ignora la porzione di messaggio che fa riferimento al contenuto.
- La sintassi di un messaggio KQML si basa su una notazione a lista simile a quella di Lisp.
- Consiste in una performativa seguita da un numero di coppie "parola chiave/valore".

Parole chiavi riservate di KQML:

- sender: il mittente della performativa.
- receiver: il destinatario della performativa.
- reply-with: se il mittente si aspetta una risposta.
- in-reply-to: se il mittente si aspetta una risposta.
- language: il linguaggio di rappresentazione del content.
- ontology: l'ontologia assunta dal parametro content.
- content: il contenuto del messaggio.

Alcune parole performative di KQML:

- achieve: il sender vuole che il receiver esegua qualcosa.
- advertise: il sender vuole una delle risposte del receiver alla domanda espressa dal content.
- ask-one: il sender vuole una delle risposte del receiver alla domanda espressa dal content.
- ask-all: il sender vuole tutte le risposte del receiver alla domanda espressa dal content.
- reply: si comunica una risposta attesa.
- sorry: il sender non può fornire una risposta più specifica.
- tell: il sender informa il receiver che conosce il content.

Corollario 4.1.1 Communication Facilitators

KQML introduce una classe speciale di agenti chiamati communication facilitators che dispongono di un insieme di performatives per inoltrare messaggi, trovare servizi, etc.

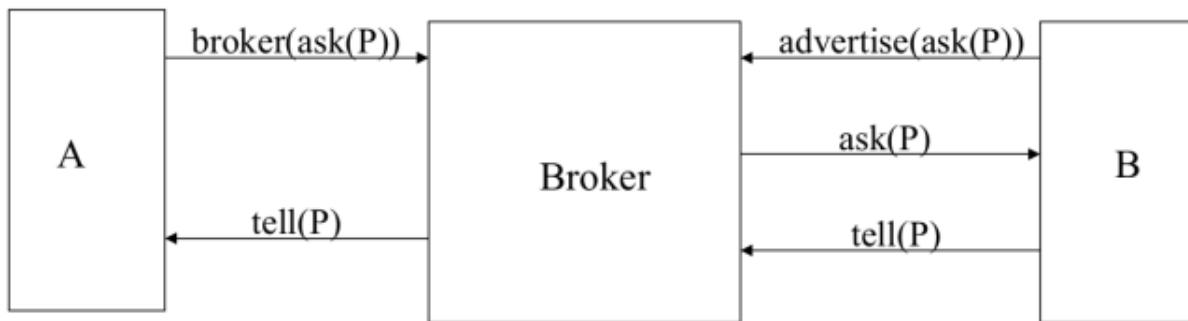


Figure 4.1: Communication Facilitators.

Corollario 4.1.2 Semantica di KQML

Precondizioni, postcondizioni e condizioni di completamento descrivono lo stato degli agenti in un linguaggio basato su stati mentali espressi mediante attitudini e descrivono azioni.

Note:-

Non è fornito un modello semantico per gli stati mentali espressi mediante attitudini.

Definizione 4.1.4: FIPA ACL

La Foundation for Intelligent Physical Agents (FIPA) è stata fondata con il fine di produrre degli standard per il software per agenti interagenti ed eterogenei e sistemi basati su agenti. FIPA opera attraverso una collaborazione aperta internazionale di organizzazioni associate, aziende, centri di ricerca e università. FIPA ACL è un linguaggio per comunicazione per agenti simile a KQML.

KWML vs. FIPA ACL:

- I due linguaggi sono simili alla base.
- Non sono vincolati a un linguaggio per esprimere il contenuto.
- La differenza principale è la semantica.
- Un'altra differenza è la mancanza di primitive di tipo facilitators in FIPA ACL.

Corollario 4.1.3 Semantica di FIPA ACL

Il Semantic Language (SL) è un linguaggio formale per definire la semantica di FIPA ACL. È una logica multimodale quantificata con operatori modali. Per permettere il ragionamento su azioni, l'universo del discorso coinvolge sequenze di eventi.

- I seguenti operatori sono introdotti per ragionare sulle azioni:
 - $\text{Feasible}(a, \phi)$: a può occorrere e se occorre, ϕ sarà vera dopo tale occorrenza.
 - $\text{Done}(a, \phi)$: a è appenaoccorsa e ϕ è vera dopo tale occorrenza.
 - $\text{Agent}(i, a)$: i è il solo agente che esegue l'azione a .
- Sulla base di beliefs, choices ed eventi viene definito il concetto di *goal persistente* $PG_{i\phi}$.
- L'intenzione $I_{i\phi}$ è definita come un goal persistente che impone all'agente di agire.
- La semantica degli atti comunicativi è specificata come un insieme di formule SL che descrivono le precondizioni di fattibilità e gli effetti razionali.
- *Feasibility Conditions*: condizioni che devono valere per il mittente per eseguire in modo appropriato gli atti comunicativi.
- *Rational Effects*: gli effetti che un agente può aspettarsi che occorrono come risultato dell'esecuzione di un'azione. All'agente destinatario non è richiesto di garantire che occorra l'effetto atteso.
- Per *conformità* di FIPA ACL si intende che quando un agente A esegue un atto comunicativo c , le $FP(c)$ per A devono valere. I $RE(c)$ non sono garantiti e sono irrilevanti al fine della conformità.

Test di conformità:

- Sintattico: l'obiettivo è determinare se un agente rispetta la sintassi del linguaggio ogni volta che comunica.
- Semantica: l'obiettivo è determinare se un agente rispetta la semantica del linguaggio ogni volta che comunica.

Note:-

Wooldridge propose un approccio basato sulla riduzione a un problema di verifica di proprietà di un programma, ma la maggior parte dei vincoli sono espressi sul mittente e questo comporta la necessità di parlare di stati mentali che non sono sempre desumibili.

4.2 Semantica Sociale per la Comunicazione

La semantica basata su stati mentali è adeguata per sistemi di agenti cooperanti ma ha alcuni problemi per quanto riguarda i sistemi ad agenti composti da agenti eterogenei e competitivi.

Note:-

È impossibile fidarsi completamente di altri agenti o fare forti assunzioni a proposito dello stato interno degli agenti e sul loro modo di agire.

L'approccio sociale:

- Considera le conseguenze sociali di eseguire un atto comunicativo.
- Riconosce che la comunicazione è inerentemente pubblica e che dipende dal contesto sociale degli agenti.

4.2.1 Un Buon ACL

Le caratteristiche di un buon ACL:

- *Formal*: chiarezza della specifica per guidare al meglio colui che realizza un sistema.
- *Declarative*: descrivere cosa piuttosto che come.
- *Verifiable*: determinare se un agente sta agendo in accordo con una data semantica.
- *Meaningful*: trattare la comunicazione secondo il suo significato e non come arbitrari token che devono essere ordinati in una qualche maniera.

Semantica per ACL basata su stati mentali e verifica:

- Rispettare un protocollo: un protocollo è rispettato se e solo se l'interazione specificata/desiderata occorre durante l'esecuzione.
- Verificare un protocollo: necessita di una semantica basata su fatti osservabili.
- La semantica basata su stati mentali è particolarmente adatta per ragionare in modo statico sulle proprietà del protocollo e sulle politiche adottate dagli agenti.
- Non è possibile fare introspezione sugli agenti perché solitamente lo stato mentale degli agenti non è accessibile.

È necessaria una semantica che sia:

- Indipendente dagli stati mentali dei partecipanti.
- Verificabile sia da un partecipante sia da una terza parte che deve monitorare l'interazione.
- Possibile rendere osservabile la violazione di una specifica da parte di un partecipante all'interazione.

Osservazioni 4.2.1

- L'approccio è basato sui practical commitment tra agenti: un agente (il debtor) si impegna verso un altro agente (il creditor) a rendere vero un qualche fatto o a eseguire una qualche azione.
- I vari atti illocutori possono essere visti in termini di commitment sociali che i partecipanti creano.

Note:-

La comunicazione nell'approccio sociale è inherentemente pubblica.

Commitment (per Castelfranchi):

- Un commitment sociale ha natura normativa e genera diritti.
- Cancellare o ritirare un commitment viola un'obbligazione non implica un'informazione verso la controparte, ma frustra delle aspettative e dei diritti che l'agente ha creato.

Commitment (per Singh):

- Include una nozione di *contesto sociale* nella definizione di commitment (il team in cui l'agente partecipa e in cui comunica).
- Permette *metacommitment* per catturare un'ampia varietà di relazioni legali e sociali in un unico framework.
- Il comportamento degli agenti è influenzato dai commitment perché l'assunzione è che gli agenti rispettino gli impegni presi.
- I commitment sono manipolabili dagli agenti.

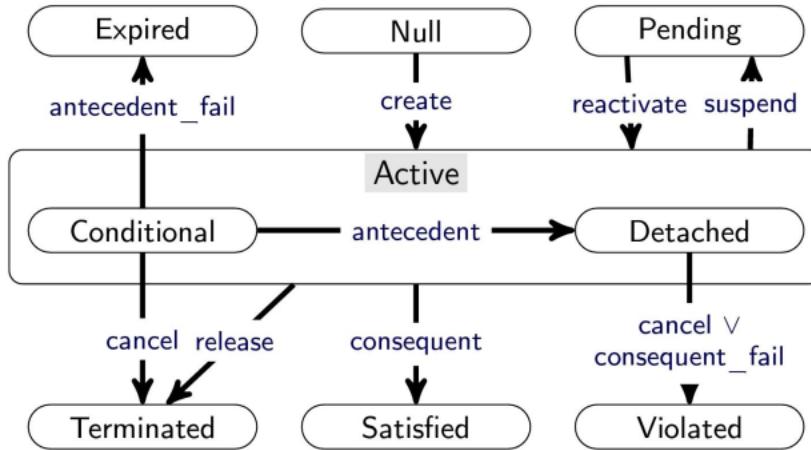


Figure 4.2: Ciclo di vita dei commitment per Singh.

4.3 Protocolli di Interazione

- Gli agenti non possono prendere parte a un dialogo semplicemente scambiandosi messaggi ACL.
- L'analisi di numerose conversazioni umane mostra che sono presenti dei pattern ricorrenti nelle conversazioni.
- La semantica mentalistica per gli atti comunicativi è troppo complessa per determinare una possibile risposta a un messaggio.
- Un agente ha bisogno di *conversation policies / Protocolli di interazione*.
- In generale i protocolli specificano il contenuto della comunicazione.

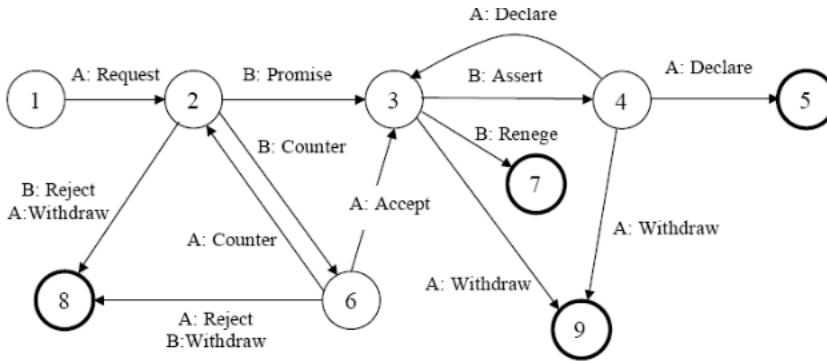


Figure 4.3: Protocollo rappresentato come DFA.

Definizione 4.3.1: Contract Net Protocol (task sharing)

Un protocollo per realizzare sistemi di agenti cooperanti. È modellato sul meccanismo di contrattazione utilizzato per governare gli scambi di merci e servizi. Un agente che desidera che un task sia risolto viene chiamato manager (initiator) e gli agenti che potrebbero risolvere il task sono chiamati contractors (participants).

Un agente ha un goal e realizza che:

- O non è in grado di soddisfarlo in autonomia, ma non ne ha le capacità.
- O preferirebbe non soddisfarlo in autonomia, per ottenere una soluzione migliore.

Task Sharing:

- Manager:
 - Annuncia un task che necessita di essere risolto.
 - Riceve e valuta le offerte dei possibili contractors.
 - Sceglie un contractor e lo informa.
 - Riceve il risultato.
- Contractor:
 - Ricever gli annunci di task dal manager.
 - Valuta il task.
 - Risponde declinando o facendo un'offerta (bidding).
 - Esegue il task se l'offerta è stata accettata.
 - Riporta il risultato (expediting).
- Un contractor per un task può agire a sua volta da manager sollecitando l'aiuto di altri agenti per risolvere parte di quel task.
- Una deadline può essere fissata per effettuare offerte da parte dei contractors.

4.3.1 Protocolli a commitment

- I protocolli possono essere specificati come un insieme di commitment piuttosto che come DFA.
- Gli agenti giocano ruoli diversi nella società e i ruoli definiscono l'associazione con gli impegni sociali.
- In generale, gli agenti possono gestire i propri impegni manipolandoli o cancellandoli.
- Poiché i requisiti di un protocollo vengono espressi solamente attraverso commitment, gli agenti possono essere conformi sulla base della loro comunicazione.
- L'idea è di catturare la relazione "count-as" che descrive il significato dell'azione.
- Il solo vincolo che un protocollo a commitment deve soddisfare per avere successo è che tutti i componenti siano discharged.

Definizione 4.3.2: Run del Protocollo a Commitment

Una run del protocollo a commitment è una sequenza finita di azioni che portano in uno stato in cui tutti i commitment sono discharged, ossia non ci siano commitment pendenti.

Osservazioni 4.3.1

- Data la precedente specifica delle azioni, si può costruire delle sequenze che permettono di raggiungere uno stato in cui non ci siano commitment pendenti.
- Verifica: data una sequenza di azioni eseguite dagli agenti, si può verificare se essa sia o non sia una sequenza ammessa dal protocollo.

4.3.2 Information Protocols

Note:-

Proposta recente di Singh, presentati nel 2011.

Proprietà dei partecipanti all'interazione:

- Autonomia.
- Miopia:
 - Tutte le scelte devono essere locali.
 - La correttezza non deve basarsi su interazioni future.
- Eterogeneità:
 - Stato locale:
 - * Pubblico od osservabile.
 - * Deve essere rivelato, per correttezza.
 - Stato interno:
 - * Privato.
 - * Non deve mai essere rivelato, per evitare falsi accoppiamenti.
- Rappresentazione condivisa dello stato locale:
 - Eseguita tramite messaggistica.

Definizione 4.3.3: Blindingly Simple Protocol Language (BSPL)

Ci sono due notazioni sintattiche:

- Dichiarare uno schema di messaggio come protocollo atomico.
- Dichiarare un protocollo come un insieme di riferimenti a protocolli.

I parametri sono centrali:

- Base per esprimere il significato in termini di binding.
- Specifica non ambigua.
- Catturano:
 - La progressione della conoscenza di un ruolo.
 - La *completeness* di un *enactment* di protocollo.
 - L'unicità degli enactment tramite chiavi (key).

Corollario 4.3.1 Key

Le Key:

- Tutti i parametri chiave insieme formano una chiave.
- Almeno un parametro chiave per protocollo.
- Almeno un parametro chiave in comune durante messaggi e riferimenti ad altri protocolli.
- La chiave di un protocollo fornisce una base per l'unicità dei suoi enactment.

Ornamenti dei parametri:

- in: informazioni che devono essere fornite per istanziare un protocollo.
- out: informazioni generate dalle istanze del protocollo.
- nil: informazioni assenti dall'istanza di protocollo.

Idee principali dei protocolli BSPL:

- Dichiarativo:
 - Nessun flusso o stato di controllo.
- Basato sulle informazioni:
 - Specifica la computazione dell'informazione distribuita:
 - * La specifica di un messaggio è esso stesso un protocollo.
 - Specificano tramite parametri.
- La causalità è esplicita:
 - I messaggi che un agente può inviare dipendono da ciò che conosce.
- Integrity:
 - L'agente invia solo messaggi che preservano la coerenza delle informazioni.
 - Tramite vincoli di chiave.
- Messaggistica sincrona.
- Non richiede alcun ordinamento dall'infrastruttura.
- Composizione e verifica.

Domanda 4.3

Quando un messaggio è valido? Quale effetto ha sulla conoscenza locale di un ruolo?

- Out: crea e trasmette conoscenza.
- In: trasmette conoscenza.
- Le ripetizioni attraverso percorsi multipli sono innocue e superflue.

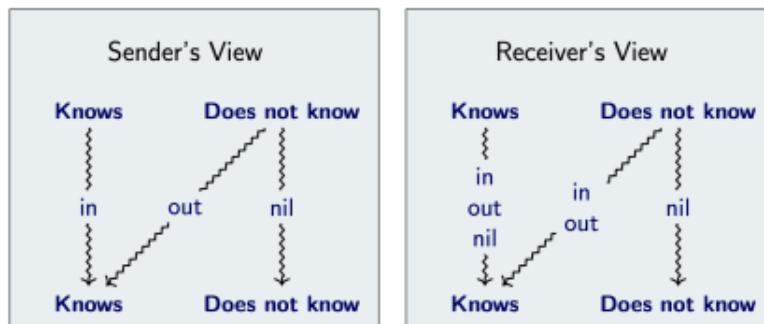


Figure 4.4: Schema di BSPL.

Struttura causale come vincoli temporali:

- *Ricezione*: se un messaggio viene ricevuto, è stato precedentemente inviato.
- *Trasmissione delle informazioni*:
 - I parametri "in" si verificano prima del messaggio.
 - I parametri "out" si verificano simultaneamente al messaggio.
- *Ricezione delle informazioni*: qualsiasi parametro "in" o "out" si verifica prima o simultaneamente al messaggio.
- *Minimalità delle informazioni*: se un ruolo osserva un parametro deve essere simultaneamente a un messaggio inviato o ricevuto.
- *Ordinamento*: se un ruolo invia due messaggi li osserva in ordine.

Implementazione di BSPL:

- Per ogni ruolo:
 - Per ogni messaggio che si invia o che si riceve:
 - * Mantieni una relazione locale dello stesso schema del messaggio.
- Ricevi e archivia ogni messaggio ricevuto:
 - Che non sia un duplicato.
 - Che i vincoli di integrità siano rispettati.
 - Cancella le sessioni scadute.
- Invia qualsiasi messaggio univoco purché:
 - I binding dei parametri concordino con i binding precedenti per le stesse chiavi per i parametri "in".
 - Non esistano binding per i parametri "out" e "nil".

Note:-

Ogni interazione è caratterizzata solo in termini di informazione: causalità esplicita, chiavi, integrità, immutabilità.

4.3.3 Modelli di Coordinazione

Definizione 4.3.4: Blackboard

Un approccio alternativo alla progettazione, realizzazione e gestione di sistemi multiagente. È basato su modelli di coordinazione che ad alto livello sono finalizzati a regolare il comportamento delle diverse componenti di un sistema. Un Blackboard system è un approccio basato sull'architettura a lavagna dove una base di conoscenza comune è iterativamente aggiornata da sorgenti di conoscenza, partendo da una specifica di un problema alla soluzione.

Note:-

Ogni sorgente di conoscenza aggiorna la blackboard con una soluzione parziale.

Definizione 4.3.5: Norme

Le norme e le leggi sociali sono uno dei meccanismi più diffusi per coordinare le attività. Una norma esprime uno schema di comportamento atteso. Le norme possono essere enforced o meno e possono emergere dal sistema o essere progettate.

5

JADE e JASON

5.1 JADE

5.1.1 FIPA

Definizione 5.1.1: FIPA

Consorzio per la standardizzazione di sistemi ad agenti. Il suo obiettivo era quello di promuovere tecnologie interoperabili: sistemi di agenti intelligenti che lavorano insieme.

Domanda 5.1

Cosa rientrò nelle competenze di FIPA?

- Gestione del ciclo di vita di un agente.
- Come trasportare un messaggio.
- La struttura di un messaggio.
- Protocolli di interazione tra agenti.
- Ontologie e sicurezza.

Note:-

Gli agenti sono al di fuori degli obiettivi di FIPA.

Caratteristiche che vengono assunte per gli agenti:

- Autonomi.
- Reattivi.
- Proattivi.
- Goal-driven.
- Sociali.
- Adattivi.
- Cognitivi.

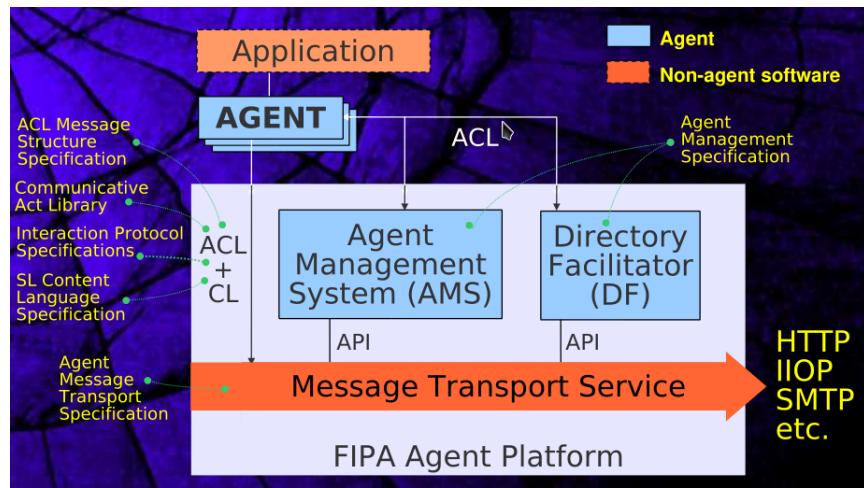


Figure 5.1: Piattaforma FIPA per agenti.

Corollario 5.1.1 Agent Management

Gli agenti hanno una descrizione di sé stessi e dei servizi che vengono offerti. L'Agent Management system è a sua volta un agente con cui si interagisce con scambi di messaggi. In jade si può accedere così oppure accedere direttamente agli oggetti Java.

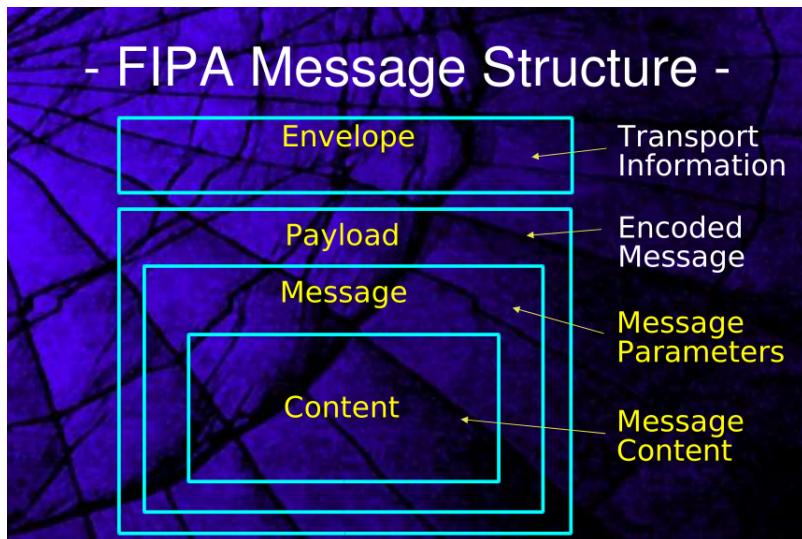


Figure 5.2: Specifica per i messaggi.

5.1.2 Introduzione a JADE

Domanda 5.2

Che cos'è JADE?

Risposta: JADE è la piattaforma, all'interno del consorzio FIPA, che implementa lo standard sopra citato. Si presenta come codice Java puro. Offre una serie di librerie per agenti e un runtime environment per gli agenti creati. L'obiettivo è quello di "nascondere" FIPA al programmatore.

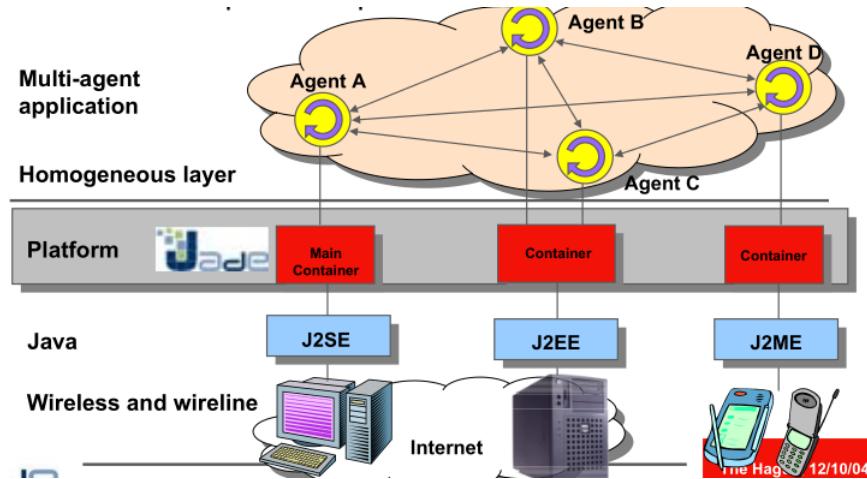


Figure 5.3: Modello architetturale di JADE.

Caratteristiche importanti:

- *HelloWorld Agent:*

- Un tipo di agente è creato estendendo `jade.core.Agent` e ridefinendo il metodo `setup()`.
- Ogni agente è identificato da un AID (nome univoco e qualche indirizzo).
- A ogni agente è assegnato uno e un solo thread (per evitare deadlock).

- *Local names, GUID and addresses:*

- Il nome completo di un agente deve essere globalmente unico.
- In una singola piattaforma JADE ci si riferisce agli agenti solamente mediante il loro nome.
- È possibile creare AID.
- Si possono passare argomenti agli agenti.

- *Terminazione:*

- Un agente termina quando è chiamato il metodo `doDelete()`. Questo metodo non cancella all'istante, ma segnala che l'agente deve essere eliminato (solo dopo aver completato la `setup()`).
- Nella terminazione è chiamato il metodo `takeDown()` per fare operazioni di clean-up.

Definizione 5.1.2: Classe Behaviour

Il lavoro di un agente è eseguito dalle sue "behaviours". Per far sì che un agente esegua un task è sufficiente definire una behaviour e aggiungerla all'agente.

Ogni behaviour deve implementare:

- `void action()`: cosa la behaviour fa.
- `boolean done()`: quando la behaviour finisce.

Note:-

Un agente può eseguire più behaviours in parallelo, il loro scheduling è cooperativo e tutti avvengono nello stesso thread java.

The agent execution model

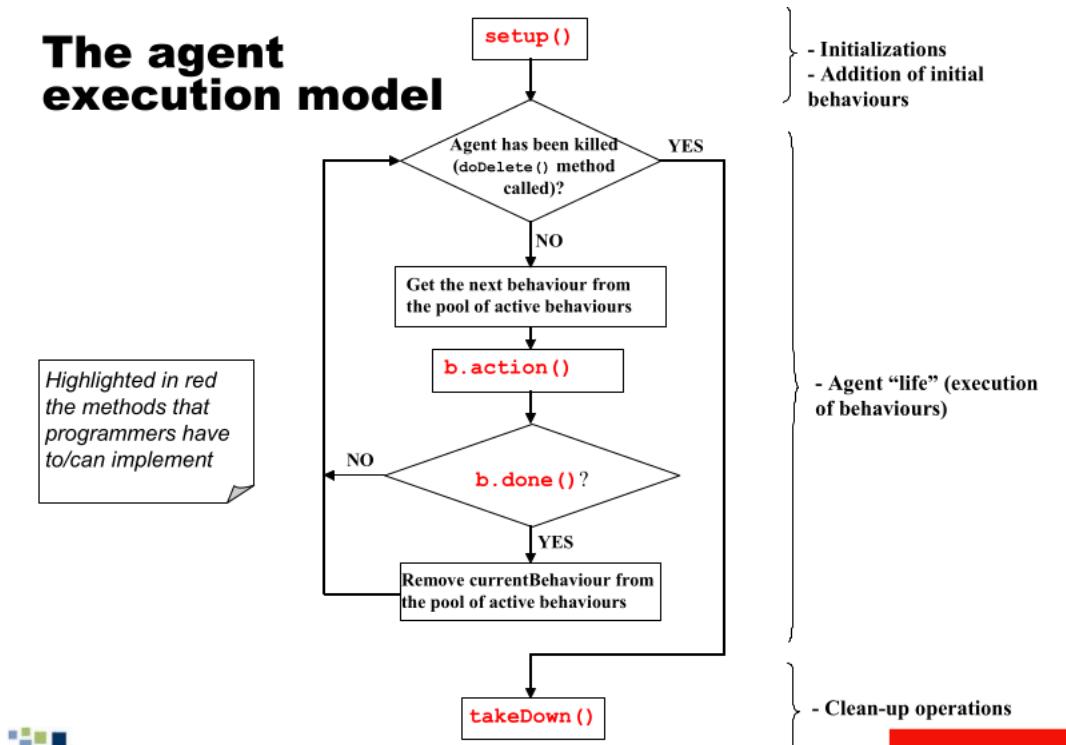


Figure 5.4: Modello di esecuzione di un agente.

Tipi di behaviour:

- *One shot:*
 - La loro `action()` viene eseguita solo una volta.
 - `done()` restituisce sempre true (non si deve reimplementare).
- *Cyclic:*
 - Non si completano mai, `action()` effettua sempre le stesse operazioni.
 - `done()` restituisce sempre false (non si deve reimplementare).
- *Complex:*
 - Hanno uno stato (e.g. DFA) e a seconda di esso svolgono operazioni differenti.
 - Completano quando una data condizione diventa vera.

Altri tipi:

- *WakerBehaviour:*
 - Al posto di `action()` e `done()` si deve ridefinire `onWake()` che verrà eseguito dopo un certo timeout.
 - Subito dopo l'esecuzione il behaviour è completato.
- *TickerBehaviour:*
 - Al posto di `action()` e `done()` si deve ridefinire `onTick()` che verrà eseguito periodicamente.
 - L'esecuzione va avanti all'infinito o finché non viene invocato `stop()`.
 - Solitamente si usa per aggiungere periodicamente delle behaviours.

Other shits on behaviours:

- `onStart()`: invocato prima della prima esecuzione di `action()`. Sono operazioni che devono essere eseguite prima di ogni behaviour.
- `onEnd()`: invocato dopo che `done()` restituisce true. Sono operazioni che devono essere eseguite alla fine di ogni behaviour.
- `removeBehaviour()`: well, prova a indovinare cosa fa... RIMUOVE LA FUCKING BEHAVIOUR DAL POOL DI UN AGENTE¹.
- Quando il pool di behaviours attive di un agente è vuoto esso entra in IDLE a il suo thread va in sleep.

5.1.3 Comunicazione tra Agenti

Definizione 5.1.3: Modello di Comunicazione

Il modello di comunicazione è basato sul passaggio asincrono di messaggi. I messaggi sono definiti dal linguaggio ACL (FIPA).

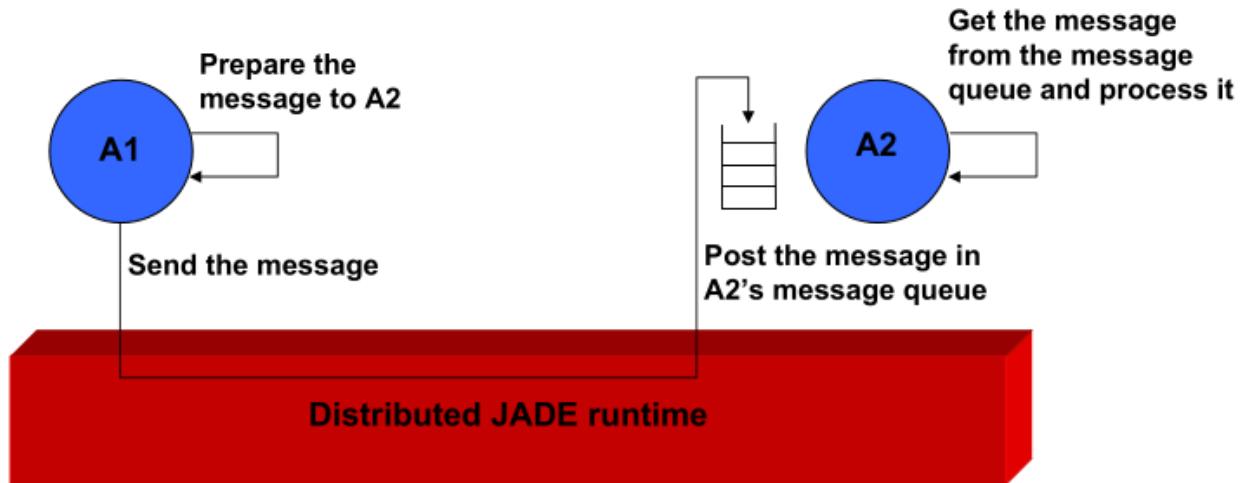


Figure 5.5: Modello di comunicazione.

I messaggi:

- Invio: inviare i messaggi è la creazione di un messaggio ACL e l'invio del messaggio tramite `send(ACLMessage)`.
- Ricezione: per leggere i messaggi si usa `receive()`.

Bloccare una behaviour mentre si aspetta per un messaggio:

- Il metodo `block()` non blocca il thread, ma sposta il behaviour dal pool di comportamenti attivi in un pool di comportamenti bloccati.
- Non è una chiamata bloccante.
- Tutte le behaviours vengono riattivate quando arriva un nuovo messaggio.

¹Non chiama `onEnd()`.

Leggere selettivamente una coda di messaggi:

- `receive()` restituisce il primo messaggio nella coda di messaggi e lo rimuove.
- Se ci sono più behaviours che aspettano un messaggio una può rubare il messaggio a un'altra behaviour interessata.
- Per evitare questo è possibile leggere solo certi tipi di messaggi con certe caratteristiche.

Ricevere un messaggio in "modalità bloccante":

- Gli agenti hanno anche un metodo `blockingReceive()` che restituisce quando c'è un messaggio nella coda di messaggi.
- Questa è una chiamata bloccante, per cui nessuna altra behaviour può eseguire finché la behaviour che ha invocato il metodo non ha ricevuto un messaggio.

Definizione 5.1.4: Pagine Gialle

Un servizio che consente agli agenti di pubblicare uno o più servizi che forniscono in modo che gli altri agenti possano trovarli.

Note:-

In JADE ci si può accedere come oggetto remoto o interagendo come fosse un agente con messaggi ACL (modalità imposta da FIPA).

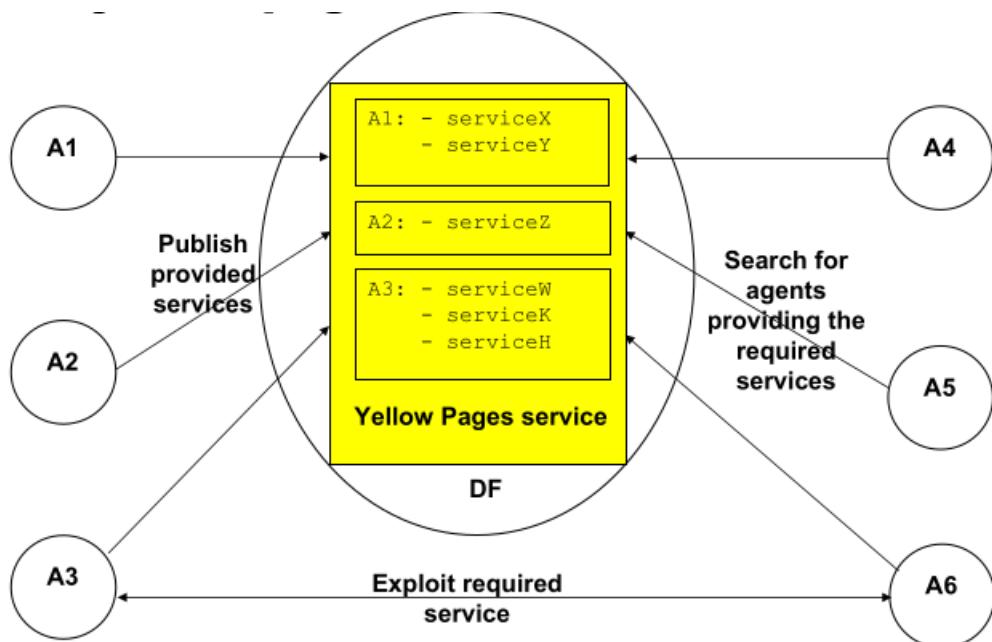


Figure 5.6: Servizio di pagine gialle.

5.1.4 Protocolli di Interazione

Definizione 5.1.5: Protocolli di Interazione

Hanno un insieme di tipi di messaggio (INFORM, REQUEST, PROPOSE, ...) che consentono lo scambio di specifiche sequenze predefinite di messaggi durante una conversazione.

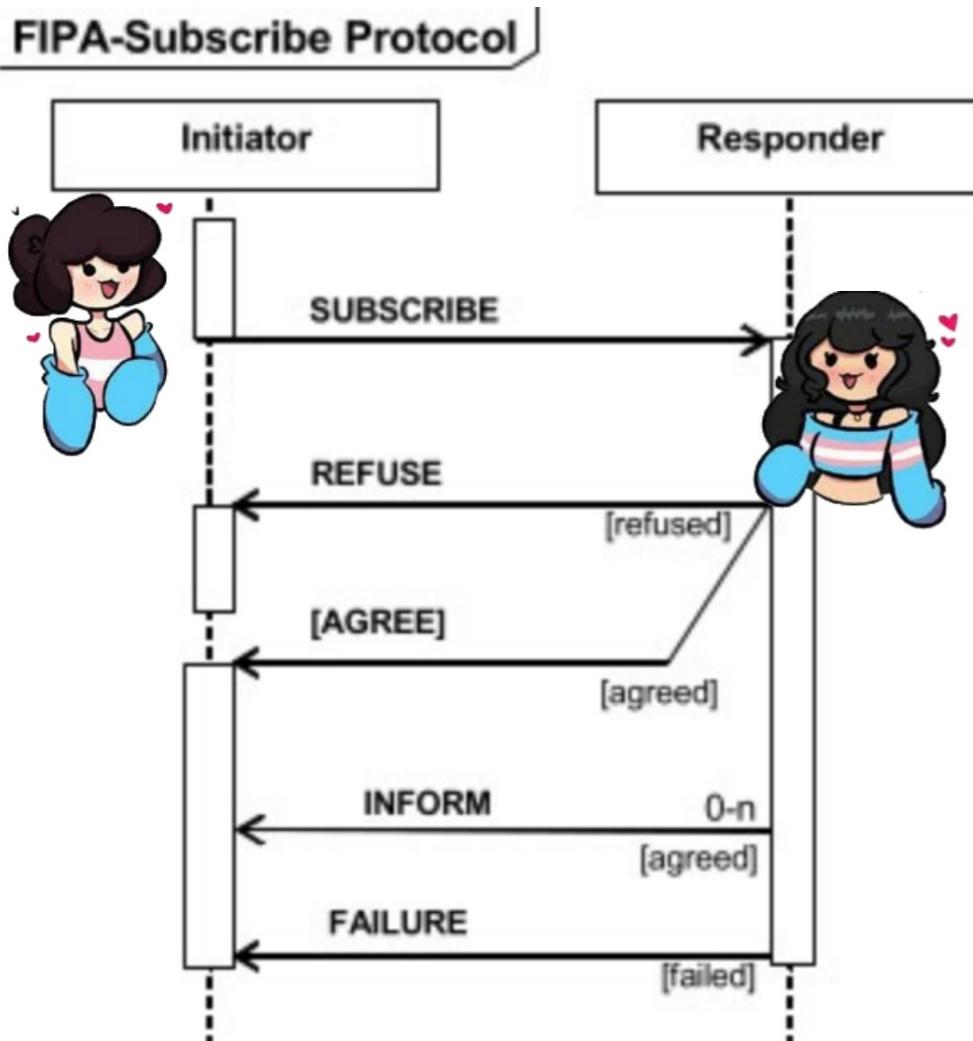


Figure 5.7: FIPA Protocol.

Alcuni protocolli comuni sono già presenti:

- Request.
- Contract-net.
- Subscribe.

Essi garantiscono:

- Che il flusso di messaggi sia compliant con il protocollo.
- Che se è presente un timeout esso sia corretto.

5.2 JASON

5.2.1 Introduzione

Definizione 5.2.1: AgentSpeak

Un linguaggio di programmazione per agenti che si propone di colmare il gap tra specifica teorica e implementazione di un agente.

Note:-

JASON è un'implementazione di AgentSpeak, scritta in java.

Osservazioni 5.2.1

- Gli agenti BDI vengono trattati sotto due punti di vista:
 - Specifica teorica.
 - Implementazione.
- Le implementazioni esistenti usano strutture dati per rappresentare *belief*, *desires* e *intentions*.
- JASON è un'estensione della programmazione logica (e.g. prolog).

Agenti BDI:

- Sistemi collocati in un ambiente dinamico, che muta nel tempo.
- Sistemi in grado di percepire informazioni provenienti dall'ambiente.
- Sistemi in grado di compiere delle azioni sulla base dei propri stati mentali, in particolare:
 - Belief: cattura le informazioni.
 - Desire: cattura le motivazioni.
 - Intention: cattura le decisioni.

5.2.2 AgentSpeak(L)

Domanda 5.3

Ma com'è fatto AgentSpeak(L)?

- Si parte da un BDI.
- Il sistema è PRS (Procedural Reasoning System) e dMARS (Distributed Multi-Agent Reasoning System).
- Si basa sulla logica del primordine con eventi e azioni.
- A differenza di BDI si vanno a scrivere programmi.
- Dettaglio dell'agente:
 - Current belief: modello di sé stesso, dell'ambiente e degli altri agenti.
 - Desires: stati che l'agente vuole raggiungere.
 - Intentions: adozione di programmi per soddisfare certi stimoli.

Il linguaggio AgentSpeak(L):

- Testuale.
- Possiede belief (predicati), piani, azioni, intentions, eventi e goals.
- Piani:
 - Context-sensitive.
 - Possono essere invocati dall'utente.
 - Decomposizione gerarchica dei goals.
 - Simili alle clausole di prolog.
- Usa un alfabeto composto da variabili, costanti, simboli funzionali, simboli predicativi, simboli di azione, connettivi, quantificatori e punteggiatura.
- I connettivi sono:
 - &: congiunzione.
 - not: negazione.
 - \leftarrow : implicazione.
 - !: achievement goal.
 - ?: test goal.
 - ;: sequenza.

Definizione 5.2.2: Belief

Siano $b(t)$ e $c(s)$ belief atoms, allora $b(t) \& c(s)$ e $\text{not } b(t)$ sono beliefs.

Corollario 5.2.1 Belief Atom

Sia b un simbolo predicativo e siano t_1, \dots, t_n termini, allora: $b(t_1, \dots, t_n)$ è un belief atom e si scrive anche $b(t)$.

Note:-

Un belief atom e la sua negazione sono detti belief literal. Se un belief atom è ground (senza variabili libere) si chiama belief base.

Definizione 5.2.3: Goal

Sia g un simbolo predicativo e siano t_1, \dots, t_n termini, allora: $!g(t_1, \dots, t_n)$ è un achievement goal, $?g(t_1, \dots, t_n)$ è un test goal.

Triggering event:

- Quando un agente acquisisce un nuovo goal oppure nota una modifica nell'ambiente può far scattare aggiunte/cancellazioni di goals o beliefs.
- L'aggiunta è rappresentata dall'operatore +.
- La rimozione dall'operatore -.

Definizione 5.2.4: Azione

Sia a un simbolo di azione e siano t_1, \dots, t_n termini, allora $a(t_1, \dots, t_n)$ è un'azione.

Piano:

- Un piano specifica il modo in cui un agente potrebbe raggiungere un determinato obiettivo.
- Un Triggering event specifica perché il piano è stato attivato.
- Il context specifica quali beliefs dovrebbero essere soddisfatti nel set di credenze dell'agente quando il piano scatta.
- Il body è una sequenza di goals o azioni e specifica:
 - I goals.
 - Le azioni.
- true rappresenta un componente vuoto.

Definizione 5.2.5: Piano

Siano:

- e un triggering event.
- b_1, \dots, b_m belief literals.
- h_1, \dots, h_n goals o azioni.

allora

$$e : b_1 \& \dots \& b_m \leftarrow h_1, \dots, h_n$$

è un piano.

Note:-

La parte sinistra della freccia è detta *head*, la parte destra della freccia è detta *body* (un body vuoto è true). La parte a destra dei ":", nell'head, è il contesto.

5.2.3 Semantica Operazionale

Un agente può essere visto come costituito da:

- Un insieme di beliefs B .
- Un insieme di piani P .
- Un insieme di intentions I .
- Un insieme di eventi E .
- Un insieme di azioni A .
- Un insieme di funzioni di selezione: S_ϵ, S_O, S_I .

Progetto di un agente:

- Il progettista specifica un agente scrivendo:
 - Un insieme di base belief.
 - Un insieme di piani.
- Si definiscono:
 - Un insieme di fatti.
 - Un insieme di regole.

Reasoning Cycle:

1. Viene generato un triggering event quando un agente nota una modifica dell'ambiente circostante (esterno) o quando un utente esterno chiede all'agente di raggiungere un goal (interno).
2. Gli eventi vengono inseriti in un set E .
3. Viene applicata la funzione di revisione dei beliefs:
 - S_ϵ seleziona in E un evento E_0 da processare.
 - E_0 viene rimosso da E e viene usato per unificare con i triggering events dei piani P .
 - I piani che unificano sono detti *piani rilevanti* e il loro unificatore si chiama *unificatore rilevante*.
4. L'unificatore rilevante è applicato al contesto dei piani rilevanti. Una *correct answer substitution* è ottenuta dal contesto.
5. Per alcuni piani le condizioni dei contesti risultano essere conseguenze logiche del set dei base beliefs B . Questi piani sono detti *piani applicabili* mediante un *unificatore applicabile* (derivato dalla composizione tra correct answer substitution e unificatore rilevante).
6. Dato un evento E_0 diversi piani risultano applicabili, per cui la funzione di selezione S_0 sceglie uno di questi (P_0). Applicando l'unificatore applicabile a P_0 si ottiene l'*intended means* in risposta al triggering event. Ogni intention è uno stack di piani parzialmente istanziati (intention frames).
7. A seconda del tipo di evento:
 - Esterno: l'intended means è usato per creare una nuova intention che viene aggiunta a I .
 - Interno: l'intended means è inserito in cima all'intention esistente che ha fatto scattare l'evento.
8. Quando l'agente esegue un intention:
 - Eseguire un achievement goal equivale a generare un evento interno per aggiungere il goal alla corrente intention.
 - Eseguire un test goal equivale a trovare una sostituzione per il goal che lo renda una conseguenza logica dei base belief. Se viene trovata una sostituzione il test goal è rimosso.
 - Eseguire un'azione equivale ad aggiungerla al set di azioni A e rimuoverla dal corpo.
9. L'agente torna a valutare il set degli eventi E e il ciclo ricomincia fino a quando il set E è vuoto o non è possibile eseguire altre intentions.

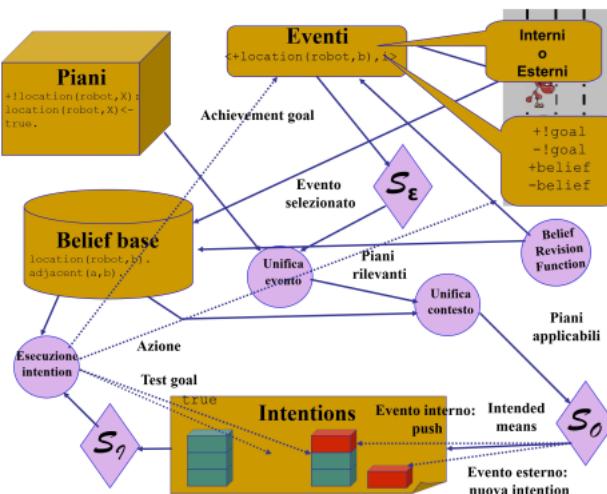


Figure 5.8: Reasoning Cycle.

Definizione 5.2.6: Intention

I è l'insieme delle intentions. Ogni intention è uno stack di piani parzialmente istanziati, ossia piani dove alcune variabili sono state istanziate. Un'intention è denotata come: $[p_1|p_2|\dots|p_z]$

Note:-

Caso particolare (true intention): $[+!true: true \leftarrow true]$

Definizione 5.2.7: Evento

L'insieme E è l'insieme degli eventi. Ogni evento è una coppia $\langle e, i \rangle$ dove:

- e è un triggering event.
- i è una intention.

Note:-

Se i è la true intention T , l'evento è esterno, altrimenti è interno.

Ancora sui piani:

- Scelto un evento $\langle d, i \rangle$ dal set E , il triggering event d viene unificato con i triggering event di tutti i piani contenuti nel set P .
- Il *most general unifier*² (mgu) che unifica i due eventi è detto relevant unifier.
- L'intention i può essere:
 - La true intention.
 - Un'intention esistente che ha generato l'evento.

Osservazioni 5.2.2

- Un piano rilevante è anche applicabile se esiste una sostituzione che composta con l'unificatore rilevante e applicata al contesto fa sì che quest'ultimo risulti una conseguenza logica dei base beliefs B .
- La condizione del contesto di un piano rilevante deve essere conseguenza logica di B affinché il piano sia applicabile.

5.2.4 JASON: Un'Implementazione di AgentSpeak(L)**Definizione 5.2.8: JASON**

Java-Based agentSpeak interpreter used with saci for multi-agent distribution Over the Net. Si tratta della prima implementazione significativa di AgentSpeak(L) che utilizza SACI: un'infrastruttura per la comunicazione tra agenti per creare un sistema distribuito sulla rete.

Aggiunge alcune cose ad AgentSpeak(L):

- Gestione del fallimento dei piani.
- Supporto per lo sviluppo di ambienti in java.
- Possibilità di eseguire un ambiente multi-agente.
- Possibilità di personalizzare funzioni di selezione, di belief-revision, di azione, di comunicazione tra agenti.

²Che schifo IALAB.

- Libreria di azioni interne di base.
- Possibilità di definire azioni interne.
- Aggiunta della negazione forte.

Note:-

Le versioni fino alla 2 utilizzavano un IDE che era una versione modificata di gedit, dalla 3 in poi hanno capito che la command line è oggettivamente superiore.

Possibile uso della negazione forte:

- La *negazione debole* è usata nel contesto dei piani come in AgentSpeak(L).
- La *negazione forte* è usata per negare un predicato/fatto.

Predicati atomici annotati:

- Si possono specificare annotazioni ai predicati nella base belief.
- Annotazione: lista di termini associati a un predicato.
- Annotazioni particolari:
 - [source(percept)]: l'informazione è stata percepita dall'ambiente.
 - [source(self)]: l'informazione è stata aggiunta dall'agente stesso durante l'esecuzione di un piano.

Piani con labels:

- È possibile associare una label a ciascun piano.
- La label può essere un qualsiasi predicato, anche un predicato con annotazione.

Gestione del fallimento dei piani:

- Esistono eventi in caso di fallimento.
- Tali eventi sono generati se un'azione fallisce o non vi sono piani applicabili per un evento con l'aggiunta di un goal +!g.
- Viene generato un evento interno per -!g associato alla stessa intention.
- Se il programmatore ha previsto un piano per -!g e questo è applicabile allora verrà eseguito, altrimenti verrà eliminata l'intera intention.

Azioni interne:

- Possono essere usate sia nel contesto che nel body.
- Sono azioni distinte dalle azioni che compie l'agente sull'ambiente mediante gli attuatori.
- Possono servire per fare introspezione.

Sistemi Multi-Agente con JASON

- Si possono definire sistemi con più agenti.
- Ogni sistema prevede un ambiente in cui gli agenti vengono collocati e un set di istanze di agenti.
- JASON offre una serie di script e un'interfaccia grafica.

