

Ciao a tutti. Vi lascio i miei appunti di Agenti Intelligenti - parte di Matteo Baldoni. Mancano appunti ben approfonditi su Jade e Jason.

Punti FONDAMENTALI:

1. potrebbero (e ci sono) esserci errori grammaticali, una *e* non accentata, una *p* di troppo e robe del genere... Ma ne rendo SEMPRE conto una volta che studio;
2. questi appunti non sostituiscono le lezioni e soprattutto sono personali.

Vi voglio bene. Un abbraccio dal vostro **Cirino Pomicino**.

Contents

1 Introduzione agli agenti	ii
1.1 Perchè agenti intelligenti?	ii
1.1.1 Interconnessione e distribuzione	iii
1.1.2 Un agente	iii
1.1.3 Sistema Multiagente	iii
1.1.4 Sistemi multiagenti: un argomento interdisciplinare	iv
1.2 Cosa si intende per sistemi intelligenti	v
1.2.1 Che cos'è un agente intelligente?	v
1.2.2 Agente intelligente o generico programma?	vii
1.3 Il Triangolo della Computazione	vii
1.3.1 Decomposizione funzionale	vii
1.3.2 Decomposizione Object-Oriented	viii
1.3.3 Attori e Oggetti attivi	ix
1.3.4 Business process	ix
1.3.5 Artifact-centric Process Management	x
1.3.6 Agent-Oriented Paradigm	x
1.3.7 Triangolo della computazione - Agent-Oriented Paradigm	xii
1.4 Architettura degli Agenti	xii
2 Agenti Razionali	xv
2.1 Il practical reasoning	xvi
2.1.1 Intenzioni nel practical reasoning	xvii
2.1.2 Proattività e intenzioni	xvii
2.1.3 Come sono caratterizzate le intenzioni?	xviii
2.1.4 Proprietà sulle intenzioni	xviii
2.2 Realizzazione di agenti che sfruttano il practical reasoning	xix
2.2.1 Agent control loop vers. 1	xix
2.2.2 Agent control loop vers. 2	xx
2.2.3 Agent control loop vers. 3 - Deliberazione	xi
2.2.4 Agent control loop vers. 4	xxiii
2.2.5 Agent control loop vers. 5	xxv
2.2.6 Agent control loop vers. 6 - Riconsiderazione delle intenzioni	xxvi
2.2.7 Agent control loop vers. 7 - Add funzione di riconsiderazione	xxvii
2.3 Procedural Reasoning system	xxviii
2.3.1 Procedural Reasoning System: stack delle intenzioni	xxix
2.3.2 AgentSpeak(L) e Jason	xxix
3 Sistemi Multiagente e Comunicazione tra Agenti	xxx
3.1 Perchè sistemi distribuiti d'agenti?	xxx
3.2 Distributed Artificial Intelligence (DAI)	xxxi
3.2.1 Sistemi autonomi di agenti	xxxi
3.2.2 La comunicazione	xxxi
3.2.3 Le infrastrutture della comunicazione	xxxi

3.3	Linguaggi per la comunicazione tra agenti	xxxiii
3.3.1	Perchè comunicare	xxxiii
3.3.2	Atti comunicativi	xxxiv
3.3.3	Struttura di un messaggio nella teoria degli atti comunicativi	xxxiv
3.3.4	Modellazione di atti comunicativi	xxxv
3.3.5	Agent Communication Languages (ACL)	xxxv
3.4	KQML e FIPA	xxxvi
3.4.1	Origini (di un ACL) e Knowledge Sharing Effort	xxxvi
3.4.2	Knowledge Interchange Format (KIF)	xxxvi
3.4.3	Knowledge Query and Manipulation Language (KQML)	xxxvi
3.4.4	KQML facilitators	xxxvii
3.4.5	Semantica di KQML	xxxvii
3.4.6	FIPA ACL	xxxviii
3.4.7	Semantica di FIPA ACL	xxxviii
3.4.8	Il test di conformità	xxxix
3.4.9	La semantica sociale per la comunicazione	xl
3.4.10	Semantica per ACL basata sugli stati mentali e verifica	xli
3.4.11	Semantica Sociale	xli
3.4.12	Commitments	xlii
3.4.13	Singh's Commitments	xlii
3.4.14	Il ciclo di vita dei commitment secondo Singh	xliii
3.4.15	Operazioni sui commitment (Singh, 1999)	xliv
3.4.16	Postulati sui commitment (Singh, 2008)	xliv
3.5	Protocolli di interazione	xlv
3.5.1	Specifica di protocolli	xlvii
3.5.2	Contract Net protocol (task sharing)	xlvii
3.6	Protocolli a commitment	xlviii
3.6.1	Protocolli di interazione con commitment	xlviii
3.6.2	Un esempio: The NetBill Protocol	xlix
3.6.3	The NetBill Protocol con i commitment	l
3.6.4	Un altro esempio: The Contract Net Protocol	li
3.6.5	Run del protocollo a commitment	li
3.7	Modelli di coordinazione	lii
3.7.1	Le norme	lii
4	Robustezza e ABS	liii
4.1	Agent Based Modelling (ABS)	liv
4.1.1	Complex Adaptive Systems	liv
4.1.2	Agent-based Modeling	lv
5	FIPA	lvii
5.1	The FIPA Agent Platform	lvii
5.2	The Core Specifications	lviii
5.3	Il Message Transport	lviii
5.4	FIPA Message Structure	lviii
6	JADE	lix

Chapter 1

Introduzione agli agenti

Gli agenti sono un tipico modo in cui si va a modularizzare e organizzare il software. Nel concetto di agente, rispetto a quello di componente c'è già qualcosa che lo distingue. Agente ha di per se una connotazione legata all'**autonomia**. L'agente persegue un certo **obiettivo**.

D. cosa intendiamo con la parola intelligente?

In queste prime righe andremo ad esplicitare cosa vuol dire per noi *intelligente*

1.1 Perchè agenti intelligenti?

Nella computazione, storicamente, sono cinque i **trends** che hanno caratterizzato la computazione:

- **ubiquità**, la continua riduzione dei costi di elaborazione ha permesso di introdurre capacità di elaborazione in luoghi e dispositivi che una volta sarebbero stati anti-economici. Quindi, ci siamo ritrovati sempre con più dispositivi dalle capacità di elaborazione distribuiti. Man mano che questi sistemi hanno iniziato a diffondersi, desideriamo che siano dotati di elementi intelligenti in modo da essere utilizzati più facilmente. Uno degli obiettivi di affrontare questa situazione è trarre vantaggio da questa **capacità di elaborazione distribuita**;
- **interconnessione**, i sistemi informatici non esistono più soli, ma sono collegati in rete (anche in rete ad hoc, la domotica che abbiamo in casa ne è un esempio). Dal momento che i sistemi distribuiti e concorrenti sono diventati la norma, alcuni ricercatori stanno proponendo modelli teorici che ritraggono l'informatica principalmente come un processo di interazione. Questa, non può essere un'interazione che modelliamo semplicemente come invocazione di un metodo remoto. Già quello risulta complesso, perché abbiamo uno stack di complessità notevole. Quello che vorremmo fare è elevare ulteriormente per avvicinare sempre più la computazione ad un uso intuitivo possibile.
- **intelligenza e delega**, la complessità dei compiti che siamo capaci di automatizzare e quindi delegare ai computer è cresciuto costantemente. I computer sono capaci di fare tanto senza il nostro intervento, quindi quello che ci aspettiamo è **intelligenza**. Un esempio lampante sono i stempi di guida automatica di aerei, macchine
- **human orientation**, spostarsi da una vista orientata alla macchina della programmazione, verso concetti e metafore che sono più vicine al modo con cui noi comprendiamo e vediamo il mondo, è sempre stato uno dei tende forte della programmazione. La modellazione object-oriented (OO) è un esempio di questo trends. In sviluppo e applicazione software si è visto che si parlava di attribuzione di responsabilità ad oggetti e concetti del modello che andavamo ad analizzare e progettare, dove la progettazione veniva fatta attraverso attribuzione di responsabilità. L'attribuzione di responsabilità

alle componenti è un modo per semplificare lo sviluppo e l'organizzazione del software. Anche da un punto di vista ingegneristico ci aspettiamo che gli agenti siano intelligenti, ovvero, avere una certa facilità nel programmarli (utilizzare strumenti più vicini a noi per facilitare la risoluzione di alcuni problemi). Il concetto di intelligenza quindi viene visto come un qualcosa che va oltre al comportamento di intelligente di un agente.

Tutto questo dove cazzo ci porta?

Delegazione e intelligenza implicano la necessità di costruire sistemi informatici in grado di agire efficacemente per nostro conto. Questo implica

- la capacità di agire dei sistemi informatici in modo indipendente
- 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.

1.1.1 Interconnessione e distribuzione

L'interconnessione e la distribuzione sono diventati fondamentali. Accoppiati con la necessità di sistemi che rappresentino nel migliore dei modi i nostri interessi, portano a sistemi che possono **cooperare** e **raggiungere accordi** o **competere** con altri sistemi che hanno interessi diversi. È difficile rappresentare sistemi che cooperano e competono andando a ragionare su istruzioni a basso livello. Strumenti di astrazione ad alto livello ci permettono di farlo in maniera più facile. Tutti questi problemi sono stati studiati in informatica e sono tendenze che sono recepite anche nei campi dei **sistemi multiagenti**.

1.1.2 Un agente

È un sistema di computazione, un sistema informatico, capace di agire (caratteristica fondamentale di un agente) 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).

1.1.3 Sistema Multiagente

Consiste in una serie di agenti che interagiscono l'uno con l'altro. In generale, gli agenti agiscono per conto di utenti con differenti obiettivi e motivazioni (agiscono per conto). Per interagire con successo richiedono la capacità di **cooperare**, **coordinarsi** e **negoziare** tra loro, in modo analogo a quanto fanno le persone. In merito, vedremo degli standardi di comunicazione, cooperazione, competizione e meccanismi d'asta formalizzati. Un sistema multiagente risponde a diverse domande:

1. come può emergere la cooperazione in società composte da agenti *self-interested*?
2. quali tipi di linguaggi possono essere utilizzati dagli agenti per comunicare?
3. come possono gli agenti *self-interested* riconoscere conflitti) e come possono (tuttavia) raggiungere accordi?
4. come possono gli agenti autonomi coordinare le proprie attività in modo da raggiungere cooperativamente gli obiettivi?

Vedremo strumenti e software che sono stati sviluppati per rispondere a queste cose.

Agent Design, Society Design

Un sistema multiagente risponde a queste domande:

1. **Come possiamo creare agenti capaci di agire in modo indipendente, autonomo, in modo che possano svolgere con successo i compiti che gli deleghiamo?**
2. **Come possiamo creare agenti capaci di interagire (cooperare, coordinare, negoziare) 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/obiettivi?**

Il primo problema è di progettazione/design di un agente, il secondo è di progettazione/design di società di agenti (micro/macro).

1.1.4 Sistemi multiagenti: un argomento interdisciplinare

Il campo dei sistemi multiagenti come un po tutta l'AI è quella di avere una forte componente interdisciplinare.

Tale campo è ispirato e influenzato da molti altri campi:

- Economia;
- Filosofia;
- Teoria dei giochi;
- Logica;
- Ecologia;
- Scienze Sociali;

Sono aree che influenzano molto. Potrebbe essere visto sia come un punto di forza, ma anche uno svantaggio in quanto ci sono più punti di vista che vanno coniugati.

D. Perchè dedicare attenzione ai sistemi multiagenti e ad agenti intelligenti?

R. L'obiettivo dei sistemi ad agenti e sistemi intelligenti è risolvere tutti i problemi dell'intelligenza artificiale. Il solo obiettivo è quello della costruzione di agenti utili, ovvero quelli che risolvono i problemi. L'AI all'inizio aveva ignorato gli aspetti sociali dell'*agire* (caratteristica forte dei sistemi multiagenti). L'AI si era concentrata su quello che era la costruzione di un componente intelligente. *L'algoritmo di Search, quello per calcolare il pane, si concentrava su un agente.* Sistemi intelligenti e multiagenti si concentrano su aspetti sociali. Il fatto che **agire** abbia un'implicazione non solo su me stesso, ma anche su altri e che questo influenzi l'agire degli altri. C'è una forte connessione con i sistemi multiagenti e l'economia, in particolare con la **teoria dei giochi** (Nash, Nobel per l'economia). Il problema che nell'economia, non vi è interesse verso l'aspetto computazionale, cosa che hanno i sistemi multiagenti e intelligenti. Questi sono astrazioni computazionali, dove l'enfasi verrà data su aspetti computazionali e non semplicemente su aspetti teorici e modellistic. Uno dei più noti contributori è Cristiano Casterfranchi (sociale al CNR di Roma). Lui, ha portato notevoli contributi, proprio partendo dalle teorie sociali. *Social Science*, è d'interesse per la modellazione di sistemi distribuiti. Anche al contrario, i sistemi computazionali offrono dei modelli e strumenti ai sociologi (o a chi studia sistemi complessi distribuiti). I sistemi di simulazione basati su agenti, sono uno degli strumenti più diffusi per fare simulazioni. Queste partono da ipotesi su azioni semplici che possono evolversi in sistemi complessi quando vengono fatte interagire.



Figure 1.1: Che cos'è un agente?

1.2 Cosa si intende per sistemi intelligenti

- la principale caratteristica degli agenti è la loro **autonomia**. Ovvero, la capacità di agire indipendentemente ed eseguire controllo sul proprio stato interno. Quindi, 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. 1.1¹

Esempio

Prendiamo un algoritmo. Parte di queste caratteristiche le ha già. È capace di agire *autonomamente*. Raggiungere un *obiettivo*. Un programma qualsiasi farà sempre quello che è stato scritto come programma, dove questo può essere scritto in maniera più o meno flessibile e soddisfare più o meno i nostri obiettivi, ed è proprio qui che sta l'**AI**.

Nel paradigma ad agenti abbiamo:

- agente e ambiente come due elementi fondamentali di pari importanza.

Un esempio di agente autonomo è il **termostato**. Percepisce una temperatura, decide cosa fare, accende/spegne il riscaldamento, oppure non fa nulla. Lo stesso si può dire di un **aspirapolvere**. Il primo spesso viene inteso in maniera un po' più stupido anche se non è vero, alcuni di essi decidono cosa fare in maniera più sofisticata, questo perché considerano una serie di parametri. Il concetto di agenti è comunque condiviso da tutti

- percepire l'**ambiente** e **agire** sull'ambiente.

1.2.1 Che cos'è un agente intelligente?

È un sistema di computazione capace di esibire azioni autonome in modo glessibile (caratteristica fondamentale che permette ad un agente di essere intelligente) in un ambiente. In modo **flessibile** intendiamo:

- **reattivo**, se l'ambiente di un software è statico/fisso non è necessario preoccuparsi di esso. *Compilatore prende un programma e lo compila*. Per lui l'ambiente è statico. Il programma in input è quello che è. Se volessimo invece modellare un agente che agisce nel mondo reale, la cosa cambia. Il mondo è **dinamico**. Proprio per cui è necessario considerare che l'esecuzione delle nostre istruzioni non abbiano successo e considerare se vale la pena eseguire una certa azione. Un agente **reattivo** 1.2 viene visto in questo modo. Si nota in figura 1.2, abbiamo l'ambiente e agente. L'agente agisce sull'ambiente e quest'ultimo viene visto come input per l'agente stesso. Un agente **reattivo** è un sistema che mantiene una costante interazione con l'ambiente e risponde ai cambiamenti che occorrono su di esso (in tempo perché la risposta sia utile). *Se il robot vede l'ostacolo e ci si schianta sopra non è tanto intelligente, se lo evita sì. La reattività è associata al concetto di intelligenza.*

¹Notare come l'output è sull'ambiente e l'input è l'ambiente

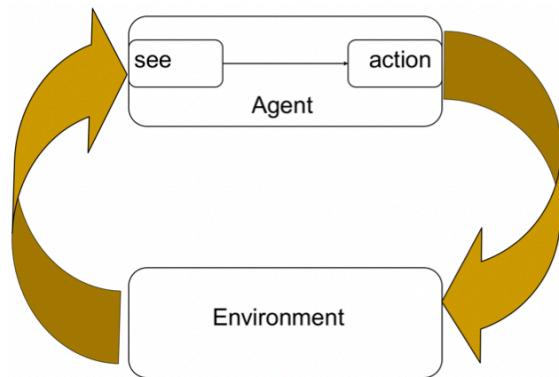


Figure 1.2: Agente reattivo

- **proattivo.** Reagire ad un ambiente è facile (es. evento/frena → freno anch'io). In generale, ad intelligenza viene attribuita la capacità di fare cose per noi. In che senso? Il fatto che noi attribuiamo degli obiettivi (a questa componente) che persegue. Cioè dei comportamenti guidati da **goal** (goal directed behaviour). *Un pianificatore ha un goal e poi in base al modello e ai vincolo ha un piano per raggiungerlo.* Proattività → generare e tentare di raggiungere obiettivi non solo guidata da eventi, **prendere l'iniziativa.** **Gli agenti proattivi mantengono uno stato.**

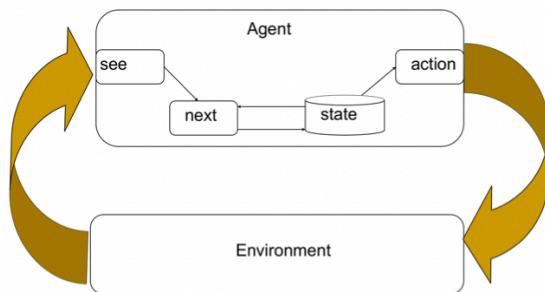


Figure 1.3: Agente Proattivo

Lo stato dell'agente contiene due tipi di conoscenza:

- i. l'agente necessita di informazioni su come l'ambiente evolve;
- ii. l'agente necessita di informazioni su come le proprie azioni impattano sull'ambiente. Questo in modo da prevederne gli effetti e se prevedo degli effetti l'agente deve essere in grado di **ragionare**.

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 ragionare su piani. *L'aspirapolvere viene visto anche come un agente proattivo nel momento in cui decide se continuare ad agire e aspirare un poco oppure andare a ricaricarsi.* Deve decidere se continuare ad aspirare e quindi l'effetto dell'azione sarà quella di non andare più a ricaricarsi. L'obiettivo è quello di aspirare e continuare ad aspirare e se si ferma (batteria scarica) non continua più non raggiungendo più il suo obiettivo.

- **sociale.** Il mondo reale è spesso un **ambiente multi-agente**: non possiamo raggiungere i propri obiettivi senza l'aiuto di altri. *Noi vorremmo un oggetto, se non c'è qualcuno che lo vende, noi non riusciamo a raggiungere l'obiettivo, non siamo in grado di produrre da soli.* Alcuni obiettivi possono essere raggiunti solo con la collaborazione di altri. In modo analogo i sistemi sono immersi in una rete di computer. 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. (KQML e FIPA).

Bilanciare reattività e comportamento guidato dagli obiettivi

Bilanciare reattività e comportamento guidato dagli obiettivi è una delle cose che si fa. Si desidera che gli agenti siano reattivi e che rispondano ai cambiamenti per tempo, ma altresì, si desidera che gli agenti lavorino in modo sistematico verso gli obiettivi di lungo termine. In genere, sono in conflitto. Progettare agenti che hanno un buon bilanciamento viene visto sempre, a causa del dominio e dal problema, un problema aperto. Non c'è una soluzione generale.

1.2.2 Agente intelligente o generico programma?

Un agente autonomo è un sistema situato in un ambiente che può percepire ed agire su di esso nel tempo con il fine di perseguire una propria agenda, così da **influire** nelle successive percezioni. Quello che faremo avrà un effetto che riscontreremo in modo successivo. Un programma per le paghe fa sensing sul mondo attraverso input e output. Non è un agente perché il suo output non ha effetto su quello che lui percepisce in seguito. Inoltre non c'è continuità temporale over time.

1.3 Il Triangolo della Computazione

La modularizzazione del software è sempre stato uno degli obiettivi principali dell'ingegnerizzazione del software. Se vogliamo software di qualità, occorre garantire:

- correttezza;
- robustezza;
- estensibile;
- riusabilità.

Uno dei principali strumenti per raggiungere queste proprietà è quella della **modularizzazione**. Meyer introduce il concetto del **triangolo della computazione** 1.4. Meyer, dice che tre sono gli assi principali, ovvero le forze che la tirano, che influenzano/modellano la computazione. L'esecuzione di un sistema software usa certi **processori**² che applicano certe **azioni**³ su certi **oggetti**⁴. Ogni forza, per conto proprio, agisce sulla computazione. Un buon linguaggio di programmazione dovrebbe avere un buon **equilibrio** di queste componenti, in modo da descrivere computazioni efficaci e raggiungere una buone ingegnerizzazione del software. Meyer, fornisce questo come elementi su cui confrontare diversi approcci alla modularizzazione. Ne è un esempio la **decomposizione funzionale**

1.3.1 Decomposizione funzionale

Ottenuta tramite i paradigmi imperativi. Questi paradigmi operano, da un punto di vista dello strumento di modularizzazione, sulla **decomposizione funzionale** 1.5.

D. che cosa vuol dire?

²una CPU fisica, un processo o thread

³operazioni che compongono il calcolo (operazioni in linguaggio macchina fino a subroutine)

⁴strutture dati a cui si applicano le azioni (dipendenti dal calcolo o file, database, ecc.)

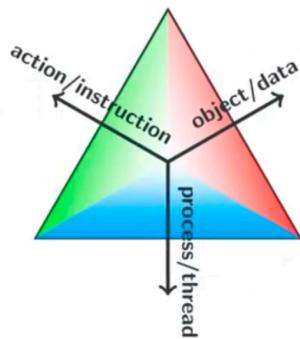
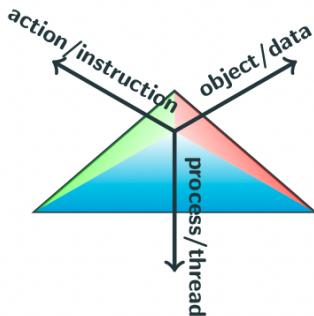


Figure 1.4: Triangolo della computazione di Meyer

Prendo il programma (che è una funzione) e lo descrivo tramite una decomposizione di sottofunzioni (le procedure). Questo è lo strumento di modularizzazione che abbiamo nei paradigmi imperativi. Tutta la modularizzazione è focalizzata sulla decomposizione funzionale. Se volete, le istruzioni e i dati passano in secondo ordine.

- **vantaggi:** è intuitivo. Spesso ragioniamo in questi termini. Decomporre un qualcosa che dobbiamo fare in passi più semplici;
- **svantaggi:** difficile manutenzione. È difficile introdurre al suo interno ulteriori nuovi goal. Il programma scala in modo difficile ad altre funzionalità. Cosa che l'OO ha permesso di fare molto più facilmente

Figure 1.5: Decomposizione funzionale - *Puts at the center the process force*

1.3.2 Decomposizione Object-Oriented

Applica una decomposizione sui dati. Quindi, anzichè concentrarsi sulla forza algoritmo, ci concentriamo sulla forza dato. *Trova le responsabilità di vari dati*. Così facendo passa in secondo ordine l'**obiettivo**, ovvero il concetto algoritmo. È difficile definire cosa faccia un programma OO. Spesso il main è un metodo, seguito da un intreccata sequenza di chiamate distribuite. Se dovessimo distinguere il programma processo da quello dati, il tutto risulta veramente difficile.

ES. Pensiamo a quando scriviamo i processi thread e consideriamo Java. I thread Java non c'entrano nulla con il linguaggio stesso. Per fare un thread in Java, prendiamo una classe, seguita da *extends Runnable* o *Thread* e al suo interno inseriamo il metodo *run*. Il metodo, *run* non ha niente a che fare con la classe che la ospita. Potenzialmente potrebbe fare tutt'altro. È solo un posto dove andiamo a mettere *run* che è eseguito in un thread. Potrebbe anche usare oggetti diversi rispetto a dove è la posizione. Tutto questo fa capire come il thread non c'entra nulla con l'oggetto.

- **vantaggi:** i processi hanno vita propria indipendente dai processi che li utilizzano. Fornire le azioni che permettono di operare sui dati. L'oggetto è la nozione fondamentale del modello
- **svantaggi:** gli oggetti sono passivi: i processi esterni prendono le decisioni su quali azioni invocare. Nessun disaccoppiamento tra l'utilizzo di un oggetto e la gestione di tale oggetto. Nessun supporto concettuale alla specificazione di compiti (processi).

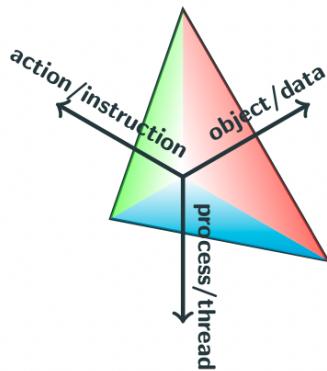


Figure 1.6: Decomposizione funzionale - *Puts at the center the object force*

1.3.3 Attori e Oggetti attivi

Ci sono altre soluzioni. Un esempio è il **modello attori** o active objects. Un thread può essere messo su un oggetto e questo non vuol dire che quell'oggetto ha un processo thread separato, ma vuol dire che quando lui risponde a dei metodi, c'è un processo separato che risponde ai metodi. Molto utile quando facciamo programmazione distribuita su GPU. Un oggetto viene messo su ogni core e questi forniscono la comunicazione e ognuno ha il suo thread separato che esegue e collaborando, raggiungono un certo obiettivo. Quindi, un oggetto è un thread e l'eccezioni fanno riferimento all'**attore** che ha eseguita quella computazione. Ogni insieme di record di attivazione è un oggetto, solo che interagisce con altri. Si, il costo è *superiore*, ma si ammortizza con strumenti multi-core. Comunque, questo il focus sui dati, perché un oggetto risponde solo a quella che è la richiesta specifica, ma non ha **reattività** e **proattività**, ha solo **socialità** per quanto riguarda l'interazione con altri.

- gli attori hanno il loro thread di computazione;
- il modello degli attori non è adeguato per risolvere problemi di **coordinamento** (altre estensioni comunque sono state proposte)
- in termini di forze di Meyer, il modello attore:
 - supporta processi di gestione oggetti/dati (ad es. internal behaviors of actor)
 - non supporta la progettazione dei processi che utilizzano questi oggetti (es. processes external to actors)

1.3.4 Business process

I **business process** (BP) rappresentano attività di un'organizzazione e il processo rappresenta degli obiettivi che l'organizzazione vuole raggiungere. La cosa interessante è che il processo per intero viene visto come l'espressione di un **goal** che include anche il come raggiungere quel particolare goal. Se in un processo esprimi che una particolare attività deve esser svolta prima di un'altra, stiamo esprimendo un **vincolo**. Quindi, si esprimono degli obiettivi e le

modalità con cui vengono raggiunti quei particolari obiettivi (es. constraint language). Il BP è una descrizioni di vincoli **dinamici**, dove il goal è raggiungere quel particolare obiettivo, rispettando questi vincoli. Se non sono vincoli, allora, dobbiamo esprimere de modi alternativi. Il più delle volte nel organizzare le arrivit si vuole imporre una rappresentazione molto fine dei vincoli. Tornando a noi, i BP, sono una sorta di decomposizione funzionale, perchè sono concentrati sul processo, dove la decomposizione viene fatta con dei sotto processi. *Dati e oggetti sono rappresentati tramite uno store*

- STESSI LIMITI DELLA DECOMPOSIZIONE FUNZIONALE!

1.3.5 Artifact-centric Process Management

I **business artifact** (BA), non è altro che un'entità aziendale concettuale utilizzata per guidare le operazioni di un'azienda (ad es. la consegna di pacchi). Include

- **information model**: conserva i dati rilevanti sull'artefatto mentre si muove attraverso il flusso di lavoro;
- **lifecycle model**: stati attraverso i quali i dati possono evolversi + transizioni tra stati
- enfasi sull'**object-data** force:
 - a. progettazione e modularizzazione dei processi operanti sui BA non viene esplicitamente considerata

Si è fatto un spostamento dalla modellazione dei processi focalizzati su attività *activity centric*, a processi espressi in maniera *data centric*, cioè sul dato. *Prendiamo un ordine Amazon*. Se lo guardiamo nell'interfaccia web, ci possiamo accorgere che in realtà il processo di compravendita che stiamo conducendo, non è guidato da un obiettivo finale, ma da uno **stato dell'ordine**. Sono gli stati che assume un ordine, che permettono agli attori coinvolti nel processo, di eseguire alcune azioni. La spedizione viene fatta solo dopo il pagamento, dopo che l'oggetto è stato preso e imbustato. Solo in quel momento lo stato dell'ordine cambia in *spedito*. La mansione del lavoratore di imbustare l'oggetto non viene fatta se non dopo il pagamento. Lo stato dell'ordine coordina. Il tutto si può pensare come se fosse una pila (stack). Le azioni dipendono dallo stato e la *pop* è possibile farla solo se lo stato non è vuoto, proprio per cui metteremo una *throws exception*. Esistono proposte di linguaggi *state oriented* che permettono di definire classi i cui metodi, quindi le proprietà, variano in base allo stato dell'oggetto stesso. Hanno dei tipi complessi e comportamentali. Il **data-centric** introdotto da IBM permette di esprimere processi che dipendono dallo stato dei dati. Prenderà i soli dati rilevanti (BA). Questi non sono un qualsiasi dato, ma informazioni, artefatto - obiettivo. Andremo a definire quale sarà:

- l'oggetto del business e in seguito andremo a definire il processo basandolo sullo stato;
- le operazioni verranno definite sul ciclo di vita del dato e quindi in qualche modo le attività che possono compiere gli attori dipenderanno da questo stato

1.3.6 Agent-Oriented Paradigm

Tutto questo per capire come è collocato l'**Agent-Oriented Paradigm**. Punto fondamentale. NON SI PUÒ PARLARE DI AGENTI SENZA PARLARE DI AMBIENTE.

- **agenti**, visti come forza processo, quindi hanno un obiettivo da raggiungere;

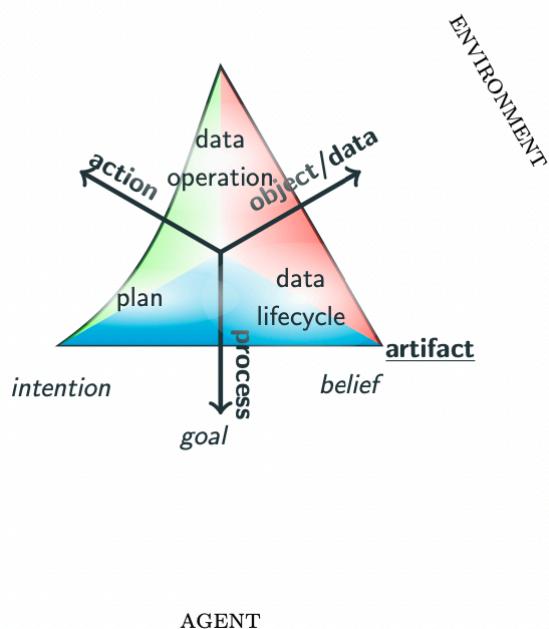


Figure 1.7: Agent-Oriented Paradigm

- **ambiente**, è la forza dato oggetto. L'ambiente non è solo quello realtà, ma anche quello che elabora i dati. L'ambiente, dice Weyns, è un'astrazione di prima classe che fornisce le condizioni (di esistenza) intorno all'agente e che fa da mediatore con l'interazione con altri agenti. Dice che l'interazione tra gli agenti è **mediata** dall'ambiente, quindi anche l'interazione viene vista come *ambiente*. *Il professore comunica con gli studenti grazie all'aria.*

Il concetto di interazione senza quello di ambiente **non ha alcun senso**.

Questo paradigma deriva dalla visione di sistemi di coordinazione a black-board. Negli anni '90, si è studiato tantissimo il problema della coordinazione. Sia mediante protocolli che mediante spazi condivisi (di coordinazione). Uno dei metodi di coordinazione di processi era così fatto. C'era una *shared memory* con dati dentro, dove i dati potevano essere *read* o *write*. Un processo era in attesa di leggere o scrivere. Questi sistemi sono stati ampiamente utilizzati con linguaggi tipo Linda.

Gli agenti

- **autonomi**, hanno un ciclo deliberativo e guidati da un goal (hanno un obiettivo da raggiungere);
- **situati**, nel senso che sono posizionati in un ambiente, dove quest'ultimo deve essere programmabile allo stesso modo di quanto si fa con gli agenti.

Agenti vs Oggetti

- gli oggetti non hanno il controllo sul proprio comportamento (passivo/reactivo);
- gli oggetti non mostrano un comportamento flessibile (nessun ciclo deliberativo);
- gli oggetti sono a thread singolo.

Spesso si dice che gli **agenti** sono un'evoluzione degli **oggetti**, perché sono anche attivi. È sbagliato. Gli agenti sono un'evoluzione del processo, di quello che in una visione dell'ambiente non c'è. Gli oggetti in un OO ci sono ed è l'ambiente non l'agente che ha acquisito capacità in più.

Agenti vs Attori

Gli **attori** non sono deliberativi/proattivi. Non hanno un obiettivo.

1.3.7 Triangolo della computazione - Agent-Oriented Paradigm

- **processi**, permettono un'astrazione verso oggetti
- **oggetti/dati** verso l'ambiente
- i *process* rappresentano il goal
- i *belief* rappresentano l'intersezione tra l'ambiente e il processo (il ponte che c'è tra i due)
- le *intenzioni* sono un passaggio da quello che posso fare a quello che vorrei (agente BDI → le intenzioni vengono usate per determinare le azioni che dobbiamo eseguire per raggiungere l'obiettivo). Usate per determinare i *piani*
- gli *artefatti* non sono soltanto dati, ma dati con un ciclo di vita
- i *piani* rappresentano la decomposizione funzionale del goal. Vedremo come il goal negli agenti viene determinato da un piano. Piani che dipendono dalle azioni che dispongono. A seconda delle azioni avremo una particolare possibilità;
- quando definiamo delle operazioni sui dati (quindi metodi)- *data operation*- lavoriamo sull'angolo tra quando definiamo le azioni e i dati (vertice alto del triangolo). Definiamo cosa fanno gli oggetti e quindi come i processi potranno operare con le azioni sugli oggetti

Operazioni vs Eventi

Gli **eventi** sono quelli che l'agente fa, le **operazioni** è un qualcosa che viene offerto. Se un sistema ha un pulsante, l'agente deve avere la capacità di agire, quindi schiacciare il pulsante, ma l'altro deve avere la capacità di eseguire un qualcosa. Quando abbiamo agenti, dati e ambiente, da una parte l'ambiente metterà a servizio delle operazioni, ma l'agente deve avere la capacità (piano) di premere quell'azione e quindi scatenare l'evento

1.4 Architettura degli Agenti

Abbiamo chiarito cosa si intende con agente, agente intelligente e di come il paradigma si rapporta con tutto il resto. Adesso ci focalizziamo sulla costruzione di un agente. Quali saranno le architetture che ci serviranno? L'idra è quella di costituire agenti che abbiano le caratteristiche di **autonomia, reattività, proattività e abilità sociali**.

- **1956-1985**: la maggior parte dei progetti di architetture per agenti si basano sul **ragionamento simbolico**. Quello che viene proposto è un encapsulamento di algoritmi di risoluzione, deduzione dentro agenti. Questo era il modo comune di vedere le architetture ad agenti;

- **1985-presente** (reactive agents movements): viene fuori tutto un filone di ricerca e offerta di linguaggi basati sulla **reattività** (parte del professore Martelli). Il problema principale è il **tempo**. In molte applicazioni il tempo di fare *reasoning* non è sufficiente, proprio per cui si sono proposte soluzioni basate sulla reattività. Soluzioni utilizzate anche adesso. In IALAB viene insegnato il linguaggio a regola CLIPS (IN BOCCA AL LUPO!!!)
- **1990-presente: architetture ibride.** cercano di coniugare un po' di ragionamento simbolico con la *reattività*. Questo ragionamento non è quello del solo ragionamento logico. (Il professor Martelli ci farà vedere come il ragionamento sulla logica si è evoluto rapidamente)

Il classico approccio per costruire agenti intelligenti nel *symbolic reasoning* è quello di vederli come casi particolari di sistemi basati sulla conoscenza. L'architettura di un agente deliberativo:

- contiene un'esplicita rappresentazione (modello simbolico) dell'ambiente;
- prendere decisioni (es. quale azioni eseguire) e questo lo si fa attraverso un ragionamento simbolico;

In questo approccio si presentano due problematiche.

1. **problema della trasduzione**, tradurre l'ambiente in una rappresentazione che possiamo utilizzare e questa traduzione deve essere fatta molto veloce. In **tempo** affinché possa essere utile;
2. **problema della rappresentazione/ragionamento**. *Abbiamo il modello e dobbiamo decidere che azioni fare.* Se dobbiamo calcolare l'azione possiamo usare ad esempio il **meccanismo della logica classica**. Questo meccanismo di risoluzione costa (proprio per cui faremo delle approssimazioni). Dobbiamo dedurre cosa fare in tempo, perché se l'ambiente è dinamico e non riesco ad agire in tempo in modo che le azioni abbiano l'effetto desiderato ... **me cojoni**

Questi problemi non sembrano facilmente risolvibili, servono per cui delle alternative.

Esempio di agente deduttivo

L'aspirapolvere è un esempio di agente deduttivo.

- l'obiettivo è quello di ripulire tutto lo sporco
- D. come un agente può decidere cosa fare utilizzando tecniche di ragionamento?
- R. L'idea di base è usare la logica per codificare una teoria permettendo di ottenere la migliore azione da eseguire in qualsiasi situazione. Quindi per ragionare su dove deve andare a pulire, occorre rappresentare l'ambiente su dove deve agire e quindi rendere l'ambiente discreto.

Per fare ciò, possiamo pensare di usare tre predicati:

1. $In(x, y)$, l'agente è nella posizione (x, y)
2. $Dirt(x, y)$, c'è dello sporco nella posizione (x, y)
3. $Facing(d)$, l'agente è rivolto in direzione d

Le possibili azioni da applicare sono le seguenti:

- $Ac = \{turn, forward, suck\}$

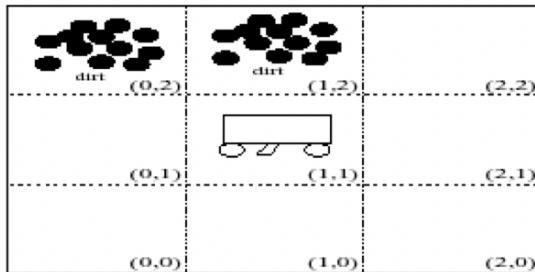


Figure 1.8: Agente deduttivo - L'aspirapolvere

Regole per determinare cosa fare:

- $In(0, 0) \wedge Facing(north) \wedge \neg Dirt(0, 0) \rightarrow Do(forward)$
- $In(0, 1) \wedge Facing(north) \wedge \neg Dirt(0, 1) \rightarrow Do(forward)$
- $In(0, 2) \wedge Facing(north) \wedge \neg Dirt(0, 2) \rightarrow Do(turn)$
- $In(0, 2) \wedge Facing(east) \rightarrow Do(forward)$

Utilizzando queste regole (più ovvie altre), a partire dalla posizione (0, 0) il robot rimuoverà lo sporco.

- Problemi:
 - a. come convertire l'ingresso della videocamera in $Dirt(0, 1)$?
 - b. il *decision making* basato sulla logica del primo ordine **indecidibile**
- Anche dove si usi logica proposizionale, il *decision making* nel peggior dei casi significa risolvere problemi di complessità **co-NP-complete**⁵
- tipiche soluzioni:
 - a. indebolire la logica
 - b. utilizzare una rappresentazione simbolica non logica, non derivata dalla logica
 - c. spostare l'enfasi dal ragionamento al tempo di esecuzione (*run time*) al tempo di progettazione (*design time*)

⁵co-NP-complete = bad news.

Chapter 2

Agenti Razionali

Entriamo nel dettaglio di come gli agenti possono essere costruiti e in particolare vediamo il concetto di **agente razionale**. Incominciamo nel fare qualche valutazione del tipo intuitiva. Spesso quando facciamo riferimento all'attività umana lo facciamo con frasi di questo genere:

- Cirino ha lavorato duramente perchè desidera un titolo PhD;
- Pomicino ha preso l'ombrelllo perchè credeva che stesse per piovere.

Si nota come c'è un *azione* (prendere l'ombrelllo) e una *spiegazione* (credeva che si sarebbe messo a piovere). Ogni azione è seguita da una spiegazione. In questo si nota come delle azioni, effettuate da agenti, sono viste come conseguenza di qualcosa che è creduto (*believed*) e desiderato (*desired*). Questo fa riferimento a quello che si chiama **psicologia popolare**, ovvero spiegare il comportamento umano in termini di attitudini come credere, volere, sperare e così via. Quello che a noi interessa è prevedere un'esecuzione in base a certe attitudini. Queste attitudini spesso sono note come **nozioni intenzionali**. Abbiamo l'intenzione di fare qualcosa, quando lo diciamo stiamo comunicando che stiamo lavorando per organizzandi in modo da conseguire/ottenre quella particolare cosa. Se dicesse ho intenzione di andare in piscina noi ci aspettiamo, osservando il comportamento, tutto il ciclo che porta la persona a conseguire quel particolare obiettivo, quindi vestirsi, uscire da casa, parcheggiare la macchina e infine nuotare. Aver comunicato quell'intenzione, autorizza (per chi ascolta) prevedere dei comportamenti. Se abbiamo una certa intenzione, il comportamento possiamo programmarlo/organizzarlo in modo da ottenere una determinata cosa. Queste nozioni intenzionali vanno in qualche modo a spiegare quello che sarà il comportamento. **Dennet** ha coniato il termine **sistema intenzionale** per descrivere entità "il cui comportamento può essere previsto dal metodo di attribuzione di credenze, desideri e acume razionale". Quindi, un sistema intenzionale è un sistema (non facciamo più riferimento alle persone) il cui comportamento può essere previsto.

Proviamo a ragionare al contrario. Se noi in qualche modo possiamo inserire o rappresentare delle credenze, dei desideri, delle acume razionali, ci si può aspettare che questo (il sistema) evolva le sua azioni per poterle raggiungere. Ci si aspetta quindi che un sistema intenzionale faccia qualcosa per raggiungerle. Adesso ci si chiede:

- è legittimo o utile attribuire credenze, desideri e così via ai sistemi informatici? È un po difficile pensarlo. Comunque se si potesse farlo, la risposta sarebbe dipende. L'utilità comunque dipende se questi elementi in qualche modo permettono di codificare quello che potrebbe essere il corso delle sue azioni. Se l'aspirapolvere crede che ci sia sporco in un certo punto, come tradurre questa credenza in azioni in modo che lo sporco venga aspirato? Il fatto che lo creda o no potrebbe essere possibile. Agenti basati su nozioni intenzionali verranno visti più in avanti quando parliremo dell'Agent Control Loop.
- McCarthy dice che può essere interessante e appropriato utilizzare nozioni intenzionali in un sistema informatico.

Agenti come sistemi intenzionali

- quali oggetti possono essere descritti da una posizione intenzionale? In realtà un po tutto.
- Shoam dice che potrebbe essere perfettamente coerente trattare un interruttore come sistema intenzionale che in qualche maniera adotta le intenzioni di chi lo usa. Se vogliamo accendere la luce, lui (l'interruttore) in qualche modo mi metterà a disposizione un qualcosa che mi permette di accedere la luce e questo qualcosa avrà appunto l'intenzione di accendere la luce.

Shoam dice che non ha tanto senso che l'interruttore ha adottato la mia intenzione. Il sistema è banale, sappiamo benissimo spiegarlo in termini di sistema stesso. L'aspirapolvere invece è difficile da spiegare. In altre parole, più sappiamo di un sistema, meno dobbiamo fare affidamento su spiegazioni animistiche, intenzionale dei suoi comportamenti. D'altro canto, con sistemi più complessi, una spiegazione meccanicistica del suo comportamento potrebbe non essere praticabile. Le **nozioni intenzionali** sono astrazioni che forniscono un modo semplice e familiare per descrivere, spiegare e prevedere il comportamento di sistemi complessi. Tutta l'informatica è basata su **astrazioni**

- procedure/funzioni
- tipi di dati astratti
- oggetti

Gli oggetti intelligenti e in particolare i sistemi intenzionali rappresentano un ulteriore astrazioni. L'importante è ricondurre l'astrazione a livello più basso in modo da riuscire a programmare il comportamento di un agente tramite l'intenzioni. I ricercatori partono dalla visione (forte) degli agenti come sistemi intenzionali: la cui versione più semplice consiste nella descrizione di una posizione intenzionale e questa posizione codifica in qualche modo in **azione**. Vedremo in seguito come arrivarci

- la posizione intenzionale verrà vista come **strumento di astrazione**: modo conveniente di parlare di sistemi complessi che permettono di prevedere e spiegare il loro comportamento senza capire come funziona

L'idea sviluppata a partire da questa osservazione è quella di usare la posizione intenzionale come strumento di astrazione nell'informatica, per spiegare, capire e soprattutto programmare sistemi informatici.

2.1 Il practical reasoning

Il **practical reasoning** (PR) è il ragionamento sulle azioni, sul processo di capire cosa fare e si distingue dal *theoretical reasoning* in quanto quest'ultimo riguarda le sole credenze. Il PR dovrebbe essere il meccanismo che permette di passare da un'intenzione ai fatti, quindi ragionare sulle sole azioni necessarie per soddisfare le intenzioni, basandomi su quelle che sono le informazioni di credenze e desideri. Abbiamo dei desideri, dei goal, delle credenze e da queste determiniamo delle **intenzioni**. Quando abbiamo un'intenzione adottiamo un PR, ovvero un processo che permetta di capire cosa fare per ottenere quello desiderato. Nelle attività umane il PR è caratterizzato da due fasi/attività principali:

- **deliberazione**, serve per decidere quali stato di cose *state of affairs* vogliamo raggiungere;
- **pianificazione** *means-ends reasoning-planning*, serve a decidere come raggiungere questi stati di cose

L'output della deliberazione è proprio l'intenzione.

Credo di avere del tempo questa sera, perchè credo che la spesa fatta sia sufficiente per mangiare. Desidero andare a fare una corsa, desidero andare a fare una corsa, desidero andare in piscina, desidero, desidero, desidero, alla fine decido di andare in piscina.

Questa è un'intenzione e il processo appena descritto è il **processo di deliberazione**, ovvero delibero quella che è l'intenzione. La pianificazione invece è il susseguirsi di step, ovvero quelle che sono le azioni che mi permettono in questo caso di andare in piscina.

Quando una persona si laurea all'università, si trova di fronte con la scelta di decidere che tipo di carriera seguire. Ad esempio, potrebbe considerare una carriera come accademico o una carriera nell'industria.

- il processo per decidere a quale carriera puntate è **deliberazione** (deliberation);
- il passo successivo è decidere come raggiungere questo stato di cose (*state of affair*). Questo processo è chiamato **means-ends reasoning**. Il fine del means-ends reasoning è un piano per raggiungere il prescelto stato di cose
- dopo aver ottenuto un piano, in genere un agente tenterà di eseguirlo

2.1.1 Intenzioni nel practical reasoning

Deliberation e means-ends reasoning sono processi **computazionali**. Come tali, in tutti gli agenti reali questi processi avranno luogo in presenza di risorse limitate (problema quando si vuole trasferire), questo vuol dire che prima o poi vorremo una risposta. Il tempo è una risorsa e quindi la risposta la dobbiamo ottenere entro un certo momento

- 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**

Con **intenzioni** ci riferiamo allo stato di cose che un agente ha scelto e al quale si impegna come sue intenzioni.

2.1.2 Proattività e intenzioni

Le scorse volte abbiamo attribuito intelligenza ad un agente quando tra le varie cose aveva una capacità *proattiva*, cioè si impegna a raggiungere degli obiettivi che noi li comuniciamo. È un modo/qualcosa che corrisponde esattamente a quello descritto fino ad adesso, ovvero deliberare delle intenzioni e impegnarsi a raggiungerle. Il ruolo delle intenzioni è quello di favore la proattività, cioè tendono a portare ad agire all'azione. Quindi la proattività possiamo introdurla introducendo le intenzioni, quindi non è altro che un meccanismo computazionale per deliberare e pianificare. Attribuiamo la parola intelligente ad un agente no perchè l'agente abbia consapevolezza o capacità extra programma, ma perchè riconosciamo in lui alcune caratteristiche tra cui la **proattività**.

Se abbiamo intenzione di scrivere un libro, allora ci si aspetta che faccia un ragionevole tentativo di raggiungere tale obiettivo

Bratman osserva che le intenzioni giocano un ruolo forte nell'influenzare l'azione rispetto ad altri atteggiamenti proattivi come il *desiderio*. Bratman inizia a mettere a confronto intenzioni con desideri e individua nelle intenzioni e no nei desideri, l'elemento che scatena il PR. Questi

desideri potrebbero essere contrastanti. *Vado a fare una corsa oppure vado in piscina?* Vorremmo farli tutti e due ma ne esaudisco solo uno. I desideri sono in contrasto ma no le intenzioni, non posso dire ho intenzione di fare X e intenzione di fare Y. uè cumbà na cosa a fa.

2.1.3 Come sono caratterizzate le intenzioni?

Prima di tutto sono **persistenti** in modo ragionevole. Se appunto ci viene detto *ho intenzione di andare in piscina* non è che poi andando su in ufficio non faccio niente per farlo. Alle intenzioni quindi viene attribuito della persistenza, ovvero persistere all'esecuzione di altri azioni. Se facciamo delle cose, l'intenzione non può decadere immediatamente, è ragionevole che duri e persista attraverso le nostre azioni. Prima di andare in piscina noi compiamo delle azioni (entrare in macchina, parcheggiare ...), l'intenzione finale non viene intaccata dalle azioni, ma le azioni persistono. Solo se la motivazione dell'intenzione non persiste più è razionale abbandonare tale intenzione e quindi viene ragionevole che smetta di fare l'azione.

Le intenzioni vincolano il futuro del PR, ad esempio non sceglierà le opzioni che non sono coerenti con le proprie intenzioni. Abbiamo determinato un'intenzione e continuiamo a lavorare. Nel mentre possono essere deliberate altre intenzioni e queste qua non possono andare in contrasto, devono poter essere coerenti. Non possiamo deliberare l'intenzione di andare in piscina e subito dopo deliberare l'intenzione di andare a giocare a calcetto (che non sarebbe male). Quindi, una volta che è stata deliberata la fese di PR, tutte le intenzioni devono essere coerenti.¹.

2.1.4 Proprietà sulle intenzioni

1. Le intenzioni pongono problemi agli agenti, che devono determinare modi per realizzarli.
 - a. se ho intenzione di fare compiere un'intenzione φ , ci si aspetta che impieghi delle risorse per decidere come rendere vero ϕ . Dobbiamo impiegare risorse per decidere come rendere vero φ
2. Le intenzioni forniscono un "filtro" per l'elaborazione di altre intenzioni, che non devono entrare in conflitto.
 - b. se ho intenzione φ allora non ci si aspetta che si adotti un'altra intenzione ψ tale che $\varphi \wedge \psi$ sono esclusive. Se abbiamo l'intenzione di andare in piscina e abbiamo il desiderio di giocare anche a calcetto (che ripeto non sarebbe male) ci si aspetta che questo desiderio non diventi intenzione. Se abbiamo deliberato un'intenzione questa intenzione fa da *filtro* per successive intenzioni. Il desiderio si trasformerà in intenzione ma solo dopo aver soddisfatto la precedente intenzione.
3. Gli agenti monitorano il successo delle loro intenzioni e sono inclini a riprovare se i loro tentativi falliscono.
 - c. *monitorano il successo* → quando decidiamo che abbiamo l'intenzione di andare in piscina dobbiamo monitorare il corso delle azioni e lo stato di evoluzione che porta ad andare in piscina. Se falliamo in qualche modo (azione che non dovevamo fare), riproveremo a trovare un altro piano. Andiamo avanti fin tanto che quell'intenzione sia realizzabile. Se un agente al primo tentativo di raggiungere l'intenzione fallisce, allora continuerà a raggiungere l'intenzione fin tanto che non riesce a raggiungerla
4. Gli agenti credono che le loro intenzioni siano possibili.

¹Piccola nota. In JASON le intenzioni vengono rappresentate da dei record di attivazione

- d. se abbiamo l'intenzione di andare in piscina e perchè credo che la piscina sia aperta e che quindi sia possibile raggiungere tale obiettivo. Se crediamo che ci sia almeno un modo perchè l'intenzione possa, credo che esista almeno un modo perchè l'intenzione possa essere resa vera.
- 5. Gli agenti non credono che non porteranno avanti le loro intenzioni.
 - e. non è che adottiamo un'intenzione e dopo l'agente fa finta di niente. Non credo che non possa portarla avanti, io (inteso come agente eh) credo di portarla avanti. Non sarebbe razionale se credo che quell'intenzione non sia possibile. Io devo credere che sia possibile e se credo che non lo sia allora non deve diventare un'intenzione
- 6. In determinate circostanze, gli agenti credono che realizzeranno le loro intenzioni.
 - f. cioè è normale credere che io renderò vera l'intenzione. Le intenzioni possono fallire ma non è razionale pensare che sia inevitabile che falliscano. Possiamo avere anche l'intenzione di fare qualcosa e pensare di fare qualcosa per raggiungerla. Possibile che questa intenzione fallisca, ma in qualche modo non devo credere che sia inevitabile, altrimenti non sarebbe un'intenzione. La caratterizzazione dell'intenzione rispetto al desiderio è molto più forte. Noi potremmo avere benissimo un desiderio che credo irrealizzabile, ma non possiamo avere un'intenzione che credo che sia irrealizzabile. O meglio, possiamo averla ma non possiamo etichettarla come intenzione, ma semplicemente come desiderio
- 7. Gli agenti non devono necessariamente avere intenzione su tutti gli effetti collaterali delle loro intenzioni
 - g. in termini matematici le intenzioni non sono chiuse rispetto all'implicazione. Se credo che $\varphi \rightarrow \psi$ e io intendo φ non necessariamente intenda ψ .

Quest'ultimo problema è noto come effetto collaterale o problema del **package deal**. Potrei credere che andando al dentista comporta dolore e potrei anche avere intenzione di andare dal dentista ma questo non implica che intenda soffrire di dolore!

2.2 Realizzazione di agenti che sfruttano il practical reasoning

2.2.1 Agent control loop vers. 1

```
while not at end of this document do
  observe the world;
  update internal world model;
  deliberate about what;
    intention to achieve;
    use means-ends reasoning to;
    get a plan for the intention;
    execute the plan;
end
```

Algorithm 1: Agent control loop vers. 1

loop infinito - osserva il mondo - e poi aggiorna il modello interno del mondo. Dopo che deliberiamo quella che è l'intenzione che vogliamo raggiungere, usiamo il ragionamento (pianificazione) per ottenere un piano per l'intenzione - eseguo il piano - ricominciamo da capo. Questa è l'architettura base di un agente che sfrutta il PR.

PROBLEMA, il processo di deliberazione e means-end reasoning non sono **instantanei**.

Questi hanno un costo in termini di tempo. Osserva - *delibera* - *pianifico*, ma nel frattempo il mondo in questo lasso di tempo potrebbe modificarsi e questo significa che quando abbiamo ottenuto un piano potrebbe essere quello che non serve. Anche la fase di deliberazione potrebbe non essere più veritiera. Quindi, deliberare e ragionare sul pianificare hanno dei tempi, dove t_0 è il momento in cui l'agente inizia a deliberare, t_1 comincia a pianificare e t_2 incomincia ad eseguire il piano. Supponiamo che la deliberazione sia **ottimale**, cioè se seleziona un'intenzione da raggiungere, allora questa è la migliore per l'agente (massimizza l'utilità attesa)

- quindi, al tempo t_1 l'agente ha selezionato l'intenzione da raggiungere che sarebbe stata ottimale se fosse stata raggiunta all'istante t_0 . Cioè ad un certo punto è ottimale, è stata deliberata a t_1 ma in realtà era ottimale al tempo t_0 - si nota come sia passato del tempo. A meno che il tempo di deliberazione sia stato abbastanza piccolo, l'agente corre il rischio che l'intenzione non sia più ottimale nel momento in cui l'agente ha deliberato. Questo è noto come **calculative rationality**.

La fase di deliberazione comunque è solo metà del problema, l'agente deve ancora deliberare come realizzare l'intenzione e anche qui c'è del tempo che viene impiegato. In sostanza, l'agente avrà un comportamento **complessivo ottimale** solo nelle seguenti circostanze

1. Quando il tempo impiegato per i processi di deliberazione e di means-ends reasoning è **incredibilmente piccolo**, oppure
2. Quando il mondo è garantito rimanere **statico** mentre l'agente sta eseguendo i processi di deliberazione e di means-ends reasoning, in questo modo le ipotesi sulla scelta dell'intenzione da raggiungere e il piano per raggiungere l'intenzione rimangono validi fino a quando l'agente non ha completato tali processi; oppure
3. Quando un'intenzione ottimale se raggiunta al tempo t_0 (cioè il momento in cui si osserva il mondo) è **garantita rimanere ottimale** fino al tempo t_2 (cioè il tempo in cui l'agente ha determinato le azioni per raggiungere l'intenzione)

Queste sono condizioni non sempre possibili soddisfarle e per farlo bisogna cercare un buon compromesso. Uso di euristiche. Valutare se una certa intenzione potrebbe rimanere nel contesto attuale ottimale per il tempo necessario che io deliberi e, adattare strategie perché i tempi di esecuzione di deliberazione e pianificazioni siano più piccoli possibili idea è quella di pre-compilare dei piani.(graphplan è unidea)

2.2.2 Agent control loop vers. 2

```

/* initial beliefs */ B := B0;
while true do
  get next percept p;
  B := brf(B, p);
  I := deliberate(B);
  π := plan(B, I);
  execute(π)
end

```

Algorithm 2: Agent control loop vers. 2

Assumiamo che ci sia inizialmente un insieme di *belief* messi dentro un contenitore B. Insieme di *belief* iniziale soggetto ad una revisione in un secondo momento.

- il ciclo parte

- fa percezione dell’ambiente, prende tramite dei sensori/eventi/qualche strumento quella che è la percezioni dell’ambiente;
- applica una funzione **brf** (belief revision function), il cui obiettivo è prendere l’insieme corrente di credenze. Utilizza poi il risultato della percezione per filtrare/rivedere/aggiornare l’insieme dei propri belief. È un processo non banale. Se pensiamo che sono dei fatti (l’insieme delle credenze) innanzitutto questo implica che *labrf* è non monotona. Non è detto che se abbiamo una credenza precedente questa non possa essere cambiata e cambia appunto proprio in funzione delle percezioni, proprio per cui occorre rivedere le credenze ed eventualmente abbandonare alcune delle precedenti, o scegliere in base alle percezioni tra credenze contrastanti. *Prendo una cosa ma ne osservo un’altra*, quindi capire quale adottare. Se mantenere quella precedente o no.
- ottenuto l’insieme coerente di credenze che noi assumiamo, prendiamo queste credenze e andiamo a deliberare l’intenzione / basata proprio su queste credenze
- presa l’intenzione e prese le credenze pianifichiamo un piano, ovvero una sequenza di azioni da eseguire
- eseguo il piano

D. Belief, desire, intention (BDI), come delibera un agente?

- inizia cercando di capire quali sono le *opzioni* a sua disposizione (**desires**) sulla base delle proprie informazioni e credenze ()**beliefs**)
- sceglie tra le opzioni possibili e si impegna (**commit**) verso queste

N.B. Le opzioni scelte sono le **intenzioni**

SVANTAGGI: B, I , possono impiegare del tempo, proprio per cui le percezioni potrebbe anche cambiare.

2.2.3 Agent control loop vers. 3 - Deliberazione

```

/* initial beliefs */  $B := B_0;$ 
/* initial intentions */  $I := I_0;$ 
while true do
    get next percept  $p$ ;
     $B := brf(B, p);$ 
     $D := options(B, I);$ 
     $I := filter(B, D, I);$ 
     $\pi := plan(B, I);$ 
    execute( $\pi$ )
end

```

Algorithm 3: Agent control loop vers. 3

Assumiamo che inizialmente oltre ad un insieme di belief B possiamo avere anche delle **intenzioni** I , quindi obiettivi che l’agente ha sin dall’inizio. Se non dovesse averne basta mettere a 0 tale insieme.

- percepisce
- aggiorna il proprio insieme di B
- date le intenzioni che ha, i belief che ha, viene determinato un insieme di opzioni. Queste opzioni vengono etichettate come **desideri** D . La funzione *options* restituisce l’insieme delle possibilità che sono offerte tenendo conto delle precedenti intenzioni e belief correnti.

- dato l'insieme di B , D e I applichiamo un filtro (si ricorda come le intenzioni fungono anche da filtro per le future intenzioni) e determino il nuovo insieme di intenzioni
- tenendo conto dei B e avendo stabilito quale sia l'insieme delle intenzioni I possiamo pianificare il modo con cui raggiungiamo anche parzialmente qualcuna di queste intenzioni. C'è un ciclo, quindi non occorre che vengano eseguite subito tutte
- eseguo il piano

In particolare, gli ultimi due punti vengono visti come **impegno** a raggiungere l'intenzioni.

La funzione di deliberazione può essere scomposta in due distinte componenti:

- **option generation**, in cui l'agente genera un insieme di possibili alternative. Rappresenta la generazione di opzioni tramie una funzione, la *options*, che prende i correnti *beliefs* e *intentions* dell'agente e da esse determina un insieme di opzioni (*desideri - desires*)
- **filtering**, in cui l'agente sceglie tra alternative che competono e si impegna (*commit*) a raggiungerle. Per scegliere tra le opzioni concorrenti, un agente utilizza una funzione denominata *filter* (*filtro*)

Considerazione più dettagliata su questa versione - I Commitment

Quando un'opzione è restituita da *filter* e quindi scelta dall'agente con intenzioni, diciamo che l'agente ha fatto un **commitment** (infatti pianificazione - scelta dell'intenzione - commitment verso l'intenzione scelta), ha preso un impegno verso tale opzione. Il *commit* implica **persistenza temporale**. Il fatto che pianifico ed eseguo in questo particolare punto, l'agente sta proseguendo verso questa intenzione e questa persiste durante l'intenzione. L'intenzione, una volta adottata, non dovrebbe **immediatamente** essere abbandonata. *Se decidiamo una certa intenzione ci impegniamo*. Il problema in questa versione è capire come un'agente è impegnato verso le sue intenzioni

D.Quanto a lungo un'intenzione dovrebbe persistere?

D. Quali circostante possono cancellare un'intenzione?

R.Problema di questa versione. Dopo che abbiamo fatto un *commitment*(nel punto dove avviene la *filter*) il piano viene eseguito fino in fondo, ma quanto a lungo deve persistere questa intenzione? Se mentre faccio il piano, mi rendo conto di un qualcosa, è possibile cambiare idea? E soprattutto quali circostanze potrebbero far cambiare idea?

Commitment Strategies

Il meccanismo che un agente usa per determinare quando e come un'intenzione possa essere abbandonata è conosciuta come **commitment strategies**.

Considerazioni esempi dell'autore slide 27

1. robot che manca nell'impegno. Se abbiamo chiesto di far qualcosa ci si aspetta che l'impegno venga mantenuto (il robot in questo caso ha deciso di fare altro);
2. il robot porta una birra de merda che al cliente non piaceva. Il robot non ha fatto più percezione sulmondo, ovvero non ha considerato che il cliente non voleva quella birra;
3. la birra viene rotta volutamente dal robot mentre la stava portando. Ci si chiede. È giusto pensare che l'impegno valga finto tanto che sia possibile e che se non raggiungibile non si applica più applica più. Qui, nell'esempio il robot ha volutamente

rotto la bottiglia di birra e in questo modo non ha reso più possibile il soddisfacimento dell'obiettivo, ha reso perfettamente coerente il comportamento, cioè ha soddisfatto il suo obiettivo. Prima però abbiamo detto, a meno che non sia più possibile o meglio che non abbia più senso. Non è facile quindi stabilire fin tanto che sia possibile.

Sono state individuate diverse **commitment strategies**, ovvero decidere quando un'intenzione deve essere abbandonata.

- **blind commitment**, (*fanatico*) un agent *blind committed* continuerà a mantenere un'intenzione fino a quando non crederà che l'intenzione sia stata effettivamente **raggiunta**. Il *blind commitment* è talvolta indicato anche come **fanatical commitment**
- un agente *open-minded committed* manterrà un'intenzione fintanto che la ritiene, la **crede possibile**
- **single-minded commitment** (*risoluto*). Un agente *single-minded committed* continuerà a mantenere un'intenzione finché non ritiene che l'intenzione sia stata **raggiunta**, o che **non sia più possibile raggiungerla**.

Nella versione 3, possiamo concludere dicendo che un agente si impegna sia al fine *fine* (*ends*, cioè i desideri da realizzare), che ai *mezzi* (*means*, cioè al meccanismo attraverso il quale l'agente desidera raggiungere lo stato delle cose)

2.2.4 Agent control loop vers. 4

```

/* initial beliefs */ B := B0;
/* initial intentions */ I := I0;
while true do
    get next percept p;
    B := brf(B, p);
    D := options(B, I);
    I := filter(B, D, I);
    π := plan(B, I);
    while not empty(π) do
        α := hd(π);
        execute(α);
        π := tail(π);
        get next percept p;
        B := brf(B, p);
        if not sound(π, I, B) then
            | π := plan(B, I)
        end
    end
end

```

Algorithm 4: Agent control loop vers. 4

La versione 4 permette all'agente di **riplanificare** quando qualcosa va storto.

- viene fatta percezione
- applico la *brf*
- determina le possibili intenzioni
- l'agente si impegna verso una di queste

- trova il piano
- adesso applichiamo un modo diverso di esecuzione. Fintanto che il piano non è vuoto
 - prendo la testa del piano e la vedo come prima azione
 - la eseguo
 - considero le rimanenti e determino il nuovo piano.
 - vado a guardare quello che è il mondo
 - riconsidero quelli che sono i *belief*
 - se le intenzioni non sono più corrette rispetto al nuovo insieme di *B* allora
 - * abbiamo bisogno di ripianificare

L'idea è quella di fare un passo alla volta e ogni volta che lo facciamo andiamo a riconsiderare quanto fatto in termini di nuovo *B* e in questo chiediamo se le intenzioni *I* e il piano che rimangono sono ancora coerenti con il raggiungimento delle intenzioni. Questo non è detto che garantisca di raggiungerlo, ma in modo ipotetico facciamo un passo. Ancora *overcommitted* rispetto le intenzioni: non si ferma mai valutare se le sue intenzioni siano o meno adeguate.

Considerazioni sulla sound e sul plan

- *sound*, chiede solo di verificare se qualcosa è corretto rispetto ad un punto di partenza. Non dobbiamo trovare un piano e quindi di solito il check è molto rapido. Solitamente, almeno idealmente, ha un ordine di complessità quasi lineare perché si tratta di fare i passi simulandoli e verificare che sia tutto corretto;
- il *plan* è molto più complesso. Pianificare è qualcosa di estremamente complesso. Avremmo potuto fare/dire fai un passo e ripianifica, ma il problema è che la pianificazione è complessa, proprio per cui è preferibile usare un check e questo ci dice semplicemente quando applicare la pianificazione

Abbiamo di sicuro un miglioramento. Il problema che qui in vers. 4 stiamo rivalutando i mezzi ma non il fine. Vero che il piano potrebbe non essere più corretto ma non è detto che sia solo un problema di piano potrebbe essere che un nuovo contesto non renderebbe più il fine valido. Comunque diciamo che qui, l'agente risulta ancora *overcommitted* rispetto le intenzioni: non si ferma mai valutare se le sue intenzioni siano o meno adeguate.

2.2.5 Agent control loop vers. 5

```

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

```

Algorithm 5: Agent control loop vers. 5

Si nota come nel while interno vi è stata fatta una modifica. Fermarsi per determinare se le intenzioni hanno avuto successo o se sono diventate impossibili da soddisfare (*singles-minded commitment*). Qui come prima, ci chiediamo, ad ogni passo, se abbiamo finito con il piano (come succedeva prima), ma in più notiamo come andiamo a verificare se siamo già arrivati ad avere successo con l'intenzioni oppure è impossibile raggiungerla. Se impossibile, inutile andare avanti con il piano. Il problema che queste funzioni nel while non è che siano tanto semplici. Dimostrarle non è facili. Comunque, potremmo farle anche con delle *euristiche/approssimazioni*. Il nostro agente può riconsiderarre le sue intenzioni ogni volta che il controllo è sul ciclo esterno, ovvero quando:

- ha completamente eseguito un piano per raggiungere le intenzioni correnti; oppure
- 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. In questa versione andiamo a vedere se l'intenzione è impossibile da raggiungere oppure se è raggiunta. Abbiamo soltanto migliorato leggermente, ma non abbiamo ancora permesso all'agente di riconsiderare le intenzioni (I). Questo viene si fatto, ma solo all'interno del primo while.

2.2.6 Agent control loop vers. 6 - Riconsiderazione delle intenzioni

```

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

```

Algorithm 6: Agent control loop vers. 6

La riconsiderazione delle intenzioni viene aggiunta al fondo, subito dopo le *brf*. Dopo l'esecuzione di ogni azione, l'intenzione viene riconsiderata. La tripla (B, D, I) viene considerata anche sotto (ciclo while interno). Si nota come il piano potrebbe non essere corretto non solo perché quel particolare piano non permette più di raggiungere l'intenzione in quanto i B siano cambiati, ma perché è possibile che anche l'intenzioni stesse siano cambiate e quindi il piano non è più adeguato a quelle che erano le nostre intenzioni. Il problema di riconsiderare le I **costa**. Abbiamo un dilemma:

- un agente che non si ferma per riconsiderare le proprie intenzioni abbastanza spesso continueraà a tentare di raggiungere le sue intenzioni anche dopo che sarà chiaro che non possono essere raggiunte, o che non c'è più motivo per raggiungerle
- un agente che riconsidera costantemente le sue intenzioni può dedicare un tempo insufficiente a raggiungerle effettivamente, e quindi corre il rischio di non raggiungerle mai realmente

2.2.7 Agent control loop vers. 7 - Add funzione di riconsiderazione

```

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

```

Algorithm 7: Agent control loop vers. 7

In questa versione è stata aggiunta una **funzione di riconsiderazione dell'intenzioni reconsider**.

- B, I iniziali
- ciclo più esterno
 - percepisco
 - applico funzione brf
 - determino le opzioni
 - filtro quelle che sono le options e faccio commit
 - calcolo il piano
 - incomincio ad eseguire il piano ripetendo questo fin tant che il piano non sia vuoto, o non ho avuto successo nel raggiungere l'intenzione o determino che sia impossibile raggiungere l'intenzione
 - * prendo la prima azione
 - * eseguo
 - * determino come nuovo piano la tail
 - * percepisco
 - * rivedo i beliefs B
 - * a questo punto decido se è opportuno o meno riconsiderare le intenzioni I
 - se così fosse rivedo quelle che sono le intenzioni
 - filtro le nuove intenzioni
 - * adesso mi chiedo se il piano rimanente è ancora valido

- se così non fosse ripianifico e ricomincio il ciclo

La **reconsider** è una componente di controllo meta-livello, che decide se eseguire o meno la riconsiderazione. Ma ci si chiede, sulla base di che decido? Bisognerebbe considerare il dominio e applicare delle euristiche legate al dominio.

Riconsiderazione ottimale delle intenzioni

Kinny e Georgeff investigano in maniera sperimentale l'efficacia delle strategie di riconsiderazione delle intenzioni. Hanno individuato due tipi di strategia di riconsiderazione:

- **bold agents**, (*audaci*), agenti che non si fermano mai a riconsiderare le intenzioni
- **cautions agents** (*cauti*), agenti che si fermano a riconsiderare le intenzioni dopo ogni azione.

Il **dynamismo** dell'ambiente è rappresentato dal **tasso di cambiamento del mondo** γ . I loro esperimenti hanno prodotto due tipi di risultati che si dimostreranno non sorprendenti. Nella loro simulazioni hanno determinato che tutto dipende, ovvero la bontà di adottare una strategia anzichè un'altra (agente cauto o audace), dal tasso di modifica/dynamismo dell'ambiente

- se γ è basso (cioè, l'ambiente non cambia rapidamente), gli agenti *audaci* fanno meglio rispetto a quelli *cauti*. Questo perché sono quelli cauti a perdere tempo a riconsiderare i propri impegni mentre agenti audaci sono impegnati a lavorare per raggiungere le loro intenzioni
- se γ è alto (cioè, l'ambiente cambia frequentemente), gli agenti *cauti* tendono a superare gli agenti audaci. Questo perché sono in grado di riconoscere quando le intenzioni sono perse, traendo anche vantaggio da situazioni fortuite e nuove opportunità quando si presentano

2.3 Procedural Reasoning system

Il **procedural reasoning system** (PRS) fu la prima architettura per agenti sviluppata ad includere il paradigma *belief-desire-intention*. Architettura sviluppata da M. Georgeff e A. Lansky. Il PRS, ha avuto diverse applicazioni significative (es. sistema di controllo del traffico aereo per l'aeroporto di Sydney). In PRS gli agenti non pianificano, utilizzando una libreria di piani precompilati.

I piani in PRS

I piani sono costruiti (manualmente) dal programmatore e sono caratterizzati da:

- **goal**, la post-condizione del piano
- **contesto**, la pre-condizione del piano
- **corpo**, la sequenza di azioni da eseguire

Abbiamo inteso l'esecuzione di un piano come una sequenza di azioni. I piani in PRS possono essere anche più complessi, in particolare possono contenere a loro volta dei *goal*. I piani in PRS possono contenere disgiunzione di *goal*, *loops*, ecc.

Un agente PRS ha un insieme di piani, alcuni *belief* iniziali riguardo il mondo. I *belief* in PRS sono rappresentati come fatti prolog-like. Un agente PRS ha un *goal* iniziale

2.3.1 Procedural Reasoning System: stack delle intenzioni

- Quando un agente inizia la sua esecuzione, il goal iniziale è inserito in uno stack, lo stack delle intenzioni (*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-condizioni
- tra questi piani alcuni avranno le loro pre-condizioni soddisfatte dall'insieme di *belief* correnti
- l'insieme di piani (*i*) permettono di raggiungere il goal e (*ii*) hanno le precondizioni soddisfatte, diventano l'insieme delle possibili **opzioni** dell'agente.

Processo di selezione dei vari piani

- il processo di selezione tra i diversi piani è la *deliberazione*
- la deliberazione è ottenuta mediante l'utilizzo di un piano di meta-livello
- Effettuata la scelta del piano, la sua esecuzione può portare ulteriori goal nello stack delle intenzioni, che a loro volta possono portare ulteriori goal nello stack delle intenzioni e così via
- le azioni atomiche potranno essere eseguite direttamente dall'agente
- se un piano fallisce, l'agente può selezionare un nuovo piano tra quelli candidati.

2.3.2 AgentSpeak(L) e Jason

- AgentSpeak(L): linguaggio di programmazione per agenti BDI introdotto da Rao nel 1996
- Si propone di colmare il gap tra specifica teorica ed implementazione di un agente BDI
- Jason è la prima significativa implementazione di AgentSpeak(L), realizzata in Java da R. H. Bordini e J. F. Hubner

Chapter 3

Sistemi Multiagente e Comunicazione tra Agenti

In questo capitolo tratteremo i sistemi multiagenti. Fino ad adesso abbiamo parlato di agenti singoli e architetture BDI per la realizzazione di un'agente. Adesso tratteremo il problema dei **sistemi multiagenti**, con particolare riferimento alla comunicazione tra agenti. In particolare parleremo della comunicazione tra agenti e vedremo come la comunicazione è un elemento che ha tratto estrema attenzione. Noi lo confronderemo con quelli che sono altri paradigmi di comunicazioni/interazione tra componenti e vedremo perché la comunicazione ha una grossa rilevanza. In seguito vedremo:

- **Jade**, sistema realizzato da Telecom. È un progetto open-source. Focalizzato soprattutto su aspetti di comunicazione tra componenti. Utilizzato anche in altre piattaforme proprio per la sua solidalità in fase di comunicazione;
- **Jason**, permette la realizzazione pratica di un PRS.

Jason, è focalizzato sulla semantica di un agente singolo, ma può anche interagire avvolendosi di Jade come infrastruttura di comunicazione. Infine vedremo Jade come rappresentante per la realizzazione di comunicazione e componenti.

3.1 Perchè sistemi distribuiti d'agenti?

Perchè è un buon paradigma per realizzare e affrontare sistemi distribuiti. Spesso quando abbiamo a che fare con soluzioni distribuite, le varie componenti tendiamo ad assumerle come **autonome**, questo perchè nella programmazione dei sistemi due sono i principi per realizzarli:

- **basso accoppiamento** tra agenti. Evitare l'accompiamento è uno dei principali obiettivi che abbiamo. Non vogliamo che una componente dipenda da un'altra, si tende a disaccoppiare, dove la **comunicazione** giocherà un forte ruolo fondamentale;
- **forte coesione**, dove ogni agente persegue degli obiettivi e considera quei particolari obiettivi come l'elemento costituente che lo definisce. In qualche modo un agente è solo ciò che serve per raggiungere quell'obiettivo e quindi ha una coesione forte.

Questo ci permette di vedere sistemi distribuiti come un aggregazione di agenti che interagiscono. Sistemi come Jade creano la *colla* per mettere assieme questi elementi, con le caratteristiche appena citate. Le componenti di per se possono essere anche *eterogenee*, non devono essere viste sempre come un insieme di agenti BDI che interagiscono. Possono essere anche normalissime componenti software. Jade come piattaforma decide di focalizzarsi sulla sola comunicazione lasciando ampia libertà nell'implementazione di agenti. Essendo semplice codice JAVA può essere semplicemente agganciato ad altri sistemi che si interfacci. Questo ci permette di realizzare sistemi distribuiti geograficamente, composte da parti indipendenti,

anche di grosse dimensioni. La Telecom, aveva utilizzato Jade per fare i primi sistemi di distribuzione della linea ADSL.

3.2 Distributed Articial Intelligence (DAI)

Dicevamo, modularità, distribuzione, astrazione e anche intelligenza. Se le componenti sono programmabili con un livello che noi associamo all'intelligenza, perchè no questo ci facilita la loro programmazione e quindi la realizzazione di sistemi ancora più complessi in maniera più facile. Questo fa parte di quella che si chiama **intelligenza artificiale distribuita (DAI)**. Sistemi multiagenti e agenti fanno parte di questo ambito. L'approccio tipico della DAI è proprio quello del paradigma ad agenti.

3.2.1 Sistemi autonomi di agenti

Cosa fanno gli agenti in un sistema distribuito? Possono competere tra loro, possono cooperare e coordinarsi. In particolare, parleremo di

- **coordinazione**, quando ad esempio gli agenti che fanno parte di un sistema devono collaborare ad esempio per un goal comune;
- **negoziacionem** quando bisogna conciliare quelli che sono gli interessi singoli di agenti rispetto all'interesse collettivo del sistemaM
- **competizione**, quando gli agenti hanno solo interessi personali e competono per le risorse.

I **sistemi multiagenti** sono il modo migliore per caratterizzare o progettare sistemi distribuiti di computazione utilizzando metafore che si avvicinano a quelle della società.

3.2.2 La comunicazione

La comunicazione gioca un ruolo fondamentale. Gli agenti operano tipicamente in ambienti e questi ambienti ospitano altri agenti. L'**ambiente** è anche un mezzo per realizzare la comunicazione. Questa viene vista come l'elemento caratterizzante dei sistemi multiagenti.

1. deve essere realizzata mediante un'opportuna **infrastruttura** che ne specifichi le caratteristiche della comunicazione;
2. si occupi del **trasporto** dei messaggi e quindi offrire dei protocolli **aperti** e **distribuiti** in modo da poter accogliere agenti eterogenei e sistemi diversi che possono entrare in contatto con questa infrastruttura;
3. gli agenti ospitanti devono mantenere la caratteristica di essere **autonomi**. L'infrastruttura di comunicazione non deve assolutamente imporre un rilascio della caratteristica di **autonima, self-interested, o cooperativi**

Gli agenti comunicano con il fine di raggiungere i loro obiettivi o quelli della società/sistema in cui esistono

3.2.3 Le infrastuture della comunicazione

Le infrastrutture includono:

- **protocolli di comunicazione**, devono permettere agli agenti di comprendere i messaggi scambiati. Non è come dire stiamo in una stessa piattaforma di programmazione. Prendiamo JAVA e quindi ci scambiamo i mesaggi facendo delle invocazioni di metodi, il

cui formato e la firma sono quelle definita dal linguaggio stesso. Qui, gli agenti sono autonomi e anche eterogenei. Dobbiamo trovare un linguaggio comune che permetta lo scambio di messaggi. Dobbiamo trovare un modo che ci sia accordo su quella che è la struttura del messaggio, ma non solo, anche sulla lingua del messaggi. Ci potrebbe essere una comunicazione, dove gli agenti comunicano, quindi utilizzano termini all'interno dei messaggi. Il problema è capire se entrambi gli agenti attribuiscono lo stesso significato ai termini utilizzati all'interno del messaggio. La semantica di un linguaggio di programmazione distribuito è molto più semplice. Viene stabilito che un messaggio corrisponde ad un'invocazione di un metodo e quei parametri passati devono essere del linguaggio stesso. In un messaggio scambiato tra agenti invece i termini sono semplicemente delle sequenze di carattere, proprio per cui occorre necessariamente definire un'ontologia;

- **protocolli di interazione**, per permettere agli agenti di svolgere conversazioni, ossia scambi strutturati di messaggi. Non c'è un solo messaggio, ma anche sequenze di messaggi per ottenere un certo messaggio

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

- proposta di esecuzione di un'azione
- accettare una proposta di esecuzione di un'azione
- rifiutare una proposta di esecuzione di un'azione
- ritirare una proposta di esecuzione di un'azione
- esprimere disaccordo rispetto una proposta di esecuzione di un'azione
- proporre una differente esecuzione di azione

Abbiamo parlato di **ontologie**. Queste permettono di usare termini nel messaggio e indicare quando si trasmettere il messaggio a quale vocabolario stiamo facendo riferimento. In Jade, ad esempio, si permette di stabilire anche per i contenuti dei messaggi le ontologie utilizzate e agganciarle al messaggio stesso. L'agente che riceverà il messaggio potrà comprendere meglio il contenuto. Jade usa **FIPA ACL** e decide di adottare questo linguaggio in modo da impostare uno standard sulla struttura dei messaggi in modo che siano di facile comprensione. La comunicazione inoltre permette agli agenti di coordinare le loro azioni e i loro comportamenti. Per poterlo fare deve aver bisogno naturalmente di **ricevere**, **inviare** messaggi.

- **ricevere**. La ricezione viene vista come se fosse una **percezione**. Percepisco dall'ambiente un messaggio, dove l'ambiente non è altro che il mezzo con cui i messaggi vengono scambiati. In questo modo, se leggiamo *ricevere messaggio* come percezione, non andiamo a modificare il ciclo di vita dell'agente. L'agente percepisce e agisce e quando agisce esegue delle azioni.
- **invio**. Possiamo leggere l'invio di un messaggio come l'esecuzione di un agente, in questo modo il tutto rimane coerente con il modello di agenti introdotte nelle prime righe di testo del "LIBRO".
- **coordinazione**. Proprietà del sistema di agenti che eseguono attività in un ambiente condiviso, dove l'esecuzione di queste attività è fatta in maniera coordinata;
- **cooperazione**. Particolare forma di coordinazione tra agenti non antagonisti. In questo ambito ricadono le problematiche di *cooperative distributed problem solving*¹, *task sharing, distributed planning*;

¹Quando abbiamo un problema da affrontare e il problema è la distribuzione di attività tra un insieme di componenti che devono cooperare per contribuire alla soluzione del problema iniziale.

- **negoziazione.** coordinazione tra agenti antagonisti, competitivi o semplicemente *self-interested* (un esempio sono le ASTE). Con il professore Martelli vedremo un esempio di aste. Non sono altro che delle negoziazioni come coordinazione tra agenti, dove ognuno è interessato per se e quindi **non** ha come obiettivo, un obiettivo comune da raggiungere, ma ognuno ha comunque bisogno degli altri per raggiungere il proprio obiettivo. L'interazione è essenziale per raggiungere il proprio obiettivo.

3.3 Linguaggi per la comunicazione tra agenti

Quello che andremo a fare sarà confrontare due paradigmi, quello **object-oriented** e quello **agent-oriented**

3.3.1 Perchè comunicare

Consideriamo 3.1, dove abbiamo un oggetto O_1 che invia un messaggio ad un oggetto O_2 . L'invio avviene tramite la chiamata $O_2.m(args)$. Quello di cui vorremmo discutere è un po cosa succedere a livello di agenti. Ci si chiede, chi prende la decisione di eseguire il metodo m ? Ovviamente O_2 lo esegue, ed è lui che decide di eseguire il metodo? In realtà, O_2 lo esegue ma la decisione di eseguire il metodo risiede in O_1 . Quindi, O_2 **non ha nessun controllo** sul proprio comportamento, la decisione è di un altro agente (in questo caso O_1), non sua. In altri termini, un sistema ad oggetti non è realizzato da **componenti autonome**. Sono incapsulate, cioè come si vede lo stato è incapsulato da un'interfaccia di un metodo, ma questo non vuol dire che sono autonome, proprio perchè la decisione di esecuzione del metodo m non risiede nell'oggetto stesso, ma fuori. Questo, che sia un sistema ad oggetti passivo (come JAVA), che sia un sistema ad oggetti attivo (come H), non fa nessuna differenza. La **decisione** è presa da qualcun altro.

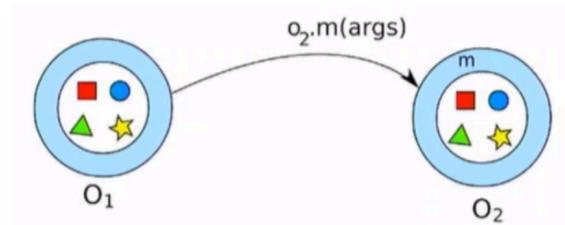


Figure 3.1: Paradigma Object-Oriented (OO)

- O_2 invia dei messaggi ad O_1 ;

Questo 3.2 a differenza di quello di prima è un paradigma Agent-Oriented, dove ci aspettiamo che la decisione sia autonoma. Non possiamo invocare, ma quello che possiamo fare è solo **chiedere**. Mandare un messaggio - *correi che eseguissi l'azione a* - l'altro agente in qualche modo decide se compiere autonomamente l'azione oppure no. Vorremo realizzare un sistema fatto in questo modo perchè qui vi è un disaccoppiamento più forte. Il sistema in cui si dice *si, forse, no* è più disaccoppiato perchè l'agente A_1 non controlla l'agente A_2 .

- A_2 è un agente autonomo: ha il controllo sia del proprio stato che del proprio comportamento;
- A_1 non ha la certezza che A_2 eseguirà l'azione a perchè lui lo ha chiesto.

Gli agenti non possono sforzare altri agenti ad eseguire azioni. Quello che possono fare è di eseguire **azioni comunicative** al fine di *influenzare* gli altri agenti in modo opportuno, comunicando una richiesta, interagendo con agenti che in qualche maniera sono tutti interessanti a raggiungere un comune obiettivo. In un'asta, c'è un comune obiettivo *andare a buon fine*

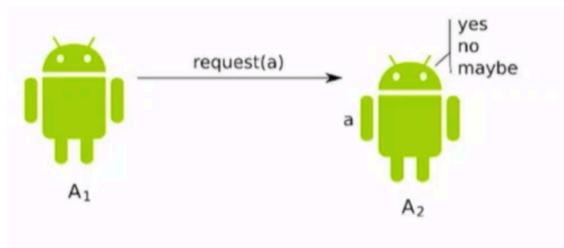


Figure 3.2: Paradigma Agent-Oriented (AO)

la trattativa. Obiettivo comune per chi vende l'oggetto e chi vuole comprare l'oggetto. C'è comunque un fine comune - guadagnare di più e spendere bene. Anche in sistemi che competono l'interesse comune, l'obiettivo è quello che l'interazione arrivi in un punto in cui gli obiettivi singoli (almeno per qualcuno) siano soddisfatti.

3.3.2 Atti comunicativi

Il linguaggio parlato è spesso usato come modello per la comunicazione tra agenti. Negli anni '60, Austin e Searle introducono lo **Speech Act Theory** (teoria degli atti comunicativi), dove vedono ogni comunicazione come l'esecuzione di un'azione. Es. *Ti dichiaro colpevole*. Qualunque comunicazione è vista come un'**azione**, ma allora se un'azione ha degli effetti comunicativi (abbiamo visto che se eseguite hanno l'effetto di modificare l'ambiente), qual è il loro effetto? Austin e Searle vedono l'effetto delle azioni comunicative in termini di **modifiche di stati mentali**. Ogni comunicazione va a modificare *credenze*, *desideri*, *intenzioni* di chi è coinvolto nella comunicazione. Data una comunicazione e dato un certo contesto l'effetto sarà una modifica dello stato mentale (dovuto alla credenza dell'informazione) e a questo punto potremmo utilizzare la comunicazione per pianificare ad esempio l'approccio all'esame (nel caso in cui la comunicazione era quella dell'appello dell'esame).

3.3.3 Struttura di un messaggio nella teoria degli atti comunicativi

Searle e Austin, hanno individuato tre aspetti in un atto comunicativo:

- **locuzione**, atto fisico compiuto da chi parla per produrre l'enunciato;
- **illocuzione**, il significato inteso dell'enunciato da parte del parlante, ovvero quello che il parlante desidera esprimere;
- **perlocuzione**, l'azione che risulta dalla locuzione nell'ascoltare.

L'illocuzione nella speech act theory viene anche chiamata **performativa** per identificare la forza illocutoria dell'enunciato. Il significato di una comunicazione viene categorizzata e associata ad una categoria come *promesse*, *comunicazioni*, *richieste*. Vedremo ad esempio come standard **FIPA ACL** vanno a stabilire una categoria di performatives, quindi forze illocutorie usate per esprimere il significato del messaggio.

Esempio di applicazione

- una stessa frase - *the doors is closed* - con performatives diverse, ha dei risultati diversi
 1. se associo la performativa *request* al contenuto, allora l'effetto risultante che vogliamo ottenere da chi riceve il messaggio è un qualcosa come *per favore chiudi la porta*;
 2. se associo la performativa *inform* al contenuto, allora l'effetto risultante è una comunicazione. Chi riceve il messaggio sa che la porta è chiusa;

3. se associo la performativa *inquire*, l'effetto è chiedere se la porta è chiusa. Nessuna modifica dello stato, ma un'interrogazione dello stato.

Condizioni per il successo di una comunicazione

Poichè la comunicazione possa avvenire , Austin dice che devono essere necessarie delle condizioni (note come *condizioni di felicità*). Deve esistere una procedura convenzionale accettata per la performativa, dove circostanze e persone coinvolte devono essere a conoscenza di tale procedura. Questa deve essere eseguita correttamente e completamente e l'atto deve essere *sincero* (cosa più interessante) e soprattutto la comprensione deve essere completa. Austin aggiunge che per poter comunicare, è imprescindibile la sincerità di chi sta parlando, altrimenti la comunicazione diventerebbe ambigua e chi riceve il messaggio deve avere una completa comprensione dello stesso e delle sue parti per utilizzarla. Questa è una parte **complicata** da **raggiungere**, ed è qui che gli standard si preoccupano di lavorare. Vedremo che l'elemento della **sincerità** non è così secondario. Anche nella realizzazione dei sistemi, il fatto che venga comunicata un'informazione e che possa utilizzarla, ha come base la **sincerità**. *Se ci viene detto che l'esame è il 10 di Giugno, si parte dal presupposto che noi studenti attribuiamo della sincerità verso chi ha comunicato il messaggio.*

3.3.4 Modellazione di atti comunicativi

Gli atti comunicativi che vengono introdotto nell'ambiente ad agenti vengono accompagnati da una **semantica**. Semantica che permette di comprendere il significato. DARE UN SIGNIFICATO PRECISO AGLI ATTI COMUNICATIVI. Quello che fanno Coen e Perrault è quella di associare una semantica molto vicino a quella che può essere la rappresentazioni di azioni che un robot esegue in un ambiente con *pre-condizioni* ed *effetti*. Un esempio è il linguaggio *STRIPS*.

- Request(S, H, α). La request di S verso H per un'azione α .
- pre-condizioni: $(S \text{ BELIEVE}(H \text{ CANDO } \alpha)) \wedge (S \text{ BELIEVE } (H \text{ BELIEVE } (S \text{ WANT } \alpha)))$ S crede che H possa fare l'azione e S crede che H crede che S vuole che venga eseguita l'azione α ;
- effetti: $(H \text{ BELIEVE } (S \text{ BELIEVE } (S \text{ WANT } \alpha)))$ H crede che S crede che S vuole che venga eseguita α

L'obiettivo di usare pre-condizioni ed effetti è quello di poter pianificare dei **dialoghi** in modo che ci viene permesso costruire un dialogo in modo da ottenere il nostro obiettivi complesso interagendo con un altro agente.

Questo tipo di visione è stato preso in considerazione da due linguaggi FIPA ACL e KQML. Entrambi sono degli Agent Communication Language

3.3.5 Agent Communication Languages (ACL)

Un **ACL** fornisce agli agenti mezzi per scambiarsi informazioni e conoscenza. Ha un livello più alto rispetto agli strumenti di comunicazione per i sistemi distribuiti. Un ACL tratta con *proposizioni, regole, azioni* anzichè solo *oggetti*. Descrive uno stato desiderato attraverso linguaggio dichiarativo e spesso lo tratta in termini di **attitudini mentali**.

3.4 KQML e FIPA

3.4.1 Origini (di un ACL) e Knowledge Sharing Effort

KQML ha origine nel progetto **Knowledge Sharing Effort (KSE)** avviato da DARPE con l'obiettivo di sviluppare tecniche, metodologie e strumenti software per la condivisione della conoscenza e il riutilizzo della conoscenza. **Condivisione** e **conoscenza** richiede la comunicazione, che a sua volta richiede un linguaggio. KQML voleva essere un linguaggio ad alto livello (messo in cima alla pila dell'interazione) utilizzava **KIF**, linguaggio logico basato sulla logica del primo ordine che permetteva di esprimere proprietà di oggetti in una base di conoscenza, tipicamente scritto a fatti. Definiva anche un linguaggio per definire ontologie condivise proprio come detto prima, dove i termini per poter comprendere il significato facevano riferimento ad una ontologia. KMQL è utilizzato dentro JASON e questo utilizza il formato di KMQL come struttura dei messaggi.

3.4.2 Knowledge Interchange Format (KIF)

La componente KIF è un linguaggio che permette di dichiarare:

- proprietà di cose in un dominio;
- relazioni tra cose in un dominio;
- proprietà generali di un dominio;

Si nota come tutto ciò era ispirato da una logica del primo ordine.

3.4.3 Knowledge Query and Manipulation Language (KQML)

- KQML, è un linguaggio di comunicazione di alto livello e indipendente da:
 - trasporto (TCP/IP, RMI,...);
 - il linguaggio in cui è espresso il contenuto (KIF, Prolog,...);
 - l'ontologia utilizzata
- Un messaggio KQML specifica il tipo di messaggio (**performativa**)
- KMQL ignora la porzione di messaggio che fa riferimento al contenuto.

Il linguaggio si basa su una notazione a lista simile a quella utilizzata da linguaggio Lisp e consiste in una performativa seguita da un numero di coppie chiave/valore.

- **:sender**, il mittente della performativa
- **:ricevente**, il destinatario della performativa
- **:reply-with**, se il mittente si aspetta una risposta, rappresenta l'identificatore della risposta
- **:in-reply-to**, l'identificatore della risposta specificata da **:reply-with**
- **:language**, il linguaggio di rappresentazione del **:content**
- **:ontology**, l'ontologia assunta dal parametro **:content**
- **:content**, il contenuto del messaggio

Figure 3.3: Parole chiavi riservate di KQML

- **achieve**, il *sender* vuole che il *receiver* esegua qualcosa
- **advertise**, il *sender* afferma di essere adatto a gestire una certa performativa
- **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*

Figure 3.4: Alcune performativhe di KQML

Un problema di KQML è che le performativhe (elemento base per dare un significato ad un messaggio) non era stato standardizzato e quindi il fatto di non aver preso una decisione portò ad implementazioni che non si capivano. L'interazione veniva difficile proprio per l'utilizzo di performativhe diverse. Insieme di performativhe utilizzate non coprivano tutti i possibili utilizzi. Al contrario FIPA-CL, invece standardizzava le performativhe (sin dall'inizio). Siamo pochi anni dopo la nascita di KQML.

3.4.4 KQML facilitators

Una delle cose che KQML fa invece molto bene e in maniera interessante erano i **facilitatori di comunicazione**. Per poter inviare ad esempio un email occorre sapere il destinatario. Lo stesso deve poter fare un'agente per poter comunicare con un altro agente, ovvero deve sapere l'identificazione/nome dell'agente e questa conoscenza deve essere data nell'atto di progettazione. Questo crea delle problematiche su sistemi aperti. Allora KQML ha introdotto i facilitatori di comunicazione - intermediari della comunicazione. Sono delle figure in cui un **broker** si occupa di instradare dei messaggi a degli opportuni agenti che lui conosce (lista d agenti). Anzichè scrivere e inviare un messaggio direttamente all'agente, il messaggio viene inviato ad un broker e questo si preoccupa ad inviare quel messaggio a agenti con particolari caratteristiche.

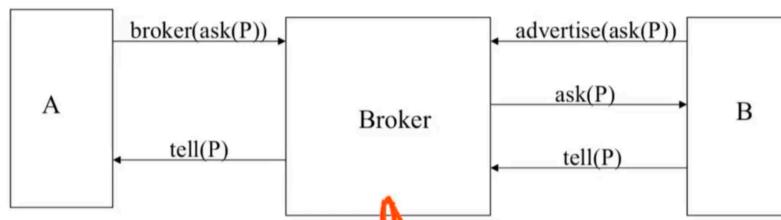


Figure 3.5: Facilitatori di comunicazione - Il Broker

3.4.5 Semantica di KQML

- Inizialmente KQML non disponeva di una semantica.
- Successivamente Labrou e Finin introducono una semantica per KQML in termini di **precondizioni**, **postcondizioni**, e **condizioni di completamento**
 - Dato un mittente **A** e un destinatario **B**, le **precondizioni Pre(A)** descrivono lo stato necessario per **A** al fine di inviare la performativa, e le **precondizioni Pre(B)** descrivono per **B** per accettarla e elaborarla con successo
 - Dato un mittente **A** e un destinatario **B**, le **postcondizioni Post(A)** descrivono lo stato di **A** dopo aver espresso la performativa e le **postcondizioni Post(B)** descrivono lo stato di **B** dopo aver ricevuto il messaggio

- Le **condizioni di completamento** di una performativa descrivono lo stato finale dopo, per esempio, che una conversazione sia stata completata

Tutte e tre permettono di descrivere lo stato degli agenti in un linguaggio basato su **sistemi mentali** espressi mediante attitudini (*beliefs, knowledge, desire* e *intention*) e descrizioni di azioni (per inviare e elaborare un messaggio).

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

3.4.6 FIPA ACL

Nel 1996, qualche anno dopo la nascita di KQML è nata FIPA ACL. Fondata con il fine di produrre degli **standard** per il software per agenti interagenti ed eterogenei e sistemi basati su agenti. Tra le specifiche, la FIPA ha definito un linguaggio per permettere la comunicazione per agenti (**FIPA ACL**) simile a KQML. FIPA ACL, a differenza di quello che succedeva con KQML, prevede **22 atti comunicativi** (*inform, request, agree, query-if...*). Cosa fondamentale e importante che queste performative sono state **standardizzate** e ovvia le problematiche introdotte da KQML. A confronto con KQML, sono presenti altre differenze:

- **semantica.** Se KQML non aveva una semantica definita, FIPA ACL ha una semantica. Questa è basata su **stati mentali**, ma anche qui rimane lo stesso problema. Se gli agenti non sono di tipi BDI, come si riesce a fare il mapping con gli altri stati mentali? La problematica, così come KQML persiste anche qui;
- **mancanza di primitive di tipo facilitator** (broker come KQML). Questi sono stati sostituiti da altri strumenti come le **pagine gialle**. Sebbene non siano potenti e eleganti come quelli di KQML, queste sono più pratiche e facili da realizzare;
- FIPA a differenza di quello che succedeva in KQML introduce anche un **test di conformità** (lo vedremo più avanti. TRANQUILLI :).

```
(inform
  :sender agent1
  :receiver agent2
  :language Prolog
  :content "weather(today, raining)"
)
```

Figure 3.6: Esempio di un messaggio tipico FIPA ACL

3.4.7 Semantica di FIPA ACL

La semantica in FIPA ACL è utilizzata con la *Semantic Language (SL)*, derivata da una **logica multimodale quantificata** sviluppata da Sadek (ricercatore di Telecom France).

- $B_i\varphi$, *i crede* φ
- $U_i\varphi$, *i è incerto* su φ ma pensa che φ è più probabile di $\neg\varphi$
- $C_i\varphi$, *i desidera* (choice, goal) che φ valga correntemente

Tramite questa logica multimodale quantificata viene dato uno specifico significato a tutte le performative, permettendo di ragionare non solo sugli stati mentali ma anche sulle **azioni** che si possono compiere. Vengono introdotti tre operatori per ragionare sulle azioni:

- **Feasible**(a, φ), a può occorrere e se occorre, φ sarà vera dopo tale occorrenza

- **Done** (a, φ), a è appena occorsa e φ è vera dopo tale occorrenza
- **Agent**(i, a) i è il solo agent che esegue l'azione a

Si nota come a differenza di quello che succedeva con KQML, dove tutto era basato sui soli stati mentali e basta, qui entrano in gioco le azioni che vengono eseguite. Sulla base di belief, choice ed eventi viene definito il concetto di **goal persistente** $PG_i\varphi$. L'intenzione $I_i\varphi$ è definita come un goal persistente che impone all'agente di agire. Infine la semantica degli atti comunicativi è specificata come un insieme di formule SL che descrivono le precondizioni di fattibilità (*feasibility preconditions*, noti anche come FP) e gli effetti razionali (*rational effects*, noti anche come RE).

- **Feasibility precondition (FP)**, condizioni che devono valere per il mittente. Non valgono per il destinatario. In KQML sia le precondizioni che le postcondizioni (se non dovesse essere postcondizioni dite RE) valevano sia per il mittente per il destinatario.
- **Rational effects (RE)**, gli effetti che un agente può aspettarsi che occorrano come risultato dell'esecuzione di una azione (la ragione per la quale l'azione è selezionata). All'agente destinatario non è richiesto di garantire che occorra l'effetto atteso.
- quello che introduce FIPA ACL è il **test di conformità**. Si intende quando un agente A esegue un **atto comunicativo** C dove i FP per A devono valere mentre i RE non sono garantiti e sono quindi irrilevanti al fine della conformità. I RE non sono importanti per dire che un'azione è conforme rispetto alla semantica.

$\langle i, \text{INFORM}(j, \varphi) \rangle$
FP : $B_i\varphi \wedge \neg B_i(Bif_j\varphi \vee Uif_j\varphi)$
RE : $B_j\varphi$
FP significa che i deve credere φ e i deve non credere che j già conosce φ o $\neg\varphi$ (ossia $Bif_j\varphi$). oppure j è incerto su φ o $\neg\varphi$ (ossia $Uif_j\varphi$).
$\langle i, \text{REQUEST}(j, a) \rangle$
FP : $FP(a)[i \setminus j] \wedge B_i\text{Agent}(j, a) \wedge B_i\neg PG_j\text{Done}(a)$
RE : $\text{Done}(a)$
dove $FP(a)[i \setminus j]$ denota la parte delle FPs di a che sono attitudini mentali di i .

Figure 3.7: Esempio di semantica in FIPA ACL

Quello che si nota dalla semantica che in FIPA ACL è bastato definire un INFORM, REQUEST e poi tutte le altre performatives sono definite sulla base della REQUEST e dell'INFORM. Non va ad inventarsi una semantica specifica per ogni specifica, ma ogni performativa viene costruita sempre sulla base di alter performatives e alla base c'è sempre una REQUEST e un'INFORM. È **modulare** e il test di conformità è facile da realizzare e implementare.

3.4.8 Il test di conformità

FIPA dice che se si vuol essere *compliance* e conformi a questo standard si deve passare un test, dove questo ha due livelli:

- **sintattico**, quando si manda una comunicazione, questa deve essere sintatticamente rispettata;
- **conformance**, basata sulla semantica, dove questa deve essere rispettata

PROBLEMA: determinare se un agente rispetta la semantica del linguaggio non è banale. Se abbiamo un agente BDI allora possiamo andare a vedere facilmente se i beliefs, intenzione, goal e azioni coinvolte rispettano nella comunicazione quello fatto (e questo sappiamo che si può fare). Ma se abbiamo un programma convenzionale (e non un agente BDI) come faccio a verificarlo? **Woolridge**, propone un approccio. Prende un certo linguaggio di riferimento/programmazione e va a mappare quello che c'è nel programma con dei concetti BDI. Tipi:

- se è una credenza, probabilmente sta dentro una variabili e quindi si può assumere che il valore di una variabili ad un certo momento rappresenti il suo belief.

L'autore non fa altro che prendere la verifica dei programmi e va a mappare questi concetti della verifica dei programmi in concetti di un agente BDI in modo da poterne verificare.

Ora guardiamo al contrario. Il nostro problema è quello di essere **conforme**.

- mandiamo un messaggio e possiamo verificare se io agente rispetto questo, ma il conformance testing si chiede come è fatto questo test. Ed è proprio qui che si concentrano le critiche verso le semantiche ACL

Le critiche verso le semantiche ACL

Questi ACL nati da studi filosofici erano basati sulla semantica degli atti comunicativi, come *azioni* che vanno a modificare lo stato mentale e che hanno un'ottima interpretazione e offrono una metafora vicina a quella che è il *linguaggio umano*, ma **falliscono** per un utilizzo pratico, proprio perché è difficile verificare. È stata proposta un'alternativa.

3.4.9 La semantica sociale per la comunicazione

Alla fine degli anni 90 è stata proposta una semantica sociale per la comunicazione. Nella proposta si dice che la semantica basata su stati mentali è adeguata per sistemi di agenti cooperanti ma presenta molti problemi quando il sistema è composto da agenti eterogenei competitivi e i sistemi sono soprattutto aperti. In altre parole, è impossibile fidarsi completamente di altri agenti o fare forti assunzioni a proposito dello stato interno degli agenti e sul loro modo di ragionare.

L'**approccio sociale** considera le conseguenze sociali di eseguire un atto comunicativo. *Il fatto che ci venga comunicato la data dell'appello, ad una certa data, ha delle conseguenze.* È come se il professore facesse un impegno verso una certa affermazione. La comunicazione assume più i toni dell'**impegno**, ma questo non è una comunicazione che ha degli effetti sociali. Come se venisse stabilito un contratto (agreement) tra chi comunica e chi ascolta la comunicazione. Il fatto che ci venga comunicato qualcosa con la precondizione che creda (il prof) qualcosa è un'azione che non potrà mai essere verificata (c'è un modo ed è quello di entrare nella testa del professore). Se invece, la comunicazione ha degli effetti sociali, i contratti sono delle cose verificabili. I contratti come tutte le norme possono essere fatte solo su eventi osservabili. **Non ha senso una norma che ti impone come pensare. Questa è dittatura.**

La semantica sociale della comunicazione dice che invece di basare la semantica dell'atto comunicativo su quello che sono degli elementi verificabili come lo stato mentale di qualcuno, si basa solo su cose osservabili e quindi si propone di introdurre un concetto di comunicazione che possa modificare no lo stato mentale, ma una realtà sociale creando/modificando dei contratti.

Le caratteristiche di un <i>buon ACL</i> :	... <i>is not good</i> :
<ul style="list-style-type: none"> • Formal: chiarezza della specifica per guidare al meglio colui che realizza un sistema • Declarative: descrivere cosa (<i>what</i>) piuttosto che come (<i>how</i>) • 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 	<ul style="list-style-type: none"> • English is not good: non è formale • FSM non è good: non dichiarativo non significativo • mentalistic approach is not good: non verificabile • Temporal logics are not good: non significativo (se applicato ai token direttamente)

Figure 3.8: "Un buon ACL" (secondo M. P. Singh)

3.4.10 Semantica per ACL basata sugli stati mentali e verifica

- **rispettare un protocollo**, un protocollo è rispettato se e solo se l'interazione specifica/desiderata occorre durante l'esecuzione
- la semantica basata sugli stati mentali per un ACL è particolarmente adatta per ragionare in modo "statico" sulle proprietà del protocollo e sulle politiche adottate dagli agenti
- MA da un punto di vista di un agente che partecipa all'interazione, come essere sicuri che gli altri agenti rispettano la semantica specificata?
- Non è possibile fare introspezione sugli agenti, solitamente lo stato mentale degli agenti non è accessibile: come possiamo dimostrare che un agente crede che ciò che dice se non è un agente BDI?

- **Verificare un protocollo:**
necessita di una semantica basata su fatti "osservabili"
 - Una semantica che sia indipendente dagli stati mentali dei partecipanti
 - Una semantica che sia verificabile sia da un partecipante sia da una terza parte che deve monitorare l'interazione
 - Una semantica che sia adatta per *open MAS*
 - Una semantica che renda osservabile la violazione di una specifica da parte di un partecipante all'interazione, senza che si debba prendere in esame la natura dell'agente o eseguire introspezione dello stato mentale dell'agente

Figure 3.9:

3.4.11 Semantica Sociale

- L'approccio è basato sui practical **commitments** (impegni) tra agenti: un agente (il debtor) si impegna verso un altro agente (il creditor) a rendere vero un qualche fatto o ad eseguire una qualche azione. *L'appello sarà in X data*. Se la semantica è quella dell'impegno, allora è verificabile. Basta andare a vedere se la data occorre oppure no. Se non dovesse occorrere, il contratto è stato violato. È tutto **osservabile**
- Secondo l'approccio sociale, i vari atti illocutori possono essere visti in termini di commitment sociali che i partecipanti creano (questo è intuitivo per atti comunicativi come promise, meno per altri tipi di comunicazioni). I vari dialoghi, le varie performatives, creano/modificano un insieme di contratti/impegni, dove questi possono essere solo su fatti osservabili, altrimenti non sarebbero impegni. Non importa a quanto che viene creduto. Questo è influente.

La **comunicazione** nell'approccio sociale è **inherentemente pubblica**. Vuol dire che ogni mia comunicazione non è privata tra stati mentali ma comunicazione che crea impegni/contratti pubblici, se così non fosse non avrebbe valore. Senza entrare troppo nel dettaglio, Singh, va a prendere i vari atti illocutori e assegna una semantica basata su azioni e impegni di queste azioni.

Illocation	Objective	Subjective	Practical
$\text{inform}(x, y, p)$	$C(x, y, p)$	$C(x, y, xBp)$	$C(x, G, C(x, y, p) \rightarrow p)$
$\text{request}(x, y, p)$	—	—	$C(x, G, \text{request}(x, y, p) \rightarrow \text{AFC}(y, x, p))$
$\text{promise}(x, y, p)$	$C(x, y, \text{RF}p)$	$C(x, y, xl_p)$	$C(x, G, C(x, y, \text{RF}p) \rightarrow \text{RF}p)$
$\text{permit}(x, y, p)$	$C(x, y, \text{EF}p)$	$C(x, y, \neg xl_p)$	$C(x, G, \text{permit}(x, y, p) \rightarrow \neg C(y, G, \neg \text{RF}p))$
$\text{forbid}(x, y, p)$	$C(x, y, \neg \text{RF}p)$	$C(x, y, \neg xl_p)$	$C(x, G, \text{forbid}(x, y, p) \rightarrow C(y, G, \neg \text{RF}p))$
$\text{declare}(x, y, p)$	$C(x, y, p)$	$C(x, y, xl_p)$	$C(x, G, \text{declare}(x, y, p) \rightarrow p)$

Table 1: Social semantics formalized

3.4.12 Commitments

- Siamo nel 1990, Cohen e Levesque definiscono le intenzioni come **goal persistenti**, dove le intenzioni sono alla base dei *commitments*.
- Nel 1997 dicono che le intenzioni condivise sono la base per un commitment sociale. Quando prendiamo un impegno e lo comunichiamo, questo viene rappresentato come un commitments sociale
- Castelfranchi (ricercatore al CNR di Roma) dice di essere in accordo con i commitment individuale, ma in disaccordo per quanto riguarda i commitment sociali.

Castelfranchi dice che le caratteristiche fondamentali per definire i commitment sociali, sono di natura distribuita (coinvolgono più elementi) e normativa (devono coinvolgere più di uno e deve essere basata su elementi osservabili/tangibili).

- un commitment sociale ha natura normativa e genera diritti;
- a differenza di quello che dicevano Cohen e Levesque, cancellare o ritirare un commitment sociale non implica una informazione verso la controparte;
- cancellare o ritirare un commitment viola un'obbligazione, frustra delle aspettative e dei diritti che l'agente ha creato

3.4.13 Singh's Commitments

- 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 relazione legali e sociali (come gli impegni, le richieste, i diritti, i privilegi, i poteri) in un unico framework. I **metacommitments** sono dei commitments che coinvolgono commitments. Così come succede in FIPA, Singh con i commit riesce a costruire un sacco di altri concetti deontici come quelli in parentesi.
- il comportamento degli agenti è influenzato dai commitment perchè l'assunzione è che gli agenti rispettino gli impegni presi
- i commitment sono manipolabili dagli agenti

Singh introduce i commitments come una **quadrupla** - C(debotr, creditor, antecedent, consequence)

- il debtor è socialmente legato² al creditor a rendere vera la condizione conseguente se la condizione antecedente è vera

Esempio, il commitment

C(Bookie, Alice, 12\$, BeatingtheOdds)

Significa che Bookie si impegna nei confronti di Alice che se ella paga \$12, allora Bookie le spedirà il libro Beating the Odds

- **Detach**, se Alice effettua il pagamento, allora il commitment è detached C(Bookie, Alice, T, BeatingtheOdds)
- **Discharge**, se bookie spedice il libro, il commitment è discharged

3.4.14 Il ciclo di vita dei commitment secondo Singh

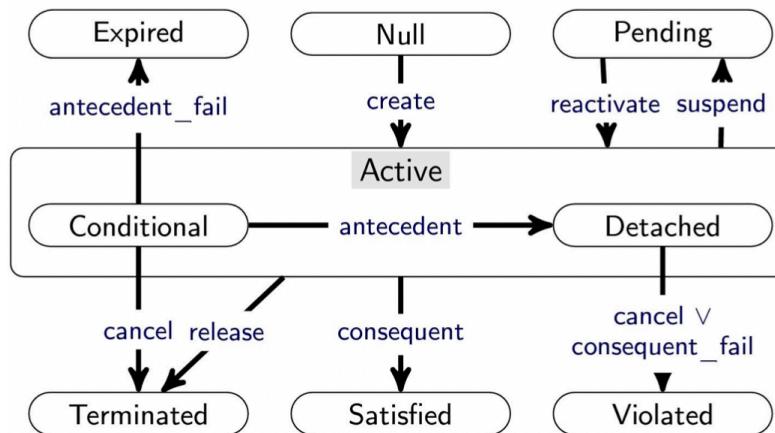


Figure 3.10: Ciclo di vita del commitment di Singh

Singh, definisce un ciclo di vita dei commitment.

1. il commitment viene **creato** (create) e subito dopo **attivato** (active)
2. una volta attivata può essere sia **condizionale** che **detached**. Quando l'antecedente diventa **vero** allora diventa un "obbligo". Infatti fintanto che il commit rimane *conditional* e viene cancellato, semplicemente termina. *Se il panettiere leva l'insegna il giorno dopo sti cazzo, non ha violato niente*. Se invece fosse **detached**, e al panettiere abbiamo pagato il pane e lui leva il prezzo eh no questo non va bene pezzo di merda. Quindi, se il debitore cancella il commitment, si viola e questa violazione è causata dal fatto che il conseguente fallisce (non mi è stato dato il pane).
3. se invece il commitment è **release**, il creditore dice pazienza, quindi che sia condizionale o detached non importa e viene terminato. *Abbiamo dato i soldi e me ne vado, non c'è nessuna violazione*.
4. se è **conditional** o **detached** e si soddisfa il conseguente, il commitment anche se non era stato attivato, è **soddisfatto**.
5. Mentre il commitment è in **active**, può essere **suspend** e va in **Pending**. Se invece l'antecedente diventa falso quindi impossibile trasformazione in detached allora **expired**, non potrà essere più un obbligo o meglio un detached.

²vuol dire che in qualche modo c'è un contratto pubblico

3.4.15 Operazioni sui commitment (Singh, 1999)

- $\text{create}(x, y, r, u)$ è eseguita da x e causa $C(x, y, r, u)$. Operazione creata ed eseguita sempre da chi s'impegna e causa la creazione del commitment;
 - x è il debitore;
 - y è il creditore;
 - r è l'antecedente;
 - u è il conseguente;
- $\text{cancel}(x, y, r, u)$ è eseguita da x e rimuove $C(x, y, r, u)$. Eseguita dal debitore;
- $\text{release}(x, y, r, u)$ è eseguita da y e rimuove $C(x, y, r, u)$. Eseguita dal creditore e rimuove il commitment C ;
- $\text{delegate}(x, y, z, r, u)$ è eseguita da x e causa $C(z, y, r, u)$. Eseguita dal debitore e delega l'impegno a qualcun altro
- $\text{assign}(x, y, z, r, u)$ è eseguita da y e causa $C(x, z, r, u)$. Passa l'impegno del creditore a qualcun altro

3.4.16 Postulati sui commitment (Singh, 2008)

Nel 2008 definisce anche dei **postulati** che devono valere sui commitment;

- **Discharge:** $u \rightarrow \neg C(r, u)$. se è vero il conseguente, il commitment non è più vero. *Mi impegno a far sorgere il sole, ma il sole è già sorto;*
- **Detach:** $C(r \wedge s, u) \wedge r \rightarrow C(s, u)$. Se l'antecedente è una congiunzione possiamo effettuare un progresso, se faccio diventare vero un pezzo mi manca solo un altro pezzo da far diventare vero;
- **Augment:** da " $C(r, u) \wedge$ " infers s " deriviamo $C(s, u)$. Se m'impegno su una condizione complesso e come se m'impegnassi anche sulle sottocondizioni della condizione complessa;
- **L-disjoin** $C(r, u) \wedge C(s, u) \rightarrow C(r \vee s, u)$. Se m'impegno su r ed s allora è come se m'impegnassi sull' \vee dei due perchè basta uno dei due per far diventare vero l'impegno
- **R-conjoin** $C(r, u) \wedge C(r, v) \rightarrow C(r, u \wedge v)$. Se invece lo stesso antecedente mi impegna su due diversi commitment e come se mi fossi impegnato sulla congiunzione;
- **Consistency** $\neg C(r, \text{false})$. Non possiamo impegnarci su cose false, non le renderei mai vere;
- **Nonvacuity**, da "r infers u" deriviamo $\neg C(r, u)$. Se l'antecedente deriva il conseguente allora non è un impegno. Non ha senso perchè già rendere vero il commitment lo scarica completamente;
- **Weaken** $C(r, u \wedge v) \wedge \neg u \rightarrow C(r, u)$. Anche l'impegni possono essere raggiunti progressivamente. **Questo è da rivedere perchè è da controllare... NON L'HA MAI FATTO**

3.5 Protocolli di interazione

Prima di addentrarci in questo capitolo, si ricordano le seguenti definizioni, potrebbero essere utili:

1. **protocolli di comunicazione**, servono per definire la sintassi degli atti comunicativi
2. **protocolli d'interazione**, stabiliscono le sequenze legali, ovvero stabiliscono come costruire un dialogo.

Per quanto concerne i protocolli d'interazione.

- gli agenti non possono prendere parte a un dialogo semplicemente scambiandosi dei messaggi ACL
- l'analisi di numerose **conversazioni** umane mostra che sono presenti degli schemi (pattern)
- ...
- in generale, i protocolli specificano il contenuto della comunicazione, non solo la forma. Punto fondamentale. I protocolli dicono anche quale siano i messaggi da includere all'interno di una comunicazione, non solo la sequenza. **L'alzata di mano in un'asta specifica anche il messaggio.**
- non mi ricorderò mai tutti i punti quindi sti cazzo e spero Dio che non me li chieda

Sono una serie di standard di dialogo che permettono di raggiungere un certo stato di cose d'interesse per entrambi. Sono interessanti perché permettono, se studiati una volta (anche se complessi) di semplificare notevolmente il ragionamento dell'agente. *ES. In un asta noi conosciamo quale sia il protocollo di interazione, ne sappiamo le regole, lo studiamo ed è stato verificato che funziona alla perfezione. Non c'è bisogno che pianifichi le azioni per partecipare all'asta.* Se quello è uno standard noi possiamo utilizzare questo blocco come se fosse una procedura da mettere all'interno del piano. Abbiamo l'obiettivo di prendere quell'oggetto e quindi partecipo all'asta. Non dobbiamo pianificare come agire durante un'asta. Questi si chiamano **protocolli d'interazioni**, conosciuti anche come politiche di conversazioni e l'ACL questi li include negli standard.

3.5.1 Specifica di protocollî

Un modo semplice per specificare protocolli è l'automa a stati finiti, dove se abbiamo piu di una interazione tra agenti, sugli archi sono etichettati gli agenti che effettuano la comunicazione. Altri esempi sono le reti di Petri, i DCG (Definite Clause Grammars) e l'AUML un

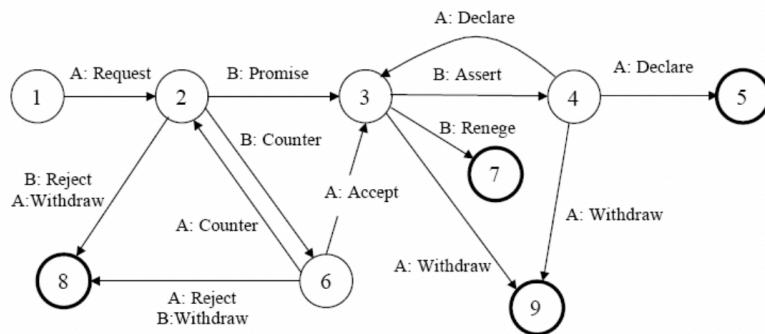


Figure 3.11: Automa a stati finiti per la rappresentazione di un esempio di protocolli di interazione

derivato di UML. Quest'ultimo in particolare viene usato da per definire le specifiche FIPA che permettono di definire quelli che sono i protocolli. Entrambi comunque sono modi per specificare protocolli.

FIPA Request Interaction Protocol

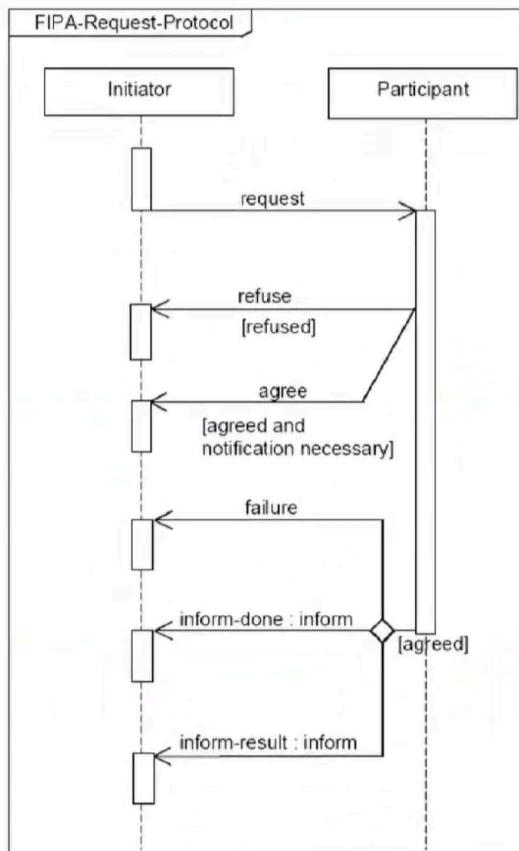


Figure 3.12: FIPA Request Interaction Protocol

3.5.2 Contract Net protocol (task sharing)

Il **Contract Net Protocol** è un esempio famoso inserito dentro FIPA. Permette di realizzare sistemi di agenti **cooperanti**, permette di effettuare quello che si chiama **task-sharing**. Utilizzato da alcuni sistemi P2P. Viene modellato sul meccanismo di contrattazione per governare gli scambi di merce e servizi.

- un agente che desidera che un certo *task* sia risolto viene chiamato **manager** (o *initiator*);
- gli agenti che potrebbero risolvere il task si chiamano **contractors** (o *participants*);

Un agente ha un goal e realizza che:

- non ha le capacità di realizzarlo in autonomia;
- preferirebbe non soddisfarlo in autonomia, per ottenere una soluzione migliore secondo qualche criterio

Due sono gli elementi fondamentali che si distinguono all'interno di un Contract Net Protocol.

- **il Manager**

- colui che annuncia il task che deve essere risolto (insieme ai vincoli e la deadline - *announcement*)
- riceve e valuta le offerte dei possibili contractor
- sceglie un contractor e lo informa (*award*)
- riceve il risultato

- **il Contractor**

- riceve gli annunci di task dal manager
- valuta il task (proprie capacità, vincoli e prezzo)
- risponde declinando o facendo un'offerta (*bidding*)
- esegue il task se l'offerta è stata accettata
- riporta il risultato (*expediting*)
- un contractor per uno specifico 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 contractor

Funzionamento

I manager potrebbero essere anche più e sono coloro che annunciano i task. Il potenziale contractor nel ricevere, valuta quello che è d'interesse per lui. Una volta fatta la valutazione fa il bidding, *mi offro di farlo esprimendo un costo*. Più di uno puo fare bidding questo perchè l'annuncio è rivolto a molti. Il manager sceglie uno qualunque (azione di award) e a questo punto il contract è stabilito ed esiste l'**aspettativa** che venga effettuato quel lavoro.

- *cfp* call for proposal è l'annuncio in FIPA che viene mandaato potenzialmente a *m* participants
- entro una certa deadline un sottoinsieme di *m* manda una *refuse* (rifiuta) o *proposed*, non è obbligatoria la risposta tra i partecipanti
- per quelli che hanno rifiutato, l'esecuzione termina. Per gli altri viene valutato un sottoinsieme di questi d'accettare e altri vengono rifiutati e per quest'ultimi l'interazione viene terminata
- per quelli accettati ci si aspetta che venga inviato un risultato oppure presentarsi per qualche ragione il **fallimento**

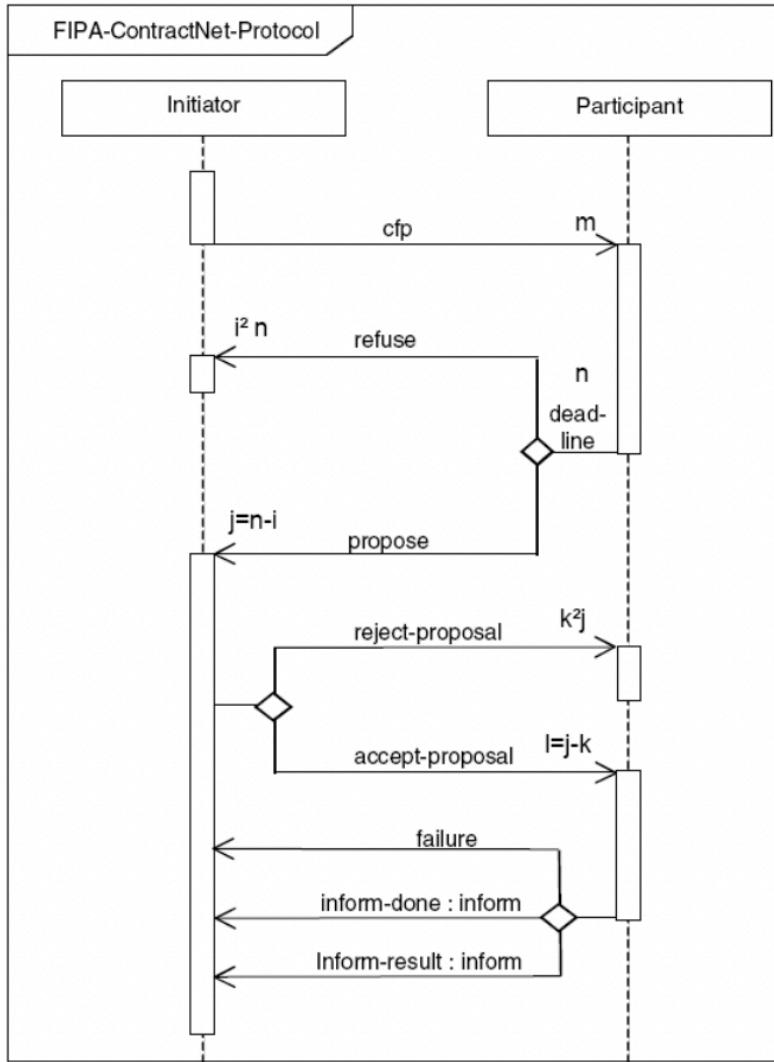


Figure 3.13: FIPA Contract Net Protocol

3.6 Protocolli a commitment

3.6.1 Protocolli di interazione con commitment

M. P. Singh, propone oltre alla definizione di commitment anche dei protocolli basati sulle operazioni per la creazione e la modifica di commitment. Invece di lavorare sui protocolli UML o macchine a stati finiti, propone protocolli che sono costruiti da un **insieme di commitment**. Gli agenti giocano ruoli nella società e i ruoli definiscono l'associazione con impegni sociali. A seconda del ruolo prendiamo un impegno oppure no. *Chi si iscrive ad un esame prende l'impegno a presentarsi all'esame.* In generale gli agenti possono gestire i propri impegni (commitment) manipolandoli o cancellandoli. Poiché i requisiti di un protocollo vengono espressi solamente attraverso commitment, gli agenti possono essere conformi sulla base della loro comunicazione (non è necessaria conoscere la loro implementazione)

Yolum e Singh presentano i protocolli nel 2001. Come insieme di azioni il cui significato espresso in termini di effetti sullo stato sociale sono d'accordo con tutti gli agenti interagenti. *Alzare la mano crea un impegno del tipo che se il banditore mi assegna io che ho alzato la mano mi impegno a pagare l'oggetto. Il banditore s'impegnerà a dare l'oggetto a chi ha vinto l'asta.* Gli effetti dell'azione alzare la mano sono espressi in termini di operazioni su commitment, nel nostro caso alzare la mano significa creare un impegno. L'idea è di

catturare la relazione "count-as" alzare la mano conta come impegnarsi, e questo count-as descrivere il significato istituzionale dell'azione. **Il solo vincolo che il protocollo a commitment deve soddisfare è che al termine dell'esecuzione tutti i commitment devono essere scartati senza che ci sono commitment pendenti.**

3.6.2 Un esempio: The NetBill Protocol

Serve per l'acquisto di merci su internet. Questo protocollo tra cliente e venditore ha una sequenza rigida, ma ci sono anche delle implementazioni diverse. La *quote* ad esempio potrebbe essere eseguita senza che ci sia una *request* (pubblicità, newsletter, oppure l'oggetto arriva a casa e se lo vogliamo dobbiamo pagare. La *send* della merce viene invitata insieme alla quotazione e prima dell'accettazione del nostro rifiuto). L'obiettivo di Singh e Yolum è quella di dare **libertà assoluta** agli agenti (proprio perchè sono autonomi). L'unica cosa che guida le azioni sono i **commitment**. Se facciamo una certa azione che crea un impegno, per avere successo quella deve essere scartata. In questo momo Yolum e Singh dicono che i protocolli sono più **flessibili** e con un'unica descrizione si riesce ad accogliere più varianti. Questo perchè, quando definiamo un processo sequenziale (come in 3.14) noi non diamo solo una condizione di successo finale (protocollo completato) ma diamo anche un vincolo di come deve essere completato. I protocolli se eseguiti secondo un certo ordine garantiscono un successo finale che è naturalmente il completamento. Le condizioni di completamento aggiuntive sono rappresentate dalle sequenze. Se volessimo dire che il protocollo va bene anch per altri ordine dovrebbe dare anche tutte le possibili alternative (con un sequence diagram è un casino e non agevole). Allora, per ovviare a questo problema i **commitment protocol** hanno permesso di semplificare questa descrizione.

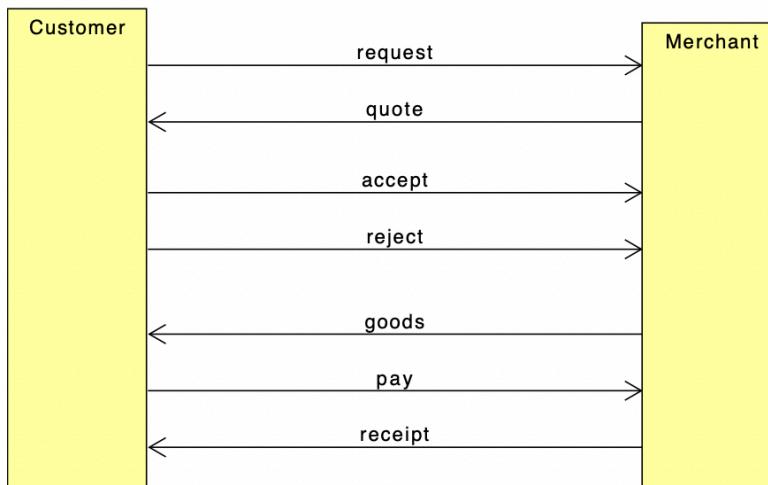


Figure 3.14: The NetBill Protocol

Piccolo riepilogo prima di proseguire

Abbiamo detto che Singh introduce i commitment come quadrupla e li introduce per dare **semantica alle azioni comunicative**. Anzichè utilizzare *belief*, *desire*, *intention* utilizza i *commitment*, quindi **impegni sociali**. Poi quello che fa è provare ad utilizzarli non solo come **semantica**, ma anche come **strumento** per definire **protocolli**. Quindi, come strumento direttamente **manipolabile** dagli agenti. In un primo momento, definisce un protocollo a commitment come un insieme di commitment, dove le operazioni base (*cancellazione*, *release*, ...) sono le **azioni** che gli agenti fanno per manipolare i commitment. Con Yolum, in un secondo momento, trova un'altra strada per rendere tutto più efficace. Non solo usano le operazioni base, ma le azioni che compiono gli agenti hanno conseguenze in termini di

modifica dei commitment. Quindi, *alzare la mano significa creare un certo commitment*. In un'asta l'alzata di mano implica che se vinco, sono impegnato a pagare la cifra. Il banditore quando dice "offerta più alta", rilascia un commitment dell'ultimo che ha vinto. Si nota, come le azioni **causano** delle operazioni sui commitment stessi, come se fossero delle macro-operazioni sui commitment. L'obiettivo è quello di permettere ai protocolli la massima **flessibilità** non vincolata da una sequenza. Massima libertà quindi nell'agire da parte dell'agente, quindi a questo punto massima libertà diventa un **vantaggio quanto proprio l'unico obiettivo che si vuole dare è dire il protocollo è completato quando non c'è più nessun impegno da parte di nessuno**.

NON IMPORTA LA STRADA, MA IMPORTA CHE TUTTI GLI IMPEGNI SIANO STATI RISOLTI

Il NetBin Protocol è interessante perché ha delle applicazioni diverse rispetto a quella che è la semplice sequenza. Non considera la sola sequenza data in origine (ordine sequenziale), ma anche delle altre.

Nota finale. Se considerassimo il NetBill come un *sequence diagram*, si capisce che dovremmo fare tante alternative, non possiamo avere un'unica strada. Queste alternative devono essere rappresentate andando ad aggiungere ad esempio dei paralleli e questo potrebbe essere davvero complicato.

Il punto è che qualunque sia l'ordine il protocollo dovrà essere SEMPRE il seguente:

- tutti devono essere soddisfatti alla fine;

Soddisfatti nel senso che hanno assolto ai propri impegni e l'interazione in qualche modo è arrivata a conclusione in modo (scusate il gioco di parole) soddisfacente. Obiettivo di Yolum e Singh nell'usare i protocolli a commitment.

- un protocollo può essere eseguito in modo diversi (vedremo come l'esecuzione di un **run** può essere fatta in vari modi);
- tutto però devono trovare la soddisfazione in termini di impegni. Questa è intesa come tutti gli impegni sono soddisfatti e non ci sono impegni pendenti.

3.6.3 The NetBill Protocol con i commitment

- **request**, non crea e non modifica nessun impegno;
- **quote** $\text{create}(C(m, c, \text{accept}, \text{goods}))$, quando viene fatta una quotazione questa viene vista come impegno del mercante verso il customer che se accetta allora spedirà la merce;
- **accept** $\text{create}(C(c, m, \text{goods}, \text{pay}))$, viene creato un impegno da parte del customer verso il mercante che pagherà solo se il mercante manderà la merce;
- **reject** $\text{create}(C(m, c, \text{accept}, \text{goods}))$, rifiuta la quotazione allora rilascia l'impegno, ovvero rilascia il commitment *quote*. Non interessa al customer la quotazione proposta;
- **goods** $\text{create}(C(m, c, \text{pay}, \text{receipt}))$, avviene l'invio della merce. Il mercante si impegna a creare un impegno verso il customer che se paga allora manderà la ricevuta;
- **pay** e **receipt**, non fanno nulla, ovvero non creano commitment, ma poiché queste azioni sono coinvolte in **goods**, fanno progredire il commitment.

Contract Net Protocol
• cfp means $\text{create}(C(i, p, \text{propose}, \text{accept} \vee \text{reject}))$
• propose means $\text{create}(C(p, i, \text{accept}, \text{done} \vee \text{failure}))$
• refuse means $\text{release}(C(i, p, \text{propose}, \text{accept} \vee \text{reject}))$
• accept means none
• reject means $\text{release}(C(p, i, \text{accept}, \text{done} \vee \text{failure}))$
• done means none
• failure means none

Figure 3.15: The Contract Net Protocol

3.6.4 Un altro esempio: The Contract Net Protocol

- **cfp**, l'iniziatore prende un impegno condizionale verso il partecipante che se gli fa una proposta (il partecipante verso l'iniziatore) lui accetterà o rifiuterà
- crea impegno del partecipante verso l'iniziatore che se accetta manderà una comunicazione che ha fatto il task assegnato oppure dirà di aver fallito;
- rilascia l'impegno dell'iniziatore verso il partecipante (freccia a primo punto)
- non ha nessuna modifica diretta (creazione, rilascio, cancellazione) ma farà progredire il secondo punto facendo attivare
- **reject**, rilascia il secondo punto
- **done** e **failure** faranno progredire precedenti commitment

Prima di andare a vedere nel dettaglio gli esempi, dobbiamo iniziare a chiederci cosa sia un run.

3.6.5 Run del protocollo a commitment

Questo **run** non è nient'altro che un'esecuzione del protocollo

- è una sequenza finita di azioni che portano in uno stato in cui tutti i commitment sono **fulfilled** (ovvero sono **Discharge**), ossia sono stati **soddisfatti**. L'insieme dello stato sociale (quindi l'insieme dei commitment) è vuoto. Non ci sono commitment pendenti, ma il commitment ha la seguente forma $C(x, y, , p)$. Questo è detached dove p vero l'antecedente. *Il panettiere ci dice che se gli diamo i soldi ci darà il pane, ma se nessuno da i soldi non è che il protocollo non funziona, semplicemente nessuno ha interagito con lui.*
- Data la precedente specifica delle azioni, posso costruire delle sequenze che permettono di raggiungere uno stato in cui non ci siano commitment pendenti. Essi sono i run del protocollo. Quindi questo significa quando arrivano in uno stato in cui non ci sono più commitment pendenti;
- possiamo sempre **verificare** che data una sequenza di azioni eseguite dagli agenti, posso verificare se essa sia o non sia una sequenza ammessa dal protocollo (un run del protocollo)

La cosa interessante che la verifica avviene in base alle azioni compiute e quindi anche da questo punto di vista solo elementi osservabili. Non andiamo a vedere se è conforme se quello pensava giusto e io penso giusto, ma se questa sequenza è corretta e porta a scartare tutti. Lo stato sociale è nella mente di ognuno. *Se l'alzata di mano dell'asta comporta un gesto del banditore, noi sappiamo che c'è un impegno perché abbiamo condiviso il significato di quell'azione in termini di impegni. Entrando nell'asta abbiamo accettato un protocollo.*

3.7 Modelli di coordinazione

Abbiamo visto i protocolli per coordinare attività, ma esistono altri metodi per coordinare, quindi non solo protocolli. Un sistema molto utilizzato sono quelli noti come **blackboard system**, astrazione di una shared memory. L'idea è quella di sincronizzarsi per scrivere sulla lavagna prendendo un gessetto. Quest'ultimo viene preso e quando lo poso l'altro lo può prendere I sistemi a blackboard hanno uno spazio condiviso dove si fanno delle operazioni di *in* e *out* e tra queste operazioni ci si può mettere in attesa. Sono stati sviluppati anche per JAVA dei sistemi di coordinazione a memoria condivisa più ad alto livello. La cosa interessante di questi sistemi è che ad esempio hanno portato ad avere delle **lavagne programmabili**, quindi programmare le lavagne, una volta che si fa qualcosa, a **reagire** cambiando il suo stato. Considerando gli agenti:

- la lavagna è l'ambiente
- gli agenti si coordinano tramite ambiente

I sistemi a lavagna sono stati sviluppati in sistemi di agenti che si coordinano tramite **ambiente**. È quello che normalmente avviene. Un sistema a blackboard molto evoluto che rappresenta ambiente è stato sviluppato a Bologna e prende il nome di **Kartago**. Sistema integrato con altri sistemi come JASON offrendo ambiente e agenti che interagiscono per coordinarsi.

3.7.1 Le norme

Altro modo per coordinarsi sono le **norme**. C'è un area di ricerca nei sistemi multiagenti che si dedica ai normative multi agent system. Vedere come rappresentare, esplicitare norme e leggi che permettono di coordinarsi. Anche qui fatti diversi esperimenti, il MOISE crea un'organizzazione di agenti che hanno degli obiettivi. Vengono messi degli obblighi dove gli agenti s'impegnano a soddisfarli. JASON, KARTAGO e MOISE hanno fatto nascere JAKAMO. Creare agenti programmabili in jason che agiscono sull'ambiente kartago che vengono regolati nei loro comportamenti da delle norme specificate dall'organizzazione tramite Moose.

Chapter 4

Robustezza e ABS

Con **robustezza** intendiamo il grado in cui un sistema o un componente può funzionare correttamente in presenza di input non validi o condizioni ambientali stressanti. Secondo Alderson e Doyle, una proprietà di un sistema è robusta se è invariante rispetto a un insieme di perturbazioni.

- affidabilità, come robustezza al guasto dei componenti;
- efficienza, come robustezza alla mancanza di risorse;
- scalabilità, come robustezza per modificare le dimensioni e la complessità del sistema nel suo insieme;
- modularità, come robustezza riarrangiamento strutturato delle componenti;
- evolvibilità, come robustezza dei lignaggi ai cambiamenti su scale temporali lunghe

Piccola parentesi

Immaginate in un sistema elementare come Java la robustezza potrebbe essere interpretata in termini di robustezza a certi eventi. Supponiamo di poter determinare l'evento, questo vuol dire che il sistema in caso di quell'evento sa assorbire l'impatto dell'evento stesso. Evento, dove avviene una divisione per 0, viene lanciata un'eccezione e io programmatore dove vado ad eseguire delle divisione, che potrebbe essere divise per 0, vado ad introdurre catch che permette di ovviare a quell'errore. Ad esempio potremmo richiedere all'utente un input

Fine parentesi

Se pensiamo a questo sistema facendo riferimento ad un sistema multi-thread le cose si complicano. Supponiamo di avere una parte che faccia l'input, dove l'invio di questo input viene mandato ad un altro thread. Ora, supponiamo che quest'ultimo thread lanci un'eccezione, questa viaggerà nel suo stack, ma non tornerà mai al primo thread. La robustezza dei sistemi distribuiti è molto complessa. Alderson e Doyle individuano questo come legame in cui c'è un **feedback**

- quando assegniamo un compito, pretendo che un feedback ritorni indietro, ovvero una risposta che sia andato tutto bene oppure no.

Questo è importante, perché in uno sistema distribuito i compiti di questo sistema vengono distribuiti su più elementi, ma questo non significa che ognuno di loro lavora per "un boss", questi devono collaborare per compiere un lavoro. I vari elementi che svolgono compiti sono autonomi e ognuno di essi è dotato di propria vita. L'obiettivo è capire come far viaggiare le informazioni.

- ci vuole un particolare accordo tra gli elementi.

Il nostro obiettivo è avere dei legami tra elementi che interagiscono tra loro.

Se c'è un problema (incendio) è importante che tutti gli elementi siano collegati tra loro in un modo che ci ha la competenza possa intervenire per riparare la situazione sbagliata. I legami devono essere chiari, quindi chiaro che ci deve essere qualcuno che aspetta un feedback da un altro ancora in modo che possa intervenire.

- chi deve essere informato se le cose non vanno bene?

*Se si dovesse rompere il computer in aula, il prof che entrerà alla seconda ora se lo ritroverà appunto rotto. Se non ci fosse una comune responsabilità che permetta in questo al prof di comunicare a qualcuno che il computer è rotto, è un problema. Il professore ha la responsabilità/dare il **feedback** verso qualcuno in certe situazioni. Le situazioni, a differenza delle eccezioni che viaggiano su uno stack, viaggiano su delle relazioni tra persone che si sentono co-responsabili verso il raggiungimento di un certo obiettivo.*

Questo elemento è la **accountability**. Questa è una relazione che lega due persone, dove una ha il diritto di chiedere se tutte le cose andavano bene, quindi ci viene chiesto *è andato tutto bene?* e dall'altra il dovere dell'altra persona di informare di certe situazioni. Questa relazione non può che non essere introdotta da una relazione **normata**¹, cioè un permesso e un obbligo nel fare qualcosa.

Nello studio che ha condotto il Professore Baldoni con il suo gruppo di ricerca, si creano delle catene/strutture informative con l'accountability.

- il professore che ha rotto il pc segnala al responsabile del dipartimento l'evento
- il responsabile lo segnala all'ufficio tecnico dell'università,
- l'ufficio tecnico ha il trattamento che lo ripara

Se quest'ultimo (l'ufficio tecnico) ha bisogno di qualche informazione non la chiede al primo della lista, ma al suo predecessore (al responsabile) e così via.

4.1 Agent Based Modelling (ABS)

4.1.1 Complex Adaptive Systems

Esiste da parecchi anni, ancora prima che il concetto di software agenti prendesse piede. È un metodo di indagine molto utilizzato.

- la prima cosa che ci si chiede è, com'è fatto un sistema adattativo complesso?

È un sistema (Complex Adaptive Systems) che ha:

- delle unità **interagenti**, che possono essere componenti, elementi primitivi, costituenti;
- presenta delle proprietà emergenti, cioè proprietà derivanti dalle interazioni delle unità che non sono delle singole unità stesse

Alcuni esempi:

1. un sistema immunitario
2. organismi multi-cellulare
3. colonie di insetti

¹permessi e obblighi come coordinazione quando abbiamo presentato Jakamo

4. economie di mercato decentralizzate

Un "Complex Adaptive Systems" è un sistema complesso che contiene almeno qualche unità tali che:

- sono **reactive**, cioè reagiscono sistematicamente alle diverse condizioni ambientali;
- sono **goal-directed**, cioè le reazioni sono guidate al raggiungimento di obiettivi
- sono in grado di **pianificare**, cioè sono goal-directed e sono finalizzate a modificare l'ambiente in modo da facilitare il successivo raggiungimento dei propri obiettivi

Un esempio tipico di Complex Adaptive Systems sono le scienze sociali. Queste cercano di

- capire non solo come si comportano gli individui, ma anche come l'interazione di molti individui porta a risultati su larga scala
- Comprendere un sistema politico o economico richiede più che la comprensione degli individui che compongono il sistema. Richiede anche la comprensione di come gli individui interagiscono tra loro e di come i risultati possono essere più della somma delle parti
- Sono un complex adaptive system

Questi sistemi si focalizzano su quella che è la comprensione dell'individuo. Il panico è un esempio classico. Una persona potrebbe non reagire, ma interagendo con altri, reagisce. Le situazioni di panico in luogo affollati sono complex adaptive system

4.1.2 Agent-based Modeling

- L'agent-based modeling è adatto allo studio delle scienze sociali
- L'agent-based modeling è infatti un metodo per studiare i sistemi che presentano le seguenti due proprietà:
 - a. il sistema è composto da agenti interagenti
 - b. il sistema mostra proprietà emergenti, cioè proprietà derivanti dalle interazioni degli agenti che non possono essere dedotte semplicemente aggregando le proprietà degli agenti
- Quando l'interazione degli agenti è contingente all'esperienza passata, e specialmente quando gli agenti si adattano continuamente a quell'esperienza, l'analisi matematica è tipicamente molto limitata nella sua capacità di derivare le conseguenze dinamiche ²

Allora in virtù dell'ultimo punto, gli agent based modelling sono stati molto utili.

- l'agent-based modeling parte da ipotesi sul comportamento di agenti e delle interazioni tra agenti e utilizza la simulazione per generare "storei" che possono rivelare le conseguenze dinamiche delle ipotesi fatte sugli agenti e sulle loro interazioni)
- l'agent-based modeling permette di studiare come gli effetti su larga scala derivino dalle interazioni tra molti agenti
- gli agenti possono rappresentare consumatori, venditori, elettori, gruppi sociale come famiglie aziende comunità.

²qui c'entra l'esperienza passata, se la mia reazione dipende anche dalla mia storia (non solo come individuo ma storia delle interazioni) questo è difficile da studiare matematicamente. L'analisi matematica lo tratta male.

La simulazione

- La simulazione in generale e l'agent-based modeling sono un terzo modo di fare indagine, oltre alla deduzione e all'induzione
- La deduzione serve per derivare teoremi da ipotesi, l'induzione per trovare modelli nei dati empirici
- La simulazione, come la deduzione, inizia con un insieme rigorosamente specificato di assunzioni riguardanti un sistema di interesse
- **ma**, a differenza della deduzione, la simulazione **non dimostra** teoremi con generalità
- La simulazione genera dati adatti all'analisi per induzione
- **ma**, a differenza dell'induzione tipica, i dati simulati provengono da esperimenti controllati piuttosto che da misurazioni dirette del mondo reale

Di conseguenza:

- la simulazione differisce dalla deduzione e dall'induzione standard sia nella sua implementazione che nei suoi obiettivi
- La simulazione consente una maggiore comprensione dei sistemi attraverso esperimenti computazionali controllati

Chapter 5

FIPA

FIBA (The Foundation for Intelligent Physical Agents) aveva come obiettivo la standardizzazione di comunicazione, piattaforma e strumenti che permetessero di costruire software su una struttura condivisa e che ha rappresentato uno standard. Ha 60 pattern e la prima specifica è stata rilasciata nel 2002, una seconda versione nel 2007.

L'interesse di FIPA era creare delle **piattaforma ad agenti** che potessero interagire nella maniera più trasparente possibile. Quindi, tutto quello che serviva (agenti, pagine gialle, pagine bianche dei servizi, sistemi di gestione degli agenti (MTS)) doveva essere implementato nel modo più trasparente possibile in modo da creare piattaforma distribuite in maniera tale da andare a facilitare l'operato del programmatore che aveva il compito di creare le applicazioni.

Le applicazioni in FIPA sfruttano la realizzazione ad agenti e questi hanno la caratteristica di essere:

- **autonomi, reattivi, proattivi, guidati da goal, sociali, adattivi, cognitivi**

In realtà, possono essere questo, ma di per sé FIPA non se ne occupa. Lei, si preoccupa della sola standardizzazione dell'infrastruttura.¹.

5.1 The FIPA Agent Platform

- Le applicazioni (quelle sul top) vanno a sfruttare la possibilità di avere elementi distribuiti tramite agenti.
- gli agenti interagiscono tra loro tramite un ACL standardizzato (in modo che non ci fossero problemi di chiamate, d'invocazione).
- ci sono anche altri due tipi di agenti che si occupano di compiti specifici:
 - i. **AMS** (Agent Management System), si occupa dell'effettiva gestione degli agenti, quindi parliamo ad esempio della creazione
 - ii. **DF** (Directory Facilitator), è una sorta di pagine gialle per la pubblicazione di servizi

La gestione degli stessi nella piattaforma sono a loro volta degli agenti e ci si può interagire tramite ACL.

- **MTS** (Message Transport Service), un servizio che fornisce supporto per la comunicazione tra agenti tramite una qualche forma di protocollo di rete sottostante (HTTP, IIOP, XMPP, STMP, ...)

¹ JADE ne è un esempio

5.2 The Core Specifications

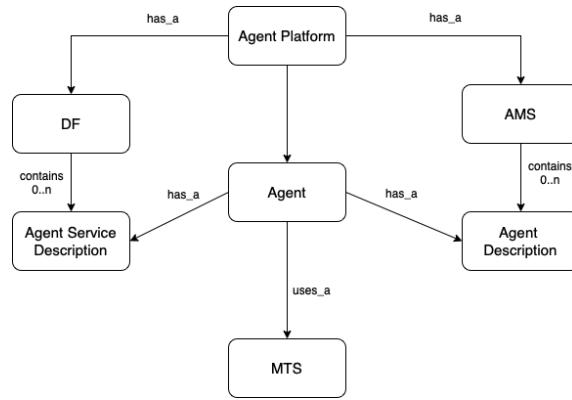


Figure 5.1: Agent Management

Oltre ad avere un Directory Facilitator e un AMS, presenta un certo numero di agenti. Ogni DF, contiene una descrizione di quelli che sono i servizi offerti dagli agenti (un esempio di servizio potrebbe essere la possibilità di essere contattati da altri agenti). Ha anche delle descrizioni. Questa 5.1 è un'organizzazione che deve essere **standardizzata**. Il DF, deve offrire delle operazioni come la *registrazione di agenti*, *modificare alcune descrizioni*, *cercare delle descrizioni*. Il tutto deve essere standardizzato.

5.3 Il Message Transport

L'Agent Message Transport comprende due livelli:

1. il **Message Transport Protocol** (MTP), effettua il trasferimento fisico tra due messaggi ACC;
2. il **Message Transport Service** (MTS), è fornito dall'AP a cui è collegato un agente. L'MTS supporta il trasporto di messaggi ACL FIPA tra agenti su qualsiasi AP e agenti tra diversi AP

5.4 FIPA Message Structure

Il messaggio essendo anch'esso standardizzato presenta una forma ben precisa.

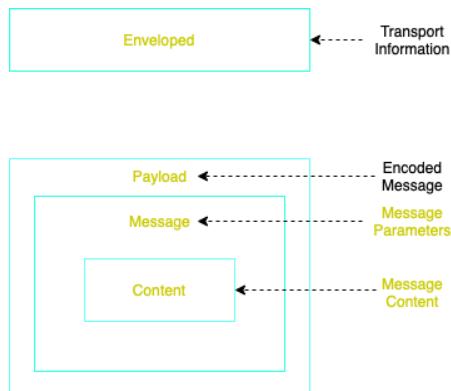


Figure 5.2: FIPA Message Structure

I messaggi presentavano anche un'ontologia per comprendere al meglio il content.

Chapter 6

JADE

Jade è una piattaforma ad agenti che implementa tutti i servizi base e infrastruttura per realizzare applicazioni multi-agenti distribuite.

- specifica il ciclo di vita di un'agente;
- possibilità di fare mobilità degli agenti dove gli agenti stessi si possono spostare dai vari nodi;
- specifica servizi di pagine gialle e pagine bianche;
- specifica e mette a disposizione tutti elementi per parsificare/trasportare messaggi in modo che i collegamenti siano punto-punto;
- specifica un'infrastruttura di sicurezza;
- mette a disposizione tutto un insieme di strumenti grafici per il monitoraggio, la raccolta dei log e il debugging di sistemi distribuiti

Jade, era stato sviluppato anche per Java Micro Edition(J2ME). Jade è una libreria da utilizzare insieme a Java per compilare ed eseguire

- un agente viene definito estendendo la classe *AgentThatSearchesAndUseAService*

```
public class AgentThatSearchesAndUseAService
    extends JadeCore.Agent {
    public void setup() {
        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setService("SearchedService");
        DFAgentDescription[] agents =
            DFService.search(this, dfd);
        ACLMessage msg = new
            ACLMessage(ACLMessage.REQUEST);
        msg.addReceiver(agents[0].getAID());
        msg.setContent("execute service");
        send(msg);
        System.out.println(blockingReceive());
    }
}

public class PeerThatSearchesAndUseAService {
    private void startUxta() {
        netPeerGroup =
            PeerGroupFactory.newNetPeerGroup();
        discoUc =
            netPeerGroup.getDiscoveryService();
        pipeSvc = netPeerGroup.getPipeService();
    }
    private void startClient() {
        Enumeration enum1 =
            discoUc.getLocalAdvertisements(
                DiscoveryService.ADV, "SearchedService",
                SERVICE);
        Enumeration enum2 =
            discoUc.getRemoteAdvertisements(
                null, DiscoveryService.ADV,
                "SearchedService", SERVICE, 1, null);
        EnumerateAdvertisement ead =
            EnumerateAdvertisement.createEnum();
        ModuleSpecAdvertisement nsadv =
            (ModuleSpecAdvertisement)enum.nextElement();
        StructuredTextDocument doc =
            (StructuredTextDocument)
            nsadv.getDocument(new
                MimeMediaType("text/plain"));
        PipeAdv adv =
            nsadv.getAdvertisement();
        PipeAdv pipeadv =
            adv.createAdvertisement();
        Pipe sendpipe = pipeadv.createOutputPipe(
            pipeadv, 10000);
        msg = msg.createMessage();
        msg.setString(TAG, "Request Service");
        sendpipe.send(msg);
        Pipe mypipe =
            pipeSvc.createInputPipe(pipeadv);
        System.out.println(mypipe.waitForMessage());
    }
}
```

Figure 6.1: Esempio di codice in linguaggio Java di Jade