UNIVERSITÀ DEGLI STUDI DI CAGLIARI

DOTTORATO DI RICERCA IN INGEGNERIA

ELETTRONICA ED INFORMATICA

XV CICLO

MODELLI DI COMPUTAZIONE PER LO SVILUPPO DI SISTEMI SOFTWARE AD ELEVATA FLESSIBILITÀ

Relatore:

Giuliano Armano

Tesi di dottorato di:

Eloisa Vargiu

Coordinatore:

Massimo Vanzi

Novembre 2002

RINGRAZIAMENTI

Alla fine di tre anni sono sicuramente tante le persone da ringraziare.

Inizierei da tutti coloro che hanno reso possibile questo dottorato, in modo particolare il mio relatore Giuliano Armano e il CRS4 nelle persone di Gavino Paddeu, Jean Christhope Pazzaglia, Sylvain Giroux e Pietro Zanarini.

Un grazie particolare va anche al gruppo IASC del DIEE, al gruppo NDA del CRS4 e a tutti coloro che mi hanno sopportato e supportato in questi anni…in modo particolare tutta la mia famiglia a cui dedico questo lavoro di tesi!

Con affetto,

Eloisa

INDICE

Introduzione **1**

Studio di Modelli Flessibili 1

Sistemi Orientati agli Oggetti 1

Sistemi Basati sui Componenti 2

Sistemi Orientati agli Agenti 2

Confronto tra le Metodologie Presentate 2

Breve Panoramica degli Argomenti Trattati 5

PARTE I ***7***

I I Sistemi ad Agenti **9**

I.1 Gli Agenti Autonomi 9

I.2 Comportamento Flessibile 10

I.3 Architetture per Sistemi ad Agenti 12

I.3.1 Micro-Architetture 12

I.3.2 Macro-Architetture 18

II Comportamento Proattivo **23**

II.1 Pianificazione Classica 24

II.1.1 Ricerca nello spazio degli stati 25

II.1.2 Ricerca nello spazio dei piani 26

II.1.3 Pianificazione Logica e Problem Solving 29

II.1.4 Pianificazione come Soddisfacimento dei Vincoli 29

II.2 Pianificazione in Domini Reali 31

II.2.1 Caratteristiche dei Domini Applicativi 32

II.2.2 Ambiente Statico ma Parzialmente Accessibile 33

II.2.3 Ambiente Accessibile ma Dinamico 35

II.2.4 Azioni non Deterministiche 37

II.3 Pianificazione Gerarchica 38

II.4 Esempi di Sistemi Esistenti 39

II.4.1 Pianificazione a Ciclo Aperto 39
II.4.2 Pianificazione a Ciclo Chiuso 41

III Comportamento adattativo 43
III.1 Classificazione dei Sistemi di Apprendimento 44
III.1.1 Classificazione Basata sulla Strategia di Apprendimento 45
III.1.2 Classificazione Basata sul Tipo di Conoscenza Acquisita 48
III.1.3 Classificazione Basata sul Dominio Applicativo 50
III.1.4 Classificazione Basata sui Paradigmi di Ricerca 50
III.1.5 Classificazione Basata sugli Orientamenti all'Apprendimento 51

IV Integrazione del Comportamento Proattivo e Adattativo 53
IV.1 PRODIGY 53
IV.2 SOAR 55

V Tecniche di Astrazione 59
V.1 Astrazione nello Spazio degli Stati 59
V.1.1 Modelli Rilassati 59
V.1.2 Modelli Ridotti 60
V.2 Astrazione sugli Operatori 61
V.2.1 Astrazione di Tipo HTN 61
V.2.2 Astrazione Tramite Macro-Operatori 62
V.3 Proprietà Formali delle Gerarchie d'Astrazione 63
V.4 Generazione di Gerarchie di Astrazione 64

PARTE II 65

I Agenti in Ambiente Dinamico 67
I.1 Caratteristiche dell'Ambiente Applicativo 67
I.2 Fonti di Informazione 68
I.3 Panoramica delle Problematiche Generali 68

II Architettura Proposta 73
II.1 Schema dell'Architettura 73

II.1.1 Interazioni Verticali 75
II.1.2 Interazioni Orizzontali 76

III Ruolo dell'Astrazione 79
III.1 Concetti Generali 79
III.2 Il Modello di Astrazione Proposto 80
III.2.1 Astrazione nello Spazio degli Stati 81
III.2.2 Astrazione sugli Operatori 82
III.2.3 Proprietà Formali del Modello Proposto 83

IV Comportamento Proattivo 85
IV.1 Il Sistema di Pianificazione Proposto 85
IV.1.1 HIPE (Hierarchical Interleaving Planning and Execution) 86
IV.1.2 Interazioni con gli Altri Moduli 88

V Comportamento Adattativo 89
V.1 Il Sistema di Apprendimento Proposto 89
V.1.1 Modulo di Chunking 91
V.1.2 Modulo di Filtering 91
V.1.3 Modulo di Schematizing 93
V.1.4 I Moduli di Astrazione e Aggiornamento 93

VI Risultati Sperimentali 97
VI.1 Generazione del Dominio Applicativo 98
VI.1.1 Gerarchia dei Predicati 98
VI.1.2 Gerarchia degli Operatori 99
VI.2 Comportamento Proattivo 100
VI.2.1 Un Caso Studio 100
VI.3 Comportamento Adattativo 101
VI.3.1 Addestramento 101
VI.3.2 Sperimentazione 102
VI.3.3 Un Caso Studio 103

VII Sviluppi Futuri 107

PARTE III ***109***

I Verso un Modello ad Elevata Flessibilità **111**

II HW[ ] un Pianificatore Gerarchico Parametrico **113**

II.1 Un'Estensione del PDDL per Gestire l'Astrazione 114

II.2 Algoritmo di Pianificazione 116

II.3 Generazione di Astrazioni 116

III Risultati Sperimentali **119**

III.1 Test sui Domini Classici 120

III.1.1 Test sul Dominio *Elevator* 120

III.1.2 Test sul Dominio *Logistics* 121

III.1.3 Test sul Dominio *Blocks-World* 122

III.1.4 Test sul Dominio *Gripper* 124

III.1.5 Test sul Dominio *Zeno-Travel* 124

IV Sviluppi Futuri **127**

Conclusioni **129**

Bibliografia **131**

INDICE DELLE FIGURE

Figura 1 - La subsumption architecture. 14
Figura 4 – Architettura INTERRAPP. 16
Figura 5 - Architettura BDI 17
Figura 6 - Architettura FIPA 20
Figura 11 - L'ambiente: caratteristiche, problematiche, ipotesi e soluzioni. 71
Figura 12 - Schema generale dell'architettura. 74
Figura 13 - Interazioni Verticali. 75
Figura 14 - Interazioni orizzontali. 76
Figura 15 - Piano astratto 80
Figura 16 - Relazione di astrazione nel modello proposto. 82
Figura 17 - Struttura di un piano gerarchico. 86
Figura 18 - L'esecuzione può iniziare prima di terminare il raffinamento a livello astratto. 87
Figura 19 - Ciclo di vita del modulo di apprendimento. 90
Figura 20 - Una vista del mondo virtuale. 97
Figura 21 - Esempio di operatori del livello situato. 98
Figura 22 - Esempio di operatori del livello strategico. 99
Figura 23 – Un esempio di piano che mostra come il livello strategico e quello situato collaborano per risolvere un dato problema. 101
Figura 24 - Struttura della rete neurale utilizzata. 102
Figura 25 – Scenario per l'addestramento. 103
Figura 26 - Risoluzione del problema di addestramento proposto. 104
Figura 27 - Scenario per il testing. 105
Figura 28 – Architettura di HW[ ]. 113
Figura 29 - Descrizione del dominio *elevator* a livello ground. 114
Figura 30 - Descrizione del dominio *elevator* a livello astratto. 115
Figura 31 - Gerarchia di astrazione per il dominio *elevator*. 116
Figura 32 - Esempio di problema per il dominio *elevator*, (a) stato iniziale (b) stato finale. 121
Figura 33 - Comparazione dei tempi di CPU per il dominio *elevator*. 121
Figura 34 - Esempio di problema per il dominio *logistics*, (a) stato iniziale (b) stato finale. 122

Figura 35 - Comparazione dei tempi di CPU per il dominio *logistics*. 122
Figura 36 - Esempio di problema per il dominio *blocks-world*, (a) stato iniziale (b) stato finale. .. 123
Figura 37 - Comparazione dei tempi di CPU per il dominio *blocks-world*. 123
Figura 38 - Comparazione dei tempi di CPU di LPG e HW[LPG] per il dominio *blocks-world*. 124
Figura 39 - Esempio di problema per il dominio *gripper*, (a) stato iniziale (b) stato finale. 124
Figura 40 - Comparazione dei tempi di CPU per il dominio *gripper*. 125
Figura 41 - Esempio di problema per il dominio *zeno-travel*, (a) stato iniziale (b) stato finale. 125
Figura 42 - Comparazione dei tempi di CPU per il dominio *zeno-travel*. 126

INDICE DELLE TABELLE

Tabella 1 – Classificazione delle strategie di pianificazione, degli approcci. 40

Tabella 2 – Metriche proposte. 92

Tabella 3 – Un esempio di sequenze identificate come rilevanti per l'astrazione. 102

Tabella 4 – Output della rete neurale sulle sequenze del training set. 104

Tabella 5 – Comparazione dei risultati ottenuti dai pianificatori GP, BB e LPG e dalle loro controparti gerarchiche. 119

INTRODUZIONE

Negli ultimi anni si è assistito al diffondersi di applicazioni e servizi software in grado di prendere decisioni in maniera autonoma. In particolare, sono sempre più richieste architetture aperte in grado di evolvere. Questo fenomeno ha quindi portato al crescente aumento della complessità nella progettazione di sistemi software. La definizione di appropriati meccanismi di comunicazione, di negoziazione e di coordinamento tra i componenti software e gli attori umani motiva lo studio di tecniche e metodi che supportino il progettista attraverso tutto il processo di sviluppo del software e l'utente finale durante il reale utilizzo del sistema.

Studio di Modelli Flessibili

Per trattare con sistemi particolarmente complessi, un aiuto viene dall'ingegneria del software, che ha fornito nel corso degli anni molte tecniche, metodi e metodologie che hanno permesso di trattare sistemi software sempre più complessi. L'approccio sistematico all'analisi, alla progettazione, alla programmazione e manutenzione di sistemi software ha obbligato molte comunità scientifiche a darsi delle regole e linguaggi comuni per risolvere problemi di grossa complessità.

Dal punto di vista dell'ingegneria del software, storicamente negli anni '80 si è delineato il successo dell'approccio orientato agli oggetti, in seguito negli anni '90 ha cercato di emergere il paradigma basato sui componenti, infine, sul finire degli anni '90 è emerso l'approccio orientato agli agenti. Nel seguito del paragrafo mostreremo brevemente l'evoluzione dell'ingegneria del software nel senso dell'aumento della flessibilità: dagli oggetti agli agenti, passando per i componenti, a questo proposito in [Armano00a] viene proposta un'architettura ad agenti basata su un modello ad oggetti attivi.

Sistemi Orientati agli Oggetti

Un oggetto è un'entità software che incorpora, idealmente, dati e procedure atte a manipolarli. Tipicamente gli oggetti sono istanze di classi. Una classe specifica il comportamento di ogni oggetto che sarà istanziato da essa, e può ereditare comportamenti da altre classi tramite il meccanismo dell'eredità. In una visione object-oriented, un sistema software sarà costituito da oggetti che offrono (ed eseguono su richiesta) i servizi specificati dalle procedure che incapsulano. Per approfondimenti, si consulti, ad esempio, [Meyer97].

Sistemi Basati sui Componenti

Un componente è un'entità software che incorpora dati e procedure atte a manipolarli. Rispetto agli oggetti, però, un componente aderisce ad uno standard (formale o "de facto") che ne specifica le modalità di utilizzo, consentendo così di realizzare sistemi complessi a partire da librerie di componenti preesistenti. In una visione basata sui componenti, un sistema software sarà costituito da componenti tipicamente reperiti da apposite librerie, e integrati nel sistema in accordo con lo standard adottato. Per approfondimenti, si consulti, ad esempio [Szyperski97].

Sistemi Orientati agli Agenti

Un agente è un'entità software in grado di eseguire azioni complesse in maniera autonoma. Rispetto ad un oggetto, un agente manifesta un comportamento flessibile che gli consente di raggiungere i propri obiettivi autonomamente. In una visione orientata agli agenti, un sistema software sarà costituito da comunità di agenti che cooperano e interagiscono secondo modalità stabilite dai messaggi che gli agenti si scambiano. Per approfondimenti, si consulti, ad esempio [Ciancarini01].

Confronto tra le Metodologie Presentate

Per poter effettuare un confronto tra il ben noto approccio orientato ad oggetti e gli approcci di nuova generazione –basato sui componenti e orientato agli agenti– ci baseremo sulle tecniche proposte da [Booch94] per affrontare la complessità nel software: decomposizione, astrazione e organizzazione.

Sistemi Orientati agli Oggetti

Decomposizione

La decomposizione, tecnica fondamentale nella progettazione orientata agli oggetti, consiste nel suddividere i problemi in sotto-problemi (dunque più piccoli e maneggiabili) ognuno dei quali possa essere trattato indipendentemente. Com'è facile intuire, questa tecnica agevola il progettista nella gestione della complessità in quanto ad ogni istante viene considerata solamente una porzione del codice.

Astrazione

L'astrazione è il processo di definizione di un modello semplificato del sistema che enfatizza alcuni dettagli o proprietà, eliminandone altri. Poiché questa tecnica limita il campo di interesse del

progettista, in ogni istante l'attenzione può essere focalizzata sugli aspetti salienti del problema, anziché su dettagli poco importanti.

Organizzazione

L'organizzazione rappresenta il processo di identificare e gestire le interrelazioni tra i vari componenti. L'abilità di specificare e notificare le relazioni organizzative aiuta il progettista in due modi. In primo luogo, permettendo il raggruppamento di vari componenti base in modo da poter essere trattati come un unità di analisi di alto livello. In secondo luogo, essa fornisce un mezzo per descrivere le relazioni di alto livello tra le varie unità.

In base a queste tecniche verrà effettuato il confronto con le altre metodologie considerate.

Sistemi Basati sui Componenti

Decomposizione

La decomposizione consiste nella suddivisione in componenti. Durante la fase di ingegnerizzazione del dominio, si crea un modello del dominio dell'applicazione che costituirà poi la base per l'analisi dei requisiti dell'utente. L'individuazione dell'architettura fornirà in seguito il punto di partenza per la progettazione. A conclusione di questa fase, saranno stati individuati e acquistati i componenti riutilizzabili, selezionate le opportune librerie preesistenti e, se necessario, realizzati componenti ad hoc, al fine di dotare gli ingegneri del software degli strumenti adatti per lo sviluppo a componenti.

Astrazione

La nozione di componente è strettamente legata a quella di astrazione: un'entità racchiusa in se stessa, con certe linee di confine attorno ad essa, che possa essere composta successivamente con altre entità. Il fatto che le astrazioni abbiamo limiti è di importanza cruciale per la composizione software, dato che questo fornisce un mezzo per strutturare il software, controllando le interazioni tra componenti e verificando un appropriato assemblaggio. Purtroppo, la maggior parte degli ambienti software impongono alcune restrizioni all'uso di astrazioni: le linee di confine non possono essere tracciate arbitrariamente, secondo le esigenze dell'utente, ma devono seguire specifici modelli.

Una completa flessibilità nel tracciare linee di confine tra astrazioni presuppone che tutti i componenti software siano trattati come valori di prima classe che possono essere passati come parametri ad altri componenti. I linguaggi più avanzati in questa direzione sono linguaggi funzionali, in cui "tutto è una funzione", e le funzioni sono dati. Dal momento che l'astrazione

funzionale è il solo meccanismo di astrazione, i programmatori godono di una grande flessibilità nella scelta di quali aspetti inserire nella definizione di una funzione e quali aspetti lasciare non fissati nei parametri.

Oltre a trattare i componenti come valori, un'altra proprietà delle astrazioni, che ha un grande impatto sulla componibilità è la scalabilità, vale a dire la possibilità di usare gli stessi meccanismi di astrazione e di composizione in ogni livello di una configurazione. Il vantaggio è l'economia di concetti e il fatto che non è presente alcun limite alla granularità dei componenti.

Organizzazione

Esistono parecchi motivi per cui si ha bisogno di un ciclo di vita orientato ai componenti, ed esiste un gran numero di tendenze nei moderni linguaggi che dimostrano la possibilità di migliorare il tradizionale assemblaggio del software a tre fasi.

Ogni volta che i componenti vengono assemblati per assolvere un compito comune, c'è sempre un contratto implicito fra essi sui termini della collaborazione. Affinché si possa verificare la correttezza di una configurazione, i contratti devono essere resi espliciti e confrontati per evidenziare eventuali discrepanze. Questo problema può essere risolto da sistemi tipizzati, sebbene i sistemi tipizzati convenzionali non colgano in generale tutti gli aspetti di un contratto a causa della loro limitata espressività.

Sempre riguardo al problema di espliciti contratti tra componenti, si dovrebbe ricordare un'altra famiglia di soluzioni che non pongono il contratto dentro i componenti, bensì al di fuori. La composizione interlinguistica è persino l'unica possibilità, dal momento che sarebbe piuttosto difficile confrontare contratti specificati in diversi linguaggi e modelli.

Sistemi Orientati agli Agenti

Decomposizione

L'approccio agent-oriented sostiene la decomposizione dei problemi in termini di agenti autonomi che possono impegnarsi in interazioni flessibili e di alto livello. Flessibilità, in questo contesto, significa che i risolutori del problema hanno un proprio thread di controllo (ovvero sono attivi) e che decidono quali azioni eseguire e in quale istante. Decomporre un problema in questo modo aiuta a ingegnerizzare i sistemi complessi in due modi:

- è una rappresentazione naturale per sistemi complessi che inevitabilmente sono distribuiti e hanno diversi centri di controllo. Questa decentralizzazione, a sua volta, riduce la complessità del controllo e produce un basso grado di dipendenza tra componenti. Il fatto che gli agenti siano attivi significa che essi sanno quando dovrebbero agire e quando invece devono semplicemente

aggiornare il loro stato (al contrario degli oggetti passivi che hanno bisogno che qualche entità esterna ordini loro come agire). Tale autonomia riduce la complessità del controllo poiché la conoscenza di controllo si sposta da un archivio centralizzato verso l'interno di ogni singolo componente.

- poiché le decisioni su quali azioni debbano essere eseguite sono delegate ad entità autonome, la decisione può essere basata sulla situazione locale del componente. Ciò permette alla selezione di rispondere allo stato effettivo dell'agente, piuttosto che alla percezione di questo stato che può avere una qualche entità esterna.

Astrazione

L'astrazione è un concetto fondamentale anche nell'ingegneria del software orientata agli agenti. Il concetto di astrazione in questo contesto si affianca al concetto di interazione, ovvero politiche di collaborazione e coordinamento. L'astrazione, infatti, è concepita nel senso che ad ogni livello di astrazione, si hanno insiemi di entità che collaborano per raggiungere un risultato di alto livello. Il concetto di collaborazione, infatti, risulta essere molto importante

Organizzazione

I costrutti organizzativi sono entità di primaria importanza nei sistemi ad agenti. Vengono quindi fatte rappresentazioni esplicite di relazioni organizzative e strutture. Inoltre, i sistemi ad agenti hanno i meccanismi di elaborazione che permettono di creare, mantenere e distruggere le organizzazioni.

La potenza rappresentativa permette ai sistemi ad agenti di sfruttare due aspetti della natura dei sistemi complessi: (i) la nozione di componente primitivo può essere modificata a seconda delle necessità dell'analista; (ii) i singoli agenti o i gruppi organizzativi possono essere sviluppati in relativa indipendenza e poi aggiunti nel sistema in maniera incrementale.

Breve Panoramica degli Argomenti Trattati

In questo lavoro di tesi, poiché si è interessati allo studio di modelli di computazione per lo sviluppo di sistemi software ad elevata flessibilità, è stato preso in considerazione il modello ad agenti.

Nella prima parte verrà esposto lo stato dell'arte riguardante tali modelli. In particolare, dapprima verranno dati i concetti base relativi al significato di agente e alle sue possibili implementazioni architetturali (Capitolo I). In seguito si focalizzerà l'attenzione sugli aspetti, a nostro parere, chiave: il comportamento proattivo (Capitolo II), il comportamento adattativo

(Capitolo III) e la loro integrazione nei sistemi esistenti (Capitolo IV). Infine, poiché determinante in tutto il lavoro di tesi e poiché ortogonale agli argomenti trattati, verrà data una descrizione delle tecniche di astrazione note in letteratura (Capitolo V).

Nella seconda parte di questo lavoro, verrà presentato un prototipo che è stato realizzato in un progetto reale. Dapprima, verrà definito l'ambiente applicativo (Capitolo I) e l'architettura ad agenti proposta (Capitolo II); in seguito verrà introdotto il ruolo dell'astrazione e verrà proposto un modello di astrazione (Capitolo III). In linea con quanto argomentato nella prima parte, verranno dunque esposte le caratteristiche, per noi salienti, del comportamento dell'agente, ovvero le capacità proattive (Capitolo IV) e adattative (Capitolo V). Infine verranno mostrati i risultati sperimentali (Capitolo VI) e discussi i possibili sviluppi futuri (Capitolo VII).

Nella terza parte, infine, verranno ulteriormente evidenziati gli aspetti di flessibilità allo scopo di realizzare un modello di computazione ad elevata flessibilità. In particolare, verranno dapprima presentati i concetti generali (Capitolo I) e verrà proposto un approccio parametrico gerarchico (Capitolo II). Infine verranno mostrati dei preliminari risultati sperimentali (Capitolo III) e verranno discussi i possibili sviluppi futuri (Capitolo IV).

Parte I

I I SISTEMI AD AGENTI

Sebbene si possa affermare che il fine ultimo della ricerca nel settore dell'Intelligenza Artificiale sia quello di realizzare sistemi che esibiscano, in modi differenti, un comportamento intelligente (che esso si rifaccia o no al modello umano), l'interesse verso lo studio e la progettazione dei sistemi ad agenti autonomi ha subito una forte crescita solo a partire dalla fine degli anni '80. Da allora molti sono stati i temi di discussione, non ultimo dei quali la definizione stessa di agente e delle principali proprietà che permettano di caratterizzare, e quindi distinguere, tali sistemi. Potrebbe quindi suscitare una certa sorpresa il fatto che, ancora oggi, non esista una definizione universalmente accettata di agente, per quanto si sia raggiunto un accordo generale su alcune proprietà fondamentali che un sistema deve possedere per essere definito tale.

I.1 Gli Agenti Autonomi

Un agente è un sistema (hardware e/o software) progettato affinché operi autonomamente in un ambiente in modo da soddisfare le proprie specifiche di progetto [Wooldridge95].

Strettamente legato al concetto di agente, come suggerisce il nome stesso, è il concetto di azione in un dato ambiente (*environment*), che può essere fisico o software; la proprietà distintiva di tali sistemi è sicuramente un superiore livello di autonomia, dove con questo termine si intende la capacità di svolgere un compito senza ricorrere all'intervento del progettista, mantenendo il controllo sul proprio stato interno e sulle proprie strutture dati. Ovviamente queste definizioni hanno un senso solo in riferimento ad un certo livello complessità delle funzioni da svolgere: se si trascura questa condizione, anche un semplice termostato può essere definito un agente autonomo. La genericità della definizione data fa comunque sì che molti sistemi, da alcuni programmi software ai sistemi di controllo automatico, possano essere classificati come agenti: in particolare un sistema di controllo automatico è in grado di garantire il comportamento desiderato anche in presenza di disturbi, ossia sotto condizioni di variabilità ambientale.

La definizione di agente intelligente aggiunge alle caratteristiche già descritte la nozione di flessibilità [Wooldridge99]:

Un agente intelligente opera in un dato ambiente in modo flessibile, allo scopo di perseguire in esso i propri obiettivi, acquisendo informazioni tramite un sistema di percezione ed eseguendo effettivamente delle azioni per mezzo di un sistema attuatore.

La flessibilità di comportamento è mostrata sia nei confronti dell'ambiente che degli obiettivi: la prima caratteristica permette di aumentare ancora il livello di complessità del contesto applicativo,

mentre la flessibilità rispetto agli obiettivi indica la possibilità di cambiarli, o di averne più di uno contemporaneamente, condizione molto differente da quella di un sistema che rispetti semplicemente le proprie specifiche; in altre parole, ad un agente intelligente non occorre dire che cosa fare, ma che cosa deve ottenere come risultato finale della propria attività. In questo ulteriore livello si può racchiudere il significato della caratteristica aggiuntiva di "intelligenza".

Il contesto applicativo può essere un ambiente fisico reale, come nel caso di un robot, oppure una sua simulazione più o meno fedele, come un videogame, o infine un ambiente puramente software (come Internet); in quest'ultimo caso si parla di agenti software (*software agents*). Le azioni e le percezioni ovviamente differiscono da caso a caso: per un agente software, ad esempio, esse consisteranno in chiamate a funzioni o metodi. L'enfasi sulla caratteristica di integrazione in un ambiente ha dato luogo allo specifico attributo "*situato*": con questo termine, divenuto comune in IA verso la fine degli anni '80, viene enfatizzato il fatto che il processo di deliberazione avviene in un agente direttamente connesso all'ambiente, da cui riceve le percezioni ed in cui agisce effettivamente, modificandolo [Russell95]. In questo senso un sistema esperto non è un agente, perché agisce da consulente di un utente umano, il quale gli pone le domande e si sostituisce ad esso nell'azione. Un sistema esperto è svincolato dall'ambiente, e sebbene possieda capacità di ragionamento autonomo, non agisce autonomamente. Esso piuttosto potrebbe far parte, come modulo di inferenza, dell'architettura di un agente autonomo.

Gli agenti intelligenti sono quindi entità dotate di un comportamento flessibile, si adattano ai cambiamenti dell'ambiente, possono conseguire obiettivi complessi in maniera autonoma e mostrano una capacità sociale che può arrivare sino alla collaborazione di più agenti verso un obiettivo comune.

I.2 Comportamento Flessibile

In letteratura sono state date due diverse definizioni di comportamento flessibile di un agente in base alle caratteristiche di cui un agente è dotato: la definizione debole e la definizione forte.

In base alla definizione debole, un agente esibisce un comportamento flessibile se è dotato delle seguenti caratteristiche:

- *Proattività*[1]: indica la capacità di porre in atto un comportamento che conduca al raggiungimento di un dato obiettivo. Fondamentale a questo scopo è la capacità di sintetizzare ed eseguire azioni o sequenze di azioni (denominate piani) in funzione dell'obiettivo e della

[1] Indicata anche come comportamento orientato all'obiettivo (*goal-oriented*).

situazione di partenza. Ricade nel comportamento proattivo anche la capacità di modificare, se possibile, i propri piani, adattandoli alle mutevoli condizioni di un ambiente dinamico.

- *Autonomia*: il significato di questa proprietà si chiarisce non tanto fornendo un'interpretazione del concetto di autonomia quanto indicando in che cosa un agente è autonomo, e quali sono i limiti della sua autonomia. Anche a questo proposito sono presenti due interpretazioni, una debole ed una forte: la prima, e la più comunemente usata in letteratura, si riferisce all'autonomia dell'agente nella scelta ed esecuzione delle azioni per il raggiungimento di un obiettivo assegnato dall'esterno; la seconda (e più radicale) prevede che anche la scelta degli obiettivi venga fatta in modo autonomo, prospettando in tal modo una sorta di vita propria del sistema [Castelfranchi94].
- *Reattività*: indica la capacità di monitorare l'ambiente per individuare eventi di interesse, siano essi pericoli oppure opportunità favorevoli, e reagire a tali eventi ponendo in atto opportuni comportamenti.
- *Capacità sociale*: questa proprietà è legata ad un contesto più ampio in cui coesistono ed operano più entità autonome. Il livello di interazione, ovviamente, differisce da sistema a sistema: si va da una relativa autosufficienza a forme complesse di competizione e cooperazione in vista di obiettivi comuni o divergenti.

In base alla definizione forte, un agente è un sistema software che ha le seguenti proprietà:

- *Mobilità*: indica la capacità di un agente di muoversi attraverso la rete spostandosi tra i nodi componenti.
- *Veracità*: indica l'impossibilità di un agente di comunicare informazioni false.
- *Benevolenza*: un agente dotato di questa caratteristica, non può rifiutarsi di effettuare un dato compito e/o di non portarlo a termine.
- *Razionalità*: indica la capacità di un agente di agire per il raggiungimento dei propri obiettivi, senza ostacolare se stesso in questo compito.
- *Apprendimento / Adattività*: questa proprietà indica la possibilità che un agente possa apprendere nuove nozioni derivanti dalla sua attività precedente. Inoltre, un agente dotato di questa caratteristica può modificare il proprio comportamento in base al reperimento di nuove informazioni o per adattarsi ad eventuali cambiamenti nell'ambiente circostante.

In questo lavoro di tesi siamo interessati allo studio di agenti che esibiscano un comportamento flessibile in base a una definizione ibrida tra le due presentate. Gli agenti presentati nella parte seconda di questo lavoro di tesi saranno, infatti, autonomi, dotati di un comportamento proattivo e adattativo, e, infine, in grado di interagire con altri agenti secondo delle semplici politiche di

comunicazione. Gli agenti saranno inoltre considerati sempre benevoli e razionali e non saranno presi in considerazione problemi di non veracità.

I.3 Architetture per Sistemi ad Agenti

Dal punto di vista della progettazione di sistemi ad agenti, esistono due punti di vista differenti: un primo approccio focalizza sulle caratteristiche del singolo agente (la microarchitettura), mentre il secondo si occupa di definire i protocolli di interazione, il cosiddetto livello sociale (la macroarchitettura).

I.3.1 Micro-Architetture

Una volta definite le principali caratteristiche comportamentali di un agente, non è affatto immediato individuare un'architettura o delle tecnologie che presentino dei vantaggi in senso assoluto rispetto alle altre: ogni scelta è il risultato di un compromesso fra potere di modellazione ed efficienza di comportamento.

Quel che appare chiaro è la necessità di dotare tali sistemi di un sottosistema di percezione, un attuatore, un meccanismo di selezione delle azioni e di reazione ad eventi esogeni. Tali caratteristiche possono essere realizzate tramite il ricorso a soluzioni di rappresentazione, architetturali e tecnologiche anche molto differenti, e, infatti, molto differenti sono le soluzioni proposte nel corso degli anni dai ricercatori IA. E' possibile individuare alcuni fondamentali tipi di approccio alla progettazione pratica di sistemi intelligenti: uno basato sulla logica (approccio deliberativo), uno basato su riflessi (approccio reattivo), uno ibrido tra i due, ed infine l'approccio di tipo Belief Desire Intention (BDI).

I.3.1.1 Architetture Deliberative

L'approccio deliberativo è quello più antico, esso consiste nel considerare un agente intelligente come un particolare tipo di sistema basato sulla conoscenza. Esso prevede che l'agente possieda una rappresentazione esplicita del mondo, espressa come un insieme di formule logiche, un sistema a frame o una rete semantica e, tramite una qualche forma di manipolazione simbolica, sia in grado di:

- aggiornare la propria conoscenza sul mondo in base alle percezioni ricevute, in questa fase può anche essere compresa la deduzione di proprietà del mondo non evidenti;
- decidere autonomamente, in base alle informazioni disponibili, le azioni da eseguire per il raggiungimento del proprio obiettivo;
- aggiornare la propria conoscenza in base alle azioni eseguite.

Il processo di manipolazione simbolica della conoscenza per eccellenza è l'inferenza (adottato inizialmente da [Green69]); il suo utilizzo però si rivelò presto inadatto a causa della sua intrattabilità[2] che lo rende impraticabile in ambienti dinamici con un tempo fra una variazione e l'altra inferiore al tempo richiesto dal processo di inferenza.

Il processo di inferenza puro venne prima sostituito dal calcolo delle situazioni (*situation calculus*) [McCarthy69], che però rivelò anch'esso delle gravi carenze, e successivamente, all'inizio degli anni '70, dalla pianificazione (*planning*) [Fikes71].

Il principale problema connesso alla realizzazione di una architettura deliberativa, dunque, è la complessità computazionale; vi sono inoltre altri problemi ancora aperti, come la mappatura tra percezioni provenienti dai sensori e rappresentazione simbolica interna.

I.3.1.2 Architetture Reattive

L'approccio reattivo[3] nasce nella seconda metà degli anni '80 come reazione ai problemi di intrattabilità ed efficienza tipici delle architetture logiche. I sistemi reattivi sono accomunati dal rifiuto di qualsiasi forma di rappresentazione e ragionamento simbolici; il grande vantaggio che ne deriva è la semplicità e quindi velocità dei processi di decisione, che rende queste architetture particolarmente indicate per condizioni di rapida variabilità ambientale, in cui è importante agire nel minor tempo possibile. L'esempio più rappresentativo è la *"subsumption architecture"* [Brooks86] (vedi Figura 1), in cui il comportamento complessivo discende dall'interazione fra molteplici comportamenti elementari (*behaviors*) nei quali la scelta delle azioni da compiere è il risultato di una mappatura percezione-azione. I behaviors competono per il controllo del sistema, nel rispetto di una precisa gerarchia.

Sono stati realizzati diversi sistemi puramente event-driven che manifestano un comportamento reattivo e proattivo, non si tratta comunque di un metodo applicabile alla generalità dei casi, ma solo quando l'azione migliore può essere facilmente calcolata a partire dallo stato corrente del mondo, ovvero quando non è necessario uno sguardo al futuro e le azioni non interferiscono le une con le altre.

---

[2] L'inferenza in logica del prim'ordine è semi-decidibile e per le sue estensioni modali il problema è drammaticamente amplificato.

[3] Noto anche come approccio *situated action* o *event-driven*.

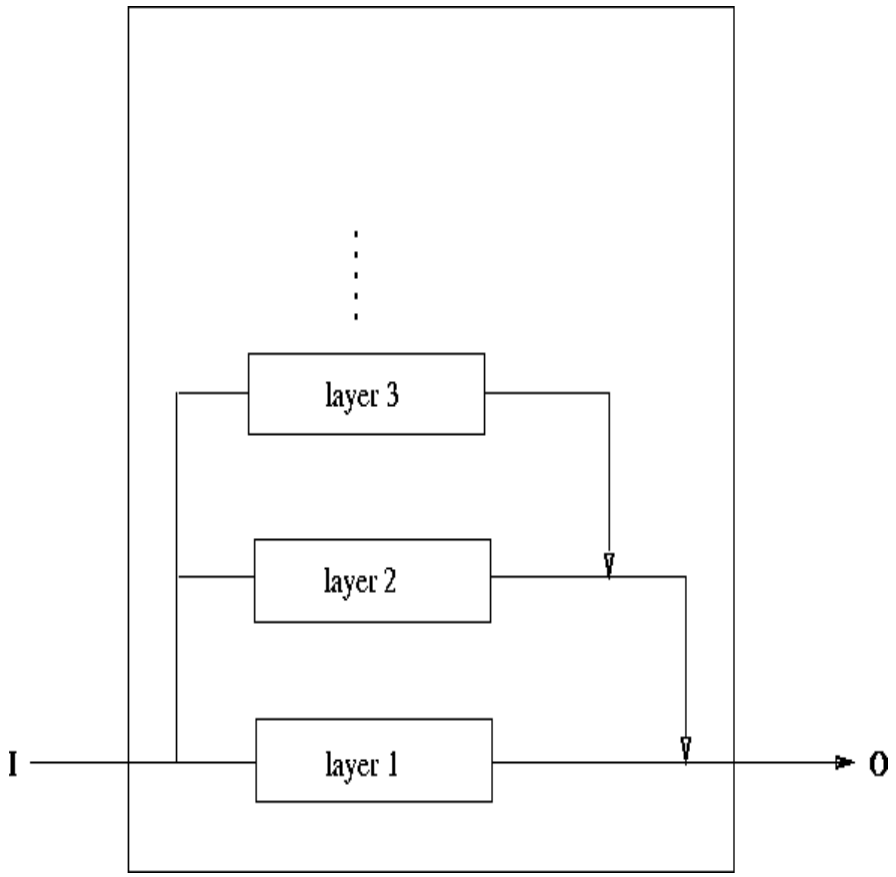


Figura 1 - La subsumption architecture.

I.3.1.3 Architetture Ibride

Le architetture ibride utilizzano congiuntamente le caratteristiche delle architetture reattive e di quelle basate sulla logica. Infatti, i limiti dei sistemi event-driven corrispondono ai punti di forza dall'approccio logic-based e viceversa. Un approccio combinato, o ibrido, sembra essere dunque promettente.

Architetture layered

Un particolare tipo di soluzione ibrida è costituito dalle architetture layered: in esse ad ogni singolo comportamento dell'agente è dedicato un livello software a parte. Considerando le tre caratteristiche fondamentali (reattività, proattività, comportamento sociale), si capisce che il numero di livelli è tipicamente tre, anche se in generale può essere N.

In base al modo in cui è concepito il flusso del controllo, possiamo suddividere le architetture layered in due tipologie fondamentali: architetture orizzontali ed architetture verticali.

Architetture orizzontali

Nelle architetture orizzontali, ciascun livello è a contatto diretto con l'ambiente, ed ha la facoltà di suggerire l'esecuzione di una certa azione; la scelta fra le azioni proposte viene fatta da un modulo di controllo centrale, che assume il ruolo di mediatore fra i livelli (vedi Figura 2).

Un esempio di architettura orizzontale è dato dal sistema TouringMachines [Ferguson92], composto da tre livelli: un reactive layer, un planning layer, ed un modeling layer; quest'ultimo contiene una rappresentazione del mondo in quanto composto da entità autonome ed interagenti.

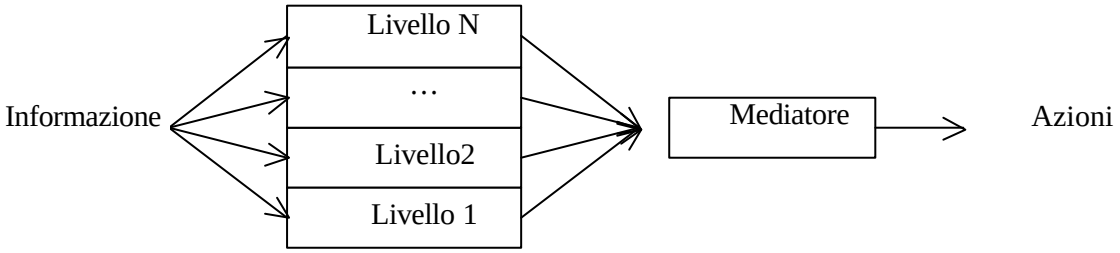


Figura 2 - Architettura orizzontale.

Architetture verticali

Nelle architetture verticali non c'è bisogno di un controllo centralizzato perché esiste una gerarchia ben definita. Una caratteristica tipica delle architetture verticali è la presenza, in ogni livello, di una rappresentazione interna del mondo differente: quindi, per N livelli, N basi di conoscenza.

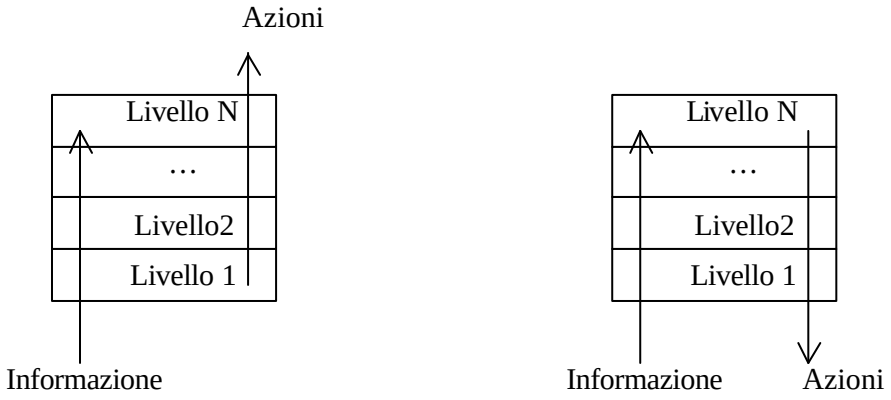


Figura 3 - Architetture verticali one-pass e two-pass.

L'accesso agli input ed alle azioni sul mondo è detenuto da un livello solo; nelle architetture verticali di tipo one-pass i due livelli non coincidono, mentre in quelle di tipo two-pass esiste un unico livello che raccoglie l'informazione ed esegue le azioni, e che quindi rappresenta l'unico punto di contatto del sistema con il mondo. Nelle prime, informazione e controllo fluiscono nella stessa direzione, dal primo livello fino all'ultimo, che esegue le azioni; nelle seconde, l'informazione fluisce dal primo livello all'ultimo mentre il controllo segue la direzione inversa (vedi Figura 3).

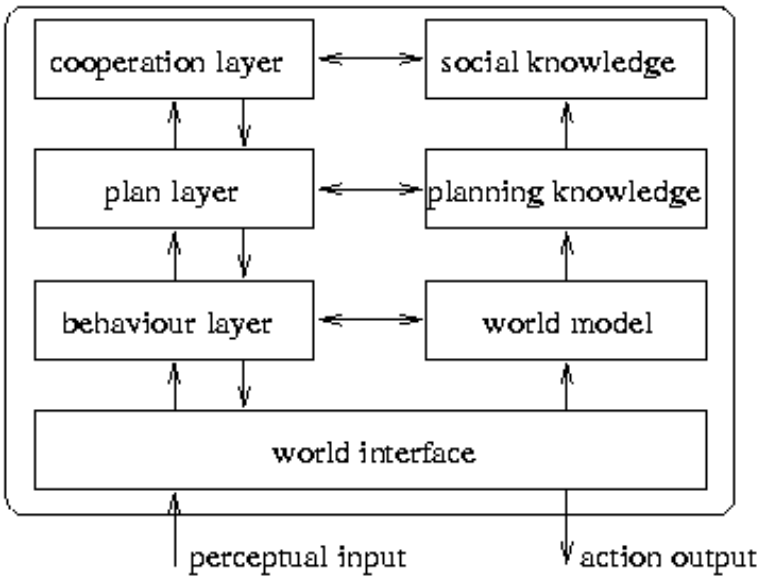


Figura 4 – Architettura INTERRAPP.

Un esempio di architettura verticale two-pass è INTERRAPP [Muller97], anch'essa a tre livelli: un behaviour based layer, un local planning layer, ed un cooperative planning layer (vedi Figura 4).

Nella parte seconda di questo lavoro di tesi verrà presentato un approccio ibrido, basato su un'architettura verticale di tipo two-pass, che presenta delle differenze concettuali con le architetture presenti in letteratura.

I.3.1.4 Architettura Belief Desire Intention (BDI)

Queste architetture si basano sul concetto di *practical reasoning* [4], proposto dal filosofo Michael Bratman nel 1987 [Bratman87]. Il practical reasoning è composto da due principali processi: il

[4] Analisi del processo di ragionamento compiuto delle persone quando mirano a soddisfare le loro aspettative nel mondo reale, analisi compiuta nella prospettiva di realizzare un sistema artificiale che simuli un tale comportamento.

processo di decisione tra un insieme di differenti prospettive; e il processo per il raggiungimento della condizione prospettata. Momento fondamentale del practical reasoning è quindi la scelta, fra quelli che si ritiene raggiungibili, dei propri obiettivi. Questo processo di scelta presuppone la conoscenza sia delle opzioni a disposizione, sia dei propri desideri. Le decisioni prese hanno differenti gradi di complessità e d'importanza: si possono fare scelte fondamentali, come decidere di sfruttare un'opportunità totalmente nuova e inaspettata, o decidere semplicemente il modo migliore di fare qualcosa. Una volta scelto un determinato obiettivo, ci si adopera per conseguirlo finché non lo si raggiunge; l'obiettivo viene abbandonato solo se si ritiene che la possibilità di conseguirlo è divenuta remota o se si prospetta un'alternativa più desiderabile. Nella teoria del practical reasoning questo passaggio è descritto come trasformazione delle opzioni scelte in intenzioni, che hanno la caratteristica di persistere finché le nostre credenze (*belief*) le fanno ritenere ragionevoli.

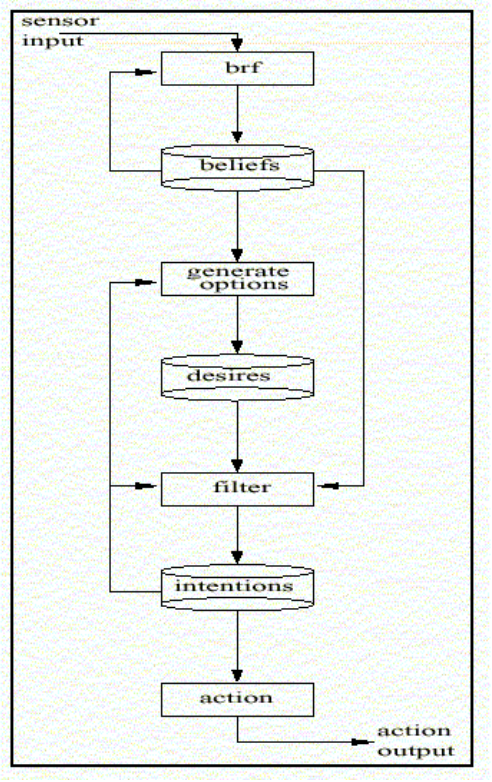


Figura 5 - Architettura BDI

Nel processo descritto si possono chiaramente identificare il comportamento proattivo (persistenza delle intenzioni) e reattivo (abbandono delle intenzioni perché non più convenienti ed adozione di un nuovo obiettivo). Il comportamento reattivo ci permette di verificare le nostre scelte alla luce delle ultime informazioni, invece di concentrarci ciecamente su ciò che abbiamo precedentemente deciso.

La caratteristica fondamentale delle architetture Belief Desire Intention (BDI) è quella di utilizzare strutture dati che corrispondono ai *beliefs, desires* ed *intentions* cui si riferisce la teoria del practical reasoning; i processi di decisione sono realizzati da funzioni che agiscono su queste strutture (si veda la Figura 5). Al modello BDI non corrisponde un'unica realizzazione architetturale concreta: i principali riferimenti sono il sistema IRMA [Bratman88], di tipo deliberativo ed il sistema PRS [Georgeff87], di tipo ibrido.

I.3.2 Macro-Architetture

Appare pressoché universalmente accettato il fatto che l'esistenza di un agente è subordinata all'esistenza nel proprio ambiente di altri agenti con cui interagire. In un contesto di questo tipo, risulta necessario introdurre dei modelli per la coordinazione tra gli agenti. Gli agenti inoltre devono essere in grado di comunicare, e di cooperare e/o competere con gli altri agenti del dominio per il raggiungimento di un obiettivo comune.

I.3.2.1 Coordinazione

Il paradigma di coordinazione nasce per alleviare i problemi correlati con lo sviluppo di applicazioni distribuite complesse [Papadopoulos98]. Poiché il concetto di coordinazione non viene applicato solamente al campo informatico, ne esistono diverse definizioni, una definizione di carattere generale è la seguente [Malone94]:

La coordinazione gestisce le dipendenze tra le attività.

Una definizione più ingegneristica afferma che [Carriero92]:

La coordinazione è il processo di costruire programmi unendo insieme pezzi attivi.

Di conseguenza in accordo con [Ciancarini97] possiamo considerare un modello di coordinazione come una tripla $<E,L,M,>$, in cui E rappresenta le entità da coordinare, L il mezzo utilizzato per la coordinazione e M il framework semantico a cui il modello aderisce. Un linguaggio per la coordinazione può essere invece definito come il contenitore linguistico di un modello di coordinazione che fornisce gli strumenti per la sincronizzazione, la comunicazione e la creazione e terminazione delle attività computazionali.

A partire da questi concetti generali molti studi sono stati fatti per applicare tali concetti al campo degli agenti, in particolare degli agenti per il web [Omicini01]. Un sistema multi-agente deve infatti definire gli obiettivi individuali dei singoli agenti e gli obiettivi complessivi (sociali) della comunità. L'aspetto sociale viene quindi definito progettando i protocolli e le regole di interazione basati su opportuni modelli di coordinazione, inoltre vengono stabiliti i ruoli degli

agenti nella comunità a cui appartengono definendo (ed implementando) leggi di coordinazione ad-hoc.

I.3.2.2 Comunicazione

Per sfruttare le risorse fornite dall'ambiente l'agente deve poter comunicare con esso. Inoltre l'agente deve comunicare con gli altri agenti in modo da procurarsi informazioni utili al raggiungimento del proprio obiettivo, oppure ancora, per attuare delle forme di collaborazione e cooperazione per il raggiungimento di obiettivi comuni. Il risultato di tali interazioni è un sistema complesso in continua evoluzione il cui stato sarà in generale imprevedibile. Il concetto di interazione tra agenti si riduce ad un problema di comunicazione tra due o più parti.

Il problema della comunicazione porta ad affrontare tre importanti questioni:

- adottare un protocollo di trasporto comune al fine di consentire a richieste e risultati di essere trasmessi;
- concepire un linguaggio di comunicazione comune che entrambe le parti siano in grado di comprendere;
- trovare un accordo sulla lista dei termini che devono poter essere usati nel contenuto dei messaggi e sul significato di tali termini.

Il problema della comunicazione consiste nell'individuare un linguaggio standard per esprimere la struttura e il contesto dei messaggi. Questo problema richiede che gli organi comunicanti si accordino su quali istruzioni, asserzioni, richieste dovranno essere supportate, e quale sintassi utilizzare. Fino ad ora sono stati proposti un certo numero di linguaggi di comunicazione inter-agente. I più noti sono il KMQL (Knowledge Query and Manipulation Language) [Finin94] e il FIPA ACL (Agent Communication Language) [FIPA]. Questi linguaggi sono basati sulla teoria degli atti comunicativi (*speech act*) [Austin62], dove le intenzioni del mittente, ovvero gli effetti del messaggio diretto all'ascoltatore, sono comunicati specificando il tipo di messaggio.

In particolare in questo lavoro di tesi sono state adottate delle semplici politiche di comunicazione basate sul linguaggio presentato da FIPA (FIPA-ACL), al momento lo standard de-facto dei linguaggi di comunicazione per agenti. FIPA è un'associazione internazionale no-profit con l'obiettivo di produrre delle specifiche standard per lo sviluppo di agenti. FIPA si basa su due assunzioni: (i) il tempo per raggiungere uno standard completo non deve essere troppo lungo e non deve agire come un freno per il progresso tecnologico; (ii) solamente il comportamento esterno del sistema deve essere specificato, lasciando agli sviluppatori i dettagli implementavi e la definizione dell'architettura interna dell'agente. Le specifiche FIPA definiscono le regole che consentono agli agenti di interagire. In Figura 6 è mostrato il modello della piattaforma ad agenti proposto da FIPA;

in pratica vengono identificati i ruoli di alcuni agenti necessari a supportare i principali servizi della piattaforma, e specifica il contenuto del linguaggio e l'ontologia. Gli agenti principali evidenziati in tale piattaforma sono l'*Agent Management System* (AMS), il quale ha il compito di supervisione sulla piattaforma ed è inoltre responsabile della registrazione e dell'autenticazione degli agenti. L'*Agent Communication Channel* (ACC) è l'agente che fornisce il supporto per i contatti tra gli agenti (internamente ed esternamente alla piattaforma), l'ACC inoltre supporta l'IIOP per l'interoperabilità tra gli agenti di piattaforme differenti. Infine, come accennato, FIPA definisce un linguaggio standard di comunicazione, l'ACL (Agent Communication Language). La comunicazione tra agenti è basata sul passaggio dei messaggi, in pratica gli agenti comunicano formulando e spedendo messaggi individuali tra di loro. Le specifiche FIPA-ACL definiscono la codifica, la semantica e la pragmatica dei messaggi ACL, lo standard però non specifica un meccanismo per il trasporto interno dei messaggi. D'altro canto, poiché gli agenti possono trovarsi su piattaforme differenti, le specifiche FIPA prevedono che il messaggio venga convertito in formato testuale.

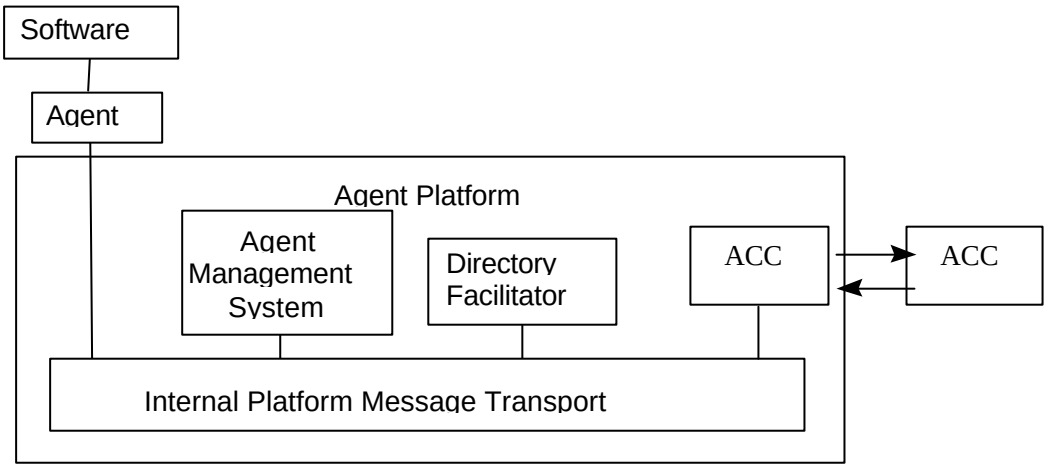


Figura 6 - Architettura FIPA

I.3.2.3 Cooperazione e Competizione

In un sistema in cui coesistono agenti intelligenti di varia natura e dai differenti comportamenti è di primaria importanza disporre di un meccanismo di cooperazione. Tale meccanismo deve facilitare la cooperazione dinamica degli agenti con il fine ultimo di soddisfare sia gli obiettivi locali sia quelli totali della comunità. Ogni agente agisce con l'intento di conseguire determinati risultati (obiettivi locali), la totalità degli agenti opera in accordo con le specifiche del problema prefissato (obiettivi globali). Una struttura di cooperazione di base deve essere in grado di effettuare l'assegnamento dei compiti e la raccolta dei risultati. Ad ogni agente viene affidato un obiettivo, cosicché un insieme di agenti può condividere un obiettivo comune, o più agenti possono entrare in competizione per aggiudicarsi delle risorse comuni. Le caratteristiche di autonomia degli agenti

comportano dunque una vasta gamma di possibili situazioni in cui gli agenti interagiscono tra loro. Le tipiche situazioni in cui è necessario risolvere un problema di cooperazione sono le seguenti:

- un agente deve compiere una scelta tra le sue azioni e tale scelta è determinante ai fini delle prestazioni complessive del sistema;
- l'ordine di esecuzione delle azioni determina le prestazioni;
- il tempo nel quale le azioni sono eseguite determina le prestazioni.

Il problema della cooperazione è in generale molto complesso in quanto un agente avrà difficoltà nel scegliere e temporizzare l'ordine delle sue azioni. Un singolo agente ha una vista incompleta della struttura globale dei compiti alla quale è tenuto a partecipare. Inoltre spesso la struttura dei compiti cambia dinamicamente e l'agente è incerto circa l'esito delle sue azioni.

Molte delle potenziali azioni di un agente sono correlate con quelle degli altri agenti: un agente non è isolato in quanto è incluso in un ambiente (o un insieme di ambienti). Tipici problemi di cooperazione possono essere:

- decidere che cosa fare;
- decidere chi deve fare un determinata azione;
- decidere l'ordine delle azioni e il tempo in cui eseguire le azioni.

Il compito di coordinare un sistema di agenti intelligenti è dunque quello di mediare le esigenze di ogni singolo agente cercando di conseguire gli obiettivi complessivi del sistema. Normalmente il meccanismo di mediazione è affidato ad agenti specializzati che operano nello stesso ambiente: tale meccanismo deve poter contare su degli opportuni protocolli di comunicazione attraverso i quali dare direttive e acquisire informazioni. Nel caso i cui per esempio l'organizzazione di un ambiente complesso può essere decomposto in componenti modulari, tali componenti raggiungono soluzioni locali attraverso una condivisione dell'informazione (formulando dei piani locali).

II COMPORTAMENTO PROATTIVO

Nel capitolo precedente sono state introdotte le principali caratteristiche di cui deve essere dotato un agente per esibire un comportamento flessibile. In questo capitolo, ci concentreremo sugli aspetti relativi al comportamento proattivo, ovvero la capacità di un agente di agire per il raggiungimento di un prefissato obiettivo. Il comportamento proattivo di un agente risulta dunque fortemente legato alle capacità di pianificazione di cui è dotato l'agente.

La nascita della pianificazione si colloca nel contesto degli sforzi, attuati dai ricercatori IA alla fine degli anni '60, di applicare la rappresentazione logica ed i metodi di inferenza alla sintesi di sequenze di azioni, sforzi che avevano dato luogo alla nascita del *situation calculus*. La principale difficoltà dell'utilizzo dell'inferenza era legata al fatto che il ragionamento su sequenze di azioni implica un ragionamento su fatti veri in differenti istanti di tempo. Il formalismo del situation calculus prevede che la definizione di ogni azione venga fatta tramite assiomi: assiomi degli effetti (che descrivono come un'azione cambia il mondo) ed assiomi di frame (che descrivono cosa un'azione lascia invariato nel mondo). Proprio la numerosità di questi ultimi, unitamente ai gravi problemi di efficienza connessi al theorem proving, diedero adito a forti dubbi sulla ragionevolezza di tale metodo, tanto che il problema del frame venne addotto come prova dell'inapplicabilità dei metodi della logica alla sintesi di sequenze di azioni.

La pianificazione logica si propone dunque come un metodo alternativo al situation calculus. Il problema generale della pianificazione può essere riassunto in questo modo:

- dato un ambiente (*environment*) che può assumere diverse configurazioni (*stati*) la cui descrizione si assume data in un linguaggio formale;
- dato un set di azioni disponibili, descritte anch'esse in un linguaggio formale, la cui esecuzione genera un cambiamento di stato dell'environment;
- assegnato lo stato attuale del sistema (*stato iniziale*);
- sintetizzare una sequenza di azioni che, una volta eseguita, conduca dallo stato iniziale a un prefissato stato obiettivo (*goal*).

Sotto ipotesi particolarmente semplificative sull'ambiente e sulle azioni si parla di *pianificazione classica*, modalità con la quale la disciplina si è affermata in principio e che continua a svilupparsi tuttora. Le semplificazioni adottate dalla pianificazione classica però non possono adattarsi alle caratteristiche di un reale ambiente applicativo, per cui al di fuori del ceppo primitivo sono stati sviluppati diversi algoritmi che rilassano una o alcune di tali ipotesi, al fine di rendere la disciplina della pianificazione un efficace strumento di risoluzione di problemi reali.

In questo capitolo verranno dunque affrontate le problematiche inerenti la pianificazione logica, descrivendo dapprima le ipotesi e metodologie della pianificazione classica, passando quindi all'esame dei diversi metodi per l'eliminazione delle semplificazioni.

II.1 Pianificazione Classica

Prima di entrare nel merito della pianificazione classica, è importante focalizzare l'attenzione sull'ambito di validità della pianificazione classica in base alle assunzioni che la caratterizzano:

- *Azioni atomiche*: ogni azione è indivisibile e la sua esecuzione non è interrompibile.
- *Effetti deterministici*: gli effetti di ogni azione sono completamente noti ed interamente descritti nella definizione dell'operatore corrispondente.
- *Accessibilità dell'ambiente*: conoscenza completa dello stato iniziale.
- *Staticità dell'ambiente*: la sola causa di cambiamento del mondo sono le azioni che l'agente stesso vi compie. In questo modo non occorre prevedere cambiamenti di stato inattesi dovuti ad eventi esogeni.

Si intuisce che l'ipotesi fondamentale sottostante è l'assenza totale di incertezza e la completa predicibilità del mondo: il sistema ha una conoscenza completa delle condizioni sotto le quali il piano verrà eseguito, possiede tutte le informazioni necessarie per generarlo, compresa la perfetta conoscenza degli effetti delle azioni a sua disposizione.

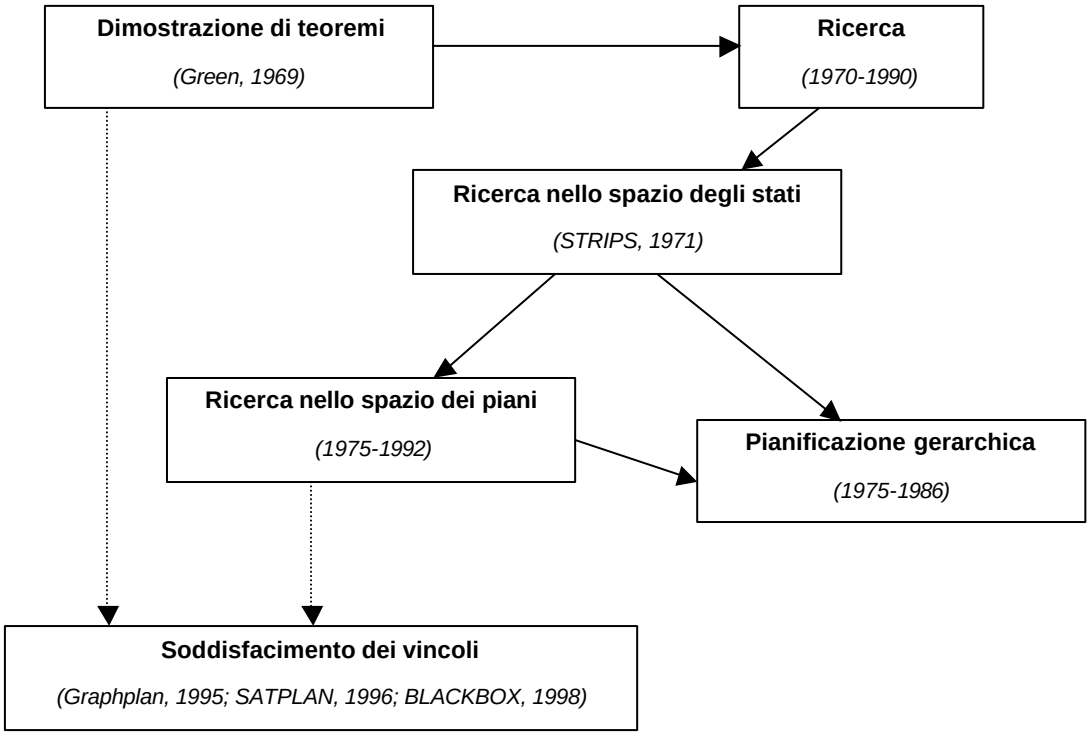


Figura 7 – Cronologia degli approcci alla pianificazione.

In Figura 7 è rappresentata una cronologia degli approcci storici alla pianificazione classica [Kambhampati97].

II.1.1 Ricerca nello spazio degli stati

L'approccio utilizzato nei primi pianificatori logici è molto simile a quello dei problem solver. La procedura di tali pianificatori, infatti, permette di applicare un'azione ad uno stato ed ottenere lo stato risultante, cioè di effettuare l'espansione di un nodo, operazione fondamentale per un algoritmo di ricerca.

Uno dei metodi di pianificazione basato sulla ricerca degli stati è basato su una ricerca in avanti (esattamente come un problem solver); in realtà la rappresentazione degli obiettivi come congiunzione di formule permette di effettuare una *means-end analisys*, ovvero vengono scelte per l'espansione in avanti le azioni i cui effetti sono giudicati rilevanti per il raggiungimento degli obiettivi. Questo metodo non sfrutta la possibilità di applicare una strategia del tipo "*divide et impera*", offerta dall'espressione degli stati e dell'obiettivo come congiunzione di più proprietà, invece che tramite una rappresentazione monolitica.

Un altro metodo, migliore del precedente, è la ricerca all'indietro: si parte dall'espressione del goal e si selezionano le azioni i cui effetti unificano con uno dei suoi congiunti; ciascuna di esse (con un ordine che dipende dalla strategia di ricerca) viene applicata all'indietro, ossia si ricava l'espressione del set di stati a partire dai quali l'esecuzione dell'azione porta ad uno stato compatibile con l'espressione del goal. Il processo si ripete finché l'applicazione all'indietro non restituisce un'espressione compatibile con lo stato iniziale. Si noti che vengono selezionate solo le azioni rilevanti per il raggiungimento degli obiettivi, permettendo così lo sfoltimento (*pruning*) dello spazio di ricerca.

II.1.1.1 Definizione delle azioni mediante il formalismo STRIPS

La sintassi STRIPS fu la prima sintassi dedicata per la descrizione delle azioni [Fikes71]; il suo potere espressivo è quello della logica proposizionale. Con la definizione del formalismo STRIPS il problema della sintesi di sequenze di azioni passa da una formulazione in termini di dimostrazione di teoremi ad una in termini di ricerca.

Secondo tale sintassi un dominio di planning (*problem space*) è formalmente definito come una terna $<L, S, W>$, dove L è un linguaggio proposizionale, S un insieme di stati, W un insieme di azioni.

Uno stato $S \hat{I} S$ è rappresentato da una congiunzione di formule atomiche appartenenti a L, ossia predicati che rappresentano le proprietà del mondo vere in quel particolare stato. L'ipotesi

sottostante è la cosiddetta *ipotesi di mondo chiuso,* secondo la quale se un predicato non è presente nella descrizione di uno stato, si considera vera la sua negazione (ossia, tutto ciò che non è esplicitamente affermato è falso); questa caratteristica, che si è mantenuta anche nelle evoluzioni successive del formalismo STRIPS, configura i sistemi di pianificazione classica come sistemi a due gradi di conoscenza (vero/falso).

Un'azione è definita tramite le sue precondizioni, che ne costituiscono le condizioni di applicabilità, ed i suoi effetti, che specificano i cambiamenti che l'esecuzione di tale azione produce. Formalmente, un'azione $w \hat{I} W$ è definita da una terna $<g_w, a_w, d_w>$, dove g_w è una congiunzione di letterali che deve essere verificata in un qualsiasi stato S_i in cui si voglia applicare w; α_ω è la cosiddetta *add-list*, ossia l'insieme di predicati che l'azione rende veri nello stato risultante S_{i+1}; δ_ω è la *delete-list*, ossia l'insieme di predicati che, veri in S_i, non lo sono più in S_{i+1}.

Dato uno stato S_i ed un'azione ω, posto che in S_i sia verificato g_w, lo stato S_{i+1}, risultante dall'applicazione di w ad S_i è dato dalla seguente *application procedure*:

$$S_{i+1} = S_i \cup a_w \setminus d_w$$

Un' istanza di pianificazione è costituita da:

- uno stato iniziale S_0;
- una descrizione dell'obiettivo, costituito da un'insieme di proprietà (*predicati*);

Il risultato della pianificazione è una lista di azioni che, eseguite in sequenza a partire dallo stato iniziale, conduce ad uno stato finale S_g che soddisfi le proprietà assegnate nell'obiettivo. Si noti che lo stato iniziale è uno stato completo ed univocamente identificato, mentre il goal è la descrizione di un set di proprietà possedute in generale da un insieme di stati.

La fondamentale differenza fra il formalismo di tipo STRIPS ed il situation calculus consiste nel fatto che non occorre dichiarare quel che resta invariato tra S_i ed S_{i+1} per effetto dell'azione ω, devono essere indicati solo i cambiamenti tra uno stato e l'altro, mentre tutto il resto si assume inalterato.

II.1.2 Ricerca nello spazio dei piani

Sacerdoti formulò i principi di un metodo di ricerca in uno spazio alternativo: lo *spazio dei piani* [Sacerdoti74]. In esso un nodo non rappresenta uno stato del mondo ma è costituito da una sequenza di azioni ed un insieme di vincoli su di esse, quali ad esempio i vincoli di ordinamento. Tali vincoli sono in generale parziali, ad esempio, nel caso dei vincoli di ordinamento, essi non definiscono un ordinamento totale della sequenza, per cui si parla in generale di piani parzialmente specificati.

La ricerca passa da un nodo all'altro tramite operazioni di raffinamento, che possono consistere nell'inserimento di un'azione nella sequenza, ma anche nella semplice aggiunta di un vincolo. Si noti che mentre la ricerca nello spazio degli stati restituisce come soluzione la sequenza di azioni (*path*) fra nodo iniziale e nodo obiettivo, la ricerca nello spazio dei piani restituisce direttamente il nodo obiettivo, che è il piano cercato.

La possibilità di procedere nella ricerca tramite il progressivo inserimento e la precisazione di vincoli permette di compiere ad ogni passo solo le scelte che appaiono ragionevoli, rimandando le altre fino al momento in cui non appaia una chiara motivazione per prendere una direzione piuttosto che un'altra. Questo evita di prendere direzioni sbagliate ed essere quindi costretti a tornare indietro. Questo principio è alla base del *least commitment planning* [Weld94] che vede nell'utilizzo dei vincoli il suo strumento più importante e nel *partial order planning*, la sua principale applicazione.

II.1.2.1 Partial Order Planning

Il partial order planning deriva dall'applicazione del principio del least commitment planning all'ordinamento delle azioni nel piano, e nella sua formulazione proposizionale effettua una ricerca nello spazio dei piani parzialmente ordinati. La scelta di inserire un'azione nella sequenza è svincolata dalla scelta della sua precisa posizione all'interno della sequenza. Durante la ricerca vengono progressivamente definiti dei vincoli di ordinamento perché il piano mantenga la sua consistenza. La posizione effettiva delle azioni potrà essere determinata alla fine del processo, in base al soddisfacimento di tutti i vincoli imposti.

La separazione fra la scelta delle azioni e della loro posizione rende possibile l'applicazione al planning del principio "*divide et impera*": per problemi semplici la manipolazione di una descrizione completa dello stato è infatti ragionevole, ma all'aumentare della complessità risulta vantaggioso operare una separazione del problema in sottoproblemi e riunire le soluzioni parziali. Il motivo per cui ciò non è sempre fattibile è che non è possibile trascurare le interazioni fra i sottoproblemi.

Il partial order planning non modifica l'ordine di complessità della pianificazione, che rimane sempre esponenziale, ma si dimostra più efficiente perché riduce il branching factor dello spazio di ricerca. In altre parole, se la complessità della ricerca nello spazio degli stati è $O(b^d)$, dove b è il branching factor e d il numero di passi per giungere alla soluzione, la ricerca ad ordinamento parziale riduce il valore di b, infatti non è necessario considerare contemporaneamente tutte le azioni che hanno come effetto uno qualsiasi dei congiunti, ma si può considerare un congiunto alla volta. Anche se aumenta il valore di d, poiché il numero di passi è pari al numero dei congiunti,

quindi aumenta all'aumentare del numero di precondizioni per azione, l'effetto dominante è comunque il primo.

II.1.2.2 Aumento dell'Espressività di STRIPS: il Linguaggio PDDL

Il formalismo di STRIPS, pur rivelandosi molto più adatto alla definizione delle azioni di quanto non fosse il situation calculus, possiede un potere espressivo pari a quello della logica proposizionale. In particolare, esso non permette l'utilizzo delle variabili nella definizione delle azioni.

Pednault [Pednault89] tentò di colmare il divario tra la rappresentazione di STRIPS e la logica del primo ordine (LPO) definendo l'Action Description Language (ADL), un linguaggio per la descrizione delle azioni con l'espressività della LPO, le cui caratteristiche sono state implementate con gradi differenti di completezza in diversi planners, ma mai totalmente. L'esigenza di utilizzare un linguaggio comune, che permetta di mettere a confronto i diversi algoritmi testandoli su identici problemi, ha dato luogo alla definizione del PDDL (Planning Domain Definition Language) [Ghallab98], un linguaggio per la specifica dei problemi di planning; esso si basa sul linguaggio del pianificatore ad ordinamento parziale UCPOP [Penberthy92], che supporta gran parte dell'ADL. Le principali estensioni supportate dal PDDL, sono:

- *Schemi d'azione con variabili*: con questa funzionalità l'intero set di azioni definite in modo proposizionale si riduce ad un unico schema (*operatore*).
- *Precondizioni disgiuntive*: esse vengono definite per la definizione di azioni che possono essere eseguite sotto differenti, alternative condizioni.
- *Effetti condizionali*: sono effetti che si verificano in dipendenza dal contesto nel quale l'operatore è applicato; un effetto condizionale è subordinato a precondizioni secondarie, le quali non rappresentano condizioni necessarie per l'applicabilità dell'operatore, bensì per il verificarsi di quel particolare effetto.
- *Precondizioni ed effetti quantificati universalmente*: è possibile ad esempio dichiarare, come effetto dello spostamento di un contenitore, che tutti gli oggetti all'interno del contenitore subiranno lo stesso spostamento.
- *Precondizioni quantificate esistenzialmente*: è possibile dichiarare in una precondizione l'esistenza di almeno un oggetto per cui vale quella particolare condizione.

Non sono invece ammessi effetti quantificati esistenzialmente, che implicherebbero la creazione di oggetti, ed effetti contenenti disgiunzioni, perché l'operatore che li contenesse potrebbe avere più effetti alternativi sotto condizioni non determinabili a priori, assumendo quindi caratteristiche di non-determinismo.

II.1.3 Pianificazione Logica e Problem Solving

Il problema della pianificazione classica è identico, come formulazione, a quelli affrontabili con un semplice algoritmo di ricerca.

Planning e problem solving si sono distinti, almeno in una fase iniziale, soprattutto per la rappresentazione di stati e operatori di un problema: il problem solving si serve di una rappresentazione implicita, nella quale ad uno stato è associata una determinata struttura dati (ad esempio una matrice, o una lista) e ad ogni operatore è associata una determinata operazione su tale struttura dati; il planning fa uso invece di una rappresentazione esplicita, in cui stati ed operatori sono rappresentati mediante un linguaggio dalla logica, con la possibilità di diversi e crescenti gradi di espressività.

Da questo punto di vista non c'è una differenza sostanziale fra le due metodologie, o un motivo per trovare preferibile, in generale, l'una all'altra: la rappresentazione esplicita si rivela una buona scelta nei casi in cui non sia immediata la mappatura fra uno stato del problema ed una qualche struttura dati, ad esempio nel caso in cui si debba rappresentare un mondo virtuale di una certa complessità.

Le reali potenzialità della pianificazione logica sono emerse con la possibilità di effettuare un cambiamento dello spazio di ricerca, ovvero di passare da una ricerca nello spazio degli stati ad una ricerca nello spazio dei piani parzialmente specificati, a branching factor inferiore; la generalizzazione di questo approccio ha dato luogo al paradigma del least commitment planning.

Inoltre alla pianificazione va attribuito il merito di aver permesso l'integrazione della risoluzione di problemi con i sistemi basati su rappresentazione esplicita della conoscenza, permettendo l'interfacciamento diretto di un pianificatore con una knowledge base.

II.1.4 Pianificazione come Soddisfacimento dei Vincoli

A metà degli anni '90 la ricerca sul planning ha dato luogo ad una nuova generazione di algoritmi che hanno creato grande entusiasmo, per due motivi: si sono dimostrati molto più veloci degli algoritmi tradizionali in diversi domini di test, ed il principio su cui essi si basano ha poco in comune con la ricerca nello spazio degli stati o dei piani.

Essi sono:

- l'algoritmo GRAPHPLAN;
- algoritmi di pianificazione basati sul soddisfacimento di vincoli (algoritmi di tipo SAT)

L'algoritmo GRAPHPLAN [Blum97] procede in modo iterativo rispetto alla lunghezza della soluzione: a partire dalla lunghezza massima K del piano cercato (un numero intero crescente al

progredire delle iterazioni), l'algoritmo procede in due fasi: *graph expansion* e *solution extraction*. La prima fase (Figura 8) consiste in un'espansione in avanti a partire dallo stato iniziale, applicando tutti gli operatori (totalmente istanziati) eseguibili, finché non viene raggiunta la lunghezza massima scelta o fino a che non viene raggiunta una condizione per cui si può ritenere possibile l'esistenza di una soluzione; nel primo caso si parte con un'altra iterazione dopo aver incrementato K, nel secondo caso si procede con la seconda fase. La fase di solution extraction esegue un *backward-chaining* sul grafo creato dall'espansione, alla ricerca di una soluzione; se la ricerca dà esito negativo, si procede all'iterazione successiva.

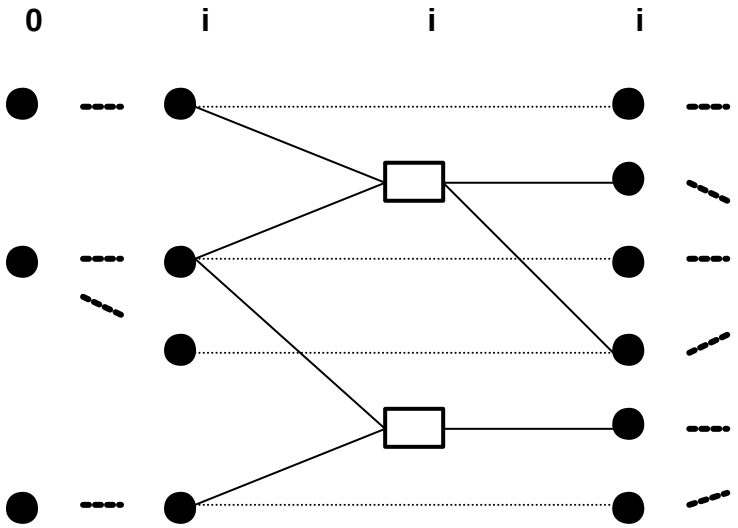


Figura 8 – Grafo dei piani.

Gli algoritmi di tipo SAT si basano sulla preliminare codifica del problema di planning in un problema di soddisfacibilità proposizionale (SAT problem), e quindi nella soluzione di quest'ultimo per mezzo di un *SAT solver*. Anche questi algoritmi, come i precedenti, procedono iterativamente in relazione alla lunghezza della soluzione. L'approccio SAT è stato implementato nel pianificatore SATPLAN [Kautz96].

A partire dai due approcci presentati in questo paragrafo, è stato generato anche un algoritmo che riunisce le caratteristiche di entrambi i sistemi (BLACKBOX [Kautz98]), che procede in questo modo (vedi Figura 9):

- scelta la lunghezza K della soluzione, viene effettuata la graph expansion in modo analogo all'algoritmo GRAPHPLAN;
- il grafo viene codificato in un problema SAT;
- quest'ultimo viene risolto tramite un SAT solver,
- se il SAT solver fornisce una soluzione, questa viene convertita nel piano corrispondente; diversamente si dà luogo all'iterazione successiva.

Per quanto gli ultimi algoritmi creati costituiscano una promessa per il futuro della ricerca AI sul planning, essi attualmente costituiscono una soluzione al solo problema della velocità, mentre è ancora ad uno stadio immaturo il problema dell'aumento dell'espressività delle azioni e la gestione di complessità, dinamicità e conoscenza incompleta. La risoluzione di questi problemi è molto complessa, ed esula dagli scopi di questo lavoro.

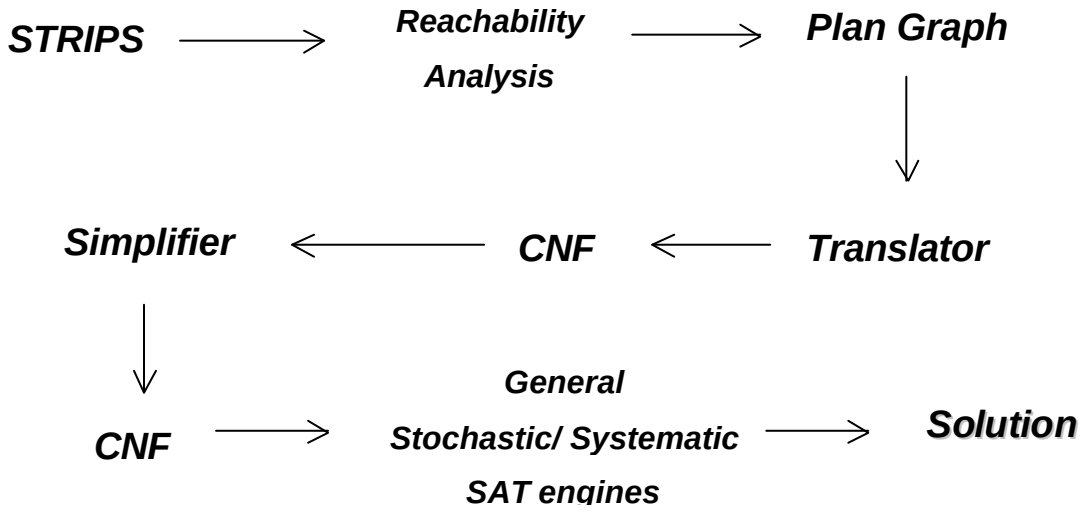


Figura 9 – Algoritmo BLACKBOX.

II.2 Pianificazione in Domini Reali

L'assunzione sottostante la pianificazione classica è che nell'ambiente non vi sia alcun tipo di incertezza: il pianificatore possiede una perfetta conoscenza dello stato dell'ambiente nel quale il proprio piano andrà eseguito, e l'effetto delle proprie azioni gli è completamente noto. Durante l'esecuzione del piano, inoltre, l'ambiente cambierà solo per effetto delle azioni programmate dal planner, e pertanto in modo completamente prevedibile.

Sfortunatamente tali assunzioni, utili come ipotesi semplificative nello sviluppo di algoritmi di planning orientato verso proprietà come correttezza, completezza ed efficienza, si rivelano pesantemente non realistiche e limitanti non appena si voglia applicare lo strumento della pianificazione a contesti che rispecchino, sia pure in maniera semplificata, problemi del mondo reale, come quelli in cui si immagina collocata l'attività di un agente pianificatore.

Nel seguito, in accordo con [Nareyek98], saranno analizzate separatamente le implicazioni che il rilassamento di ciascuna delle ipotesi classiche ha sulle caratteristiche e proprietà dell'ambiente, e descritte le principali metodologie di approccio alle problematiche introdotte; verranno infine presentati alcuni sistemi particolarmente significativi, e la loro collocazione in base alle

problematiche che si propongono di risolvere. La sfida affrontata dai sistemi che descriveremo è quella di estendere l'applicabilità della pianificazione a domini sempre più realistici, rilassando progressivamente tutte le ipotesi della pianificazione classica.

II.2.1 Caratteristiche dei Domini Applicativi

Richiamiamo brevemente le proprietà che comunemente si assumono come principali dimensioni di caratterizzazione di un dominio applicativo.

E' consuetudine classificare le tipologie di ambiente in base alle seguenti proprietà [Russell95][5]:

- *Accessibile/ non accessibile*. Un ambiente è accessibile per un agente se questi può ottenere in ogni istante tutte le informazioni che sono necessarie al corretto svolgimento della sua attività. Si noti che l'attribuzione di questa caratteristica va fatta in relazione all'informazione di interesse per l'agente. Purtroppo questa proprietà viene meno in ambienti anche solo moderatamente complessi: tipicamente gli ambienti applicativi sono solo parzialmente accessibili. Cause frequenti di parziale accessibilità negli ambienti fisici sono la limitatezza del raggio di percezione dell'agente rispetto all'estensione dell'ambiente, e la presenza di ostacoli fisici alla percezione (muri). La parziale accessibilità dell'ambiente viene indicata anche come conoscenza incompleta sullo stato.
- *Statico / dinamico*. Un ambiente è statico per un agente se la sola causa delle sue variazioni è costituita dalle azioni che l'agente vi compie. La staticità viene meno in un ambiente multi-agente (in cui più processi sono contemporaneamente attivi) o comunque in presenza di fattori di evoluzione esogeni all'agente stesso. In generale quindi l'agente non ha il controllo totale dell'ambiente, ma può solo influenzarne i cambiamenti.
- *Deterministico / Non deterministico*. Un ambiente è deterministico dal punto di vista di un agente che vi opera, se questi può sempre prevederne autonomamente l'evoluzione ed i cambiamenti. In base a tale definizione, un ambiente può risultare non-deterministico agli occhi di un agente perché non interamente accessibile (se non si conosce esattamente lo stato in cui ci si trova non se ne possono prevedere i cambiamenti), dinamico (il comportamento di altri agenti ed eventi esogeni rendono imprevedibile lo stato futuro dell'ambiente) o anche qualora l'agente non abbia il controllo e la conoscenza completa delle proprie azioni.
- *Azioni deterministiche / non deterministiche*. L'ipotesi di azioni deterministiche implica che l'agente conosca perfettamente l'effetto delle proprie azioni sull'ambiente. Un'azione è

[5] Le proprietà descritte non intendono dare una classificazione dell'ambiente in termini assoluti, bensì dal punto di vista dell'agente che vi opera.

non-deterministica se il suo modello non ne comprende alcuni effetti, o non specifica le condizioni sotto le quali tali effetti si verificano. Il carattere non-deterministico di un'azione può discendere dall'intrinseco modello probabilistico di essa (si pensi al lancio di una moneta) o dall'incompletezza del modello di essa contenuto nell'agente. In realtà si può pensare di ricondurre queste due circostanze al secondo caso, se si pensa che il modello probabilistico di un'azione deriva dalla mancanza, o dall'eccessiva complessità, di un modello del tipo causa-effetto. Il non-determinismo delle azioni viene indicato anche come conoscenza incompleta sulle azioni.

- *Complessità.* E' consuetudine classificare i domini applicativi in base al loro grado di complessità: esso si può misurare in base al carattere più o meno articolato delle azioni a disposizione, ed al numero complessivo di stati che l'ambiente può assumere.

Secondo la classificazione introdotta, un ambiente che rispetti tutte le ipotesi della pianificazione classica è un ambiente statico, totalmente accessibile, con azioni deterministiche. Come abbiamo già detto, esso non rappresenta un reale ambiente applicativo della pianificazione, ma le ipotesi fortemente semplificative che lo caratterizzano sono essenziali ai fini dello sviluppo, sperimentazione e confronto di nuovi algoritmi corretti e completi, e dei linguaggi di rappresentazione utilizzati.

Anche la nuova generazione di algoritmi basati su espansione tipo GRAPHPLAN e la compilazione dei problemi di planning in problemi di soddisfacibilità proposizionale sono stati sviluppati in tale contesto, sviluppo cui ora sta facendo seguito uno sforzo per estendere il potere espressivo del linguaggio e l'applicabilità di tali algoritmi a problemi di conoscenza incompleta.

Possiamo definire la pianificazione classica come una pianificazione a ciclo aperto, perché non riceve alcun feedback dall'ambiente, che si assume "congelato" durante l'elaborazione; tipicamente alla fase di pianificazione non segue alcuna esecuzione, perché il problema si considera risolto quando il piano è stato generato. La pianificazione classica non si presta alla realizzazione di un agente situato, perché le ipotesi su cui si basa non sono mai verificate nei casi di interesse.

II.2.2 Ambiente Statico ma Parzialmente Accessibile

L'ipotesi che viene qui rilassata è quella di accessibilità dell'ambiente, ossia che, dato un obiettivo da conseguire, l'agente possieda dall'inizio tutte le informazioni necessarie per creare un piano d'azione completo[6].

[6] L'accessibilità dell'ambiente è indicata anche come "onniscienza" dell'agente.

Nella realtà i limiti dell'apparato percettivo dell'agente e/o le caratteristiche dell'environment fanno sì che l'agente non possieda tutte le informazioni che gli sono necessarie; poiché tali informazioni fanno parte dello stato iniziale del task di pianificazione, la non accessibilità dell'ambiente implica l'incompletezza dello stato iniziale, che viene sostituito da un set di stati possibili. Supponendo il permanere dell'ipotesi di ambiente statico, l'informazione, sebbene incompleta, è comunque corretta, nel senso che essa non cambia durante l'elaborazione del piano (non ci sono eventi esogeni).

La conseguenza principale sugli algoritmi è che questi devono pianificare esplicite azioni di recupero dell'informazione (*information gathering*): sarà cioè necessario inserire in un piano azioni che hanno il fine unico di ottenere l'informazione mancante. Tali azioni determinano l'univoca identificazione dello stato iniziale all'interno del set di stati possibili, con la differenza che tale identificazione avviene quando il piano è stato parzialmente o totalmente creato, e parzialmente eseguito.

La pianificazione di contingenza (o condizionale) offre un potente strumento in caso di incompletezza dello stato iniziale; essa consiste nel pianificare anticipatamente alternativi corsi d'azione ("rami" del piano), uno per ogni stato nel set dei possibili stati iniziali, rimandando la decisione su quale alternativa intraprendere al momento in cui si disporrà dell'informazione necessaria per prendere tale decisione. L'effettivo ramo da seguire viene quindi determinato durante l'esecuzione, a seguito di operazioni di information gathering e di successiva decisione [Pryor96]. La pianificazione condizionale si basa su un meccanismo di previsione anticipata, quindi off-line, di tutte le possibilità, la scelta tra le quali andrà fatta durante l'esecuzione del piano, in base al feedback dall'ambiente. Tale feedback non comporta però alcun lavoro di pianificazione aggiuntivo, ma solo la scelta fra possibilità già previste a priori. L'algoritmo di planning non viene più richiamato, dopo aver prestato la sua opera. Per questo motivo si può considerare tale approccio ancora come un meccanismo a ciclo aperto, al pari della pianificazione classica. Sfortunatamente la complessità computazionale rende la pianificazione condizionale troppo lenta per applicazioni che aspirino al real-time. Inoltre, si assume di poter non solo individuare l'informazione mancante, ma anche enumerare tutti i possibili valori che tale informazione può assumere. Se l'informazione mancante consiste nel semplice valore di una o più variabili, da essa non dipende la scelta di azioni differenti, ma al massimo una differente istanziazione dei perametri degli operatori. In questo caso non è necessario costruire un piano ramificato, ma un piano a singolo corso d'esecuzione parametrizzato rispetto alle variabili incognite: queste sono dette variabili a tempo di esecuzione (in contrapposizione alle variabili a tempo di pianificazione) perché il loro valore sarà noto solo dopo l'esecuzione dell'azione di information gathering pianificata.

Un'alternativa alla pianificazione di contingenza è costituita dall'approccio *interleaved planning and execution* [Nourbakhsh97a]: la pianificazione viene interrotta quando il piano è ancora incompleto, lasciando il passo all'esecuzione, e così via in alternanza in un (auspicabile) progressivo avvicinamento all'obiettivo. Il vantaggio è che l'esecuzione precoce permette di acquisire ulteriori informazioni utili nelle successive fasi di pianificazione; inoltre il procedere per segmenti rende il sistema più reattivo ai fallimenti, evitando di lanciarsi in lunghe fasi di pianificazione il cui risultato potrebbe essere invalidato da un cambiamento delle condizioni ambientali [Haigh96b]. Ovviamente il principale tema di discussione sollevato da questo approccio verte sui criteri che possano guidare l'alternanza tra pianificazione ed esecuzione. Ulteriori problemi nascono dalle possibili interazioni fra azioni facenti parte di segmenti diversi: esse possono condurre a piani sub-ottimi o addirittura rendere impossibile il raggiungimento del goal, a meno che non sia verificata la completa reversibilità dalle azioni.

Il rilassamento dell'ipotesi di informazione completa ha delle importanti conseguenze anche sull'organizzazione della conoscenza interna all'agente, conseguenze che si possono riassumere nella caduta dell'ipotesi di mondo chiuso.

II.2.3 Ambiente Accessibile ma Dinamico

L'ipotesi di ambiente dinamico (anche se totalmente accessibile), implica la possibilità che l'informazione posseduta dall'agente, pur corretta e completa ad un dato istante, divenga successivamente inesatta in conseguenza del verificarsi di un evento esogeno non prevedibile né controllabile dall'agente stesso, come l'azione di un altro agente o un episodio di evoluzione spontanea dell'ambiente.

Sebbene l'ambiente sia soggetto a cambiamenti che possono temporaneamente separare lo stato attuale del mondo dalla rappresentazione interna che l'agente ha di esso, si suppone comunque che tali cambiamenti siano interamente percepibili dall'agente, che può quindi aggiornare correttamente le proprie conoscenze in una successiva azione di percezione (conoscenza completa). Sotto tali condizioni è possibile che un piano, già generato e parzialmente eseguito, diventi inadeguato, alcune azioni non possono più essere eseguite; o addirittura inutile, l'obiettivo potrebbe essere già stato raggiunto, o potrebbe essere divenuto impossibile perseguirlo. Condizione fondamentale per poter gestire tali eventualità è la possibilità di eseguire un monitoraggio frequente dell'ambiente durante l'esecuzione.

Una prima soluzione è quella di integrare il pianificatore con un sistema di esecuzione che compia il monitoraggio e sia in grado di identificare le condizioni in cui il piano attuale diviene obsoleto, dando luogo alla generazione di un nuovo piano aggiornato. In questa soluzione il

processo di pianificazione ed esecuzione sono due processi separati che si invocano a vicenda: prima che una fase di esecuzione abbia inizio, deve essere stato generato un piano completo. L'efficienza di questo metodo dipende fortemente dal rapporto fra il periodo medio di variazione dell'ambiente ed il tempo medio necessario alla pianificazione: se tale quantità è inferiore all'unità si ha un'alta probabilità di entrare in un loop senza soluzioni di fatto.

Una soluzione più evoluta è l'integrazione del monitoraggio con l'approccio interleaved, nel quale pianificazione ed esecuzione sono processi interfacciati [Ambros-Ingerson88]: lo stato corrente viene periodicamente aggiornato in base alle percezioni ed alle stesse azioni eseguite dall'agente, ed il piano subisce un processo di adattamento molto più frequente. Questo tipo di pianificazione dà luogo ad un sistema a ciclo chiuso, completamente integrato con l'ambiente.

La soluzione del monitoraggio, detta anche a piano singolo, si basa sulla possibilità di rimediare sul momento agli imprevisti senza prevedere alcun inconveniente in anticipo: il suo limite risiede proprio nell'ipotesi che questo sia sempre possibile. In realtà esistono contingenze che se non previste in anticipo possono portare al fallimento del piano; inoltre le stesse azioni eseguite dall'agente possono condurre a punti di non ritorno, a meno che esse non siano completamente reversibili. L'importanza della previsione anticipata è direttamente proporzionale al costo del fallimento ed alla probabilità di questo. Quando il costo del fallimento è alto, non prevedere in anticipo possibili problemi futuri può essere un errore oneroso. L'eventualità di un cambiamento inaspettato nell'ambiente può essere affrontata anche con la pianificazione di contingenza: in tale caso deve essere possibile prevedere in anticipo tutti i possibili eventi esogeni (che quindi farebbero parte di una categoria di "imprevisti" molto particolare) per pianificare anticipatamente le azioni di recupero che permettano il raggiungimento dell'obiettivo sotto ogni condizione. Si noti la differenza con il precedente utilizzo della pianificazione di contingenza: in questo caso non c'è uno stato iniziale incompleto.

I limiti di tale approccio risiedono evidentemente nella possibilità di prevedere tutte le contingenze in anticipo, e nell'onere computazionale richiesto dalla generazione di un piano ramificato, con una biforcazione per ogni contingenza. Un elemento che può rendere la pianificazione di contingenza preferibile a quella integrata con l'esecuzione è la possibilità di una reazione veloce agli imprevisti: mentre la seconda necessita di un tempo di riadattamento del piano, la prima seleziona immediatamente il ramo dell'esecuzione adatto alla contingenza attuale, a fronte, ovviamente, di un tempo di start-up maggiore.

Dato che in un ambiente reale sono sempre presenti contemporaneamente più fattori che portano ad esigenze contrastanti, è teoricamente interessante la proposta di un approccio ibrido, con una pianificazione anticipata di alcune contingenze (quelle a più alto costo di fallimento o di recupero, o

con maggiori probabilità di accadimento, o per le quali sia necessaria una reazione tempestiva), e che faccia affidamento sulla ripianificazione per il resto. Questa soluzione prevede però che il pianificatore conosca e sia in grado di gestire quantità numeriche come probabilità o costi.

II.2.3.1 Tecniche di Monitoraggio

Il monitoraggio di un piano durante la sua esecuzione si divide fondamentalmente in due tipi di controllo: il controllo sull'applicabilità delle azioni (consiste nella verifica delle precondizioni) ed il controllo sull'utilità delle azioni (consiste nel verificare che non sia stato, per un evento fortuito, già ottenuto un effetto desiderato); il controllo sull'applicabilità si divide ancora in monitoraggio di azione, che si limita al controllo delle azioni prima di mandarle in esecuzione, e monitoraggio di esecuzione, che estende il controllo a tutte le azioni future. In riferimento al piano di Figura 10, ove la linea tratteggiata indica il livello dell'esecuzione, il monitoraggio di azione effettuato prima di

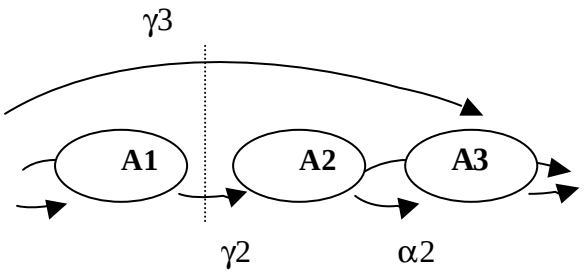


Figura 10 – Monitoraggio d'azione e monitoraggio di esecuzione.

eseguire l'azione A2 può rilevare la mancanza della precondizione γ_2, ma non di γ_3, perché quest'ultima è precondizione di un'azione successiva. Il sistema si renderebbe conto del problema solo prima di eseguire A3. Il controllo sui goals già ottenuti potrebbe invece rilevare che l'effetto α_2 è in realtà già presente nello stato attuale, mettendo in dubbio l'utilità di A2.

II.2.4 Azioni non Deterministiche

In un dominio di planning un'azione è non-deterministica se il suo modello non comprende alcuni effetti rilevanti per il contesto applicativo, o non comprende le condizioni sotto le quali tali effetti si verificano.

Nel primo caso lo schema dell'azione non tiene traccia esplicita dell'incompletezza del modello, gli effetti sono semplicemente ignorati, ed il problema può essere affrontato come si affronta la dinamicità dell'ambiente, con gli effetti imprevisti considerati al pari di eventi esogeni, di cui ci si occupa dopo che sono avvenuti.

Nel secondo caso lo schema dell'azione tiene traccia dell'incompletezza del modello, perché riporta gli effetti in questione, senza però esplicitare le condizioni sotto cui questi si verificano, oppure associa ad essi una probabilità di accadimento. In questo caso è possibile applicare metodi di previsione anticipata, come la pianificazione condizionale o probabilistica. Ovviamente la scelta fra le possibilità va fatta tenendo conto del rischio associato all'approccio a piano singolo, e del costo computazionale associato ai meccanismi di previsione anticipata.

II.3 Pianificazione Gerarchica

In problemi con una dimensione dello spazio degli stati molto grande nemmeno l'utilizzo di euristiche e di algoritmi di planning di ultima generazione può rendere accettabile il costo (in termini di memoria e di tempo) della ricerca. D'altro canto molti contesti applicativi sono dotati di una struttura tale che è spesso possibile individuare gli aspetti fondamentali di un problema, distinguendoli da dettagli trascurabili, tramite un processo di astrazione. Una volta individuata una sequenza di sottoproblemi, è possibile risolvere ciascuno di essi separatamente, con l'aggiunta dei dovuti dettagli, e poi ricongiungere i singoli risultati nella soluzione finale; i vantaggi che derivano da questo processo sono tanto più evidenti quanto maggiore è la dimensione dello spazio di ricerca e la complessità del problema da risolvere.

Lo scopo della pianificazione gerarchica è sostanzialmente quello di trovare una decomposizione del problema nei suoi principali aspetti, che divengono sottoproblemi da risolvere separatamente in un progressivo raffinamento. Un vincolo fondamentale perché questo procedimento abbia successo è l'indipendenza reciproca dei sottoproblemi individuati, mentre la condizione che garantisce l'efficacia nel ridurre la complessità è un'esatta individuazione degli aspetti fondamentali di un problema e di ciò che ne costituisce invece un dettaglio.

Sotto la denominazione pianificatore gerarchico vengono accomunati sistemi che implementano tutti, seppure in modi differenti, un meccanismo di astrazione. E' possibile riunire tali meccanismi sotto tre differenti categorie (si rimanda al capitolo sull'astrazione per un'analisi più dettagliata):

- astrazione nello spazio degli stati
- astrazione sugli operatori

Il primo metodo definisce, a partire dall'analisi dello spazio di partenza $<\Lambda,\Sigma,\Omega>$ (*ground problem space*), un insieme di regole di astrazione che conducono ad uno spazio astratto $<\Lambda',\Sigma',\Omega'>$. Il piano viene dapprima ricavato in quest'ultimo spazio astratto, e poi ricondotto ad una soluzione nello spazio ground tramite operazioni di raffinamento. Definendo delle regole di

astrazione anche sugli spazi astratti si ottiene una gerarchia multilivello. In generale con questo metodo è necessario effettuare anche un'astrazione sul goal.

Il secondo approccio consiste nella definizione di un set di operatori astratti associando ad essi dei metodi di decomposizione in operatori ground; al problema viene prima associato un piano di operatori astratti, ai quali vengono poi applicati i metodi di decomposizione. In questo secondo approccio non c'è un'astrazione sullo stato, ma solo sugli operatori.

Il terzo metodo si può paragonare all'utilizzo delle macro in un linguaggio di programmazione: intere sequenze di operatori dello spazio ground vengono etichettate e memorizzate (macro-operatori); a partire dalla definizione del problema si ricava un piano composto da macro-operatori, quindi questi vengono tradotti nelle sequenze originarie. A differenza dei primi due metodi, il lavoro di pianificazione si conclude una volta ricavato il piano astratto (a parte il processo di espansione delle macro).

II.4 Esempi di Sistemi Esistenti

Nella Figura 6 è stata mostrata una cronologia dell'approccio classico alla pianificazione, a partire da tale figura e in accordo con quanto discusso nel paragrafo precedente, in Tabella 1 vengono mostrati alcuni dei pianificatori più famosi per ciascuna categoria considerata. La tabella mostra anche le caratteristiche del dominio per ciascuno dei casi considerati. Da notare che all'ultima voce della tabella non corrispondono pianificatori, mentre alla penultima voce, al caso gerarchico corrisponde il pianificatore presentato in questo lavoro di tesi (denominato HIPE).

Nel resto del paragrafo verrà analizzato brevemente un pianificatore per ogni categoria, rimandando al paragrafo sulla pianificazione classica per i pianificatori appartenenti a tale categoria e alla seconda parte per il pianificatore HIPE.

II.4.1 Pianificazione a Ciclo Aperto

II.4.1.1 Pianificazione Condizionale

Cassandra [Pryor96] è un pianificatore di contingenza appartenente alla famiglia dei pianificatori SNLP [McAllester91], progettato per operare in domini in cui due dei tre principali vincoli della pianificazione classica sono stati rilassati: il vincolo di azioni deterministiche e di completa conoscenza dello stato iniziale. Questo pianificatore non gestisce quindi eventi esogeni, e gestisce solo un particolare tipo di informazione mancante, presupponendo che tutte le incertezze abbiano outcomes numerabili ed elencabili; le azioni non deterministiche in Cassandra sono azioni con un certo numero di effetti alternativi, trattati come effetti dipendenti dal contesto le cui precondizioni

secondarie siano però ignote. Inoltre, la pianificazione fallisce se anche una sola delle possibili contingenze dà luogo ad un piano impossibile. Infine, è capace di pianificare le azioni di information gethering al pari di qualsiasi altra azione.

Tabella 1 - Classificazione delle strategie di pianificazione, degli approcci.

Pianificatore				***Caratteristiche dell'ambiente***		
Strategia	**Approccio**		**Pianificatore**	**Completamente Accessibile**	**Deterministico**	**Statico**
Ciclo Aperto	Classico	Dimostratore di teoremi	Pianificatore di Green	Sì	Sì	Sì
		Ricerca	*Spazio degli stati:* STRIPS, HSP *Gerarchico:* ABSTRIPS			
			Spazio dei piani: UCPOP, NOAH, Interplan, TWEAK *Gerarchico:* ABTWEAK			
			Spazio dei "task-networks": UMCP, SHOP, SIPE			
		Soddisfacimento dei vincoli	GRAPHPLAN, SATPLAN, BLACKBOX			
	Condizionale		Cassandra, Warplan	No	Sì	Sì
	Probabilistico		Buridan	No	No	Sì
Ciclo Chiuso	Pianificazione ed Esecuzione Integrate		IPEM, Sage, ROUGE, XII *Gerarchico:* HIPE	Sì	Sì	No
	Pianificazione Contingente ed Esecuzione Integrate		--	No	Sì	No

II.4.1.2 Pianificazione Probabilistica

Associare ad un evento caratterizzato da incertezza la sua probabilità significa fare un ulteriore passo avanti nella conoscenza di esso. Come abbiamo già detto, ciò che rende un evento incerto è la non conoscenza dei meccanismi che lo determinano, o l'eccessiva complessità di questi. Un primo passo è quello di associare al fenomeno grandezze statistiche e trattarlo tramite la teoria della probabilità, come nel pianificatore BURIDAN [Kushmerick95]. In questo senso un pianificatore probabilistico è l'evoluzione di un pianificatore di contingenza, in cui ad ogni possibile outcome è associata una probabilità; il piano che viene eseguito è quello con maggior probabilità di successo.

A partire da BURIDAN è stato realizzato anche un sistema ibrido probabilistico-condizionale (C-BURIDAN [Draper94]) in grado di effettuare una pianificazione condizionale servendosi di un modello probabilistico delle azioni.

II.4.2 Pianificazione a Ciclo Chiuso

II.4.2.1 Pianificazione ed Esecuzione Integrate

Pianificazione in Assenza di Information gathering

ROUGE [Haigh96a] è il sistema di planning ed esecuzione del robot Xavier, progettato per operare in ambiente complesso, dinamico, parzialmente accessibile, con azioni non deterministiche[7].

Il problema della conoscenza incompleta viene risolto in ROGUE tramite l'interleaving planning and execution: il pianificatore utilizzato è PRODIGY [Veloso95], un planner classico, non gerarchico, che conduce una ricerca nello spazio degli stati guidata da means-end analisys e backward chaining. Quando si raggiunge l'eseguibilità di alcune azioni che abbiano una ragionevole probabilità di essere rilevanti per il raggiungimento del goal, ROGUE parte con l'esecuzione, durante la quale acquisisce ulteriori informazioni tramite attività sensoria; terminata l'esecuzione del segmento il planner riparte, forte delle nuove conoscenze acquisite. Il problema della conoscenza incompleta viene così risolto in modo "implicito", senza pianificare appositamente per l'information gathering.

Per gestire la dinamicità ROUGE adotta il monitoraggio dei fallimenti con eventuale ripianificazione (quindi l'approccio a piano singolo): è in grado di eseguire un insieme minimo di operazioni di monitoraggio, essenzialmente controllare se gli effetti reali di un'azione coincidono con quelli attesi, determinare l'effetto di un'azione con diversi possibili esiti, ed individuare semplici eventi esogeni.

[7] Le ultime versioni di Rogue includono anche un modulo di learning.

Pianificazione con Information Gathering

Il pianificatore XII [Golden 96] è in grado di operare in un ambiente supposto statico e con azioni deterministiche, ma in condizioni di conoscenza incompleta. XII affronta in maniera esplicita il problema della parziale accessibilità, ossia della conoscenza incompleta dello stato iniziale, servendosi di un'estensione del linguaggio utilizzato dal pianificatore UCPOP per rappresentare ed utilizzare azioni di information gathering, che non cambiano lo stato dell'ambiente, ma la conoscenza che l'agente ha di esso. Condizione fondamentale per gestire esplicitamente la conoscenza incompleta è l'abbandono dell'assunzione di mondo chiuso; la logica utilizzata da XII ha infatti tre gradi di conoscenza: vero, falso, ignoto.

L'esecuzione di un'azione è trattata in XII come uno step del percorso di ricerca, al pari dell'inserimento di un'azione o di un vincolo di ordinamento, ovviamente l'azione deve soddisfare alcuni vincoli di eseguibilità, come quello, evidente, che tutte le sue precondizioni siano verificate nello stato corrente. In realtà la differenza fra l'esecuzione di un'azione ed un'altra operazione di raffinamento è che non sempre è possibile effettuare il backtracking "fisico", con ovvie conseguenze sulla completezza del planner; in assenza di azioni irreversibili però i costruttori di XII considerano sostenibile il rischio teorico di incompletezza.

III COMPORTAMENTO ADATTATIVO

In questo capitolo, ci concentreremo sugli aspetti relativi al comportamento adattativo, ovvero la capacità di un agente di adattarsi all'ambiente in cui è situato, imparando da esso. Appare chiaro che l'aspetto adattativo nel comportamento di un agente è fortemente legato alle capacità di apprendimento di cui un agente è dotato.

Nella sua accezione più generale, il termine *apprendimento* richiama un qualche cambiamento in un sistema che porta ad un miglioramento dello stesso. Una più precisa caratterizzazione di questo aspetto risiede nella seguente definizione [Simon83]:

L'apprendimento denota dei cambiamenti in un sistema che sono adattativi, nel senso che consentono al sistema di eseguire successivamente lo stesso problema o un problema della stessa categoria in modo più efficiente.

La richiesta che un sistema migliori le sue prestazioni grazie all'apprendimento è stata ampiamente accettata. Ci sono però delle attività che possono essere intese come forme di apprendimento in cui il criterio di miglioramento è difficile da applicare.

Il criterio precedente potrebbe dunque essere sostituito da uno più generale che richiede che i cambiamenti siano semplicemente utili: "L'apprendimento è effettuare dei cambiamenti utili nel nostro modo di pensare".

Un approccio migliore alla definizione di apprendimento può nascere dall'osservazione che spesso ciò che conta è l'acquisizione di nuova conoscenza, da immagazzinare sotto qualche forma. Ciò conduce ad una più significativa caratterizzazione dell'apprendimento: "L'apprendimento è la creazione o la modifica della rappresentazione di ciò che si sta sperimentando".

Da questo punto di vista, l'aspetto centrale dell'apprendimento è il processo di costruzione di una rappresentazione di qualche realtà, piuttosto che un semplice miglioramento delle prestazioni. Tuttavia, non sempre ogni atto di apprendimento determina tale conseguenza; anzi, ci sono delle situazioni in cui le prestazioni possono decadere.

La bontà della rappresentazione costruita viene spesso valutata in base a tre criteri fondamentali:

- *Validità.* Si riferisce al grado di accuratezza con cui la rappresentazione copre la realtà; è una sorta di corrispondenza tra la realtà e la sua rappresentazione.
- *Efficienza*. E' un criterio che cerca di catturare l'aspetto delle prestazioni dell'apprendimento e caratterizza l'utilità della rappresentazione ai fini dell'ottenimento di un dato obiettivo: migliore è la rappresentazione, migliori saranno le prestazioni del sistema.

- *Astrazione.* Si focalizza sul dettaglio e la precisione dei concetti utilizzati nella descrizione; in pratica ne definisce il suo potere esplanatorio.

Le varie rappresentazioni possono essere in forma di descrizioni simboliche, algoritmi, modelli di simulazione, procedure di controllo, piani, immagini o teorie formali generali. Un problema fondamentale di qualunque tipo di ricerca sul machine learning, riguarda la forma e il metodo utilizzati per rappresentare e modificare la conoscenza o le abilità che devono essere acquisite.

Esistono fondamentalmente due forme base di apprendimento: acquisizione di conoscenza e raffinamento di abilità.

La prima viene definita come apprendimento di nuova informazione simbolica finalizzata alla sua applicazione in modo efficace. Ad esempio, dire che una persona ha imparato la fisica, significa che tale persona ha acquisito dei concetti, ha compreso il loro significato e le relazioni con tutti gli altri elementi e il mondo fisico. Si tratta quindi di acquisizione di nuova conoscenza, di un modello del sistema fisico e del suo significato che include varie rappresentazioni: da modelli mentali intuitivi, esempi, immagini ad equazioni matematiche e leggi fisiche. Diciamo che la persona ha imparato di più se la sua conoscenza è in grado di spiegare un ventaglio di situazioni più ampio e più accurato ed è in grado di spiegare meglio il significato del mondo fisico.

La seconda forma di apprendimento riguarda il miglioramento graduale delle abilità motorie e cognitive attraverso la pratica, come ad esempio suonare il piano o andare in bicicletta. In questo caso, l'essenza dell'apprendimento consiste nel rifinire le abilità acquisite, sia mentali che motorie, attraverso la pratica continua e correggendo gli scostamenti dal comportamento desiderato. Quest'ultima forma di apprendimento differisce sostanzialmente dalla precedente, soprattutto per il fatto che avviene ad un livello inconscio. Ad ogni modo, la natura umana sembra gestire una mistura delle due attività, con le attività intellettuali che privilegiano la prima e le coordinazioni motorie la seconda.

III.1 Classificazione dei Sistemi di Apprendimento

In letteratura, sono state suggerite diverse dimensioni per classificare e confrontare i campi di investigazione dell'apprendimento, tra le più significative sono da segnalare: strategia di apprendimento utilizzata, rappresentazione della conoscenza acquisita e dominio applicativo. La prima si riferisce al grado di inferenza ottenuto dal sistema sull'informazione disponibile durante l'apprendimento; la seconda riguarda il modo di codificare le informazioni acquisite; la terza, infine, indica l'area su cui viene sperimentato il sistema.

Risultano altresì interessanti anche altri due criteri di classificazione: i paradigmi di ricerca e gli orientamenti di apprendimento. I primi si riferiscono all'approccio utilizzato nella costruzione del sistema; mentre i secondi allo scopo e all'oggetto di studio.

III.1.1 Classificazione Basata sulla Strategia di Apprendimento

Per distinguere le strategie di apprendimento attraverso il grado di inferenza che il sistema introduce sull'informazione fornita, vengono considerati i due estremi: nessuna inferenza ed elevata inferenza. Se un computer è programmato direttamente, la sua conoscenza aumenta ma non effettua nessun tipo di inferenza: tutto lo sforzo è a carico del programmatore. Al contrario, se un sistema scopre da solo nuove teorie o inventa nuovi concetti, esso sta effettuando un sostanziale grado di inferenza, visto che sta derivando e organizzando dall'esperimento e dall'osservazione nuova conoscenza. Una posizione intermedia tra i due estremi potrebbe essere quella di uno studente che cerca di risolvere un problema per analogia verso casi simili svolti in un eserciziario; il processo richiede sicuramente inferenza, ma molto meno rispetto allo scoprire una nuova branca della matematica senza la guida di un insegnante o di un libro.

Appare chiaro che, all'aumentare del grado di inferenza che un sistema di apprendimento è in grado di compiere, si ridurrà il lavoro necessario per l'addestramento del sistema stesso: è più difficile spiegare ad una persona ogni passo di un compito complesso che non mostrargli il modo in cui compiti simili possono essere risolti. Così come è più oneroso programmare un computer che non istruire una persona alla risoluzione di un compito, visto che il primo necessita la precisazione esplicita di tutti i dettagli, mentre l'altra può usare le sue conoscenze e il buon senso per riempire i dettagli meno significativi. Esiste quindi un compromesso tra sforzo richiesto nell'apprendimento e nell'istruzione del sistema, che dà luogo ad una casistica dei sistemi di apprendimento in ordine di complessità:

- *"Rote learning" e impiantazione più o meno diretta della nuova conoscenza*: non è richiesta inferenza né alcuna trasformazione della conoscenza da parte del sistema di apprendimento. L'informazione da parte dell'insegnante è accettata più o meno direttamente e memorizzata dal sistema di apprendimento. In questa categoria ricadono ad esempio lo stile tipico di programmazione o la memorizzazione di dati come nei primitivi database.
- *Apprendimento dall'istruzione*: acquisizione di conoscenza da un insegnante o un'altra sorgente specializzata, come un libro; è richiesta al sistema di apprendimento la trasformazione della conoscenza dal linguaggio di input ad una rappresentazione utilizzabile internamente e che la nuova informazione sia integrata con la conoscenza già presente. E' necessaria una certa inferenza, ma gran parte dello sforzo è a carico comunque dell'insegnante, che deve presentare

ed organizzare la nuova conoscenza in maniera che possa aumentare incrementalmente la conoscenza del sistema. Le trasformazioni principali eseguite dal sistema sono la selezione e la riformulazione (principalmente a livello sintattico) dell'informazione fornita in ingresso.

- *Apprendimento per deduzione*: comprende la riformulazione e compilazione della conoscenza, creazione di macro-operatori, più varie operazioni che preservino la correttezza delle trasformazioni. Se il processo di trasformazione implica generalizzazione dell'informazione in ingresso e selezione del risultato più interessante, cioè inferenza induttiva, parleremo di apprendimento induttivo.
- *Apprendimento per analogia*: è una combinazione tra apprendimento induttivo e deduttivo. Riguarda l'acquisizione di nuovi fatti, o l'abilità mediante trasformazione ed incremento della conoscenza esistente che mostra forti similitudini con i nuovi concetti in una forma utile per la nuova situazione. Ad esempio, una persona che non ha mai guidato un camion ma che sa guidare un'automobile, potrebbe adattare le sue abilità anche se non in modo perfetto al nuovo compito. Analogamente, un sistema *learning-by-analogy* potrebbe essere applicato per convertire un programma per computer già esistente in un altro che esegue una funzione simile ma per cui non era stato progettato. E' dunque richiesta maggiore inferenza da parte del sistema di apprendimento rispetto ai casi precedenti. In definitiva un caso analogo secondo determinati parametri deve essere ripescato dalla memoria, trasformato, applicato alla nuova situazione ed eventualmente conservato per un uso futuro.
- *Apprendimento induttivo*: può ulteriormente suddividersi in apprendimento tramite esempi (o acquisizione di concetti) e apprendimento dall'osservazione e sperimentazione.
  - *Apprendimento tramite esempi*: dato un insieme di esempi e controesempi di un concetto, il sistema induce una descrizione generale che includa tutti gli esempi positivi e nessuno dei negativi. E' una metodologia che è stata ampiamente investigata nell'Intelligenza Artificiale. Il grado di inferenza effettuato è maggiore rispetto all'apprendimento dall'istruzione, dato che non sono forniti dall'istruttore concetti generali. E' in qualche modo superiore anche all'apprendimento per analogia, visto che non sono forniti concetti simili come base di partenza su cui costruire nuovi concetti. Tipicamente questo tipo di apprendimento comprende alcune sottocategorie in base alla sorgente degli esempi forniti e al loro tipo:
    - La sorgente è l'insegnante che conosce il concetto e genera delle sequenze di esempi le più significative possibili. Se si conosce lo stato del sistema è possibile selezionare gli esempi per accelerare la convergenza al concetto desiderato;

- La sorgente è lo stesso sistema, che conosce il suo stato interno ma non il concetto che deve acquisire. Perciò può generare esempi sulla base delle informazioni che ritiene necessarie per poter discriminare tra più alternative nella descrizione del concetto, e lasciar giudicare ad una entità esterna (tipicamente l'istruttore) le istanze positive e negative;
- La sorgente è l'ambiente esterno e il processo generazionale è casuale visto che il sistema deve affidarsi ad osservazioni relativamente incontrollate;
- Sono disponibili solo esempi positivi, così manca l'informazione che impedisce l'eccessiva generalizzazione di un concetto;
- Sono disponibili esempi sia positivi che negativi ed è il caso più tipico, visto che i primi inducono alla generalizzazione, mentre i secondi ne prevengono l'eccesso (il concetto indotto non sarà mai così generale da includere anche gli esempi negativi).

L'apprendimento tramite esempi può inoltre essere di tipo one-trial o incrementale. Nel primo caso tutti gli esempi sono presentati una volta soltanto. Nel secondo caso il sistema costruisce inizialmente una o più ipotesi del concetto consistente con i dati disponibili e rifinisce in seguito le ipotesi dopo aver considerato ulteriori esempi.

- *Apprendimento dall'osservazione e scoperta*: si tratta della forma più generale di apprendimento induttivo che include la scoperta di sistemi, astrazione di teorie, creazione di criteri di classificazione e compiti simili senza l'aiuto di un insegnante esterno. L'inferenza richiesta in questo caso è superiore a quella richiesta in tutti gli altri casi precedentemente illustrati. Al sistema non viene fornito nessun insieme di istanze di un particolare concetto, né un oracolo che possa classificare le istanze di qualsivoglia concetto generate internamente come positive o negative. Inoltre le osservazioni possono interessare più concetti contemporaneamente. Vengono cercate dal sistema delle regole generali che spiegano tutte o la maggior parte delle osservazioni. Questa forma di apprendimento include il *conceptual clustering* (generazione di classi di oggetti descrivibili con semplici concetti), costruzione di classificazioni, riempimento di equazioni con dei dati, scoperta di leggi in grado di spiegare un insieme di osservazioni e formulazione di teorie sul significato di un sistema. Questo tipo di apprendimento può essere ulteriormente categorizzato in base al grado di interazione con l'ambiente esterno. Gli estremi in questa direzione sono due: (i) osservazione passiva, in cui il sistema classifica le osservazioni di aspetti multipli dell'ambiente e (ii) sperimentazione attiva, in cui il sistema introduce delle perturbazioni, in modo casuale o secondo criteri di interesse nell'ambiente esterno per poterne osservare i risultati.

Data la sua importanza, chiariamo ulteriormente il meccanismo con cui avviene l'inferenza induttiva. L'inferenza induttiva parte da un insieme di osservazioni (ed eventualmente delle ipotesi a priori su tali fatti), producendo una generalizzazione che le spieghi. Occorre precisare che la pura induzione, ovvero l'inferenza diretta dai fatti alle teorie senza alcun concetto interpretativo, è impossibile. Sono infatti necessari dei concetti per descrivere le osservazioni e fanno parte della conoscenza di sfondo. Questa conoscenza di base è una componente necessaria per qualunque processo induttivo. Essa include anche obiettivi di apprendimento, vincoli specifici per un dominio, relazioni causali, euristiche e soglie che guidino il processo di generalizzazione, nonché criteri per scegliere tra ipotesi contrastanti.

Si possono distinguere due tipi di tecniche come guida e vincolo nella generalizzazione: tecniche basate sulla similitudine (*similarity-based techniques*) e tecniche basate sui vincoli (*contraint-based techniques*). Le prime esplorano le relazioni tra gli esempi, cioè esempi e contro-esempi di un concetto per poterne creare una descrizione; cercano caratteristiche condivise tra gli esempi della stessa classe individuandone aspetti comuni e spiegando perché esempi differenti appartengono alla stessa classe. Le seconde vincolano i concetti interpretativi o esplanatori applicati ad uno o più esempi in modo che ogni generalizzazione di essi soddisfi tali vincoli. Ad esempio, generalizzare il fatto che una scatola sta sul tavolo dovrà soddisfare il vincolo che tutto ciò che sta sul tavolo non può essere così pesante da romperlo o così grande da non poterci stare. Una importante variante di questo tipo di tecniche è chiamata generalizzazione basata sulla spiegazione (*explanation-based*), che enfatizza il ruolo dalla conoscenza esplanatoria: applica una conoscenza di base in un sistema per formulare spiegazioni concettuali di più alto livello o interpretazioni di un dato evento.

Da notare che le due tecniche sono complementari e possono essere usate simultaneamente nei sistemi di apprendimento.

III.1.2 Classificazione Basata sul Tipo di Conoscenza Acquisita

Un sistema di apprendimento può acquisire regole comportamentali, descrizioni degli oggetti fisici, euristiche, e tanti altri tipi di conoscenza utili per un'ampia varietà di problemi. Descriviamo alcuni casi tipici:

- *Parametri in espressioni algebriche*: in questo contesto l'apprendimento consiste nel regolare parametri numerici o coefficienti nelle espressioni algebriche di una funzione prefissata in modo da ottenere gli effetti desiderati.
- *Alberi di decisione*: alcuni sistemi acquisiscono degli alberi per distinguere tra classi di oggetti. I nodi dell'albero corrispondono alle caratteristiche dell'oggetto e i rami a valori alternativi di tali caratteristiche. Le foglie corrispondono ad insiemi di oggetti con la medesima classificazione.

- *Grammatiche formali*: sono generalmente utilizzate nel riconoscimento di qualche linguaggio artificiale, inducendole da sequenze di espressioni del linguaggio stesso. Tipicamente vengono rappresentate come espressioni regolari, automi a stato finito o regole di trasformazione.
- *Regole di produzione*: si tratta di coppie condizione-azione $<C,A>$ dove C è un insieme di condizioni e A una sequenza di azioni. Se tutte le condizioni di una regola di produzione sono soddisfatte, allora viene eseguita la sequenza di azioni. Per via della loro semplicità di interpretazione, le regole di produzione sono ampiamente utilizzate come rappresentazione di conoscenza nei sistemi di apprendimento. Esse sono acquisite e rifinite secondo quattro operazioni di base: creazione, il sistema costruisce una nuova regola o la acquisisce da una entità esterna; generalizzazione, le condizioni vengono rese meno restrittive per poter applicare la regola in un numero maggiore di situazioni; specializzazione, vengono aggiunte nuove condizioni o quelle esistenti si fanno più restrittive in modo che la regola si possa applicare ad un numero inferiore di situazioni; composizione, due o più regole che sono state applicate in sequenza vengono composte in una regola più grande, formando un processo compilato ed eliminando qualunque condizione o azione ridondante.
- *Espressioni logiche formali e formalismi affini*: sono rappresentazioni che sono state utilizzate per formulare delle descrizioni di oggetti individuali (in input) e per produrre descrizione di concetti risultanti (in output). Hanno la struttura di espressioni logico-formali i cui elementi sono proposizioni, predicati, variabili a valori finiti, vincoli sulle variabili o combinazione di espressioni logiche .
- *Grafi e reti*: in molti domini i grafi e le reti costituiscono una rappresentazione più conveniente ed efficiente rispetto alle espressioni logiche, conservandone in un certo senso il potere espressivo. Alcune tecniche di *graph-matching* e *graph-transformation* consentono di confrontare ed indirizzare la conoscenza in modo molto efficiente.
- *Frames e schemi*: consentono rappresentazioni più estese rispetto alle precedenti. Si presentano sotto forma di collezioni di slot (entità con etichetta), ciascuno dei quali svolge un determinato ruolo nella rappresentazione. Hanno mostrato utilità in molte applicazioni. Ad esempio, un sistema che acquisisca dei piani generalizzati deve essere in grado di rappresentarli e manipolarli come unità, nonostante la loro struttura interna possa essere arbitrariamente complessa.
- *Programmi per computer e altre codifiche procedurali*: l'obiettivo di alcuni sistemi di apprendimento è quello di acquisire l'abilità di portare a termine efficientemente un determinato compito più che ragionare sulla struttura interna del processo. Le codifiche procedurali rappresentano delle abilità pre-compilate, ignorando le loro strutture interne e rendendo disponibile il significato esterno delle abilità acquisite.

- *Tassonomie*: consentono la strutturazione degli oggetti di un dominio secondo gerarchie. Il raggruppamento di descrizioni di oggetti all'interno di nuove categorie distribuite secondo classificazioni gerarchiche richiede al sistema la formulazione di criteri di rilevanza.
- *Rappresentazioni multiple*: alcuni sistemi di acquisizione utilizzano diversi schemi di rappresentazione per la nuova conoscenza acquisita.

III.1.3 Classificazione Basata sul Dominio Applicativo

Un'ulteriore dimensione utile per classificare i sistemi di apprendimento riguarda la loro area applicativa. Tra i vari campi applicativi in cui diversi sistemi sono stati applicati ricordiamo: agricoltura, chimica, modellazione cognitiva (simulazione dei processi di apprendimento umani), programmazione dei computer, educazione, sistemi esperti (programmi di intelligenza artificiale ad alte prestazioni orientati ad un preciso dominio), giochi (scacchi, poker), metodi generali (indipendenti dal dominio), riconoscimento di immagini, matematica, diagnosi medica, musica, elaborazione del linguaggio naturale, caratterizzazione degli oggetti fisici, fisica, pianificazione e risoluzione di problemi, robotica, predizione di sequenze, riconoscimento del parlato.

III.1.4 Classificazione Basata sui Paradigmi di Ricerca

Possiamo distinguere tre grossi filoni di ricerca nell'area dell'apprendimento automatico: modellazione neurale e tecniche teoretiche di decisione; acquisizione di concetti simbolici; e apprendimento basato sulla conoscenza e strettamente connesso al dominio. Questi approcci di ricerca differiscono sostanzialmente nella quantità di conoscenza inserita a priori nel sistema di apprendimento e nel modo in cui questa è rappresentata ed aggiornata internamente.

- *Approccio di modellazione neurale*: tenta di sviluppare dei sistemi di apprendimento di tipo general-purpose che partono da una piccola conoscenza iniziale. Solitamente questa categoria di sistemi vengono definiti come reti neurali o sistemi auto-organizzanti. Consistono in una rete di elementi interconnessi, tipicamente neuroni, che eseguono alcune semplici funzioni logiche, in genere effettuano operazioni di soglia. L'apprendimento è basato sulla modifica incrementale delle intensità di connessione tra i vari elementi, in genere tramite il cambiamento continuo dei pesi associati alle varie connessioni. La conoscenza iniziale del sistema è fornita dalla scelta di elementi di input che rappresentano delle caratteristiche selezionate degli oggetti che si stanno considerando e attraverso la struttura e il valore iniziale dei pesi nella rete. Può trattarsi di una struttura casuale, scelta dall'utente o un mix di entrambe. Un'importante caratteristica di questa categoria di sistemi è l'essenza numerica dei metodi di apprendimento e degli algoritmi, in

contrasto con gli altri paradigmi in cui è posta enfasi sulla creazione e manipolazione di complesse strutture simboliche.

- *Approccio dell'acquisizione di concetti simbolici (SCA)*: prevede che il sistema apprenda mediante la costruzione di una rappresentazione simbolica di un dato insieme di concetti attraverso l'analisi di esempi e controesempi di essi. Le rappresentazioni in genere si presentano sotto forma di un'espressione logica, un albero di decisione, regole di produzione o una rete semantica. Secondo questo paradigma, le caratteristiche rilevanti per un concetto sono fornite al sistema da parte dall'utente.
- *Apprendimento basato sulla conoscenza e strettamente connesso al dominio (KDL)*: il sistema contiene numerosi concetti predefiniti, strutture di conoscenza, vincoli sul dominio, regole euristiche e trasformazioni interne orientate al dominio specifico per cui è stato costruito il sistema. Inizialmente non vengono provati tutti gli attribuiti rilevanti o i concetti; ci si aspetta che il sistema ne derivi di nuovi durante il processo di apprendimento (si parla infatti di costruzione induttiva).

La differenza principale tra i paradigmi KDL e SCA riguarda la quantità e il tipo di conoscenza iniziale fornita al sistema e la ricchezza di strutture di conoscenza da esso generate. I sistemi basati sull'approccio KDL, infatti, sono tipicamente ottimizzati per un dominio specifico e non possono essere direttamente usati in altri domini.

Esistono sistemi che combinano tra loro alcuni aspetti degli approcci precedenti, come ad esempio l'interessante combinazione SCA + KDL, secondo cui vengono combinati meccanismi di apprendimento general-purpose con la possibilità di definire e utilizzare conoscenza specifica per un dominio. Applicare questo sistema ad un dato problema significa poter separare capacità generali di inferenza da conoscenza specifica del dominio, sfruttandone entrambi i vantaggi; ovvero l'applicabilità del sistema ad un vasto insieme di domini differenti pur avvalendosi di conoscenze specifiche durante l'apprendimento.

III.1.5 Classificazione Basata sugli Orientamenti all'Apprendimento

La ricerca nel campo dell'apprendimento automatico si configura secondo tre principali orientamenti tra loro interconnessi:

- *Analisi teorica e sviluppo di algoritmi generali di apprendimento*: prevede l'investigazione sui compiti di apprendimento teorici e cerca di sviluppare degli algoritmi che risolvono tali compiti indipendentemente dall'applicazione. Non c'è alcuna restrizione sul tipo dell'algoritmo introdotto. Non è infatti necessario che sia simile a quello di un uomo a cui fosse chiesto di

risolvere il medesimo problema. I ricercatori in questo campo hanno sempre cercato di studiare lo spazio teorico dei possibili algoritmi di apprendimento.

- *Sviluppo di modelli computazionali dei processi cognitivi umani*: si focalizza sull'apprendimento umano e l'obiettivo finale riguarda teorie computazionali e modelli sperimentali dell'apprendimento umano.
- *Studi sull'apprendimento orientati a specifiche applicazioni*: si sofferma su compiti specifici e pratici di apprendimento e cerca di sviluppare sistemi ingegneristici capaci di risolvere tali compiti. Un esempio potrebbe essere quello di un programma che impara a riconoscere condizioni pericolose per gli aerei in volo. Ogni idea utile dei due orientamenti precedenti viene adottata in questa dimensione. Spesso, quando viene trovata la soluzione ad uno specifico problema, essa viene generalizzata in un metodo in grado di riconoscere una classe di problemi simili.

IV INTEGRAZIONE DEL COMPORTAMENTO PROATTIVO E ADATTATIVO

Il processo di pianificazione, data la sua elevata complessità, ben si presta allo studio del miglioramento delle prestazioni attraverso l'apprendimento inteso, in questo caso, come accumulazione, generalizzazione ed interpretazione di esperienza precedente di pianificazione.

In questo capitolo mostreremo, senza entrare nei dettagli, due approcci proposti in letteratura riguardo l'argomento: il sistema PRODIGY e il sistema SOAR.

IV.1 PRODIGY

L'esempio più illustre di un'architettura in grado di integrare la pianificazione con meccanismi multipli di apprendimento è PRODIGY [Carbonell90]. Tale sistema consiste in un nucleo principale di pianificazione (*general-purpose planner*) e diversi moduli di apprendimento in grado sia di perfezionare la conoscenza del dominio che quella di controllo, al fine di guidare più efficacemente il processo di ricerca delle soluzioni.

La prima realizzazione implementava un pianificatore con decomposizione lineare di subgoaling, cioè senza interleaving, e utilizzava come tecnica di apprendimento una forma di EBL (Explanation Based Learning) applicata alla conoscenza di controllo. Una volta riconosciuto l'impatto positivo sulle prestazioni del pianificatore dell'acquisizione della conoscenza di controllo, l'algoritmo fu perfezionato combinando diversi moduli di apprendimento nell'intento di ridurre il tempo di pianificazione, migliorare la qualità delle soluzioni e rifinire la conoscenza sul dominio.

L'algoritmo di pianificazione di PRODIGY è combinato ad alcuni moduli di apprendimento, progettati per ridurre il tempo di ricerca della soluzione, migliorare la qualità del risultato e raffinare la conoscenza del dominio. Cinque di essi tentano di acquisire conoscenza di controllo sotto forma di *control rules* per poter migliorare l'efficienza del pianificatore e sono:

- EBL introduce delle regole di controllo a partire dalla traccia di un problema. Esaminando lo spazio di ricerca dell'algoritmo di pianificazione, tenta di spiegare le ragioni che hanno consentito la scelta di un percorso rispetto ad un altro durante la ricerca della soluzione.
- STATIC produce delle regole di controllo senza esaminare degli esempi di problemi ma soltanto la descrizione del dominio. E' più veloce ed accurato di EBL ma non è in grado di trovare tutte le regole trovate da EBL, che ha maggiori potenzialità dinamiche. E' basato su una dettagliata teoria predittiva di EBL [Etzioni90].
- DYNAMIC è una combinazione tra STATIC ed EBL.

- ALPINE è un modulo di pianificazione e apprendimento basato sull'astrazione [Knoblock94] che ripartisce il dominio su livelli multipli secondo un modello stratificato. Il pianificatore utilizza la descrizione più astratta del dominio per trovare una traccia della soluzione che viene rifinita tramite l'aggiunta di ulteriori dettagli passando attraverso i livelli.
- ANALOGY risolve un problema utilizzando la conoscenza su problemi simili che sono stati risolti precedentemente, memorizzando i motivi per cui sono state effettuate determinate scelte nello spazio di ricerca e utilizzandoli in seguito per guidare la ricerca delle soluzioni di altri problemi simili.

Tre moduli sono finalizzati a supportare l'utente nello specificare la conoscenza del dominio, sia durante l'acquisizione che il raffinamento dello stesso:

- EXPERIMENT è un modulo del tipo learning-by-experimentation ed è utilizzato quando vengono rilevate delle differenze tra la descrizione del dominio attuale e il significato del mondo reale. E' finalizzato a raffinare la conoscenza sulle caratteristiche del mondo reale più che sull'acquisizione di conoscenza di controllo.
- OBSERVE [Wang94] cerca di accumulare nuova conoscenza sul dominio apprendendo dall'osservazione di agenti esperti e da pratica personale, utilizzando un processo di generalizzazione induttivo (*specific to general*).
- APPRENTICE è un'interfaccia grafica che consente all'utente di giudicare e guidare il sistema durante la pianificazione e l'apprendimento.

Sono state inoltre sviluppate alcune tecniche in grado di migliorare la qualità dei piani prodotti dal pianificatore. Si tratta di strumenti molto importanti per poter utilizzare il pianificatore in applicazioni del mondo reale invece che come semplice strumento di ricerca. E' importante notare che è molto difficile trattare con la qualità dei piani, visto che la teoria del dominio (l'insieme degli operatori) non comprende criteri di qualità. Inoltre, per la maggior parte dei domini, non vale come metrica di qualità la lunghezza dei piani prodotti, visto che gli operatori hanno costo differente.

Sono stati implementati due moduli di apprendimento come due strategie differenti che si occupano dell'aumento della conoscenza di controllo della qualità:

- QUALITY introduce della conoscenza per controllare il processo decisionale in nuove situazioni di pianificazione per generare piani di qualità superiore. Fornisce un controllo sui piani prodotti che consente all'utente di suggerire delle variazioni sui risultati introdotti. Il sistema analizza quindi le differenze tra le sequenze di decisioni che ha intrapreso il pianificatore e quelle che avrebbe dovuto compiere per ottenere piani di qualità migliore.

- HAMLET apprende da una profonda esplorazione delle spazio di ricerca su problemi semplici, dando una spiegazione alle scelte di maggiore qualità e rifinisce la sua conoscenza con l'esperienza.

Il limite fondamentale di tutta l'architettura è la scarsa integrazione tra le parti: tutti i moduli vengono utilizzati con lo stesso algoritmo di pianificazione e sullo stesso dominio.

IV.2 SOAR

Un altro tentativo di combinazione tra capacità di apprendimento e risoluzione dei problemi in un'architettura generale di intelligenza artificiale è il progetto SOAR [Laird87].

La strategia di apprendimento di SOAR è basata sulla tecnica di *chunking* delle sequenze di regole di produzione che è stata sviluppata [Rosenbloom89]. Il chunking è un metodo di apprendimento universale, tale tecnica, inoltre, è particolarmente potente quando combinata con le due architetture USG e UWM [8].

L'architettura di SOAR è basata sull'*ipotesi dello spazio del problema*, il fatto che tutte le attività intelligenti avvengono in un spazio relativo al problema. Quest'idea è incorporata in SOAR permettendo che tutte le decisioni siano fatte nello stesso modo, cioè tramite ricerca nello spazio del problema. In ogni istante SOAR lavora su un *contesto corrente* che descrive lo stato della ricerca in qualunque spazio di problema al momento utilizzato. Più precisamente, il contesto corrente consiste di quattro parti: (i) obiettivo (*goal*), (ii) spazio, (iii) stato e (iv) operatore. Il contesto corrente può essere agganciato a quello precedente in modo da formare una gerarchia di goal e sotto-goal. Le componenti di ciascun contesto vengono annotate con informazione addizionale definita aumenti (*augmentations*). La gerarchia dei contesti e degli aumenti associati genera la *memoria di lavoro* di SOAR.

Per controllare la ricerca negli spazi di problema viene utilizzato un meccanismo apposito. Le regole di produzione contenute nella memoria a lungo termine sono incaricate del compito di decidere quale dei quattro elementi nel contesto corrente dovrebbe essere cambiato e come. Esistono quattro possibili tipi di cambiamento, corrispondenti alle quattro parti del contesto: (i) cambiare l'operatore da applicare allo stato corrente; (ii) cambiare lo stato corrente per espanderlo;

[8] I metodi di problem-solving sono basati sul subgoaling universale (*Universal Sub-Goaling*, USG) e sul metodo debole universale (*Universal Weak Method*, UWM). L'USG è una tecnica che consente di effettuare nello stesso modo tutte le decisioni di controllo durante la ricerca. L'UWM è un'architettura che fornisce tutte le caratteristiche e funzionalità dei metodi deboli [Newell90].

(iii) cambiare lo spazio del problema usato per risolvere il goal corrente; (iv) cambiare l'obiettivo corrente con qualche altro.

Le regole di produzione rendono le decisioni di controllo nella ricerca un processo a due fasi: elaborazione e decisione. Nella prima tutte le regole sono ripetutamente applicate in parallelo alla memoria di lavoro; le regole determinano delle precedenze riguardo a quale parte del contesto debba essere cambiata e in quale modo. Nella seconda si tiene conto delle preferenze per valutare se esiste un'unica scelta migliore delle altre; se quest'ultima può essere determinata, SOAR effettua automaticamente il cambiamento.

Talvolta le regole di produzione non hanno conoscenza a sufficienza per intraprendere una decisione. Questo problema si manifesta quando la fase decisionale non riesce a determinare un'unica scelta riguardante il modo di cambiare il contesto corrente. In questa circostanza si dice che il sistema ha raggiunto un'impasse. Un'impasse è affrontata nella stessa maniera con cui SOAR risolve i problemi, ovvero cercando nello spazio del problema. Quando l'architettura di SOAR si rende conto che è stato raggiunto un'impasse, imposta automaticamente un sotto-goal ed un nuovo contesto per risolverlo. Il sotto-goal *resolve-impasse* è risolto nello stesso modo, scegliendo uno spazio di problema, degli stati e operatori. Durante l'elaborazione del sotto-goal, il sistema dovrebbe accumulare informazione sufficiente per intraprendere la decisione nella ricerca che aveva generato l'impasse; in tal caso il sotto-goal risulta concluso e SOAR ritorna al contesto e goal originali.

Il meccanismo di apprendimento di SOAR intende acquisire conoscenza di controllo di ricerca (*search control knowledge*) dall'esperienza di problem-solving. In particolare, il sistema che effettua il chunking crea nuove regole di produzione che aiutano SOAR a intraprendere più facilmente delle decisioni durante la ricerca. Le nuove regole consentono tali decisioni direttamente attraverso le fasi di elaborazione e decisione descritte prima. Il risultato è che capitano pochi impasse e non si rende necessario processare dei sotto-goal. Il meccanismo di chunking opera continuamente e viene invocato ogniqualvolta termina un subgoal. Tale meccanismo tenta di costruire una nuova regola che sintetizzi i risultati dell'elaborazione del subgoal. Quando capita lo stesso sub-goal in una situazione simile, la nuova regola entra in gioca e aiuta a prendere una decisione che precedentemente conduceva ad un impasse.

Assumendo che un sotto-goal *G* sia terminato con successo, il processo di chunking crea una nuova regola di produzione *R* avente lo stesso effetto dell'intera sequenza delle regole di produzione entrate in gioco durante il processing del goal *G*. Il primo passo consiste nel raccogliere le condizioni e le azioni per la nuova regola *R*. Le condizioni si trovano in una lista (*referenced list*) che è stata costruita durante la ricerca del goal. Tale lista contiene tutti gli elementi di memoria che

sono stati creati prima del goal *G* e che sono stati richiamati da regole entrate in azione durante la ricerca di *G*. Se questi elementi di memoria dovessero presentarsi in altre situazioni, abiliterebbero la stessa sequenza di regole; diventerebbero così le condizioni della nuova regola *R*. Le azioni di *R* si trovano determinando quali elementi di memoria di lavoro sono stati creati durante la ricerca del goal *G* e che sono diventati super-goal di *G* mediante loro attaccamento come aumenti al contesto di un supergoal di *G*. Queste azioni sono l'unica informazione che è stata memorizzata dal sistema dopo la terminazione del goal. Costituiscono l'informazione richiesta per risolvere l'impasse che ha portato alla creazione del goal *G*.

Per applicare la nuova regola *R* ad una varietà di situazioni, alcune delle costanti nelle condizioni e azioni di *R* devono essere generalizzate. In particolare, gli identificatori devono diventare delle variabili. Ciò è necessario dal momento che ciascun identificatore è unico per un elemento di memoria di lavoro. Per poter scegliere le variabili, SOAR deve determinare quali identificatori devono essere uguali agli altri e quali devono essere distinti. La procedura di chunking precedentemente descritta intraprende le decisioni in modo conservativo, portando a condizioni di applicabilità che siano le più ristrettive possibili. Assume che identificatori uguali nell'esempio debbano essere uguali e li sostituisce con un'unica variabile. Assume inoltre che identificatori distinti nell'esempio debbano essere distinti e li sostituisce con variabili differenti. Si aggiunge un ulteriore vincolo perché sia garantito che a variabili distinte corrispondano identificatori distinti. Fatta eccezione per alcuni casi particolari, gli sviluppatori di SOAR garantiscono che questo approccio non determina regole troppo generalizzate. Il passo finale consiste nella creazione di una nuova regola più efficiente riordinando le condizioni ed effettuando altri cambiamenti ai fini dell'efficienza.

Le statistiche [Laird91] mostrano che la tecnica di chunking riduce il numero totale di decisioni di controllo durante la ricerca che il sistema deve fare. Purtroppo il numero di decisioni non è un'appropriata misura delle prestazioni di un sistema. Così questa tecnica migliora le prestazioni generali su alcuni problemi, ma le degrada quando vengono creati dei chunk difficili da far corrispondere. Gli sviluppatori di SOAR hanno verificato che la potenza del meccanismo viene esaltata quando tutte le decisioni sono effettuate nella stessa maniera, mediante ricerca in uno spazio di problemi.

V TECNICHE DI ASTRAZIONE

Come accennato nel paragrafo sulla pianificazione gerarchica, è possibile riunire i meccanismi di astrazione sotto tre differenti categorie:

- astrazione nello spazio degli stati;
- astrazione sugli operatori;
- astrazione tramite macro-operatori.

In questo capitolo analizzeremo tali tecniche e accenneremo alle tecniche esistenti per la generazione automatica di gerarchie di astrazione.

V.1 Astrazione nello Spazio degli Stati

Gli algoritmi che si rifanno a questo tipo di astrazione puntano fondamentalmente all'individuazione di stati intermedi fra stato iniziale ed obiettivo [Knoblock91a]. Il raffinamento successivo consiste nell'individuare un percorso che congiunga fra loro gli stati intermedi. All'interno di questa classe va fatta un'ulteriore divisione fra modelli rilassati e modelli ridotti.

V.1.1 Modelli Rilassati

Questo meccanismo di astrazione è detto anche *ABSTRIPS-style abstraction*, dal pianificatore che per primo la implementò [Sacerdoti74], e consiste nell'eliminare, dalle precondizioni di un operatore ground, quelle ritenute meno critiche: in sostanza, nel rilassare le condizioni di applicabilità dell'operatore stesso. In questo modo si ottiene un modello semplificato (rilassato, appunto) del dominio di partenza. Il problema viene dapprima risolto in questo modello semplificato, ottenendo però un piano che non è ancora eseguibile; il raffinamento a livello ground avviene reintegrando le precondizioni trascurate in ciascuno degli operatori, ed inserendo tramite una nuova sessione di planning gli operatori necessari all'ottenimento di tali precondizioni.

Generalizzando ad N livelli, ad ogni precondizione degli operatori ground viene associato un valore da 1 ad N, che ne rappresenta il livello di priorità; quindi se ω_1 è un operatore di livello ground definito, secondo la sintassi STRIPS, dalla terna $<\gamma_1, \alpha_1, \delta_1>$, e p_{i-1} l'insieme di precondizioni con livello di priorità pari ad i-1, il corrispondente di ω_1 al livello i, ω_1, sarà definito dalla terna $<\gamma_i, \alpha_i,\delta_i>$, dove

$\alpha_i = \alpha_{i-1}$

$\delta_i = \delta_{i-1}$

$\gamma_i = \gamma_{i-1} \setminus p_{i-1}$

Il problema viene risolto dapprima nello spazio astratto *N*; il processo di raffinamento dal livello *i* al livello *i-1* è identico a quello descritto per la gerarchia bilivello. La pianificazione stile ABSTRIPS è spesso indicata come *pianificazione gerarchica lenght-first*, poiché un problema è interamente risolto nello spazio astratto i prima di esserlo nello spazio i-1. ABSTRIPS utilizza la sintassi STRIPS ed effettua una ricerca nello spazio degli stati; l'astrazione basata su modelli rilassati è stata estesa anche ai pianificatori ad ordinamento parziale (in ABTWEAK [Yang96]).

Si noti che l'astrazione non opera sugli effetti degli operatori, che vengono trasportati intatti a qualsiasi livello; inoltre ad uno stesso predicato che compaia come precondizione in diversi operatori possono essere assegnate priorità differenti. La priorità non è quindi funzione del singolo predicato, ma del predicato e dell'operatore in cui compare come precondizione. I modelli rilassati non dividono quindi i predicati del dominio in classi di equivalenza, perché la priorità non è assegnata ad un predicato in sé ma in quanto precondizione in un determinato operatore.

V.1.2 Modelli Ridotti

Questi modelli si basano sull'assegnamento di un livello di priorità (da 1 ad N in una gerarchia di N livelli) ad ogni predicato $\in \Lambda$; viene cioè definita una *partitioned abstraction hierarchy*, che divide l'insieme dei predicati in classi disgiunte [Knoblock94].

Se indichiamo con $<\Lambda_1,\Sigma_1,\Omega_1>$ il problem space di livello ground, con *L(p)* la funzione che associa al predicato p il suo livello di priorità, e con $F_i(s)$ la funzione di filtraggio tale che $F_i(p1, p2,\ldots pm)=\{ pk / L(pk) \geq i \}$lo spazio astratto di livello i è definito dalla terna $<\Lambda_i,\Sigma_i,\Omega_i>$, dove

$\Lambda_i = \Phi_i(\Lambda_1)$

$\Sigma_i = \{\Phi_i(\Sigma), \Sigma \in \Sigma_1\}$

$\Omega_i = \{\omega_i \text{ con } \alpha_i = \Phi_i(\alpha_1), \delta_i = \Phi_i(\delta_1), \gamma_i = \Phi_i(\gamma_1), \omega_1 \in \Omega_1\}$

A differenza dei modelli astratti, il linguaggio del livello *i* è un sottoinsieme di quello del livello *i-1*, ed anche lo spazio degli stati astratto cambia conformazione: se due stati di livello *i-1* differiscono solo per proprietà che non compaiono al livello *i*, essi collassano in un unico stato in tale livello; per questo motivo si parla di modelli ridotti: ad uno stato astratto possono corrispondere uno o più stati dello spazio originario.

Il modello ridotto è stato utilizzato nel sistema ALPINE [Knoblock91b], un sistema dotato di un pianificatore ad ordinamento totale, capace anche di generare autonomamente le proprie gerarchie. In ALPINE l'interpolazione fra gli stati intermedi avviene ricavandoli dall'applicazione degli operatori astratti e proponendoli come obiettivi al livello sottostante; anche il goal di un problema va ripartito ai vari livelli, in relazione alla classe di appartenenza dei suoi congiunti

Si noti che nei modelli ridotti diminuisce non solo il numero di stati, e quindi la lunghezza media dei piani astratti, ma anche il numero di operatori, e quindi il branching factor. Nei modelli rilassati invece quello che cambia non è il numero degli stati ma la connessione fra essi (aumenta la connettività), con la comparsa di scorciatoie che riducono la lunghezza dei piani astratti, sebbene il branching factor in questo caso aumenti.

V.2 Astrazione sugli Operatori

Si basa sulla definizione un set di operatori astratti, ad ognuno dei quali è associato un metodo di decomposizione in termini di operatori più semplici (*action reduction schemata*). Il piano astratto è costituito da una sequenza di operatori astratti, ed il raffinamento consiste nell'espansione di questi. Non è necessario procedere in modalità lenght first, perché l'operazione di raffinamento ha per oggetto il singolo operatore. Proprio la possibilità di un raffinamento non omogeneo può generare dei problemi, perché può portare a trascurare importanti interazioni globali [Knoblock91b]. Uno sviluppo da destra a sinistra ad esempio può far sì che vengano negate in un punto alcune precondizioni su cui si fa affidamento più avanti, in una parte già sviluppata in dettaglio.

Si noti che non vi è alcuna astrazione sugli stati (e quindi sui predicati), ma solo sugli operatori. L'esempio più importante di questo tipo di astrazione è costituito dal Hierarchical Task Network (HTN) planning [Erol94a].

V.2.1 Astrazione di Tipo HTN

Un operatore astratto HTN non deriva dall'astrazione di un operatore ground, ma risponde piuttosto all'idea di task, ossia compito, che coinvolge in generale più azioni.

Gli operatori possono essere di due tipi distinti:

- *Primitive Task.* Sono le azioni elementari del dominio, definite tramite una sintassi precondizioni / effetti stile STRIPS;
- *Compound Task.* Gli operatori astratti sono definiti tramite schemi di decomposizione in termini di primitive task ed altri compound task.

Anche l'obiettivo, finora inteso come set di proprietà che devono essere rese vere, è sostituito dall'espressione di un task, che può essere: (i) un Goal Task: è un obiettivo espresso nella forma convenzionale già nota; (ii) un Compound Task; oppure (iii) una composizione dei due.

Questo modo di concepire gli obiettivi è più flessibile ed espressivo di quello convenzionale [Erol94b]. La pianificazione consiste in un processo iterativo di introduzione di compound e

primitive task, espansione dei compound task e risoluzione dei conflitti tramite critici; il processo termina quando sono presenti solo primitive task.

V.2.2 Astrazione Tramite Macro-Operatori

Come per gli operatori ground, la sintassi di definizione di un operatore astratto è il PDDL. Il generico operatore astratto w_i di livello i è definito quindi da precondizioni ed effetti nel linguaggio L_i: non è escluso l'utilizzo di quantificatori ed effetti condizionali. Un operatore astratto di livello i non deriva dall'astrazione uno o più operatori di livello *i-1*, come accade nei modelli ridotti (anche se questa circostanza è contemplata come caso particolare), ma è il risultato di un ulteriore sforzo di progetto da parte del *knowledge engineer*.

In generale, un operatore astratto deriva dall'astrazione di una o più sequenze di operatori del livello inferiore: sebbene la sua sintassi sia quella di un operatore ground, la sua semantica è quella di un task, in accordo con modelli di astrazione come l'HTN.

In riferimento ad una gerarchia a due livelli, il primo passo è quello di individuare, a partire dallo spazio ground e dal tipo di problemi che si intende formulare in tale contesto, i tasks che si vuole tradurre in operatori astratti. Ad uno stesso task corrispondono, in dipendenza delle circostanze, diversi piani, ossia diverse sequenze di operatori ground. Ad ogni sequenza s possono essere associati precondizioni ed effetti (γ_σ, α_σ, e δ_σ), che si ottengono da precondizioni ed effetti degli operatori ground secondo la seguente relazione ricorsiva: posto $s = w_1 w_2 \ldots w_n = s_1 w_n$, abbiamo:

$$\begin{cases} g_s = g_{s_1} \cup (g_{w_n} \setminus h_{s_1}) \\ a_s = (a_{s_1} \setminus d_{s_1}) \cup a_{w_n} \\ d_s = (d_{s_1} \setminus a_{s_1}) \cup d_{w_n} \end{cases}$$

Ci riferiremo a queste sequenze con il termine di macro-operatori [Korf87]. I macro-operatori che permettono di raggiungere un dato task hanno in comune, tra i loro effetti, quelli da cui è caratterizzato il task in questione. Le loro pre-condizioni saranno diverse, perché essi corrispondono a piani che raggiungono obiettivi uguali sotto condizioni iniziali differenti, anche se in generale sarà presente un'intersezione anche tra gli insiemi delle pre-condizioni. Questo tipo di astrazione è basata sull'aggregazione, infatti in questo modo non si fa alcuna astrazione sui predicati. Una volta che un problema viene risolto nello spazio più astratto, è risolto anche nello spazio originario senza alcun ulteriore processo di pianificazione, ma tramite un'immediata traduzione consistente nel raffinamento di ciascun operatore astratto nella corrispondente sequenza di operatori ground.

Ovviamente il problema di questo approccio sta nel trovare un buon set di macro-operatori, tale da assicurare la copertura desiderata dello spazio degli stati.

E' possibile riunire diversi macro-operatori che garantiscono il raggiungimento di un task in differenti condizioni di partenza, sotto una multi-sequenza che avrà come effetti primari quelli del task, e come pre-condizioni l'unione delle pre-condizioni di tutti i macro-operatori che la compongono. Gli effetti che differiscono da sequenza a sequenza vengono inseriti come effetti condizionali. Ovviamente le pre-condizioni comuni compariranno una sola volta, mentre le altre compariranno in forma disgiuntiva, come alternative. L'operatore astratto associato al task si ottiene effettuando l'astrazione di questa multi-sequenza.

Per definire un operatore astratto è quindi necessario uno sforzo di immaginazione delle possibili sequenze che permettono di raggiungere il task che esso rappresenta. Si noti che la definizione delle pre-condizioni rispetta la semantica delle pre-condizioni di un operatore di livello ground: come queste ultime garantiscono l'eseguibilità di un operatore astratto, perché assicurano l'eseguibilità di almeno una delle sequenze che lo espandono.

V.3 Proprietà Formali delle Gerarchie d'Astrazione

E' possibile caratterizzare i modelli di astrazione in base alle seguenti proprietà formali:

- *Completezza*. Posto che il problema abbia soluzione nello spazio ground, l'algoritmo è in grado di trovarla. La completezza, sia nei modelli rilassati che in quelli ridotti, si basa sull'ipotesi di decomponibilità del problema, ossia sulla possibilità di suddividerlo in sottoproblemi da risolvere indipendentemente. Tale proprietà è prerogativa di una classe di problemi, e solo all'interno di questa classe i pianificatori gerarchici sono completi.
- *Correttezza*. Se l'algoritmo fornisce una soluzione, questa è effettivamente una soluzione del problema posto. La correttezza di un algoritmo gerarchico discende dalla correttezza dell'algoritmo non gerarchico su cui esso si basa.
- *Monotonicità.* Questa proprietà discende dalla definizione di raffinamento monotono. Un raffinamento di una soluzione astratta è monotono se ne lascia inalterata la struttura, ossia se, astraendo la soluzione ground, si riottiene la soluzione astratta. Un raffinamento monotono sfrutta il lavoro effettuato dai livelli superiori, senza sconvolgerlo e rifarlo. Una gerarchia è monotona se per ogni soluzione ground esiste almeno una soluzione astratta il cui raffinamento monotono conduca alla prima. Questa proprietà è utilizzata per effettuare un pruning dello spazio di ricerca senza compromettere la completezza dell'algoritmo: di qualsiasi soluzione astratta si

cercano solo i raffinamenti monotoni, scartando gli altri. Si dimostra che le gerarchie d'astrazione appartenenti ai modelli descritti sono monotone.

- *Upward Solution Property* [Tenenberg88]. L'esistenza di una soluzione nello spazio ground implica l'esistenza di una soluzione astratta. Questo implica che se non si trova soluzione nello spazio astratto si può sospendere la ricerca e concludere che non esiste una soluzione nello spazio ground. Non implica affatto che ogni soluzione astratta trovata sia raffinabile in una soluzione ground; in generale prima di trovare una soluzione astratta "vera", sarà necessario effettuare un certo backtracking fra livelli per scartare le false soluzioni astratte [Giunchiglia90], sia ABSTRIPS che ABTWEAK contengono infatti un meccanismo di backtracking. Si noti che la USP è implicata dalla monotonicità, per cui è verificata nei modelli descritti.
- *Downward Solution Property* [Tenenberg88]. L'esistenza di una soluzione a livello astratto implica l'esistenza di un suo raffinamento a livello ground. Questa proprietà esclude l'esistenza di false soluzioni a livello astratto. E' una proprietà piuttosto rara, perché in genere la perdita di informazione dovuta al processo di astrazione fa apparire possibili soluzioni non praticabili.
- *Downward Refinement Property* [Bacchus94]. Una gerarchia possiede la DRP se, posto che esista una soluzione ground, per ogni soluzione astratta esiste un raffinamento monotono. Se quindi si trova una soluzione astratta non raffinabile in modo monotono, invece di fare backtracking si può concludere che non esiste soluzione ground, perché se esistesse, la soluzione astratta sarebbe raffinabile senza bisogno di backtracking.
- *Monotonicità Ordinata* [Knoblock91b]. E' una proprietà definibile per i modelli ridotti; discende dalla definizione di raffinamento monotonicamente ordinato, che non cambia alcun letterale dei livelli superiori. Una gerarchia è monotonicamente ordinata se ogni raffinamento di un piano astratto è monotonicamente ordinato [Knoblock91c].

V.4 Generazione di Gerarchie di Astrazione

Per quanto diversi sistemi utilizzino la pianificazione gerarchica allo scopo di ridurre il costo della ricerca, il processo di definizione di una buona gerarchia d'astrazione per un dato contesto è tuttora una black art, affidata alla sensibilità del *domain engineer*: non esistono infatti metodologie standard per la creazione di una buona gerarchia di astrazione e il concetto stesso di *buona gerarchia* non è rigorosamente definibile. Una conseguenza di questo è la difficoltà di definire delle valide prove di benchmark per i sistemi di pianificazione gerarchica. Lavori in questo campo sono stati compiuti da [Knoblock91b] che fornisce algoritmi per la generazione automatica di gerarchie di astrazione basate sui modelli ridotti.

Parte II

I AGENTI IN AMBIENTE DINAMICO

Il sistema proposto si pone come obiettivo di dare agli agenti un comportamento autonomo che consenta loro di agire in un ambiente complesso, dinamico e parzialmente accessibile. Il sistema fa parte di un progetto in cui viene simulato un ambiente virtuale in cui gli agenti possono operare su due scenari differenti: il mondo fisico, rappresentazione del mondo reale, e una rete di telecomunicazioni, ispirata ad Internet. Uno scenario di questo genere risulta notevolmente complesso da trattare, data la necessità di gestire interazioni di diversa natura e in numero elevato.

La soluzione proposta è quella di implementare ogni entità autonoma che opera nell'ambiente simulato per mezzo di un agente. Per quanto riguarda la collocazione dell'architettura presentata nello spettro dei sistemi esistenti, essa appartiene alla classe dei sistemi ibridi: utilizza infatti una rappresentazione esplicita del mondo ed il suo cuore è il modulo proattivo totalmente integrato con un modulo adattativo.

I.1 Caratteristiche dell'Ambiente Applicativo

Nel progetto di un sistema ad agenti è di fondamentale importanza l'individuazione delle caratteristiche dell'ambiente in cui esso si troverà ad operare: sono queste caratteristiche, infatti, a determinare le problematiche di progetto e quindi le soluzioni architetturali e tecnologiche più adeguate.

Nel nostro caso l'obiettivo è quello di realizzare un sistema in grado di operare in un ambiente fisico complesso, dinamico e solo parzialmente accessibile, le cui caratteristiche sono:

- *Multi-agente.* La conseguenza principale di ciò è la dinamicità dell'environment, che può cambiare in modo imprevisto agli occhi di ciascun agente, in modo favorevole o sfavorevole ai suoi obiettivi; una seconda conseguenza è la necessità di corredare gli agenti della capacità di interagire. In questo lavoro ci si è limitati all'implementazione di capacità di interazione deboli consistenti nella possibilità per un agente di richiedere o fornire un servizio ad un altro; sono state escluse forme più evolute di collaborazione e competizione.
- *Complesso*. L'ambiente è vasto e vi si trova una quantità di oggetti di diverso tipo che l'agente può e deve utilizzare ai propri scopi.
- *Parzialmente accessibile.* Sono stati considerati i principali fattori che intervengono in ambienti fisici, ossia la limitatezza del raggio di percezione dell'agente rispetto all'estensione dell'ambiente e la presenza di ostacoli fisici alla percezione.

I.2 Fonti di Informazione

E' necessario dedicare uno spazio a parte alle diverse fonti di informazione contemplate nel sistema ed alle loro caratteristiche:

- *Ambiente.* Un primo tipo di informazione è quella derivante dall'attività sensoria che l'agente compie sull'ambiente: le percezioni. Tale informazione va considerata limitata spazialmente e temporalmente: in un ambiente fisico, reale o simulato, il limite spaziale dipende dal raggio d'azione dei sensori. Per acquisire ulteriori informazioni l'agente dovrà quindi spostarsi e per mantenere l'informazione acquisita durante gli spostamenti dovrà possedere una memoria interna.
- *Agenti interagenti.* Informazione proveniente da altri agenti o dall'utente. Può essere richiesta esplicitamente o fornita spontaneamente, da questo punto di vista può comportarsi, rispettivamente, come fonte passiva o attiva rispetto all'agente. Va notato che in generale essa può avere una forma più astratta ed aggregata rispetto alla prima.
- *Fonti esterne*. Informazione di tipo aggregato proveniente da fonti interrogabili, come un database o una mappa. Tale tipo di informazione è di tipo passivo.

I.3 Panoramica delle Problematiche Generali

La prima, evidente difficoltà è dovuta alla complessità del dominio considerato che rende la pianificazione intrattabile se non con l'utilizzo di opportuni metodi di controllo della complessità, come l'astrazione. Inoltre la parziale accessibilità dell'ambiente implica la possibilità che l'agente debba perseguire un obiettivo senza avere inizialmente tutte le informazioni necessarie per farlo.

Come abbiamo già avuto modo di sottolineare, la conoscenza incompleta dello stato iniziale rende impossibile la creazione di un piano completamente specificato sin dall'inizio, perché la necessità di inserire un'azione scaturisce dalla presenza o assenza di condizioni che possono essere inizialmente ignote. Del resto è impossibile, per un agente pianificatore, agire senza aver prima creato un qualche piano. Sappiamo che una possibile soluzione consiste nel generare un piano ramificato che contenga in sé tutte le contingenze e le azioni da compiere in corrispondenza ad esse; l'eventualità dell'utilizzo della pianificazione condizionale è però esclusa nel nostro caso, a causa della sua grave inefficienza. D'altro canto, anche se fosse possibile generare un piano completo prima di dare inizio all'esecuzione, questo potrebbe rivelarsi un lavoro in parte inutile in un ambiente dinamico: un ritmo di variabilità ambientale sufficientemente elevato potrebbe rendere infatti il piano inadeguato prima che sia stata ultimata non solo la sua esecuzione, ma la sua stessa creazione. Ricordiamo che solo sotto l'ipotesi di ambiente statico è possibile separare

completamente la fase di generazione del piano da quella di esecuzione effettiva, perché le condizioni dell'ambiente non cambiano. In ambiente dinamico il gap temporale fra pianificazione ed esecuzione condurrebbe ad un pericoloso sfasamento con la realtà: le condizioni iniziali fornite in ingresso al pianificatore potrebbero cambiare ed esso potrebbe fornire in uscita un piano non più applicabile. In tali casi si ricorre alla pianificazione con monitoraggio: ad una prima fase di pianificazione segue una fase di esecuzione con monitoraggio dell'ambiente ed eventuale riadattamento del piano, si noti che comunque è necessario avere un piano completo prima di dare inizio alla seconda fase. La pianificazione con monitoraggio si rivela però insufficiente se il tempo necessario per l'adattamento del piano è superiore al tempo di variazione dell'ambiente, il processo potrebbe risolversi in un loop senza uscita: questo si verifica in condizioni di alta variabilità o complessità dell'ambiente. La complessità allunga i tempi della pianificazione e della ripianificazione, aumentando così la probabilità che durante tale fase avvengano ulteriori cambiamenti nell'ambiente. Inoltre, quanti più dettagli vengono considerati, tanto più alta è la probabilità che alcuni di questi cambino durante la pianificazione e l'esecuzione, rendendo necessario un riadattamento del piano.

Ciò che in realtà appare inutile e dannoso in ambiente dinamico non è l'utilizzo del monitoraggio e della ripianificazione, che il nostro sistema necessariamente adotta, ma il generare un piano completo di ogni dettaglio prima di dare inizio all'esecuzione. A queste difficoltà si aggiunge la necessità che l'agente cominci ad agire il prima possibile a partire dal momento in cui riceve un incarico, per mantenere i tempi di reazione entro l'intervallo di tollerabilità per l'utente.

Riassumendo, le problematiche principali derivanti dal dominio applicativo preso in considerazione sono le seguenti:

- Necessità dell'utilizzo di una gerarchia di astrazione a causa della complessità ambientale.
- Impossibilità della generazione di un piano completamente specificato prima dell'esecuzione, a causa dell'incompleta conoscenza dello stato iniziale.
- Inutilità della generazione di un piano completamente specificato prima dell'esecuzione, data la variabilità delle condizioni ambientali.
- Inefficienza della generazione di un piano completo prima dell'esecuzione, data l'ipotesi di attività in real-time nell'ambiente di riferimento.

Le argomentazioni riportate tendono a suggerire la soluzione di una pianificazione minima precedente all'esecuzione: un piano definito per linee generali, privo di dettagli (perché sconosciuti o perché soggetti a variazione) e per questo di veloce generazione, che venga progressivamente raffinato nelle parti di imminente esecuzione. Il raffinamento del piano avverrebbe quindi durante

l'esecuzione, guidato dalle azioni di information gathering necessarie, in modo da scegliere i dettagli e le azioni precise in un momento che sia il più vicino possibile alla loro esecuzione, riducendo il rischio della loro obsolescenza. Si noti che questo modo di procedere è perfettamente in linea con i principi del least commitment planning. La pianificazione gerarchica si presta a costituire un valido strumento per questo scopo, grazie al suo sistema di generazione di piani astratti e successivo raffinamento.

A tutte queste problematiche se ne aggiungono delle altre, se si pensa che il raggiungimento di un comportamento adattativo rende necessari studi di significatività sulle sequenze di azioni, affinità tra le stesse, memorizzazione di schemi e varie elaborazioni notevolmente complesse.

Per poter soddisfare i vincoli imposti nella progettazione del sistema, sono state fatte alcune ipotesi semplificative. In particolare:

- *Azioni deterministiche*. Il corredo di operazioni (elementari e non) eseguibili dall'agente ha sempre un effetto ben preciso e prevedibile a priori. L'agente quindi conosce sempre e perfettamente l'esito delle proprie azioni. Possono comunque venire a mancare, in qualunque momento e per diversi motivi, le condizioni che consentono l'applicabilità di un'azione. Inoltre è invariante anche il numero e il tipo delle azioni elementari che l'agente è in grado di eseguire.
- *Variabilità contenuta.* Nonostante sia prevista una certa dinamicità dell'ambiente, è necessario che questa non sia troppo elevata. In altri termini, occorre che il tempo di adattamento del sistema sia inferiore al tempo di variazione dell'ambiente. Ciò comporta, di fatto, un limite superiore alla dinamicità del modello dell'ambiente considerato.
- *Staticità delle entità del dominio*. Non è prevista l'interazione con oggetti sconosciuti, nel senso che non siano già stati classificati secondo tipo e proprietà (informazioni che fanno parte della base di conoscenza dell'agente), né si pretende che l'agente ne studi e scopra le caratteristiche dall'osservazione ed eventuale sperimentazione. Possono naturalmente cambiare attributi connessi alla dinamicità dell'ambiente come posizione e possesso di un determinato oggetto.

Proponiamo in Figura 11 uno schema che riepiloga graficamente le considerazioni fatte sinora. Al centro è posto l'ambiente, modello di una città virtuale, connesso alle sue principali caratteristiche. Sulla frontiera si trovano i requisiti, richiesti al sistema ed esprimibili in termini di vincoli progettuali. Ogni aspetto del mondo genera delle problematiche differenti, per cui vengono proposte e studiate delle possibili soluzioni, ognuna delle quali assume una o più direzioni nel raggiungimento delle specifiche imposte. Infine, si evidenziano le ipotesi semplificative che hanno permesso di affrontare le varie problematiche.

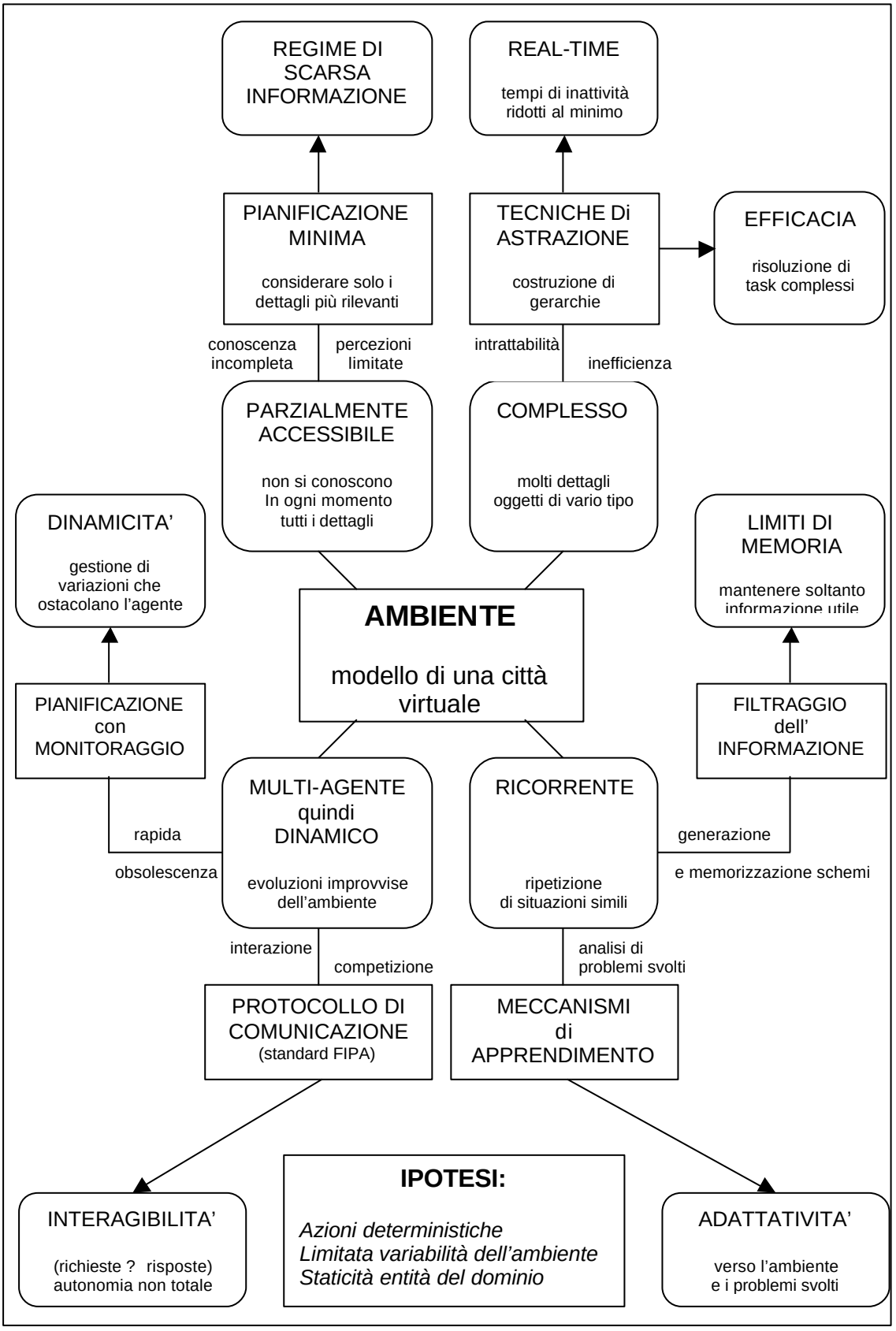


Figura 11 - L'ambiente: caratteristiche, problematiche, ipotesi e soluzioni.

II ARCHITETTURA PROPOSTA

Nella progettazione l'enfasi è stata sicuramente posta sulla microarchitettura, sebbene sia stato sviluppato anche un semplice protocollo di interazione fra agenti che si è servito di una preesistente architettura di comunicazione, conforme alle direttive FIPA.

L'architettura del sistema, di tipo layered verticale, presenta alcune analogie con il sistema INTERRAP [Muller97] ma ha importanti differenze concettuali riguardo alle funzionalità di ciascun singolo layer. A questo proposito una peculiarità del sistema, importante differenza rispetto ad INTERRAP, è la completa omogeneità dei vari livelli: questo dà luogo alla totale scalabilità dell'architettura e rende inoltre possibile descrivere l'intero sistema mediante la descrizione del singolo livello e dei protocolli di interazione fra esso ed i livelli adiacenti. Inoltre, contrariamente alle principali architetture che si ispirano al practical reasoning, non si basa su una libreria di piani definita dal progettista, ma è concepita per generare autonomamente i propri schemi d'azione.

La soluzione utilizzata per superare i limiti comuni alle architetture deliberative e rendere il sistema adatto ad un ambiente dinamico in regime di informazione incompleta è l'utilizzo sinergico di pianificazione gerarchica ed interleaved planning and execution (HIPE). Entrambi i meccanismi si appoggiano alla struttura intrinsecamente gerarchica dell'architettura verticale: in altre parole, il superamento delle ipotesi classiche si appoggia su una soluzione architetturale, più che algoritmica (come invece accade nei sistemi descritti in precedenza).

II.1 Schema dell'Architettura

Come detto nell'introduzione si tratta di un'architettura di tipo layered, verticale. E' costituita da tre livelli, come si può vedere nello schema di Figura 12, ma come già detto si può facilmente generalizzare ad N livelli all'aumentare della complessità ambientale [Armano00b].

In accordo alle architetture verticali, anche la conoscenza è stratificata: ogni livello possiede una propria base di conoscenza (KB), contenente una rappresentazione del mondo e dello stato interno dell'agente adeguata alla funzione cui tale layer è designato. La relazione tra i linguaggi utilizzati nelle diverse rappresentazioni è di astrazione, secondo il modello che verrà definito nel prossimo capitolo. L'attributo verticale indica che ogni livello ha il controllo sul livello sottostante, e la coerenza del comportamento emergente deriva dall'interazione tra livelli nel rispetto delle regole della gerarchia: il flusso del controllo, inteso come invocazione dei servizi di un livello da parte di un livello adiacente, avviene dall'alto verso il basso.

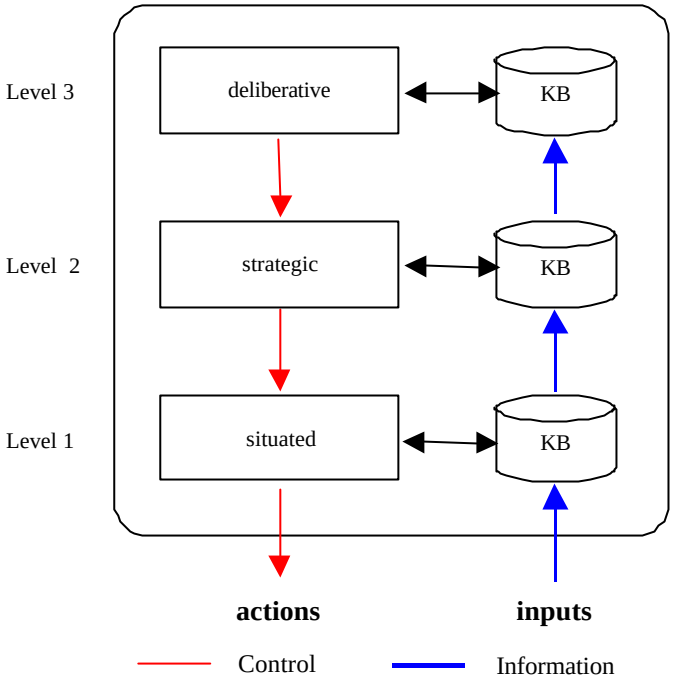


Figura 12 - Schema generale dell'architettura.

Per quanto riguarda il flusso dell'informazione, il meccanismo è un po' più articolato: l'informazione proveniente dall'environment fa il suo ingresso nel sistema dal livello più basso, e viene distribuita alle KB dei vari livelli secondo un percorso che dipende dal livello di appartenenza dell'informazione; supponendo che sia di primo livello, il flusso sarà diretto verso l'alto, diversamente il percorso sarà più complesso. Per questo motivo l'architettura non può essere definita "two pass": tale espressione si riferisce infatti a due opposte direzioni di flusso del controllo e dell'informazione, la prima verso il basso, la seconda verso l'alto.

Il primo livello è il punto di contatto con il mondo: riceve le percezioni, i messaggi ed esegue le azioni elementari; il secondo livello gestisce la strategia d'azione, ossia si occupa di un tipo di ragionamento più astratto cui fa seguito l'azione; il terzo livello è quello in cui risiedono le facoltà deliberative e di comprensione del contesto più alte dell'agente.

Contrariamente alle architetture layered presenti in letteratura, nelle quali ogni livello realizza una precisa caratteristica comportamentale che si vuole presente nell'agente (comportamento reattivo, proattivo, capacità sociale), ed ha una struttura diversa dagli altri, nell'architettura proposta i livelli sono identici: tutti possiedono funzionalità reattive, proattive e deliberative, ma le esercitano a differenti gradi d'astrazione. Le diverse caratteristiche comportamentali scaturiscono piuttosto dalle interazioni verticali fra livelli, ossia dallo svolgimento congiunto, disciplinato da una precisa gerarchia, di funzionalità analoghe presenti in tutti i livelli [Armano01a].

Nei paragrafi seguenti entreremo maggiormente nel dettaglio della struttura dell'architettura evidenziando le interazioni verticali e orizzontali presenti tra e intra i livelli (vedi Figure 13 e 14).

II.1.1 Interazioni Verticali

- *Comportamento Reattivo.* Tutti gli eventi provenienti dall'ambiente circostante e dalle interazioni con gli altri agenti fluiscono dal livello ground al livello più alto nella gerarchia di astrazione attraverso il modulo reattivo. Dal punto di vista dell'informazione scambiata, il ruolo del modulo reattivo è quello di adattare ciascun evento con la tipologia di informazione contenuta dalla KB in quel determinato livello. In particolare, ciascun evento ha bisogno di essere opportunamente codificato a ciascun livello K > 0, dato che l'informazione contenuta da questi livelli è più astratta di quella contenuta nei livelli precedenti.

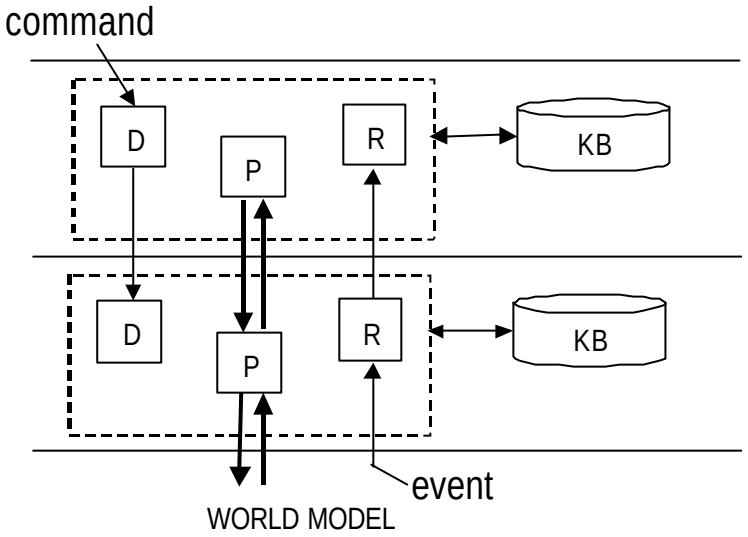


Figura 13 - Interazioni Verticali.

- *Comportamento Proattivo.* Come verrà mostrato meglio nei prossimi capitoli, il comportamento proattivo è supportato da un pianificatore gerarchico distribuito su N livelli, ciascuno dei quali destinato a lavorare a differenti livelli di granularità. Tra i pianificatori dei diversi livelli esiste una comunicazione bidirezionale: un pianificatore a livello K > 0 può ordinare a quello a livello sottostante di raffinare un operatore astratto passandogli una tripla <*pre*, *oper*, *post*> indicante rispettivamente pre-condizioni, operatore e post-condizioni. In realtà solamente le post-condizioni (opportunamente tradotte in base alla granularità del livello considerato) sono usate per instanziare un nuovo goal (locale al pianificatore considerato). D'altra parte, per eseguire le azioni ai livelli sottostanti, il pianificatore di livello ground usa le informazioni contenute nell'oerpatore: un pianificatore a livello K < N-1 può informare il pianificatore del livello sovrastante che un piano è stato eseguito con successo o che è avvenuto un fallimento. A questo proposito va sottolineato che attualmente non è previsto di propagare le informazioni sulle cause del fallimento.

- *Comportamento Deliberativo.* Ciascun comando, se accettato dall'agente, fluisce dal livello più astratto verso il livello ground attraverso il modulo deliberativo. In questo modo si consente all'utente di mandare comandi all'agente attraverso un canale preferenziale. Da un punto di vista delle strategie di controllo, il ruolo del modulo deliberativo è quello di propagare i comandi esterni verso il basso sino a trovare il livello in grado di interpretarlo. In questo modo inoltre i livelli superiori sono in grado di inibire determinati controlli ai livelli sottostanti che, dunque, non possono prendere delle decisioni in merito.

II.1.2 Interazioni Orizzontali

- *Comportamento Reattivo.* Dato un livello, il suo modulo reattivo contiene un set parzialmente ordinato di regole, ciascuna delle quali nella forma <*pre*, *post*, *priority*>. Una regola ha dunque associata una priorità (compresa nel range 1-10) e può essere applicata quando le sue pre-condizioni sono verificate. Il compito principale del modulo reattivo è di verificare se e quando certe regole esistenti possono essere applicate, in base ai cambiamenti ambientali causati dagli eventi esterni. Se almeno una regola può essere applicata, il modulo reattivo invia un comando di interruzione (*break*) al pianificatore e informa il modulo deliberativo (*break-info*). Se il modulo deliberativo accetta il comando di interruzione, le post-condizioni della regola che ha sollevato l'interruzione diventano un nuovo goal da raggiungere, goal che viene dato in input al modulo di pianificazione che aveva precedentemente interrotto il piano corrente. Al raggiungimento degli obiettivi posti dalla regola, verrà ripristinato il piano precedentemente interrotto.

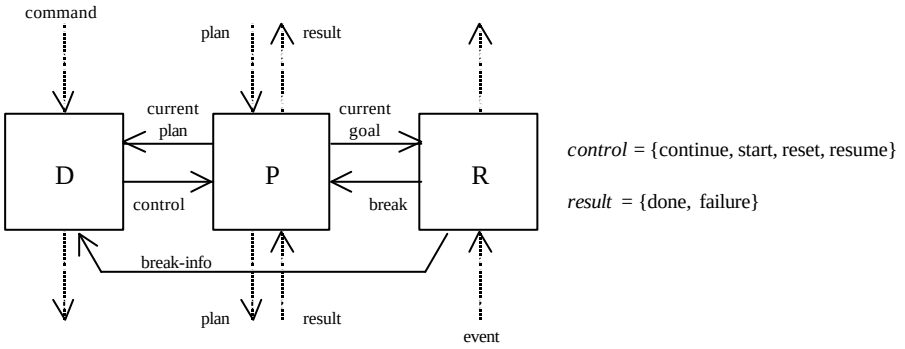


Figura 14 - Interazioni orizzontali.

- *Comportamento Proattivo.* Dato un livello, il comportamento proattivo può sottostare a costanti interazioni con i moduli reattivo e deliberativo. Infatti, il modulo proattivo può ricevere un comando *break* dal modulo reattivo che causa l'interruzione dell'attività che stava svolgendo in quel momento in attesa di una decisione del modulo deliberativo. Dopo la ricezione del comando di interruzione, il pianificatore locale informa il modulo deliberativo riguardo il piano corrente (*current plan*) in modo da aiutarlo nelle decisioni da prendere. A seconda della decisione presa

dal modulo deliberativo (una delle quattro appartenenti al set di controllo), il pianificatore può continuare la propria attività (*continue*), iniziare un nuovo piano (*start*), rimanere in attesa (*reset*) o riprendere il piano precedentemente salvato (*resume*). Si può notare che per come sono state progettate le interazioni verticali tra i pianificatori locali, l'attività di pianificazione può ripartire anche dopo la ricezione di un fallimento da un pianificatore di livello inferiore. Infine, il pianificatore va in stato di attesa (*reset*) al completamento dell'attività di pianificazione.

- *Comportamento Deliberativo.* Nell'attuale sistema, il modulo deliberativo non svolge alcuna complessa attività di ragionamento e interagisce con il pianificatore inviando dei comandi del tipo: {*continue*, *start*, *reset*, *resume*} come descritto sopra. In particolare, quando deve essere attivato un nuovo piano il modulo deliberativo agisce in base alla seguente politica: dopo aver ricevuto dal modulo reattivo l'insieme delle regole attuabili, viene selezionata la regola con la priorità più alta (nel caso di regole di pari priorità viene fatta una scelta casuale). Dopodiché, viene spedito un messaggio al pianificatore per iniziare una nuova attività di pianificazione segnalandogli che la priorità di tale regola è maggiore di quella del piano che sta attualmente eseguendo.

III RUOLO DELL'ASTRAZIONE

Nell'ambito della pianificazione classica l'astrazione viene considerata come il metodo più efficace per ridurre la complessità computazionale in domini con vasti spazi di ricerca, e per ottenere un maggiore potere espressivo nella definizione di azioni, stati ed obiettivi.

Abbiamo visto che le principali tecniche di cui la pianificazione gerarchica si serve consistono nel ragionare prima in termini di aggregati di azioni (operatori astratti o macro-operatori) oppure considerare in prima istanza solo le pre-condizioni più critiche, e procedere successivamente ad un raffinamento incrementale dei piani, tramite l'espansione degli operatori astratti o l'introduzione di azioni per il soddisfacimento delle pre-condizioni secondarie.

Da questo punto di vista la progettazione di sistemi ad agenti non si discosta, almeno nella sua corrente principale, dalla prospettiva della pianificazione classica: anche nelle architetture per agenti situati ci si serve dell'astrazione essenzialmente per ridurre la complessità computazionale della pianificazione in domini complessi e con vasti spazi di ricerca quali sono i problemi nei domini reali. Un ulteriore, importante vantaggio è costituito dalla possibilità di dotare l'agente di distinte, coesistenti prospettive sul mondo: da quella fornita dalle semplici percezioni, di cui spesso si serve il sistema reattivo, a prospettive fatte di concetti derivati, su sé stesso e sugli altri agenti, che permettono comportamenti complessi e l'esecuzione di piani che coinvolgono più agenti (come nelle architetture layered).

Posta la necessità dell'utilizzo della pianificazione gerarchica per motivi di intrattabilità, al pari di altri sistemi, nel sistema proposto i vantaggi ottenuti dall'astrazione non si esauriscono nella risposta all'esigenza di dominare la complessità ambientale in ambito statico (e quindi classico); si fa piuttosto dell'astrazione lo strumento fondamentale per gestire i problemi di dinamicità e conoscenza incompleta (al di fuori, quindi, dell'ambito classico).

III.1 Concetti Generali

Come già detto, il nostro obiettivo è utilizzare l'astrazione per permettere all'agente di cominciare ad agire anche senza avere un'idea precisa di tutte le azioni che dovrà eseguire. Ovviamente questo discorso ha dei limiti: ciò che si può trascurare all'inizio sono appunto i dettagli, non la struttura principale del piano, che deve essere ben definita.

Riferendoci ad un esempio di robot-planning task, non si può pretendere che l'agente prenda un oggetto da qualche parte senza aver fornito il luogo in cui esso si trova: su questa informazione non si può transigere, perché la sua mancanza mina la struttura portante del piano. Si può però

desiderare che l'agente cominci a muoversi verso il luogo designato senza conoscere le azioni esatte che dovrà compiere per procurarsi l'oggetto (aprire delle porte, cercare all'interno di una stanza).

Per svolgere il task assegnato, l'agente potrebbe generare come prima cosa un piano astratto, decomponendo il problema in due task principali, come mostrato in Figura 15.

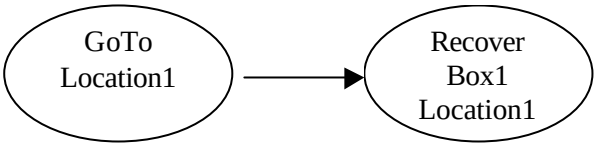


Figura 15 - Piano astratto

E' evidente che non si tratta di azioni elementari, e quindi effettivamente eseguibili, ma di operatori astratti che necessitano di un raffinamento. Essi costituiscono un'utile astrazione per ragionare a grandi linee: ad esempio, non esiste un'azione reale corrispondente al "fare la spesa al supermercato" (si tratta infatti di un'intera sequenza di azioni reali), eppure essa costituisce un concetto aggregato cui si fa riferimento comunemente.

A questo punto solo il primo operatore può essere raffinato (se l'agente possiede una mappa che gli permette di elaborare il percorso); il raffinamento del secondo necessita di informazioni che giungeranno in possesso dell'agente solo quando si troverà sul posto (del resto il raffinamento del secondo operatore è inutile e prematuro al momento); il raffinamento parte quindi dal primo operatore, che viene espanso in operatori più semplici.

L'ipotesi sottostante è che prima di procedere al raffinamento di un operatore, l'agente deve trovarsi ad un punto dell'esecuzione in cui abbia già raccolto le informazioni necessarie per procedere a tale raffinamento. Questi concetti saranno approfonditi nel prossimo capitolo.

III.2 Il Modello di Astrazione Proposto

Le principali tipologie di astrazione non si adattano, per differenti motivi, alle nostre esigenze:

- La pianificazione gerarchica basata su astrazione nello spazio degli stati è vincolata ad un meccanismo di raffinamento length-first: prima di procedere al raffinamento al livello i, il piano viene completamente raffinato al livello i-1. Questo esclude il raffinamento non-omogeneo (depth-first) prospettato in precedenza.
- La possibilità di un raffinamento non omogeneo del piano astratto è presente invece nel meccanismo di astrazione sugli operatori; quello che non va bene sono i metodi di decomposizione definiti dall'utente, che limitano il controllo sul piano (troppo statici); da questo punto di vista è migliore il primo metodo, in cui il raffinamento è ottenuto tramite un ulteriore lavoro di planning.

La soluzione adottata è in effetti un ibrido tra i due metodi: si basa congiuntamente sulla definizione di operatori astratti (da cui eredita la possibilità di raffinamento non-omogeneo) e su un meccanismo di astrazione nello spazio degli stati classificabile come un'estensione di quello utilizzato nei modelli ridotti (dal quale eredita il meccanismo di raffinamento mediante pianificazione).

La gerarchia di astrazione utilizzata si può pensare sviluppata su due dimensioni: quella di astrazione nello spazio degli stati e quella di astrazione sugli operatori (operatori astratti). Il primo meccanismo di astrazione si può considerare come un'estensione di quello utilizzato da [Knoblock91a]. E' importante sottolineare che il raffinamento di un operatore astratto avviene per mezzo di un'attività di planning e non tramite l'applicazione di metodi di decomposizione definiti dal progettista come nell'HTN, o di una decomposizione totalmente predefinita come per i macro-operatori di [Korf87]. E' quindi necessaria un'attività di planning sia per la scelta degli operatori astratti che per la scelta delle azioni concrete con cui espanderli; in generale, per ogni passaggio di livello nella gerarchia delle azioni è richiesta un'attività di pianificazione.

III.2.1 Astrazione nello Spazio degli Stati

Il modello proposto definisce una relazione $L_{i-1} \mathbin{®} L_i$ più generale rispetto ai modelli ridotti, rilassandone le caratteristiche digitali: i predicati del generico livello *i* sono una funzione di un sottoinsieme Δ_{i-1} di predicati del livello *i-1*, ove per funzione si intende un'espressione logica che può contenere connettivi in and oppure in or.

Definiamo tre tipi di associazioni possibili fra D_{i-1} e L_i:

- *Associazione di tipo identità.* Dato $P_i \mathbin{Î} L_i$, $P_i = P_{i-1}$, ove $P_{i-1} \mathbin{Î} D_{i-1}$, il predicato P_{i-1} filtra inalterato in L_i. Questa associazione viene ereditata dai modelli ridotti.
- *Associazione in or.* Dato $P_i \mathbin{Î} L_i$, $P_i = P_{i-11} \mathbin{Ú} P_{i-12}$, ove P_{i-11}, $P_{i-12} \mathbin{Î} D_{i-1}$. Questo tipo di associazione aggrega i predicati P_{i-11}, P_{i-12} rendendo indistinguibili in L_i stati ed operatori che in L_{i-1} differiscono per la loro presenza, ma senza bisogno di eliminare totalmente i predicati: P_{i-11} e P_{i-12} in realtà filtrano in L_i, anche se come concetto aggregato. Il risultato è una minore perdita di informazione rispetto ai modelli ridotti, perché un predicato di livello *i-1* cui non venga attribuita dignità di livello *i* può comunque lasciare traccia di sé in tale livello, seppure in forma aggregata con altri predicati di livello *i-1*. Questo crea un compromesso fra la brusca perdita di informazione conseguente all'eliminazione di un predicato, da un lato, ed il trasporto di troppi predicati ai livelli superiori, dall'altro.

- *Associazione in and.* Dato $P_i \in L_i$, $P_i = P_{i-11} \cup P_{i-12}$, ove P_{i-11}, $P_{i-12} \in D_{i-1}$. L'effetto dell'associazione in and è completamente duale rispetto all'associazione in or; in particolare, mentre la seconda elimina la differenza fra due affermazioni, la prima elimina la differenza fra due negazioni.

Si noti che l'operazione di eliminazione di un predicato, sulla quale si basano i modelli ridotti, è comunque contemplata nel modello esteso, ma come caso particolare (in Figura 16), ai predicati Λ_{i-1} ma non appartenenti al sottoinsieme Δ_{i-1}, non corrisponde alcuna associazione in Λ_i). Di questo particolare tipo di associazione si può fornire un'interpretazione in linea con le associazioni precedenti: essa in effetti elimina la differenza fra l'affermazione di un predicato e la sua negazione.

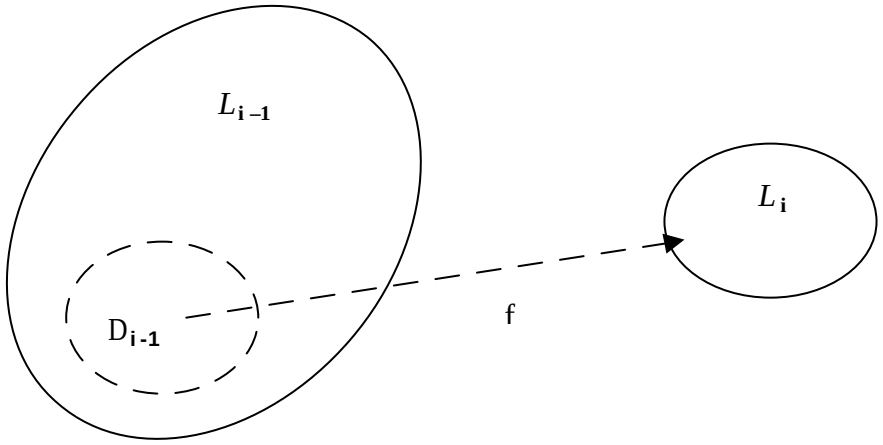


Figura 16 - Relazione di astrazione nel modello proposto.

III.2.2 Astrazione sugli Operatori

Come esposto nel paragrafo relativo all'astrazione mediante macro-operatori, la definizione manuale di un operatore astratto richiede uno sforzo di immaginazione da parte del progettista del dominio, sulle possibili sequenze che permettono di raggiungere il task che esso rappresenta. La possibilità di eseguire un operatore astratto sarà assicurata dalla possibilità di eseguire almeno una delle sequenze che lo espandono.

Uno degli obiettivi che questa tesi si propone è quello di automatizzare, tutto o in parte, questo processo. A partire da esempi di piani risolti con successo dall'agente vengono individuate le sequenze interessanti come possibili macro-operatori di supporto di un eventuale operatore astratto.

III.2.3 Proprietà Formali del Modello Proposto

Come evidenziato nella prima parte, data una gerarchia di astrazione, le proprietà fondamentali su cui investigare sono la *Downward Solution Property* (DSP) e la *Upward Solution Property* (USP), legate rispettivamente alla proprietà di correttezza e di completezza. Qui ricordiamo solamente che la DSP assicura che per ogni soluzione astratta esiste almeno una corrispondente soluzione a livello ground; mentre l'USP assicura che per ogni soluzione ground esiste una corrispondente soluzione astratta.

In genere, i metodi di astrazione assumono che sia verificata la USP, introducendo il problema delle cosiddette *false* soluzioni [Giunchiglia90], cioè soluzioni a livello astratto che non conducono a soluzioni di livello ground. Se il rapporto tra soluzioni *false* e *vere* (che hanno corrispondenza a livello ground) diventa elevato, l'utilizzo della tecnica di astrazione risulta addirittura meno efficiente della ricerca completa delle soluzioni senza astrazione (eseguita cioè completamente a livello ground).

Nel contesto di questa architettura si cerca di evitare l'introduzione delle false soluzioni, facendo in modo che la mappatura di astrazione sia corretta (quindi il raffinamento di un operatore astratto deve rispettarne la frontiera). Non vengono invece fatte particolari assunzioni sulla completezza, introducendo quindi il problema che non tutte le soluzioni ground possano essere trovate a partire dal livello astratto. Pertanto, se il livello astratto fallisce, occorre effettuare la ricerca a livello ground fino a trovare la soluzione o incontrare un'ulteriore situazione di fallimento (ad esempio per problemi di spazio o tempo).

Un'altra importante proprietà per il corretto funzionamento del sistema è la *Downward Refinement Property* (DRP). Come già detto, tale proprietà impone che, data una soluzione astratta, se esiste una corrispondente soluzione ground, allora essa è raffinabile; al lato pratico questo implica che, una volta ottenuto un piano ad un livello astratto, sia possibile ottenere un suo raffinamento senza mai fare backtracking fra un livello e l'altro. Questa caratteristica, desiderabile in un algoritmo classico perché garantisce la massima efficienza del processo di raffinamento e consente l'utilizzo di una strategia di ricerca che non preveda affatto backtracking tra livelli [Bacchus94], diviene indispensabile in un contesto in cui l'esecuzione ha inizio quando gran parte del piano è ancora allo stato astratto, come nel nostro caso. Il motivo per cui molte gerarchie di astrazione non soddisfano la DRP è la perdita di informazione che si verifica salendo di livello; questo fa sì che ai livelli astratti vengano considerate soluzioni in realtà impraticabili quando si cerchi di raffinarle. Il modello di astrazione nello spazio degli stati proposto, pur limitando, come abbiamo già mostrato, questo fenomeno, non lo elimina, e quindi non offre da solo una valida soluzione al problema della DRP. Il soddisfacimento della DRP deriva piuttosto dall'astrazione sugli

operatori: il pianificatore di livello più alto garantisce le connessioni fra operatori astratti, mentre i pianificatori dei livelli inferiori garantiscono sul loro contenuto. A questo punto l'esistenza di un raffinamento del piano discende dall'esistenza di un raffinamento per ogni operatore, e dal fatto che tale raffinamento rispetti perfettamente la frontiera dell'operatore. La prima condizione è garantita dalla semantica delle pre-condizioni di ogni operatore (le pre-condizioni di un operatore astratto costituiscono condizioni sufficienti per l'esistenza di una sua espansione); la seconda condizione dipende dal meccanismo di raffinamento.

Non sono verificate, invece, proprietà come la Ordered Monotonicity (OM) ed i raffinamenti non sono monotoni; essi potrebbero essere definiti globalmente monotoni, in quanto c'è la possibilità del temporaneo annullamento di una condizione durante l'espansione di un operatore, seguito dal suo ripristino alla fine, eventualità prospettata anche in [Knoblock91a] come possibile e utile rilassamento della proprietà di monotonicità.

IV COMPORTAMENTO PROATTIVO

Nel capitolo II è stata mostrata l'architettura generale del sistema proposto, in quel contesto è stato messo in luce come il ruolo centrale sia dato dal sistema di pianificazione che si appoggia al modello di astrazione proposto nel capitolo precedente. Risulta dunque necessario mostrare ora le caratteristiche del sistema di pianificazione.

IV.1 Il Sistema di Pianificazione Proposto

La pianificazione svolge un ruolo centrale nel comportamento generale dell'agente, dato che nella presente architettura l'enfasi è stata rivolta al comportamento proattivo di cui l'agente deve essere dotato per poter raggiungere i propri obiettivi.

Come implicito nell'architettura proposta e sottolineato nel paragrafo relativo alle interazioni tra i livelli, la generazione del piano è attuata su più livelli ciascuno dei quali utilizza solamente il set di operatori del proprio livello. In particolare, ciascun livello tratta solamente con gli operatori noti al proprio livello di astrazione, il livello ground dunque utilizza solamente operatori atomici, mentre gli altri livelli lavorando con operatori astratti. L'approccio seguito è stato chiamato *Hierarchical Interleaving Planning and Execution* (HIPE).

Per ciascun livello le azioni sono rappresentate in una sintassi PDDL [Ghallab98] mentre come pianificatore per ciascun livello (pianificatore locale) è stato scelto un pianificatore basato su UCPOP [Penberthy92]. La scelta del pianificatore è stata fatta solamente per motivi di rapidità di implementazione, in generale può essere incorporato qualsiasi pianificatore PDDL-compliant.

Poiché si è scelto un approccio di tipo HIPE, gli agenti continuamente pianificano, eseguono le proprie azioni e monitorano i cambiamenti che avvengono nell'ambiente, a tutti i livelli di astrazione. Come meccanismo di funzionamento generale, ciascun pianificatore locale è attivato da un goal imposto dal pianificatore del livello sovrastante (se esiste). Per cui un goal che deve essere raggiunto al livello $K > 0$ impone un goal al livello sottostante $(K - 1)$, per questo motivo un operatore astratto di un dato livello corrisponde ad un piano complesso a livello sottostante. In altre parole, le post-condizioni di un operatore astratto a livello $K > 0$ sono un goal per il pianificatore locale del livello $K - 1$. In particolare, riferendoci al caso più semplice in cui sono presenti solamente due livelli di astrazione (*ground* e *abstract*), il pianificatore di livello ground ha il compito di pianificare i goal provenienti dal livello abstract, in questo modo eseguire un'azione a livello abstract in realtà vuol dire sottoporre un goal a livello ground.

IV.1.1 HIPE (Hierarchical Interleaving Planning and Execution)

Soffermandoci su un'architettura caratterizzata da due soli layer (*strategico* e *situato*) ovvero da due soli livelli di astrazione (*abstract* e *ground*) descriviamo l'algoritmo di pianificazione.

La strategia di pianificazione è basata sul seguente comportamento. Prima di tutto a livello strategico viene creato un piano astratto, successivamente il primo operatore astratto viene raffinato a livello situato che genera il piano corrispondente. A questo punto, le azioni di livello ground possono essere eseguite, nel mentre il pianificatore astratto raffina l'operatore astratto successivo. Appena il piano corrente del livello situato è stato completato, il successivo operatore astratto viene eseguito e così via. La forma generale di un piano gerarchico è mostrata in Figura 17.

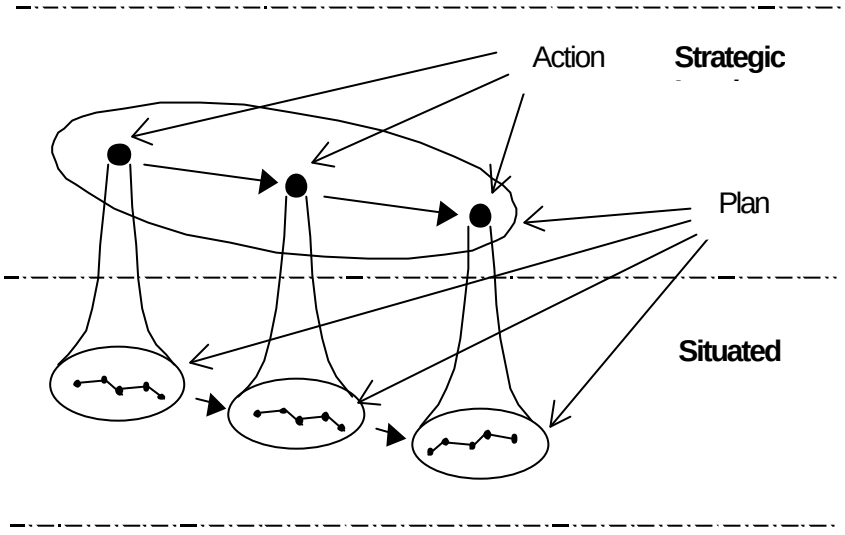


Figura 17 - Struttura di un piano gerarchico.

E' importante notare come un operatore astratto, considerato come un'unità atomica ed inscindibile nel proprio livello, divenga un piano se espanso in termini di operatori del livello inferiore. Se queste sono azioni elementari, allora è già possibile eseguire la prima parte del piano totale; altrimenti il processo di raffinamento dovrebbe procedere in questa modalità depth-first fino a raggiungere il livello ground (si veda la Figura 18). A questo punto sono disponibili le prime azioni elementari e l'esecuzione può avere inizio.

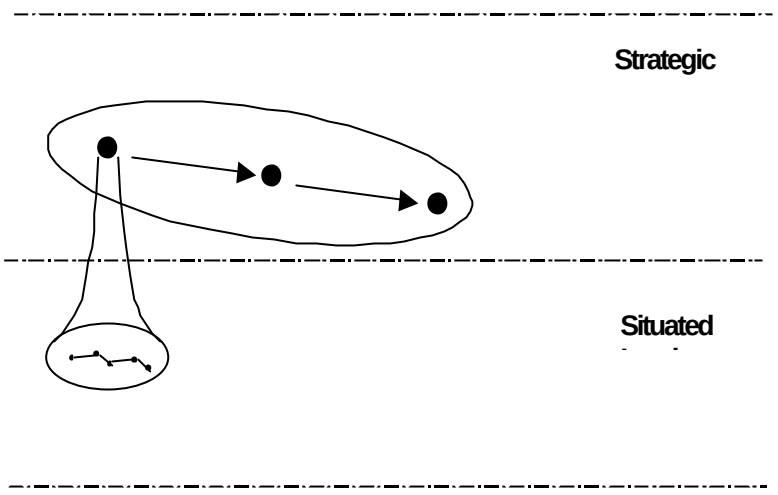


Figura 18 - L'esecuzione può iniziare prima di terminare il raffinamento a livello astratto.

Si noti che quando la prima azione elementare viene eseguita, la maggior parte del piano è costituita da operatori astratti a differenti livelli di raffinamento; solo una piccola parte del lavoro di panificazione necessario è stata effettuata, ed il resto avverrà durante l'esecuzione stessa. Si può affermare che l'astrazione sugli operatori costituisce un'euristica che guida l'alternanza tra pianificazione ed esecuzione (fondamentalmente l'esecuzione parte non appena viene generato un segmento di livello ground). Questo meccanismo consente di cominciare ad agire il prima possibile: la prima azione viene eseguita quando ancora gran parte del piano complessivo è definita solo per linee generali, quindi dopo una pianificazione ed un tempo di attesa minimi. La possibilità di espandere un operatore astratto quando ci si trova temporalmente e spazialmente vicini alla sua esecuzione permette di sospendere il più a lungo possibile la scelta della giusta espansione, fino a quando non si possiedano le informazioni necessarie per farlo. In altre parole, possiamo dire che un operatore astratto ingloba e nasconde dentro di sé l'incertezza.

La nostra necessità non è solo quella di generare il piano progressivamente durante l'esecuzione, ma anche quella di monitorarlo e correggerlo nelle parti divenute inconsistenti, si noti che la pianificazione gerarchica classica si limita alla creazione del piano; ora questo non deve solo essere generato, ma rivisto e riadattato durante l'esecuzione.

Da notare che questa strategia ha senso s'intanto che viene fatta l'assunzione che non esistono false soluzioni a livello astratto o, nel caso peggiore, che non possano essere eseguite azioni irreversibili che impediscano all'agente di raggiungere il piano corrente.

Essendo interessati a non introdurre false soluzioni a livello astratto, si assume che l'astrazione presentata è corretta in accordo con la definizione data in [Nourbakhsh97b], dove nessun'ipotesi è

fatta sulla completezza. Ovviamente, l'assenza di quest'ultima proprietà implica che non tutte le soluzioni ground possono essere trovate a partire dalla soluzione astratta. Tuttavia, quando avviene un fallimento a livello astratto, una ricerca complessiva deve essere effettuata a livello ground finché viene trovata una soluzione o viene segnalato un fallimento.

IV.1.2 Interazioni con gli Altri Moduli

Come mostrato nel Paragrafo II.1.2 le interazioni tra i moduli non avvengono solo tra di diversi livelli, ma anche all'interno di ciascun livello tra i moduli che si occupano della gestione di una particolare caratteristica del comportamento dell'agente.

A questo proposito ricordiamo le interazioni che coinvolgono il modulo proattivo. Nel suo libello di appartenenza, il modulo proattivo può sottostare a costanti interazioni con gli altri moduli (reattivo e deliberativo). Il modulo reattivo, infatti, quando una regola diventa vera può inviare un comando di interruzione (*break*) al modulo proattivo che provvederà ad interrompere l'attività che stava svolgendo in quel momento in attesa di una decisione del modulo deliberativo. Dopo la ricezione del comando di interruzione, il pianificatore locale informa il modulo deliberativo riguardo il piano corrente (*current plan*) in modo da agevolarlo per decidere se proseguire con quel piano o se dare priorità all'evento scatenante l'interruzione. A seconda della decisione presa dal modulo deliberativo, il pianificatore può continuare la propria attività (*continue*), iniziare un nuovo piano (*start*), rimanere in attesa (*reset*) o riprendere il piano precedentemente salvato (*resume*).

Come abbiamo già detto, l'attività di pianificazione può ripartire anche dopo la ricezione di un fallimento da un pianificatore di livello inferiore. Infine, il pianificatore, al termine dell'attività di pianificazione, si pone in stato di attesa (*reset*).

V COMPORTAMENTO ADATTATIVO

Per essere efficace l'approccio HIPE ha bisogno di un meccanismo adattativo, il cui obiettivo è quello di identificare nuovi operatori, che saranno utilizzati successivamente nella ricerca a livello astratto [Armano01b]. A questo scopo viene analizzato un certo numero di piani in modo da trovare sequenze rilevanti che possono giocare il ruolo di macro-operatori di supporto per la creazione di nuovi operatori per il livello astratto.

V.1 Il Sistema di Apprendimento Proposto

Come si è detto in precedenza, i livelli dell'architettura lavorano ad un livello di dettaglio differente e ciascuno di essi ha una propria base di conoscenza contenente le informazioni relative al proprio livello. Nel modulo di apprendimento viene dunque definito un *GOS REPOSITORY*, contenente l'insieme di operatori eseguibili dal livello ground e l'*AOS REPOSITORY* fondamentale per l'utilizzo del pianificatore gerarchico senza il sistema di apprendimento (che può essere inizialmente vuoto).

L'apprendimento presentato non gestisce l'astrazione dei predicati, che deve anch'essa essere fornita a priori nel DOMAIN ONTOLOGY REPOSITORY, si tratta in pratica di una tabella di corrispondenza tra i predicati dei due livelli.

Come illustrato in Figura 19, lo stimolo del sistema è rappresentato dagli obiettivi (goal) assegnati all'agente. L'interazione principale avviene col pianificatore HIPE che, nel caso di fallimento nella ricerca della soluzione a livello astratto, si occupa di rimandare la ricerca della soluzione completamente a livello ground (quindi senza approccio gerarchico).

In caso di successo, viene presentata una sequenza di passi (*plan*) a cui corrisponderanno le azioni elementari eseguite in successione dall'agente, in accordo al sistema di interleaving tra pianificazione ed esecuzione. Questo è il punto di partenza delle elaborazioni successive che costituiscono il sistema di apprendimento vero e proprio.

Prima di descrivere in dettaglio ogni modulo presente nello schema generale del sistema di apprendimento, riassumiamo le fasi principali in cui è articolato:

1. Attivazione del pianificatore HIPE su un determinato obiettivo. Se tutto va a buon fine, si ottiene un piano che viene eseguito.
2. Estrazione di sottosequenze di dimensione prefissata dal piano precedente (tecnica di chunking).

3. Filtraggio delle sequenze significative tramite una rete neurale feed forward (FFNN) opportunamente addestrata al loro riconoscimento. Come input della rete viene utilizzato un vettore di metriche, calcolato in corrispondenza ad ogni sequenza.
4. Processo di generalizzazione delle sequenze e creazione di macro-operatori come schemi corrispondenti (MOS). Questi vengono memorizzati nel MOS REPOSITORY.
5. Processo di astrazione di ogni macro-operatore per la creazione di un corrispondente schema astratto (AOS). Viene modificato l'AOS REPOSITORY, che contiene tutti i candidati AOS e i relativi MOS di supporto. Se l'AOS non è presente viene aggiunto col MOS da cui è stato generato, altrimenti viene modificato incorporando in esso il nuovo MOS di supporto.

Procedura di promozione da AOS candidato ad operatore astratto effettivo, utilizzabile dal pianificatore HIPE. Questo processo deve tener conto del compromesso tra l'aumento della complessità del livello astratto (causata dall'aumento del branching factor medio) e dell'aumento della capacità di risoluzione dei problemi (allargamento dello spazio delle soluzioni) a livello astratto.

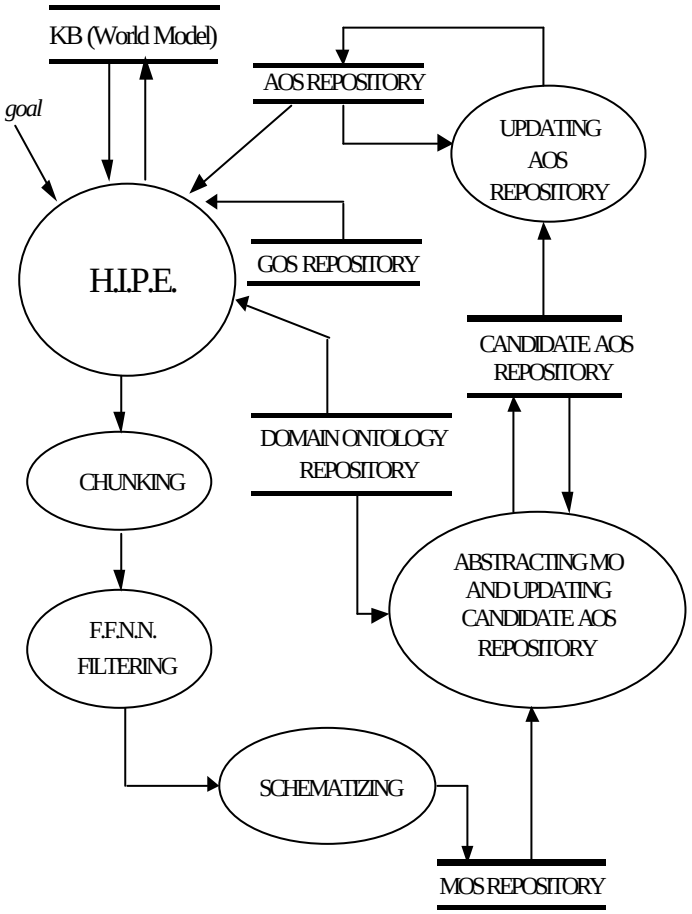


Figura 19 - Ciclo di vita del modulo di apprendimento.

Nel resto del capitolo verranno descritti brevemente i vari moduli componenti l'architettura, per la descrizione del modulo di pianificazione si rimanda al capitolo precedente.

V.1.1 Modulo di Chunking

Si occupa di estrarre, a partire da una sequenza di passi, tutte le possibili sottosequenze di dimensione massima prefissata. Chiaramente la dimensione minima è 2, diversamente si tratterebbe di un'unica azione (quindi non una sequenza in senso stretto) o di un piano nullo (nessuna azione). Se indichiamo con l la lunghezza del piano P e con N la dimensione massima accettata per una sottosequenza, avremo in uscita dal modulo esattamente $\sum_{k=1}^{N-1}(l-k)$ possibili sottosequenze. Ognuna di esse viene definita *chunk*. E' importante notare che il valore di N va scelto tenendo conto delle capacità di pianificazione ad un solo livello, poiché un operatore astratto potrebbe aver necessità, per il suo raffinamento, di un'azione di pianificazione. In ogni caso, il numero totale di chunks è controllato e prevedibile a priori.

Per rendersi conto del loro ordine di grandezza, basti pensare che ad un piano di 18 passi, limitando la dimensione massima di ogni sottosequenza al valore 5, corrispondono 62 differenti sottosequenze.

V.1.2 Modulo di Filtering

E' un modulo complesso, che si occupa di selezionare, in base a diversi criteri, le sequenze di azioni, da intendersi come possibili espansioni di un operatore astratto. Nella realizzazione di un filtraggio di questo tipo, si incontrano diverse problematiche, dipendenti in primo luogo dalla struttura dei chunks. Innanzitutto, la rilevanza di una sequenza, intesa come possibile espansione di un operatore astratto, è di natura semantica. Questo perché in un certo senso un operatore astratto rappresenta un concetto e i macro-operatori (le sequenze) ad esso associati sono le sue possibili concretizzazioni. Giudicare che una sequenza rappresenta meglio un concetto di un tipo rispetto ad un'altra, richiama proprietà semantiche delle azioni che non sono presenti nella struttura del dominio. Occorre quindi cercare di introdurre questo aspetto basandosi su proprietà sintattiche della sequenza: lunghezza, numero di pre-condizioni, relazioni logiche tra pre- e post-condizioni, e così via.

Data la difficoltà della definizione di una funzione in grado di tenere conto di tutti gli aspetti che concorrono alla classificabilità di una sottosequenza, intesa come ideale espansione di un operatore astratto, è stata utilizzata allo scopo una rete neurale. La natura della rete neurale è numerica, nel

senso che accetta dei vettori di ingresso di dimensione costante, producendo in corrispondenza un'uscita, anch'essa numerica.

Tabella 2 – Metriche proposte.

Nome	Formula	Breve Descrizione
Chunk Length	$LC(s)=\begin{cases}\frac{(\lvert s\rvert-2)^2}{16}-\frac{\lvert s\rvert-2}{2}+1, & 2\le\lvert s\rvert\le \\ 0\ otherwise\end{cases}$	Penalizza le sequenze lunghe associando un valore differente a ciascuna lunghezza nella sequenza.
Preconds-Length	$LPRE(s)=\frac{1}{\sqrt{\lvert pre(s)\rvert}} \quad \lvert pre(s)\rvert>0$	Penalizza le sequenze con un alto numero di pre-condizioni.
Postconds-Length	$LPOST(s)=\frac{\lvert s\rvert}{L_0+\lvert post(s)\rvert}$	Penalizza le sequenze con un alto numero di post-condizioni. L_0 indica la lunghezza preferenziale.
Establishment	$E(s)=\sum_{i=1}^{\lvert s\rvert-1}\frac{\lvert eff(op_i(s))\cap pre(sseq(s))\rvert}{\lvert pre(sseq_i(s))\rvert}$	Promuove le sequenze in cui ciascun operatore istanzia letterali utili al resto della sequenza. op_i i-ma azione della sequenza s. $sseq_i$ estrae il resto della sequenza s, partendo dall'i+1-mo operatore.
Redundancy	$R(s)=\frac{\lvert Ops(s)\rvert}{\lvert s\rvert}$	Penalizza le sequenze in cui uno stesso operatore è replicato. ops(s) estrae l'insieme di operatori di s.
Cost	$C(s)=\frac{1}{\sqrt{\sum_{i=1}^{L}opc(op_i(s))}}$	Promuove le sequenze che hanno un basso costo, il costo di una sequnza è stimato in base al numero di pre- e post-condizioni tramite la funzione opc(s). Un peso differente è dato alle pre- e post-condizioni, dato che il costo di una sequenza è dato principalmente dalle pre-condizioni.
Weighted-Postconds	$WPOST(s)=\frac{\sum_{i=1}^{\lvert eff(s)\rvert}\lvert occur(eff_i(s))\rvert}{numOps}$	Promuove le sequenze in base alla loro capacità di instanziare effetti che avvengono frequentemente come pre-condizioni di altri operatori. eff_i restituisce l'i-mo effetto introdotto dalla sequenza s. occur restituisce l'insieme di operatori che necessitano di un dato effetto. numOps è il numero totale di operatori.

Per far fronte al duplice problema di uniformare la dimensione degli ingressi e di misurare alcuni aspetti ritenuti significativi delle varie sottosequenze, sono state introdotte alcune metriche di supporto [Armano01c]. Viene così fatto corrispondere ad ogni chunk (di dimensione variabile) un vettore di numeri reali contenente la misura di alcune sue caratteristiche, assunte come significative per la sua classificazione. In Tabella 2 vengono mostrate le metriche che sono state introdotte.

V.1.3 Modulo di Schematizing

Il modulo che si occupa del filtraggio segnala, mediante un valore numerico compreso tra 0 e 1, i macro-operatori che presentano caratteristiche interessanti per il loro incorporamento in un operatore astratto. Tuttavia, le sequenze di azioni, così come sono state elaborate dal modulo di chunking, non sono direttamente utilizzabili, in quanto troppo specifiche.

Il modulo schematizing è incaricato di questo compito: a partire da un chunk ritenuto interessante (*relevant chunk*) viene creato uno schema più generale di macro operatore, di cui il chunk in esame è un caso particolare. Cioè le istanze particolari delle variabili vengono sostituite con il loro tipo, avendo cura di assegnare ad entità dello stesso tipo un numero progressivo per distinguerle.

Questa operazione di generalizzazione viene eseguita anche sulle precondizioni e gli effetti di ciascun operatore della sequenza, creando un particolare schema di macro-operatore definito MOS (*Macro Operator Schema*). In definitiva, un MOS incorpora sequenze simili (aventi lo stesso numero di parametri e le stesse relazioni logiche tra i parametri), che differiscono soltanto per il valore attribuito a ciascuna variabile.

Riepilogando, possiamo dire che il modulo schematizing effettua una generalizzazione intesa come de-istanziazione delle variabili presenti nel chunk al suo ingresso, creando una particolare struttura, denominata MOS. Tramite un algoritmo di matching si controlla che il MOS appena ottenuto non sia già stato memorizzato nell'area di sistema *MOS REPOSITORY*, che contiene tutti i MOS elaborati sino a quel momento.

V.1.4 I Moduli di Astrazione e Aggiornamento

Il MOS REPOSITORY è la fonte principale per il successivo modulo *Abstracting MOS and Updating candidate AOS*. Per essere utilizzati in un sistema gerarchico, i MOS necessitano di un'operazione di astrazione. Infatti, potrebbero essere pensati come raggruppamenti di GOS, utili in un contesto di apprendimento orizzontale, cioè in grado di stabilire delle scorciatoie tra gli stati di livello ground. Il motivo fondamentale per cui si è trascurato questo tipo di utilizzo (come aggiunte al corredo del GOS REPOSITORY) è essenzialmente il problema dell'aumento del branching factor medio. Infatti, aumentando il numero dei GOS, il pianificatore è costretto ad esaminarli tutti durante l'algoritmo di ricerca, creando dei problemi di efficienza. I MOS acquistano invece maggior potenza allorché si considerino come possibili raffinamenti di operatori astratti ancora più generali.

V.1.4.1 Modulo di Astrazione

Obiettivo del modulo di astrazione è la creazione dei *candidate AOS* (*Abstract Operator Schema*), intesi come possibili operatori astratti da inserire nell'AOS REPOSITORY e quindi utilizzabili dal pianificatore gerarchico HIPE. Come più volte accennato, l'astrazione automatica effettuata da questo sistema di learning riguarda gli operatori. Nel sistema, infatti, sono presenti altri tipi di astrazione, vale a dire sui predicati e sulle entità del dominio. E' fondamentale per il funzionamento del modulo di astrazione che venga fornita una ontologia del dominio, quindi le relazioni logiche tra i predicati e le entità dei due livelli.

I MOS, per il fatto che sono delle sequenze generalizzate, avranno delle pre- e post-condizioni, la loro *frontiera*, che è espressa in funzione di predicati appartenenti al linguaggio del livello ground. Per essere compresi al livello superiore (abstract), è necessaria una traduzione, in accordo a delle tabelle di corrispondenza fornite nel *Domain Ontology Repository*.

Il modulo in esame ha una duplice funzione: raccogliere i vari MOS dal MOS REPOSITORY astraendone la frontiera e creare gli AOS candidati per essere inseriti nell'AOS REPOSITORY. Il candidate AOS che si ottiene potrebbe essere nuovo (non ancora presente nell'appropriato repository) o già esistente, in tal caso si incorpora in quello già esistente il MOS da cui è stato generato, arricchendo la libreria delle sue possibili espansioni.

L'AOS è quindi definito dalla sua frontiera (pre- e post-condizioni) espressa in termini di predicati comprensibili a livello astratto e contiene al suo interno uno o più MOS che ne costituiscono possibili espansioni.

V.1.4.2 Modulo di Aggiornamento

Questo modulo si occupa di promuovere eventuali AOS presenti nel *CANDIDATE AOS REPOSITORY*, quindi non ancora utilizzati dal pianificatore, a veri e propri operatori astratti, da inserire nell'AOS REPOSITORY, in modo che diventino effettivamente utilizzabili dal pianificatore gerarchico HIPE.

Al di là delle apparenze, il suo obiettivo è uno dei più complicati di tutto il sistema. Bisogna infatti tenere conto di due aspetti contrastanti nella dimensione dell'AOS REPOSITORY. Il primo (positivo) è legato al fatto che l'aumento del corredo degli operatori astratti, allarga lo spazio dei problemi risolvibili in tale livello, riducendo la necessità di richiamare completamente il pianificatore di livello inferiore per la ricerca della soluzione. Il secondo (negativo) è una perdita di efficienza dovuta all'aumentare del branching factor medio durante la ricerca della soluzione a livello astratto, causato dal maggior numero di operazioni da esaminare. Affinché il corredo di operatori astratti raggiunga dimensione e contenuto ideali per bilanciare in modo ottimale i due aspetti, occorre effettuare ulteriori indagini a riguardo.

Nell’attuale implementazione, l’operazione di promozione dei candidati operatori astratti ad operatori astratti effettivamente utilizzabili dal pianificatore gerarchico, è a carico del progettista, rendendo l’operazione di generazione di operatori astratti non completamente automatica. Per chiudere il ciclo occorre introdurre delle metriche che consentano di valutare la convenienza dell’inserimento di un candidato AOS nella lista di quelli attivi.

VI RISULTATI SPERIMENTALI

L'architettura proposta è stata progettata per il progetto di un videogioco, come mostrato in [Armano00c]. In particolare, il gioco è caratterizzato da utenti che possono interagire con un mondo virtuale popolato da due differenti entità: fisiche e di rete. Quelle del primo tipo rappresentano gli abitanti del mondo virtuale (chiamati *avatars*), mentre i secondi (come per esempio i virus) vivono in una rete virtuale, ispirata ad Internet. Come detto, ciascun agente può interagire con l'ambiente sottostante, il giocatore e gli altri agenti, in Figura 20 è mostrata una vista della città virtuale.

Figura 20 - Una vista del mondo virtuale.

Sebbene l'applicazione sia stata realizzata in C++, è stato scelto di implementare il modulo di intelligenza artificiale in CLOS (Common Lisp Object System), tale modulo è stato poi integrato nel resto dell'applicazione tramite una DLL.

I test effettuati hanno lo scopo di verificare l'efficacia dell'approccio gerarchico proposto, in particolare per la verifica delle capacità proattive e adattative degli agenti progettati in base all'architettura proposta. Prima però verrà mostrato com'è stato creato il dominio, in particolare la parte gerarchica, per questo particolare ambiente applicativo.

VI.1 Generazione del Dominio Applicativo

VI.1.1 Gerarchia dei Predicati

Nella base di conoscenza del livello grpund è contenuta l'informazione sul mondo così come viene percepita a livello di sensori, più la conoscenza che l'agente ha del proprio stato; i predicati che rappresentano tale conoscenza sono detti predicati di base. Si tratta di informazioni dettagliate sullo stato interno dell'agente e sull'area del mondo fisico in cui l'agente si trova e sta agendo al momento (il raggio di percezione dell'agente). L'informazione sul mondo fisico presente nel primo livello è non solo spazialmente, ma anche temporalmente, circoscritta: quando l'agente si sposta da un luogo all'altro non conserva memoria di tutte le percezioni che ha ricevuto, il che equivarrebbe ad una memoria fotografica di oggetti e luoghi visitati. Quindi la conoscenza acquisita entro il raggio di percezione è in parte volatile e spetta al livello superiore filtrarne e trattenerne gli aspetti significativi.

```
(define (domain file-world-1)

  (:operator move
   :parameters ((Pl ?l1) (Pl ?l2))
   :precondition (:and (next-to ?l1) (:neq ?l1 ?l2))
   :effect (:and (next-to ?l2)(:not (next-to ?l1))))

  (:operator take-file
    :parameters ((file ?f) (PL ?l))
    :precondition (:and (located ?f ?l) (inside ?l))
    :effect (own ?f))

  (:operator insert-file
    :parameters ((file ?f) (PL ?l))
    :precondition (:and (own ?f) (inside ?l))
    :effect (located ?f ?l))

  (:operator upload-file
    :parameters ((file ?f) (PL ?l) (TL ?t))
    :precondition (:and (inside ?l) (connected ?l) (located ?f ?l))
    :effect (available ?f ?t))

  (:operator open-door
    :parameters ((door ?d) (tool ?t))
    :precondition (:and (own ?t)(closed ?d))
    :effect (opened ?d))

  ...

) ; end domain file-world-1
```

Figura 21 - Esempio di operatori del livello situato.

La conoscenza al livello astratto è espressa in forma di predicati complessi. Tramite i predicati complessi è possibile esprimere in modo più potente concetti, stato dell'agente e proprietà del mondo. Come abbiamo già detto, i predicati complessi sono definiti come funzione logica di un sottoinsieme dei predicati di base. Come già detto in precedenza, il meccanismo che dai predicati di

base permette di dedurre predicati complessi, che è stato definito come un filtraggio, corrisponde in effetti ad un meccanismo di inferenza logica in avanti.

Passando da un livello a quello superiore, diminuisce il livello di dettaglio nella conoscenza sul mondo, ma aumenta il raggio spaziale e temporale di percezione (perché i livelli superiori conservano memoria di cose e fatti non più appartenenti al raggio di percezione fisico), il potere espressivo e di conseguenza il potere di ragionamento.

VI.1.2 Gerarchia degli Operatori

Gli operatori del primo livello (situato), detti operatori atomici, corrispondono alle azioni che l'agente può effettivamente eseguire nel mondo. Gli operatori astratti definiti per il livello strategico sono detti invece operatori complessi e costituiscono un'astrazione interna. In Figura 21 è mostrato un insieme di operatori atomici, mentre la Figura 22 mostra i rispettivi operatori complessi.

Si può notare come la complessità del set di operatori sia elevata, ed equiparabile per i due livelli; infatti, sebbene la natura aggregata degli operatori complessi tenda ad aumentarne la complessità, l'astrazione sui predicati produce un effetto contrario e riequilibrante.

```
(define (domain file-world-2)

  (:operator goto
    :parameters ((PL ?l1) (PL ?l2))
    :precondition (:and (:neq ?l1 ?l2)(is-near ?l1))
    :effect (:and (is-near ?l2) (:not (is-near ?l1))))

  (:operator put-file
    :parameters ((file ?f) (PL ?l))
    :precondition (:and (own ?f) (is-near ?l))
    :effect (located ?f ?l))

  (:operator get-file
    :parameters ((file ?f) (PL ?l))
    :precondition (:and (located ?f ?l) (is-near ?l))
    :effect (own ?f))

  (:operator receive-file
    :parameters ((file ?f) (PL ?l) (TL ?t))
    :precondition (:and (is-near ?l) (connected ?l) (available ?f ?t))
    :effect (located ?f ?l))

  (:operator send-file
    :parameters ((file ?f) (PL ?l) (TL ?t))
    :precondition (:and (is-near ?l) (connected ?l) (located ?f ?l))
    :effect (available ?f ?t)))

  ...

) ; end domain file-world-2
```

Figura 22 - Esempio di operatori del livello strategico.

VI.2 Comportamento Proattivo

Per verificare l'efficacia del pianificatore gerarchico sono state messe a confronto le sue prestazioni con quelle del pianificatore non gerarchico su cui esso si basa, che usa un set di operatori di primo livello.

Come è ben noto, le risorse di tempo e memoria richieste da un pianificatore aumentano, a parità di numero di azioni costituenti un piano, all'aumentare della complessità di queste. Con il set di operatori utilizzato, un pianificatore poco efficiente come UCPOP ha incontrato difficoltà a fornire un risultato (nei limiti di tempo e memoria stabiliti) già con piani di 8 azioni. Il dato importante non è comunque costituito dai valori in sé quanto dal miglioramento delle prestazioni ottenuto grazie all'algoritmo gerarchico.

La misura tipica dei piani generati per raggiungere gli obiettivi richiesti durante i test è di 13 azioni, dunque la quasi totalità dei piani eseguiti dall'agente durante i test sono impossibili da generare ad un unico livello senza superare le risorse di tempo e memoria stabilite. L'utilizzo della pianificazione su due livelli ha permesso, come del resto previsto dall'analisi di complessità della pianificazione gerarchica, di superare tutti i test con un ampio margine. E' opportuno precisare che 13 azioni non rappresentano assolutamente il limite del pianificatore su due livelli, ma la misura media dei piani necessari a raggiungere gli obiettivi nelle simulazioni, escluse le azioni elementari di spostamento lungo le strade. In base ad una valutazione approssimata le prestazioni teoriche del pianificatore su due livelli migliorano quadraticamente rispetto a quelle del corrispondente pianificatore non gerarchico, con un limite quindi di circa 64 azioni nel dominio di valutazione.

VI.2.1 Un Caso Studio

Consideriamo ora il seguente scenario: "un agente si trova all'interno di un edificio connesso alla rete. Il suo obiettivo è di prendere possesso di un file localizzato in un altro edificio (non connesso alla rete), e di spedirlo via e-mail alla casella postale del giocatore".

Per risolvere questo semplice problema a livello astratto, l'agente deve nell'ordine: (i) andare nella locazione fisica in cui si trova il file; (ii) prendere il file; (iii) ritornare nell'edificio in cui si trovava inizialmente; e (iv) spedire il file via e-mail. Naturalmente, per quanto detto riguardo la distribuzione delle informazioni all'interno dell'architettura, il goal viene prima imposto dal modulo deliberativo e successivamente si inizia la fase di ricerca che porta alla creazione del piano al livello strategico.

Come sottolineato in precedenza, le azioni del livello strategico sono operatori astratti che devono essere opportunamente raffinati dal livello situato. Per esempio, l'operatore di livello

strategico *goto* origina un problema di pianificazione a livello situato. Un possibile raffinamento di tale operatore astratto è per esempio: *open*, *go-outside*, *move*, questi operatori corrispondono ad azioni che possono essere effettuate direttamente sull'ambiente sottostante.

In Figura 23 è mostrato un possibile piano di livello strategico e uno dei suoi possibili raffinamenti.

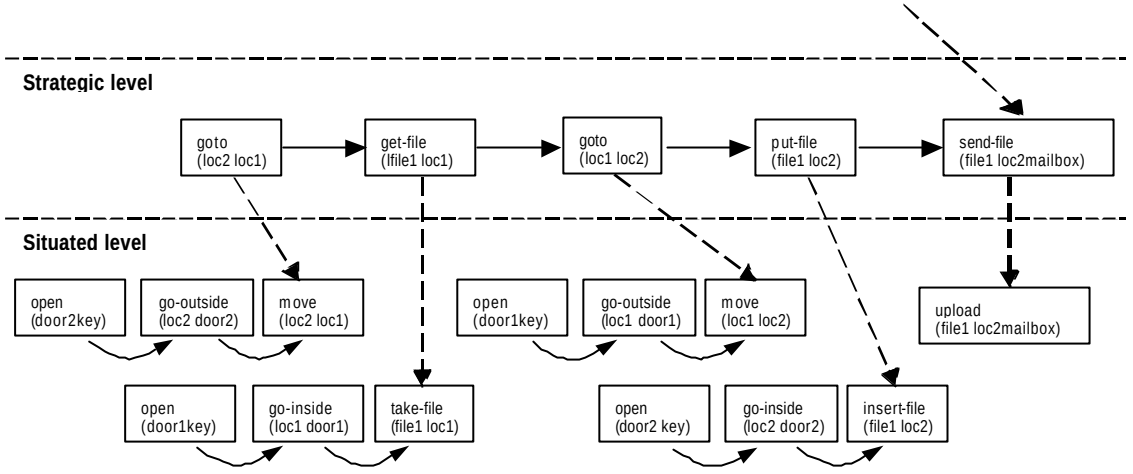


Figura 23 – Un esempio di piano che mostra come il livello strategico e quello situato collaborano per risolvere un dato problema.

VI.3 Comportamento Adattativo

Del ciclo di vita mostrato in precedenza, sono stati implementati i passi dall'1 al 5. Per quanto riguarda l'ultimo passo stiamo attualmente lavorando per risolvere il problema della stima del grado di completezza di un piano astratto, dato un insieme di operatori astratti. In effetti, questo è un punto molto importante e di difficile soluzione, dato che non è facile stabilire se un insieme di operatori disponibile al pianificatore del livello astratto è utile oppure no.

VI.3.1 Addestramento

La rete neurale utilizzata è di tipo feed-forward a 8 ingressi, con 3 neuroni nello strato intermedio e restituisce in output un numero reale appartenente al range [0,1] (vedi Figura 24). Il dimensionamento è stato effettuato tenendo conto del problema dell'"overfitting", che impone un limite superiore al numero dei suoi elementi, proporzionalmente al numero di esempi nel training-set. Infatti, se la rete risulta troppo grande, si corre il rischio che acquisisca a memoria gli elementi propostigli durante l'addestramento, perdendo così la capacità di generalizzazione. D'altra parte non può essere troppo piccola, pena la sua saturazione. Il compromesso lo si stabilisce empiricamente, verificando che la rete abbia imparato gli esempi forniti con un certo scostamento: non deve essere eccessivo, ma nemmeno nullo.

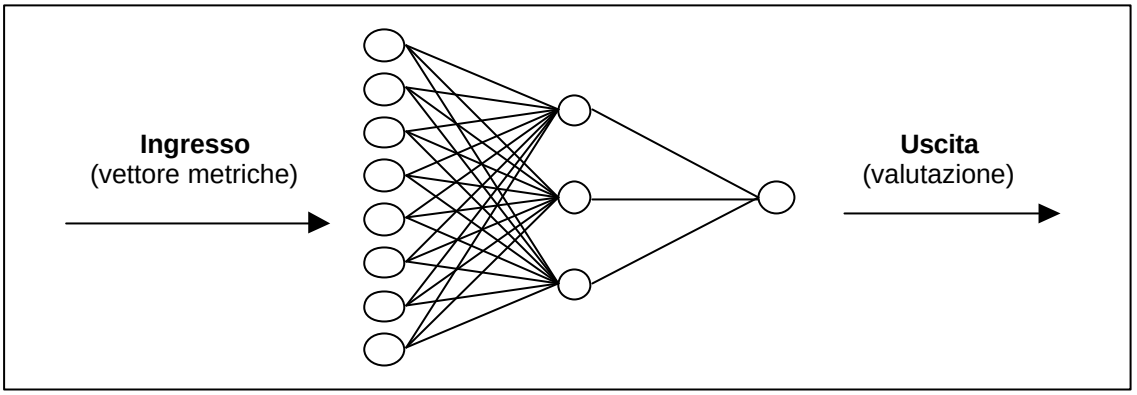


Figura 24 - Struttura della rete neurale utilizzata.

La rete è stata addestrata con 180 sequenze estratte da piani generati dal pianificatore HIPE. L'addestramento ha dato luogo a circa il 60% di esempi positivi e il 40% negativi. Gli esempi positivi sono stati presi da raffinamenti di operatori astratti definiti in precedenza "a mano".

Una soglia μA è stata usata per separare le sequenze rilevanti da quelle non rilevanti. Dato il vettore di valutazione delle metriche (M) e il classificatore neurale (f_{NN}), una sequenza s può essere considerata rilevante per l'astrazione se e solo se $f_{NN}(M(s)) \geq m_A$. Con $m_A = 0.65$ circa il 97% delle sequenze di training sono state riconosciute come appartenenti alla loro classe di appartenenza (rilevanti o non-rilevanti). Per quanto detto poco fa, è stata abbandonata l'idea di raggiungere un rapporto di 100% sul training set per evitare problemi di overfitting.

Tabella 3 – Un esempio di sequenze identificate come rilevanti per l'astrazione.

Ground Level		
Supporting MO - #1	Actions	((*open-door* door house) (*go-inside* door house) (*move-to-obj* key house) (*take-obj* key house) (*move-to-obj* friend house) (*give* key friend))
	Preconds	((:NOT (opened door house)) (located friend house) (next-to self house) (:NOT (blocked road)) (located key house))
	Postconds	((owns friend key) (next-to-obj friend) (:NOT (located key house)))
	Metrics	(0.04 0.28 0.66 0.83 0.83 0.20 0.68 0.62)
	FFNN output	0.84
Supporting MO - #2	Actions	((*move-to-obj* key house) (*take-obj* key house) (*move-to-obj* friend house) (*give* key friend))
	Preconds	((located friend house) (inside self house) (:NOT (blocked road)) (located key house))
	Postconds	((owns friend key) (next-to-obj friend) (:NOT (located key house)))
	Metrics	(0.09 0.35 0.57 0.61 0.75 0.26 0.62 0.56)
	FFNN output	0.72
Abstract Level		
	Preconds	((located agent building) (is-near self building) (located object building) (movable object))
	Postconds	(owns agent object) (:NOT (located object building)))
Ontology		
	IS-A	(is-a house building) (is-a friend agent) (is-a key object) (is-a next-to is-near) (is-a inside is-near)
	Invariants	(movable key)

VI.3.2 Sperimentazione

Durante gli esperimenti, la rete neurale ha identificato diverse sequenze utili per l'astrazione che non erano state create come raffinamenti di un operatore astratto esistente. Possiamo notare che

tutte le sequenze trovate dalla rete neurale sono rilevanti. In Tabella 3 è dato un esempio in cui sono mostrati due macro-operatori, che corrispondono all'azione astratta "prendi un oggetto e dallo ad un altro agente". Per ciascun macro-operatore di supporto sono illustrati: (i) la sequenza delle azioni; (ii) le pre- e post-condizioni; (iii) il vettore delle metriche; e (iv) l'output offerto dalla rete neurale.

Ulteriori informazioni, ovvero le pre- e post-condizioni dell'operatore astratto, così come un frammento dell'ontologia del dominio, sono riportate per comprendere meglio il meccanismo sottostante.

VI.3.3 Un Caso Studio

VI.3.3.1 Addestramento

In Figura 25 è mostrato un semplice scenario utilizzato per l'addestramento della rete. L'agente si trova all'esterno di un edificio (*loc4*), che ha la porta aperta. Il suo obiettivo è quello di fare in modo che la scatola *box1*, situata all'interno di *loc2* si trovi all'interno dell'edificio *loc4*. Tuttavia, l'edificio *loc2*, di natura nemica, è sotto sorveglianza luminosa; occorre, quindi, spegnere la luce tramite l'interruttore *lsw1* situato nella strada *loc1*. Inoltre, *loc2* è protetto da una porta elettrica *door2*, controllata da un altro interruttore *sw3* che si trova nella strada *loc3*. Prima di accedere all'edificio *loc2* è pertanto necessario utilizzare i due interruttori: uno per spegnere la luce, l'altro per aprire la porta. Successivamente l'agente può raccogliere la scatola e passare da un edificio all'altro per compiere l'operazione di trasferimento dell'oggetto.

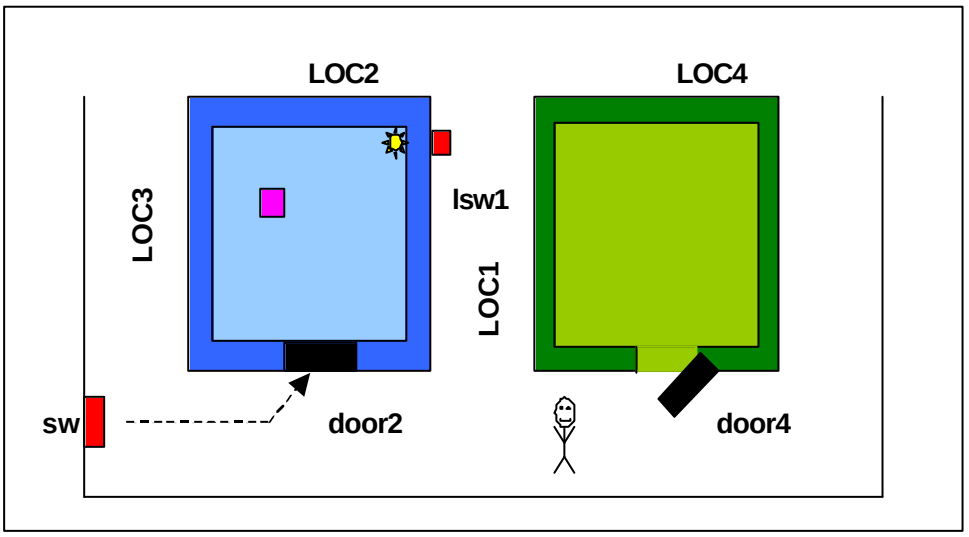


Figura 25 – Scenario per l'addestramento.

La semplicità del problema è soltanto apparente. Infatti, è necessario tener conto di parecchi vincoli e di oggetti di vario tipo, ragion per cui sarebbe impossibile, o quanto meno dispendioso in termini di tempo e spazio, cercare di risolvere il problema senza alcun approccio gerarchico (cioè ad

un solo livello). Il problema dato in ingresso al pianificatore HIPE genera un piano di 16 passi, ben oltre il limite del pianificatore ad un solo livello (tipicamente 8-10 passi). Precisiamo che l'utilizzo dei due livelli è servito soltanto per poter ottenere un piano di dimensione adeguata, da cui estrarre un training set per la rete neurale. Si veda la Figura 26 per la sequenza completa di azioni che, come si può facilmente verificare, conduce dallo stato iniziale al goal .

(MOVE loc4 loc1) (MOVETOOBJ lsw1 loc1) (USESWITCH lsw1 loc2)
(MOVE loc1 loc2) (MOVE loc2 loc3)
(MOVETOOBJ sw3 loc3) (USESWITCH sw3 door2)
(MOVE loc3 loc2) (GOINSIDE loc2 door2)
(MOVETOOBJ box1 loc2) (TAKEOBJECT box1 loc2)
(OPENDOOR door2 loc2) (GOOUTSIDE loc2 door2)
(MOVE loc2 loc4) (GOINSIDE loc4 door4) (DROPOBJECT box1 loc4)

Figura 26 - Risoluzione del problema di addestramento proposto.

Il passo successivo consiste nell'estrarre dal piano precedente tutte le possibili sottosequenze di lunghezza massima prefissata. Come detto in precedenza, i valori possibili che si possono assegnare alle sequenze appartengono all'insieme [0,1]. Ad una sequenza valutata significativa si possono dunque dare valori compresi tra 0.65 e 0.95, mentre ad una ritenuta scarsamente significativa i valori tra 0.05 e 0.35. Riportiamo in Tabella 4, come verifica della correttezza dell'apprendimento, alcune delle risposte della rete agli esempi del training set proponendo affianco i valori originali proposti durante l'addestramento.

Tabella 4 - Output della rete neurale sulle sequenze del training set.

SEQUENZA	Voto	Orig.
((MOVE LOC4 LOC1) (MOVETOOBJ LSW1 LOC1))	0.36	0.1
((MOVETOOBJ LSW1 LOC1) (USESWITCH LSW1 LOC2))	0.81	0.9
((USESWITCH LSW1 LOC2) (MOVE LOC1 LOC2))	0.11	0.2
((MOVE LOC1 LOC2) (MOVE LOC2 LOC3))	0.14	0.1
((MOVE LOC2 LOC 3) (MOVETOOBJ SW3 LOC3))	0.36	0.1
((MOVETOOBJ SW3 LOC3) (USESWITCH SW3 DOOR2))	0.62	0.9
((USESWITCH SW3 DOOR2) (MOVE LOC3 LOC2))	0.09	0.1
((MOVE LOC3 LOC2) (GOINSIDE LOC2 DOOR2))	0.57	0.75
((GOINSIDE LOC2 DOOR2) (MOVETOOBJ BOX1 LOC2))	0.54	0.1
((MOVET OOBJ BOX1 LOC2) (TAKEOBJECT BOX1 LOC2))	0.64	0.95
((TAKEOBJECT BOX1 LOC2) (OPENDOOR DOOR2 LOC2))	0.1	0.2
((OPENDOOR DOOR2 LOC2) (GOOUTSIDE LOC2 DOOR2))	0.89	0.75
((MOVE LOC4 LOC1) (MOVETOOBJ LSW1 LOC1) (USESWITCH LSW1 LOC2))	0.83	0.85
((MOVETOOBJ LSW1 LOC1) (USESWITCH LSW1 LOC2) (MOVE LOC1 LOC2))	0.42	0.25
((USESWITCH LSW1 LOC2) (MOVE LOC1 LOC2) (MOVE LOC2 LOC3))	0.07	0.05
((MOVE LOC1 LOC2) (MOVE LOC2 LOC3) (MOVETOOBJ SW3 LOC3))	0.12	0.1
((MOVE LOC2 LOC3) (MOVETOOBJ SW3 LOC3) (USESWITCH SW3 DOOR2))	0.64	0.85
((MOVETOOBJ SW3 LOC3) (USESWITCH SW3 DOOR2) (MOVE LOC3 LOC2))	0.25	0.2
((USESWITCH SW3 DOOR2) (MOVE LOC3 LOC2) (GOINSIDE LOC2 DOOR2))	0.61	0.7
((MOVE LOC3 LOC2) (GOINSIDE LOC2 DOOR2) (MOVETOOBJ BOX1 LOC2))	0.63	0.1
((GOINSIDE LOC2 DOOR2) (MOVETOOBJ BOX1 LOC2) (TAKEOBJECT BOX1 LOC2))	0.77	0.85
((MOVETOOBJ BOX1 LOC2) (TAKEOBJECT BOX1 LOC2) (OPENDOOR DOOR2 LOC2))	0.26	0.3

((TAKEOBJECT BOX1 LOC2) (OPENDOOR DOOR2 LOC2) (GOOUTSIDE LOC2 DOOR2))	0.26	0.4
((OPENDOOR DOOR2 LOC2) (GOOUTSIDE LOC2 DOOR 2) (MOVE LOC2 LOC4))	0.91	0.9
((GOOUTSIDE LOC2 DOOR2) (MOVE LOC2 LOC4) (GOINSIDE LOC4 DOOR4))	0.82	0.8
((MOVE LOC2 LOC4) (GOINSIDE LOC4 DOOR4) (DROPOBJECT BOX1 LOC4))	0.89	0.85
((MOVE LOC4 LOC1) (MOVETOOBJ LSW1 LOC1) (USESWITCH LSW1 LOC2) (MOVE LOC1 LOC 2))	0.24	0.2
((MOVETOOBJ LSW1 LOC1) (USESWITCH LSW1 LOC2) (MOVE LOC1 LOC2) (MOVE LOC2 LOC3))	0.36	0.15
((MOVE LOC2 LOC3) (MOVETOOBJ SW3 LOC3) (USESWITCH SW3 DOOR2) (MOVE LOC3 LOC2))	0.2	0.4
((MOVETOOBJ SW3 LOC3) (USESWITCH SW3 DOOR2) (MOVE LOC3 LOC2) (GOINSIDE LOC2 DOOR2))	0.87	0.85
((USESWITCH SW3 DOOR2) (MOVE LOC3 LOC2) (GOINSIDE LOC2 DOOR2) (MOVETOOBJ BOX1 LOC2))	0.56	0.2
((MOVE LOC3 LOC2) (GOINSIDE LOC2 DOOR2) (MOVETOOBJ BOX1 LOC2) (TAKEOBJECT BOX1 LOC2))	0.67	0.75
((GOINSIDE LOC2 DOOR2) (MOVETOOBJ BOX1 LOC2) (TAKEOBJECT BOX1 LOC2) (OPENDOOR DOOR2 LOC2))	0.32	0.4
((MOVETOOBJ BOX1 LOC2) (TAKEOBJECT BOX1 LOC2) (OPENDOOR DOOR2 LOC2) (GOOUTSIDE LOC2 DOOR2))	0.74	0.7
((TAKEOBJECT BOX1 LOC2) (OPENDOOR DOOR2 LOC2) (GOOUTSIDE LOC2 DOOR2) (MOVE LOC2 LOC4))	0.49	0.65
((OPENDOOR DOOR2 LOC2) (GOOUTSIDE LOC2 DOOR2) (MOVE LOC2 LOC4) (GOINSIDE LOC4 DOOR4))	0.94	0.85
((GOOUTSIDE LOC2 DOOR2) (MOVE LOC2 LOC4) (GOINSIDE LOC4 DOOR4) (DROPOBJECT BOX1 LOC4))	0.91	0.85
((MOVE LOC4 LOC1) (MOVETOOBJ LSW1 LOC1) (USESWITCH LSW1 LOC2) (MOVE LOC1 LOC2) (MOVE LOC2 LOC3))	0.28	0.25

VI.3.3.2 Testing

Come esempio di casi di test, consideriamo un problema abbastanza simile a quello utilizzato per l'addestramento. L'agente (Figura 27) si trova all'interno dell'edificio *loc1* con la porta chiusa. L'obiettivo è quello di portare la scatola *box*, dalla costruzione nemica *loc2* all'interno dell'edificio *loc3*. Per semplicità si considera la porta di *loc3* già aperta. L'edificio *loc2* è protetto dalla porta elettrica *door2*, controllata dall'interruttore *sw2* che si trova sulla strada *loc4*. Una differenza con il caso proposto per l'addestramento è che l'interruttore *sw2* controlla sia l'illuminazione di *loc2* che la porta *door2*; pertanto è necessario un duplice azionamento di *sw2*, prima di accedere all'edificio.

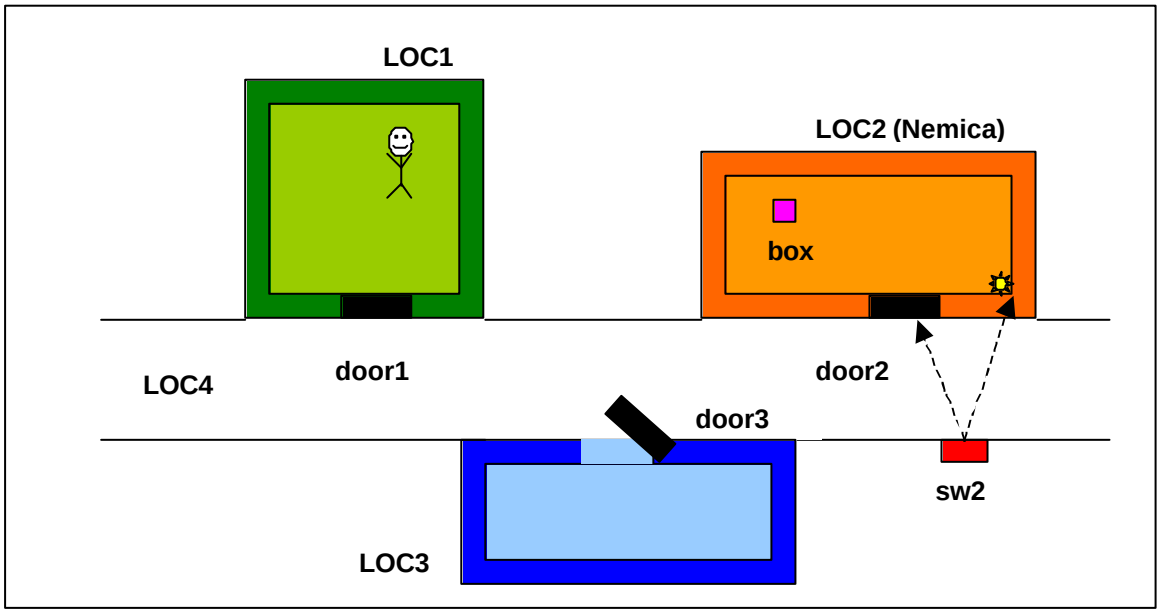


Figura 27 - Scenario per il testing.

Anche in questo caso, sono stati presentati stato iniziale e obiettivi al pianificatore gerarchico, che ha fornito la soluzione in 18 passi. A partire dal piano ground è possibile ottenere 62 chunks, utili per valutare il filtraggio effettuato dall'architettura neurale. Fissata come soglia il valore 0.65,

in accordo con la suddivisione dell'intervallo effettuata durante l'addestramento, è possibile esaminare le sequenze ritenute valide espansioni di un operatore astratto. Su 62 sequenze esaminate, l'architettura neurale ne ha rilevate 27 interessanti (il 43.5%), 15 negative (il 24.2%) e 20 (il 32.3%) appartengono alla regione di indecisione. Ai fini della classificazione, intesa come filtraggio degli elementi più interessanti, gli elementi appartenenti alla regione di indecisione possono comunque intendersi come negativi. Quindi considereremo come interessanti gli elementi per cui l'output della rete neurale è maggiore di 0.65, gli altri verranno scartati.

E' fondamentale precisare che non tutti i chunk, aventi valutazione superiore alla soglia prefissata, possono essere incorporati in un operatore astratto. Occorre esaminarne la frontiera (espressa in termini di pre- e post-condizioni) dopo l'operazione di mappatura verso il livello superiore. I macro-operatori di supporto proposti dal sistema sono mostrati nella Tabella 3.

VII SVILUPPI FUTURI

Gli agenti presentati in questa applicazione, si configurano come agenti autonomi nell'accezione più comune e meno ambiziosa: scelta e attuazione autonoma di un comportamento tale da raggiungere un obiettivo fornito dall'esterno. La conformazione <*condition, goal*> delle regole utilizzate nella realizzazione della componente reattiva contiene però un meccanismo di generazione autonoma degli obiettivi, che apre una breccia nella direzione di un livello più alto di autonomia e merita di essere ulteriormente esplorato e potenziato.

Per quanto riguarda il comportamento proattivo dell'agente, nella modellazione dell'ambiente applicativo è stata trascurata la gestione delle risorse (cibo, denaro, carburante, ecc.) e i relativi problemi di pianificazione che la loro introduzione comporterebbe. Inoltre, il pianificatore utilizzato nell'implementazione è UCPOP, ma si può facilmente incorporare un altro pianificatore più efficiente per migliorare le prestazioni complessive dell'attività di pianificazione dell'agente. Si può anche pensare di implementare e incorporare un pianificatore ad-hoc che intrinsecamente gestisca l'astrazione.

Per quanto riguarda il comportamento adattativo, rimangono ancora dei problemi aperti. Rimane infatti da terminare l'ultima fase del ciclo di vita dell'apprendimento. In particolare, bisognerebbe indagare gli aspetti che consentono di valutare, dato un insieme di operatori astratti, quale sia il bilanciamento ottimale tra le soluzioni individuabili a livello astratto e il branching factor medio dello spazio di ricerca. Rimangono inoltre due ulteriori direzioni su cui indagare, relative all'astrazione automatica di predicati del linguaggio e di entità del dominio (i tipi). Un sistema di apprendimento dovrebbe, infatti, generare automaticamente delle gerarchie sulle tre dimensioni di predicati, operatori e tipi.

Anche per quanto riguarda le tecniche di astrazione utilizzate, attualmente molto semplici, risulta interessante studiare tecniche più evolute che sfruttino relazioni di tipo is-a e part-of.

Infine, per quanto concerne l'interazione tra gli agenti, attualmente essi interagiscono con l'utente finale e solo in minima parte tra di loro. In particolare, al momento gli agenti sono in grado solamente di scambiarsi oggetti. La naturale estensione del comportamento degli agenti porterebbe quindi allo studio e all'implementazione di tecniche più evolute di comunicazione, che consentano agli agenti di scambiarsi informazioni utili al conseguimento dei propri obiettivi. In una visione multi-agente, inoltre, gli agenti potrebbero cooperare per raggiungere più velocemente i propri goal.

Parte III

I VERSO UN MODELLO AD ELEVATA FLESSIBILITÀ

Nell'introduzione è stato mostrato come l'approccio orientato agli agenti può essere utilizzato per lo sviluppo di applicazioni complesse. In particolare, dal nostro punto di vista, gli agenti per potere operare in ambienti complessi (come ad esempio il mondo reale) necessitano di manifestare un comportamento proattivo e, possibilmente, adattativo.

Nella seconda parte di questa tesi, è stata presentata un'architettura atta a mostrare come un approccio agent-based bene si presta alla realizzazione di sistemi software in grado di operare in un ambiente complesso. In particolare, gli agenti presentati in questo lavoro sono dotati di un comportamento proattivo e adattativo.

La ricerca di modelli computazionali per lo sviluppo di sistemi software ad elevata flessibilità, deve, quindi, mirare allo studio di tecniche evolute di pianificazione e apprendimento in grado di dotare l'agente di un comportamento il più possibile autonomo e intelligente. In questo senso diventa centrale il ruolo dell'astrazione per aumentare il livello di indiretezza nella direzione di un sistema ad agenti generico per la risoluzione di problemi complessi.

Per quanto riguarda le caratteristiche proattive, un ulteriore passo avanti nella ricerca di un approccio il più generale e flessibile possibile, deve portare a considerare le caratteristiche degli agenti in senso generale prescindendo dal particolare pianificatore reale che l'agente incorpora al proprio interno. Questo significa, a livello teorico, che un agente può incorporare un qualsiasi pianificatore decidendo lui stesso quale modulo di pianificazione esibire a seconda del particolare problema che è chiamato a risolvere.

Per quanto riguarda il comportamento adattativo, assume importanza lo studio di tecniche di apprendimento che consentano all'agente di adattarsi alle esigenze mutevoli dell'ambiente in cui opera. L'integrazione degli aspetti adattativi con quelli proattivi nel senso generale appena proposto, potrebbe portare allo sviluppo di sistemi ad agenti, in cui gli attori del sistema sono in grado di adattare anche le proprie capacità di pianificazione in base al particolare problema proposto.

Soffermandoci solamente sugli aspetti proattivi, nei capitoli successivi proporremo un'architettura, nata dalla fusione di regole di astrazione con l'idea di poter incorporare un qualsiasi pianificatore, volta a sperimentare tecniche di astrazione per la realizzazione di un sistema di pianificazione parametrico.

II HW[ ] UN PIANIFICATORE GERARCHICO PARAMETRICO

Essendo interessati allo studio di modelli flessibili che riducano la complessità computazione intrinseca dei sistemi complessi, è stato implementato un sistema parametrico in grado di incorporare un pianificatore che è in grado di risolvere problemi di planning a qualsiasi livello di astrazione ([Armano03]). In pratica, il sistema opera come un "wrapper" per un pianificatore esterno in grado di risolvere problemi espressi secondo la sintassi PDDL [Ghallab98].

Il sistema è stato chiamato HW[ ] (Hierarchical Wrapper). Il nome enfatizza la sua natura parametrica; da notare che anche le parentesi quadre fanno parte del nome e sottolineano la possibilità di incorporare un pianificatore esterno. Per esempio, se P fosse il nome del pianificatore incorporato, la notazione HW[P] indica un'istanza di HW[ ] che manifesta le capacità di pianificazione di P.

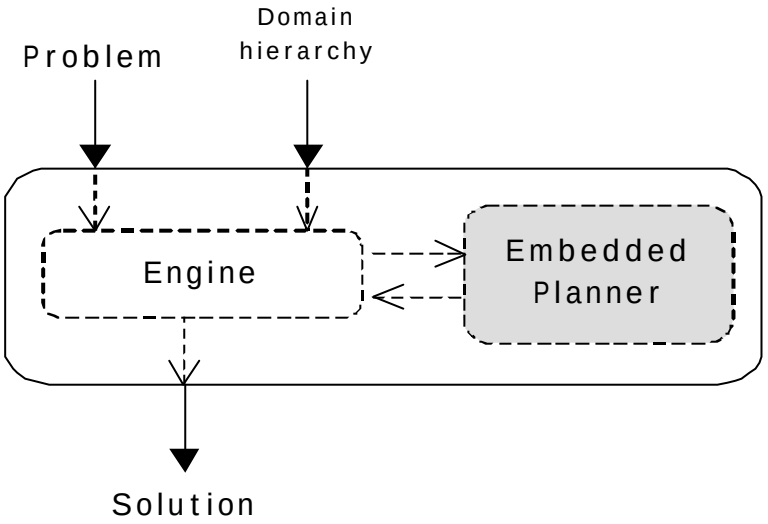


Figura 28 – Architettura di HW[ ].

La Figura 28 mostra l'architettura del sistema proposto focalizzandosi sui principali componenti: un'engine e un pianificatore esterno. Il primo ha il compito di controllare la comunicazione tra i livelli adiacenti, mentre il secondo è in grado di risolvere problemi di pianificazione a qualsiasi livello di astrazione.

In linea di principio, ogni pianificatore indipendente dal dominio e rispondente alla sintassi PDDL, potrebbe essere incorporato dal sistema, congiuntamente a una descrizione data all'engine per il controllo dell'attivazione del pianificatore in base alla comunicazione tra i livelli di astrazione adiacenti. Per realizzare la pianificazione a diversi livelli di astrazione, l'engine deve essere in grado di operare traduzioni bi-direzionali, in modo da garantire la comunicazione tra i livelli. Ovviamente si sta facendo l'ipotesi che ogni livello della gerarchia possieda il proprio insieme di

operatori, tipi e predicati in grado di interpretare e risolvere il problema a differenti livelli di granularità. Il particolare, la versione di HW[ ] che presentiamo in questo lavoro di tesi effettua un'astrazione sui predicati, come avviene nei modelli ridotti. E' in fase di progetto e sviluppo anche una generazione semi-automatica delle gerarchie di astrazione basata sull'astrazione sugli operatori e sui predicati.

In questo lavoro di tesi prenderemo in considerazione solamente l'astrazione sui predicati, in pratica viene fatta una selezione on/off dei predicati presenti a ciascun livello verso il successivo. Sebbene, in generale, l'architettura presentata possa operare su un numero qualunque di livelli di astrazione, per semplicità in questo contesto verranno considerati solamente due livelli di astrazione: *ground* e *abstract*.

II.1 Un'Estensione del PDDL per Gestire l'Astrazione

Istanziato il pianificatore P, HW[P] riceve in ingresso il problema da risolvere e una descrizione strutturata del dominio, compreso un insieme di regole da usare per tradurre i predicati ground in predicati astratti e viceversa. Da notare, infatti, che per pianificare a diversi livelli di astrazione, l'engine di HW[ ] deve operare traduzioni bi-direzionali –verso l'alto e verso il basso– atte a permettere la comunicazione tra livelli adiacenti.

Il problema e la corrispondente descrizione del dominio sono descritti in accordo con la sintassi del PDDL 2.1 tramite la dichiarazione *define problem* e *define domain*, rispettivamente. In Figura 29 è mostrato un semplice esempio di definizione estrapolato dal dominio *elevator* [Koehler00], usato nella competizione tra pianificatori di AIPS 2000 [Bacchus00].

```
(define (domain elevator-ground)
 (:requirements :typing)
 (:types passenger floor)
 (:predicates
  (origin ?person – passenger ?floor - floor)
   [...etc...]
  (lift-at ?floor - floor))
 (:action board
   :parameters (?f - floor ?p - passenger)
   :precondition (and (lift-at ?f) (origin ?p ?f))
   :effect (and (boarded ?p)))
    [...etc...]
 (:action down
   :parameters (?f1 ?f2 - floor)
   :precondition (and (lift-at ?f1) (above ?f2 ?f1))
    :effect (and (lift-at ?f2) (not (lift-at ?f1)))))
```

Figura 29 - Descrizione del dominio *elevator* a livello ground.

Per aggiungere un livello di astrazione al dominio devono essere fornite ulteriori informazioni che consistono in (i) una descrizione del livello astratto e (ii) una descrizione della modalità in cui avvengono le comunicazioni bi-direzionali tra i livelli. Come mostrato nelle Figure 30 e 31, si è deciso di mantenere la prima descrizione nel formato PDDL, e di estendere lo stesso PDDL per gestire la seconda.

```
(define (domain elevator-abstract)
 (:requirements :typing)
 (:types passenger floor)
 (:predicates
  (origin ?person - passenger ?floor - floor)
  (destin ?person - passenger ?floor - floor)
  (boarded ?person - passenger)
  (served ?person - passenger))
 (:action load
  :parameters (?p - passenger ?f - floor)
  :precondition (and (origin ?p ?f))
  :effect (and (boarded ?p)))
 (:action unload
  :parameters (?p - passenger ?f - floor)
  :precondition (and (boarded ?p) (destin ?p ?f))
  :effect (and (served ?p))))
```

Figura 30 - Descrizione del dominio *elevator* a livello astratto.

Focalizziamo l'attenzione sulla Figura 31, la forma generale per identificare una traduzione verso l'alto consiste nello descrivere, attraverso una clausola in stile LISP, un predicato che fa parte del livello più alto e la corrispondente traduzione –rispettivamente a destra e a sinistra della clausola. In particolare, l'estensione del PDDL proposta, è intesa a definire le regole di traduzione. La forma generale di tali regole è la seguente:

```
((abstract-predicate ?pa1 – ta1 ?pa2 – ta2 …)
 (ground-predicate ?pg1 – tg1 ?pg2 – tg2 …))
```

Ovviamente, questa descrizione può essere usata per effettuare sia la traduzione verso l'alto che verso il basso, dipendendo dalle operazioni effettuate dall'engine per far lavorare il pianificatore P al giusto livello di astrazione. In particolare, quando l'obiettivo del livello astratto può essere impostato, viene effettuata una traduzione verso l'alto in modo da convertire il problema di livello ground nel corrispettivo di livello astratto. Analogamente, quando un operatore astratto può essere definito, avviene una traduzione verso il basso.

Come esempio di regola di traduzione consideriamo la clausola:

```
((origin ?p – passenger ?f - floor)
(origin ?p - passenger ?f - floor))
```

la quale specifica che il predicato `origin` rimane invariato passando da livello ground a livello astratto e viceversa.

Un ulteriore esempio illustra come un predicato può essere trascurato nel dominio astratto:

```
(nil (lift-at ?f - floor))
```

Tale formula indica che il predicato `lift-at` non viene tradotto in nessun predicato del livello astratto. Da notare che questo tipo di clausola potrebbe anche essere omessa e assunta di defaulf. Tuttavia, per chiarezza l'ingegnere della conoscenza potrebbe comunque esplicitare quali predicati

```
(define (hierarchy elevator)
 (:domains elevator-ground elevator-abstract)
  (:translations
   (:level 0
    (:predicates
      ((origin ?p - passenger ?f - floor) (origin ?p - passenger ?f - floor))
      ((destin ?p - passenger ?f - floor) (destin ?p - passenger ?f - floor))
      ((boarded ?p - passenger) (boarded ?p - passenger))
      ((served ?p - passenger) (served ?p - passenger))
      (nil (above ?f1 - floor ?f2 - floor))
      (nil (lift-at ?f - floor)))))))
```

non vengono tradotti a livello astratto seguendo la sintassi appena mostrata.

Figura 31 - Gerarchia di astrazione per il dominio *elevator.*

II.2 Algoritmo di Pianificazione

Per trovare una soluzione di un dato problema, prima di tutto l'engine di HW[P] traduce la sezione *init* e *goal* dal livello ground al livello astratto. Il pianificatore P viene quindi attivato per ricercare *m* soluzioni astratte, ordinate per lunghezza. Successivamente, viene selezionata una soluzione e ogni operatore astratto viene raffinato invocando P ripetutamente.

Il raffinamento di un operatore astratto è effettuato attivando P, a livello ground, traducendo verso il basso gli effetti del goal appena raggiunto. Da notare che lo stato iniziale di ciascun raffinamento dipende dal raffinamento precedente; di conseguenza, i raffinamenti devono essere effettuati in base all'ordine in cui gli operatori sono stati trovati nel piano astratto. Quando un raffinamento fallisce, viene attivata una fase di backtracking. Il processo termina quando viene trovata una soluzione oppure quando la ricerca globale fallisce. Va sottolineato che per evitare di annullare incidentalmente dei sotto-goal già raggiunti durante i precedenti raffinamenti, essi vengono aggiunti alla lista dei sotto-goal nel momento in cui viene tradotto verso il basso l'operatore astratto da raffinare.

II.3 Generazione di Astrazioni

Come visto anche nella seconda parte di questo lavoro di tesi, il ruolo dell'astrazione è centrale nell'ambito della pianificazione gerarchica.

Nelle sperimentazioni che verranno esposte nel prossimo capitolo, le astrazioni sono state codificate “a mano”. In particolare, sono state studiate le caratteristiche di ciascun dominio di interesse. Con l’ausilio della semantica di tipi, operatori e predicati e considerando piani svolti in precedenza, sono stati individuati i dettagli rilevanti e quelli trascurabili. In seguito a queste considerazioni, sono stati considerati i possibili macro-operatori, ovvero le sequenze di azioni maggiormente ricorrenti.

E’ facile intuire che questo modo di procedere, oltre ad essere dettato dalle attitudini dell’ingegnere della conoscenza, risulta molto lungo e laborioso. Per questo motivo, è in fase di studio un approccio automatico per la generazione di astrazioni. L’approccio automatico su cui stiamo lavorando, prevede due fasi: (i) l’individuazione di schemi di macro-operatori in base ad un’analisi a posteriori; (ii) la selezione di alcuni dei macro-operatori trovati e la successiva traduzione in operatori astratti.

Il primo passo verrà realizzato tramite un algoritmo per la creazione un grafo diretto, i cui nodi rappresentano gli operatori e i cui archi rappresentano relazioni tra gli effetti del nodo sorgente e le precondizioni del nodo destinazione. Il grafo così ottenuto verrà quindi sfoltito in base a un insieme di metriche atte a valutare ciascuna sequenza individuata.

Il secondo passo verrà realizzato tramite un opportuno algoritmo per la selezione degli schemi di macro-operatori ottenuti al passo precedente e la successiva traduzione in operatori astratti. La traduzione viene effettuata eliminando dal livello astratto tutti i predicati che non sono presenti come precondizioni e/o effetti di ciascun macro-operatore selezionato. Un’operazione analoga è prevista per i tipi. Di conseguenza, a livello astratto rimarranno solamente tipi e predicati presenti negli operatori astratti trovati.

Da notare, infine, che questo approccio, potrebbe essere generalizzato al caso in cui si vogliano realizzare delle astrazioni orientate ad un particolare problema. In tal caso, nella generazione del grafo basterà introdurre un operatore fittizio che tenga conto degli obiettivi del problema. Tale nuovo operatore non avrà effetti ed avrà come precondizioni i predicati che caratterizzano il goal da raggiungere. In questo modo, un’ulteriore metrica di valutazione terrà conto di sequenze di operatori che terminano con tale operatore.

III RISULTATI SPERIMENTALI

Il lavoro presentato in questa sessione della tesi, tuttora in fase di sviluppo, nasce per sperimentare tecniche di astrazione per la realizzazione di sistemi flessibili. Attualmente il prototipo è implementato in C++.

Tabella 5 – Comparazione dei risultati ottenuti dai pianificatori GP, BB e LPG e dalle loro controparti gerarchiche.

Problem	*GP*	*HW[GP]*	*BB*	*HW[BB]*	*LPG*	*HW[LPG]*
elevator-1-4	0.007	0.062	0.098	0.333	0.01	0.11
elevator-3-1	0.234	0.364	1.342	1.208	0.02	0.15
elevator-4-1	1.956	0.837	1.030	1.744	0.02	0.16
elevator-4-4	10.114	0.839	311.046	1.792	0.02	0.16
elevator-5-1	364.74	2.032	180.781	2.548	0.02	0.18
elevator-7-2	--	12.043	--	3.899	0.03	0.29
logistics-4-2	0.682	1.227	0.266	0.461	17.93	--
logistics-5-2	0.085	0.165	0.151	0.466	0.02	--
logistics-7-0	--	10.931	4.494	2.176	2.12	--
logistics-8-1	--	16.265	2.898	3.027	1.55	--
logistics-10-0	--	43.435	8.272	3.763	2.17	--
logistics-15-0	--	203.467	10.915	6.333	0.15	--
blocks-4-0	0.345	0.317	0.162	0.674	0.02	0.08
blocks-6-0	3.040	1.821	0.263	1.684	0.05	0.23
blocks-8-0	31.612	11.123	0.924	2.467	0.36	0.31
blocks-10-0	--	--	6.821	5.003	0.62	0.67
blocks-11-0	--	--	16.236	4.254	4.23	0.83
blocks-14-0	--	--	--	9.845	5.00	1.91
blocks-15-0	--	--	--	--	7.49	2.07
blocks-17-0	--	--	--	--	33.93	3.49
blocks-20-0	--	--	--	--	66,78	7.88
blocks-22-0	--	--	--	--	183.16	12.21
blocks-25-0	--	--	--	--	668.98	24.94
zeno-1	0.027	0.519	0.223	0.369	0.02	0.03
zeno-8	--	42.549	0.941	2.360	0.14	0.49
zeno-9	--	--	0.344	3.377	0.13	1.08
zeno-11	--	--	11.203	2.786	0.16	1.06
zeno-13	--	--	62.999	20.524	0.42	2.47
zeno-14	--	--	--	20.042	3.90	21.93
gripper-2	4.722	0.565	0.424	0.635	0.02	0.07
gripper-3	7.914	1.732	5.224	1.203	0.02	0.12
gripper-4	18.321	2.630	268.737	1.554	0.02	0.14
gripper-5	57.212	4.377	421.186	1.548	0.03	0.15
gripper-6	--	7.968	586.439	2.267	0.03	0.17
gripper-9	--	24.294	--	3.631	0.05	0.36

Gli esperimenti sono stati realizzati incorporando tre diversi pianificatori che rappresentano in un certo qual modo lo stato dell'arte dei pianificatori: GRAPHPLAN [Blum97], BLACKBOX [Kautz98] e LPG (Local Planning Graph) [Gerevini02]. Nel seguito useremo le abbreviazioni GP, BB ed LPG per indicare rispettivamente GRAPHPLAN, BLACKBOX ed LPG e HW[GP], HW[BB] e HW[LPG] per

indicare i corrispettivi pianificatori gerarchici. La rilevanza dei risultati sperimentali è garantita dal fatto che per ciascun test vengono confrontati i risultati dei pianificatori non gerarchici e delle rispettive controparti gerarchiche.

Per verificare l'efficienza dell'approccio proposto, sono stati fatti alcuni test preliminari su domini presi dalle competizioni internazionali di pianificazione, AIPS 2002 [Long02], 2000 [Bacchus00] e 1998 [Long98]. In particolare, in questo capitolo mostreremo i risultati ottenuti con i seguenti domini: *elevator*, *logistics*, *blocks-world*, *gripper* e *zeno-travel*.

Gli esperimenti sono stati realizzazti sun una macchina con processore Intel Celeron a 1200 MHz e con 256 MB di RAM.

III.1 Test sui Domini Classici

Tutti i domini sono stati strutturati in due livelli di astrazione: concreto e astratto, seguendo l'approccio mostrato nelle Figure 29, 30 e 31.

Per ciascun dominio sono stati eseguiti diversi test, caratterizzati da un aumento incrementale della complessità, la Tabella 5 confronta i tempi misurati per ciascun pianificatore sull'insieme di problemi presi dalle competizioni internazionali. Le linee tratteggiate indicano problemi che non è stato possibile risolvere dal corrispondente sistema nell'arco di tempo massimo consentito (1000 secondi).

III.1.1 Test sul Dominio *Elevator*

Il dominio *elevator* è costituito da un ascensore che deve trasportare dei passeggeri da un piano ad un altro. Per ciascun passeggero è noto il piano di partenza e di arrivo. L'obiettivo è di servire tutti i passeggeri. L'aumento di complessità in questo dominio è dato dall'aumentare del numero di passeggeri e di piani. La Figura 32 mostra un esempio di stato iniziale (a) e finale (b) relativo a questo dominio.

Gli esperimenti mostrano in maniera chiara che per GP e BB il tempo di CPU cresce molto rapidamente all'aumentare della lunghezza del piano. Per quanto riguarda HW[GP] e HW[BB], invece, il tempo impiegato cresce più regolarmente, anche se la relazione tra il numero di passi e il tempo di CPU rimane esponenziale (si veda la Figura 33, in cui l'asse delle ordinate è espresso in scala logaritmica). LPG, infine, è in grado di risolvere piani lunghi in pochi secondi al massimo; questo vanifica l'uso dell'astrazione in questo dominio con questo pianificatore.

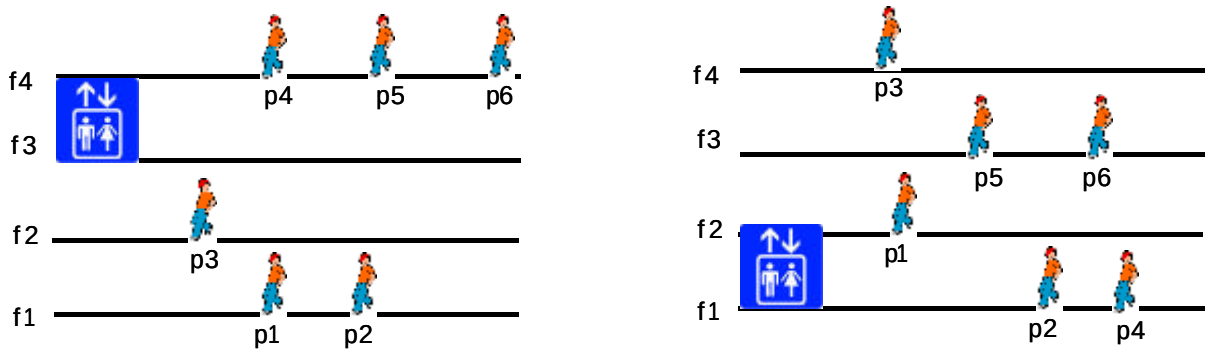


Figura 32 - Esempio di problema per il dominio *elevator*, (a) stato iniziale (b) stato finale.

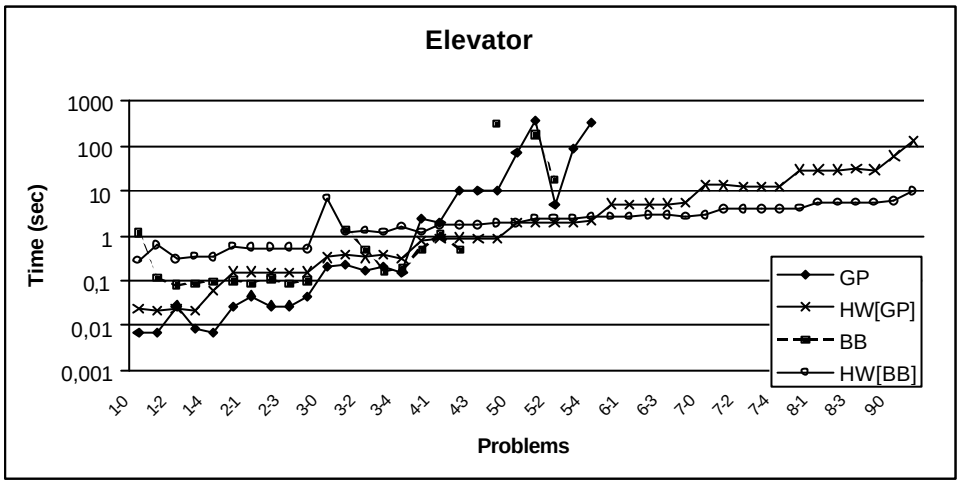
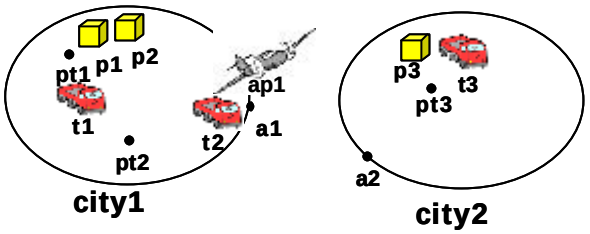


Figura 33 - Comparazione dei tempi di CPU per il dominio *elevator*.

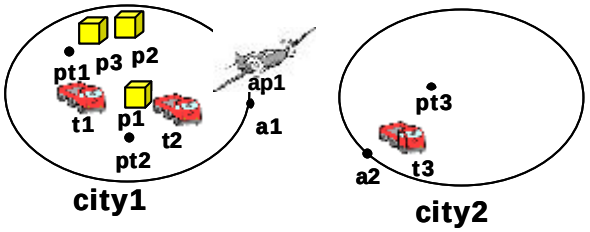
III.1.2 Test sul Dominio *Logistics*

Il dominio *logistics* descrive un insieme di città, ciascuna contenente diverse locazioni, alcune delle quali sono aeroporti. Ci sono, inoltre, carri merci, usati per i collegamenti interni alla città, e aerei, usati per i collegamenti tra città diverse. L'obiettivo è di sposare dei pacchi da una posizione ad un'altra (quella di destinazione). L'aumento della complessità del dominio è dato dall'aumentare del numero di oggetti presenti. La Figura 34 mostra un esempio di scenario in cui sono evidenziati stato iniziale (a) e finale (b).

In questo dominio GP, come mostrato in Figura 35, risolve facilmente problemi di una cerca lunghezza ma non è in grado di risolvere problemi da una certa complessità in poi con i limiti temporali imposti nelle nostre sperimentazioni. D'altra parte, HW[GP] riesce a risolvere problemi anche particolarmente complessi senza manifestare tale fenomeno. Per quanto riguarda BB, esso



(a)



(b)

Figura 34 - Esempio di problema per il dominio *logistics*, (a) stato iniziale (b) stato finale.

lavora meglio di HW[BB] per problemi piccoli, ma per problemi maggiormente complessi HW[BB] riesce ad ottenere risultati migliori. LPG è in grado di risolvere piani lunghi in pochi secondi. Per motivi ancora in fase di studio, LPG non è risultato essere in grado di raffinare gli operatori astratti quando è stato invocato attraverso HW[LPG].

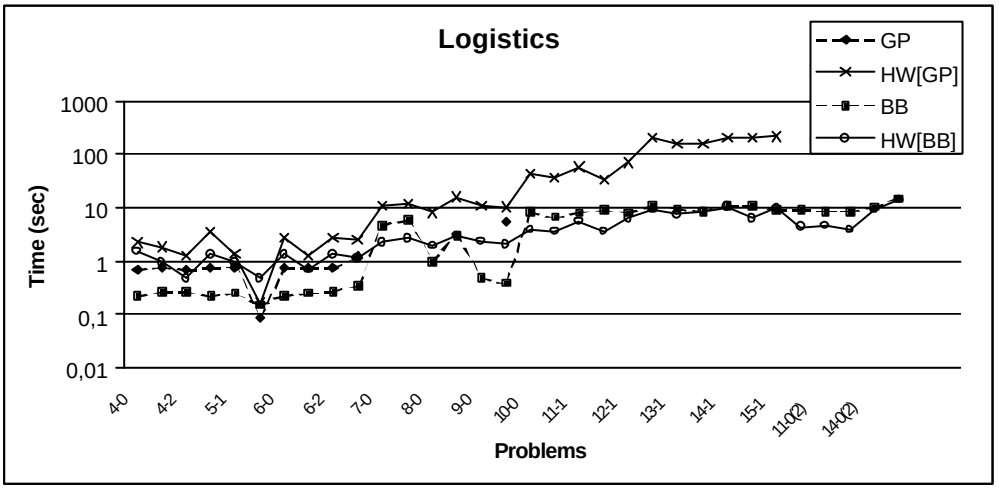


Figura 35 - Comparazione dei tempi di CPU per il dominio *logistics*.

III.1.3 Test sul Dominio *Blocks-World*

Il dominio del mondo dei blocchi (*blocks-world*) è uno dei più noti domini di planning e risulta ampiamente usato per le sperimentazioni, poiché nonostante appaia come un dominio facile manifesta in realtà caratteristiche interessanti per lo studio della pianificazione. Per questo motivo è stato usato anche in questa tesi come banco di prova per l'astrazione.

Questo dominio è formato da un insieme di blocchi (vedi Figura 36 per un esempio di problema) posti sopra un tavolo, che possono essere impilati uno sull'altro. Attraverso un braccio meccanico è possibile spostare i blocchi uno alla volta impilandoli o de-impilandoli su altri blocchi, oppure prendendoli o mettendoli sul tavolo.

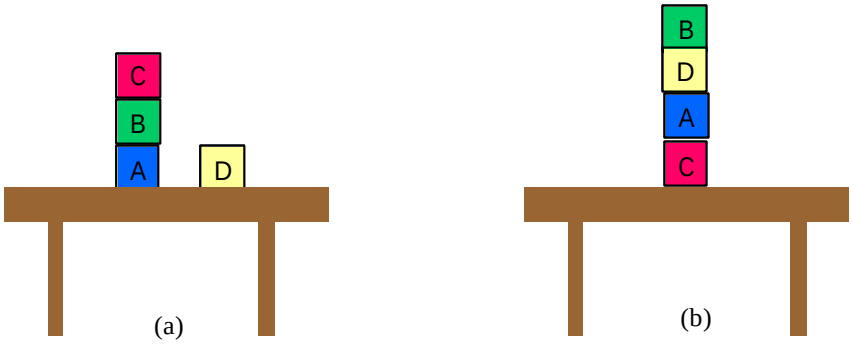


Figura 36 - Esempio di problema per il dominio *blocks-world*, (a) stato iniziale (b) stato finale.

Come mostrato in Figura 37, le prove eseguite su questo dominio hanno dimostrato un andamento simile tra GP e HW[GP], anche se quest'ultimo ha delle prestazioni leggermente migliori. BB, dal canto suo, lavora più velocemente di HW[BB] per problemi semplici, mentre HW[BB] risulta migliore di BB per problemi di una certa complessità. Per quanto riguarda LPG, esso è in grado di risolvere problemi la cui lunghezza della soluzione sia limitata a circa 100 passi (vedi Figura 38). In questo dominio, risulta chiaro che HW[LPG] ha delle prestazioni migliori di LPG per problemi complessi.

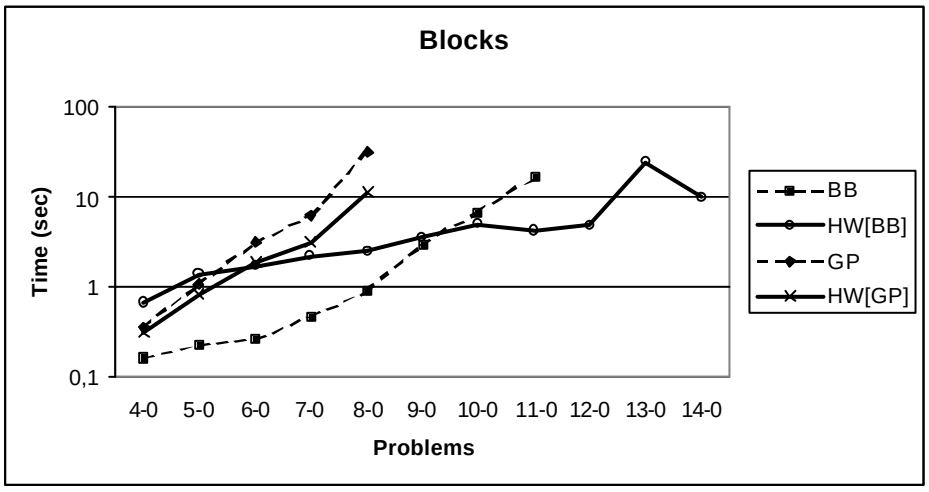


Figura 37 - Comparazione dei tempi di CPU per il dominio *blocks-world*.

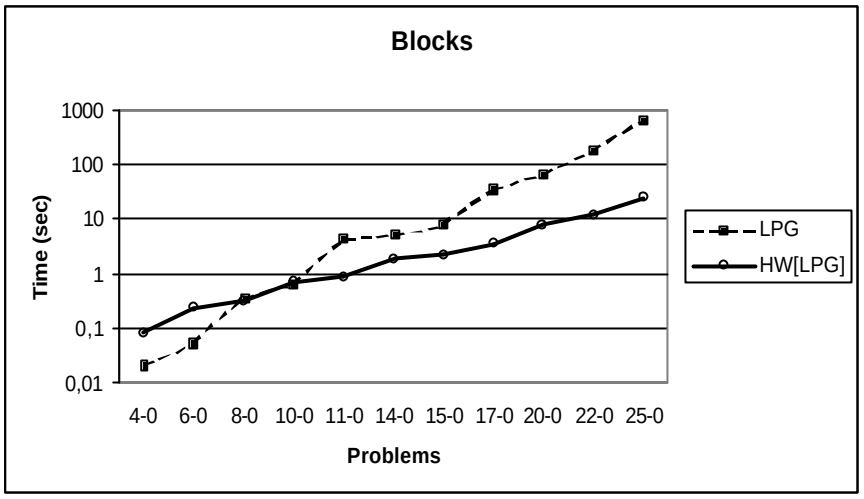
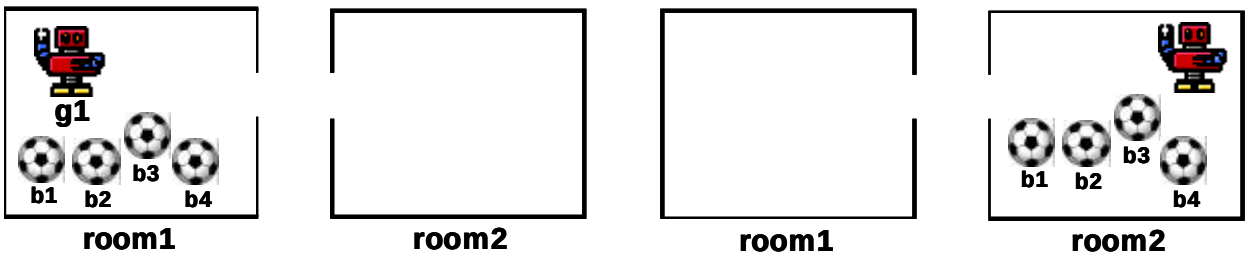


Figura 38 - Comparazione dei tempi di CPU di LPG e HW[LPG] per il dominio *blocks-world*.

III.1.4 Test sul Dominio *Gripper*

In Figura 39 è mostrato un semplice esempio di problema per il dominio *gripper*. Questo domino è caratterizzato da un robot con due bracci meccanici (i gripper). Il robot deve essere usato per caricare e spostare delle palle da una stanza ad un'altra.

La Figura 40 riporta alcune informazioni sulle prestazioni di GP e BB e dei corrispettivi sistemi gerarchici. Sia HW[GP] che HW[BB] migliorano le prestazioni dei corrispettivi pianificatori non gerarchici. LPG, invece, non mostra nessuna fase di transizione poiché riesce a risolvere in pochissimi secondi problemi particolarmente complessi.



(a) (b)

Figura 39 - Esempio di problema per il dominio *gripper*, (a) stato iniziale (b) stato finale.

III.1.5 Test sul Dominio *Zeno-Travel*

Il dominio *zeno-travel* è costituito da persone che viaggiano adoperando aeroplani in grado di effettuare movimenti lenti o veloci (vedi Figura 41), i movimenti veloci consumano un quantitativo di carburante maggiore rispetto ai movimenti lenti.

In questo dominio, si osserva un miglioramento di HW[BB] rispetto a BB, simile a quello ottenuto nel caso del mondo dei blocchi (vedi Figura 42). Purtroppo, né GP né HW[GP] sono risultati in

grado di risolvere i problemi di questo dominio. Per quanto riguarda LPG, invece, esso è in grado di risolvere piani lunghi in pochi secondi al massimo; questo vanifica l'uso dell'astrazione in questo dominio con questo pianificatore.

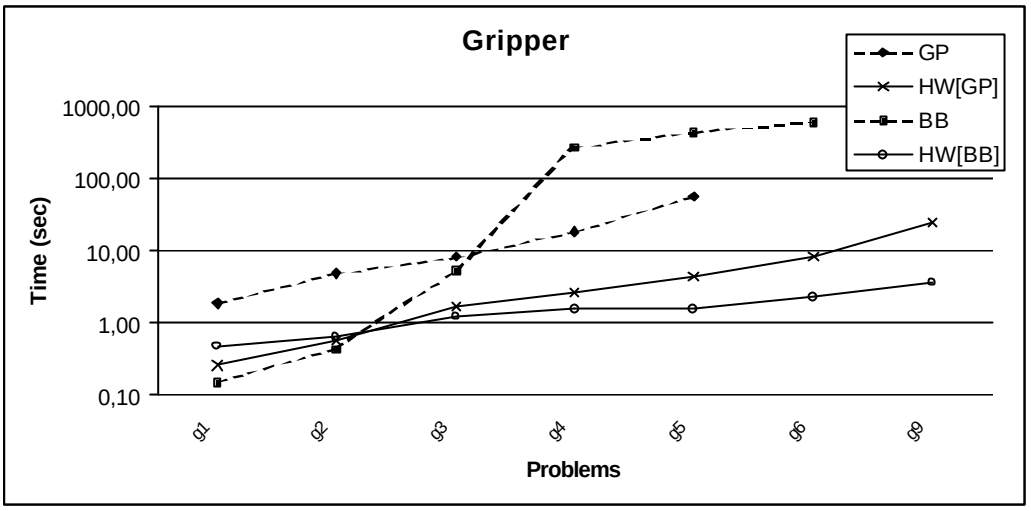
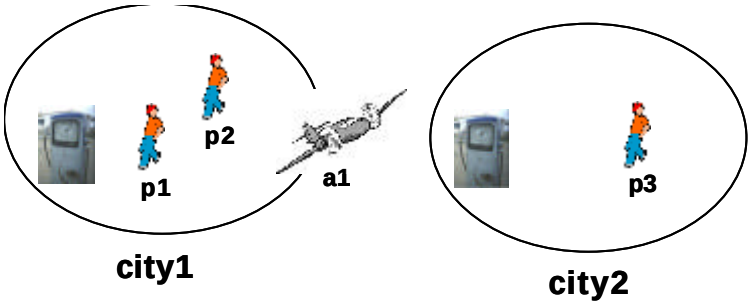
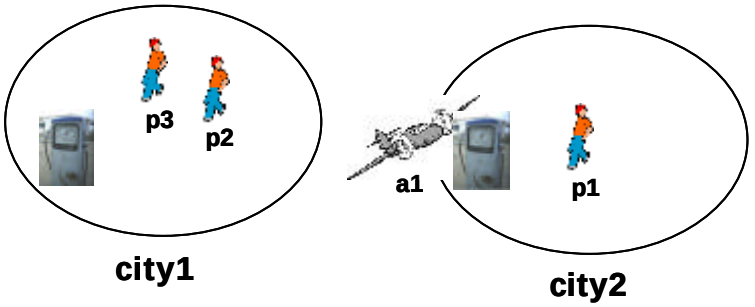


Figura 40 - Comparazione dei tempi di CPU per il dominio *gripper*.



(a)



(b)

Figura 41 - Esempio di problema per il dominio *zeno-travel*, (a) stato iniziale (b) stato finale.

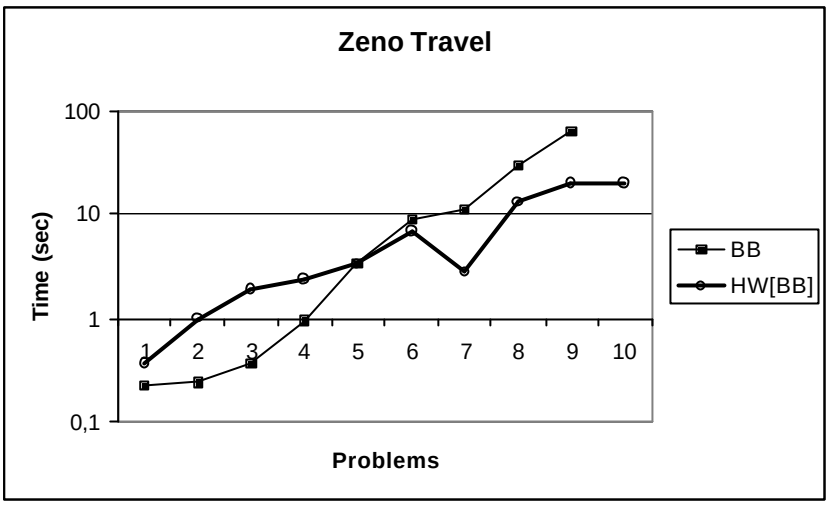


Figura 42 - Comparazione dei tempi di CPU per il dominio *zeno-travel*.

IV SVILUPPI FUTURI

I risultati sperimentali hanno dimostrato che, per problemi di una certa complessità, l'uso di tecniche di astrazione migliora le prestazioni dei pianificatori. Le argomentazioni presentate in quest'ultima parte della tesi, sono volte a sperimentare nuovi modelli per lo sviluppo di sistemi software ad elevata flessibilità. Il lavoro presentato presenta, infatti, delle interessanti caratteristiche che meritano di essere ulteriormente indagate al fine di arrivare ad un sistema ad agenti il più completo e flessibile possibile.

Il primo passo da fare in questa direzione è dunque quello di incorporare il pianificatore parametrico proposto nell'architettura di un agente, in modo da dotare l'agente di un comportamento proattivo più flessibile, tale da consentirgli di risolvere in maniera efficiente problemi complessi. Da notare, inoltre, che negli esperimenti presentati sono stati presi in considerazione solamente domini classici di pianificazione, ma il sistema può, in maniera semplice, essere adattato anche a domini non classici (per esempio dinamici).

Inoltre, attualmente la generazione di gerarchie di astrazione è fatta a mano, stiamo però lavorando alla generazione automatica di gerarchie di astrazione basata sull'astrazione di macro-operatori e successivamente, operando astrazioni su predicati e tipi, di operatori astratti.

Infine, un approccio alternativo, anch'esso in fase di studio ma che esula dagli scopi di questa tesi, riguarda la sperimentazione di tecniche di astrazione per la realizzazione di un pianificatore gerarchico e parametrico realizzato attraverso l'interazione di un insieme di agenti che coesistono in un sistema multi-agente ([Armano02]).

CONCLUSIONI

In questo lavoro di tesi sono stati indagati aspetti di ricerca legati a modelli di computazione innovativi, per lo sviluppo di sistemi software ad elevata flessibilità. In particolare, una volta inquadrato il problema nel contesto dell'ingegneria del software, è stato approfondito il paradigma ad agenti, considerato da più parti come paradigma emergente per la specifica e la realizzazione di sistemi complessi.

Nella prima parte della tesi, sono stati individuati e illustrati due tra gli aspetti principali che consentono di ottenere modelli di computazione ad elevata flessibilità: le capacità proattiva e adattativa.

Nella seconda parte della tesi, nell'ambito dello studio della proattività, è stata definita un'architettura ad agenti che fa uso di tecniche di astrazione e che alterna pianificazione ed esecuzione per risolvere problemi complessi in un dominio ispirato al mondo reale. In questo ambito, è stato messo in evidenza il ruolo centrale giocato dal modulo di pianificazione (HIPE). Nell'ambito dello studio della adattatività, è stato definito un algoritmo di apprendimento in grado di fare evolvere il comportamento di un agente sulla base dei problemi che gli sono stati proposti in passato. L'architettura complessiva, che incorpora in ogni agente un modulo proattivo e uno adattativo, è stata quindi collaudata in un videogioco caratterizzato da utenti che interagiscono con un mondo virtuale popolato da agenti eterogenei.

Nella terza parte della tesi, infine, è stata studiata la possibilità di rendere il comportamento proattivo di un agente indipendente dal particolare pianificatore utilizzato per la ricerca nello spazio degli stati, aumentando ancora il grado di flessibilità del modello indagato. Un'implementazione di tale modello è stata collaudata su domini classici di pianificazione, mostrando dei risultati incoraggianti.

BIBLIOGRAFIA

[Ambros-Ingerson88] J.A. Ambros-Ingerson, and S. Steel. Integrating planning, execution and monitoring. In *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-88).* St. Paul, MN, (Menlo Park, CA: AAAI Press), pp. 83-88, 1988.

[Armano00a] G. Armano, and E. Vargiu. Implementing Autonomous reactive Agents by using Active Objects. In *Proceedings of WOA: Workshop "Dagli Oggetti agli Agenti: Tendenze Evolutive dei Sistemi Software"*, pp. 35-40, Parma, 29-30 Maggio 2000.

[Armano00b] G. Armano, V. Galaffu, C. Muntoni, and E. Vargiu. A Layered Architecture for Implementing Autonomous Planning Agents. In *Proceedings of AI*IA2000*, pp. 5-8, Milano 13-15 Settembre 2000.

[Armano00c] G. Armano, and E. Vargiu. Using Autonomous Agents to Implement Avatars for a Computer Game. In *Proceedings of AICA2000*, Taormina 27-30 Settembre 2000.

[Armano01a] G. Armano, G. Cherchi, and E. Vargiu. An Agent Architecture for Planning in a Dynamic Environment. In *Proceedings of AI*IA 2001*, 25-28 Settembre 2001, Bari, Italy.

[Armano01b] G. Armano, and E. Vargiu. An Adaptive Approach for Planning in Dynamic Environments. In *Proceedings of IC-AI'2001, the 2001 International Conference on Artificial Intelligence*, Las Vegas, Nevada, USA, June 25-28, 2001.

[Armano01c] G. Armano, G. Cherchi, and E. Vargiu. Implementing Adaptive Capabilities on Agents that Act in a Dynamic Environment, In *Proceeding of WOA 2001*, 4-5 Settembre 2001, Modena, Italy.

[Armano02] G. Armano, G. Cherchi, and E. Vargiu. Experimenting Abstraction Mechanisms through an Agent-Based Hierarchical Planner. In *Proceedings of WOA 2002 (Dagli Oggetti agli Agenti, dall'Informazione alla Conoscenza)*, Milano 18-19 Novembre 2002.

[Armano03a] G. Armano, G. Cherchi, and E. Vargiu. Experimenting the Performance of Abstraction Mechanisms through a Parametric Hierarchical Planner. *IASTED*

International Conference on Artificial Intelligence and Applications (AIA'03), in corso di pubblicazione.

[Austin62] J.L. Austin. *How to do Things with Words*. Cambridge, MA, Havard University Press, 1962.

[Bacchus94] F. Bacchus, and Q. Yang. Downward refinement and the efficiency of Hierarchical Problem Solving. *Artificial Intelligence*, vol. 71(1), pp. 43-101, 1994.

[Bacchus00] F. Bacchus. *Results of the AIPS 2000 Planning Competition*. 2000. Url: http://www.cs.toronto.edu/aips2000.

[Blum97] A. Blum, and M. Furst, Fast planning through plan-graph analysis. *Artificial Intelligence*, 90:281-300, 1997.

[Booch94] G. Booch. *Object-Oriented Analysis and Design (second edition)*. Addison-Wesley: Reading, MA, 1994.

[Bratman87] M. E. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press: Cambridge, MA, 1987.

[Bratman88] M. E. Bratman, D. J. Isreal, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(4), 1988.

[Brooks86] R.A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2(1):14--23, March 1986.

[Carbonell90] J.G. Carbonell, C.A. Knoblock, and S. Minton. *PRODIGY: An Integrated Architecture for Planning and Learning*, in K. VanLehn (ed.), Architectures for intelligence, pp. 241-278, Lawrence Erlbaum Associates, Hillsdale, NJ, 1991.

[Carriero92] N. Carriero, and D. Gelernter. Coordination Languages and their Significance. *Communications of the ACM*, vol. 35(2), pp. 97-107. 1992.

[Castelfranchi94] C. Castelfranchi, *Guarantees for Autonomy in Cognitive Agent Architecture*. In M. Wooldridge, and N.R. Jennings (eds.) Intelligent Agents. Lecture Notes in AI, Vol. 890, pp. 56-70, Springer, 1995.

[Ciancarini97] P. Ciancarini. Coordination Models, Languages, Architectures and Applications: a Personal Perspective. University of Leuven, 1997.

[Ciancarini01] P. Ciancarini, and M. Wooldridge. *Agent-Oriented Software Engineering*. Springer-Verlag Lecture Notes in AI Volume 1957, January 2001.

[Draper94] D. Draper, S. Hanks, and D. Weld Probabilistic Planning with Information Gathering and Contingent Execution. *Proceedings of the Second International Conference on AI Planning Systems (AIPS-94)*, pp. 32-36. 1994.

[Erol94a] K. Erol, J.A. Hendler, and D.S. Nau. HTN Planning: Complexity and Expressivity. Proceedings of the 11th National Conference on AI (AAAI-94). AAAI Press, 1994.

[Erol94b] K. Erol, J. Hendler, and D.S. Nau. UMCP: A sound and complete procedure for hierarchical task{network planning. *Artificial Intelligence Planning Systems*, pp. 249-254, 1994.

[Etzioni90] O. Etzioni. *A Structural Theory of Explanation-Based Learning*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1990.

[Ferguson92] I. A. Ferguson, *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents.* Ph.D. thesis, Comput. Sci. Lab., Univ. Cambridge, U.K., 1992.

[Fikes71] R. Fikes, N. Nilsson, STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2, pp. 189-203, 1971.

[Finin94] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an agent communication language. *Proceedings of the 3rd International Conference on Information and Knowledge Management*, pp 456-463, Gaithersburg, Maryland, ACM Press, 1994.

[FIPA] FIPA web-site: http://www.fipa.org/

[Georgeff87] M. Georgeff, and A. Lansky, Reactive reasoning and planning. *Proceedings of the 6th National Conference on Artificial Intelligence*, *pp.* 677-682, Morgan Kaufmann, 1987.

[Gerevini02] A. Gerevini, and I. Serina. LPG: a Planner based on Local Search for Planning Graphs. In *Proceedings of the Sixth Int. Conference on AI Planning and Scheduling (AIPS'02)*, AAAI Press, 2002.

[Ghallab98] M. Ghallab, et al. *PDDL --- The Planning Domain Definition Language (Version 1.2).* Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computaional Vision and Control, October 1998.

[Giunchiglia90] F. Giunchiglia, and T. Walsh. A Theory of Abstraction. Technical Report 9001-14, IRST, Trento (Italy), 1990.

[Golden96] K. Golden, O. Etzioni, and D. Weld. XII: Planning with universal quantification and incomplete information. In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning, KR'9J,* 1994.

[Green69] C. Green. Application of theorem proving to problem solving. In *Proceedings of 1st International Joint Conference on Artificial Intelligence (IJCAI-69),* Washington, DC, May 7-9, 1969.

[Haigh96a] K. Z. Haigh, and M. Veloso. Planning with Incomplete Information for Robot Problems. AAAI Spring Symposium 1996, pp. 35-44, AAAI Press, Menlo Park, California, March 1996.

[Haigh96b] K.Z. Haigh, and M. Veloso. Interleaving Planning and Robot Execution for Asynchronous User Requests. In *Proceedings of the AAAI-96 Spring symposium ``Planning with Incomplete Information for Robot Problems'',* March 1996. Stanford, California.

[Kambhampati97] S. Kambhampati. Refinement Planning as a Unifying Framework for Plan Synthesis. *AI Magazine,* 18(2), pp. 67-97, 1997.

[Kautz96] H. Kautz, and B. Selman. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96),* Portland, OR, August 1996.

[Kautz98] H. Kautz, and B. Selman. BLACKBOX: A new approach to the Application of Theorem Proving to Problem Solving. In *Working notes of the Workshop on Planning as Combinatorial Search,* 1998.

[Knoblock91a] C.A. Knoblock. Search reduction in hierarchical problem solving. Proceedings of the 9th National Conference on Artificial Intelligence, pp. 686-691, AAAI Press, Menlo Park, CA, 1991.

[Knoblock91b] C.A. Knoblock. *Automatically generating abstractions for problem solving.* PhD Thesis, School of Computer Science, Carnegie Mellon University, 1991.

[Knoblock91c] C.A. Knoblock, J.D. Tenenberg, and Q. Yang. Characterizing Abstraction Hierarchies for Planning, *Proceedings of the 9th National Conference on Artificial Intelligence,* pp. 692-697, 1991.

[Knoblock94] C.A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence,* vol. 68, pp. 243-302, 1994.

[Koehler00] J. Koehler, and K. Schuster. Elevator Control as a Planning Problem, *Proceedings of the 5th International Conference on AI Planning ad Scheduling*, pp. 331-338, AAAI Press, Menlo Park, 2000.

[Korf87] R.E. Korf. Macro-operators: A Weak Method for Learning. Artificial Intelligence, vol. 33(1), pp. 65-88, 1987.

[Kushmerick95] N. Kushmerick, S. Hanks, and D. Weld. An Algorithm for Probabilistic Planning. *Artificial Intelligence*, 76, 239-286. 1995.

[Laird87] J. Laird, A. Newell, P. Rosenbloom, P. SOAR: an architecture for general intelligence. *Artificial Intelligence*, vol. 33, 1-64, 1987.

[Laird91] J. Laird, M. Hucka, S. Huffman. *An Analysis of SOAR as an Integrated Architecture*. SIGART Bulletin 2, 85-90, 1991.

[Long98] D. Long. The AIPS-98 Planning Competition. *AI Magazine*. Vol. 21(2) 13-3, 2000.

[Long02] D. Long. *Results of the AIPS 2002 Planning Competition*. 2002. Url: http://www.dur.ac.uk/d.p.long/competition.html.

[Malone94] T. W. Malone, and K. Crowston, The Interdisciplinary Study of Co-ordination, *ACM Computing Surveys*, vol. 26, pp. 87-120, N1, March 1994.

[McAllester91] D. McAllester, and D. Rosenblitt, D. Systematic nonlinear planning. In Proceedings of AAAI-91, pp. 634-639, 1991.

[McCarthy69] J. McCarthy, and P. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. *Machine Intelligence*, 4, Edinburgh University Press, 1969.

[Meyer97] B. Meyer. *Object-oriented software construction*. Prentice-Hall, Inc., Upper Saddle River, NJ, 1997.

[Muller97] J. Muller. A cooperation model for autonomous agents. In J. Muller, M. Wooldridge and N. Jennings, eds, *Intelligent Agents III - Proceedings of the 3rd International Workshop on Agent Theories, Architectures, and Languages (ATAL96)*, pp. 135-147, Budapest, Hungary, 1996.

[Nareyek98] A. Nareyek. A Planning Model for Agents in Dynamic and Uncertain Real-Time Environments. In *Proceedings of the Workshop on Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments at the 4th International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, AAAI Press, Menlo Park, California, 1998.

[Newell90] A. Newell. *Unified Theories of Cognition*. Harvard University Press. Cambridge, Massachusetts, 1990.

[Nourbakhsh97a] I. Nourbakhsh. *Interleaving Planning and Execution for Autonomous Robots*. Dordrecht, Netherlands: Kluwer Academic. PhD thesis, Department of Computer Science, Stanford University, Stanford, CA, 1997.

[Nourbakhsh97b] I. Nourbakhsh. *Interleaving Planning and Execution for Autonomous Robots*. Kluwer Academic Publishers, 1997.

[Omicini01] A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf. *Coordination of Internet Agents: Models, Technologies and Applications* (eds.). Springer-Verlag, Berlin, Aug. 2001.

[Papadopoulos98] G. A. Papadopoulos, and F. Arbab. Coordination Models and Languages. *Advances in Computers*, vol. 46, 329-400, 1998.

[Pednault89] E.P.D. Pednault. ADL: Exploring the middle ground between strips and the situation calculus. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, pp. 324--332, 1989.

[Penberthy92] S. Penberthy, and D. Weld. UCPOP: A sound, complete partial order planner for ADL. In *Proceedings of the Knowledge Representation Conference*, 1991.

[Pryor96] Pryor, L. and G. Collins. Planning for Contingencies: A Decision-Based Approach. *Journal of Artificial Intelligence Research* 4, 287-339, 1996.

[Rosembloom89] P. Rosenbloom. *A Symbolic Goal-Oriented Perspective on Connectionism and SOAR*. In R. Pfeifer, Z. Schreter, F. Fogelman-Soulie, L. Steels (eds.), Connectionism in Perspective. pp. 245-263. Amsterdam, The Netherlands: Elsevier Science Publications B.V., 1989.

[Russell95] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[Sacerdoti74] E. D. Sacerdoti. Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence*, Vol. 5, pp. 115-135, 1974.

[Simon83] H. Simon. Why should machines learn? In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, chapter 2, pp. 25-37. Tioga Publishing Comp., Palo Alto, CA, 1983.

[Szyperski97] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison Wesley, 1997.

[Tenenberg88] J.D. Tenenberg. Abstraction in Planning. PhD Thesis, Computer Science Department, University of Rochester, 1988.

[Veloso95] M. Veloso, J. Carbonell, A. Perez, D. Borrajo, E. Fink, and J. Blythe. Integrating Planning and Learning: The PRODIGY Architecture. *Journal of Experimental and theoretical Artificial Intelligence*, 7(1), pp. 81-120, 1995.

[Wang94] X. Wang. Learning Planning Operators by Observation and Practice. *Proceedings of the 2nd International Conference on AI Planning Systems, AIPS-94*, pp. 335-340, Chicago, IL, 1994.

[Weld94] D. S. Weld. An Introduction to Least Commitment Planning. *AI Magazine*, 1994.

[Wooldridge95] M. Wooldridge, and N.R Jennings. Intelligence Agents: Theory and Practice. *Knowledge Engineering Review*, 10(2), pp. 115-52, 1995.

[Wooldridge99] M. Wooldridge. *Intelligent Agents.* In Weiss, G. (ed.), Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, MIT Press, MA, pp.27-78, 1999.

[Yang96] Q. Yang, J. Tenenberg, and S. Woods. On the implementation and evaluation of ABTWEAK. *Computational Intelligence*, vol. 12(2) pp. 295--318, 1996.