UNIVERSITÀ DELLA CALABRIA

Dipartimento di Elettronica,
Informatica e Sistemistica

Dottorato di Ricerca in
Ingegneria dei Sistemi e Informatica
XX ciclo

Tesi di Dottorato

Tecniche e algoritmi per la ripianificazione in ambienti dinamici

Fabio Palopoli

UNIVERSITÀ DELLA CALABRIA

Dipartimento di Elettronica,
Informatica e Sistemistica

Dottorato di Ricerca in
Ingegneria dei Sistemi e Informatica
XX ciclo

Tesi di Dottorato

Tecniche ed algoritmi per la ripianificazione in ambienti dinamici

Fabio Palopoli

Fabio Palopoli

Coordinatore
Prof. Domenico Talia

Supervisore
Prof. Sergio Greco

Sergio Greco

DEIS

DEIS- DIPARTIMENTO DI ELETTRONICA, INFORMATICA E SISTEMISTICA
Novembre 2008

Settore Scientifico Disciplinare: ING-INF/05

Sommario

In un ambiente dinamico, lo stato del sistema cambia al trascorrere del tempo, anche indipendentemente dalle azioni eseguite dall'agente che è in esso collocato. In particolare, un ambiente dinamico può cambiare mentre l'agente sta deliberando. Sia s_0 lo stato del sistema al tempo t_0. Si supponga che l'agente inizi a ragionare al tempo t_0 sullo stato del sistema s_0, al fine di selezionare un'azione ottima da eseguire. Si supponga inoltre che il processo di reasoning avviato dall'agente al tempo t_0 termini al tempo t_1. Sia s_1 lo stato del sistema al tempo t_1. Se l'ambiente è dinamico, lo stato s_1 potrebbe essere sostanzialmente diverso da s_0, e l'azione selezionata dall'agente, benché ottima in funzione dello stato s_0, potrebbe non esserlo in funzione dello stato s_1. Queste considerazioni hanno portato diversi autori a mettere in dubbio l'efficienza e l'efficacia delle architetture *logic based* in ambienti dinamici. Ciò ha portato all'introduzione di nuovi modelli e architetture, come quelle reattive, e all'introduzione di quelle ibride.

In questo contesto, abbiamo proposto un nuovo modello astratto, basato su Ipersfera di Approssimazione e Predizione, che si configura come sotto-classe dei modelli ibridi stratificati orizzontalmente. L'agente è caratterizzato da due moduli di decision making, il *deep reasoning module* e il *fast reasoning module*. Il deep reasoning module tenta di calcolare la deliberazione ottima in funzione del (futuro) stato del sistema nell'istante in cui dovrebbero essere eseguite le azioni selezionate grazie al processo di reasoning, e non più in funzione dello stato del sistema nell'istante in cui il processo di reasoning inizia; durante l'elaborazione del deep reasoning module, l'agente esegue le azioni indicate dal fast reasoning module. E' stata introdotta una nuova struttura, l'Ipersfera di Approssimazione, che consente tolleranza sulla precisione dell'output del processo di predizione. Le caratteristiche del modello HPA sono le seguenti: *i)* non presenta i tipici rischi e inefficienze legati ad elaborazioni troppo lunghe da parte del deep reasoning module, in quanto il controllo, fin quando il processo di deep reasoning non termina ed è al contempo stata generata una corretta predizione, è demandato al modulo di fast reasoning; *ii)* è implicitamente adattativo, in quanto maggiore è il grado di dinamismo dell'ambiente, meno spesso il controllo sarà demandato al modulo di deep reasoning, minore è il grado di dinamismo dell'ambiente, più spesso il controllo sarà automaticamente assegnato al modulo di deep reasoning; *iii)* consente la selezione di azioni in funzione del (futuro) stato del sistema nell'istante in cui esse saranno eseguite, come già accennato; *iv)* permette uno *switch* di controllo implicito e senza costi aggiuntivi tra i due moduli di decision making.

Il modello astratto è stato successivamente istanziato al caso del *plan fixing*; in questo contesto, viene messo in evidenza una possibile mediazione tra un comportamento reattivo e un comportamento orientato alla ri-pianificazione. Altra

possibile applicazione specifica che si basa su predizione è quella del *plan recovering*, anch'essa trattata nel contesto delle attività di dottorato.

Infine, le tecniche di ripianificazione sono state applicate nell'ambito di un container terminal. Quest'ultima attività è stata svolta del contesto del progetto di ricerca Promis.

Rende (CS), *Fabio Palopoli*
Novembre 2008

Ringraziamenti

Alla fine di questo lavoro di tesi, ringrazio il prof. Sergio De Julio, presidente di Exeura s.r.l., l'azienda per la quale lavoro, che ha facilitato in varie forme le mie attività di dottorando. Ringrazio il prof. Sergio Greco, il supervisore delle mie attività di dottorato di ricerca, per la guida attenta e la sua disponibilità. Ringrazio inoltre il prof. Domenico Talia, coordinatore del Dottorato di Ricerca in Ingengeria Informatica, per i preziosi consigli. Grazie, infine, a Teresa e alla mia famiglia, per gli incoraggiamenti e il supporto di questi anni.

Indice

Parte I

Stato dell’arte

1

Gli Agenti Intelligenti

Sommario: In questo capitolo si propone una panoramica sullo stato dell'arte riguardante gli agenti intelligenti.

1.1 Introduzione

Sembra opportuno, trattando questo capitolo di agenti intelligenti, proporre innanzitutto una definizione del termine agente. Sfortunatamente, in ambito accademico, come del resto in ambito industriale, non vi è un'opinione unanime sul significato di tale termine. La ragione principale di ciò è legata all'etimologia della parola: "Agente" (o, in lingua inglese, "Agent") deriva dal vocabolo latino *Agens*, participio presente del verbo *Agĕre*, cioè *"L'entità che agisce"*, ovvero, *"L'entità che compie azione/i"*. Sembra superfluo a questo punto argomentare ulteriormente sulla generalità del termine, e di conseguenza, sulla mancanza di una opinione condivisa sul suo significato. E, anzi, le varie definizioni proposte in letteratura sono figlie di una contestualizzazione più o meno marcata del termine. Ad esempio, Russell e Norvig, pur perseguendo chiaramente l'obiettivo di fornire una definizione astratta, restringono il dominio applicativo nel quale si configurano gli agenti. In [RN03], infatti, asseriscono che *Un agente può essere concepito come qualunque cosa in grado di percepire il proprio ambiente attraverso dei sensori e di agire in quell'ambiente attraverso degli attuatori*. E' interessante notare alcuni elementi di novità rispetto alla definizione precedente: innanzitutto l'agente viene visto adesso come un'entità inglobata in un generico ambiente, il quale rappresenta una delle fonti della conoscenza a sua disposizione (l'altra fonte può essere informalmente riferita come conoscenza "innata"). L'ambiente è inoltre lo scenario all'interno del quale esso compie azioni. Ogni agente può essere rappresentato come la composizione di vari moduli. Le categorie più comuni di questi ultimi sono le seguenti:

- *Moduli di input o sensori*:
  - Per un agente umano, gli occhi, le orecchie e, più in generale, gli organi di senso;
  - Per un robot, telecamere, sensori di distanza, GPS, etc.;
  - Per un agente software, ad esempio, stringhe di bit;
- *Moduli di output o attuatori*:
  - Per un agente umano, le gambe, le mani, etc.;
  - Per un robot, ad esempio, motori e braccia meccaniche;

  - Per un agente software, ancora una volta, stringhe di bit;

- *Moduli di decisione* (che saranno oggetto di discussione di gran parte del capitolo).

La figura 1 sintetizza la definizione proposta da Russell e Norvig:

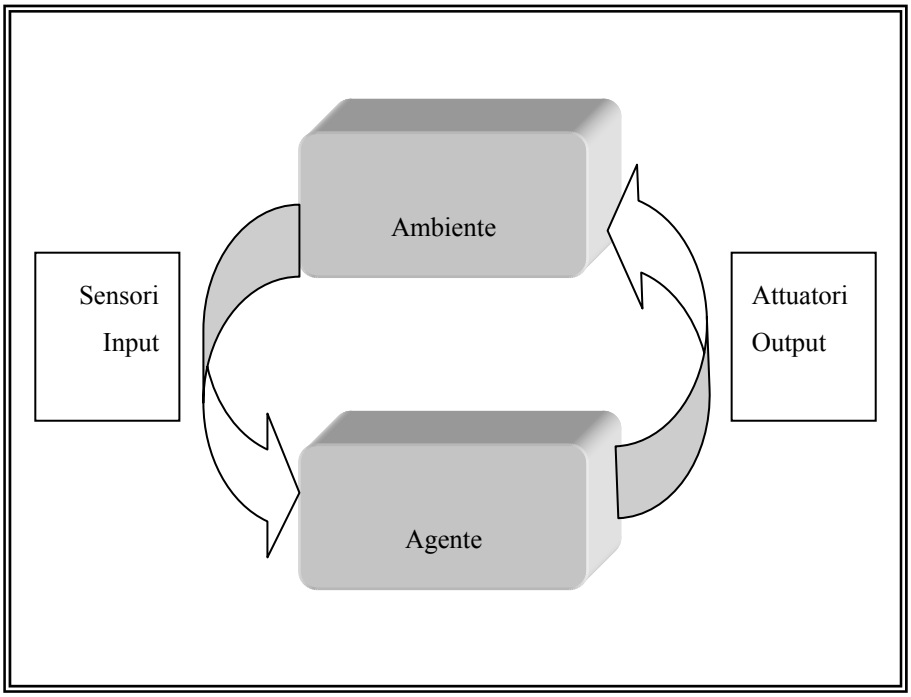

Figura 1: Interazione tra agente e ambiente

Si rimanda al paragrafo 1.2 per un approfondimento del concetto di ambiente.

Wooldridge e Jennings [JW95] definiscono un agente come *un sistema di elaborazione che è collocato in un qualche ambiente, e che può eseguire in esso azioni autonome, al fine di raggiungere gli obiettivi per i quali è stato progettato*.

Anche in questo caso è possibile notare elementi di novità rispetto alle definizioni precedenti. In primo luogo, il dominio è ulteriormente ristretto ai sistemi di elaborazione, e ciò, se da un lato toglie la generalità in precedenza ottenuta, dall'altro focalizza meglio gli obiettivi di questo capitolo. La definizione appena proposta introduce inoltre il concetto di autonomia, anch'esso molto dibattuto in letteratura. In questa sede si considereranno autonome quelle entità capaci di eseguire azioni senza l'intervento e le direttive di uomini o di altri sistemi. Gli agenti, quindi, hanno diretto controllo sul loro stato interno e sul loro comportamento. Essi possono inoltre controllare l'ambiente, ma, molto spesso, solo parzialmente, esercitando quindi semplicemente un'influenza su esso.

Questa distinzione suggerisce già una prima possibile classificazione degli agenti. Un agente che ha una percezione completa dello stato dell'ambiente, potrà predire in anticipo in quale stato si troverà il sistema subito dopo l'esecuzione dell'azione scelta. Tale tipo di agente è detto *onnisciente*. Viceversa, se un agente non ha una percezione completa dello stato reale dell'ambiente, ed esegue più volte la stessa azione in condizioni ambientali *apparentemente* simili, potrebbe ottenere di volta in volta effetti diversi. In particolare, l'agente potrebbe fallire. Tale agente è detto *non onnisciente*. E' interessante notare che, dal punto di vista di quest'ultimo, anche ambienti di fatto deterministici possono sembrare non deterministici.

Prima di procedere con un'analisi puntale delle proprietà di un agente, sembra opportuno mettere l'accento sulle caratteristiche distintive dell'ambiente.

1.2 L'ambiente

Secondo Russell e Norvig [RN03], "gli ambienti possono essere di diversi tipi. Le distinzioni principali da fare sono le seguenti:

- *Accessibile* vs *inaccessibile*.

Se l'apparato sensorio di un agente gli conferisce accesso allo stato completo dell'ambiente, allora diciamo che l'ambiente è accessibile a quell'agente. Un ambiente è realmente accessibile se i sensori rilevano tutti gli aspetti che sono importanti per la scelta dell'azione. Un ambiente accessibile è conveniente perché l'agente non ha bisogno di mantenere alcuno stato interno per tenere conto del mondo.

- *Deterministico* vs *non deterministico.*

Se lo stato successivo dell'ambiente è completamente determinato dallo stato attuale e dalle azioni selezionate dagli agenti, allora diciamo che l'ambiente è deterministico. In teoria, un agente non deve preoccuparsi dell'incertezza in un ambiente accessibile e deterministico. Se l'ambiente è inaccessibile, tuttavia, può sembrare che sia non deterministico. Ciò vero specialmente se l'ambiente è complesso e diventa difficile tenere conto di tutti gli aspetti inaccessibili. Perciò, spesso è meglio pensare un ambiente come deterministico o non deterministico dal punto di vista dell'agente.

- *Episodico* vs *non episodico*.

In un ambiente episodico l'esperienza dell'agente è divisa in "episodi". Ogni episodio è dato da un agente che percepisce e poi agisce. La qualità delle sue azioni dipende solo dall'episodio stesso, perché gli episodi successivi non dipendono da quali azioni accadono negli episodi precedenti. Gli ambienti episodici sono molto più semplici poiché l'agente non deve fare previsioni.

- *Statico* vs *dinamico*.

Se l'ambiente può cambiare mentre un agente sta deliberando, allora diciamo che l'ambiente è dinamico per quell'agente; altrimenti è statico. Gli ambienti statici sono

facili da trattare perché l'agente non ha bisogno di continuare a guardare il mondo mentre sta decidendo un'azione, né ha bisogno di preoccuparsi del trascorrere del tempo. Se l'ambiente non cambia con il trascorrere del tempo, ma cambia soltanto il valore delle prestazioni dell'agente, allora diciamo che l'ambiente è *semidinamico*.

- *Discreto* vs *continuo*.

Se ci sono un numero limitato di percezioni ed azioni distinte e definite, chiaramente, diciamo che l'ambiente è discreto. Il gioco dagli scacchi è discreto – ci sono un numero fissato di mosse possibili ad ogni turno. Guidare un taxi è continuo – la velocità e la posizione del taxi e degli altri veicoli varia in un intervallo di valori continui."

1.3 Le caratteristiche di un agente intelligente

E' possibile individuare diverse proprietà che caratterizzano gli agenti intelligenti, oltre a quelle descritte nell'introduzione di questo capitolo (in particolare, ci si riferisce all'autonomia). Se ne propongono alcune:

- *Razionalità*: un agente intelligente eseguirà ogni azione possibile per raggiungere i propri obiettivi, e non agirà mai in modo tale da pregiudicare a suo giudizio il raggiungimento di tali obiettivi.
- *Reattività*: come detto, gli agenti sono in grado di percepire (spesso solo parzialmente) lo stato dell'ambiente, e quindi anche i cambiamenti avvenuti o in atto. Gli agenti intelligenti si dicono reattivi se sono in grado di eseguire azioni in risposta a tali cambiamenti (ovviamente solo nel caso in cui tale reazione sia coerente con il perseguimento degli obiettivi per i quali è stato progettato).
- *Proattività*: gli agenti intelligenti possono prendere iniziativa per raggiungere gli obiettivi per i quali sono stati progettati.
- *Abilità sociale*: gli agenti intelligenti sono in grado di interagire con altri agenti (anche umani).

Altre proprietà sono le seguenti:

- *Mobilità*: alcuni agenti intelligenti sono in grado di spostarsi all'interno dell'ambiente in cui sono collocati (ad esempio, per agenti software, ciò potrebbe significare potersi spostare tra un punto e un altro di una rete).
- *Veracità*: se l'agente è caratterizzato da questa proprietà, l'insieme delle azioni a sua disposizione non include quella di comunicare consapevolmente informazioni false.
- *Benevolenza*: se presente, gli agenti non hanno obiettivi contrastanti, e cercano sempre di concedere quanto gli viene chiesto.
- *Capacità di apprendimento*: in tal caso, l'agente intelligente è in grado di migliorare le proprie *performance* grazie alle proprie esperienze.

1.4 Generica architettura di un agente intelligente

In questo paragrafo verrà descritta la generica architettura di un agente intelligente.

Un *agente standard* può essere rappresentato dalla funzione:

$azione : S^* \rightarrow A$

dove l'insieme $S = \{s_1, s_2, \ldots\}$ identifica tutti i possibili stati dell'ambiente, ed $A = \{a_1, a_2, \ldots\}$ rappresenta l'insieme delle azioni eseguibili dall'agente.

Quindi un agente standard decide quale azione eseguire sulla base anche della sua esperienza, che è rappresentata da una sequenza di stati.

Il *comportamento* di un ambiente può essere modellato con la seguente funzione:

$env : S \times A \rightarrow \rho(S)$

che riceve in input uno stato $s \in S$ (lo stato corrente dell'ambiente) una azione $a \in A$ (l'azione eseguita dall'agente nell'ambiente con stato s) e restituisce in output un insieme di stati $\rho(S)$, ovvero l'insieme degli stati in cui si potrebbe trovare l'ambiente dopo l'esecuzione dell'azione a nello stato s. Se tutti i possibili insiemi ρ di *env* hanno cardinalità uguale a 1, allora l'ambiente *env* è deterministico e il suo comportamento di conseguenza può essere predetto con precisione.

L'interazione tra agente e ambiente può essere modellata per mezzo della *history* h, cioè una sequenza così definita:

$$h : s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots s_{u-1} \xrightarrow{a_{u-1}} s_u \xrightarrow{a_u} \ldots$$

dove s_0 denota lo stato iniziale dell'ambiente (cioè, ad esempio, lo stato in cui si trova l'ambiente nel momento in cui l'agente comincia a interagire con esso), a_u è la *u-esima* azione eseguita dall'agente, e s_u è l'*u-esimo* stato dell'ambiente, cioè uno tra gli stati in cui si potrebbe trovare l'ambiente dopo l'esecuzione dell'azione a_{u-1} nello stato s_{u-1}.

Sia $azione : S^* \rightarrow A$ un agente, $env : S \times A \rightarrow \rho(S)$ un ambiente, ed s_0 il suo stato iniziale. La sequenza

$$h : s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots s_{u-1} \xrightarrow{a_{u-1}} s_u \xrightarrow{a_u} \ldots$$

rappresenta una delle possibili *history* di tale agente nell'ambiente, se e solo se sono soddisfatte le seguenti condizioni:

(1) $\forall u \in N$, $a_u = azione(s_1, \ldots, s_u)$

(2) $\forall u \in N$ tale che $u > 0$, $s_u \in env(s_{u-1}, a_{u-1})$

Il comportamento caratteristico di un agente $azione : S^* \rightarrow A$ in un ambiente $env : S \times A \rightarrow \rho(S)$ è dato dall'insieme di tutte le history che soddisfano le proprietà 1 e 2.

Una proprietà ϕ è detta *invariante rispetto ad un ambiente env*, se è soddisfatta per tutte le possibili history dell'agente in *env*. Si definisce *hist(agent, env)* l'insieme di tutte le history di un agente in un ambiente. Due agenti $agent_1$ e $agent_2$ si definiscono equivalenti dal punto di vista comportamentale nel contesto di un determinato ambiente *env,* se e solo se $hist(agent_1, env) = hist(agent_2, env)$, e semplicemente equivalenti dal punto di vista comportamentale se $hist(agent_1, x) = hist(agent_2, x)$ per ogni possibile ambiente *x*.

Se si prendono in considerazione interazioni senza un limite temporale, le history in esame sono di lunghezza infinita.

Si esamineranno due possibili architetture astratte di agenti, classificate in base al fatto che tengano o non tengano conto delle history, nella scelta dell'azione da eseguire. Gli agenti appartenenti al primo gruppo sono noti come agenti con stato interno, gli altri come agenti puramente reattivi.

Prima di ciò, è utile introdurre il concetto di percezione.

1.4.1 Percezione

Per definire la funzione di decisione *azione*, è necessario rifinire l'architettura appena proposta. In questo paragrafo sarà illustrata un'architettura meno astratta, nella quale verrà inserito un modulo contenente due sottosistemi, sostituendo così il modello astratto proposto precedentemente. Verranno inseriti in particolare due moduli:

- Percezione
- Azione

A tal fine vengono introdotte due corrispondenti funzioni:

- La funzione *see,* che rappresenta la capacità dell'agente di osservare il suo ambiente; in un robot, tale funzione potrebbe essere implementata per mezzo di una telecamera o di un sensore a raggi infrarossi. In un agente software i sensori potrebbero essere costituiti da *parser* che ottengono informazioni da un testo. L'output della funzione *see* è chiamato *percezione*. Formalmente *see* si definisce come segue:

$see : S \rightarrow P$

dove P è un insieme non vuoto di percezioni. Tale funzione è definita su un insieme S di stati dell'ambiente S ed ha valori in P.

- La funzione *azione,* che, in questa sede, descrive il processo di *decision making* dell'agente. Alla luce di quanto detto, la funzione *azione* può essere ridefinita nel seguente modo:

$azione : P^* \rightarrow A.$

Azione è quindi definita su un insieme di sequenze di percezioni dello stato dell'ambiente (e quindi, non più semplicemente su un insieme di sequenze di stati) e ha valori nell'insieme A delle azioni che possono essere eseguite dall'agente.

La rappresentazione grafica dell'architettura proposta è mostrata nella seguente figura:

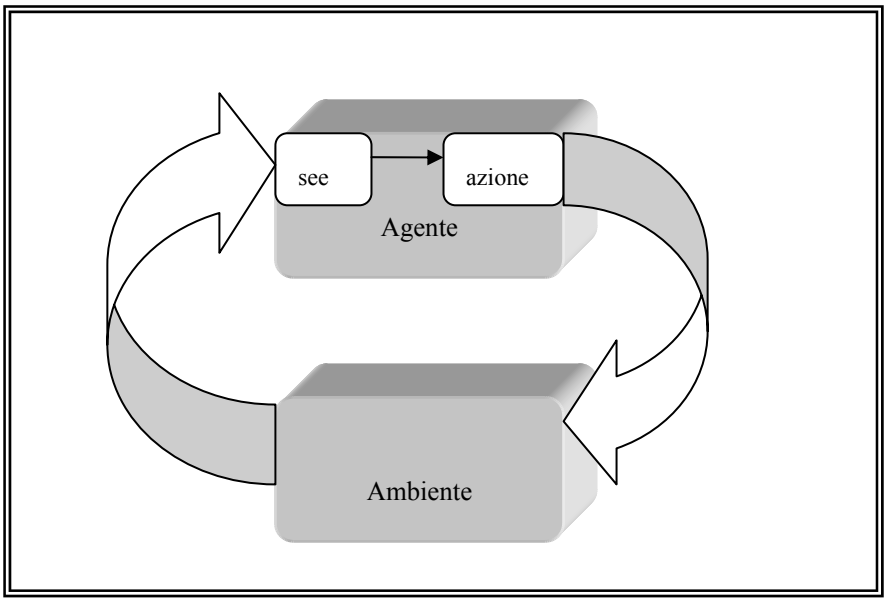

Figura 2: Agenti con funzioni di percezione

1.4.2 Agenti puramente reattivi

La caratteristica principale degli agenti puramente reattivi è quella di non tenere conto della loro history. Di conseguenza essi basano il processo di decisione esclusivamente sullo stato corrente dell'ambiente e non già sugli stati passati (e, ovviamente, quello attuale). Il comportamento di un agente puramente reattivo può quindi essere rappresentato formalmente da una funzione così definita:

$azione : S \rightarrow A$.

L'insieme degli agenti puramente reattivi è un sottinsieme di quello degli agenti standard.

1.4.3 Agenti con stato

In questo capitolo è stato definito un agente *standard* come una funzione che riceve in input una sequenza di stati e restituisce in output una azione. Questa rappresentazione, sebbene corretta, è contro-intuitiva. E' utile a questo punto sostituire il modello di agente standard con un modello equivalente, ma più naturale: oggetto di questo paragrafo è infatti l'architettura di un *agente con stato*. Si suppone che l'agente possa tenere traccia degli stati dell'ambiente, per mezzo di un qualche strumento di memorizzazione. Tutto ciò porta a un riadattamento delle funzioni prima presentate ed alla introduzione della funzione *next*.

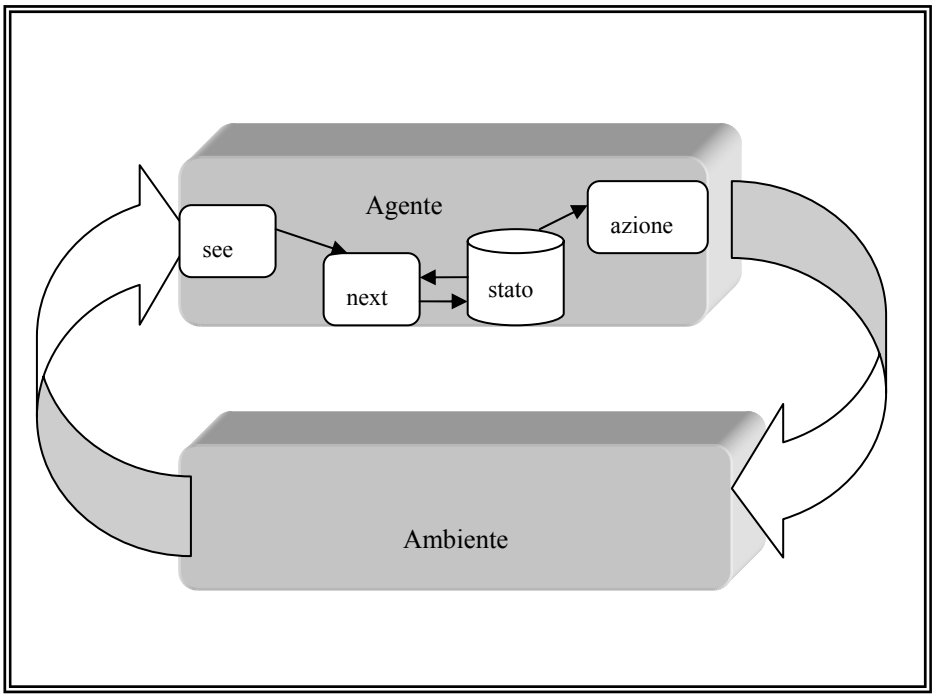

Figura 3: Agenti con stato

Si analizzano ora i componenti dell'architettura presentata in figura:

- La funzione *see* rimane invariata:

see : $S \rightarrow P$.

- La funzione di scelta dell'azione da eseguire, *azione*, viene ridefinita come segue:

azione : $I \rightarrow A$,

dove I rappresenta l'insieme degli stati interni dell'agente.

- La funzione *next* è formalmente definita come:

next : $I \times P \rightarrow I$.

Il comportamento di un agente basato su stati può essere implementato dal seguente algoritmo:

1. L'agente comincia a interagire con l'ambiente in un certo istante t_0. In tale istante lo stato (iniziale) interno è i_0.
2. L'agente osserva lo stato dell'ambiente s e genera una percezione *see*(s).

3. Lo stato interno dell'agente viene aggiornato per mezzo della funzione *next*, diventando dunque *next*(i_0, *see*(s)).
4. L'agente seleziona l'azione *azione*(*next*(i_0, *see*(s))).
5. GOTO 2.

1.5 Architetture concrete per agenti intelligenti

Le architetture presentate nel paragrafo precedente non danno alcuna indicazione sulla struttura dei singoli moduli. Non è chiaro, ad esempio, come possa essere rappresentato lo stato interno di un agente, né, tanto meno, è stata data alcuna indicazione su come implementare la funzione *azione*. In questo paragrafo si colmerà questa lacuna, presentando le quattro categorie rispetto alle quali è possibile classificare la maggior parte dell'architetture di agente finora progettate nel mondo accademico ed in quello industriale. In particolare, si prenderanno le architetture che caratterizzano le seguenti categorie di agente:

- *Agenti basati su logica*, nei quali il processo di decisione è realizzato tramite la deduzione logica;
- *Agenti reattivi*, di cui è stata già presentata una architettura astratta, e nei quali il processo di decisione è realizzato utilizzando una qualche struttura dati che descrive corrispondenze tra stati (o percezioni) e azioni;
- *Agenti BDI,* ovvero agenti di tipo *Beliefe-Desire-Intension* (credenze, desideri, intenzioni), nei quali il processo di decisione è realizzato mediante l'elaborazione di dati relativi alle credenze, ai desideri e alle intenzioni degli agenti.
- *Agenti a strati,* in cui il processo di decisione è realizzato grazie all'interazione tra i vari strati che costituiscono questo tipo di agenti.

Si descrivono adesso più in dettaglio gli approcci progettuali relativi rispettivamente agli agenti basati su logica, agli agenti BDI e agli agenti a strati. Per approfondimenti sull'architettura degli agenti reattivi, si faccia riferimento a quanto già detto in precedenza.

1.5.1 Agenti basati su logica

In tale architettura, il comportamento intelligente è ottenuto fornendo all'agente una rappresentazione simbolica del suo ambiente e del suo comportamento, e mettendogli a disposizione gli strumenti per ragionare su essa. In questa sede per rappresentazione simbolica si intenderà un insieme di formule logiche, e il ragionamento sarà ottenuto per mezzo di deduzioni logiche. L'agente può essere visto, cioè, come un dimostratore di teoremi.

Si supponga che lo stato interno di un agente sia rappresentato per mezzo di un *database* di formule logiche del primo ordine, che codificano, ad esempio, le

informazioni che esso ha a disposizione sull'ambiente nel quale è collocato. Sia L l'insieme di tutte le frasi, ad esempio, della logica del primo ordine, e sia $D = \rho(L)$ l'insieme degli *L-database*, ovvero l'insieme degli insiemi delle formule L. In questa sede $\Delta_1, \ldots, \Delta_n$, identificano gli elementi di D. In particolare, lo stato interno di un agente è un elemento di D. Il processo di decisione di un agente è modellato per mezzo di un insieme di regole di decisione, ∂, che in questo contesto può essere visto come un insieme di regole di inferenza. Si dice che $\Delta \vdash_{\partial} \rho$ se la formula ρ può essere dimostrata a partire dal database Δ, utilizzando esclusivamente le regole di deduzione ∂. Si analizza ora nel dettaglio come sono modificate le funzioni *see*, *next* e *azione*, successivamente all'adozione di tale architettura:

La funzione *see* rimane invariata:

see : $S \rightarrow P$.

- La funzione *next* si definisce come:

next : $D \times P \rightarrow D$.

- Azione è modificata nel seguente modo:

azione : $D \rightarrow A$.

Si propone ora una implementazione concreta di quest'ultima funzione, in un linguaggio *java-like*:

```
public azione seleziona_Azione(D Δ)
{
   for(int i = 0; i < A.size(); i++)
{
   if(Δ ⊢∂ Do(A(i))
         return A(i);
   }
   for(int i = 0; i < A.size(); i++)
{
   if(Δ !⊢∂ ¬Do(A(i))
         return A(i);
   }
   return null;
}
```

L'idea alla base dell'algoritmo è la seguente. *Seleziona_Azione* verifica, tra tutte le azioni a disposizione dell'agente, quella che è modellata da una formula che può essere dedotta dal database Δ, per mezzo delle regole di deduzione ρ. L'agente eseguirà quindi la prima azione che soddisferà tale requisito. Se non esiste un'azione così fatta, la funzione cercherà un'azione modellata da una formula che sia almeno non in contraddizione con gli elementi di Δ. Più precisamente, se non è possibile dedurre da Δ $\neg$Do(A(i)), per mezzo delle regole di deduzione ρ. Se anche in questa iterazione nessuna formula avrà soddisfatto i requisiti, l'agente non eseguirà alcuna azione.

1.5.2 Agenti B.D.I.

Il ragionamento umano è stato spesso usato come modello per progettare il processo di decisione degli agenti intelligenti. In particolare, il processo di decisione degli agenti *B.D.I.* (*Beliefs*, *Desires*, *Intententions*) è basato in larga misura sul *practical reasoning*. Per *practical reasoning* si intende il ragionamento orientato alla scelta delle azioni da eseguire (esistono altri tipi di ragionamento, come il *theoretical reasoning*, orientato allo sviluppo e alla revisione delle basi di conoscenza). In particolare, il processo di *practical reasoning* può essere scisso in due sotto-processi:

- *Deliberation*, ovvero la scelta degli obiettivi che si intendono raggiungere;
- *Means-ends reasoning*, ovvero la scelta della sequenza di azioni da eseguire per raggiungere gli obiettivi definiti al passo precedente.

Un generico algoritmo che modella il comportamento di un agente B.D.I. è il seguente:

```
while(true){
    Osserva l'ambiente;
    Aggiorna la rappresentazione dell'ambiente;
    Identifica il prossimo obiettivo i;
    Definisci un piano per raggiungere l'obiettivo i;
    Esegui il piano;
}
```

1.5.3 Agenti ibridi

Per sommare i vantaggi e minimizzare gli svantaggi portati da architetture che modellano comportamenti di tipo reattivo e proattivo, sono stati introdotti gli agenti intelligenti di tipo ibrido.

L'agente ibrido è costituito da due o più sottosistemi, tra cui saranno certamente presenti:

- Un *sottosistema proattivo*, contente ad esempio una rappresentazione simbolica dell'ambiente, che produce piani e prende decisioni, e
- Un *sottosistema reattivo*, che è caratterizzato dalla capacità di reagire rapidamente agli eventi.

Gli agenti ibridi possono essere classificati rispetto a due categorie:

- Agenti ibridi con architettura orizzontale
- Agenti ibridi con architettura verticale.

Nei primi, ogni modulo è connesso direttamente con i sensori e gli attuatori, comportandosi, di fatto, ognuno come un agente a se stante. Il problema di tale architettura è che ogni modulo potrebbe suggerire un comportamento diverso, creando inconsistenza. E' richiesta quindi l'introduzione di un modulo *arbitro*, che gestisce le interazioni tra strati.

Nei secondi, un solo modulo è connesso con i sensori e un solo modulo è connesso con gli attuatori. In questo contesto, il processo di decisione "fluisce" tra i vari strati, non rendendo necessaria l'introduzione di ulteriori sottosistemi.

La seguente figura, proposta da J. S. Rosenschein, mostra tali architetture.

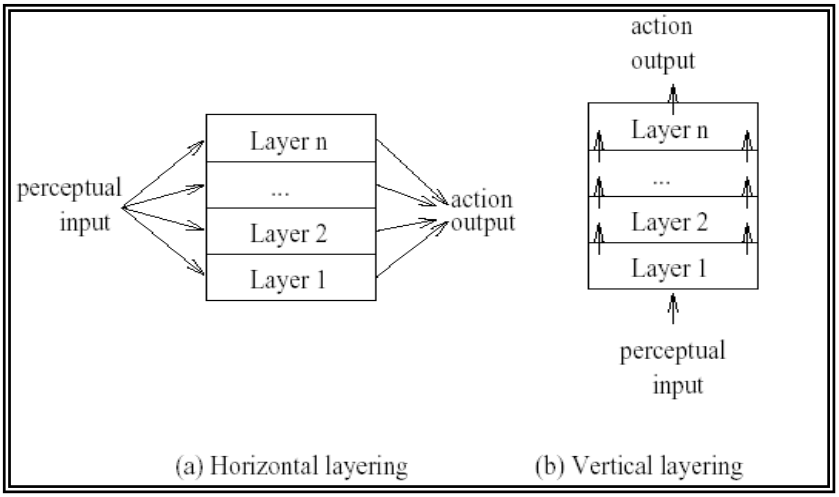

Figura 4: Architetture stratificate

Part 2

Approcci alla mediazione fra moduli di reasoning

2

Mediazione tra deep reasoning e fast reasoning in ambienti dinamici

Sommario: Dopo una introduzione sulle problematiche affrontate, nel presente capitolo sarà presentato un nuovo modello astratto basato su Ipersfera e Predizione, utile, tra l'altro, per la mediazione tra deep reasoning e fast reasoning. Per rendere più chiara la presentazione del modello, sarà proposta una instanziazione di quest'ultimo al problema della ripianificazione in ambienti dinamici. Esso è da considerarsi una specializzazione dell'approccio di mediazione basato sulla mediazione tra fast e deep reasoning, da cui discende, oltre a quella presentata, possono discendere altre specializzazioni, a seconda del problema da affrontare. Infine, è presentata una possibile istanziazione dell'approccio nel contesto degli smart environment.

2.1 Introduzione

Nel presente paragrafo e nei paragrafi 2.2 e 2.2 vengono presentate le problematiche messe in evidenza da Stevart Russell nell'articolo "Rationality and Intelligence" [R97].

In particolare, Russell asserisce che il concetto di intelligenza è strettamente legato alla capacità di adottare un comportamento di successo. Per Russell, un agente intelligente dovrebbe essere caratterizzato dalle seguenti proprietà:

1. *Perfect rationality,* ovvero, la capacità di calcolare decisioni perfettamente razionali, date le informazioni disponibili;
2. *Calculative rationality,* ovvero, la capacità di calcolare decisioni perfettamente razionali, date le informazioni disponibili nell'istante in cui l'agente inizia a ragionare;
3. *Metalevel rationality*, ovvero, la capacità di selezionare la perfetta sequenza composta da computazioni e azioni, con il vincolo che le azioni debbano essere stabilite nelle fasi di computazione;
4. *Bounded optimality*, ovvero, la capacità di generare il miglior comportamento possibile, date le informazioni e le risorse di calcolo disponibili.

Nei seguenti paragrafi saranno discusse in dettaglio le proprietà 2 e 3, maggiormente rilevanti nel contesto di tale tesi.

2.2 Calculative Rationality

In intelligenza artificiale, un agente è implementato come un programma, ad esempio detto *l*, che viene eseguito su una macchina, ad esempio detta *M*. Il programma di un agente riceve in input le percezioni correnti e mantiene in memoria uno stato interno che riflette, in una qualche forma, le percezioni precedenti. L'output di tale programma è rappresentato da azioni, che sono selezionate in funzione dell'input e dello stato. Agli occhi di un osservatore esterno, il comportamento di un agente è costituito dalle azioni selezionate, inframmezzate da periodi di inattività, o azioni *nulle* (o da azioni di *default*, generate dalla macchina).

La *calculative rationality* caratterizza i programmi che, se eseguiti a velocità infinita, genererebbero comportamenti perfettamente razionali. Diversamente dalla *perfect rationality*, la calculative rationality può essere effettivamente implementata in programmi reali. Purtroppo, però, in molti contesti essa non è una proprietà desiderabile. Ad esempio, un programma di scacchi che implementa la calculative rationality potrebbe scegliere sempre la migliore mossa possibile, ma impiegherebbe ragionevolmente un tempo di gran lunga superiore al massimo consentito.

Wooldridge [W02] asserisce che un approccio puramente *logic based* al decision making, benché possa essere caratterizzato dalla *calcolative rationality*, è spesso impraticabile. Secondo Wooldridge la calculative rationality non è una proprietà desiderabile in una ambiente real time. Per asserire ciò, l'autore si serve di un esempio. Si supponga che sia stato definito un insieme di regole ρ dell'agente tale che per ogni database Δ, se è possibile dimostrare *Do(a)*, allora *a* è una azione ottima – in altri termini, *a* è la migliore azione che può essere eseguita quando l'ambiente è descritto da Δ. Si supponga di eseguire l'agente. Al tempo t_1, l'agente ha a disposizione un certo database Δ_1 che descrive l'ambiente in tale istante, e comincia ad applicare le regole ρ al fine di selezionare l'azione da eseguire. Qualche istante più tardi, si supponga al tempo t_2, l'agente deduce che $\Delta_1 \vdash_{\partial} Do(a)$ per una qualche $a \in A_c$, dove A_c è l'insieme delle azioni eseguibili dall'agente. Pertanto, *a* è la migliore azione che l'agente può eseguire al tempo t_1. Ma se nell'intervallo di tempo che intercorre tra t_1 e t_2 l'ambiente è cambiato, non c'è alcuna garanzia che *a* sia ancora ottima. Essa può essere invece molto lontana dall'essere ottima, soprattutto se l'intervallo di t_1 e t_2 è grande. Se $t_2 - t_1$ è infinitesimale, ovvero, se il *decision making* è effettivamente istantaneo, o se l'ambiente è statico e in esso opera un solo agente, allora è possibile trascurare il problema. Ma, in realtà, il reasoning usato da un agente logic based è molto lontano dall'essere istantaneo (se l'agente usa la logica dei predicati del primo ordine per rappresentare l'ambiente, non c'è alcuna garanzia che le procedure di decision making terminino). Come già accennato prima, un agente rispetta la proprietà della *calculative rationality* se e solo se il suo sistema di *decision-making* gli suggerisce un'azione che è ottima rispetto allo stato nell'istante in cui il processo di *decision-making* ha avuto inizio. Ovviamente tale proprietà è inaccettabile in ambienti che cambiano più velocemente di quanto l'agente possa generare decisioni.

2.3 Metalevel Rationality

La *Metalevel rationality*, anche chiamata razionalità di Tipo II [G71], è basata sull'idea di trovare un compromesso ottimo tra i costi computazionali e la qualità della decisione, ovvero massimizzare l'utilità attesa tenendo conto dei costi dei processi di deliberazione. L'obiettivo potrebbe essere perseguito definendo una sorta di architettura di metalivello per implementare tale compromesso. Secondo I. J. Good [G71] è possibile dividere l'agente in due (o più) parti:

- L'*object level*, che esegue i calcoli relativi al dominio applicativo. Ad esempio, esso calcola il risultato di azioni, l'utilità di certi stati, e così via.
- Il *metalevel*, che è un secondo processo di decisione, il cui dominio applicativo è costituito dalle elaborazioni dell'object level stesso, e dagli oggetti computazionali e dagli stati che sono interessati da tali elaborazioni.

Il metareasoning ha origini molto antiche nel campo dell'intelligenza artificiale. Vi sono vari problemi aperti nell'area del metarisoning razionale. Una prima ovvia difficoltà è che quasi tutti i sistemi che lo adottano sono caratterizzati da una strategia miope – ricerca greedy e deep-one al metalevel. Sebbene il metareasoning razionale sembri uno strumento utile in contesti complessi, esso non sembra particolarmente adatto a caratterizzare utilmente agenti dotati di risorse di calcolo limitate. La ragione è che, dal momento che il metareasoning è computazionalmente costoso, in molti contesti non può essere eseguito in maniera ottima. In molti contesti, inoltre, la razionalità perfetta al metalivello non è ottenibile e la calculative rationality è inutile. Di conseguenza, in essi sembra opportuno cercare un compromesso tra tempo di elaborazione e ottimalità. Purtroppo, però, non è possibile identificare l'opportuno compromesso tra tempo e qualità delle decisioni. Infatti, ogni tentativo per ottenere ciò per mezzo di un meta-metalivello ha portato semplicemente a un regressione concettuale. Inoltre, in alcuni ambienti, il migliore modo di disegnare un agente consiste nel caratterizzare quest'ultimo solo in termini di risposte reattive a certi stimoli, con una completa assenza di qualsivoglia livello di meta-reasoning per guidare o indirizzare il processo deliberativo.

2.4 Un approccio predittivo alla ripianificazione

Al fine di presentare in maniera chiara il nuovo modello astratto *HPA,* basato su Ipersfera e Predizione, per la mediazione tra deep reasoning e fast reasoning, si propone una sua istanziazione nel contesto della ripianificazione in ambienti dinamici [GGP08].

2.4.1 Definizioni di base

Lo scenario che si propone è il seguente: il *goal* di un Agente di tipo *HPA* è di trasformare uno stato $s_0 \in S_0$ (l'insieme degli stati iniziali dell'agente) in uno stato

$s_g \in S_g$ (l'insieme degli stati *goal* o stati finali dell'agente). Per raggiungere lo stato s_g partendo dallo stato iniziale s_0, l'agente esegue un piano p. Durante l'esecuzione di un piano, lo stato di un agente può cambiare a causa: *(i)* degli effetti dell'esecuzione delle azioni, i quali dipendono fortemente dall'ambiente nel quale le azioni sono eseguite; *(ii)* degli eventi che nel frattempo accadono nell'ambiente nel quale l'agente è collocato.

Dato un possibile insieme di stati $S = \{s_1, \ldots, s_n\}$ ed un insieme di azioni $A = \{a_1, \ldots, a_n\}$, un piano $p = <a_i, \ldots, a_j>$ è una sequenza di azioni appartenenti all'insieme A. L'insieme di azioni definisce una funzione di transizione e l'applicazione di un'azione a_i ad uno stato s_j, denotata con $a_i(s_j)$, restituisce un nuovo stato s_k. La notazione $a_1.a_2(s_j)$ sarà utilizzata al posto di $a_2.a_1(s_j)$).

Definizione 1 *Un Agente di tipo HPA è una quintupla $AG = (S_0, S_g, S, P, R)$, dove S denota l'insieme dei suoi stati, $S_0 \subseteq S$ denota l'insieme dei suoi stati iniziali, $S_g \subseteq S$ denota l'insieme dei suoi stati finali. P denota l'insieme dei piani, dove per ogni $s_0 \in S_0$, esiste $p \in P$ tale che $p(s_0) \in S_g$, e R denota l'insieme delle regole reattive, dove ogni $r \in R$ è una sequenza di azioni tali che esiste $s \in (S - S_0)$ e $r(s) \in S$.*
Un HPA Active Agent $AAG = (s_0, s_g, s, p, q, R)$ che deriva da un agente di tipo HPA $AG = (S_0, S_g, S, P, R)$, è caratterizzato da uno stato iniziale $s_0 \in S_0$, uno stato finale s_g, lo stato corrente (o osservato) s, il piano già eseguito p e il piano da eseguire q.

Un *HPA Active Agent* $AAG = (s_0, s_g, s, p, q, R)$ è in uno stato corretto se $q(s) = s_g$; altrimenti, esso è in uno stato scorretto ($q(s) \neq s_g$).

Definizione 2 *Dato un HPA Active Agent $AAG = (s_0, s_g, s, p, q, R)$, un evento inatteso influente e è un'azione, non necessariamente appartenete ad A, che è eseguita prima di q, tale che $e.q(s) \neq s_g$.*

Quindi, quando un Active Agent $AAG = (s_0, s_g, s, p, q, R)$ si trasforma in un Agente $AAG = (s_0, s_g, s', p, q, R)$, dove s' è un stato scorretto, si suppone che si sia verificato un evento inatteso influente e, che ha trasformato lo stato corretto s nello stato scorretto s' (ovvero, $e(s) = s'$). Inoltre, se dopo l'esecuzione di un'azione a, appartenente ad A, l'Agente evolve in uno stato scorretto $s' \neq a(s)$, si assume che si è verificato un evento inatteso influente tale che $a.e(s) = s'$.
La sequenza di azioni che caratterizza un comportamento reattivo è diversa da un piano (ovvero, $P \cap R = \varnothing$), dal momento che essa non è concepita per far raggiungere uno stato obiettivo partendo da uno stato iniziale, ma piuttosto per reagire al verificarsi di un evento inatteso influente, parallelamente all'esecuzione del processo di ripianificazione, come descritto nel paragrafo successivo.

2.4.2 Il processo di ripianificazione

Le principali attività che caratterizzano il comportamento di un agente di tipo HPA sono mostrate nella figura seguente. Ogni attività viene eseguita da un modulo dedicato, come descritto in dettaglio nel paragrafo successivo.

Si riporta di seguito una descrizione del comportamento di un agente di tipo *HPA*. Al tempo t_0, nello stato s_0, l'Agente di tipo *HPA* inizia a interagire con l'ambiente iniziando a eseguire un piano p_0 tale che $p_0(s_0) = s_g$, al fine di raggiungere lo stato obiettivo s_g. Si supponga che al tempo t_j si verifichi un evento inatteso influente (si veda la Definizione 2). A causa dell'occorrenza di tale evento, l'Agente evolve in uno stato *s'* dal quale, continuando ad eseguire p_0, non è possibile raggiungere lo stato goal s_g.

L'Agente di tipo *HPA* percepisce l'evento, lo analizza e riconosce che si è verificato un evento inatteso influente. Pertanto, l'Agente sospende l'esecuzione del piano corrente (che è divenuto inefficace) e, contemporaneamente, adotta un comportamento reattivo e inizia a ripianificare.
Il comportamento reattivo consiste nella selezione e nell'esecuzione di regole reattive dell'insieme *R* (si veda la Definizione 1). Più in dettaglio, *R* può essere visto come una tabella nella quale le regole reattive sono selezionate usando lo stato corrente come un indice. In altri termini, da questo istante e fino a quando non avviene uno *switch* di controllo che porta all'esecuzione di un nuovo piano, dal punto di vista di un osservatore esterno l'Agente di tipo HPA è caratterizzato da un comportamento puramente reattivo.

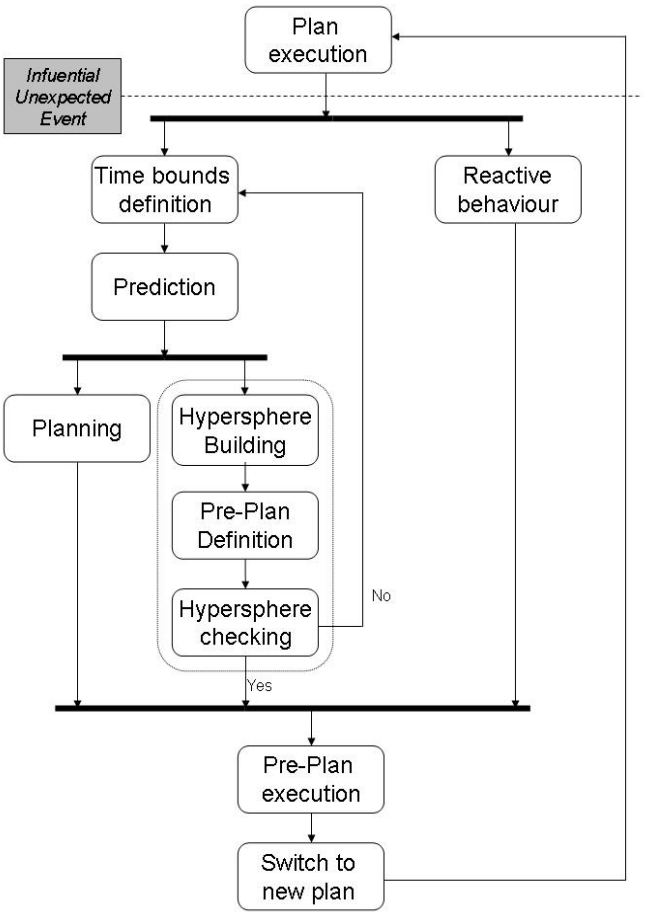

Figura 5: l'approccio HPA

Concorrentemente all'esecuzione del comportamento reattivo, l'Agente di tipo *HPA* inizia a ripianificare, eseguendo le seguenti attività:

1. *Definizione dei time bound*. L'Agente di tipo *HPA* definisce i time bound relativi ai tempi di elaborazione e di esecuzione delle altre attività che compongono il processo di ripianificazione.
2. *Predizione*. L'Agente di tipo *HPA* calcola lo stato futuro s_e nel quale si dovrebbe trovare al tempo t_e al quale dovrebbe essere iniziata l'esecuzione del nuovo piano. L'attività di predizione dovrebbe essere terminata entro ΔT_{prd} unità di tempo (t_e e ΔT_{prd} sono stati specificati nel passo 1).
3. *Pianificazione*. L'Agente di tipo *HPA* calcola il nuovo piano, che ha come stato iniziale s_e, e come stato finale s_g. Il nuovo piano deve essere calcolato

entro un tempo massimo ΔT_{pld}, come imposto nell'attività di definizione dei time bound.

4. *Definizione dell'Ipersfera di Approssimazione, definizione del pre-piano e verifica dell'Ipersfera*. Queste attività, che sono tra loro eseguite in sequenza e che sono eseguite in parallelo con l'attività di pianificazione, sono richieste in quanto, a causa della dinamicità dell'ambiente e del tempo massimo limitato dell'elaborazione della predizione, non c'è garanzia che lo stato al tempo t_e sia esattamente equivalente a s_e. L'idea è quella di calcolare un insieme di stati dell'intorno di s_e, dai quali è possibile raggiungere s_e, al massimo al più tardi al tempo t_e, eseguendo un insieme di azioni (pre-piano). Più in particolare, l'attività di definizione dell'Ipersfera di Approssimazione, definizione del pre-piano e verifica dell'Ipersfera consiste nella seguente sequenza di passi:
    a. *Definizione dell'Ipersfera di Approssimazione*. L'agente di tipo *HPA* calcola l'insieme di stati (Ipersfera di Approssimazione) dai quali è possibile raggiungere lo stato s_e entro un certo intervallo di tempo (ΔT_{ppe}) eseguendo un pre-piano. L'Ipersfera di Approssimazione deve essere calcolata entro un tempo ΔT_{hsd}, definito nel passo 1.
    b. *Definizione del pre-piano*. L'Agente di tipo *HPA*, in questa attività, definisce un insieme di azioni da eseguire prima dell'esecuzione del piano, al fine di raggiungere lo stato s_e partendo da un qualsiasi stato appartenente all'Ipersfera di Approssimazione. In effetti, nel caso in cui l'Ipersfera di Approssimazione non sia omogenea, più che di un pre-piano si potrebbe parlare di un insieme di pre-piani. Il pre-piano deve essere calcolato entro un tempo $\Delta T_{ppd,}$ definito anche esso nel contesto dell'attività di definizione dei time bound.
    c. *Verifica dell'ipersfera*. L'Agente di tipo *HPA* verifica al tempo t_e - ΔT_{ppe} se lo stato corrente appartiene all'Ipersfera di Approssimazione.

Il comportamento dell'Agente di tipo *HPA*, dopo lo svolgimento delle attività appena descritte, dipende dal risultato della verifica dell'Ipersfera di Approssimazione. In particolare:

- *Caso 1*: al tempo t_e - ΔT_{ppe} l'Agente di tipo *HPA* ha raggiunto uno stato che appartiene all'Ipersfera di Approssimazione. In questo caso, l'Agente sospende l'esecuzione del comportamento reattivo e inizia a eseguire il pre-piano precedentemente definito. Dopo l'esecuzione del pre-piano, se non si sono verificati altri eventi inattesi nel frattempo, l'Agente di tipo *HPA* può cominciare a eseguire il nuovo piano, che ha come stato iniziale s_e.
- *Caso 2*: al tempo t_e - ΔT_{ppe} l'agente di tipo *HPA* ha raggiunto uno stato che non appartiene all'Ipersfera di Approssimazione. In questo caso non viene sospesa l'esecuzione del comportamento reattivo, e viene riavviata l'esecuzione del processo di ripianificazione, a partire dall'attività di definizione dei time bound. Nell'ambito di quest'ultima vengono rimodulati i time bound, in funzione dei fattori che hanno portato a una

non efficace predizione o a una non efficace definizione dell'Ipersfera di Approssimazione.

E' interessante notare che l'eventuale verificarsi di un altro evento inatteso influente durante una qualsiasi fase del processo di ripianificazione causa l'immediata sospensione di tale processo e l'inizio di un nuovo processo di ripianificazione per gestire l'occorrenza di tale evento. Anche in questo caso, parallelamente al processo di ripianificazione, l'Agente inizia nuovamente a reagire, ora in funzione del nuovo evento inatteso influente. In ogni modo, il caso appena esaminato è privo di rischi, anche se si dovesse verificare un grande numero di eventi inattesi influenti, dal momento che l'Agente semplicemente "azzera" il processo di ripianificazione ogni volta che si verifica un evento inatteso influente, continuando al contempo ad adottare un comportamento reattivo.

Vengono discussi di seguito ulteriori dettagli su alcuni importanti aspetti del processo di ripianificazione.

2.4.2.1 Definizione dei time bound

L'attività di definizione dei time bound definisce i time bound per l'esecuzione delle attività del processo di replanning. In particolare, essa definisce i seguenti time bound:

- ΔT_{prd}: tempo massimo per calcolare l'output dell'attività di predizione;
- ΔT_{hsd}: tempo massimo per definire l'Ipersfera di Approssimazione;
- ΔT_{ppd}: tempo massimo per definire il pre-piano;
- ΔT_{pld}: tempo massimo per definire il nuovo piano;
- ΔT_{ppe}: tempo massimo per eseguire il pre-piano.

E' utile ricordare che l'ordine temporale di esecuzione delle attività che compongono il processo di ripianificazione è il seguente: inizialmente, vengono eseguite in sequenza le attività di definizione dei time bound e di predizione; successivamente, l'Agente esegue in parallelo l'attività di pianificazione e la sequenza composta dalle attività di definizione dell'Ipersfera di Approssimazione, di pre-pianificazione e di verifica dell'Ipersfera di Approssimazione. Il tempo di esecuzione dell'attività di verifica dell'Ipersfera di Approssimazione è trascurabile, in quanto tale attività consiste semplicemente nel verificare se lo stato al tempo t_e - ΔT_{ppe} appartiene all'Ipersfera di Approssimazione. E' possibile ora calcolare il tempo t_e, ovvero il tempo al quale dovrebbe iniziare l'esecuzione del nuovo piano:

$$t_e \geq t_{bde} + \Delta T_{ppe} + max(\Delta T_{hsd} + \Delta T_{ppd}, \Delta T_{pld})$$

dove t_{bde} è il tempo al quale termina l'esecuzione dell'attività di definizione dei time bound.

I time bound sono definiti e modulati sulla base dell'efficacia dei precedenti processi di ripianificazione (e del tipo di comportamento desiderato). Ad esempio,

due tra le situazioni che possono causare la rimodulazione dei time bound sono le seguenti:

- Al tempo $t_e - \Delta T_{ppe}$ il sistema è in uno stato che non appartiene all'Ipersfera di Approssimazione, e quindi, è necessario iniziare nuovamente a ripianificare. Probabilmente, in questo caso, il dinamismo dell'ambiente sta aumentando, e ciò richiede un replanning più veloce che in passato (si ricorda che i time bound sono modulati in funzione del successo delle ripianificazioni avvenute precedentemente). Se il dinamismo dell'ambiente aumenta, l'intervallo $t_e - t_j$ dovrebbe diminuire, in quanto, se il dinamismo dell'ambiente è alto, è più probabile che la predizione di uno stato del sistema in un istante di tempo lontano sia inefficace;
- Si verificano molti eventi inattesi influenti mentre l'Agente sta ripianificando. In questo caso, l'Agente "azzera" frequentemente tale processo, e ciò lo forza di fatto a eseguire costantemente comportamenti reattivi. Anche in questo caso può essere necessario rimodulare i time bound, al fine di ridurre la durata del processo di ripianificazione, ed essere in condizione di eseguire l'output di tale processo prima del verificarsi di un nuovo evento inatteso.

Queste capacità di rimodulazione rendono quindi l'approccio *HPA* adatto anche per ambienti altamente dinamici ed adattabile a casi in cui si verifica un grande numero di eventi inattesi a causa, ad esempio, di un alto numero di agenti che operano nello stesso ambiente.

2.4.2.2 Predizione

All'attività di definizione dei time bound segue quella di predizione; l'output di quest'ultima è una predizione dello stato s_e al tempo t_e, ovvero l'istante nel quale l'Agente dovrebbe iniziare a eseguire il nuovo piano.

Gli input di tale attività sono i seguenti:

- Una descrizione dell'evento inatteso influente;
- Una descrizione dello stato al tempo t_j – il tempo al quale l'agente ha osservato l'evento inatteso influente;
- L'insieme R delle regole reattive a disposizione dell'Agente;
- Conoscenza sull'ambiente.

Sulla base di questi input il modulo di predizione calcola s_e.

2.4.2.3 Definizione dell'Ipersfera di Approssimazione

La definizione dell'Ipersfera di Approssimazione e la pre-pianificazione sono state introdotte nell'approccio *HPA* in quanto non c'è garanzia che lo stato del sistema al

tempo t_e sia esattamente equivalente allo stato s_e, precedentemente predetto. Questo perché l'output del processo di predizione potrebbe essere impreciso, per vari motivi:

- Il tempo di elaborazione a disposizione del modulo di predizione è limitato, e questo influenza il grado di accuratezza della predizione;
- Si potrebbe verificare un evento inatteso influente dopo t_j – a questo punto è utile ricordare che:
    - La predizione è basata sullo stato al tempo t_j;
    - L'occorrenza di un evento inatteso influente causa il riavvio del processo di predizione;
- Il tempo di reazione è variabile, e la risposta dell'ambiente alla reazione dell'Agente potrebbe essere abbastanza differente da quella prevista dal modulo di predizione.

Quanto detto mette in evidenza quanto sia difficile predire esattamente lo stato al tempo t_e. E' tipicamente molto più semplice predire che, al tempo $t_e - \Delta T_{ppe}$, lo stato potrebbe appartenere all'insieme degli stati che costituiscono un intorno (data una opportuna funzione di distanza) dello stato s_e, ovvero che appartengono all'Ipersfera di Approssimazione. In questo caso, al fine di "pilotare" lo stato del sistema nello stato s_e al tempo t_e, l'Agente esegue una sequenza di azioni correttive (un pre-piano). La distanza degli stati appartenenti all'intorno di s_e deve essere opportunamente limitata, in modo tale che sia possibile raggiungere lo stato s_e, eseguendo azioni correttive (un pre-piano) non più tardi di t_e. Più formalmente:

Definizione 3 *Un'Ipersfera di Approssimazione di uno stato s_e e un time bound ΔT_{ppe} è un insieme di stati S tale che per ogni stato $s_i \in S$ esiste un pre-piano eseguendo il quale è possibile trasformare lo stato s_i in uno stato s_e in un tempo minore o uguale di ΔT_{ppe}.*

E' interessante osservare che nell'approccio *HPA* ogni stato s_e è associato ad un unico pre-piano (definito sulla base delle proprietà di s_e) ed ogni stato s_i appartenente all'Ipersfera di Approssimazione è caratterizzato dal fatto che applicando ad esso il pre-piano lo stato può essere trasformato in s_e entro il tempo t_e. Si noti che è possibile generalizzare quanto detto considerando l'Ipersfera di Approssimazione come composizione di più intorni dello stato s_e e più pre-piani che potrebbero eseguiti in parallelo.

2.4.3 Una Architettura basata su Ipersfera e Predizione

Il processo di ripianificazione descritto nel paragrafo precedente può essere supportato da un'architettura basata si Ipersfera e predizione, che viene mostrata nella figura seguente.

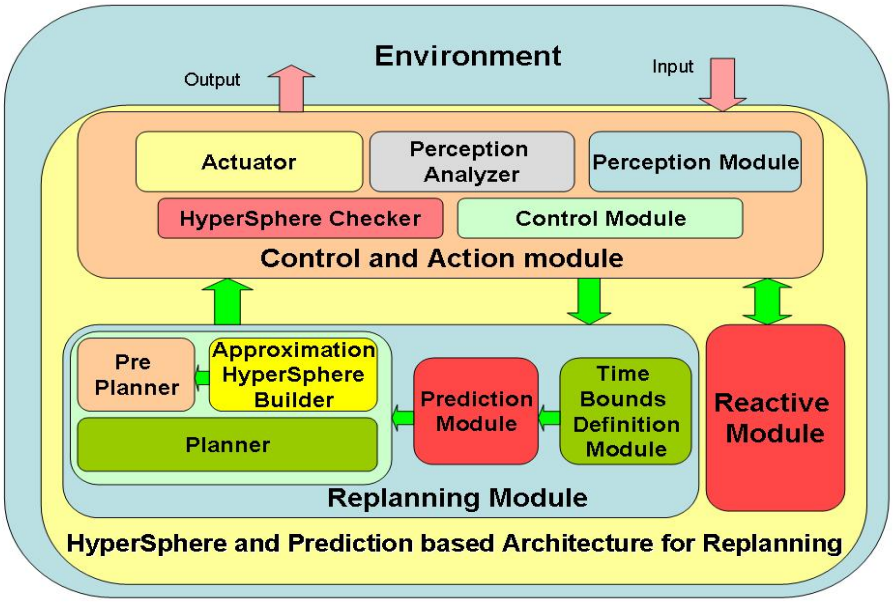

Figura 6: Un'architettura basata su Ipersfera e Predizione

L'architettura *HPA* è composta da tre macro moduli: un *Modulo di Controllo e Attuazione*, un *Modulo Reattivo* e un *Modulo di Ripianificazione*.

Il modulo di *Controllo e Attuazione* consiste dei seguenti sotto-moduli:

- *Modulo di Percezione*, che percepisce l'ambiente per mezzo di un insieme di sensori;
- *Modulo di Analisi delle Percezioni*, che elabora i dati relativi alle percezioni ricevuti dal *Modulo di Percezione*;
- *Attuatore*, che esegue azioni selezionate alternativamente dal *Modulo di Ripianificazione* e dal *Modulo Reattivo*;
- *Modulo di Controllo*, che controlla gli altri sotto-moduli e, coordinando il *Modulo di Ripianificazione* e il *Modulo Reattivo,* determina il comportamento dell'Agente. In particolare, se il *Modulo di Analisi delle Percezioni* identifica un evento inatteso influente, il *Modulo di Controllo*:
  - Sospende l'esecuzione del piano corrente;
  - Attiva sia il *Modulo Reattivo* che il *Modulo di Ripianificazione*.

  Durante la reazione dell'Agente, il *Modulo di Controllo*:
  - Richiede all'*Attuatore* l'esecuzione di azioni reattive selezionate dal *Modulo Reattivo*;

- Nel caso del verificarsi di un nuovo evento inatteso influente, sospende il processo di ripianificazione riavvia il *Modulo Reattivo* e il *Modulo di Ripianificazione*;
- Al tempo t_e - ΔT_{ppe} richiede al *Modulo di Verifica dell'Ipersfera* di verificare se lo stato appartiene all'Ipersfera di Approssimazione, e, sulla base del risultato, richiede all'Attuatore di eseguire il pre-piano o, alternativamente, riavvia il *Modulo di Ripianificazione* e richiede all'*Attuatore* di continuare a eseguire le azioni reattive.

Il *Modulo di Ripianificazione* è responsabile del processo di ripianificazione descritto nel paragrafo precedente. Esso è composto dai seguenti sotto-moduli:

- *Modulo di Definizione dei Time Bound*, che definisce i time bound come specificato in precedenza;
- *Modulo di Predizione*, che predice lo stato futuro che dovrebbe caratterizzare il sistema nell'istante in cui l'Agente dovrebbe iniziare a eseguire il nuovo piano;
- *Pianificatore*, che calcola un nuovo piano che ha come stato iniziale lo stato predetto dal *Modulo di Predizione*, e come stato finale lo stato obiettivo che caratterizzava già il piano precedente. Diversi sono gli algoritmi adottabili per eseguire il task di pianificazione [GD04, L06].
- *Modulo di Definizione dell'Ipersfera di Approssimazione*, che calcola l'insieme degli stati dell'intorno dello stato predetto dal *Modulo di Predizione*, dai quali è possibile raggiungere lo stato predetto eseguendo un pre-piano entro un certo intervallo di tempo massimo;
- *Modulo di Pre-pianificazione*, che calcola il pre-piano che potrà essere utilizzato per raggiungere lo stato predetto dal *Modulo di Predizione* partendo da uno degli stati appartenenti all'Ipersfera di Approssimazione. Anche in questo caso, è possibile adottare diversi algoritmi per il task di pre-pianificazione [GD04, L06];

Il *Modulo Reattivo* è responsabile della reazione a un evento inatteso influente; in particolare, esso seleziona una sequenza di azioni e le invia all'*Attuatore*, in modo che esse siano eseguite.

2.5 Caratteristiche e novità dell'approccio presentato

Nel paragrafo precedente è stata presentata una istanziazione dell'approccio *HPA* al caso della ripianificazione. Questo esempio è stato utile per mostrare le caratteristiche dell'approccio *HPA*, che, però, è adattabile a tutti i contesti in cui siano richieste più di un modulo di *decision making* con caratteristiche diverse tra loro in ambienti dinamici, o ancora più astrattamente, a tutti i contesti in cui sia presente il concetto di stato, che cambia nel tempo, e in cui sono richiesti dei processi di reasoning.

Nel modello astratto, sono presenti il concetto di deep reasoning, che genera decisioni ottime (o sub-ottime) e il concetto di fast reasoning, la cui velocità nel generare decisioni dovrebbe essere molto superiore alla velocità con cui cambia il sistema.

Di seguito vengono proposte alcune caratteristiche e alcune novità dell'approccio proposto:

- Con l'approccio *HPA* è possibile basare (con una certa probabilità di successo) il reasoning sullo stato futuro nell'istante in cui l'output del ragionamento verrà eseguito, e non più sullo stato al momento in cui il processo di reasoning ha inizio (come per i programmi che sono caratterizzati dalla calculative rationality). Si noti che l'approccio è scalabile rispetto al grado di sub-ottimalità desiderato, nel senso che l'approccio *HPA* è valido anche scegliendo di avere deliberazioni sub-ottime. In tal caso, la probabilità che venga effettivamente eseguito quanto deliberato dal modulo di deep reasoning, è in genere più alta che nel caso si opti per ottenere deliberazioni ottime, a parità del grado di dinamismo dell'ambiente. Infatti, la probabilità di successo dei processi di predizione e di definizione dell'Ipersfera di Approssimazione, ovvero la probabilità che a un certo istante di tempo lo stato appartenga all'Ipersfera di Approssimazione, e, dopo un certo intervallo di tempo e l'esecuzione di opportune azioni correttive, esso sia uguale allo stato predetto, dipende dal grado di dinamicità dell'ambiente e dal grado di sub-ottimalità della decisione desiderato. Si può supporre che tipicamente il tempo necessario per ottenere una elaborazione ottima sia superiore a quello necessario per determinare una elaborazione sub-ottima. Nell'approccio *HPA*, in genere, come accennato, maggiore è il tempo dedicato al deep reasoning, minore è la probabilità che quanto deliberato da quest'ultimo sia eseguito. Infatti, richiedendo il reasoning più tempo, sarà anche più lontano l'istante in cui quanto deliberato potrà essere eseguito. Più lontano è tale istante, più imprecisa sarà la predizione, in quanto è più probabile che lo stato dell'ambiente nel frattempo cambi. Più imprecisa è la predizione, più improbabile è che lo stato in un dato istante appartenga effettivamente all'Ipersfera di Approssimazione. Di conseguenza, sarà più improbabile che l'output del reasoning venga eseguito.

- Nell'approccio presentato avviene un metareasoning implicito, con costi computazionali trascurabili (l'unico costo è quello dovuto alla verifica che lo stato corrente appartenga all'Ipersfera di Approssimazione).

- L'attività di deliberazione del modulo di deep reasoning (o, nell'istanziazione proposta, di pianificazione) avviene durante l'esecuzione di quanto deliberato dal modulo di fast reasoning (nell'istanziazione proposta, durante l'adozione di un comportamento reattivo). In questo modo non vengono corsi i rischi e non vi sono le inefficienze legate al protrarsi nel tempo del processo di deliberazione. Infatti, l'elaborazione del modulo di deep reasoning potrebbe non essere disponibile in tempo utile ma, essendo in esecuzione quanto deliberato dal modulo di fast reasoning, non vi è alcuna attesa significativa prima che l'agente cominci ad agire (supponendo trascurabile il tempo necessario alle elaborazioni del modulo di fast

reasoning). Il parallelismo tra l'esecuzione delle deliberazioni del fast reasoning module (che si suppone, siano tipicamente molto più rapide della velocità di cambiamento dello stato) e il deep reasoning, rende opzionale la definizione di una strategia di alternanza tra fasi di reasoning e fasi di esecuzione degli output del reasoning, mentre, con altri approcci, essa è di importanza fondamentale.

- L'approccio *HPA* è molto facilmente adattabile per ottenere una mediazione tra comportamento reattivo e pro-attivo. In particolare, è possibile sostituire al modulo di planning un modulo che genericamente definisce comportamenti proattivi.
- Con l'approccio HPA, è possibile definire a priori il tempo da dedicare al reasoning. Un caso particolare è quello in cui si richiede di ottenere una deliberazione ottima. In tal caso, non sarò definito il time bound relativo alle elaborazioni del modulo di deep reasoning.
- Più generalmente rispetto a quanto affermato nel punto precedente, è possibile decidere il livello di accuratezza dei processi escludendo o inserendo alcuni moduli. Come accennato precedentemente, ad esempio è possibile non calcolare i time bound in ogni ciclo di deep reasoning (di ripianificazione). Si noti che i time bound possono essere funzione del grado di dinamicità dell'ambiente e delle caratteristiche dell'evento inatteso (ad esempio, se esso genera pericolo, oltre che impossibilità a raggiungere lo stato obiettivo, tipicamente si preferisce avere a disposizione una ripianificazione rapida, anche a scapito della sua qualità). Ovviamente, se il calcolo dei time bound viene fatto in funzione di entrambi i parametri, esso deve essere eseguito dopo il verificarsi dell'evento inatteso. Viceversa, si potrebbe non tenere conto delle caratteristiche dell'evento inatteso e, quindi, calcolare precedentemente al suo verificarsi i time bound. Questa strategia, se da un lato riduce i tempi di ripianificazione (o di deep reasoning), dall'altro può introdurre minore accuratezza nel processo complessivo. Tale strategia potrebbe ad esempio trovare interessante applicazione in ambienti in cui è possibile il verificarsi di una sola classe di eventi inattesi (ad esempio, per un robot mobile che debba spostarsi da un punto *A* a un punto *B* in un edificio, l'unica classe di eventi inattesi potrebbe essere quella che include gli eventi che rendono impossibile percorrere un tratto interno alla traiettoria pianificata). In tal caso, diventerebbero poco significative le caratteristiche dell'evento stesso, e, quindi, sarebbe ragionevole ed efficiente che le variabili che codificano i time bound siano funzione esclusivamente del grado di dinamicità del sistema, e che siano aggiornate periodicamente e calcolate parallelamente all'esecuzione degli altri processi. Un altro modulo che, ad esempio, potrebbe essere incluso in un sistema del tipo descritto (e cui non si è fatto cenno nei paragrafi precedenti) è quello di learning che, sulla base del successo dei processi di predizione e di definizione dell'Ipersfera di Approssimazione nei precedenti processi di replanning (di deep reasoning), fornisce opportuno supporto alla modulazione dei time bound. Queste e altre possibili scelte di inserimento o eliminazione di moduli possono rendere da un lato più o meno accurati i risultati delle elaborazioni,

dall'altro possono appesantire (o, viceversa, rendere più rapidi) le elaborazioni stesse.

- Con l'approccio HPA è possibile ottenere un implicito comportamento adattativo (senza costi aggiuntivi): l'agente si comporterà, in ambienti fortemente dinamici, come un agente puramente reattivo, mentre in ambienti con grado di dinamismo inferiore, avrà un comportamento ibrido. Negli ambienti caratterizzati da una bassa dinamicità, invece, l'agente sarà caretterizzato più frequentemente dal comportamento stabilito dal deep reasoning module (ad esempio, da un comportamento proattivo). Infatti, se maggiore è la dinamicità dell'ambiente, più difficile sarà che lo stato predetto sia effettivamente lo stato dell'ambiente al momento in cui dovrebbe iniziare l'esecuzione di quanto deliberato dal deep reasoning module (ad esempio, di un piano). L'agente, così, eseguirà più spesso il comportamento stabilito dal fast reasoning module (ad esempio, da un modulo reattivo).

Rispetto ad altre architetture ibride orizzontalmente stratificate, l'architettura HPA non necessita perciò di un modulo arbitro, che dà il controllo a l'uno o all'altro modulo di deliberazione. Come accennato, non ci sono costi aggiuntivi (se non quello, trascurabile, della verifica dell'Ipersfera di approssimazione).

2.6 Un'applicazione agli Smart Environment dell'approccio basato su Ipersfera e Predizione

In questo paragrafo viene presentata una possibile applicazione dell'approccio basato su Ipersfera e Predizione al contesto degli Smart Environment [GGP08]. Di recente, si è assistito ad un interesse intorno al disegno e allo sviluppo di smart environment, dal momento che essi possono essere utilizzati per implementare applicazioni il cui scopo è quello di migliorare la qualità della vita delle persone. Un'altra importante problematica, che non è stata ancora pienamente studiata, è quella che riguarda il disegno e lo sviluppo di agenti capaci di agire in maniera efficace ed efficiente in nuove classi di smart environment, caratterizzati dall'essere complessi e aperti. In questo paragrafo viene proposta una istanziazione dell'approccio *HPA* per agenti che sono collocati e agiscono in smart environment appartenenti a tali classi. La flessibilità dell'approccio lo rende utilizzabile anche nella progettazione di strutture di ripianificazione che potrebbero caratterizzare gli stessi smart environment, siano essi concepiti come singole entità, che modellate come sistemi multi agente.

2.6.1 Introduzione

Uno smart environment può essere definito come "una regione del mondo reale che è estensivamente equipaggiata con sensori, attuatori, e componenti computerizzate" [CD04]; esso può percepire quello che si verifica al proprio interno e nei propri dintorni, e adattare le proprie azioni in maniera adeguata, al fine di raggiungere i propri obiettivi (ad esempio, per rendere la vita dei propri abitanti più confortevole [AN06]).

Esiste un crescente interesse intorno allo sviluppo degli smart environment, dal momento che essi possono essere usati per essere usati per migliorare la qualità della vita [CY03]. In ogni caso, a causa degli ambiziosi obiettivi e delle grandi aspettative che hanno generato, si rendono necessari adeguati paradigmi e tecnologie per supportare il loro sviluppo. Inoltre, un'importante caratteristica che rende complesso il loro sviluppo è l'essere aperti: l'entità che abitano tali ambienti sono eterogenee, non conosciute a priori e possono cambiare nel tempo.

Diversi sforzi di ricerca sono stati dedicati al disegno e allo sviluppo di smart environment [CY06]. Una problematica importante non pienamente studiata è quella riguardante il disegno e lo sviluppo di entità autonome (agenti) in grado di agire in maniera efficace ed efficiente in tali ambienti aperti. Infatti, essendo gli smart environment caratterizzati da comportamenti autonomi e orientati al raggiungimento dei proprio obiettivi, l'entità autonome che sono collocate e agiscono in essi dovrebbero essere capaci di interagire in maniera adeguata con l'ambiente e adattare costantemente i propri comportamenti al comportamento degli smart environment stessi. Sebbene gli smart environment mettono a disposizione diversi servizi avanzati che possono essere usati da tali entità per raggiungere i propri obiettivi in maniera più efficace, i comportamenti orientati agli obiettivi e autonomi che caratterizzano tali ambienti, rendono questi ultimi complessi e dinamici, rendendo così l'interazione con essi più complicata. In particolare, una delle capacità desiderabili delle entità che sono in essi collocate è la pianificazione. Per supportare questa affermazione, è utile richiamare e modificare di uno scenario utilizzato nel corso del capitolo. In tale scenario, un entità autonoma (un agente) è collocata in uno smart environment aperto ed è impegnata nell'eseguire un piano, al fine di raggiungere un obiettivo. A causa della complessità e della natura dinamica dell'ambiente, può verificarsi un evento che non consente all'agente di continuare l'esecuzione del piano corrente, e così si rende necessario definire un nuovo piano per raggiungere l'obiettivo. Tipicamente, durante l'esecuzione del piano, l'agente reagisce all'evento occorso, applicando regole reattive. Dopo un certo intervallo di tempo un nuovo piano è pronto per essere eseguito, e così l'agente può abbandonare il comportamento reattivo, iniziando a eseguire tale piano. In ogni caso, a causa della complessità e del comportamento autonomo dell'ambiente, lo stato, al tempo in cui dovrebbe avvenire lo *switch* di comportamento, potrebbe essere molto diverso dallo stato al tempo in cui l'evento si è verificato. Se tale possibile evoluzione non è presa in considerazione in maniera accurata nel processo di definizione del nuovo piano, quest'ultimo potrebbe non essere eseguibile.

In questo paragrafo si propone una instanziazione dell'approccio *HPA* nel contesto degli smart environment. Sebbene siano stati proposti diversi approcci alla ripianificazione dinamica [HD06] l'approccio proposto è sostanzialmente diverso dagli altri, in quanto il processo di deliberazione (ad esempio, la pianificazione) si basa sullo stato futuro in cui verrà eseguito l'output di tale processo.

Inoltre, sebbene esistano diversi studi e proposte che mirano a definire delle strutture di pianificazione a supporto degli smart environment, non è stato ancora approfonditamente studiato come fornire capacità di pianificazione adeguate alle entità autonome che agiscono in tali ambienti. Inoltre, è interessante notare che la flessibilità dell'approccio proposto lo rende utilizzabile per fornire capacità di

ripianificazione agli smart environment stessi, sia che essi siano modellati come una singola entità razionale, o come un sistema multiagente. Nel primo caso, lo smart environment genera ed esegue piani per raggiungere i propri obiettivi. Di nuovo, a causa dei possibili eventi che potrebbero verificarsi durante l'esecuzione di questi piani, che potrebbero rendere gli obiettivi non più raggiungibili eseguendo i piani correnti, potrebbe essere necessario, da parte dello stesso smart environment, ripianificare per rendere tali obiettivi nuovamente raggiungibili, riproponendo così le stesse problematiche esposte nel primo scenario. Nel secondo caso, in cui lo smart environment è modellato per mezzo di un sistema multiagente, gli agenti autonomi che cooperano per raggiungere gli obiettivi di tale ambiente, dovrebbero avere a disposizione opportune capacità di ripianificazione.

2.6.2 Caratteristiche specifiche del processo di ripianificazione nel contesto degli smart environment

In questo paragrafo sono presentate le differenze dell'approccio *HPA* alla ripianificazione in ambienti dinamici e l'approccio HPA alla ripianificazione nel contesto degli smart environment:

- Si suppone che l'Agente di tipo *HPA* sia benevolo e non agisca in contrasto con le regole e gli obiettivi dello smart environment in cui è collocato;
- Lo smart environment permette all'Agente di tipo HPA di raccogliere informazioni riguardo i propri obiettivi e i propri comportamenti, al fine di facilitare le interazioni presenti e future tra l'Agente e se stesso;
- Gli input del modulo di predizione, oltre a quelli presentati precedentemente, sono costituiti anche da conoscenza riguardo il futuro comportamento dello smart environment. Questo consente di predire con maggiore accuratezza l'evoluzione dello stato di quanto avvenga per ambienti non caratterizzati dall'essere "smart";
- Tra i task del Modulo di Controllo vi è quello di inviare esplicite richieste di informazioni allo smart environment (e non più, semplicemente, percepire l'ambiente);

2.7 Un esempio applicativo

Al fine di mostrare una concreta dell'approccio al replanning proposto, questo paragrafo presenta un esempio applicativo nel quale un robot mobile (*Worker*) agisce in uno *smart warehouse* per conservare in maniera adeguata dei beni, sulla base della loro tipologia.

In particolare, l'obiettivo del *Worker* è di spostare una certa quantità di beni da un punto *A* a un punto *B*. Il robot ha a disposizione una mappa dell'ambiente (il *warehouse*), conoscenza riguardo il comportamento atteso di tale ambiente, ed un piano p_0 per spostare la quantità desiderata di beni da *A* a *B*. L'architettura interna del

Worker è di tipo *HPA*. Il *warehouse* è rappresentato da una griglia bi-dimensionale. Ogni cella della griglia ha una dimensione fissata ed è identificata da un indice di riga i e un indice di colonna j, con $1 \leq i \leq n$ e $1 \leq j \leq m$, dove n e m sono rispettivamente il numero di righe e il numero di colonne della griglia; a ogni cella è associata una tupla che rappresenta un insieme di proprietà della cella (ad esempio, il tipo e la quantità di beni contenuti, la percentuale di superficie occupata, la temperatura, etc.). In particolare, ad un certo istante t_k, ogni cella può essere libera, oppure occupata dal *Worker* o da un oggetto o da una sua porzione (ad esempio, uno scaffale contenente beni, un pilastro, un estintore, etc.) ed è caratterizzato da un assegnamento di valore a ognuna delle sue proprietà. Quindi, ad un certo istante t_k, lo stato dell'ambiente (il *warehouse*) è rappresentato dalla configurazione della griglia, ovvero, da un insieme di tuple associate alle celle della griglia.

Una azione è una funzione che trasforma una configurazione di griglia in un'altra configurazione di griglia. Il comportamento di un'entità attiva (agenti, *smart warehouse*) è costituito da un insieme di azioni eseguite per raggiungere un obiettivo.

Quando il *Worker* inizia ad agire acquisisce dallo *smart warehouse* una descrizione dello stato attuale dell'ambiente (una configurazione di griglia). Nel corso dell'esecuzione del piano, il *Worker* ha conoscenza parziale dell'ambiente, limitata alla cella dell'ambiente che è in grado di percepire; a causa di ciò, per prendere decisioni che potrebbero coinvolgere celle esterne all'area che riesce a percepire, è per l'agente necessario chiedere informazioni addizionali allo *smart warehouse*.

Lo stato del *Worker* è costituito da un insieme di proprietà (la sua velocità, l'azione corrente che sta eseguendo, il piano che sta eseguendo, etc.) ed un insieme di configuarazioni (parziali) di griglia: la configurazione di griglia corrente e la configurazione successiva, che dovrebbe derivare dall'esecuzione dell'azione corrente.

Un evento, che può essere visto come un'azione, è una funzione che mette in relazione una configurazione di griglia con un'altra. Gli eventi e le loro conseguenze possono essere percepiti dal *Worker* nel caso essi interessino celle che possono essere percepite da quest'ultimo, oppure se sono segnalati al *Worker* dallo *smart warehouse*.

Un evento è influente se genera una configurazione di griglia dalla quale, eseguendo la porzione di piano non ancora eseguita, la configurazione di griglia *goal* non può essere ottenuta.

Quando un *Worker* percepisce un evento (o viene informato della sua occorrenza) che riconosce come evento influente, interagisce con lo *smart warehouse* tentando di ottenere informazioni sulle conseguenze di tale evento. Tali informazioni possono essere sfruttate sia per guidare la reazione del *Worker*, sia per pilotare le attività di ripianificazione e di costruzione dell'Ipersfera di Approssimazione. Ad esempio, il *Worker* potrebbe avere a disposizione un insieme di regole reattive e scheletri di piani associati a specifiche configurazioni di griglia.

Ad esempio, se in un certo istante t_j, il modulo di analisi delle percezioni del *Worker* riconosce che uno scaffale è caduto e ostruisce l'itinerario, che, seguendo le azioni indicate nel piano corrente, dovrebbe condurre a B, il modulo di controllo del Worker avvia due processi concorrenti:

- Sospende l'esecuzione del piano corrente e inizia l'esecuzione del comportamento reattivo;
- Avvia il processo di ripianificazione.

Per semplicità, sia assuma che sia disponibile una semplicissima regola per l'evitamento degli ostacoli, che potrebbe essere come segue:

REGOLA1:

if(è presente un ostacolo sulla tua traiettoria) **then**

invia una segnalazione allo *smart environment* e chiedi indicazioni su una nuova direzione da seguire verso B;

repeat

segui la direzione suggerita;

until (non viene riconosciuto alcun nuovo ostacolo)

end if

Dopo aver iniziato a reagire, il *Worker* acquisisce dallo *smart warehouse* informazioni sulla configurazione di griglia corrente, il suo comportamento atteso (ad esempio, lo smart wareouse potrebbe aprire alcune porte precedentemente bloccate lungo un percorso alternativo, al fine di rendere nuovamente raggiungibile la configurazione di griglia finale). Successivamente, il *Worker*, al fine di predire la configurazione di griglia al tempo t_e (il tempo al quale dovrebbe iniziare l'esecuzione del nuovo piano) esegue una simulazione, che consiste nell'applicare alla configurazione di griglia corrente le azioni derivanti dalla composizione delle sue regole reattive e del comportamento dello *smart environment*. Quindi, il risultato della simulazione consiste della configurazione di griglia predetta e dello stato del *Worker* al tempo t_e.

Per esempio, si supponga che il modulo di predizione predica che la griglia al tempo t_e dovrebbe essere in una certa configurazione $config_e$, nella quale il *Worker* si trovi nella cella c_e ed abbia già trasportato il 30% dei beni da trasportare.

A questo punto, se l'ambiente fosse statico, null'altro sarebbe necessario: il modulo di pianificazione avrebbe a disposizione dati relativi la configurazione di griglia e lo stato del *Worker* al tempo t_e. Purtroppo, essendo l'ambiente dinamico, e il tempo disponibile per le elaborazioni del processo di predizione limitato, la predizione generata potrebbe essere imprecisa: non c'è garanzia che al tempo t_e l'ambiente sarà esattamente nella configurazione $config_e$. In ogni caso, nell'approccio *HPA*, è sufficiente che, al tempo $t_e - \Delta T_{ppe}$, la configurazione di griglia sia in un intorno della configurazione precedentemente predetta $config_e$. Più precisamente, in questo esempio, è sufficiente che la posizione del *Worker* sia all'interno di una cella dalla quale è possibile raggiungere la cella c_e (la cella in cui si sarebbe dovuto trovare se la predizione fosse esatta) in un intervallo di tempo minore o uguale a ΔT_{ppe} e che il *Worker* abbia trasportato una quantità di beni tale che sia possibile trasportare il 30%

della merce entro il tempo t_e. In altre parole, l'Ipersfera di Approssimazione identifica un insieme di configurazioni di griglia dalle quali è possibile raggiungere la configurazione predetta $config_e$ in un tempo minore o uguale a ΔT_{ppe}, eseguendo il pre-piano precedentemente calcolato.

Se, al tempo t_e - ΔT_{ppe}, la configurazione dell'ambiente corrente appartiene all'Ipersfera di Approssimazione, il *Worker* sospende di eseguire azioni reattive e inizia l'esecuzione del pre-piano. Successivamente, se il *Worker* riesce a ottenere in tempo utile la configurazione dell'ambiente $config_e$, inizia a eseguire il nuovo piano.

Se si verifica qualche problema, il *Worker* continua a reagire e inizia a ripianificare.

3

Una soluzione alternativa per il replanning in ambienti dinamici

Sommario: In questo capitolo presentiamo una soluzione alternativa al problema del replanning in ambienti dinamici [PG07]. In particolare, proponiamo una architettura multiagente caratterizzata dal poter generare sia decisioni complesse, sia decisioni semplici. Dal momento che gli agenti operano in un ambiente dinamico, essi sono dotati di un componente basato su regole che viene utilizzato per reagire a eventi inattesi, e utilizzano inoltre la simulazione per prendere semplici decisioni. Nell'architettura che proponiamo l'unità di supporto definisce i piani per gli agenti (decisioni complesse), mentre questi ultimi sono caratterizzati da una bassa potenza di calcolo che consente loro solo di prendere decisioni semplici, come ad esempio definire reazioni agli eventi e calcolare piani molto semplici e limitati in spazi piccoli. La novità principale del nostro approccio consiste nel fatto che la simulazione è usata sia dall'unità di supporto (per prendere decisioni complesse), sia dagli agenti (per prendere decisioni semplici).

3.1 Introduzione

Negli ultimi anni, un enorme interesse è cresciuto intorno alle tecnologie ad agenti. Uno dei problemi principali della progettazione degli agenti è quello di definire efficaci meccanismi di supporto alle decisioni. Chiaramente, la scelta di tali meccanismi influenza fortemente il processo di modellazione dell'architettura degli agenti. Prima di introdurre il nostro approccio al decision making negli agenti, è utile proporre una definizione di agente, e successivamente una definizione di agente intelligente. Un agente è un software situato in un ambiente, capace di eseguire azioni autonome nell'ambiente nel quale è collocato al fine di soddisfare i propri obiettivi. Dalla definizione appena proposta è possibile evincere che non esiste agente che non sia collocato in un qualche ambiente. Un ambiente è caratterizzato da alcune proprietà, tra le quali:

- *Accessibile* vs. *inaccessibile*. Un ambiente è accessibile per un agente se esso può osservare ogni proprietà di tale ambiente.
- *Deterministico* vs. *indeterministico*. Un ambiente è deterministico se l'insieme ($E \times A$), dove E è l'insieme dei possibili stati dell'ambiente e A è l'insieme delle azioni che può eseguire nell'ambiente, è composto esclusivamente da singoletti. In altre parole, un ambiente è deterministico se e solo se ogni azione ha un effetto singolo e garantito.
- *Episodico* vs. *non episodico.* Un ambiente è episodico se gli "episodi" sono indipendenti gli uni dagli altri. In altri termini, in ambienti episodici, possiamo vedere la storia dell'agente come divisa in episodi indipendenti, che consistono nel percepire e nell'agire.

- *Statico* vs. *dinamico*. Un ambiente è statico se non avviene alcun cambiamento mentre l'agente sta deliberando.
- *Discreto* vs. *continuo*. Un ambiente è discreto se è presente un numero limitato di percezioni e azioni distinte e chiaramente definite.

Un agente intelligente è caratterizzato dalle seguenti proprietà:

- *Autonomia*: gli agenti caratterizzati da tale proprietà operano senza l'intervento diretto di esseri umani o altri attori, ed hanno un qualche controllo sulle loro azioni, e uno stato interno.
- *Abilità sociale*: gli agenti interagiscono e comunicano con gli altri agenti.
- *Reattività*: gli agenti percepiscono il loro ambiente (che può essere un ambiente fisico, un utente per mezzo di una interfaccia grafica, una collezione di agenti, Internet, o anche la combinazione di alcuni di questi ambienti), e risponde in un tempo opportunamente limitato ai cambiamenti che si verificano in esso.
- *Pro-attività*: gli agenti non operano semplicemente in risposta al loro ambiente, ma, piuttosto, sono capaci di esibire, prendendo l'iniziativa, un comportamento orientato agli obiettivi. Un agente è capace di gestire compiti complessi e di alto livello. Le decisioni su come è meglio dividere tali compiti in sotto-compiti, e in che modo e ordine è preferibile eseguirli, dovrebbero essere prese dall'agente.

Un elenco di alcune delle più note classi di architetture concrete di agenti viene proposto di seguito:

- *Architetture di agenti basate su logica* – nelle quali il *decision making* è realizzato tramite deduzione logica;
- *Architetture di agenti reattive* – nelle quali il *decision making* è implementato in una qualche forma di mapping diretto tra situazioni e azioni;
- *Architetture BDI* (*beliefe-desire-intention*) – nelle quali il *decision making* dipende dalla manipolazione delle strutture di dati che rappresentano le credenze, i desideri e le intenzioni dell'agente; e, infine,
- *Architetture a strati* (anche note come architetture ibride) – nelle quali il *decision making* è realizzato per mezzo di diversi strati software, ognuno dei quali fa del reasoning sull'ambiente a diversi livelli di astrazione.

In questo capitolo proponiamo una architettura multi agente, chiamata *DSA* (*Decentralized and Simulation Based Architecture*) che consente di prendere decisioni complesse (grazie alle elaborazioni di una support unit), o definire un comportamento reattivo o comunque prendere decisioni semplici (grazie alle elaborazioni degli agenti). Gli agenti, operando in un ambiente dinamico, utilizzano la simulazione a supporto dei processi di *decision making*. Nell'architettura proposta, la support unit definisce piani per gli agenti (decisioni complesse), mentre gli agenti, essendo dotati di una potenza di calcolo molto limitata, possono semplicemente prendere decisioni

semplici, come ad esempio definire reazioni agli eventi e calcolare piani molto semplici e limitati in spazi piccoli. La principale novità del nostro approccio è che la simulazione è utilizzata sia dalla support unit (per prendere decisioni complesse), sia dagli agenti (per prendere decisioni semplici).

Organizzazione del capito Il capitolo è organizzato come segue: nel paragrafo 2 riproporremo brevemente il problema della pianificazione: nel paragrafo 3 descriveremo l'architettura *DSA* ed il comportamento di un agente di tipo *DSA*.

3.2 Il problema della pianificazione

Lo scenario che si propone è il seguente: il *goal* di un Agente di tipo *DSA* è di trasformare uno stato $s_0 \in S_0$ (l'insieme degli stati iniziali dell'agente) in uno stato $s_g \in S_g$ (l'insieme degli stati *goal* o stati finali dell'agente). Per raggiungere lo stato s_g partendo dallo stato iniziale s_0, l'agente esegue un piano p, definito da una support unit. Durante l'esecuzione di un piano, lo stato di un agente può cambiare a causa: *(i)* degli effetti dell'esecuzione delle azioni, i quali dipendono fortemente dall'ambiente nel quale le azioni sono eseguite; *(ii)* degli eventi che nel frattempo accadono nell'ambiente nel quale l'agente è collocato.

Dato un possibile insieme di stati $S = \{s_1, \ldots, s_n\}$ ed un insieme di azioni $A = \{a_1, \ldots, a_n\}$, un piano $p = <a_i, \ldots, a_j>$ è una sequenza di azioni appartenenti all'insieme A. L'insieme di azioni definisce una funzione di transizione e l'applicazione di un'azione a_i ad uno stato s_j, denotata con $a_i(s_j)$, restituisce un nuovo stato s_k. La notazione $a_1.a_2(s_j)$ sarà utilizzata al posto di $a_2(a_1(s_j))$.

Definizione 1 *Un Agente di tipo HPA è una tupla $AG = (S_0, S_g, S, P, R)$, dove S denota l'insieme dei suoi stati, $S_0 \subseteq S$ denota l'insieme dei suoi stati iniziali, $S_g \subseteq S$ l'insieme dei suoi stati finali. P denota l'insieme dei piani, dove per ogni $s_0 \in S_0$, esiste $p \in P$ tale che $p(s_0) \in S_g$, e R denota l'insieme delle regole reattive, dove ogni $r \in R$ è una sequenza di azioni tali che esiste $s \in (S - S_0)$ e $r(s) \in S$.*
Un HPA Active Agent $AAG = (s_0, s_g, s, p, q, R)$ che deriva da un agente di tipo HPA $AG = (S_0, S_g, S, P, R)$, è caratterizzato da uno stato iniziale $s_0 \in S_0$, uno stato finale, lo stato corrente (o osservato) s, il piano già eseguito p e il piano da eseguire q.

Un *HPA Active Agent* $AAG = (s_0, s_g, s, p, q, R)$ è in uno stato corretto se $q(s) = s_g$; altrimenti, esso è in uno stato scorretto ($q(s) \neq s_g$).

Definizione 2 *Dato un HPA Active Agent $AAG = (s_0, s_g, s, p, q, R)$, un evento inatteso influente e è un'azione, non necessariamente appartenete ad A, che è eseguita prima di q, tale che $e.q(s) \neq s_g$.*

Quindi, quando un Active Agent $AAG = (s_0, s_g, s, p, q, R)$ si trasforma in un Agente $AAG' = (s_0, s_g, s', p, q, R)$, dove s' è un stato scorretto, si suppone che si sia verificato un evento inatteso influente e, che ha trasformato lo stato corretto s nello stato scorretto s' (ovvero, $e(s) = s'$). Inoltre, se dopo l'esecuzione di un'azione a,

appartenente ad A, l'Agente evolve in uno stato scorretto $s' \neq a(s)$, si assume che si è verificato un evento inatteso influente tale che $a.e(s) = s'$.
La sequenza di azioni che caratterizza un comportamento reattivo è diversa da un piano (ovvero, $P \cap R = \oslash$), dal momento che essa non è concepita per far raggiungere uno stato obiettivo partendo da uno stato iniziale, ma piuttosto per reagire al verificarsi di un evento inatteso influente, parallelamente all'esecuzione del processo di ripianificazione, come descritto nel paragrafo successivo.

Il problema in esame è il seguente: dato un *Active Agent* $A_g = (s_0, s_g, s, p_1 \ldots p_j, p_{j+1} \ldots p_n)$, tale che $p_1 \ldots p_j(s_0) \neq s$, definire un piano $q = q_1 \ldots q_k.p_i \ldots p_n$, con $n - i \geq 0$, tale che $q(s) = s_g$, ovvero modificare il piano $p_{j+1} \ldots p_n$ in un nuovo piano $q_1 \ldots q_k.p_i \ldots p_n$ che consenta di raggiungere lo stato finale s_g partendo dallo stato corrente s.

3.3 Un'architettura basata su simulazione

In questo paragrafo presentiamo la *DSA* (*Decentralized and Simulation based Architecture*). Una rappresentazione grafica della *DSA* è mostrata nella figura seguente.

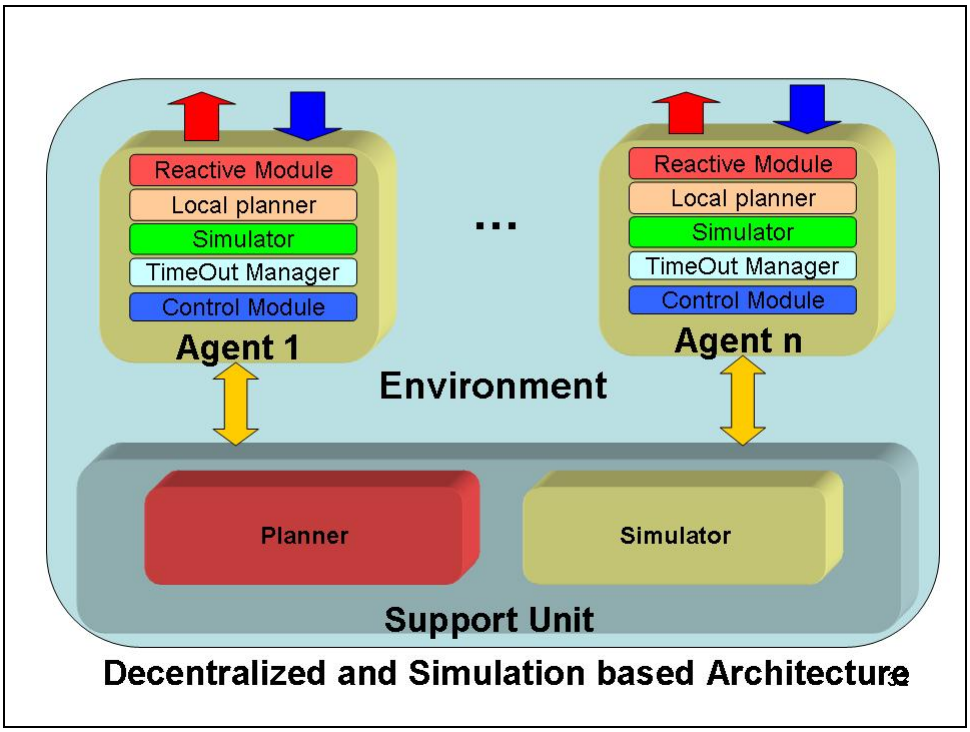

Figura 7: DSA

La support unit, che tipicamente è caratterizzata da notevoli risorse computazionali, garantisce assistenza agli agenti, fornendo loro piani e dati aggiornati sull'ambiente

nel quale essi operano. Gli agenti sono caratterizzati dal disporre solo di una conoscenza parziale dell'ambiente. La conoscenza "locale" a disposizione degli agenti viene continuamente aggiornata per mezzo delle loro percezioni e dei dati ricevuti dalla support unit.

Inoltre, la Support Unit riceve dati relativi alle percezioni degli agenti. Solo gli agenti interagiscono con l'ambiente. La Support Unit può dare assistenza a più di un agente, e gli agenti possono richiedere piani a più di una Support Unit.

Gli agenti sono caratterizzati da cinque moduli:

- Un *Modulo di Controllo*, i cui compiti sono: osservazione dello stato dell'ambiente; elaborazione delle osservazioni, esecuzione delle azioni, monitoraggio dei piani, identificazione di eventi inattesi influenti, attivazione pianificazione locale; attivazione della pianificazione della Support Unit; gestione della condivisione delle informazioni. Molte di queste attività sono eseguite in parallelo.
- Un *Pianificatore Locale*, il cui compito è di generare piani.
- Un *Simulatore*, il cui compito è quello di supportare la scelta dei piani.
- Un *Modulo Reattivo*, il cui compito è quello di definire i comportamenti reattivi. Per definizione di comportamento reattivo intendiamo la selezione di azioni utilizzando lo stato corrente come un indice in una tabella di azioni. Esso è attivato da una richiesta di servizio inviata da Modulo di Controllo.
- Un *Timeout Manager*, il cui compito è quello di generate alcuni time bound.

Assumiamo che:

- L'ambiente non è completamente osservabile,
- L'agente ha solo una conoscenza parziale dell'ambiente.

Il comportamento di un agente di tipo *DSA* viene illustrato di seguito:

Al tempo t_0, nello stato s_0, l'agente inizia a eseguire il piano p. Al tempo t_j, nello stato s_j, si verifica un evento inatteso influente. Come conseguenza dell'evento inatteso influente, l'agente $A_g = (s_0, s_g, s, p, q)$ è in uno stato s, tale che $q(s) \neq s_g$. La strategia dell'agente di tipo *DSA* in risposta a questo evento è la seguente:

1. reagisce all'evento inatteso, trasformando lo stato corrente in uno stato s',
2. inizia il processo di definizione di un piano locale q' tale che $q'(s') = s_g$,
3. manda alla Support Unit una richiesta di ripianificazione in funzione dello stato corrente s'.

La definizione del piano locale e la generazione del piano da parte della Support Unit sono attività parallele.

Pianificazione Locale La prima sotto-attività è quella di definire un insieme ordinato di stati, la finestra di stati $S_w = s_{j-x}, \ldots, s_j, \ldots, s_{j+y}$, dove $x \leq j$ *e* $y \leq n-j$. Il valore di x e y è determinato sulla base della "distanza" tra lo stato corrente s' e gli stati appartenenti

a S_w, dell'osservazione dell'ambiente, e dell'analisi dell'evento che ha causato la trasformazione dello stato prima del verificarsi dell'evento stesso nello stato *s*. Per esempio, se c'è un ostacolo che non permette all'agente di raggiungere lo stato s_j partendo dallo stato s_{j-1}, il valore di *x* viene fissato a 0, in modo tale che la finestra di stati sia composta dagli stati s_j, …, s_{j+y}. Tipicamente, si ha che $x << i$ e $y << n-i$.

Successivamente, l'agente di tipo *DSA* attiva il planner locale. Il planner locale tenta di generare un insieme ordinato di piani. In particolare, per ogni stato $s_w \in S_w$, il pianificatore locale tenta di generare un piano locale tale che $p_w(s') = s_w$. I piani saranno generati nell'ordine definito dall'algoritmo in figura, dove gli operatori $\prec$ e $\preceq$ sono usate per raffrontare piani.

```
boolean Generate_plans;
begin
    generate plan p_i;
    best_plan = p_i;   new_plan = true;
    h = max(1, x);   k = max(1, y);
    while (h + k < h + y and new_plan and not TimeOut ) do
    begin
        new_plan = false;
        if (k < y) then begin
            generate plan p_{i+k};
            if (p_{i+k} ⪰ best_plan) then begin
                best_plan = p_{i+k};
                new_plan = true;
                k = k + 1;
            end
            else
                k = y;
        end;
        if (h < x) then begin
            generate plan p_{i-h};
            if p_{i-h} ≻ best_plan then begin
                best_plan = p_{i-h};
                new_plan = true;
                h = h - 1;
            end
            else
                h = x;
        end
    end
end.
```

```
boolean Agent_procedure(executed plan p = [a_1....a_j],
                        plan to be executed q = [a_{j+1}....a_n];
                        initial state s_0; expected state s_j;
                        goal state s_g; final time t_g)
var
    s current state
    t current time
begin
    if (s_0 == s_g) then return true;
    if (s == s_j) then
        if (s_g not reachable from s_j using q in time t_g-t) then begin
            identify a support unit SI
            (q', s_f, t_f) = SI.newplan(s, s_g, t_g)
            Agent_procedure([], q', s, s, s_f, t_f)
        end
        else begin
            apply a_{j+1} to the current state s_j;
            let s_{j+1} = a_{j+1}(s_j) the expected state at step j + 1
            Agent_procedure([a_1....a_{j+1}], [a_{j+2}....a_n], s_0, s_{j+1}, s_f, t_f)
        end;
    // an unexpected event e occurred or s ≠ s_i
    if (there is an unexpected e) and (s ≠ s_j) then begin
        react(s, s_j, e);
        parallel begin
            begin
                q' = localplan(s, s_j, p, q, t_g);
                s_f = s_g);  t_f = t_g);
            end
            begin
                identify a support unit SI
                (q'', s_f, t_f) = SI.newplan(s, s_g, t_g)
            end
        parallel end;
        q̂ = compare(q', q'');
        Agent_procedure([], q̂, s, s, s_f, t_f)
    end
end.
```

Figura 8: algoritmi

In altre parole, il pianificatore locale cerca di generare un piano che potrebbe portare il sistema nello stato s_w, che appartiene all'insieme di stati S_w. S_w è un sottoinsieme dell'insieme degli stati dai quali, eseguendo il piano corrente o una porzione di esso, è possibile raggiungere lo stato goal s_g. La definizione della finestra di stati e la definizione dell'ordine nel quale i piani sono generati sono basate sull'assunzione che, data una funzione di prossimità, lo stato scorretto *s'* è tipicamente non lontano dallo stato s_i, dove $p(s_0) = s_i$. L'obiettivo del pianificatore è quindi quello di consentire

all'agente di eseguire il piano corrente o una porzione di esso. Come abbiamo visto, il pianificatore locale genererà più di una soluzione. Le soluzioni saranno valutate con il supporto del simulatore. L'agente ha, infatti, conoscenza riguardo gli stati nell'intorno dello stato *s'* (grazie all'uso delle proprie percezioni e dei dati ricevuti dalla Support Unit) e riguardo lo stato e il comportamento degli altri agenti. Essendo un piano locale tipicamente molto semplice, è ragionevole simulare il comportamento degli agenti in funzione di tale piano, al fine di valutare in anticipo gli effetti della sua esecuzione. Usando l'output del simulatore, il Modulo di Controllo sceglie un certo piano *p'*, dove q_w *.p'*(*s'*) = s_g, q_w è la porzione del piano corrente le cui precondizioni sono soddisfatte dallo stato s_w, e *p'*(*s'*) = s_w. A questo punto, un nuovo ed efficace piano *q'* = q_w.*p'* è disponibile e pronto per essere eseguito. Se l'agente è ancora in attesa della conclusione del processo di ripianificazione della Support Unit, esso comunica a quest'ultima che la precedente richiesta può essere ignorata.
Se il pianificatore locali non riesce a generare alcun piano, l'agente deve attendere che un nuovo piano sia generato dalla Support Unit. In questo caso, la Support Unit genera un nuovo piano *q''* tale che *q''*(*s'*) = s_g.
L'algoritmo *Agent_procedure* riportato in figura mostra come lavora il modulo di decisione. In questo algoritmo, l'esecuzione parallela termina quando, dopo che è terminata l'esecuzione di uno dei due rami paralleli, è trascorso un intervallo di tempo di attesa della fine dell'esecuzione dell'altro ramo.
La funzione *compare* considera due possibili piani definiti dal pianificatore locale e dal pianificatore di cui è dotata Support Unit. Se entrambi sono disponibili, tale funzione seleziona il migliore, altrimenti viene preso in considerazione l'unico disponibile.

4

Il progetto Promis

Sommario: Nel quadro delle attività di dottorato, è stato possibile applicare alcuni temi inerenti il replanning di agenti nel quadro di un progetto di ricerca denominato Promis e di cui, per completezza, si fornisce di seguito prima una descrizione generale (Sezione 4.1). Successivamente, in Sezione 4.2, si discuterà lo stato dell'arte relativo all'adozione di tecnologie ad agenti in ambito logistico [PG08]. Nella Sezione 4.3 seguente successiva si illustrerà, infine, quanto sviluppato in merito alle tematiche di planning e replanning nel contesto del progetto in questione [P08].

4.1 Descrizione del progetto Promis

Le motivazioni alla base del progetto di ricerca Promis nascono dalla consapevolezza che l'attuale stadio di maturità dei settori scientifici e tecnologici dell'informatica, delle telecomunicazioni e dell'ottimizzazione consente la definizione di metodi, modelli, algoritmi, tecniche e strumenti integrabili ai fini dell'ottenimento di un sistema innovativo per la gestione di terminali marittimi. In particolare, le attività del progetto sono state finalizzate all'ottenimento di nuove conoscenze e alla definizione di componenti di un sistema fondato su una visione orientata ai processi delle attività di logistica integrata che avvengono all'interno di grandi hub portuali collegati a reti di trasporto intermodali. A partire da questa visione, il sistema si occupa della gestione, dell'analisi e dell'ottimizzazione dei processi logistici che avvengono all'interno di grandi terminal portuali.

A tal fine, il progetto di ricerca si pone come obiettivo di identificare il complesso punto di equilibrio tra una soluzione funzionale innovativa del mondo logistico e una soluzione informatica di supporto a questa soluzione funzionale, facente uso di architetture, tecnologie, e soluzioni informatiche avanzate ed ottimizzate che permettano uno sviluppo, integrazione, ed evoluzione economicamente efficiente (nel breve e nel lungo termine).

Sono state previste, pertanto, la realizzazione di attività di Ricerca Industriale (RI) riguardanti:

- l'analisi di scenario inerente i processi logistici e la struttura organizzativa di un terminal per la movimentazione di container effettuata mediante modelli, metodi e strumenti per la rappresentazione di processi;

- i modelli e i metodi di ottimizzazione delle operazioni di banchina e di piazzale;
- le tecniche e gli ambienti software per la simulazione integrata dei processi logistici;
- le metodologie e le tecniche informatiche a supporto dei processi logistici e per il supporto decisionale;
- architetture innovative basate su griglia e orientate ai servizi per la definizione di sistemi informativi destinati alla gestione di terminal per la movimentazione di container.

PROMIS si propone, dunque, di consentire una gestione innovativa, efficace ed efficiente dei processi logistici che sottendono alle attività del terminal marittimo di Gioia Tauro. In particolare, ci si è proposto di ideare e realizzare funzionalità avanzate ed intelligenti non presenti nelle tecnologie attualmente disponibili attraverso lo sviluppo di un insieme di risultati di ricerca applicata all'interno delle aree di interesse precedentemente elencate. Tra l'altro, è stato di interesse, nel corso del progetto, lo studio e l'ideazione di metodi per la facilitazione delle interazioni tra il porto e il mondo esterno. Ciò perché i processi logistici che avvengono nel porto hanno impatto sulle filiere produttive delle aziende costituenti l'indotto del porto stesso.

La definizione delle funzionalità innovative caratterizzanti il sistema PROMIS richiedono la realizzazione di attività di ricerca industriale mirate ad affrontare le tematiche riportate di seguito. In particolare, le linee di ricerca del progetto nelle aree della logistica ma anche, in una certa misura, nelle aree di contorno e supporto, quali le tecnologie informatiche e delle comunicazioni, faranno costante riferimento ad un insieme di reali problematiche applicative di riferimento, quali:

- Pianificazione integrata delle operazioni di attracco delle navi, carico e scarico dei contenitori e gestione del piazzale;
- Modelli per la gestione dell'intermodalità e strategie di medio-lungo periodo;
- Gestione del "routing" dei mezzi per la movimentazione in piazzale;
- Simulazione e Ottimizzazione mediante Simulazione;
- Metodologie e tecniche informatiche a supporto dei processi logistici;
- Metodologie e tecniche informatiche a supporto dei processi decisionali;
- Architetture software innovative.

In particolare, per quanto riguarda le problematiche relative alle metodologie e tecniche informatiche a supporto dei processi logistici, molto interessante è l'ambito della soluzione dei problemi di pianificazione. Quest'ultimo può giovarsi, accanto all'uso di tecniche di ottimizzazione tipiche della ricerca operativa, anche dell'uso di tecniche diverse. Generalmente, gli approcci correnti sono classificati in quattro aree: Ottimizzazione Combinatoria, Programmazione a Numeri Interi, Constraint Programming e Local Search. Tuttavia, le soluzioni a problemi di pianificazione richiedono l'uso di diverse competenze provenienti da campi diversi quali, l'intelligenza artificiale e la ricerca operativa ed interessano diversi aspetti, come, ad esempio, scheduling, sequencing, routing, e assegnamento di risorse. Inoltre, al fine di gestire situazioni complesse e

dinamiche, come quelle richieste nella gestione di *Container Terminal* (CT), possono risultare utili le nuove e alternative soluzioni basate sull'uso di agenti software, recentemente proposte in letteratura. Pertanto, in questa linea di ricerca è stato affrontato, tra l'altro, lo studio del paradigma degli agenti software per la soluzione di problemi di pianificazione. Per quanto riguarda gli a*genti software intelligenti*, è stato recentemente mostrato che la modellazione e simulazione di processi decisionali mediante agenti può fornire soluzioni alternative per il miglioramento della gestione di CT. In tale ambito gli agenti operano su oggetti situati in un dato ambiente ed interagiscono tra loro. Infine l'uso degli agenti può risultare utile per gestire anche situazioni in cui le conoscenze sono incerte ed il loro comportamento può essere definito anche utilizzando soluzioni definite mediante vincoli. L'uso degli agenti può risultare molto utile in situazioni dinamiche dove il calcolo della soluzione ottima "statica" richiederebbe un tempo molto lungo, inadatto a rispondere a situazioni di emergenza. In questo caso può essere utile calcolare una soluzione sub-ottima in tempi brevi, sulla base di soluzioni ottenute dall'analisi di dati "storici", sia influenzando il comportamento degli agenti che restringendo, eventualmente, il dominio delle possibili soluzioni di problemi di vincoli.

4.1.1 Obiettivi realizzativi riguardanti gli agenti intelligenti

Nel presente paragrafo sono descritti gli obiettivi realizzativi riguardanti le tematiche relative agli agenti intelligenti.

In particolare, tali obiettivi consistono di attività di ricerca industriale inerenti la definizione di una architettura per la pianificazione, il coordinamento e la gestione delle attività di un "terminal container" basata su agenti intelligenti. Più precisamente, per quanto riguarda le tematiche riguardanti gli agenti intelligenti, sono state affrontate le seguenti attività:

- Studio riguardante il comportamento di agenti associati ad attori di CT;
- Definizione di Multi Agent System (MAS) per la gestione di attività di CT;

4.1.2 Agenti e terminal container

I sistemi basati su agenti rappresentano un nuovo paradigma di riferimento per l'analisi, la progettazione e la realizzazione di sistemi software complessi. I metodi di progettazione, infatti, trattano la complessità utilizzando tre meccanismi fondamentali: *(i)* la decomposizione del problema/sistema in piccoli problemi/sistemi, la cui trattazione può essere affrontata con successo e senza inefficienze con i mezzi disponibili; *(ii)* l'astrazione, ovvero, quel processo che consente di definire modelli del problema/sistema che ne enfatizzano alcuni dettagli e proprietà, nascondendone altri; *(iii)* l'organizzazione ovvero il processo di identificazione e gestione delle relazioni fra i vari sotto problemi/sistemi componenti. La complessità da trattare assume spesso la forma di una gerarchia di sottosistemi la cui organizzazione ha un ruolo fondamentale nel successo dell'applicazione. Nella gerarchia si procede identificando i compiti, i ruoli dei vari componenti, per poi affrontare il problema

dell'interazione fra di essi. In molti sistemi software non è possibile centralizzare il controllo. Informazioni e unità di controllo sono distribuite in più nodi e in più parti del sistema. Questi sistemi possono effettivamente lavorare efficientemente se sono in grado di dimostrare autonomia e capacità di interazione. Alla luce di quanto detto, i sistemi multi-agente si configurano come la metafora naturale dei moderni sistemi software. Un agente, infatti, *(i)* è un'entità risolutrice di problemi chiaramente identificabili con confini ed interfacce ben definite; *(ii)* è situato in un particolare ambiente e/o contesto che è in grado di percepire e sul quale è in grado di agire; *(iii)* è progettato per ricoprire uno specifico ruolo e per raggiungere propri obiettivi; *(iv)* è autonomo, cioè in grado di controllare sia il proprio stato interno che il proprio comportamento; *(v)* è capace di esibire un comportamento flessibile (dipendente dal contesto, reattivo e proattivo) orientato alla risoluzione di un problema e/o al raggiungimento di un obiettivo.

Recentemente, l'uso dei sistemi multi-agente è stato proposto anche per la soluzione di problemi di pianificazione di attività di container terminal. I sistemi multi-agente, infatti, possono offrire strumenti alternativi per le attività di coordinamento, pianificazione e gestione di CT, in quanto tale dominio è caratterizzato da una moltitudine di attori tra loro interagenti in un ambiente costituito da una varietà di risorse.

L'uso di agenti software offre un nuovo modo di comprendere e risolvere problemi complessi tipici della gestione di CT. Inoltre, l'uso di più agenti porta alla definizione di un sistema multi-agente che pone problemi non ancora completamente risolti come quelli relativi alla definizione di strategie e protocolli di cooperazione per il raggiungimento di obiettivi comuni.

Gli agenti intelligenti trovano utile applicazione non solo specificatamente come sistemi di gestione di CT, ma più in generale il loro uso offre notevoli vantaggi in ambito logistico [PG08].

4.2 Vantaggi e applicazioni di sistemi multi agente in ambito logistico

Anche il settore logistico ha dovuto affrontare, negli ultimi anni, come del resto è accaduto nella maggioranza dei settori economici, le sfide imposte dalla globalizzazione. L'assottigliamento dei margini di guadagno e gli enormi costi hanno costretto le compagnie a rivedere e modificare le loro offerte di prodotti e servizi e i loro processi di business.

Obiettivo della logistica, ad una prima analisi superficiale, è quello del trasporto di materiali e beni da una locazione a un'altra, nella quale deve essere soddisfatta una specifica richiesta di tali beni. Tale obiettivo è soggetto a diversi vincoli, come ad esempio la finestra temporale all'interno della quale il bene deve essere consegnato, oppure una particolare sequenza di consegne da rispettare, etc.. Sono tipicamente disponibili diversi mezzi di trasporto e, allo stesso tempo, le risorse sono limitate. In tali domini è inoltre generalmente necessario fare un certo numero di scelte riguardo le attività di scheduling e di routing con vincoli temporali, basandosi spesso su

conoscenza incompleta. Queste caratteristiche sono comuni sia alla logistica negli ambienti di produzione, sia alla logistica relativa ai trasporti. Le problematiche appena presentate sono adatte ad essere affrontate grazie all'utilizzo di tecnologie basate su sistemi multiagente. Per giustificare tale affermazione, saranno messe in evidenza di seguito alcune caratteristiche dei domini e, successivamente, alcune proprietà dei sistemi multi agente.

In sintesi, un dominio logistico è tipicamente caratterizzato dall'essere:

- *Decentralizzato*. In un dominio logistico, infatti, le organizzazioni che partecipano ai processi che lo caratterizzano sono geograficamente distribuite, e devono necessariamente cooperare per raggiungere i propri obiettivi. Per conseguire tali obiettivi, ogni organizzazione si serve dei propri processi logistici (servizi). Quindi, il dominio logistico non è decentralizzato semplicemente ed esclusivamente a causa di una dislocazione geografica distribuita, ma anche e soprattutto perché le organizzazioni coinvolte sono caratterizzate da un alto grado di autonomia. In particolare, esse hanno obiettivi propri, spesso non coincidenti (se non in contrasto) con gli obiettivi delle altre organizzazioni; hanno autonomia decisionale, e non sono controllate da un'organizzazione posta a livello superiore; esse, infine, potrebbero essere riluttanti a fornire informazioni alle altre organizzazioni coinvolte nei processi logistici.
- *Dinamico.* In un dominio logistico, infatti, cambiano continuamente gli obiettivi, le *capabilities* delle organizzazioni (il tipo e la disponibilità dei servizi che esse forniscono) e le conoscenze e le informazioni a disposizione.
- *Aperto*. Spesso, in un dominio logistico, le organizzazioni entrano ed escono continuamente da esso.

Per loro natura, i sistemi logistici sono modulari, ed essi sono quindi scomponibili in entità separate, come strumenti di produzione, macchinari, operatori, ordini, pacchi, parti, prodotti, etc.. Inoltre, è necessaria la coordinazione di vari elementi logistici e l'allocazione di risorse limitate.

I sistemi multiagente sono caratterizzati, generalmente, dal garantire supporto al parallelismo, dall'essere robusti e scalabili. Tali sistemi, inoltre, mostrano grande utilità in contesti in cui è desiderabile avere a disposizione tecnologie che rendano efficace ed efficiente lo scambio di informazioni tra organizzazioni, o, a livello più basso, tra sistemi software eterogenei. Infatti, in tali contesti, gli agenti non vengono utilizzati semplicemente come strumenti che rispondono a precisi standard (ad esempio, FIPA) per lo scambio di dati, ma piuttosto vengono sfruttate le potenzialità offerte da tale tecnologia per elaborare tali dati e trarre conclusioni dai risultati delle elaborazioni. Gli agenti, inoltre, possono istruire, in maniera autonoma e proattiva, altri agenti al fine di far compiere loro azioni. Tali caratteristiche sono di fondamentale importanza nei contesti in cui risultino distribuiti i dati, il controllo, le competenze. La possibilità di trattare dati distribuiti e di sfruttare la naturale predisposizione dei sistemi multiagente alle elaborazioni parallele consentono di

trattare grandi quantità di dati in tempi più brevi rispetto a un approccio centralizzato; inoltre, utilizzando gli agenti è in genere più semplice concepire algoritmi per elaborazioni rapide (con soluzioni sub-ottime). La possibilità che gli agenti possano controllare in maniera automa e proattiva altri agenti, rende naturale la decomposizione automatica del problema affrontato in diversi problemi più semplici del problema dal quale derivano. Ciò introduce una maggiore semplicità concettuale.

Tutto ciò mette in evidenza che le caratteristiche tipiche dei sistemi multiagente sembrano renderli particolarmente adatti ad una loro applicazione in ambito logistico. Essi, inoltre, sembrano trovare naturale applicazione in contesti in cui sono evidenti i limiti di un approccio centralizzato. In particolare, i sistemi multiagente sono particolarmente indicati in domini in cui sia richiesta:

- l'integrazione e l'interazione con diverse basi di conoscenza;
- la capacità di risolvere conflitti tra obiettivi diversi;
- l'elaborazione di dati con vincoli temporali.

Alcune delle più interessanti applicazioni dei sistemi multiagente alla logistica sono le seguenti:

- Sistemi di simulazione per il *traffic modelling;*
- Sistemi di supporto alle decisioni per le spedizioni;
- Sistemi di planning logistico;
- Sistemi supporto alle decisioni nell'ambito del trasporto merci marittimo;
- Sistemi di *dispatching* di veicoli;
- Sistemi di scheduling per il trasporto su rotaia.

Nei seguenti paragrafi saranno approfonditi alcune interessanti applicazioni dei sistemi multi agente in ambito logistico.

La simulazione ad agenti in domini logistici

Alcune caratteristiche degli agenti intelligenti rendono questi ultimi molto adatti a essere utilizzati come strumenti di simulazione in ambito logistico. In particolare, gli agenti possono essere visti come entità capaci di interagire autonomamente ed efficacemente con l'ambiente nel quale sono collocati, al fine di eseguire compiti auto assegnati o assegnati da altri. Le caratteristiche che li rendono particolarmente utili nell'ambito di simulazione sono l'autonomia, la proattività, le capacità di coordinazione e comunicazione. Ciò rende molto più semplice modellare domini complessi come quello logistico, nel quale gli agenti possono efficacemente modellare anche i processi e le attività. La simulazione basata su sistemi multi agente (*Multi Agent Based Simulation - MABS*) è sostanzialmente diversa dagli altri tipi di simulazione note in letteratura. In una *MABS*, le entità simulate sono modellate e implementate per mezzo di agenti. Tale caratteristica è in forte contrasto con le tecniche di macro-simulazione, nell'ambito delle quali vengono prese in

considerazione esclusivamente le caratteristiche medie della popolazione in esame. Più in particolare, nelle tecniche basate su macro simulazione, l'insieme degli individui è visto come una struttura caratterizzata da un certo numero di variabili; in esse vengono tipicamente utilizzati modelli matematici. Viceversa, nell'ambito delle tecniche di micro simulazione (come la *MABS*) la struttura è modellata partendo dai comportamenti dei singoli individui. La simulazione basata su sistemi multi agente sembra essere particolarmente appropriata per domini caratterizzati da un alto grado di decentralizzazione e distribuzione. Viceversa, la modellazione basata su equazioni trova migliore applicazione in sistemi centralizzati e nei quali la componente dinamica è caratterizzata fortemente da leggi fisiche, piuttosto che da flussi di informazione, come accade tipicamente nei contesti logistici. Se si mette a confronto la simulazione basata su sistemi multi agente con altre tecniche classiche (ad esempio, la *DES – Discrete Event Simulation*) possono essere messi in evidenza numerosi vantaggi:

- Nella *MABS* c'è una precisa relazione tra le entità del contesto reale, le entità del modello e le entità rappresentate nel software di simulazione. Questa caratteristica semplifica enormemente la progettazione e l'implementazione del software, che può essere facilmente ben strutturato;
- La *MABS* supporta la modellazione e l'implementazione di comportamenti proattivi. Ciò è molto utile, ad esempio, quando si simulano i processi di decisione di essere umani, che ovviamente sono in grado di prendere iniziative e agire senza la presenza di stimoli esterni;
- Poiché ogni agente è tipicamente implementato con un processo a se stante ed è capace di comunicare con gli altri agenti per mezzo di un linguaggio comune, è possibile aggiungere o rimuovere agenti durante la simulazione, senza alcuna interruzione. Ciò consente di generare simulazioni realistiche di ambienti estremamente dinamici, come quelli che caratterizzano il dominio logistico;
- E' possibile definire i modelli e programmare a livelli molto alti, rendendo possibile anche a chi non è un programmatore di comprendere o addirittura produrre i software di simulazione;
- La *MABS* supporta elaborazioni distribuite in modo molto naturale. Poiché ogni agente è tipicamente implementato come un pezzo di software a se stante che corrisponde a un processo (o a un *thread*), è naturale eseguire gli agenti su macchine diverse. Questo consente di ottenere migliori *performance* e una maggiore scalabilità.

La simulazione basata sistemi multi agente sembra quindi essere particolarmente appropriata per domini caratterizzati da un alto grado di decentralizzazione e distribuzione. Ciò li rende particolarmente adatti a essere utilizzati, come strumenti di simulazione, nel dominio logistico.

La ripianificazione per mezzo di agenti in ambito logistico

E' stato osservato che, in ambito logistico, riveste ruolo cruciale per il successo delle organizzazioni l'utilizzazione ottimale delle risorse disponibili. Per perseguire tale successo, molte organizzazioni si sono dotate di sistemi informatici per migliorare i processi logistici. A tal proposito, una caratteristica che rende interessante l'applicazione della tecnologia ad agenti al dominio logistico è che essa si è dimostrata particolarmente efficace nell'ambito della ripianificazione in ambienti dinamici. Prima di argomentare sul tema, sembra opportuno riprendere alcune sintetiche descrizioni dei concetti che saranno nuovamente trattati. Un *piano (plan)* è una sequenza di azioni da eseguire al fine di raggiungere uno o più obiettivi. La *pianificazione* (*planning*) è il processo di generazione di piani. Se il piano generato è corretto, se l'ambiente in cui esso viene eseguito è statico e deterministico e se tra l'esecuzione della prima e l'esecuzione dell'ultima azione non viene eseguita alcuna altra azione, allora, al termine dell'esecuzione del piano stesso, sarà raggiunto l'obiettivo. Purtroppo, in contesti reali, questi requisiti in genere non sono presenti: gli ambienti in cui vengono eseguiti i piani non sono statici, e, essendo la conoscenza dell'ambiente a disposizione del *planner* (l'entità che genera il piano) incompleta, essi possono essere visti a tutti gli effetti come ambienti non deterministici. Nel corso dell'esecuzione del piano si potrebbero verificare eventi inattesi, ovvero eventi non prevedibili sulla base della conoscenza (in genere incompleta) dell'ambiente al tempo della generazione del piano. Inoltre, non c'è alcuna garanzia che l'entità che esegue il piano sia l'unica ad eseguire azioni durante l'esecuzione dello stesso. A causa delle motivazioni appena descritte, non esiste la certezza che l'esecuzione del piano, sebbene quest'ultimo sia corretto, porti al raggiungimento dell'obiettivo.

Gli approcci classici, sebbene consentano di creare automaticamente piani, tipicamente non supportano (o supportano solo marginalmente) i processi di ripianificazione e di *plan-recovery* necessari nel caso in cui si verifichino eventi inattesi ed eventualmente non conosciuti. Più in generale, è stato osservato che gli approcci classici alla pianificazione nel settore logistico presentano limiti nel caso in cui il grado di dinamicità del sistema sia elevato. Ciò è dovuto al fatto che i sistemi prodotti con tali approcci sono tipicamente pensati per processi di trasporto relativamente stabili e ripetitivi. Purtroppo essi spesso non hanno un buon comportamento in ambienti "turbolenti", quando è necessario modificare i plan prodotti in tempi molto brevi (in *real time*, a *run time*). I domini logistici, d'altro canto, sono spesso caratterizzati da un alto grado di dinamismo, e ciò ha portato alla ricerca di nuove soluzioni tecnologiche. Uno degli approcci più interessanti per fornire una valida risposta al problema del replanning in ambienti fortemente dinamici è quello di introdurre sistemi basati su agenti. Uno dei vantaggi portati spesso da tali sistemi, infatti, è quello di avere a disposizione in tempi rapidi efficaci ripianificazioni.

Questa, e altre problematiche e caratteristiche delle applicazioni logistiche, ha fatto sì che negli ultimi anni un sempre maggiore interesse sia cresciuto intorno alle applicazioni delle tecnologie ad agenti a tale ambito.

4.2.1 Panoramica su applicazioni di sistemi multi agente in ambito logistico

Nella presente sezione sarà proposta una panoramica delle più significative applicazioni di sistemi multiagente in ambito logistico.

Alcuni tra i più interessanti sistemi multiagente applicati alla logistica sono i seguenti:

- *I-Scheduler*, una delle più interessanti applicazioni commerciali di Magenta, un framework basato su agenti. Esso è basato, come Magenta, su tre componenti principali: l'*Ontology Management Toolkit*, per mezzo del quale è possibile definire la conoscenza di dominio sotto forma di ontologie; il *Virtual Marketplace Engine*, una sorta di sistema operativo per agenti; *Java Enterprise Edition*.
- *TeleTruck*, un sistema di supporto alle decisioni per i processi di *dispatching*. L'architettura di tale sistema è incentrata sul concetto di sistema multiagente olonico; in un sistema di questo tipo, un agente che dal punto di vista di un osservatore esterno sembra essere una singola entità, potrebbe essere in effetti composto da vari sotto-agenti.
- *PLATFORM,* una estensione di TeleTruck. Esso si basa su due componenti principali: l'*ITP (Intermodal Trasport Planner)*, a cui sono delegate attività di pianificazione; il *sistema di simulazione*, il cui compito è appunto quello di simulare i processi di trasporto, per ottenere valutazioni sulla fattibilità dei piani generati dall'IPT e, più in generale, sulle prestazioni dei *terminal* intermodali.
- *MASCOT* (*Multi-Agent Supply Chain cOordination Tool*). Obiettivo principale di MASCOT è di supportare le attività di pianificazione e di scheduling di supply chain.
- *LS/ATN* (*Living Systems Adaptive Transportation Networks*). LS/ATN è un sistema basato su agenti che supporta i processi di *dispatching* delle compagnie operanti nel settore logistico.

I sistemi appena introdotti saranno descritti ora descritti in maniera più approfondita.

I-Scheduler

I-Scheduler è una delle più interessanti applicazioni commerciali di Magenta, un framework basato su agenti. Sviluppato per la Tankers International, è un sistema di scheduling logistico che fornisce supporto alle decisioni per la gestione delle attività di 46 VLCC (*Very Large Crude Carriers*). Tankers International è una società leader nel mercato del trasporto del petrolio. Quest'ultimo, per sua natura, subisce repentine e inaspettate fluttuazioni sia per quanto riguarda le rotte che i mezzi che lo trasportano devono seguire, sia per quanto riguarda i costi. E' ovvio, quindi, che risulta di vitale importanza avere a disposizione uno strumento che consenta di gestire in maniera efficace ed efficiente lo scheduling delle attività. Proprio per perseguire tale obiettivo, la Tankers International ha utilmente adottato *i*-Scheduler. Le interdipendenze tra le varie componenti del sistema, e i vincoli, sono modellati e memorizzati per mezzo

dello Ontology Management Toolkit, un modulo per la definizione e la gestione di ontologie di cui si parlerà in seguito.

I-Scheduler è basato su un sistema multiagente che consente di ottenere un'efficiente soluzione al problema affrontato.

Il tipico ciclo del sistema è il seguente. Sono innanzitutto definiti gli eventi e creati gli agenti che dovranno reagire ad essi. Ad esempio, in questo contesto un evento potrebbe essere rappresentato dall'esigenza di far navigare un cargo da una località a un'altra. Fatto questo, il sistema inizia l'esecuzione del piano a disposizione. Tale processo potrebbe essere interrotto dal verificarsi di un evento inatteso. In tal caso gli agenti rispondono negoziando una soluzione alternativa. La negoziazione è comunque eseguita in modo tale che sia perseguito l'obiettivo di massimizzare l'utilità globale, calcolata in funzione di determinati criteri, e rispettando eventuali vincoli. A tal fine, saranno coinvolti tutti gli agenti che possono in qualche modo facilitare e rendere più rapido l'ottenimento di una soluzione. Facendo riferimento al verificarsi dell'evento inatteso poco prima descritto, qualora non fosse possibile consentire al cargo di spostarsi da un punto a un altro, potrebbe ad esempio essere proposto agli agenti che lo rappresentano uno scambio con un altro cargo o un cambio di destinazione. Il processo di negoziazione stesso può fornire ampia documentazione su come viene generata la proposta di scheduling, e questo è di fondamentale importanza, in quanto tipicamente uno delle caratteristiche più gradite di uno strumento di supporto alle decisioni è quello di fornire ampia e chiara giustificazione alle proposte fatte.

Riassumendo, l'introduzione del sistema oggetto di discussione ha portato numerosi vantaggi, tra i quali:

- Possibilità di rispondere velocemente al verificarsi di eventi inattesi;
- Possibilità di documentare in maniera precisa la conoscenza di dominio, per mezzo della *Domain Ontology*;
- Possibilità di esplicare all'utente i perché di una certa proposta, documentando il processo di negoziazione tra agenti.

I-Scheduler è una estensione del framework Magenta, che sarà descritto nel seguente paragrafo.

Magenta

MAGENTA è un software commerciale per lo sviluppo e la gestione di sistemi multiagente che trova applicazione in svariati domini. In questo paragrafo sarà presentata una applicazione di tale software in ambito logistico. Prima di far ciò, però, sembra utile presentare Magenta in termini più generali, esponendo a grandi linee le caratteristiche dell'architettura e le funzionalità principali.

Magenta è basato su tre componenti principali:

- *Java Enterprise Edition*, il linguaggio su cui si basa la tecnologia ad agenti di Magenta;

- L'*Ontology Management Toolkit*, uno strumento per formalizzare la conoscenza di dominio per mezzo di ontologie. Queste ultime rappresenteranno le basi di conoscenza usate dagli agenti nei processi decisionali;
- Il *Virtual Market Engine*, che rappresenta il cuore di Magenta. Esso mette a disposizione strumenti per:
    - Creare ed "eseguire" agenti;
    - Gestire i permessi degli agenti per l'accesso alla base di conoscenza;
    - Monitorare a *run time* il sistema.

Nella seguente figura è mostrata una rappresentazione grafica della architettura di Magenta:

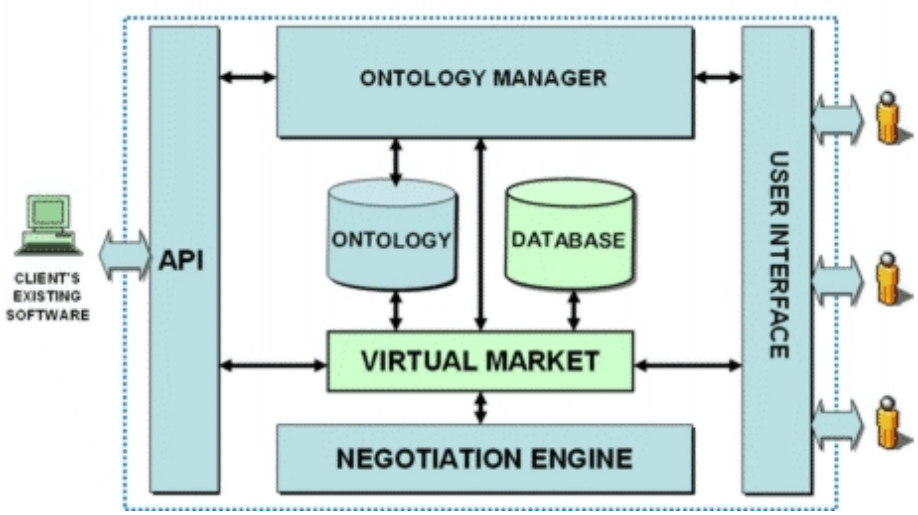

Figura 9: Architettura di Magenta

Una delle più interessanti applicazioni commerciali di Magenta è *i*-Scheduler, un software impiegato in contesti logistici.

Il seguito di questo paragrafo è organizzato come segue. Innanzitutto, saranno presentati in dettaglio l'Ontology Management Toolkit e il Virtual Market Engine mentre. Successivamente, sarà fornita una panoramica sulle caratteristiche principali di *i*-Scheduler.

L'Ontology Management Toolkit

L'Ontology Management Toolkit è lo strumento messo a disposizione degli utilizzatori di Magenta per rappresentare formalmente la conoscenza di dominio per

mezzo di ontologie. Per i nostri scopi un'ontologia è la rappresentazione formale di un dominio di conoscenza e si esplica in un insieme di concetti correlati da associazioni, links, vincoli.

In pratica, quindi, un'ontologia consiste della specificazione di oggetti e delle loro caratteristiche (proprietà) e delle relazioni che li legano.

Ad esempio, in un'ontologia che modella il dominio dei mercati immobiliari, si ritrovano oggetti quali "appartamento", "immobile", "terreno" e "terreno edificabile" e le relazioni "terreni e appartamenti sono immobili" e "terreno edificabile è un terreno".

Ai fini dell'utilizzo delle ontologie a supporto dei sistemi multiagente, mettiamo in evidenza due interessanti caratteristiche di tale formalismo:

- Le ontologie sono tipicamente *machine readable*;
- Le ontologie possono rappresentare vocabolari condivisi.

Vediamo adesso più in dettaglio come l'Ontology Management Toolkit supporta l'utente nella definizione e la gestione di ontologie. Innanzitutto, Magenta consente di creare ontologie per mezzo di comodi tool visuali.

Il linguaggio di specifica delle ontologie utilizzato in Magenta è molto simile a OWL, di cui di fatto rappresenta una estensione, offrendo esso in più alcuni costrutti utili nel contesto applicativo dei sistemi multiagente. E' comunque garantita la possibilità di creare le ontologie in OWL. Come i classici linguaggi per la specifica di ontologie, anche il linguaggio a disposizione degli utilizzatori di Magenta ha come elemento centrale l'idea di "concetto". I concetti possono essere astratti o concreti, elementari o composti, reali o immaginari; in breve, un concetto è una rappresentazione di un'entità appartenente all'esistente. Un concetto può anche essere la descrizione di un processo, di un'azione, di una strategia, di un ragionamento, etc. Un concetto rappresenta una classe di oggetti la cui esistenza è indipendente da quella di altri oggetti. Una volta definiti i concetti relativi al dominio che si intende modellare, è possibile specificare i relativi *attributi*. In Magenta, in particolare, gli attributi sono classificati come *slot* o come *relazioni*. Agli slot corrispondono, nella programmazione ad oggetti, le variabili di una classe. Ad esempio, il concetto "persona" potrebbe essere caratterizzato dall'attributo di tipo slot "età". Gli attributi di tipo relazione, invece, sono caratterizzati dal fatto che il loro valore identifica un altro concetto. Una volta definiti i concetti, è ovviamente possibile definire istanze di questi ultimi.

Una base di conoscenza in Magenta è definita dai seguenti elementi:

- Classi,
- Attributi,
- Relazioni,
- Vincoli,
- Istanze.

Il Virtual Marketplace Engine

Il Virtual Marketplace Engine può essere visto come una sorta di sistema operativo per agenti. Esso, infatti, fornisce un ambiente nel quale questi ultimi possono coesistere, comunicare e, in ultima analisi, cooperare.

Lo sviluppo di un sistema multiagente, in Magenta, si articola in alcuni passi standard, che presentiamo di seguito:

1. *Definizione della Domain Ontology*. La *Domain Ontology* modella il dominio applicativo di interesse. Essendo generale ed indipendente dal particolare sistema multiagente che si intende sviluppare, essa può essere riutilizzata in varie applicazioni;
2. *Definizione della Virtual Market Ontology*. La *Virtual Market Ontology,* a differenza della Domain Ontology, è strettamente legata all'applicazione specifica. In particolare, la costruzione della *Virtual Market Ontology* consiste di alcuni passi:
    a. *Definizione dei ruoli attribuiti agli agenti*. In questo contesto, un ruolo rappresenta un insieme di attività eseguibili dall'agente che lo ricopre. Esso specifica, tra l'altro, come l'agente interagisce con gli altri agenti, per mezzo di tecniche di negoziazione e di cooperazione.
    b. *Definizione dei tipi di agente*. Un tipo di agente, in Magenta, rappresenta semplicemente un insieme di ruoli.
    c. *Definizione dei tipi di messaggio utilizzati dagli agenti per comunicare*.
    d. *Definizione dei punti di decision making*. Un punto di decision making rappresenta l'unità elementare di ragionamento dell'agente.

E' successivamente generato il codice Java corrispondente, che può essere eseguito con l'ausilio, tra gli altri, dell'Ontology Management Toolkit e dei framework di supporto all'esecuzione degli agenti. E' interessante notare, infine, che sono forniti degli strumenti per il *logging* e il *tracing* delle attività, al fine di supportare lo sviluppatore nelle fasi di analisi e *debugging* del sistema.

TeleTruck

TeleTruck è un sistema di supporto al *dispatching* basato su agenti. Esso mette a disposizione strumenti di pianificazione dinamica e ottimizzazione dell'ordinamento dei trasporti. In particolare, esso consente sia di rigenerare completamente un plan, sia di reagire agli eventi modificando in maniera opportuna il plan in uso. Quest'ultima caratteristica è di fondamentale importanza, in quanto, data l'intrinseca dinamicità e incertezza che caratterizza il problema trattato, gli algoritmi OR (central operations research) si dimostrano inefficienti, essendo in essi necessario far ripartire "da zero" la computazione. Utilizzando sistemi multiagente, invece, è possibile migliorare a livello locale il piano e trattare inconsistenze.

Alla base dell'approccio sta la modellazione degli oggetti fisici (ad esempio, camion, rimorchi, container, etc.) per mezzo di agenti. Questi ultimi sono tra loro fortemente collaborativi, e possono essere visti dall'esterno come un unico agente, detto agente *olonico*. In generale, un *holon* è composto di sotto-agenti che lavorano insieme al fine raggiungere obiettivi comuni. Gli utenti o gli altri membri della società di agenti possono interagire con esso come se si trattasse di un unico agente. Questa caratteristica è di fondamentale importanza, in quanto consente di ottenere diversi livelli di astrazione. In un holon, tipicamente, è presente un agente speciale che ricopre il ruolo di testa dell'holon. Tale ruolo gli attribuisce compiti di coordinamento dell'allocazione delle risorse agli agenti facenti parte della stessa struttura e di controllo della comunicazione con il resto della società di agenti. Esso potrebbe, inoltre, essere dotato della capacità di fornire piani agli agenti facenti parte dell'holon. Questi ultimi, infatti, sono caratterizzati dall'avere a disposizione propri piani, obiettivi e strumenti di comunicazione, al fine di mettere a disposizione le loro risorse (in funzione del proprio ruolo) per il raggiungimento dei comuni obiettivi. In TeleTruck, i differenti tipi di agente modellati sono uniti per formare un'unica entità di trasporto.

In sintesi, la testa di un holon ha i seguenti compiti:

- Coordinamento della formazione dell'holon;
- Eventuale riconfigurazione dell'holon;
- Pianificazione delle attività;
- Coordinamento delle attività.

Architettura di Teletruck

La seguente figura mostra l'architettura di Teletruck.

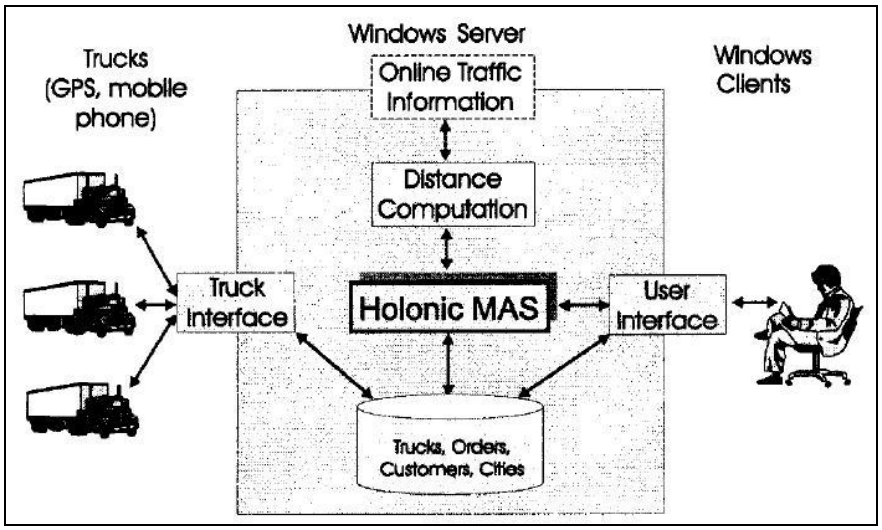

Figura 10: Architettura di TeleTruck

I camion delle compagnie di spedizione partner sono dotati di computer di bordo che, tra l'altro, sono corredati di software di comunicazione e di un GPS (*Global Positioning System*) che permette di individuare la posizione in cui si trovano. Le informazioni pervenute dai GPS ed eventuali messaggi inviati dagli autisti vengono memorizzati nel server centrale. Grazie a questi dati, il sistema e gli utenti di quest'ultimo possono monitorare l'esecuzione del plan e reagire dinamicamente in caso di problemi, dovuti ad esempio a ritardi causati dal traffico o a tempi di carico della merce inaspettatamente lunghi. Nel sistema è inoltre presente un modulo software che si occupa di fornire mappe digitali. Altro modulo disponibile è il *routing module,* il cui compito è quello di calcolare la distanza tra le località e fare stime sui tempi di percorrenza. Il *core* dell'architettura è rappresentato da un sistema multiagente *olonico*, per mezzo del quale vengono modellati i componenti del sistema di trasporto. Infine, è presente un modulo di interfaccia che consente di visualizzare la posizione dei mezzi di trasporto e di interagire con i funzionari di spedizione. Inoltre, tramite esso è possibile accedere a informazioni riguardo la configurazione delle entità di trasporto e dei piani, consentendo ai funzionari di spedizione di modificarli "a mano". Esso, oltre a quelli appena citati, consente l'accesso a una grande quantità di dati, come, ad esempio, alle informazioni pervenute dai GPS, ai messaggi degli autisti, ai dati relativi ai clienti e agli impiegati, etc..

Il sistema multiagente di Teletruck

In Teletruck, il sistema multiagente è costituito da agenti di tipo eterogeneo. Come già accennato, ogni agente ha a disposizione piani, obiettivi e strumenti di comunicazione, al fine di contribuire al raggiungimento di comuni obiettivi.

I *Plan'n' Execute Units* (PnEUs) sono agenti speciali, i cui compiti sono quelli di:

- coordinare la formazione degli holon rappresentanti le entità di trasporto;
- pianificare gli itinerari dei veicoli;
- pianificare il carico delle merci;
- pianificare i tempi di percorrenza.

Il PnEU ricopre inoltre il ruolo di rappresentante dell'holon di cui fa parte, interagendo con l'esterno. Inoltre, esso ha i permessi per riconfigurare l'holon stesso. Sebbene dotati di capacità di comunicazione, pianificazione e coordinamento, i PnEU non dispongono di risorse proprie. Nel sistema multiagente di Teletruck, è prevista la presenza di un PnEU per ogni *holon*, più un ulteriore PnEU "libero", da utilizzare per creare un eventuale nuovo *holon*.

Un altro tipo di agente è rappresentato dal *company agent*, i cui compiti principali sono:

- re-distribuire gli ordini,
- accettare o rifiutare le offerte,
- controllare l'ottimizzazione globale,
- coordinare l'esecuzione delle attività,
- gestire le comunicazioni tra il sistema e gli utenti.

In effetti, l'intero sistema può essere visto come un unico holon, la cui testa è rappresentata dal *company agent*.

Platform

Il progetto PLATFORM è una estensione di TeleTruck. Uno degli obiettivi di questo progetto è stato quello di realizzare una implementazione di un ambiente di simulazione, per valutare l'impatto dell'adozione di vari tipi di tecnologie e politiche di gestione volte al miglioramento dell'utilizzo di *terminal intermodali*. In questo contesto, un terminal intermodale può essere visto come un nodo nella rete che modella la connessione tra l'origine e la destinazione di una supply chain.

Architettura di Platform

L'architettura di PLATFORM è costituita da due sottosistemi:

- L'*ITP (Intermodal Trasport Planner)*, che gestisce la pianificazione sull'intera catena di trasporto;
- Il *sistema di simulazione*, che modella e simula i processi di trasporto, al fine di:
  - Valutare la fattibilità dei piani generati dall'IPT;
  - Valutare le prestazioni dei terminali intermodali.

A supporto dell'attività di pianificazione dei task di trasporto intermodali (ITT), L'ITP si serve dei seguenti moduli:

- I *Forwarding Agent*. Essi sono responsabili della pianificazione delle consegna degli *ITU* (*Intermodal Terminal Unit*) e del loro carico dai terminal. Ogni unità di spedizione è modellata per mezzo di questo tipo di agente. Un agente mediatore ha invece il compito di coordinare le attività di pianificazione dei forwarding agent in un certo intorno di ogni terminal.
- I *Booking Agent*. Tali agenti hanno il compito di verificare la disponibilità di posto sui mezzi programmati. Il *booking agent*, fatta tale verifica, sceglie il migliore possibile ed effettua la prenotazione.
- Gli *IPnEU (Intermodal Planning and Execution Units)*.

Il modulo di simulazione esegue la simulazione dell'esecuzione degli ITT, includendo in essa alcune operazioni interne al terminal, al fine di valutare la loro fattibilità e le *performance*. La simulazione può essere così suddivisa:

- *Road Simulation*, che simula il trasporto degli ITU per mezzo di camion. In particolare, esso simula il flusso dei camion in entrata e in uscita da ogni terminal.
- *Terminal Simulation*, che simula:
    - Il carico degli ITU sui camion e sui treni;
    - Lo scarico degli ITU dai camion e dai treni;
    - L'immagazzinamento degli ITU nel terminal intermodale.

Al fine di verificare la funzionalità delle procedure utilizzate, e di metterne in evidenza eventuali caratteristiche migliorabili, sono inoltre simulate:

- le attrezzature del terminal,
- le piattaforme di carico,
- le aree di deposito,
- le procedure di *gate*.

- *Train Simulation*, che simula il flusso di treni relativo alla tratta scelta.

Infine, è presente una interfaccia utente grafica che consente di personalizzare la simulazione, ottenere informazioni su importanti parametri, generare viste e mappe, e visualizzare gli input e gli output della simulazione.

Mascot

Nel presente paragrafo viene proposta una panoramica sull'architettura e sulle funzionalità di MASCOT. (*Multi-Agent Supply Chain cOordination Tool*). Esso è caratterizzato da una architettura altamente configurabile basata su agenti che supportano le attività di pianificazione e di scheduling di supply chain. Di

fondamentale importanza, in MASCOT, è la presenza di un agent wrapper. Le principali funzioni di quest'ultimo sono:

- Fornire un'interfaccia uniforme di comunicazione e di coordinamento tra i moduli di pianificazione e di scheduling;
- Fornire agli utenti funzionalità per modificare e visualizzare interattivamente le soluzioni di scheduling e di pianificazione proposte.

Architettura di Mascot

Le scelte architetturali di MASCOT sono state principalmente dettate dallo scopo di mettere a disposizione un framework per lo sviluppo e la modifica di pianificazioni e scheduling a vari livelli di astrazione. All'interno di tale architettura, gli agenti hanno il ruolo di wrapper a supporto dei moduli di pianificazione e di scheduling.

Una rappresentazione grafica dell'architettura di MASCOT è mostrata nella seguente figura:

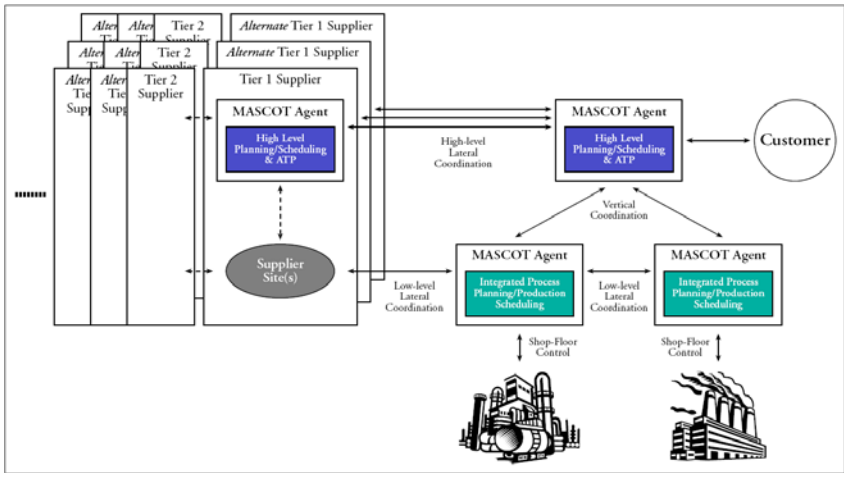

Figura 11: Architettura di MASCOT

Gli agenti, in MASCOT, supportano le seguenti funzionalità:

- *Coordinamento*: gli agenti hanno il ruolo di wrapper di coordinamento per i moduli di pianificazione e di scheduling distribuiti lungo la supply chain, supportando sia il protocollo di coordinamento orizzontale, sia quello verticale. In particolare, il protocollo di coordinamento orizzontale supporta le interazioni tra agenti posti allo stesso livello. Diversamente, il protocollo di coordinamento verticale supporta le interazioni tra gli agenti di alto livello con quelli di basso livello.
- *Integrazione tra moduli eterogenei di pianificazione e scheduling*: ogni agente di MASCOT utilizza una architettura a "lavagna", che consente di integrare facilmente svariati moduli di pianificazione e scheduling,

insieme a moduli di coordinamento e di analisi. All'interno di un dato agente, tali moduli possono essere attivati al fine di sviluppare o modificare le soluzioni integrate di scheduling e pianificazione. Queste ultime sono memorizzate in strutture dati accessibili da tutti i moduli dell'agente e dette, come accennato, "lavagne". I moduli specifici differiscono tra agente e agente, e questo è dovuto principalmente alle diverse funzionalità da questi ultimi messe a disposizione, e al livello all'interno dell'architettura in cui essi operano.

- *Funzionalità di supporto alle decisioni*: la lavagna di ogni agente di MASCOT è suddivisa in *contesti*, ognuno dei quali corrispondente a un insieme di assunzioni. Ogni assunzione può essere creata o manualmente dall'utente finale, oppure automaticamente da un agente. All'interno di ogni contesto, le soluzioni sono sviluppate o da un agente o dall'utente finale (o da entrambi), per mezzo dell'attivazione dei moduli di planning, di scheduling e di coordinamento presenti all'interno dell'agente.
- *Riconfigurabilità*: gli agenti di MASCOT possono essere rapidamente riconfigurati, in modo da inserire nuovi prodotti, fornitori, etc.

L'architettura di un generico agente di MASCOT è mostrata nella seguente figura:

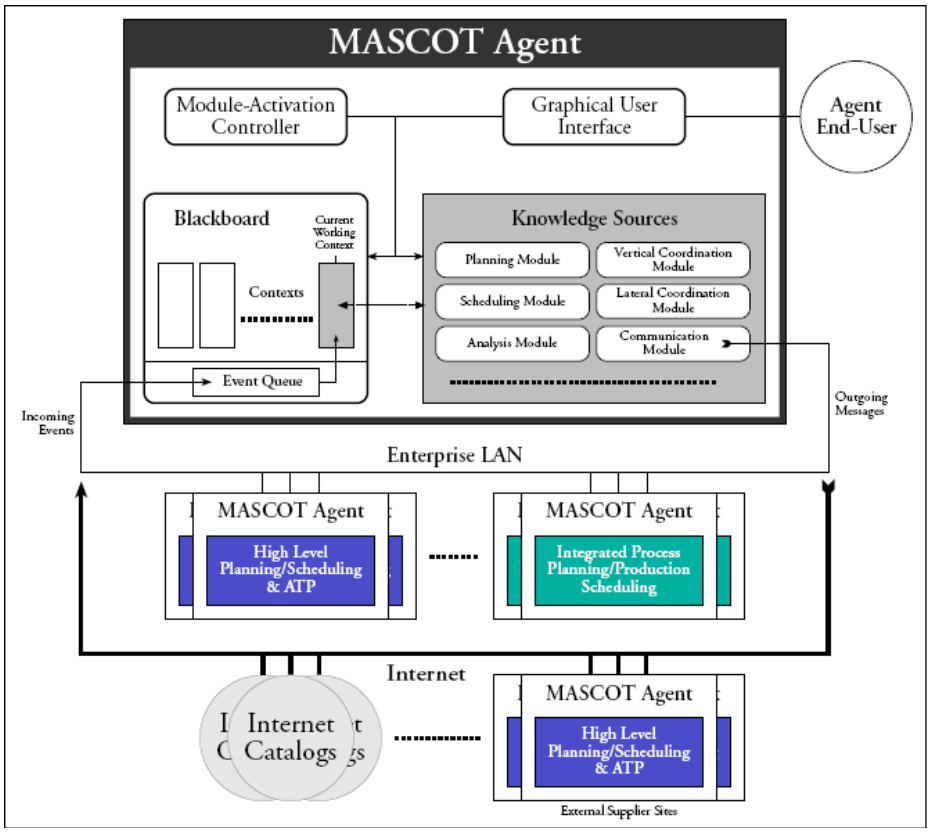

Figura 12: Architettura di un generico agente di MASCOT

Transport Network (LS/ATN)

Living Systems Adaptive Transportation Networks (LS/ATN) è un software sviluppato dalla Whitestein Technologies per offrire supporto alle compagnie logistiche.

Gli obiettivi che hanno guidato la progettazione di tale sistema sono sintetizzabili come segue:

- Ridurre significativamente i costi di trasporto;
- Migliorare la qualità dei servizi offerti al cliente.

In particolare, la riduzione dei costi è ottenuta principalmente nel seguente modo:

- Supporto alla spedizione automatica (in tale ambito viene curata, in particolare, la gestione degli eventi inattesi);

- Ottimizzazione dinamica dei trasporti in *real time;*
- Miglioramento della capacità di trasporto, mediante ottimizzazione delle allocazioni;
- Integrazione completa dei servizi telematici;
- Integrazione di network logistici di diverso tipo;
- Miglioramento della trasparenza e della visibilità all'interno del network.

In definitiva, l'obiettivo di LS/ATN è quello di supportare, ottimizzare e, in larga misura, automatizzare i processi di *dispatching* delle compagnie operanti nel settore logistico.

La complessità del problema dell'allocazione dei mezzi di trasporto merci è dovuta all'intrinseca dinamicità del trasporto:

1. Le richieste di trasporto non sono tipicamente conosciute in anticipo, in quanto nuove richieste possono arrivare in ogni momento;
2. I vecchi ordini possono cambiare in ogni momento;
3. La dimensione dell'ordine può essere correttamente valutata solo quando l'autista si reca presso i centri di raccolta.

Il problema di ottimizzazione consiste nel definire un piano di distribuzione, costituito da un insieme di itinerari, ad ognuno dei quali è associato una *schedulazione* di trasporto. In ogni schedulazione è specificato, per ogni locazione da raggiungere, l'orario di arrivo e l'orario di partenza del mezzo di trasporto. Altri parametri di ingresso del problema di trasporto sono relativi specificatamente al prodotto da trasportare (ad esempio, tipo di ordine, volume, peso, etc.) ed ai mezzi di trasporto disponibili (ad esempio, tipo di mezzo di trasporto, capacità, strumenti di carico e scarico disponibili, tariffe, etc.).

Ottimizzazione basata su agenti

La modellazione basata su agenti tiene in considerazione la dislocazione intrinsecamente sparsa dei trasporti. Gli agenti sono usati per rappresentare regioni geografiche, mentre i cargo sono modellati come informazioni (oggetti) che fluiscono tra gli agenti. Le operazioni di trasporto sono dislocate in diverse regioni di spedizione, queste ultime controllate ognuna da un agente, l'*Agent Region Manager*. Contemporaneamente, un altro agente, l'*Agent Distributor*, gestisce le richieste di trasporto.

L'ottimizzazione è articolata in due passi ed ha come obiettivo quello di minimizzare i costi di trasporto. In particolare:

1. Nel primo passo viene eseguita una ottimizzazione locale. Ogni nuova richiesta di trasporto viene ricevuta dall'*Agent Distributor* ed allocata

all'*Agent Region Manager* dell'opportuna regione geografica. Compito di quest'ultimo è ricercare una soluzione di trasporto ottima, limitatamente alla regione di competenza.

2. Nel secondo passo viene eseguita una ottimizzazione globale. Al fine di migliorare la soluzione globale corrente, la soluzione generata al passo 1 viene modificata per mezzo di una negoziazione bilaterale tra agenti di tipo *Agent Region Manager*. Nel corso di tale negoziazione, gli agenti richiedono l'un l'altro informazioni al fine di determinare, ad esempio, la disponibilità di mezzi di trasporto per un certo carico, oppure, la disponibilità di carico per un certo numero di mezzi di trasporto. La comunicazione è limitata agli agenti allocati rispettivamente alla regione geografica di partenza e a quella di arrivo, ed è progettata in modo tale da richiedere lo scambio di una minima quantità di informazioni, al fine di contenere la quantità di risorse utilizzate. Sono state sperimentate, con poco successo, altre soluzioni di negoziazione, come, ad esempio, quella di consentire la partecipazione alla stessa negoziazione ad un numero maggiore di due di agenti. E' stato verificato, però, che, a fronte di un modesto miglioramento della qualità della soluzione, era presente un aumento considerevole delle risorse utilizzate rispetto a quelle usate nella negoziazione bilaterale.

Anche se la funzione di ottimizzazione è essenzialmente basata sulla minimizzazione dei costi, altri obiettivi sono i seguenti:

- Ridurre il numero di mezzi di trasporto, utilizzandoli in maniera più efficiente;
- Ridurre la lunghezza dei percorsi.

Inoltre, la soluzione generata deve soddisfare alcuni vincoli. Alcuni di questi ultimi sono stretti (ad esempio, i limiti di capacità di carico e di peso dei mezzi di trasporto, orario di apertura dei clienti, ordinamento strettamente crescente della data di carico e della data di scarico della merce, etc.) ed altri deboli (che possono essere violati al costo di una certa penalità).

E' interessante notare che l'ottimizzazione descritta è dinamica, in quanto viene eseguita ogni qual volta venga ricevuta una nuova richiesta di trasporto. Il grande vantaggio di questo approccio è quello di rendere possibile un adattamento dinamico del sistema a possibili deviazioni dallo scheduling e a eventi inattesi. Inoltre, distribuire il processo di ottimizzazione, delegandolo agli agenti prima descritti, consente di rendere il sistema scalabile all'aumentare delle richieste di trasporto e più robusto. La negoziazione e la coordinazione tra agenti può ridurre il costo globale di trasporto e migliorare l'utilizzo delle risorse.

4.3 Architettura di un sistema multi agente per il porto di Gioia Tauro

4.3.1 Problematiche tipiche di un container terminal

I porti sono da sempre nodi di cruciale importanza nei traffici commerciali, ma, negli ultimi anni, grazie all'avvento di un'economia "globalizzata", essi stanno acquisendo un sempre maggior peso nei trasporti di merce. I *container terminal* sono strutture specifiche dei porti, nei quali le merci, all'interno di particolari contenitori detti container, vengono trasferite da un mezzo di trasporto a un altro. I container sono ormai uno standard nel trasporto via mare di beni. In un settore in rapida espansione come quello del trasporto merci via mare, è di vitale importanza una maggiore produttività, un più rapido scambio di informazioni e un più veloce turn around delle navi, e questo spinge inevitabilmente all'ideazione di nuove soluzioni per la gestione dei container terminal.

Tali esigenze sono rese ancora più pressanti da diversi fattori. Innanzitutto, il rapido incremento degli scambi commerciali porta come conseguenza un maggiore traffico delle merci. Da questo deriva un sempre maggiore livello di congestione dei porti, con conseguente maggior tempo di permanenza delle navi in essi, tariffe più alte, e, da questo, minore competitività. La ridotta competitività, d'altro canto, impone a sua volta il perseguimento di un abbassamento dei costi.

E' interessante fornire alcuni dati per apprezzare i cambiamenti che stanno occorrendo nella gestione dei porti. Il trasporto di beni per mezzo di navi atte al trasporto dei container sta crescendo annualmente del dieci per cento. Il numero di container trasportati è passato da 39 milioni nel 1984 a oltre 356 milioni nel 2004. Inoltre, la dimensione delle navi atte al trasporto dei container nel tempo è cresciuta enormemente, fino al varo di navi capaci di trasportare oltre 14000 container. I costi operativi di navi di tali dimensioni superano i 65000 dollari al giorno.

L'enorme crescita del traffico di merci ha reso inevitabile che molti porti raggiungessero la loro capacità massima, e in alcuni casi a superarla.

Tale congestione porta in particolare diversi inconvenienti, tra i quali:

- *Per le Shipping Lines*: ritardo delle navi, costi aggiuntivi, perdita di clienti, etc.
- *Per i container terminal*: necessità di manodopera aggiuntiva, congestione degli yard, riposizionamento frequente dei container, etc.
- *Per le ditte di trasporto su strada e per le ferrovie*: lunghi tempi di attesa, perdita di possibili business.

Una delle soluzioni adottabili per rispondere alle problematiche di congestione che affliggono i porti è quella di aumentare la capacità degli stessi. Ciò può essere perseguito estendendo le dimensioni fisiche oppure migliorando lo sfruttamento delle risorse disponibili. Purtroppo la prima opzione non sempre è adottabile in quanto, specialmente per i porti dislocati in Europa, non esiste lo spazio fisico per garantire

un'espansione adeguata, oppure non sono disponibili fondi sufficienti per costruire nuove infrastrutture.

La strada più praticabile e ragionevole è quella di migliorare lo sfruttamento delle risorse disponibili attraverso l'adozione di tecnologie informatiche.

4.3.2 Approccio ad agenti

Come già messo in evidenza, a causa dell'incremento del traffico di container, molti terminal sono vicini al raggiungimento della loro capacità massima. Inoltre, col passare del tempo vengono costruite navi sempre più grandi. Tutto questo rende sempre più pressante la necessità di avere strumenti per identificare soluzioni per effettuare le operazioni tipiche di un container terminal in maniera efficiente. In particolare, sarebbe molto utile avere a disposizione soluzioni per svolgere nella maniera più rapida possibile le operazioni di imbarco e sbarco dei container. Il collo di bottiglia nelle operazioni di carico e scarico dei container è in genere rappresentato dalle attività delle gru di banchina. Infatti, accade spesso che durante le operazioni di scarico della merce esse abbiano il buffer pieno; analogamente, succede frequentemente che i buffer siano vuoti durante le operazioni di carico. Tali situazioni portano inevitabilmente ad impiegare tempi più lunghi per completare le operazioni richieste, e, di conseguenza, ad un aumento dei costi.

In questo paragrafo sarà specificato il comportamento degli agenti descritti precedentemente. In particolare, sarà discusso il comportamento nei seguenti casi [P08]:

- In presenza di eventi inattesi; tali eventi, nel problema affrontato, riguardano il funzionamento dei mezzi di movimentazione. In tal caso, l'agente che non può completare il task assegnato tenta di riassegnarlo per mezzo di una negoziazione che fa uso dello strumento delle aste, in modo da limitare solo a livello locale l'attività di ripianificazione, altrimenti molto dispendiosa dal punto di vista del tempo di computazione;

- In assenza di eventi inattesi; ovvero, sarà proposto il ciclo standard di funzionamento del sistema multiagente; in particolare, sarà affrontato il problema del routing degli straddle carrier.

4.3.3 Modellazione delle entità coinvolte per mezzo di agenti

Nel presente paragrafo sarà proposta una modellazione delle entità portuali per mezzo di agenti. Non verrà proposta una modellazione per ciascuna di esse, ma solo per quelle che saranno coinvolte nei processi nell'ambito dei quali sarà proposta una soluzione facente utilizzo delle tecnologie ad agenti. In particolare, saranno presenti nel sistema i seguenti agenti:

- *Ship Agent*. Ogni nave che entra nel porto sarà modellata per mezzo di uno *Ship Agent,* che sarà generato dal sistema. Esso comunicherà con gli altri

agenti, ad esempio, inviando opportune richieste di imbarco e sbarco di container.

- *Crane Agent*. Ogni gru addetta al carico e allo scarico della merce da una nave sarà modellata per mezzo di un *Crane Agent*. Esso interagisce con lo *Ship agent* e con gli altri *Crane Agent*, contrattando con questi ultimi il numero di straddle carrier assegnati.
- *Yard Agent*. Lo *Yard Agent* è un agente introdotto per eseguire attività di gestione dello yard. Al fine di fornire opportuni input agli algoritmi di routing degli straddle carrier che saranno di seguito presentati, esso definisce il punto iniziale dal quale un container viene movimentato, il punto finale di tale movimentazione, e li comunica al *Routing Agent*. L'agente si serve di librerie già disponibili che calcolano l'allocazione dei container.
- *Routing Agent*. Il *Routing Agent* gestisce, per mezzo di aste, il routing degli Straddle Carrier. Esso riceve messaggi dallo *Yard Agent* e dai *Crane Agent*, ad esempio, relativi agli Straddle Carrier disponibili per ogni gru. Riceve e invia messaggi agli *Straddle Carrier Agent*, relativi in particolare ad aste per assegnare i job, ad eventuali riallocazioni locali dei job ed impreviste indisponibilità dei mezzi di movimentazione.
- *Straddle Carrier Agent*. Gli Straddle Carrier sono modellati per mezzo di *Straddle Carrier Agent*. Gli *Straddle Carrier Agent* comunicano per mezzo di messaggi con il *Routing Agent* e gli altri *Straddle Carrier Agent*. Essi partecipano alle aste bandite dal *Routing Agent* per l'assegnamento dei job. Inoltre, possono essi stessi bandire aste per la riallocazione dei job (al livello locale) in caso di eventi inattesi, senza l'intervento del *Routing Agent*, al quale saranno semplicemente comunicate le modifiche effettuate.

4.3.4 Gestione del routing dei mezzi per la movimentazione in piazzale

I modelli attuali di routing dei mezzi di trasporto in piazzale si basano su assegnazioni dei mezzi a precise code di lavoro e, nell'ambito della stessa coda di lavoro, il mezzo esegue lo spostamento del contenitore, facendo molta strada a vuoto. Considerando che in qualsiasi istante esistono movimentazioni in tutte le direzioni all'interno del terminal, è pensabile di combinare i diversi movimenti in modo tale da ridurre i movimenti a vuoto.

La sequenza di moviementazioni (code di lavoro) assegnate a ciascun mezzo rappresenta un piano da eseguire.

Nell'ambito della movimentazione interna del piazzale, si tratta di movimentare contenitori entro un certo tempo massimo, con un dato numero di mezzi (il numero di mezzi deve chiaramente essere tale da garantire l'espletamento del lavoro entro i limiti richiesti). Per gestire eventi inattesi, si richiede che vengano affrontati problemi di "real-time logistics", ovvero, che vengano definiti algoritmi e modelli che consentano di pervenire a decisioni in tempo reale. Il problema è quello di poter ripianificare le operazioni al variare delle condizioni al contorno che hanno determinato una

precedente pianificazione. Un esempio è il sopraggiungere di un evento che provoca il fermo di una gru per un certo tempo. Alla ripresa delle attività della gru la movimentazione ottimale potrebbe essere diversa da quella pianificata in precedenza.

Le problematiche specifiche affrontate sono presentate nei successivi paragrafi.

4.3.5 Routing degli Straddle Carrier in assenza di eventi inattesi

4.3.5.1 Il problema

In sintesi, lo scenario è il seguente: una nave che trasporta container entra in un porto e viene assegnata a una banchina equipaggiata con speciali gru per il carico/scarico di container. Ad ogni pool di gru che lavora per la nave vengono assegnati un insieme di Straddle Carrier. Infatti, i movimenti dei container da e per la banchina sono realizzati per mezzo di mezzi mobili, nel caso modellato appunto per mezzo di Straddle Carrier. Ogni gru utilizza un buffer, un'area di dimensione limitata (si supponga che un buffer possa contenere al massimo cinque container) nel quale scarica e dal quale carica i container. La maggior parte dei container scaricati viene trasferita in un'apposita area di storage nelle vicinanze del punto dove verranno imbarcati successivamente. Solo una piccola parte di essi viene caricata nuovamente senza alcun intervento di movimentazione intermedia.

L'obiettivo di questo paragrafo è quello di proporre una soluzione al problema del routing degli Straddle Carrier, per mezzo assegnamento di sequenze di movimentazioni (dei piani) a tali mezzi. Il problema può essere formulato come segue: dato un insieme di straddle carrier S e un insieme di job J, dove per job in questo contesto si intende il trasporto di un container da una locazione a un'altra, assegnare ad ogni straddle carrier s appartenente a S, una sequenza (eventualmente vuota) di job j appartenenti a J (un piano). In tale contesto, ogni job è caratterizzato dalla posizione in cui si trova il container ad esso associato, da quella in cui esso deve essere portato, dalla *release date*, ovvero, dall'istante di tempo a partire dal quale può iniziare la movimentazione, e dalla *due date*, che rappresenta l'istante di tempo entro il quale dovrebbe essere terminato il job. Uno Straddle Carrier, a sua volta, è caratterizzato dalla sua velocità, da un job corrente (il job che sta eseguendo nell'istante in cui viene preso in considerazione, eventualmente nullo), e dal tempo necessario per il completamento di quest'ultimo.

Al sistema sono forniti in input i dati relativi alle navi e ai container da scaricare o da caricare. Si assume pertanto che esternamente al sistema multiagente proposto vengano gestiti l'ingresso delle navi nell'area portuale e la loro precisa locazione di attracco. Sono inoltre precedentemente assegnate alle navi le gru necessarie, e a queste ultime i mezzi di movimentazione. Il software ad agenti si interfaccia con un simulatore, che gli fornisce in input, al variare del tempo, un certo numero di job, le due date dei quali sono più prossime rispetto a quelle dei job non ancora presi in considerazione.

4.3.5.2 La soluzione proposta

Come già accennato, il problema che si intende risolvere è quello di assegnare *n* job a *m* straddle carrier, tipicamente con $n > m$.

La soluzione del problema per mezzo degli agenti del routing dei veicoli si articola in due passi:

- Definizione di una soluzione iniziale
- Raffinamento in cicli successivi della soluzione iniziale.

Notazioni:

- $j_1 \ldots j_n$: job;
- *setup*: tempo necessario per prendere il job in carico, spostandosi dalla posizione relativa alla fine del job precedente sino alla posizione iniziale del job attuale;
- *due date*: tempo entro il quale il job deve essere completato;
- *release date*: tempo a partire dal quale il job può essere cominciato;
- $A_1 \ldots A_m$: agenti associati agli straddle carrier;
- J_{Ai}: insieme di job assegnati all'agente A_i;
- S_{Ai}: sequenza di job assegnati all'agente A_i (*job list*) – gli elementi di *S* sono piani;
- $cost(A_i, j_h)$: costo dell'esecuzione del job j_h da parte dell'agente A_i;
- $cost(A_i, S_{Ai})$: costo dell'esecuzione della sequenza di job S_{Ai} da parte dell'agente A_i.

Funzione di costo *cost*:

La funzione di costo *cost(Agent, Job)* definisce il costo dell'esecuzione del job *Job* da parte dell'agente *Agent*. In particolare, *cost(Agent, Job) = f(setup, ritardo)*. Il tempo di *setup*, in particolare, risulta essere la somma del tempo di completamento del job j_h appena antecedente al job *Job*, più il tempo necessario allo Straddle Carrier rappresentato dall'agente *Agent* per portarsi dalla posizione finale del job j_h alla posizione iniziale del job *Job*. Ovviamente, maggiore sarà il tempo di *setup*, maggiore sarà il costo che l'agente dovrà sostenere; sarà perciò preferibile assegnare il job, a parità di valore degli altri parametri della funzione *cost*, ad un agente con tempo di *setup* minore.

Il costo attribuito sarà tanto più alto quanto maggiore è il tempo al quale è prevista la conclusione del job da parte dell'agente e sarà funzione (non lineare) nelle variabili tempo di *setup* e *due date* (dalle quali dipende la variabile *ritardo*) e dipenderà dalla distanza che l'agente dovrà percorrere a vuoto (senza trasportare alcun container) per iniziare il job *Job*.

Inoltre, il costo avrà una componente costante, fino al punto in cui la *due date* sarà maggiore o uguale al tempo di completamento del job (dove il tempo di completamento si intende uguale al tempo di setup, più il tempo necessario a svolgere il job). Per tempi di completamento maggiori della *due date* il costo diverrà molto più alto (esiste un punto di discontinuità per tempo di completamento uguale alla *due date*).

Definizione di una soluzione iniziale:

In questo contesto si utilizzerà il formalismo delle aste. In particolare, sarà presente un *routing agent*, che si occuperà di assegnare per mezzo di aste i job agli agenti che modellano gli straddle carrier. Le aste saranno del tipo *one shot*. Siano n e m rispettivamente il numero di job e il numero di straddle carrier disponibili. L'algoritmo di definizione di una soluzione iniziale è costituito da n/m iterazioni, nel caso in cui n MOD $m = 0$, altrimenti da $n/m + 1$ iterazioni. Ad ogni iterazione, saranno messi all'asta i primi m job, tra gli l ancora disponibili. Gli m job scelti durante la prima iterazione saranno caratterizzati dall'avere *due date* con scadenza più prossima rispetto a quelle relative agli $n - m$ job non selezionati. Saranno bandite m differenti aste, una per ogni job disponibile nella corrente iterazione. Tutti gli agenti parteciperanno a tutte le aste. In particolare, ognuno di essi presenterà un'offerta pari al costo che esso dovrà sostenere per portare a termine il job oggetto dell'asta, come specificato nella definizione della funzione *cost*. Nel caso si verifichi che un agente risulti vincitore di più di un'asta, esso sarà estromesso dalla graduatoria per lui meno "conveniente". Ovvero, sia I_{pAi} l'insieme dei job in cui l'agente A_i è risultato vincitore nel corso dell'iterazione p. Il job effettivamente assegnato all'agente A_i apparterrà all'insieme I_h, che è definito come segue: $I_h = \{j_h \mid \forall\ j_x \in I_{pAi}\ cost(A_i, S_{Ai\_p-1} \cup j_h) \leq cost(A_i, S_{Ai\_p-1} \cup j_x)\ \}$, dove S_{Ai_p-1} è la sequenza di job già assegnati all'agente A_i. In altre parole, non esiste alcun job, tra quelli che l'agente si è aggiudicato nel corso delle aste della corrente iterazione, che abbia costo minore di quello associato agli elementi dell'insieme I_h, e perciò risulta per l'agente conveniente scegliere uno tra i job appartenenti a I_h (l'insieme I_h è costituito per definizione dai job con costo minore, e pertanto tutti gli elementi che appartengono ad esso hanno uguale costo). Dal momento che un agente può aggiudicarsi un solo job per ogni iterazione, all'agente A_i sarà assegnato uno solo tra gli elementi appartenente all'insieme I_h. Tale selezione non avverrà secondo criteri di ottimizzazione (ad esempio sarà scelto il primo, nel caso l'insieme sia implementato come un vettore), anche se scegliere in particolare uno tra i job appartenenti a I_h potrebbe portare a ridurre la somma totale dei costi che gli agenti dovrebbero sostenere per eseguire le sequenze di job provvisoriamente ad essi assegnate. Questo risulterebbe però computazionalmente oneroso. Una redistribuzione dei job avverrà in una successiva fase dell'algoritmo. E' da notare infine che il costo non è esclusivamente funzione del job j_h e dell'agente A_i, ma anche dalla sequenza di job precedentemente assegnati all'agente.

Nel caso in cui l'agente vincitore dell'asta b_h (in cui viene bandito il job j_h) sia estromesso dalla stessa in quanto vincitore di un'asta con costo per esso minore, si aggiudicherà l'asta b_k l'agente che ha presentato la seconda migliore offerta. Se anche esso dovesse essere estromesso da tale asta, risulterà vincitore l'agente autore della terza migliore offerta, e così via. In questo modo si ottiene che gli m job presi in considerazione nel corso dell'iterazione *p-esima* saranno assegnati a m differenti

agenti. Nella successiva iterazione saranno presi in considerazione, tra gli $n - p*m$ job rimasti a disposizione, gli m job caratterizzati da una due date più prossima rispetto a quelle relative ai $n - (p+1)m$ job non selezionati. Ancora una volta il *routing agent* bandirà m aste. Il meccanismo è similare per ogni iterazione, a quello definito per la prima iterazione. Sarà necessario solamente definire opportunamente il costo che ciascun agente dovrebbe sostenere per portare a termine il job j_k oggetto dell'asta, in quanto tale costo non sarà esclusivamente funzione del job j_k e della sua posizione, ma anche dalla sequenza di job precedentemente assegnati all'agente A_i.

Raffinamento in cicli successivi della soluzione iniziale:

Una volta ottenuto un assegnamento iniziale degli n job disponibili, e quindi a ogni agente risulterà assegnato un piano, ovvero una sequenza di moviementazioni, è possibile raffinare le soluzione ottenuta. Si utilizza nuovamente il protocollo del contract net, ma questa volta saranno gli agenti che rappresentano gli straddle carrier a bandire le aste. Viene introdotto il concetto della *stop list*, un insieme di job che non possono più essere messi all'asta. L'algoritmo consiste dei seguenti passi:

1 – Riorganizzazione all'interno della *job list* di ciascun agente, in modo da minimizzare i costi.

2 – Tra gli agenti che possono bandire l'asta (si veda passo 6), ovvero tra gli agenti nella cui job list è presente almeno un job non incluso nella stop list, sia A_i l'agente con maggior costo totale $cost(A_i, S_{Ai})$. A_i mette all'asta il job j_m tale che la sequenza S_{Ai} / j_m sia quella con costo minimo, tra tutte le sequenze S_{Ai} / j_x, per ogni j_x appartenente a S_{Ai}. Se non esiste alcun agente la cui *job list* contiene almeno un job non incluso nella *stop list*, allora l'algoritmo termina, in quanto è stato raggiunto un punto di equilibrio.

3 – ogni agente A_j propone una *riconfigurazione* delle sequenze S_{Ai} e S_{Aj}, dove S_{Ai} è la sequenza di job assegnati all'agente A_i, S_{Aj} è la sequenza di job assegnati all'agente j. Tale *riconfigurazione* consiste nell'inserimento del job j_m in una qualche posizione della sequenza S_{Aj}. In particolare, ogni agente A_j calcola il costo di ogni sequenza S_{Aj}' ottenute ciascuna inserendo il job j_x in una posizione diversa nella propria sequenza di job. Sia k il numero di elementi della sequenza S_{Aj}. Da questa sequenza saranno generate $k + 1$ sequenze S_{Aj}'. L'agente proporrà la sequenza S_{Aj}' con costo minimo.

4 – vince l'asta l'agente che propone una *riconfigurazione* tale che essa generi il maggiore decremento della somma $cost(A_i, S_{Ai}) + cost(A_j, S_{Ai})$. E' da notare che gli agenti che rispondono con un'offerta all'asta sono al più $m-1$ (anche gli agenti i cui job sono tutti inclusi nella stop list possono partecipare in qualità di *bidder*). Saranno perciò eseguiti $m - 1$ algoritmi paralleli che calcolano le proposte di riconfigurazione.

5 – successivamente l'agente A_i e l'agente A_j riorganizzano le rispettive *job list*.

6 – Nel caso in cui non vi sia stato alcuno scambio, inserisci il job j_m nella *stop list*. Torna al passo 2.

L'inserimento di un timeout consente di controllare il tempo di esecuzione dell'algoritmo.

4.3.6 Riallocazione dei job in presenza di eventi inattesi

4.3.6.1 Il problema

Ad un certo istante t_i, lo scenario è il seguente: ad ogni *Straddle Carrier Agent* è assegnato un job o una sequenza di job (nel caso non sia assegnato nessun job, si assume che sia assegnato un job vuoto).

Il *Routing Agent* nel passo precedente ha di fatto assegnato ad ognuno degli *n* Straddle Carrier un piano da eseguire (nel caso venga assegnato un solo compito a ogni straddle carrier, il piano sarà costituito da una sola macro azione). E' interessante notare che gli *Straddle Carrier Agent* hanno a disposizione esclusivamente i piani che gli sono stati assegnati, ma non ha conoscenze sul piano globale, che in questo caso è la composizione di tutti i singoli piani. Si supponga che uno straddle carrier debba raggiungere un certo punto di arrivo *A* previsto per il deposito del container. Al verificarsi di un evento inatteso, ad esempio causato dal malfunzionamento di un motore, tale Straddle Carrier potrebbe risultare non più in grado di raggiungere *A*. Questa situazione impone una rielaborazione dinamica del piano di movimentazione dei container che, tuttavia, se affrontato con gli strumenti già presenti, potrebbe richiedere un tempo di esecuzione più lungo di quanto potrebbe essere sufficiente con una politica di rimodulazione localizzata, demandata agli agenti associati ad alcuni degli Straddle Carrier coinvolti nelle operazioni.

4.3.6.2 La soluzione proposta

Per presentare la soluzione proposta, si fa riferimento ad un particolare caso di evento inatteso, ovvero l'avaria di uno Straddle Carrier. La soluzione proposta è però generale, e può essere utilizzata in diversi contesti.

Si supponga che alcuni job assegnati agli Straddle Carrier siano più importanti di altri, secondo una certa funzione. In tal caso, ogni agente associato a uno Straddle Carrier ha a disposizione un certo budget, equivalente all'importanza del task effettuato.

Si utilizza nuovamente il meccanismo delle aste. Infatti, l'agente associato allo Straddle Carrier in avaria, manda un annuncio con il prezzo che è disposto a pagare (il valore della sua missione). A questo punto, i potenziali sostituti valutano l'offerta in funzione della potenziale penalità per il ritardo dell'esecuzione del loro piano. Si verifica quindi un'asta "al ribasso", in cui l'agente che accetta deve avere un margine di guadagno. Ad esempio, l'agente *x* accetta per l'intero costo proposto dall'agente che ha lanciato l'asta, ma è superato, con una proposta di accettazione con costo inferiore, dall'agente *y*. La differenza tra l'offerta *i* e l'offerta *i+1* deve essere maggiore o uguale a un certo minimo, altrimenti l'asta potrebbe durare troppo a lungo. Le aste potrebbero propagarsi, ovvero, l'agente che si aggiudica l'asta potrebbe a sua volta indire una nuova asta. Tuttavia, come viene chiarito di seguito, il livello di propagazione, grazie ad alcuni accorgimenti ed alla tipologia di asta scelta, è basso. Ciò porta come conseguenza che la rimodulazione dei piani si verifica solo localmente. Quanto detto dipende dal fatto che:

- L'asta è al ribasso, quindi c'è un numero massimo di riassegnamento dei job e di propagazione dei piani, in quanto il limite inferiore oltre il quale si verifica l'accettazione del riassegnamento dei job è dato dal costo della missione meno costosa più uno.
- Si introducono inoltre dei costi aggiuntivi, legati alla distanza tra lo Straddle Carrier che propone l'asta e lo Straddle Carrier potenziale sostituto, in modo che venga preso in considerazione il fatto che la missione potrebbe continuare in ritardo, e questo favorisce gli Straddle Carrier in zona, evitando così un'eccessiva propagazione dei riassegnamenti.

Il *Routing Agent*, che ha una visione "globale" del problema e dei piani, deve solo validare il nuovo piano globale, quindi non deve generare nuovi piani a livello globale, ma semplicemente valutare la coerenza del nuovo piano con i vincoli imposti.

4.3.7 Architettura

Il sistema dovrà essere basato un'architettura ad agenti distribuita, in modo da sfruttare, tra l'altro, i vantaggi portati dal parallelismo introdotto. Ogni agente sarà eseguito su una macchina host. Ci si propone di utilizzare JADE, un middleware conforme alle specifiche FIPA. Agli scopi di questa sezione, si ritiene possa essere sufficiente ricordare che JADE è un middleware open source per lo sviluppo di architetture multi-agente distribuite, basato sul paradigma del peer-to-peer. JADE è basato sul linguaggio java, e mette a disposizione librerie per la comunicazione e l'interazione tra agenti. Inoltre, esso fornisce strumenti per la gestione del ciclo di vita degli agenti, per l'ispezione dei messaggi scambiati e per il debugging. Il simulatore sarà realizzato in java. Ogni agente sarà costituito da:

- Un modulo di comunicazione (*communication layer*). Compito di questo modulo è quello di implementare l'interazione dell'agente con gli altri agenti tramite lo scambio di messaggi. In particolare, il modulo di comunicazione dovrà sovraintendere alla costruzione dei messaggi, al loro invio, alla loro ricezione e interpretazione. I processi di comunicazione tra agenti dovranno essere vincolati da opportuni time-bound, al fine di rispettare la natura real-time del sistema. I messaggi saranno utilizzati prevalentemente per partecipare alle aste o per trattare lo scambio di job nel caso si verificasse l'occorrenza di eventi inattesi;
- Un modulo di valutazione delle proposte di partecipazione all'asta (*evaluation layer*). Si ricorda, a tal fine, che anche ogni agente che modella gli Straddle Carrier può, a sua volta, bandire aste, nel caso del verificarsi di un evento inatteso, e quindi tale modulo non è esclusivamente presente nell'architettura del *Routing Agent*;
- Un modulo di calcolo (*processing layer*), presente su ogni agente che rappresenta Straddle Carrier, che calcolerà i costi relativi all'introduzione di nuovi job nella sequenza di job assegnata all'agente. Tale modulo non sarà presente nel *Routing Agent*;

- Un modulo di controllo (*control layer*), che sovraintenderà al funzionamento degli altri moduli, attivandoli e disattivandoli in maniera opportuna.

Di seguito si propone una rappresentazione grafica degli agenti e delle loro interazioni nel caso del routing degli Straddle Carrier in assenza di eventi inattesi.

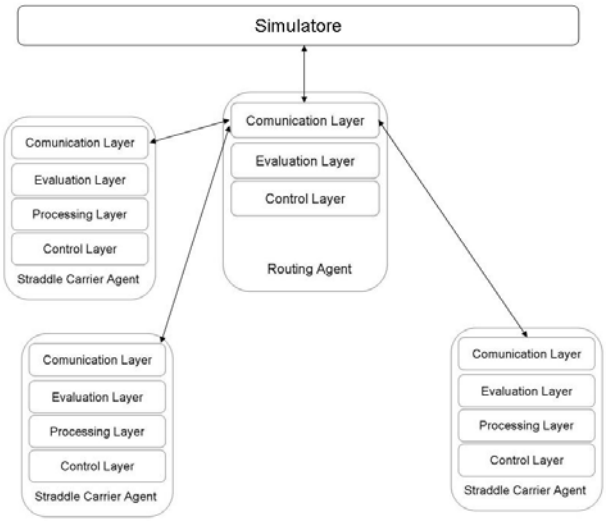

Figura 13: Routing degli Straddle Carrier

4.3.8 Conclusioni

Negli ultimi anni il traffico merce nei porti sta aumentando in maniera vertiginosa. Tale crescita ha portato a una forte congestione dei porti, riducendone la produttività e aumentando i costi, rendendoli così meno competitivi. Una possibile risposta a tale congestione è rappresentata dall'ampliamento dell'area dei porti e dall'inserimento di nuove infrastrutture. Purtroppo, però, tale soluzione è estremamente costosa e, sovente, inapplicabile, a causa della mancanza di spazi utili a consentire tale espansione. Una strada percorribile è rappresentata, viceversa, dall'introduzione di tecnologie informatiche al fine di migliorare lo sfruttamento delle risorse disponibili.

La tecnologia ad agenti, in particolare, sembra adatta ad affrontare alcune problematiche tipiche del dominio applicativo dei container terminal. Gli agenti, infatti, possono essere progettati in modo che siano: *i)* autonomi; *ii)* adattativi; *iii)* orientati agli obiettivi; *iv)* mobili; *v)* reattivi; *vi)* situati; *vii)* sociali. Tali caratteristiche

rendono particolarmente utile l'applicazione di tecnologie ad agenti nel dominio applicativo dei container terminal, in quanto essi sono tipicamente modulari, decentralizzati, mutevoli e complessi.

Nella presente sezione è stata presentata un'architettura ad agenti utile per affrontare alcune delle problematiche tipiche di un container terminal. In particolare, l'architettura proposta può essere utilizzata per risolvere i seguenti problemi: routing degli straddle carrier; riallocazione rapida e locale di job agli straddle carrier nel caso in cui uno più di essi sia impossibilitato ad eseguire (o a completare l'esecuzione) di uno o più job, a causa di un evento inatteso, e, di conseguenza, ridefinizione delle sequenza di job da eseguire (piani).

Riferimenti bibliografi

Riferimenti bibliografici

[AN06] Augusto, J. C. e Nugent, C. D.: Designing Smart Homes: The Role of Artificial Intelligence. Lecture Notes in Artificial Intelligence, vol. 4008. Springer, Berlin, 2006.

[AT04] Achour, N. e Toumi, R. e Msirdi, N.K.: Building Environment Maps using Neural Networks. International Journal of Robotics and Automation, 2004.

[BG06] Broxvall, M. e Gritti, M. e Saffiotti, A. e Seo, B. S. e Cho, Y. J.: PEIS Ecology: Integrating Robots into Smart Environments, In: Proc. of the 2006 IEEE International Conference on Robotics and Automation, Orlando, Florida, May 2006.

[B91] Brooks, R. A.: Intelligence without representation. Artificial Intelligence, 47:139-159, 1991.

[BF00] Bürckert, H. J. e Fischer, K. e Vierke G.: Holonic Transport Scheduling with Teletruck. Applied Artificial Intelligence 14(7): 697-725 (2000)

[CD04] Cook, D.J. e Das, S.: Smart Environments: Technology, Protocols and Applications, Wiley-Interscience, 2004.

[CL04] Coddington, A. e Luck, M.: A Motivation-based Planning and ExecutionFramework, In International Journal on Artificial Intelligence Tools, vol. 13(1), pp. 5-25, 2004.

[CY03] Cook, D. J. e Youngblood, M. e Heierman, E. e Gopalratnam, K. e Rao, S. e Litvin, A. e Khawaja, F.: MavHome: An Agent-Based Smart Home, In: Proc. of the Conference on Pervasive Computing, 2003.

[CY06] Cook, D. J. e Youngblood, M. e Das, S. K.:A Multi-agent Approach to Controlling a Smart Environment". Designing Smart Homes 165-182, 2006.

[FS03] Fischer, K. e Schillo, M. e Siekmann, J. H.: Holonic Multiagent Systems: A Foundation for the Organisation of Multiagent Systems. HoloMAS 2003: 71-80

[G71] Good, I. J.: Twenty-seven principles of rationality. In V. P. Godambe and D. A. Sprott, editors, *Foundations of Statistical Inference*, pages 108–141. Holt, Rinehart, Winston, Toronto, 1971.

[GD04] Ghallab, M. e Nau, D. e Traverso, P.: Automated Planning: Theory & Practice, Morgan Kaufmann, 2004.

[GGP08] Garro, A. e Greco, S. e Palopoli, F.: Smart Agents and Smart Environments: a Predictive Approach to Replanning. In: Proc. of "Intelligent Agents and Services for

Smart Environments" (IASSE) as part of the Artificial Intelligence and Simulation of Behaviour (AISB) Convention", Aberdeen, SCOTLAND, 1-4 April, 2008.

[GN04] Ghallab, M. e Nau, D. e Traverso, P.: Automated Planning: Theory & Practice, Morgan Kaufmann, 2004.

[GP98] Georgeff, M. P. e Pell, B. e Pollack, M. E. e Tambe, M. e Wooldridge, M. J.: The Belief-Desire-Intention Model of Agency. 5th Int. Ws. on Intelligent Agents, Agent Theories, Architectures, and Languages 1-10, 1998.

[GR02] Gambardella, L. M. e Rizzoli, E. e Funk, P.: Agent-based Planning and Simulation of Combined Rail/Road Transport. (www.idsia.ch/~luca/simulation02.pdf).

[JW95] Jennings, N. e Wooldridge, M.: Agent Theories, Architectures, and Languages: A Survey", Intelligent Agents: ECAI-94 Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, Berlin, 1995.

[HD06] Hu, X. H. e Dang, J.W.: Hybrid Agent Model to Design Real-time Distributed Supervising and Control Systems, In: Proc. of the 9th International Conference on Control, Automation, Robotics and Vision, pp. 1-5, Singapore, 2006.

[HR06] Himoff, J. e Rzevski, G. e Skobelev, P.: Multi-Agent Logistics i-Scheduler for Road Transportation. In: Proc. of AAMAS-06 – Industry Track, ACM Press, pp 1514-1521, 2006.

[HS05] Himoff, J. e Skobelev, P. e Wooldridge, M. J.: MAGENTA Technology: Multi-Agent Systems for Industrial Logistics, In: Proc. of AAMAS-05 – Industry Track, ACM Press, pp. 60 – 66, 2005.

[K98] Kjenstadt, D.: Coordinated Supply Chain Scheduling, In: Ph.D. Thesis, 1998.

[L06] LaValle, M.: Planning Algorithms, Cambridge University Press, 2006.

[MP95] Muller, J. P. e Pischel, M. e Thiel, M.: Modelling reactive behaviour in vertically layered agent architectures. In Intelligent Agents: Theories, Architectures and Languages (eds M. Wooldridge and N. R. Jennings), 261-276. Springer, Berlin, 1995.

[ND06] Neagu, N. e Dorer, K. e Greenwood, D. e Calisti, M.: LS/ATN: Reporting on a Successful Agent-Based Solution for Transport Logistics Optimization, IEEE 2006 Workshop on Distributed Intelligent Systems (WDIS'06), Prague, Czech Republic, June 15-16, 2006.

[PG07] Palopoli, F. e Greco, S.: A simulation based agent architecture for supporting decision making in dynamic environments. In: Proc. of ESM'2007, St. Julian, Malta, 22-24 October, 2007.

[PG07] Palopoli, F. e Greco, S.: Stato dell'arte sulle applicazioni di sistemi multi-agente in ambito logistico - Progetto Promis.

[P08] Palopoli, F.: Documento di descrizione di un ambiente multi-agente per la gestione di attività di CT - Progetto Promis.

[R97] Russell, S.: Rationality and Intelligence. *Artificial Intelligence*, 94, pagg 57-77, 1997

[RG95] Rao, A. S. e M. Georgeff, M.: BDI agents: from theory to practice. Proc. Int. Conference on Multi Agents Systems (ICMAS-95), 312-319, 1995.

[RN03] Russell, S. e Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice-Hall, 2003.

[SF98] Sadeh, N. e Fox, M.: Variable and Value Ordering Heuristics for the Job Shop Scheduling Constraint Satisfaction Problem: A State-of-the-art Review of Job-Shop Scheduling Techniques, Artificial Intelligence 86(1), pp. 1-41, 1998.

[SH99] Sadeh, N. e Hildum, D. e Kjenstadt, D. e Tseng A.: MASCOT: An Agent-Based Architecture for Coordinated Mixed-Initiative Supply Chain Planning and Scheduling, In: Proc. 3rd Intern. Conf. on Autonomous Ag.(Agents'99), Seattle, 1999.

[W99] Weiss G.: Multiagent Systems - A Modern Approach to Distributed to Artificial Intelligence, MIT-Press, 1999.

[WJ95] Wooldridge, M. J. e Jennings, N. R.: Agent theories, architectures and languages. ECAI-94 Workshop on Agent Theories, Architectures, and Languages, Amsterdam, pages 408–431, 1995.

[W97] Wooldridge, M. J.: Agent-based software engineering. IEEE Proceedings on Software Engineering, 144(1), 26-37, 1997.

[W02] Wooldridge, M.: An Introduction to Multi Agent System. Wiley and Sons, 2002.

[W04] Wooldridge, M.: Responding to Real-World Complexity: Introduction to Multi-Agent Technology – Magenta Technology Whitepaper, August 2004.

[WM95] Wilkins, D. E. e Myers, K. L. e Lowrance, J. D. e Wesley, L. P.: Planning and Reacting in Uncertain and Dynamic Environments, Journal of Experimental and Theoretical AI, vol. 7, no. 1, pp. 197-227, 1995.