ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
SEDE DI CESENA
SECONDA FACOLTÀ DI INGEGNERIA
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Simulazione dell'evoluzione di una zona balneare con tecnologia ad agenti

Relazione per il corso di:

Sistemi Multi Agenti LM

Presentata da:

Enrico Gramellini e Fabio Magnani

docente:

Andrea Omicini

Anno Accademico 2010/2011

Indice

Capitolo 1

Introduzione

L'obiettivo di questo progetto è quello di realizzare un sistema di simulazione dell'evoluzione di una zona balneare, sfruttando la tecnologia di programmazione ad Agenti. La visione di tale sistema in ottica della tecnologia utilizzata ha la seguente organizzazione:

- la zona balneare rappresenta l'ambiente
- gli stabilimenti balneari rappresentano gli artefatti
- gli agenti possono essere di due tipi diversi:
  - gestori degli stabilimenti balneari
  - utenti degli stabilimenti balneari

I gestori degli stabilimenti balneari hanno l'obiettivo di guadagnare denaro attraverso i servizi offerti agli utenti.

Gli utenti hanno ognuno il proprio obiettivo e le proprie caratteristiche, ad esempio possono esistere utenti che:

- vogliono solo prendere il sole
- vogliono solo fare sport
- usufruiscono solo dell'area bar
- usufruiscono di tutti i servizi esistenti

- ...

Gli utenti tenderanno ad andare in uno stabilimento balneare piuttosto che in un altro In base ai servizi che esso offre.

Per ottenere il proprio obiettivo i proprietari degli stabilimenti hanno più modi per farlo, ad esempio:

- migliorare i servizi che fanno fruttare di più
- eliminare spese eccessive in caso di scarsa situazione economica
- ...

L'obiettivo del progetto è quello di studiare il comportamento del sistema degli stabilimenti balneari modificando i parametri in gioco, ad esempio:

- cosa succede se viene montata una spiaggia libera piena di campi da gioco vicino ai bagni? di conseguenza come cambieranno gli stabilimenti i proprietari? come si comporteranno i clienti?
- cosa succede quando i proprietari fanno modifiche al bagno ad esempio aggiungendo campi da gioco o lettini per prendere il sole? come si comportano i clienti?

In questo modo gli agenti manifesteranno un comportamento autonomo e adattivo in base a come evolve l'ambiente. Inoltre gli artefatti possono essere creati, modificati ed usati dagli agenti; rispettivamente possono essere:

- creati dai gestori
- modificati sia dagli utenti che dai gestori (si pensi allo stato di usura delle strutture sportive: gli utenti usandole le consumano, mentre i gestori notando i consumi possono provvedere nell'eseguire della manutenzione)
- usati dagli agenti utenti

La modellazione degli stabilimenti balneari in termini di artefatti, può portare alla gestione dei servizi che esso offre sia in termini anch'essi di artefatti (artefatto composto da sotto-artefatti), sia in termini i semplici operazioni dell'artefatto. Questo progetto è stato realizzato seguendo la prima filosofia.

Capitolo 2

Analisi dei requisiti

In questo capitolo sarà definita la modellazione del sistema in base ai requisiti descritti nell'introduzione.

2.1 Struttura

Le utenze del sistema sono rappresentate dai gestori degli stabilimenti balneari e dai clienti; entrambe le utenze hanno un proprio obiettivo (descritto nell'introduzione) che ne influenzerà il comportamento sul sistema.

Gli stabilimenti balneari (chiamati più comunemente "bagni") rappresentano i componenti utilizzati e gestiti dagli utenti del sistema. Tali componenti sono in grado di offrire i seguenti servizi (non obbligatoriamente tutti):

- zona relax, composta da sdrai utilizzabili a pagamento dai clienti
- strutture sportive (campi da gioco) utilizzabili gratuitamente da almeno (e al massimo) 2 clienti
- zona bar, all'interno della quale i clienti possono effettuare consumazioni

Il comportamento delle utenze del sistema varia a seconda degli obiettivi che essi hanno e il raggiungimento di questi ultimi comporta l'analisi dei fattori in gioco, i quali possono essere ad esempio:

- qualità/quantità dei servizi e numerosità delle utenze: i clienti possono scegliere di frequentare un bagno piuttosto che un altro in base ai prezzi dei servizi offerti, ai clienti che sono presenti, alla quantità dei servizi offerti, ecc...
- costi di gestione e budget: i gestori possono prendere decisioni manageriali sul bagno in base al budget in possesso e ai costi di gestione dei servizi

2.2 Comportamento

In questa fase dell'analisi dei requisiti il comportamento del sistema si modella in termini di azioni che possono essere svolte da parte degli utilizzatori.

2.2.1 Gestore bagno

Iterazioni:

1. definizione del budget a disposizione.
2. creazione del bagno (inserimento dei servizi in base al budget e alle volontà del gestore).
3. mantenimento dell'attività: questa fase si basa sull'analisi costante dei fattori in gioco con l'unico scopo che è quello di far fruttare l'attività. I risultati ottenuti da tale analisi quando soddisfano certe condizioni, sono l'input che farà agire il gestore effettuando modifiche all'attività.

2.2.2 Cliente

Iterazioni:

1. definizione delle proprie caratteristiche che ne contraddistinguono il tipo di comportamento che avrà sul sistema (cioè quali servizi tenderà ad usufruire).
2. scelta del bagno in base alle caratteristiche del cliente e del bagno stesso.
3. scelta e esecuzione di un servizio messo a disposizione dal bagno scelto.
4. aggiornamento delle proprie caratteristiche in eseguito all'esecuzione del servizio (anche nel caso in cui l'esecuzione non sia andata a buon fine).

5. ri-esecuzione delle iterazioni dal punto 2.

2.3 Interazione

Le interazioni del sistema sono state modellate in termini di comunicazioni tra i componenti del sistema; in particolare le interazioni avvengono solo tra clienti $< - >$ bagni e gestori $< - >$ bagni, quindi clienti e gestori non comunicano mai direttamente: le uniche interazioni possibili tra clienti e gestori avvengono di riflesso passando obbligatoriamente dai bagni.

Le interazioni tra gestori bagno e bagni riguardano in primo luogo l'ottenimento delle informazioni sul bagno da parte del gestore per raggiungere il proprio obiettivo. Le informazioni ottenute vengono poi elaborate dal gestore, il quale in seguito prende decisioni in base a suoi parametri interni: tali decisioni possono scaturire altre interazioni di tipo esecutivo da parte del gestore verso il proprio bagno.

Quindi la fase di raccolta dei dati assume una rilevanza notevole sulla quantità di messaggi scambiati tra questi due componenti del sistema, in quanto eventuali interazioni esecutive da parte del gestore verso il bagno si verificano solo di fronte allo scatenarsi di certi eventi.

Le interazioni tra clienti e bagni sono suddivise anche in questo caso in una parte di scambio di messaggi di richiesta di informazioni da parte dei clienti verso i bagni; queste richieste avvengono tutte per raggiungere il parziale scopo da parte del cliente di decidere in quale bagno andare. Le altre interazioni invece riguardano le decisioni prese dal cliente verso il bagno scelto, cioè le richieste dei servizi.

Capitolo 3

Analisi del problema

Nel capitolo precedente è stato modellato il sistema sulla base dei requisiti, astraendo dalle tecnologie utilizzate per la realizzazione. In questo capitolo invece sarà affrontata la modellazione del sistema orientata alla tecnologia utilizzata per la realizzazione, cioè la tecnologia ad Agenti. Il sistema da questo punto in avanti diventa un sistema multi-agente (SMA).

3.1 Architettura

Con l'introduzione del meta-modello A & A (Agent & Artifact) [1], un sistema multi-agente può essere modellato in termini di agenti, artefatti e workspaces.

Gli agenti sono entità (pro-)attive incaricate di goals/tasks che complessivamente costruiscono tutto il comportamento del MAS, mentre gli artefatti sono entità reattive che forniscono servizi e funzioni che fanno lavorare singoli agenti insieme in un MAS, e che formano l'ambiente dell'agente a seconda delle esigenze del MAS.

In altre parole gli artefatti sono risorse e strumenti costruiti, usati, manipolati dinamicamente dagli agenti per supportare/realizzare le loro attività individuali e collettive [2].

I workspaces sono i contenitori concettuali di agenti e artefatti che definiscono la topologia del sistema.

Di seguito è riportato un elenco di caratteristiche per sottolineare quali sono le differenze che contraddistinguono gli agenti e gli artefatti [3].

- Gli agenti sono autonomi, gli artefatti non lo sono.

- Gli agenti incapsulano il controllo, gli artefatti no.
- Gli agenti sono pro-attivi, gli artefatti non lo sono.
- Gli agenti sono opachi, gli artefatti sono trasparenti.
- Gli artefatti sono prevedibili, gli agenti non lo sono.
- Gli agenti possono avere un goal/task, gli artefatti no.
- Gli artefatti hanno funzioni, gli agenti non le hanno.
- Gli agenti usano artefatti, ma non usano agenti.
- Gli agenti parlano con agenti, ma non parlano con artefatti.
- Gli agenti sono progettati per governare, gli artefatti sono progettati per servire.

3.2 Modello degli agenti

Il modello degli agenti utilizzato in questo MAS è quello Jason e per interfacciarlo adeguatamente (architettura) all'ambiente CArtAgO (vedi paragrafo 4.1.2) ogni agente in fase di creazione è dichiarato come agentArchClass c4jason.CAgentArch.

In aggiunta agli agenti modellati nel capitolo precedente che sono i protagonisti del ciclo di vita del MAS, inizialmente vengono creati altri due agenti *init* e *admin*.

Agente init: Per prima cosa questo agente crea un artefatto chiamato AInitGUI; successivamente inserisce nello spazio di tuple l'identificativo dell'artefatto appena creato: questa operazione è necessaria per dar la possibilità ad altri agenti di poterlo utilizzare (si ricorda che la tecnologia di comunicazione che sfrutta i "tuple space" è presente in ogni workspace ed è realizzata sfruttando il meccanismo await. Un esempio pratico lo si può trovare nella documentazione ufficiale del progetto CArtAgO [2]).

Agente admin: Preleva dallo spazio di tuple l'identificativo di AinitGUI e si mette in ascolto degli eventi di tipo *signal* da esso generati. Quando cattura l'evento *startSimulazione* esegue le seguenti azioni:

- Crea tanti agenti cliente e gestoreBagno quanti ne sono stati inseriti dall'utente attraverso la GUI.
- Crea alcuni artefatti di supporto al sistema (GUI, utility, ...) e inserisce i loro identificativi nello spazio di tuple.

All'arrivo dell'evento *esciSimulazione* termina la simulazione eseguendo il comando .stopMAS.

Agente gestoreBagno: di questo tipo di agente ne saranno analizzate le sue operazioni e la sua base di conoscenza, ricordando che ad ogni bagno creato verrà associato un agente di questo tipo.

- Belief: Per ogni servizio offerto dal bagno che gestisce, possiede un parametro che rappresenta il limite di richieste effettuate dai clienti che il bagno non è riuscito a soddisfare; superato tale limite, il gestore, se ha disponibilità di denaro sufficiente, proverà ad adeguare il servizio.
- Plans: Inizialmente questo agente si occupa della creazione di tutti gli strumenti (artefatti) utilizzati per eseguire la gestione del proprio bagno. Questi artefatti, devono avere la possibilità di scambiarsi le informazioni e di poter sfruttare i servizi che mettono a disposizione, quindi di interagire tra di loro; per far ciò l'agente gestoreBagno li collega attraverso le "output ports" che devono essere dichiarate nell'@ARTIFACT_INFO di ogni artefatto. Le ultime operazioni che esegue prima di poter abilitare il bagno, sono:
  - informare la AGUI (artefatto che realizza una delle GUI - vedremo in seguito il dettaglio della modellazione degli artefatti) che è stato creato un nuovo bagno
  - inizializzazione del budget
  - aggiunta di alcuni servizi.

  A questo punto l'agente ha terminato la fase di inizializzazione del sistema e inizia ad osservare i cambiamenti che avvengono nell'ambiente: le azioni successive che compirà saranno quindi dipendenti dalle situazioni che verranno a verificarsi. Per realizzare tale logica sono stati utilizzati i seguenti meccanismi:

proprietà osservabili e segnali(eventi), descritti rispettivamente nei paragrafi 4.3.1 e 4.3.2.

Agente cliente: la base di conoscenza di ciascun cliente ne caratterizza la sua tipologia e ne influenza le decisioni che esso prenderà; tali informazioni vengono settate inizialmente dall'utilizzatore del sistema attraverso una delle GUI iniziali (Figura 4.2).

- Belief: concentriamo l'attenzione solo su alcune proprietà, in particolare quelle utilizzate all'arrivo di un determinato evento:
  - bagniLettini([]) ,bagniDrink([]), bagniServizio([]), bagniClienteAttesaCampi([]): contengono rispettivamente la lista dei bagni ordinata in base al costo dell'affitto dei lettini, al prezzo dei drink, alla qualità del servizio del bar e ai clienti in attesa di poter giocare in un campo. Tali liste hanno una rilevanza importante per quanto riguarda la scelta del bagno da frequentare. Nella successiva descrizione del comportamento, si noterà meglio come vengono usate le proprietà appena descritte, mentre nel paragrafo 4.3.1 saranno analizzati i dettagli dei loro aggiornamenti.
  - attivita(Lettini,Campi,Bar): ogni parametro di questà proprietà rappresenta una percentuale di preferenza per i servizi offerti dai bagni. Il loro aggiornamento avviene in seguito al tentativo di esecuzione di una di esse: in caso di esito negativo si diminuisce la percentuale del servizio che si intendeva utilizzare e si incrementano le altre; in caso di esito positivo aumenta quella appena eseguita e di conseguenza calano le altre. La somma di questi tre valori è espressa in percentuale.
- Plans: anche per questo tipo di agente inizialmente viene eseguita una fase di inizializzazione nella quale vengono anche collegati all'agente gli artefatti che esso utilizzerà. Prima di effettuare qualsiasi azione resta in attesa che tutti i bagni siano stati abilitati; tale segnalazione viene effettuata dall'artefatto ABagni. Successivamente sceglie quale servizio intende utilizzare in base ai propri parametri presenti nella base di conoscenza. Di seguito è rappresentato il comportamento eseguito dall'agente, in base al servizio scelto:

- Lettini: la decisione sul bagno nel quale fruire il servizo la effettua basandosi sui costi attuali di noleggio dei lettini presenti nella lista bagniLettini([]); viene quindi:
  - richiesto il servizio ad uno a caso tra il 33% dei bagni che presentano il prezzo minore
  - utilizzato per un certo quantitativo di tempo
  - al termine l'agente torna alla scelta dell'attività da eseguire.

  Nel caso in cui non ci dovesse essere disponibilità di lettini, allora vengono modificate le percentuali (settate nella base di conoscenza) della proprietà attivita(Lettini,Campi,Bar), diminuendo l'attività appena scelta e aumentando le altre due. A questo punto viene ri-effettuata la scelta del servizio da usufruire: se corrisponde con il precedente, il cliente scarta il bagno corrente e sceglie il successivo bagno che presenta il costo minore di noleggio di lettini; in questo caso però per effettuare la scelta viene utilizzato l'artefatto ABagni dato che la lista è sempre in continuo aggiornamento e che potrebbe verificarsi il caso in cui scegliendo la seconda posizione si ottenga sempre lo stesso bagno.
- Campi: per occupare un campo, ci devono essere due clienti disposti a giocare nello stesso bagno e nello stesso momento; il primo che arriva attende per un determinato periodo che un altro cliente sia disposto a giocare; se il tempo scade cambia bagno. Il primo tentativo di gioco viene effettuato nel bagno corrente; in caso di insuccesso, utilizzando la lista bagniClienteAttesaCampi, il cliente sceglie di spostarsi in un bagno nel quale è presente almeno un cliente in attesa (in caso di più bagni che soddisfano questo requisito, ne sceglie uno a caso). Infine se non ci sono clienti in attesa, utilizzando l'artefatto ABagni viene scelto il bagno con il numero maggiore di clienti, perchè la probabilità che qualcuno dopo avere eseguito la propria attività scelga di giocare in un campo è più alta. Un campo occupato viene liberato dopo un determinato tempo scelto dal secondo cliente arrivato; il primo cliente si accorge che il tempo è scaduto quando gli viene notificato l'evento campoLiberato.
- Bar: per questo servizio esistono due tipi di preferenze che il cliente può

esprimere: in base alla qualità del servizio o al prezzo del drink. La preferenza di un cliente è un suo parametro settato inizialmente nella sua base della conoscenza. La gestione della non disponibilità del servizio bar in un bagno segue la linea degli altri due servizi, con la ovvia differenza che la scelta successiva si basa sul parametro di configurazione del cliente relativo al servizio bar. Le circostanze che possono causare l'insuccesso nella fruizione del servizio bar sono le seguenti:

* Il bar non esiste.
* Il bar esiste ma non ci sono baristi.
* La fila al bar è troppo lunga.

In caso di esito positivo nella fruizione del servizio, l'agente potrebbe essere messo in fila ad aspettare oppure essere servito subito. In entrambi i casi, si mette in attesa della segnalazione effettuata dall'artefatto che gestisce il bar il quale gli indicherà quando il drink richiesto è pronto. Nel caso in cui si verifichi l'evento che rappresenta il licenziamento di un barista, i clienti in attesa nella sua coda non riusciranno a completare la propria volontà e percepiranno il nuovo stato del bar che diventa non disponibile.

3.3 Modello degli artefatti

Prima di iniziare ad elencare tutti gli artefatti presenti nel sistema, definiamo maggiormente le importanti proprietà e funzionalità che ci mettono a disposizione questi componenti, oltre alle caratteristiche viste in precedenza.

Principalmente sono creati per essere usati dagli agenti e per fornirgli un ambiente a seconda delle esigenze del MAS con la tecnica delle proprietà osservabili, ma possono anche direttamente manifestargli alcune informazioni attraverso segnali specifici. Infine, grazie all'uso di un particolare meccanismo detto outports sono in grado di comunicare tra di loro e quindi di poter usufruire dei servizi messi a disposizione da altri artefatti.

Precisazione sulle figure che sono sottostanti:

- I quadrati rossi esprimono le proprietà osservabili;

- I tondi verdi caratterizzano le operazioni utilizzabili dagli agenti che compongono l'interfaccia;
- I triangoli azzurri rappresentano le funzionalità utilizzabili da altri artefatti.

AInitGUI: offre la possibilità di iniziare una simulazione facendo decidere all'utente, quanti sono i bagni ed i clienti(divisi per obiettivi).

AGUI: visualizzazione descrittiva dello stato dei bagni, con le caratteristiche utili: budget, prezzo dei lettini, del drink e la qualità del servizio offerto al bar.

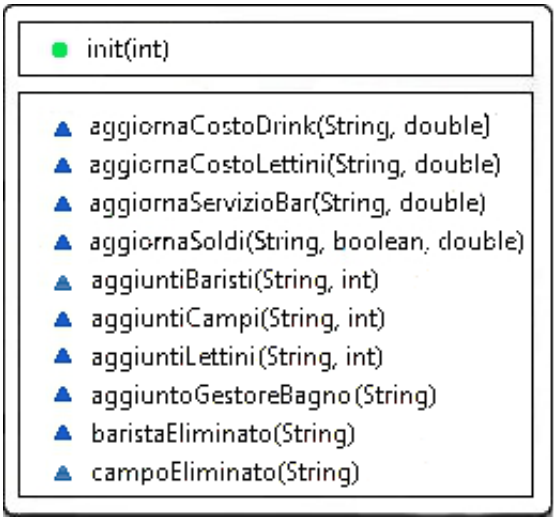

Figura 3.1: Artefatto AGUI

AGraphicsGUI: visualizzazione grafica dello spostamento dei clienti. Vengono disegnati tutti i bagni e una regione di "Attesa". Ogni cliente viene rappresentato con un cerchio colorato in modo diverso a seconda dell'attività che sta svolgendo.

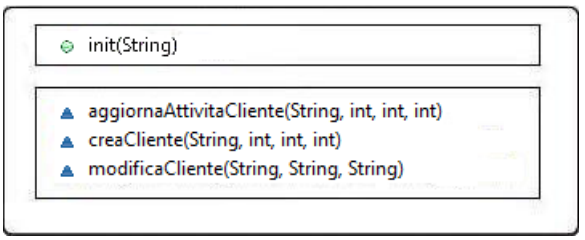

Figura 3.2: Artefatto AGraphicsGUI

Grazie a queste ultime due GUI è possibile visualizzare l'esito delle simulazioni e soprattutto lo stato di avanzamento, grazie alle visualizzazioni della maggior parte dei parametri in gioco nel sistema. Ad esempio, impostando 5 agenti con la tendenza ad utilizzare il servizio relax nei lettini. con due bagni si potrà notare facilmente che inizialmente lo spostamento avverrà maggiormente verso il bagno con il prezzo lettini inferiore, mentre successivamente accadrà che l'altro bagno tenterà di abbassare il suo costo fino a che non riuscirà ad essere il più basso; a questo punto si potrà notare che i clienti tenderanno a spostarsi dentro di esso.

AUtility: mette a disposizione degli agenti alcune funzionalità di utilità più generali.

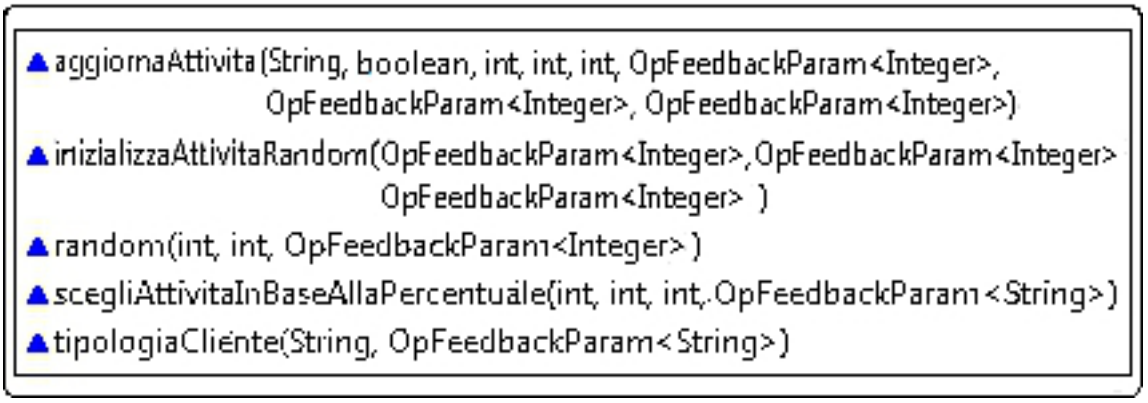

Figura 3.3: Artefatto AUtility

ABagni: memorizza i bagni creati e offre un supporto ai clienti nella scelta del bagno a seconda dei suoi belief e di quale attività vuole eseguire.

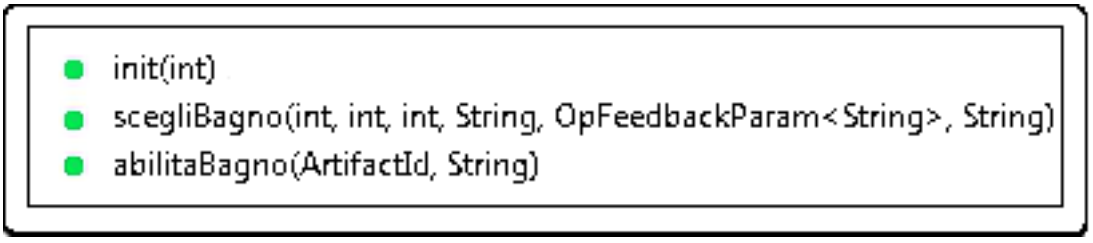

Figura 3.4: Artefatto ABagni

AGestioneBagno: si occupa della gestione dell'intero bagno ed è strutturato in sotto-artefatti: AGestioneBar, AGestioneCampi e AGestioneLettini. Più

precisamente gestisce le seguenti situazioni:

- Periodicamente detrae dal budget il costo dello stipendio dei baristi;
- Segnala ai clienti quando un barista viene licenziato;
- Segnala al gestore quando, per un determinato periodo, non avvengono richieste di lettini o di drink.
- Gestisce l'attesa del campo da parte di un cliente. Nel caso il tempo scada, lo avvisa tramite un evento dedicato.
- Attraverso le sue proprietà osservabili tiene aggiornati costantemente i clienti sui cambiamenti che avvengono nell'ambiente. Per esempio: ogni volta che viene modificato il prezzo dei lettini o del drink, viene generato un evento verso tutti i clienti.

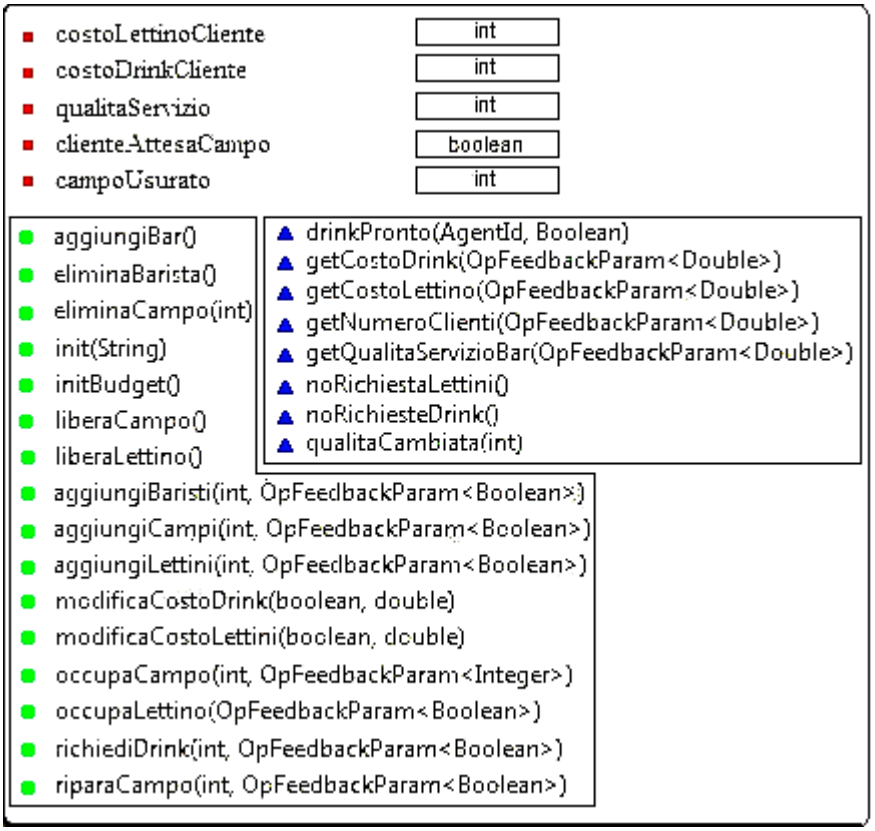

Figura 3.5: Artefatto AGestioneBagno

AGestioneBar: si occupa della gestione del servizio bar, la quale principalmente si concentra tutta sullo svolgimento del lavoro dei baristi, i quali possono essere aggiunti o licenziati; ogni barista ha a disposizione una lista utilizzata per identificare in un determinato momento quale cliente servire. Quando avviene un licenziamento, bisogna notificarlo ai clienti associati al barista. Inoltre ogni periodo di tempo prefissato verifica l'eventuale presenza di richieste del servizio e in caso negativo avvisa il gestore.

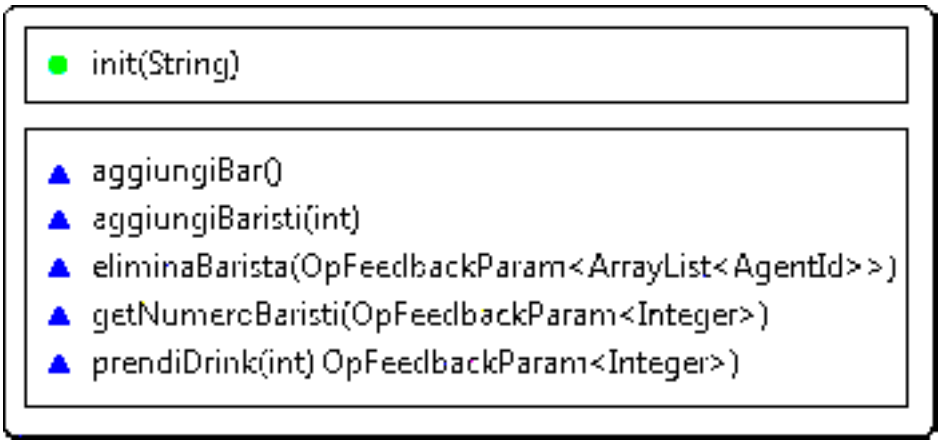

Figura 3.6: Artefatto AGestioneBar

AGestioneCampi: si occupa della gestione del servizio campi da gioco. Per ognuno di questi memorizza i clienti che lo stanno usando ed il suo stato di usura. Quando lo stato di usura raggiunge un livello limite, può essere riparato oppure rimosso. Infine, offre la possibilità di creare, occupare o liberare altri campi.

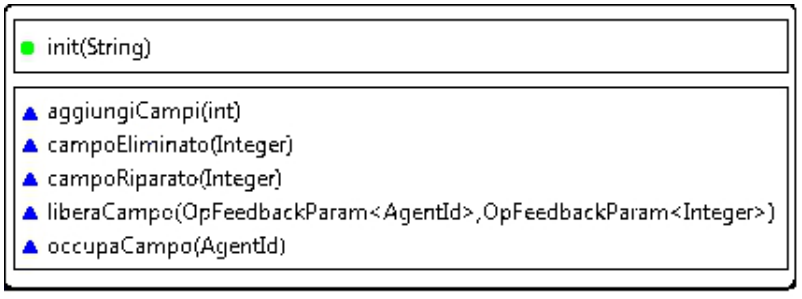

Figura 3.7: Artefatto AGestioneCampi

AGestioneLettini: si occupa della gestione del servizio di relax sui lettini. Offre i servizi base per occupare e liberare i lettini i quali saranno utilizzati dall'artefatto AGestioneBagno in base alla sua logica interna. Le operazioni di questo artefatto mettono a disposizione un parametro di ritorno che indica se l'operazione è andata o meno a buon fine. Un'ulteriore operazione (interna) presente in questo artefatto si occupa della verifica di richieste del servizio eventualmente inoltrate; è una verifica periodica effettuata per avvisare (attraverso l'artefatto AGestioneBagno) l'agente gestoreBagno in modo tale che possa prendere dei provvedimenti nel caso in cui si verifichi un numero elevato di richieste non esaudite a causa delle caratteristiche del servizio.

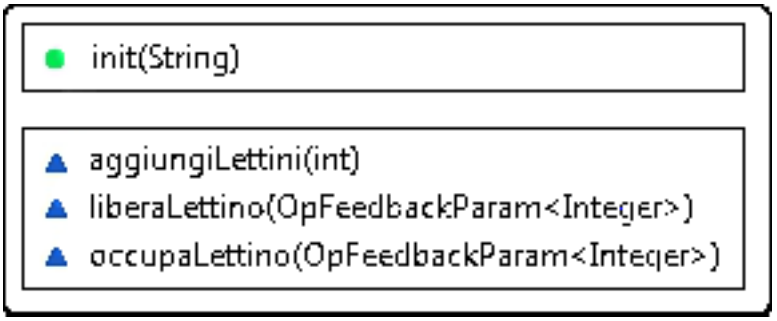

Figura 3.8: Artefatto AGestioneLettini

Capitolo 4

Progetto

4.1 Tools

4.1.1 Jason

Jason [4] è un interprete Java-based per una versione estesa di AgentSpeak. Implementa la semantica operativa di tale linguaggio e fornisce una piattaforma per lo sviluppo di sistemi multi-agente, con molte funzioni personalizzabili dall'utente. Aggiunge una serie di potenti meccanismi per migliorare le capacità dell'agente. Jason è disponibile Open Source, ed è distribuito sotto GNU LGPL.

Utilizza Java come linguaggio di basso livello per realizzare meccanismi e personalizzare l'architettura. La documentazione delle API è disponibile sul sito del progetto.

Oltre ad interpretare il linguaggio originale AgentSpeak, Jason dispone anche di:

- Una gestione dei guasti dei piani;
- Supporto per lo sviluppo di ambienti;
- Una libreria di "internal actions";
- Sostegno alle organizzazioni MAS e di agenti che ragionano su di esse;
- ecc...

Di seguito sono riportati molto in modo sintetico i costrutti principali del linguaggio AgentSpeak [5]:

- Beliefs: stato attuale dell'agente, informazioni sull'ambiente e altri agenti;
- Goals: obiettivo (desiderio) che l'agente vuole ottenere e su cui egli opera sulla base di stimoli interni ed esterni;
- Plans: strumenti procedurali che l'agente possiede per modificare l'ambiente e ottenere il suoi obiettivi (goals).

L'architettura di un agente AgentSpeak ha quattro componenti principali:

- Belief Base;
- Plan Library;
- Set of Events;
- Set of Intentions.

4.1.2 CArtAgO

CArtAgO (Common ARTifact infrastructure for AGents Open environments) [2] è un framework general purpose che rende possibile la programmazione e realizzazione di ambienti virtuali per sistemi multi-agente. Si basa sul meta-modello A&A (Agenti e artefatti) per la modellazione e la progettazione di sistemi multi-agente.

Consente di sviluppare e realizzare ambienti basati su artefatti, strutturati in "open workspaces" e gli agenti di piattaforme diverse possono unirsi in modo da lavorare insieme all'interno di tali ambienti.

CArtAgO non è vincolato ad alcun specifico modello o piattaforma ad agente, ed è particolarmente utile ed efficace se integrato con linguaggi di programmazione ad agente basato su un concetto di agenti intelligenti, in particolare quelli basati sul modello BDI (ad esempio Jason). Addirittura nell'ultima versione (2.0) è direttamente incluso Jason, ed implementato in Java. Sul sito web del progetto ci sono alcuni semplici esempi che guidano lo sviluppatore all'apprendimento.

4.1.3 Eclipse, integrazione con Jason e CArtAgO

L'ambiente di sviluppo Eclipse [6] mette a disposizione una gestione più semplice e molto più intuitiva di quella offerta da Jason, ricordando comunque che anche

quest'ultima è abbastanza facile da utilizzare. Con i seguenti passi, in pochi secondi riusciamo a costruirci un progetto Jason integrando le funzionalità del framework CArtAgO:

- Ottenere i plugin Jason per Eclipse [7] e inserirli nella cartella plugins dei Eclipse.
- Ottenere l'installazione di Jason per la propria piattaforma di lavoro e impostare nelle configurazioni le giuste librerie per un corretto supporto al plugin di eclipse.
- Ottenere CArtAgO [8].
- Creare un nuovo progetto, eseguendo New $\longrightarrow$ Project $\longrightarrow$ Jason Project.
- Aggiungere al progetto le librerie "c4jason.jar" e "cartago.jar" che si trovano all'interno della cartella "lib" di CArtAgO.

Otteniamo il file ".mas2j" del nostro progetto JAson, una cartella denominata "src/asl" dove poter inserire i nostri agenti attraverso un file ".asl" e una cartella "src/java" dove includere al nostro progetto gli artefatti con la semplice aggiunta di file ".java".

4.2 Implementazione

Prima di descrivere l'architettura generale della soluzione da noi proposta è bene precisare che è stato utilizzato il classico paradigma Model-View-Controller(MVC) per strutturare gli artefatti. In questo caso specifico il modello rappresenta gli elementi coinvolti nel sistema(bagni, campi, ...). E' importante ricordare che il modello sarà assolutamente indipendente dalle modalità di visualizzazione adottate. La parte di controller è rappresentata dalle entità che si occupano di gestire i comportamenti del sistema e utilizzano a loro volta le entità di modello. Infine la parte di view sarà responsabile della rappresentazione grafica a video del comportamento che avrà il sistema.

Nella figura 4.1 rappresentata sotto, è illustrata l'architettura del MAS da noi creato e si possono notare tutte le entità descritte nel capitolo precedente. Si riesce

a comprendere molto facilmente quali strutture vengono usate dagli agenti e il collegamenti che ci sono tra i vari artefatti.

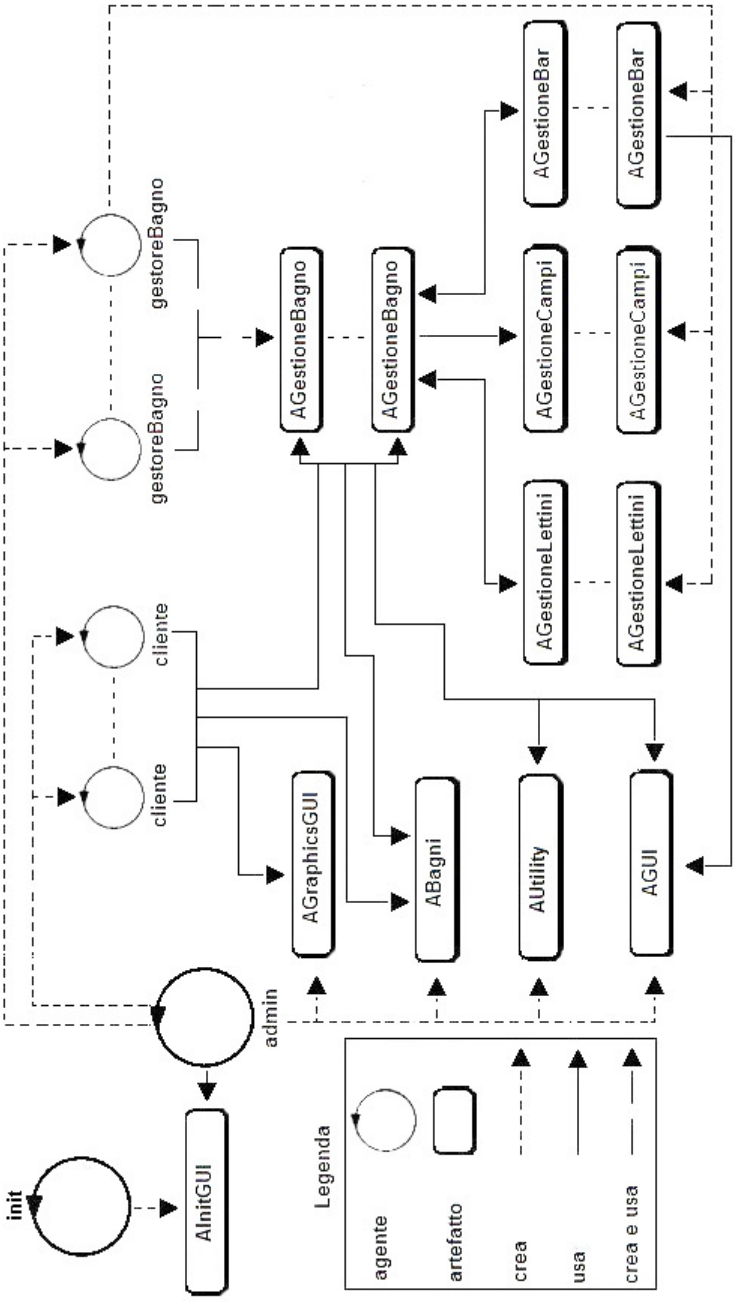

Figura 4.1: Architettura del sistema MAS

Jason fornisce un ambiente per sviluppare applicazioni ad agenti con un interfaccia grafica e alcuni strumenti per il debug. Un limite di questa tecnologia è che non consente la configurazione dinamica di un MAS perché prende in ingresso un file

statico .mas2j. Quindi per ovviare a questo problema abbiamo deciso di aggiungere due agenti al nostro sistema che si occupano soltanto dell'inizializzazione e della creazione di tutti i componenti presenti nel MAS.

Come già visto nei capitoli precedenti, l'agente init lancia un'interfaccia grafica(riportata di seguito) per far si che l'utente imposti alcuni parametri del sistema. Sarà poi compito dell'agente admin leggerli ed settarli sugli agenti e sugli artefatti da lui creati.

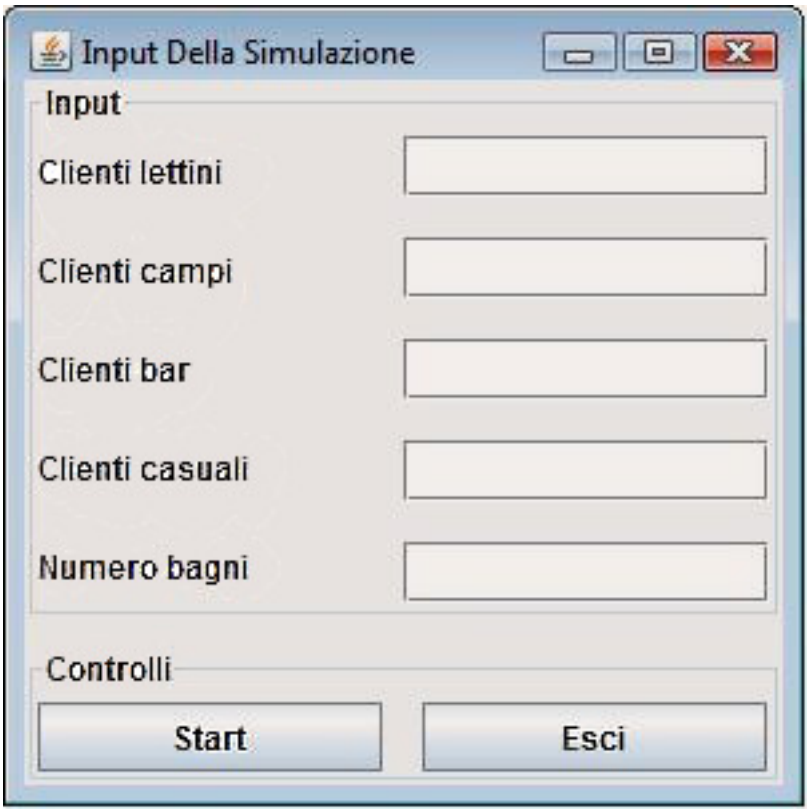

Figura 4.2: Input simulazione

Un importante limitazione presentata da jason è che non è possibile creare un numero elevato di thread quando si dispone di un elaboratore non molto potente; è stato deciso inoltre che non si possono creare più di 10 bagni a causa di una limitazione relativa alla GUI (guardare la figura 4.3).

In tale finestra è rappresentato lo spostamento che ogni cliente esegue mostrando all'utilizzatore del simulatore lo scenario corrente dell'esecuzione del MAS. Grazie ad essa si può comprendere il loro comportamento, ma soprattutto si può analizzare

su quali principi si basano le loro scelte.

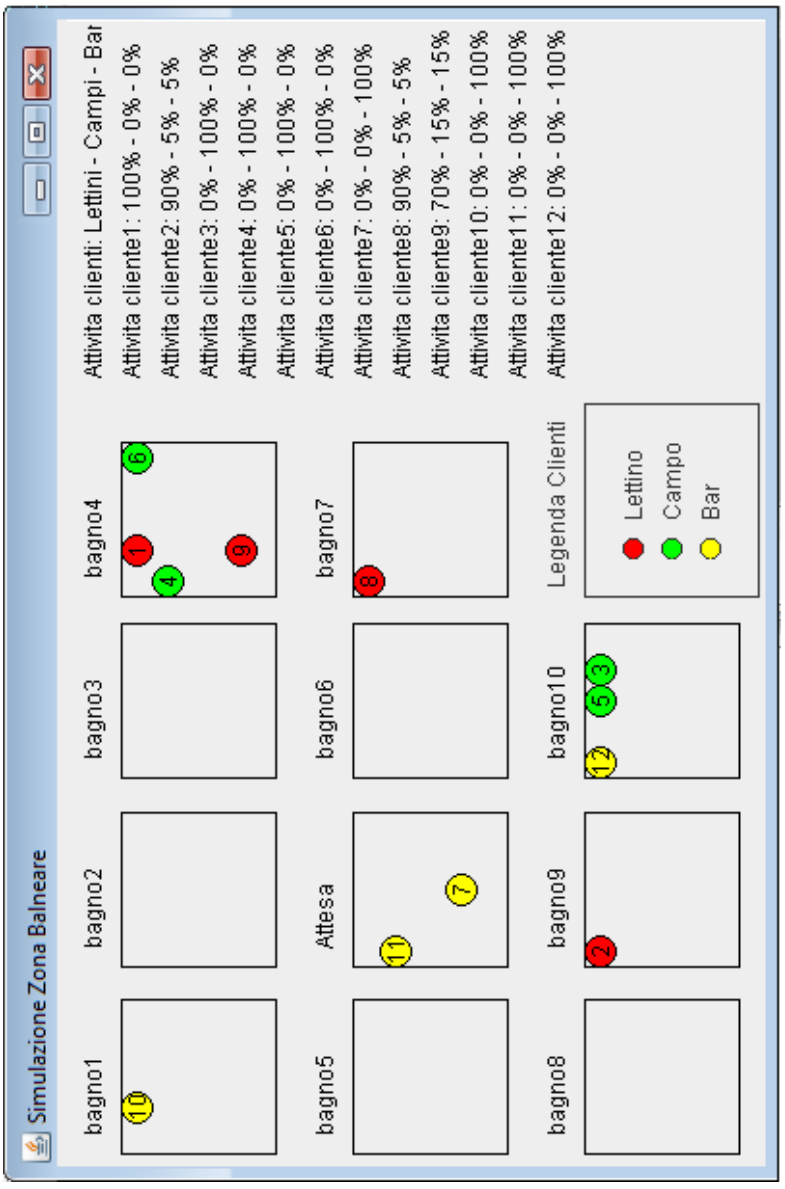

Figura 4.3: Simulazione spostamento clienti

Infine per quanto riguarda l'integrazione dell'ambiente CArtAgO in Jason, bisogna inserire nella specifica del progetto Jason MAS la seguente istruzione:

environment: c4jason.CartagoEnvironment

4.3 Dettagli implementativi

4.3.1 Proprietà osservabili

Le proprietà osservabili vengono mappate nella la base di conoscenza dell'agente che esegue il "focus" sull'artefatto che le definisce. Pertanto le modifiche alle proprietà osservabili degli artefatti vengono rilevate dall'agente come variazioni alla base di conoscenza.

Nella loro forma più generale, le proprietà osservabili sono rappresentate da una tupla, che possiede un funtore ed uno o più argomenti tipizzati. Di seguito vengono riportate le proprietà osservabili dell'artefatto AGestioneBagno:

- planCampoUsurato: di tipo intero. Questa proprietà osservata dal gestore del bagno viene modificata per notificare quale campo ha raggiunto il suo massimo stato di usura; l'agente gestore osservando questa proprietà sa quando un campo deve essere riparato e può decidere successivamente se procedere con la riparazione o se rimuoverlo (nel caso in cui ad esempio non si disponga di abbastanza denaro).

- costoLettinoCliente: di tipo intero. Questa proprietà osservata dai clienti rappresenta il costo del noleggio di un lettino in un determinato bagno. Di fronte ad una variazione, tutti gli agenti reagiscono aggiornando la propria lista bagniLettini([]) presente nei loro "belief". Questa struttura (illustrata nel paragrafo 3.2) permette di mantenere ordinati i prezzi di tutti i bagni esistenti.

- costoDrinkCliente: di tipo intero. Ogni volta che un bagno aggiorna il prezzo del drink, cambia questa proprietà in modo tale che tutti i clienti la rilevino e aggiornino la propria lista bagniDrink([]).

- qualitaServizio: tipo intero. Identifica la qualità del servizio offerto al bar e come per le precedenti due proprietà, i clienti aggiornano nelle proprie basi di conoscenza la lista bagniServizio([]) per far si che questo cambiamento possa andare ad influenzare la prossima scelta.

- clienteAttesaCampo: di tipo booleano. Rappresenta la presenza/assenza di un cliente in attesa di un altro cliente su un campo da gioco in quel bagno.

Se questa proprietà risultasse vera, aumenterebbe la probabilità di scelta di tale bagno da parte di un agente intenzionato a giocare. La lista associata utilizzata dai clienti è bagniClienteAttesaCampi([]).

4.3.2 Segnali

Come le proprietà osservabili, anche i segnali possono essere strutturati a tuple, con un funtore e uno o più argomenti tipizzati. Eseguendo il "focus" su un artefatto da parte di un cliente, si riescono a percepire i segnali da lui generati.

Questo tipo di costrutto viene usato negli artefatti riportati di seguito:

- AInitGUI, verso l'agente admin:
  - startSimulazione: per far partire la simulazione con i parametri settati nella finestra iniziale.
  - esciSimulazione: arrestare l'esecuzione della simulazione.
- ABagni, destinati all'agente cliente:
  - nuovoBagno: è stato creato un nuovo bagno, di conseguenza i clienti eseguono il "focus" su di esso in modo tale da poter percepire sia le proprietà osservabili sia i segnali.
  - bagniCreati: tutti i bagni sono stati creati, quindi i clienti possono iniziare la propria esecuzione.
- AGestioneBagno:
  - aggiungereLettini, aggiungereCampi, aggiungereBaristi: queste tre segnalazioni sono identiche e ovviamente riguardano la gestione rispettivamente dei lettini, campi e baristi. Se ci sono state richieste che non sono state soddisfatte sono inviate verso l'agente gestoreBagno che valuta la possibilità di aggiungere le strutture necessarie. Se il gestore decide di ampliare/modificare la struttura del proprio bagno, è necessario disporre dei soldi necessari per farlo.

- noRichiesteLettini, noRichiesteDrink: caso opposto al precendente. Se prima le richieste erano troppe, in questo caso abbiamo che per un determinato periodo di tempo il servizio dei lettini o quello dei drink non viene usufruito dai clienti. Ancora una volta l'agente gestoreBagno porta dei cambiamenti nella sua struttura in modo tale da attirare nuovi clienti (ad esempio calando il prezzo dei servizi).
- inRosso: il budget è andato in negativo quindi il gestore inizia a licenziare i baristi, risparmiandosi i soldi dei loro stipendi. Al gestore conviene eliminare lettini o campi visto che sono già stati affrontati i costi di costruzione.
- reimpostaLimiteBarista: quando viene eliminato l'ultimo barista reimposto i belief dell'agente gestore bagno.
- campoLiberato: quando un cliente riceve questa comunicazione significa che si sono verificate precedentemente le seguenti situazioni in questo ordine:
  1. aveva chiesto di giocare in un campo;
  2. è stato messo in attesa perchè non c'era nessun altro cliente disponibile;
  3. è arrivata un richiesta da parte di un altro cliente;
  4. hanno iniziato a giocare insieme;
  5. l'altro giocatore ha liberato il campo. Il cliente se ne accorge attraverso questo segnale e decide di ricominciare una nuova attività.
- attesaCampoScaduto: un cliente che era in attesa (in un determinato bagno) di qualche altro giocatore per occupare il campo, viene avvisato che non ci sono state richieste e quindi la sua attesa è scaduta.
- consumaDrink: quando il drink richiesto dal cliente è pronto, lo avviso tramite questo messaggio.
- baristaEliminato: un barista è stato licenziato, quindi devo avvisare tutti i clienti in fila e al balcone che non riceveranno più il drink da loro richiesto.

Capitolo 5

Installazione

Ci sono due alternative per eseguire l'applicazione:

1. Eseguendolo direttamente dall'ambiente Eclipse (la linea guida da seguire per integrare questo ambiente con Jason e CArtAgO è già stata descritta nel paragrafo 4.1.3.)

2. Usando il tools Jason (freeware) [6]. Basta cliccare su File $\longrightarrow$ Open e selezionare il file ProgettoSMA.mas2j del nostro progetto. Per poterlo lanciare cliccare sul triangolino verde come mostrato in figura.

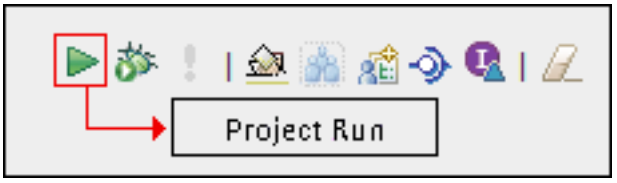

Figura 5.1: Run MAS in Jason

Capitolo 6

Conclusioni

La realizzazione di questo progetto, dimostra che è possibile creare delle entità intelligenti, basandosi su una tecnologia ad agenti; queste entità (agenti) sono quindi in grado di prendere delle decisioni per cercare di raggiungere il proprio obiettivo.

Il sistema MAS è stato implementato attraverso le tecnologie Jason e CArtAgO, integrati entrambi nell'ambiente di sviluppo Eclipse.

Queste tecnologie si sono rivelate ottimali per realizzare sistemi di simulazione come in questo caso, in quanto gli strumenti forniti proprio dalle tecnologie stesse rendono la realizzazione più veloce, ma soprattutto permettono di organizzare l'architettura del sistema in un modo più vicino alla logica di simulazione rispetto ad un possibile approccio ad esempio orientato agli oggetti, che avrebbe conseguentemente portato ad un'organizzazione orientata invece su tutti gli aspetti necessari alla realizzazione del sistema.

Il metamodello Agent & Artifact unito alle tecnologie Jason e Cartago permette di astrarre da certi aspetti sia di progetto che di implementazione, come ad esempio la realizzazione a basso livello della comunicazione tra componenti del sistema, in quanto queste tecnologie forniscono già tale supporto.

6.1 Sviluppi futuri

Questa prima versione del sistema di simulazione degli stabilimenti balneari è completa di tutte le specifiche concordate in fase iniziale. Lo stato attuale permette di

simulare diversi aspetti di questa realtà, ma i limiti essendo incalcolabili permettono di estendere questo progetto aggiungendo molte altre caratteristiche di questo tipo di realtà. Oltre alle estensioni di tipo funzionale potrebbero essere aggiunte migliorie riguardanti l'aspetto di visualizzazione della simulazione.

Per quanto riguarda quindi i miglioramenti grafici potrebbero essere apportate le seguenti estensioni:

- invece di avere due finestre separate, si potrebbe avere solo quella di visualizzazione dello spostamento dei clienti all'interno dei bagni; quando l'utente seleziona un bagno si potrebbero mostrare tutte le sue caratteristiche, ad esempio:
  - la situazione di ogni campo
  - elencare i lettini occupati
  - dare la possibilità di visualizzare tutte le decisioni prese dal gestore di quel bagno.
- avere la possibilità di studiare il comportamento del sistema degli stabilimenti balneari modificando i fattori in gioco, ad esempio: se viene montata una spiaggia libera piena di campi da gioco vicino ai bagni, come reagiscono i clienti e i proprietari degli stabilimenti? Oltre all'inserimento di spiagge libere si potrebbe anche dare la possibilità di inserire bagni in qualsiasi punto dell'ambiente.
- permettere la visualizzazione di statistiche a grana più fine.

Tra le estensioni di tipo funzionale si potrebbero apportare invece le seguenti aggiunte:

- dare la possibilità ai gestori dei bagni di prendere le decisioni non solo guardando lo stato interno del proprio bagno ma anche analizzare cosa accade in quelli vicini.
- dare la possibilità al gestore del bagno di gestire la grandezza del bagno. Quindi sapendo che i lettini occupano un certo spazio e portano un determinato

guadagno, mentre i campi attirano molti clienti ma non fruttano e occupano molto spazio, cosa conviene aggiungere? Se ad esempio il bagno non presenta ulteriore spazio libero, il gestore potrebbe pensare di eliminare qualche servizio che al momento non sta facendo fruttare abbastanza e sostituirlo con un altro.

Bibliografia

[1] Andrea Omicini, Alessandro Ricci e Mirko Viroli. Artifacts in the A&A meta-model for multi-agent systems. Autonomous agents and multi-agent systems, 2008.

[2] Alessandro Ricci, Michele Piunti e Mirko Viroli. CArtAgO. `http://cartago.sourceforge.net`.

[3] Andrea Omicini. Agents and artifacts: The A&A meta-model for multi-agent systems. `http://campus.cib.unibo.it/39515/1/6-SMA2010-metamodel.pdf`.

[4] Jomi F. Hübner e Rafael H. Bordini. Jason. `http://jason.sourceforge.net/Jason/Jason.html`.

[5] Andrea Omicini. Programming Intentional Agents in AgentSpeak(L) and Jason. `http://campus.cib.unibo.it/42284/1/L3-SMA201011-JASON_P1.pdf`.

[6] Eclipse. `http://www.eclipse.org`.

[7] Jason Eclipse plugin. `http://sourceforge.net/projects/jason/files`.

[8] Download CArtAgO. `http://sourceforge.net/projects/cartago/files/cartago/2.0/cartago-2.0.1.zip/download`.