

**PROGETTAZIONE DI UN
SIMULATORE AD AGENTI PER UN
NUOVO SISTEMA DI
VALUTAZIONE DELLE
PUBBLICAZIONI SCIENTIFICHE**

Indice

INDICE	2
1. INTRODUZIONE.....	3
2. AGENT UML.....	4
2.1 DIAGRAMMI DI SEQUENZA	5
2.2 DIAGRAMMI DI CLASSE.....	7
3. PROGETTAZIONE DEL SIMULATORE	11
3.1 AUTORE.....	11
3.2 LETTORE.....	12
3.3 SISTEMA	14
3.4 INTERAZIONI FRA GLI AGENTI	15
3.5 IL SIMULATORE	18
4. CONFIGURAZIONI POSSIBILI DEGLI AGENTI.....	20
4.1 TIPOLOGIE DI AUTORI	20
4.2 TIPOLOGIE DI LETTORI.....	20
5. ANALISI BASATA SU BDI.....	22
6. CONCLUSIONI.....	24
BIBLIOGRAFIA	25

1. Introduzione

Lo scopo di questo laboratorio è la progettazione di un simulatore basato sugli agenti per la simulazione del comportamento di un nuovo sistema di valutazione delle pubblicazioni scientifiche. Tale nuovo sistema di valutazione è stato introdotto dal Prof. Stefano Mizzaro nel suo articolo “*Quality Control in Scholarly Publishing: A New Proposal*” [8]. Il sistema prevede due diversi attori: l'autore, colui che pubblica un articolo scientifico, e il lettore, ovvero colui che legge gli articoli pubblicati ed esprime un giudizio su tale articolo. La progettazione del sistema di archiviazione pubblicazioni, gestione votazioni e interfacciamento con gli attori è stata sviluppata in altra sede. In questo lavoro ci si propone di progettare un simulatore che verifichi, mediante la creazione di una popolazione di autori ed una di lettori, la correttezza delle previsioni teoriche riguardo il corretto funzionamento del sistema di valutazione e dell'evoluzione dei punteggi di autori, lettori ed articoli.

Questo elaborato è stato suddiviso in quattro diverse sezioni. Nella prima parte verrà descritta un'estensione dell'UML per la rappresentazione degli agenti. Nella seconda parte verrà introdotto il progetto del simulatore e nella terza verranno descritte delle modalità di configurazione degli agenti per il testing di alcune particolari situazioni previste dall'ideatore del sistema. Nella quarta ed ultima parte verrà presentata una progettazione basata su BDI degli agenti proposti in questo lavoro.

2. Agent UML

Lo scopo di questo laboratorio è quindi di testare la validità del sistema di valutazione di pubblicazioni scientifiche. Per effettuare questi test si è deciso di realizzare un simulatore basato su un sistema multi-agente. Prima di procedere è quindi necessario dare qualche definizione.

Un *agente* è un'entità fisica o virtuale

- a) Che opera in un ambiente
- b) Che può comunicare direttamente con altri agenti
- c) Il cui comportamento è determinato da proprie tendenze (nella forma di obiettivi individuali o di funzioni di intenzioni/sopravvivenza che cerca di ottimizzare)
- d) Che possiede risorse proprie
- e) Che può percepire l'ambiente in cui opera (in modo limitato)
- f) Che ha una rappresentazione solamente parziale dell'ambiente
- g) Che possiede capacità e può offrire servizi
- h) Che potrebbe essere capace di riprodursi
- i) Il cui comportamento tende a soddisfare i propri obiettivi, in considerazione delle risorse e delle capacità a lui disponibili e a seconda della sua percezione, delle sue rappresentazioni e delle comunicazioni che riceve [6].

Dalla definizione si può quindi intuire che ciascun attore (autore e lettore) verrà modellato con un agente a cui verranno assegnate delle capacità e degli obiettivi da perseguire. Tali attori opereranno insieme nello stesso ambiente (il simulatore) costituendo pertanto un sistema multi-agente.

Un *sistema multi-agente* è composto dai seguenti elementi:

- a) Un ambiente
- b) Un insieme di oggetti. Questi oggetti sono situati, ovvero è possibile in ogni momento associare a ciascun oggetto una posizione nell'ambiente. Questi oggetti sono passivi, ovvero possono essere percepiti, creati, distrutti e modificati dagli agenti.
- c) Un insieme di agenti, anch'essi degli oggetti, che rappresentano le entità attive del sistema
- d) Un insieme di relazioni che legano gli oggetti (e quindi anche gli agenti) fra di essi
- e) Un insieme di operazioni che rendono possibile agli agenti di percepire, creare, distruggere e modificare gli oggetti
- f) Operatori con il compito di rappresentare l'applicazione di tali operazioni e le reazioni del mondo derivanti [6].

Per la progettazione di un software basato sugli agenti non è però sufficiente l'utilizzo dell'UML (Unified Modeling Language) standard, un linguaggio largamente accettato per la rappresentazione ingegnerizzata di software object-oriented [2]. È quindi necessario ricorrere ad un'estensione dell'UML che si occupi di introdurre tutte le caratteristiche e funzionalità richieste dagli agenti. L'UML non è sufficiente per la completa modellazione di agenti e di sistemi multi-agente principalmente per due ragioni. Innanzitutto gli agenti si differenziano dagli oggetti in quanto sono dei componenti attivi: possono prendere l'iniziativa in modo autonomo ed hanno il controllo di quando e come processare delle richieste esterne. In secondo luogo gli agenti, sempre a differenza degli oggetti, non operano in modo isolato ma, al contrario, in modo coordinato e in cooperazione con altri agenti [7]. I sistemi multi-agenti sono, infatti, delle comunità di membri indipendenti che operano in modo autonomo.

Per la progettazione di tali sistemi multi-agenti è necessario avere degli strumenti che coprano l'intero processo di creazione del software e, come detto, l'UML standard non è sufficiente. Esistono attualmente diversi progetti il cui scopo è la creazione di un'estensione per

l'UML che copra tutti i bisogni dei progettisti di sistemi ad agenti e si è resa pertanto necessaria una selezione preliminare [5]. Per questo lavoro si è quindi deciso di utilizzare l'estensione *Agent UML* (AUML) proposta per la prima volta da Bernhard Bauer e James Odell e continuamente aggiornata ed integrata [1,3]. Qui di seguito verranno approfonditi due dei diagrammi UML che hanno subito delle modifiche per la rappresentazione degli agenti e che saranno utilizzati per la modellazione di questo progetto. Questi due diagrammi sono il diagramma di sequenza, per la rappresentazione delle interazioni fra i diversi agenti, ed il diagramma di classe, per la rappresentazione degli stati interni degli agenti e delle loro interazioni con l'esterno.

2.1 Diagrammi di sequenza

Vediamo quindi i diagrammi di sequenza estesi per la gestione delle interazioni fra agenti [7]. I principi che andiamo a descrivere non coprono completamente le tematiche relative alla stesura di un diagramma di sequenza, ma riguardano solamente le caratteristiche che sono state aggiunte per trattare anche la comunicazione fra agenti.

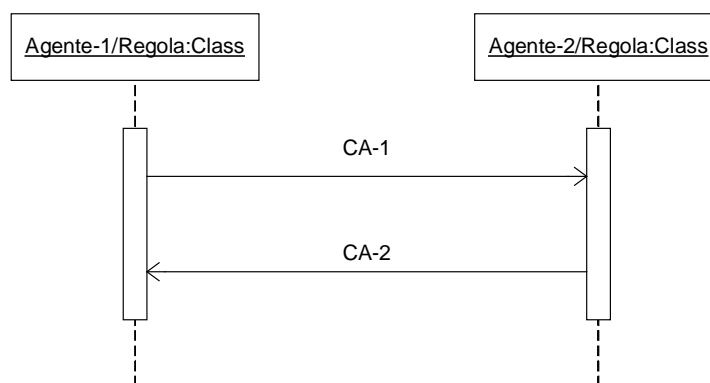
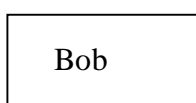


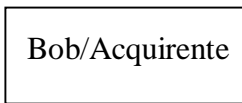
Figura 1: Modello base di comunicazione fra agenti

Nella figura 1, sebbene rappresenti una comunicazione molto banale, si possono già introdurre diverse nuove caratteristiche. Innanzitutto il rettangolo che esprime gli attori può rappresentare un semplice agente oppure un insieme di agenti, ed inoltre è possibile evidenziare non solo il tipo ma anche il ruolo dell'agente. Consideriamo l'esempio di Bob, una persona che a seconda della situazione può essere un acquirente, un venditore oppure un lavoratore. Possiamo esprimere questi fatti sfruttando appunto il rettangolo principale.

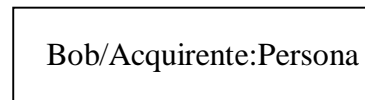
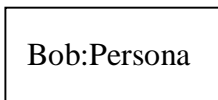
Se vogliamo esprimere il fatto che Bob è l'attore in questione, un'istanza di un agente il cui comportamento prescinde dalla situazione in cui si trova, possiamo utilizzare la seguente simbologia:



Se invece vogliamo esprimere il fatto che Bob si sta comportando da acquirente possiamo utilizzare la seguente simbologia:



Infine, se vogliamo specificare anche il tipo di agente a cui Bob fa riferimento (Persona) possiamo utilizzare le seguenti simbologie:



Nella figura 1 possiamo notare anche il significato delle frecce: nell'UML standard la freccia ha il significato di un messaggio fra due oggetti, nella sintassi di AUML tali frecce rappresentano delle *comunicazioni* fra due agenti. Poiché però gli agenti devono poter gestire più thread di comunicazione concorrenti è necessario estendere anche il sistema di comunicazione. In seguito ci si riferirà alle comunicazioni con la sigla CA (*communicative act*); per CA si intende il tipo di messaggio e tutte le informazioni correlate (per es. mittente, destinatario e contenuto).

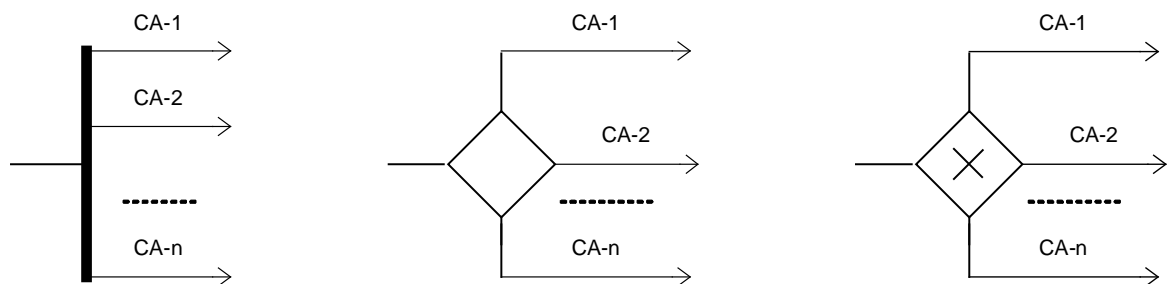


Figura 2: Estensione per le comunicazioni concorrenti

In figura 2 possiamo vedere tre diverse estensioni per la gestione di comunicazioni concorrenti. La figura 2(a) rappresenta una comunicazione concorrente di tutti i CA. La figura 2(b) rappresenta una comunicazione in cui avviene una scelta dei CA da inviare: possono essere inviati zero o più CA. Qualora si abbia un invio di più di un CA, questi vengono inviati in modo concorrente. Infine, la figura 2(c) rappresenta una comunicazione in cui esattamente uno fra gli n CA viene inviato.

In figura 3 possiamo vedere diverse modalità di impiego delle comunicazioni concorrenti. Le barre verticali multiple, o barre di attivazione, indicano che gli agenti destinatari processano tutte le comunicazioni ricevute in modo concorrente.

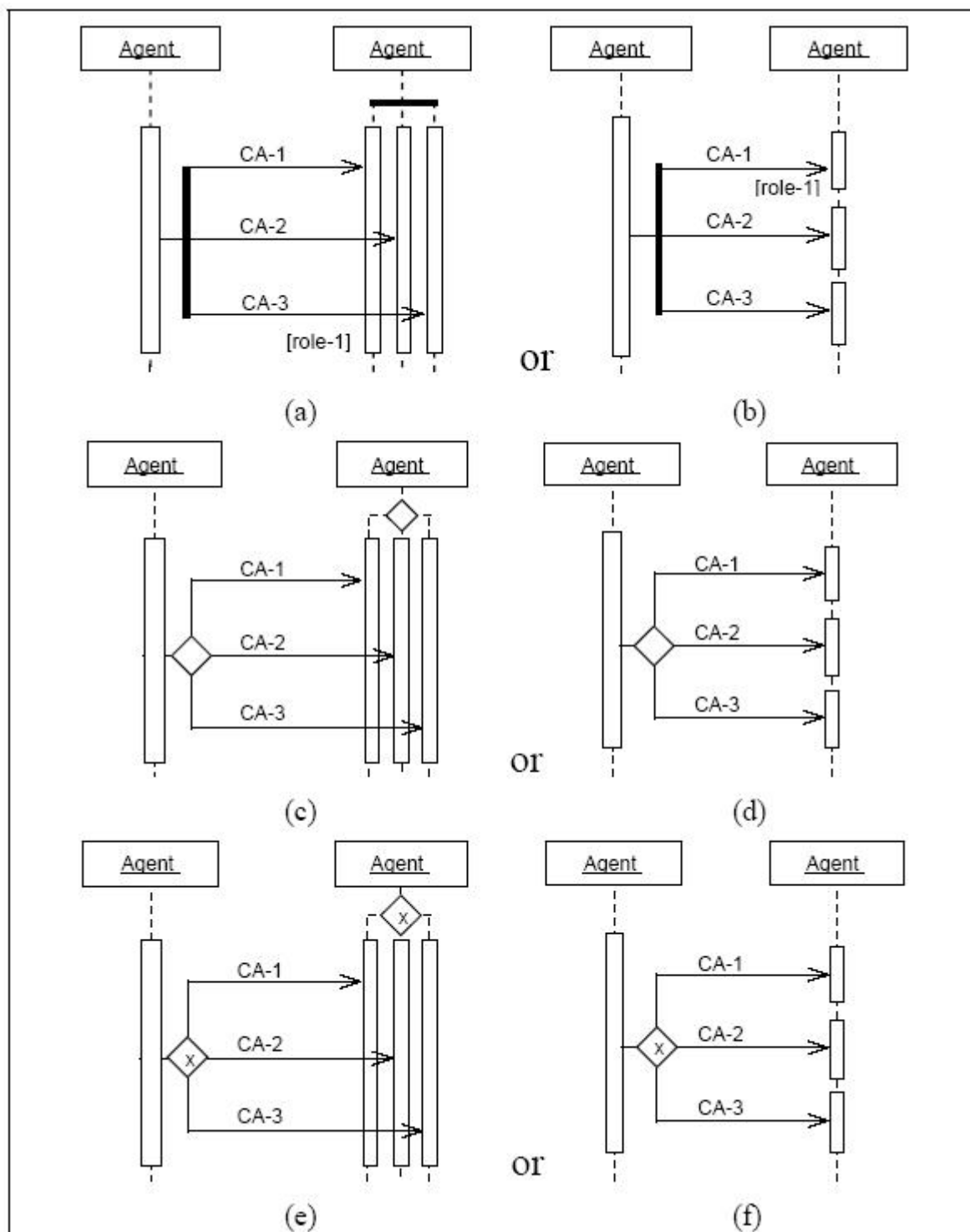


Figura 3: Diverse rappresentazioni per le comunicazioni concorrenti

2.2 Diagrammi di classe

Come è stato più volte sottolineato in precedenza, un agente si differenzia per diversi aspetti da un oggetto.

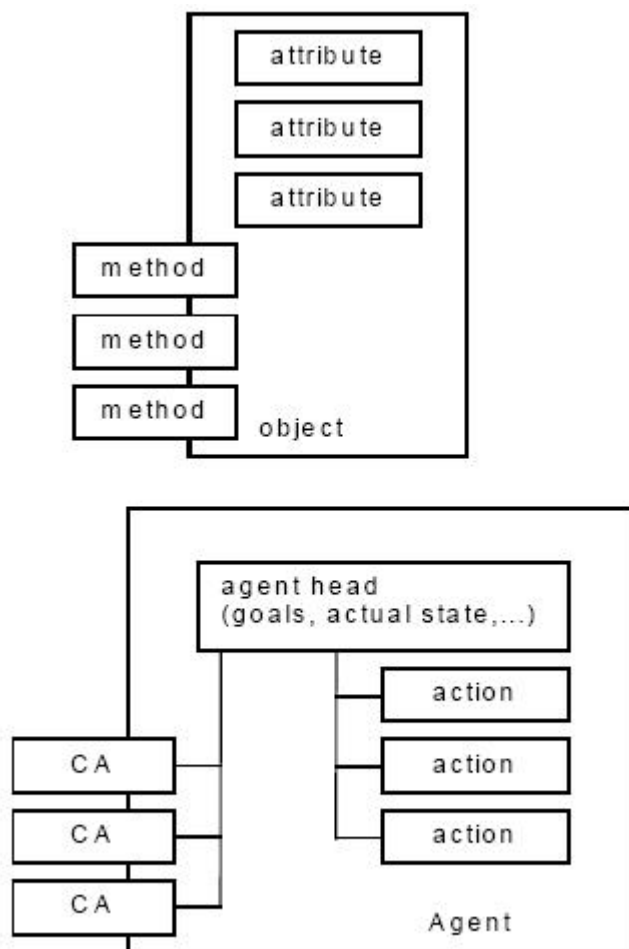


Figura 4: Struttura di un oggetto e di un agente

In figura 4 possiamo osservare la struttura di un oggetto e di un agente. Per quanto riguarda l'oggetto, si può notare come questo esponga dei metodi pubblici e mantenga al suo interno una serie di attributi. L'agente, invece, espone dei *Communicative Acts*, ovvero le tipologie di comunicazione che accetta, e mantiene al suo interno i dati inerenti il proprio comportamento: desideri, stato attuale, intenzioni, convinzioni e azioni che può intraprendere [2]. Risulta quindi evidente come i diagrammi di classe previsti dall'UML standard non siano sufficienti per esprimere tutte le caratteristiche degli agenti.

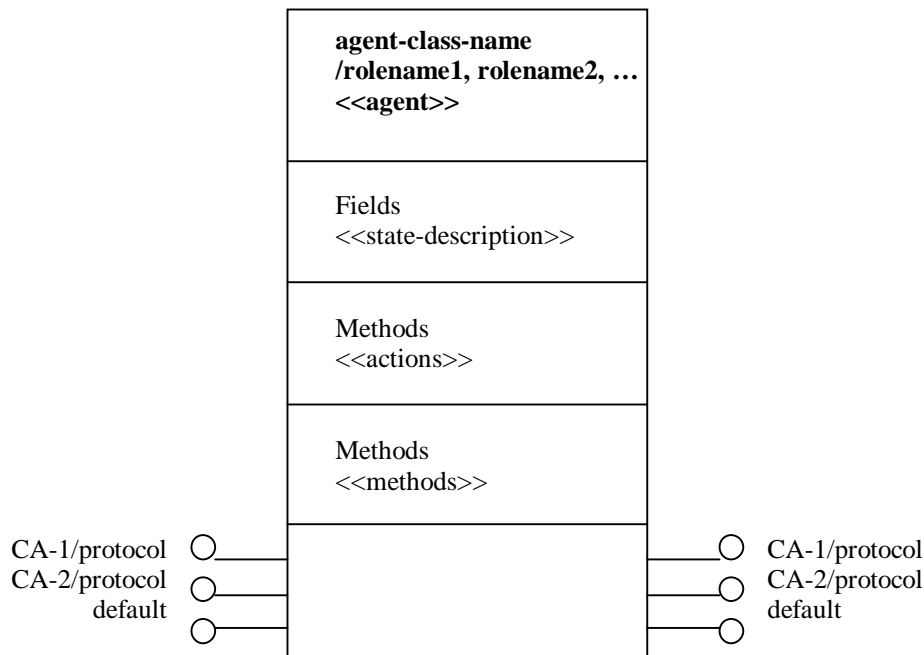


Figura 5: Diagramma di classe esteso per gli agenti

In figura 5 si può osservare come deve essere reso in AUML il diagramma di classe per rappresentare un agente. Nella prima parte possono essere specificati il nome dell'agente, il nome della classe di agenti e il ruolo dell'agente secondo quanto già descritto in precedenza.

Nella seconda parte, denominata *<<state-description>>*, devono essere specificati i campi che descrivono lo stato interno dell'agente (per es. credenze, desideri ed intenzioni). Questi valori possono essere di due tipi: persistenti (stereotipo *<<persistent>>*), se i valori vengono mantenuti tra una sessione e l'altra di esecuzione dell'agente, o non persistenti, qualora i valori vengano reinizializzati ogniqualvolta l'agente viene stoppato e fatto ripartire. Tali campi ricordano molto gli attributi degli oggetti [2].

Nella terza parte, denominata *<<actions>>*, devono essere specificate le azioni che un agente può intraprendere. Tali azioni sono visibili dall'esterno ma solamente l'agente può decidere se e quando iniziare un'azione (proprio quest'ultima caratteristica distingue le azioni dai metodi pubblici degli oggetti). Le azioni che si possono specificare sono di due tipi: autonome (stereotipo *<<pro-active>>*), se l'agente può decidere in completa autonomia di iniziare tale azione, oppure reattive (stereotipo *<<re-active>>*), qualora l'agente intraprenda tali azioni solo in risposta a determinati impulsi dall'esterno [2].

Nella quarta parte, denominata *<<methods>>*, devono essere specificati i metodi privati degli agenti. Tali metodi corrispondono ai metodi privati degli oggetti, in quanto non sono accessibili dall'esterno ma vengono utilizzati dagli agenti per compiere le proprie azioni [2].

Infine nell'ultima parte, devono essere specificati i tipi di comunicazione che l'agente può ricevere ed inviare. Tale parte rappresenta la vera e propria interfaccia fra l'agente e l'ambiente esterno. Pertanto devono essere specificati i tipi di messaggio che l'agente può ricevere e quindi elaborare ed i tipi di messaggio che l'agente può inviare all'esterno. Qualora si voglia che l'agente possa elaborare tutti i messaggi in ingresso è possibile specificare un CA di tipo *default* al quale

faranno riferimento tutti i messaggi non previsti. Se infine un CA ricevuto non può essere capito/elaborato dall'agente è possibile prevedere l'invio di un CA di tipo *not-understood*. Le informazioni riguardanti i CA possono essere modellate e trattate come classi ed oggetti [2].

Esiste una differenza piuttosto sottile fra le azioni che un agente può intraprendere e le comunicazioni che può ricevere/inviare. Come detto in precedenza le azioni sono delle operazioni che un agente può decidere in completa autonomia di eseguire o meno. Alcune azioni possono essere di tipo *re-active*, ciò vuol dire che possono essere intraprese solo in seguito al ricevimento di una particolare comunicazione (da notare il termine *possono* in quanto anche queste azioni sono a completa discrezione dell'agente). Le comunicazioni, come detto, sono un'interfaccia fra l'agente e l'ambiente: le comunicazioni in uscita vengono inviate in seguito all'esecuzione di un'azione dell'agente e hanno come scopo l'esecuzione di altre azioni da parte di altri agenti (quest'ultimi riceveranno tali comunicazioni come input) [5].

3. Progettazione del simulatore

Il sistema che si vuole testare mediante questo simulatore è stato proposto per la prima volta dal prof. Stefano Mizzaro [8]. Tale sistema prevede una moltitudine di autori e lettori: i primi pubblicano degli articoli scientifici mentre i secondi leggono e valutano tali pubblicazioni. Scopo di entrambi è di ottenere un buon punteggio di *score*: l'autore mira a pubblicare solo articoli che siano valutati con un buon voto, il lettore mira ad incrementare il proprio punteggio giudicando in maniera corretta gli articoli letti. Proprio quest'ultimo punto, giudicare in maniera *corretta* un paper, è alla base di questo simulatore: si vuole testare quale tipologia di lettore avrà il miglior punteggio al termine della simulazione. Infatti, come risulta dall'articolo in cui viene proposto questo nuovo meccanismo di valutazione delle pubblicazioni scientifiche, per un lettore è possibile incrementare il proprio punteggio in maniera involontaria (potrebbe giudicare in maniera errata un paper che tutti gli altri lettori hanno giudicato in maniera errata falsandone il punteggio) o addirittura fraudolenta (potrebbe accordarsi con altri lettori per giudicare nello stesso modo un paper). La simulazione dovrebbe quindi fornire dei dati riguardo la correttezza delle previsioni teoriche e delle formule da applicare per il calcolo dei punteggi al fine di penalizzare i lettori disonesti e impreparati.

Il primo passo per la progettazione del simulatore è quindi l'individuazione delle classi di agenti coinvolte. Le prime due risultano evidenti: il *Lettore* e l'*Autore*. Il primo verrà modellato da un agente il cui scopo è la valutazione degli articoli scientifici, mentre il secondo avrà lo scopo di produrre nuovi articoli. Ovviamente è necessario un modo per far comunicare queste due classi di agenti e per gestire l'intero processo di valutazione: è stato pertanto introdotto un nuovo agente, denominato *Sistema*, il cui scopo è appunto fare da intermediario fra gli agenti e il sistema di valutazione.

Come risulterà evidente in seguito, gli agenti di tipo autore e lettore non conservano in prima persona i dati relativi al loro *score* e alla loro *steadiness* ma sarà compito del sistema mantenere un database di tutti gli autori e lettori che interagiscono con esso e mantenere aggiornati per ciascuno di essi tali dati. Questa soluzione è stata adottata per due motivi: prima di tutto per parallelismo con quella che sarebbe un'implementazione reale del sistema di valutazione (è impensabile che un lettore umano abbia la responsabilità di mantenere i dati relativi al proprio "valore"), ed in secondo luogo in quanto, per come sono stati ideati gli agenti, non è possibile effettuare delle chiamate "obbligatorie" ad un agente. Infatti, se per esempio fosse compito dell'agente Lettore conservare il proprio *score*, quando il Sistema deve variare il punteggio di tale Lettore dovrebbe comunicargli la variazione, ma il Lettore potrebbe ignorare tale comunicazione e non apportare le modifiche al proprio punteggio. Pertanto sarà appunto il sistema di valutazione a mantenere tali dati che quindi non saranno accessibili dai singoli agenti Lettore e Scrittore.

Infine una precisazione riguardo i parametri degli agenti. Tali parametri hanno tutti un valore compreso tra 0 e 1 e sono associati ad un valore di probabilità. Alcuni parametri sono temporali in quanto specificano delle velocità di esecuzione di una certa azione. Per questo motivo si è pensato di suddividere l'esecuzione in slot temporali e quindi i parametri temporali specificheranno la probabilità che una certa azione sia eseguita in ciascuno slot temporale.

3.1 Autore

Passiamo ora a descrivere più in dettaglio le tre tipologie di agenti. L'agente Autore, che come detto ha lo scopo di produrre nuovi articoli scientifici e pubblicarli inviandoli all'agente Sistema, risulta molto semplice. Qui di seguito è possibile osservare il diagramma di classe rappresentante l'agente Autore:

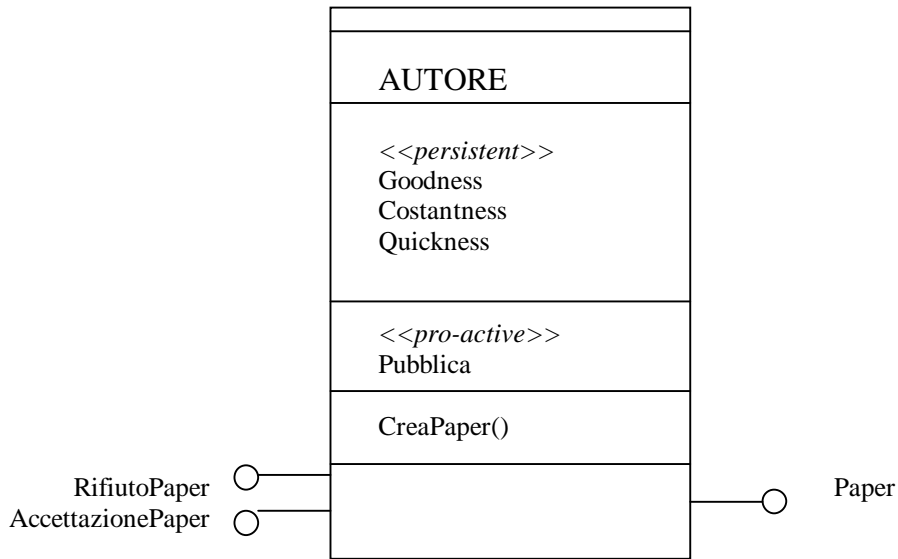


Figura 6: Diagramma di classe dell'agente Autore

Come si può vedere dal diagramma, l'agente Autore presenta tre campi rappresentanti il suo stato interno: *goodness* (media), *costantness* (varianza) e *quickness*. I primi due campi vengono utilizzati per determinare lo score ideale del paper pubblicato da questo autore: questo score ideale viene calcolato mediante una funzione gaussiana di media e varianza specificate e viene mantenuto all'interno della simulazione come attributo del paper (nella realtà potrebbe non esserci accordo relativamente allo score ideale di una pubblicazione scientifica). Questo metodo permette di avere autori bravi (media alta) e meno bravi (media bassa) ma anche autori costanti (varianza bassa) e autori meno costanti (varianza alta). Il terzo campo rappresenta la velocità di produzione di nuovi paper da parte dell'agente. Questi tre campi sono stati marcati come persistenti in quanto qualora il simulatore venga interrotto e poi fatto ripartire è desiderabile che tali valori vengano mantenuti inalterati.

Questo agente può intraprendere un'unica azione: pubblicare un articolo. Tale azione è marcata come *pro-active* in quanto l'Autore pubblica un nuovo paper di sua iniziativa e non dietro richiesta di altri agenti.

Il metodo previsto per questo agente è uno solamente: il metodo necessario per la creazione di un paper. Tale metodo fornirà anche uno score ideale per il paper creato basandosi sui valori di media e varianza dell'Autore in questione.

Infine vediamo le comunicazioni che l'Autore può ricevere ed inviare. Partendo da queste ultime vediamo che l'Autore può inviare solamente la comunicazione relativa alla pubblicazione di un paper (comunicazione denominata *Paper*). Invece può ricevere due diverse comunicazioni: la comunicazione di rifiuto o di accettazione della pubblicazione.

3.2 Lettore

L'agente Lettore, il cui compito consiste nella valutazione degli articoli pubblicati, risulta essere più complesso:

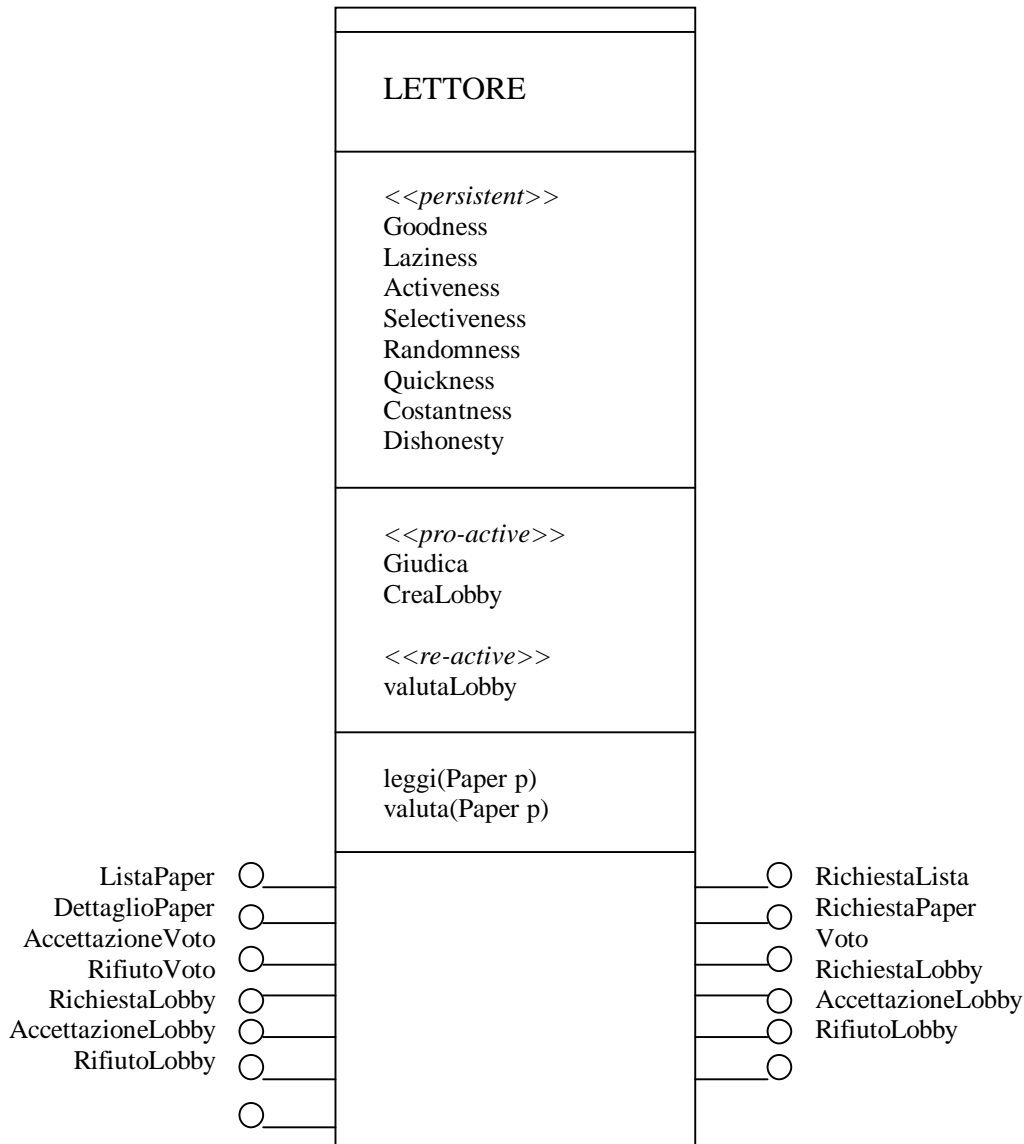


Figura 7: Diagramma di classe dell'agente Lettore

Il Lettore presenta un elevato numero di campi per rappresentare il suo stato interno, tutti persistenti. La *goodness* rappresenta la bontà media dei voti espressi dal lettore relativamente allo score ideale del paper considerato. Tale bontà può essere espressa come varianza di una funzione gaussiana la cui media è data dallo score ideale del paper. La *laziness* (pigrizia) rappresenta la probabilità del Lettore di confermare lo score attuale del paper. L'*activeness* rappresenta il grado di partecipazione del lettore, ovvero la sua propensione a giudicare un elevato numero di paper. La *selectiveness*, invece, fornisce un'idea dell'interesse del lettore: un lettore con alta *selectiveness* tenderà a giudicare solamente paper con alto *score* (punteggio), ovvero paper ritenuti buoni dalla comunità. La *quickness* è un parametro di velocità: nel caso in cui sia elevato significa che il lettore tenderà a giudicare velocemente i nuovi paper. La *dishonesty*, invece, rappresenta la probabilità di un lettore di creare o partecipare a delle lobby. Infine *randomness* e *costantness* rappresentano la probabilità di esprimere un giudizio random o costante rispettivamente.

Anche le azioni a disposizione del Lettore sono più di quelle disponibili per gli autori. Infatti un lettore può di sua iniziativa giudicare un paper o richiedere ad altri lettori la creazione di una lobby (azioni pro-active); inoltre, in risposta ad una richiesta di lobby, può anche decidere o meno di partecipare alla lobby (azione re-active).

I metodi interni del Lettore vengono utilizzati per la lettura di un paper e per il calcolo del suo score in base allo stato interno dell'agente.

Infine vediamo che anche le comunicazioni in entrata e in uscita dal Lettore sono molteplici. In uscita vediamo che il Lettore può fare richiesta di ricevere la lista di tutti i paper disponibili (ristretta magari ai paper non ancora giudicati dal Lettore in questione), il dettaglio di un paper (poiché necessario nella simulazione, riceverà fra gli altri dati anche lo score ideale del paper che servirà per il calcolo del voto), può esprimere un giudizio, può fare richiesta di creazione di una lobby e infine accettare o meno la partecipazione ad una lobby. In ingresso, invece, il lettore può ricevere la lista dei paper disponibili, il dettaglio di un singolo paper, l'accettazione o rifiuto del giudizio espresso relativamente ad un paper, una richiesta di partecipazione ad una lobby e, infine, l'accettazione o rifiuto di partecipare alla lobby.

3.3 Sistema

In ultimo, vediamo come si presenta l'agente Sistema il cui compito è di fare da intermediario fra gli agenti Lettore e Autore ed il sistema di gestione dei paper:

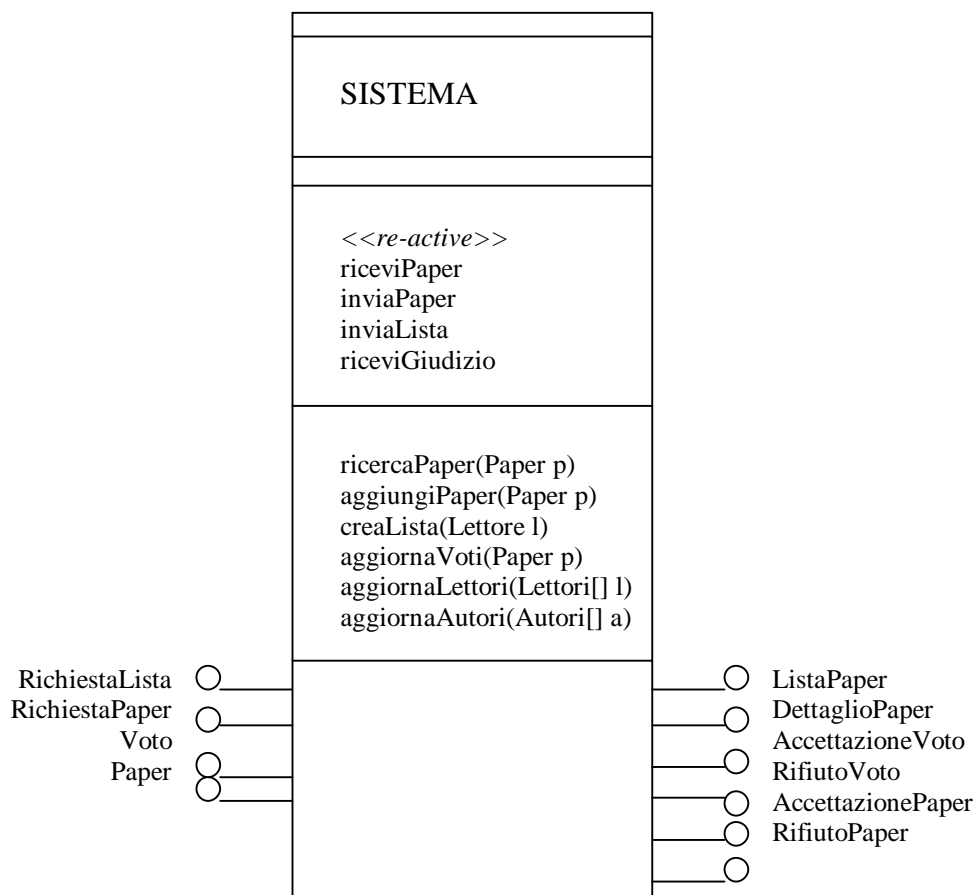


Figura 8: Diagramma di classe dell'agente Sistema

L'agente Sistema, che a differenza delle altre due tipologie di agenti sarà presente con una singola istanza nel simulatore, non presenta alcuno stato interno in quanto il suo unico scopo è di fare da tramite fra gli agenti Lettore e Autore ed il sistema di gestione paper.

Per questo motivo, inoltre, non presenta alcuna azione di tipo pro-active (autonome) ma solamente azioni di tipo re-active (reattive): può ricevere un articolo, inviare un articolo o una lista di articoli e ricevere un giudizio. La prima azione consiste nel pubblicare un nuovo articolo ricevuto da un autore; la seconda e la terza azione consistono nell'inviare ai lettori il dettaglio di un paper o la lista dei paper disponibili rispettivamente; l'ultima azione consiste nel ricevere un giudizio da un lettore relativamente ad un paper e intraprendere tutte le operazioni di aggiornamento *score* e *steadiness* del paper e di tutti i lettori e autori interessati.

I metodi a disposizione dell'agente Sistema consistono in realtà in chiamate a metodi pubblici del sistema di gestione sottostante: la ricerca di un paper, l'aggiunta di un paper alla lista delle pubblicazioni, la creazione della lista dei paper non ancora giudicati relativamente ad un dato lettore, l'aggiornamento dei punteggi di un paper, dei lettori che hanno precedentemente giudicato quel paper e dell'autore del paper in questione.

Le comunicazioni in uscita a disposizione dell'agente Sistema consistono nell'inoltro della lista dei paper e del dettaglio di un paper, nell'accettazione o rifiuto di un giudizio da parte di un lettore e dell'accettazione o rifiuto di un nuovo paper da pubblicare da parte di un autore. In ingresso, invece, il Sistema può ricevere una richiesta della lista dei paper o del dettaglio di un paper, il giudizio relativo ad un determinato paper o un nuovo paper da pubblicare.

3.4 Interazioni fra gli agenti

Ora che è stato descritto come sono state progettate le tre classi di agenti passiamo ad analizzare come queste interagiscono tra loro sfruttando i diagrammi di sequenza con le integrazioni descritte nella sezione precedente. Prima di procedere è però necessario sottolineare un dettaglio solamente accennato in precedenza: nel simulatore verranno istanziate un numero non predeterminato di autori e lettori mentre vi sarà un sola istanza di Sistema. La possibilità di creare un numero non predeterminato di autori e lettori permette di valutare il comportamento dell'intero sistema di valutazione a prescindere dal numero degli attori che vi interagiscono. Inoltre è possibile inserire nella simulazione nuove istanze di autori e lettori o toglierne in qualsiasi istante. Il fatto che vi sia una sola istanza dell'agente Sistema è in realtà una semplificazione il cui superamento non richiede alcuna modifica al progetto del simulatore ma, qualora vi siano più istanze, è necessario che tutti gli agenti Sistema condividano il medesimo database di agenti e lettori.

Analizziamo ora la procedura di pubblicazione di un nuovo articolo da parte di un Autore:

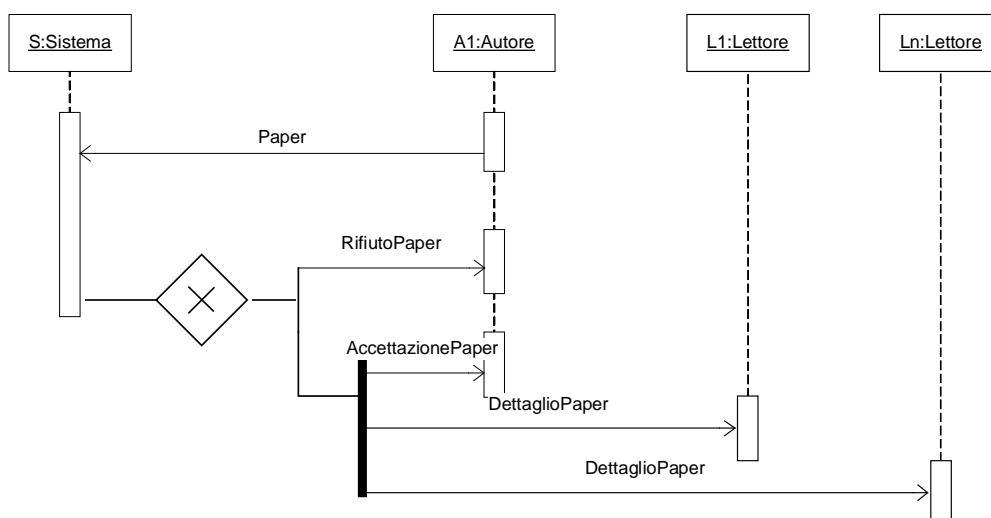


Figura 9: Diagramma di sequenza per la pubblicazione di un paper

Come si può notare dal grafico, l'Autore A1 (una istanza di un autore) inizia la comunicazione inviando un nuovo paper da pubblicare al Sistema S. Il Sistema può decidere se pubblicare tale articolo o meno (un controllo banale potrebbe essere la ricerca nella lista dei paper per verificare che non sia già stato pubblicato). Se il sistema non accetta la pubblicazione avverte l'autore mentre se la accetta dà risposta positiva all'autore ed invia un dettaglio del paper a tutti i lettori possibili. Quest'ultima operazione è stata prevista nel simulatore in quanto i lettori devono essere avvertiti delle nuove pubblicazioni (si pensi al campo *quickness*) ma nel sistema reale non è pensabile che ad ogni nuova pubblicazione vengano avvertiti tutti i lettori possibili.

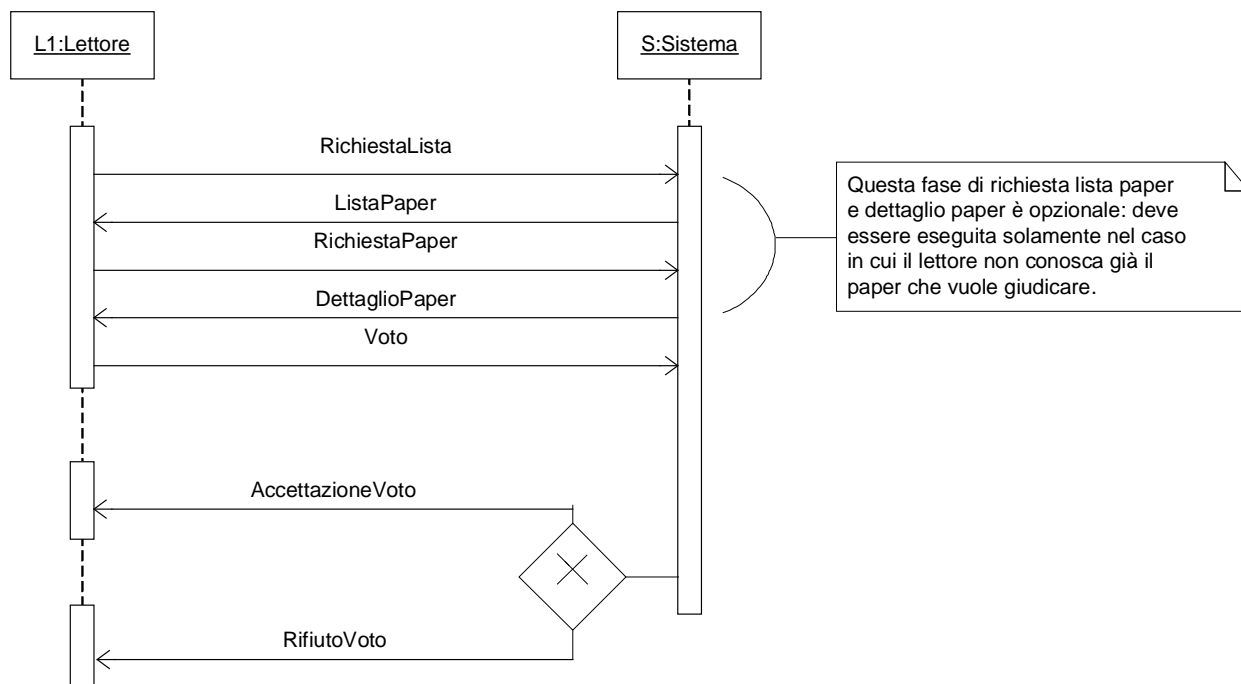


Figura 10: Diagramma di sequenza per il giudizio di un paper

In figura 10 possiamo osservare la procedura che deve seguire un lettore al momento del giudizio di un paper. Innanzitutto deve essere eseguita una fase di ricerca del paper da valutare: se il lettore non conosce in dettaglio il paper deve richiedere una lista al sistema e quindi, una volta deciso quale paper giudicare, richiede il dettaglio di un paper. A questo punto può esprimere il suo giudizio sul paper inviandolo al sistema e riceverà una notifica dal sistema stesso sull'accettazione o rifiuto del giudizio espresso.

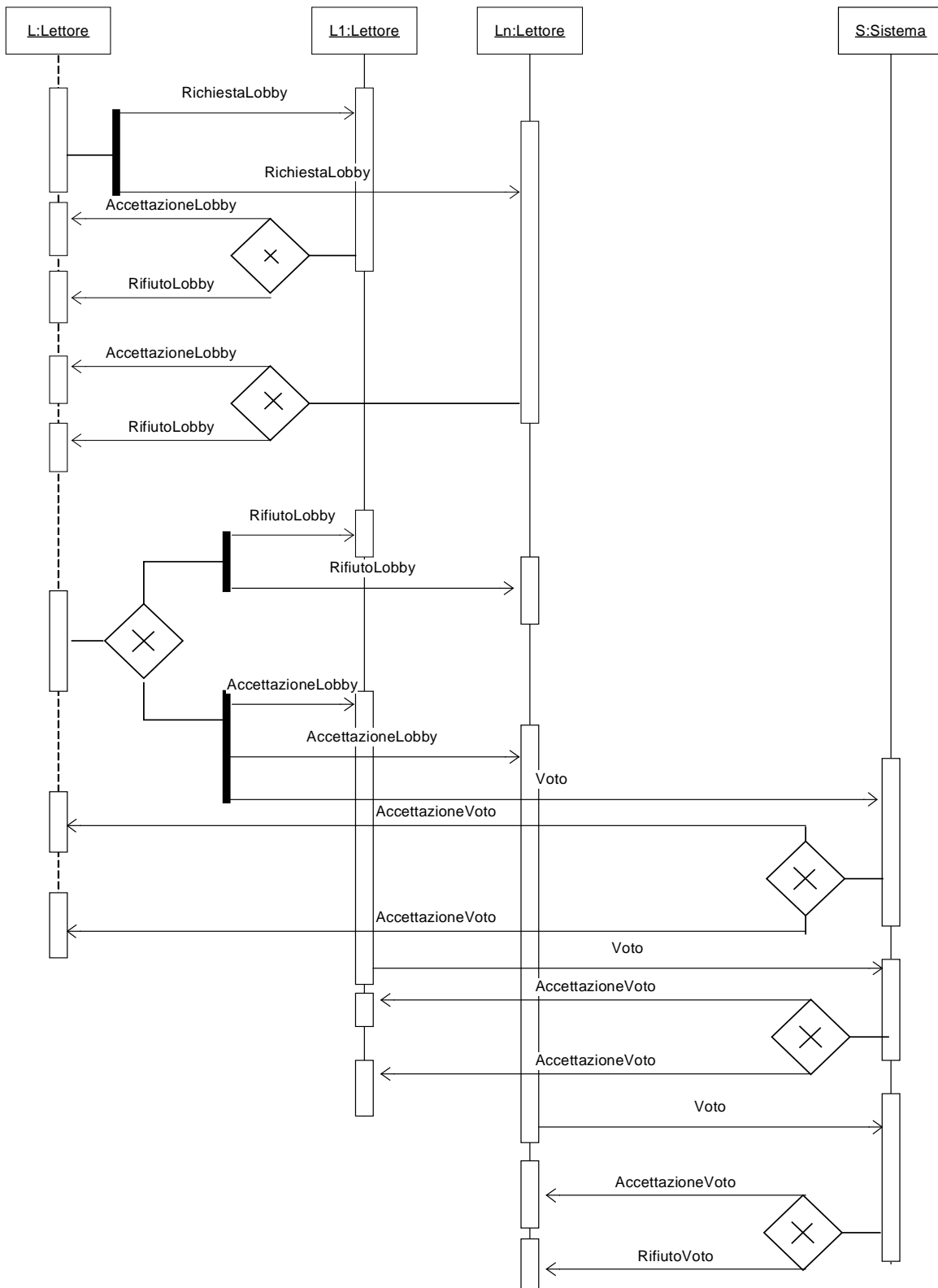


Figura 11: Diagramma di sequenza per la creazione di una lobby ed il giudizio di un paper

In figura 11, infine, è possibile osservare l'interazione fra più lettori ed il Sistema per la creazione di una lobby e conseguente espressione di un giudizio da parte dei partecipanti. La comunicazione verrà iniziata da un lettore (L) il cui scopo è la creazione di una lobby per la votazione di un paper. Tale lettore invia una richiesta di partecipazione alla lobby ad un insieme di lettori (L1...Ln) e ciascuno di questi lettori risponderà accettando o meno tale lobby. A questo

punto il lettore L deciderà se la lobby verrà creata: in caso negativo avvertirà tutti i lettori precedentemente contattati mentre, in caso positivo, darà conferma ai lettori che hanno dato la loro disponibilità e ogni lettore appartenente alla lobby procederà con la votazione del paper.

3.5 Il simulatore

Nella figura che segue (figura 12) viene presentato il diagramma di classe del simulatore completo.

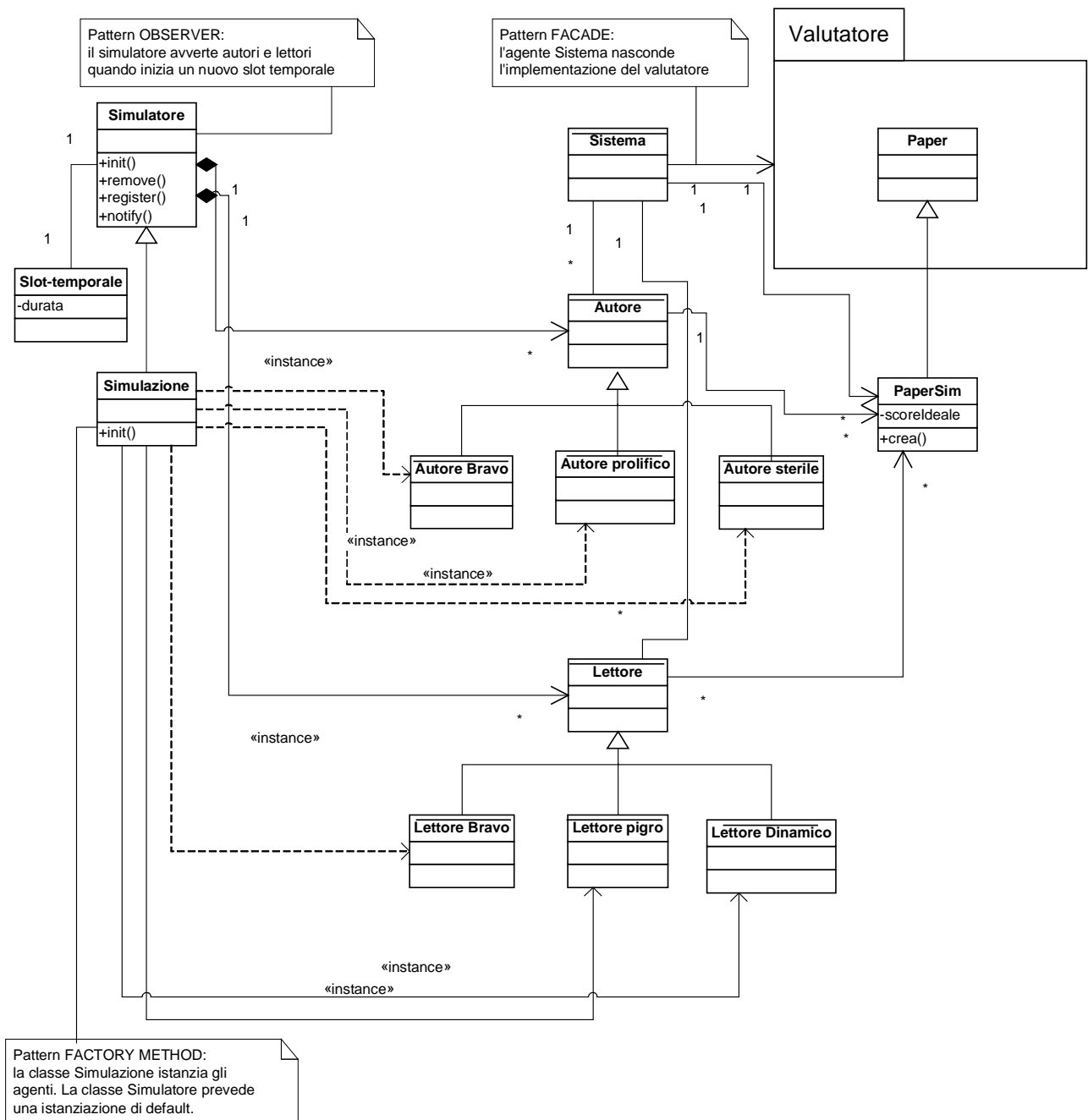


Figura 12: Diagramma di classe del simulatore completo

Come si può osservare dal diagramma di figura 12, sono state introdotte quattro nuove classi. Innanzitutto è stata aggiunta la classe *PaperSim*, la classe che descrive il paper per il simulatore estendendo la classe *Paper* del valutatore con l'attributo *scoreIdeale* ed il metodo *crea()* che tiene conto di questo nuovo attributo. Tutti gli agenti faranno riferimento a questa classe come loro Paper.

Per quanto riguarda la gestione della simulazione è stata introdotta la classe *Simulatore*. Tale classe ha il compito di istanziare (*init()*) in modo random tutti gli agenti e di avvertirli all'inizio di ogni nuovo slot temporale. Per permettere, però, a chiunque di iniziare una simulazione con l'insieme di agenti voluto, è stata introdotta la classe *Simulazione* che avrà il compito di istanziare gli agenti specificati. Per quanto riguarda, infine, la gestione degli slot temporali è stata introdotta la classe *Slot-temporale* con attributo *durata* che specifica appunto la durata di uno slot temporale.

Risulta evidente anche dalla figura l'utilizzo di tre pattern: il pattern *Façade*, il pattern *Observer* ed il pattern *Factory Method*. Il primo pattern è stato introdotto per nascondere l'implementazione del sistema di valutazione al simulatore ed è stato dato quindi l'incarico di interfacciarsi con tale sistema al solo agente Sistema. Il pattern Observer è stato introdotto per la gestione degli slot temporali: poiché ogni agente deve essere a conoscenza dell'inizio di un nuovo slot temporale è necessario che questo venga avvertito e il pattern observer permette di evitare che ciascun agente gestisca in modo autonomo un proprio timer o, peggio ancora, che sia l'agente a dover chiedere continuamente se è iniziato un nuovo slot. Infine, il pattern Factory Method è stato introdotto per permettere a chiunque di iniziare una simulazione con l'insieme preferito di agenti. In questo modo è sufficiente scrivere una propria classe *Simulazione* che estende la classe *Simulatore* e sovrascrive il metodo *init()*. La classe *Simulatore* fornirà comunque un'implementazione di default che inizierà una nuova simulazione con un insieme di agenti random.

4. Configurazioni possibili degli agenti

Durante la progettazione degli agenti Lettore ed Autore sono stati previsti dei campi per la rappresentazione dello stato interno di ciascun agente. In questa ultima parte verranno quindi descritte alcune delle possibili configurazioni di tali campi per la creazione di agenti con particolari comportamenti.

4.1 Tipologie di autori

Per gli agenti Autore è possibile specificare una *goodness*, una *constantness* ed una *quickness*. Come già descritto in precedenza, il calcolo dello score ideale per un paper pubblicato da un Autore viene ottenuto come risultato di una funzione random gaussiana con media la *goodness* e varianza la *constantness*. Questo permette di creare autori *bravi* (*goodness* alta) e autori *non bravi* (*goodness* bassa); per entrambi è possibile definire la loro costanza nel produrre paper con score ideale vicino alla loro *goodness* modificando il valore di *constantness*: un valore alto di *constantness* porta ad un autore molto irregolare mentre un valore basso porta ad un autore molto costante.

La *quickness* è un parametro che rappresenta la velocità con cui l'Autore pubblica nuovi articoli: se il valore è molto alto (Autore *prolifico*) allora l'Autore ha un'alta probabilità di pubblicare un nuovo articolo nello slot temporale attuale, altrimenti (Autore *sterile*) ha una bassa probabilità di farlo. In questo modo è possibile configurare autori molto produttivi ed altri meno.

Al termine della simulazione, lo score di un Autore dovrebbe essere il più prossimo possibile alla sua *goodness*. Qualora il valore di score ottenuto si discosti dal valore atteso e i paper pubblicati siano pochi (bassa *quickness*), è necessario valutare la *constantness*: se questa è alta allora è possibile che i lettori siano stati tratti in inganno da paper di valore molto diverso, se invece è bassa allora la simulazione ha avuto esito negativo in quanto i lettori non sono riusciti a dare il giusto giudizio per i paper da lui pubblicati. Vedremo comunque che è molto probabile che si ottenga un esito negativo a causa delle possibili combinazioni dei lettori, non tutti capaci di giudicare correttamente un paper.

4.2 Tipologie di lettori

Per i lettori il discorso è più complicato in quanto i parametri disponibili sono molto più numerosi. Possiamo però suddividerli in tre diversi insiemi: uno relativo al voto da esprimere (*goodness*, *laziness*, *randomness*, *constantness*), uno relativo a quando e quali paper giudicare (*activeness*, *selectiveness*, *quickness*) ed uno relativo alla creazione delle lobby (*dishonesty*).

Iniziamo dai quattro parametri relativi al voto da esprimere. Tutti i parametri hanno un valore compreso tra 0 e 1 ma sono in competizione tra loro: ed esempio se un Lettore ha *laziness* e *randomness* entrambe pari a 1 dovrebbe confermare lo score attuale del paper e dargli un giudizio random nello stesso momento, situazione ovviamente indesiderabile. Per tale motivo è necessario che la somma dei valori dei quattro parametri sia uguale ad 1. È necessario però prestare attenzione al parametro *goodness* in quanto in precedenza è stato definito come la varianza di una funzione random gaussiana con media pari allo score ideale del paper per il calcolo del giudizio che il Lettore fornirà per il paper. Pertanto tale parametro non deve entrare nella sommatoria ma deve essere rispettata tale equazione:

$$1 - (\text{laziness} + \text{randomness} + \text{constantness}) = \text{probabilità di calcolo del giudizio mediante funzione gaussiana}$$

In questo modo è possibile definire quattro diverse tipologie di lettori: il Lettore *passivo*, ovvero il Lettore che conferma lo score attuale del paper, avrà una alta laziness; il Lettore *random*, ovvero il Lettore che esprime un giudizio random e quindi inaffidabile, avrà una alta randomness; il Lettore *costante*, ovvero il Lettore che esprime sempre lo stesso giudizio indipendentemente dal paper valutato, avrà alta costantness; infine il Lettore *attento*, ovvero il Lettore il cui giudizio viene calcolato secondo il proprio gradimento (e in base al procedimento sopra descritto), avrà bassi valori per le tre proprietà laziness, randomness e costantness. Inoltre, per i lettori attenti, è possibile definire due ulteriori tipologie: il Lettore *bravo*, ovvero il lettore che esprime giudizi vicini allo score ideale del paper, avrà un basso valore di goodness ed il Lettore *cattivo*, ovvero il Lettore che esprime giudizi molto diversi dal punteggio di score ideale del paper, avrà un alto valore di goodness.

Per i parametri contenuti nel secondo degli insiemi individuati il discorso è più facile in quanto questi non contrastano tra loro come avviene per i parametri precedenti. Possiamo quindi definire delle ulteriori categorie. Se il Lettore ha un'elevata activeness allora è un Lettore *vorace*, in quanto è alta la probabilità che in ogni slot temporale valuti un paper, altrimenti è un Lettore *pigro*, in quanto è bassa la probabilità che nello slot temporale attuale valuti un paper. Se il Lettore ha un'elevata steadiness allora è un Lettore *snob*, in quanto tende a giudicare solamente paper con elevato score, altrimenti è un Lettore *onnivoro*, in quanto si abbassa a leggere e valutare anche dei paper con basso punteggio. Se infine il Lettore ha una elevata quickness allora è un Lettore *dinamico*, in quanto è alta la probabilità che giudichi un paper nello stesso intervallo temporale in cui lo riceve, altrimenti è un Lettore *statico*, in quanto è alta la probabilità che non sia il primo a giudicare un paper.

Le tipologie di lettori descritte considerando i parametri appartenenti al secondo insieme individuato, non devono andare a sommarsi con quelle precedentemente introdotte in quanto non contrastano in alcun modo con esse ma, al contrario, possono essere combinate tra loro. Infatti è possibile definire una configurazione di Lettore tale da renderlo, per esempio, bravo ma pigro oppure costante e vorace. Inoltre anche le ultime tipologie definite possono essere combinate tra loro: è possibile definire, per esempio, un Lettore pigro e prudente o vorace ma costante. Ovviamente esistono delle combinazioni possibili che però risultano essere in contrasto: ad esempio un Lettore potrebbe essere dinamico ma prudente.

Passando all'ultimo dei parametri relativi ai lettori possiamo distinguere due ulteriori tipologie che potranno essere combinate con le precedenti in maniera completamente libera: il Lettore *onesto*, ovvero il Lettore che avrà una bassa probabilità di aderire o creare delle lobby, ed il Lettore *disonesto*, ovvero il Lettore che cercherà di migliorare il proprio score ricorrendo a votazioni "truccate".

Risulta quindi evidente come le tipologie di lettori siano molteplici e come quindi sia possibile creare delle configurazioni che rispondano alle più diverse esigenze. Al termine della simulazione lo score di ciascun lettore considerato bravo dovrebbe essere il più possibile prossimo ad 1 mentre lo score dei lettori considerati incapaci deve tendere a 0.

5. Analisi basata su BDI

Per la progettazione degli agenti Autore, Lettore e Sistema è possibile ricorrere anche ad un'analisi basata su *belief-desire-intention* (BDI), un'architettura utilizzata per esprimere i processi decisionali. I processi decisionali includono due fondamentali step: decidere *quali* goal si vuole inseguire e quindi *come* perseguirli. Tali processi si basano su tre insiemi: un insieme di credenze e convinzioni del mondo da parte dell'agente (*beliefs*), un insieme di desideri e scopi da raggiungere (*desires*) e un insieme d'intenzioni e azioni da intraprendere per raggiungere i propri obiettivi (*intentions*). Inoltre è necessario anche prevedere quattro diverse funzioni che valutino gli elementi di questi tre insiemi, li aggiornino e determinino le azioni da compiere. La prima è una funzione, denominata *belief revision function*, che, valutando le credenze attuali dell'agente e gli input che riceve dall'ambiente, genera un nuovo insieme di credenze ed è necessaria affinché l'agente si evolva con l'ambiente che lo circonda. La seconda funzione, denominata *option generation function*, partendo dalle credenze e dalle precedenti intenzioni dell'agente genera un nuovo insieme di desideri. Tale funzione è necessaria per permettere all'agente di modificare i propri obiettivi qualora i precedenti non siano più perseguibili, interessanti o siano già stati raggiunti. La terza funzione, denominata *filter*, valutando tutti gli elementi dei tre insiemi, determina un nuovo insieme d'intenzioni dell'agente e rappresenta la vera e propria funzione di deliberazione dell'agente. Infine, la quarta funzione, denominata *action selection function*, determina un'azione da intraprendere partendo dall'insieme corrente d'intenzioni [4].

Per rivedere la progettazione degli agenti proposti in questa sede sulla base dell'architettura BDI è quindi necessario definire per ciascuno di essi i tre insiemi e le quattro funzioni sopra descritte. A puro titolo d'esempio, e per dimostrare che è possibile applicare tale architettura anche al simulatore progettato, verrà qui di seguito analizzato l'agente lettore.

Innanzitutto è necessario descrivere l'insieme *belief*. Tale insieme, come detto, conterrà tutte le credenze e convinzioni che l'agente possiede dell'ambiente che lo circonda. Nel capitolo precedente sono state descritte diverse tipologie di lettori, basate sui valori assunti dai parametri previsti: ciascuna tipologia di lettore avrà un proprio insieme *belief* diverso dagli altri. Se per esempio consideriamo il lettore passivo, questo sarà convinto che il modo migliore per aumentare il proprio score sia di confermare il voto attuale del paper che sta esaminando. Il lettore bravo ma pigro avrà delle credenze tali da permettergli di valutare correttamente un paper (da cui bravo) e altre credenze per cui ritiene che valutare solo pochi paper sia una buona strategia per incrementare il proprio score. In quest'ultimo caso non sono state specificate le credenze che rendono bravo un Lettore: sarebbe stato necessario prevedere un insieme di credenze, paragonabili ad una cultura scientifica, tali da permettere al Lettore di giudicare un paper in base alle proprie conoscenze. Tale strada, sebbene interessante, non verrà ulteriormente esplorata ma dovrebbe essere chiaro che è possibile definire lettori bravi e lettori meno bravi attraverso un insieme di credenze dell'ambiente in cui sono calati e definire delle funzioni di valutazione dei paper basate sulle conoscenze del singolo Lettore.

In secondo luogo è necessario prevedere un insieme di desideri, l'insieme *desire*. Tale insieme è popolato, in questo caso, da un unico desiderio: il Lettore ha come unico obiettivo l'aumento del proprio score.

Il terzo insieme, l'insieme *intention*, sarà inizialmente vuoto in quanto verrà popolato, e costantemente aggiornato, dalla funzione *filter* (definita in seguito).

Passando alle funzioni, per gli scopi di questo simulatore non è necessario prevedere la funzione *belief revision function* d'aggiornamento delle credenze. Infatti si vuole valutare come tipologie diverse di lettori interagiscono con il sistema per valutare se esistono strategie indesiderate

per incrementare lo score del lettore. Per future e più complesse simulazioni si potrebbe definire questa funzione al fine di permettere all'agente di migliorarsi.

Anche la seconda funzione, denominata *option generation function*, non deve essere definita in questo progetto in quanto si vuole che i lettori abbiano sempre e solo lo scopo di incrementare il proprio score.

La terza funzione, la funzione *filter*, è, come detto, la funzione principale in quanto valutando credenze, desideri ed intenzioni precedenti, aggiorna l'insieme delle intenzioni. Questa funzione risulta essere molto complessa e pertanto verranno di seguito proposti solamente degli esempi. Consideriamo un lettore costante, pigro e disonesto. Questa funzione dovrebbe quindi essere in grado di definire un insieme d'intenzioni come il seguente (x rappresenta un giudizio costante):

{attribuisci ad ogni paper letto il voto x, valuta pochi paper, convinci anche altri lettori a giudicare con il voto x il paper da te valutato }

In questo caso risulta abbastanza semplice la definizione di tale funzione in quanto i desideri dei lettori non cambiano nel tempo e sono uno solamente; inoltre le intenzioni precedenti del Lettore non influenzano in alcun modo le intenzioni correnti. Quindi è necessario considerare solamente le credenze dei lettori, semplificando di molto la funzione *filter*.

Infine è necessario definire la funzione *action selection function* che, dall'insieme corrente d'intenzioni, determina quale azione deve intraprendere il Lettore. Considerando l'insieme d'intenzioni dell'esempio precedente, questa funzione potrebbe fornire tre diversi output corretti: giudicare un paper con voto x, non fare nulla oppure creare una lobby per giudicare un paper con voto x. La scelta fra i tre possibili output potrebbe essere random oppure predefinita. Per gli scopi di questo simulatore ha senso che la scelta avvenga in modo random al fine di preservare tutte le caratteristiche del Lettore: se infatti scegliesse sempre di giudicare un paper con voto x, verrebbero perse le caratteristiche di pigrizia e disonestà del Lettore.

In questo capitolo è stata fatta solamente una presentazione dell'architettura BDI e della sua possibile utilizzazione per la progettazione dei tre agenti proposti. Una trattazione più completa deve prevedere un modo più concreto per definire gli elementi degli insiemi (magari sfruttando formule logiche del prim'ordine) e ovviamente una definizione esaustiva delle funzioni introdotte.

6. Conclusioni

Il sistema progettato nell'ambito di questo laboratorio non è sicuramente sufficiente per l'implementazione del simulatore descritto. Innanzitutto è necessaria un'implementazione del sistema di valutazione descritto nell'articolo del prof. Stefano Mizzaro a cui collegare l'agente Sistema. Inoltre bisogna progettare anche il metodo di scambio dei messaggi (CA), magari basandosi sul protocollo FIPA, e l'ambiente entro cui operano gli agenti (ad esempio andrebbe specificato meglio il concetto di slot temporale).

Un'ulteriore affinamento degli agenti di tipo lettore potrebbe essere l'introduzione di tecniche di apprendimento per migliorare le capacità dei singoli lettori e valutare come lo score di ciascun lettore rispecchia questi miglioramenti.

In conclusione vorrei presentare una mia idea per valutare la "strategia vincente" per un lettore. Come è stato più volte ripetuto in questo elaborato, lo scopo di un lettore è di giudicare i paper in modo da aumentare il proprio score. Tale obiettivo dovrebbe essere perseguibile seguendo un'unica strada: valutare correttamente tutti i paper. Si è però visto che un lettore potrebbe cercare di raggiungere il proprio obiettivo in altri modi: creando lobby, confermando il voto attuale del paper, ..., sempre nel tentativo di incrementare il proprio score. Per questo motivo ritengo sia possibile sfruttare gli algoritmi genetici per far evolvere una popolazione di lettori alla ricerca del lettore perfetto, ovvero della strategia migliore di giudicare articoli scientifici incrementando il proprio score. La mia idea consiste nel creare una popolazione random di lettori a cui dare un insieme di articoli da valutare secondo un determinato ordine; dopo un certo intervallo temporale prefissato, la simulazione viene interrotta e, partendo da un insieme di lettori scelto da una funzione di selezione basata sullo score (*fitness*) di ciascun lettore, viene effettuato un *crossover* fra i lettori selezionati e viene creata così una nuova generazione di lettori; a questo punto viene fatta ripartire la simulazione con i nuovi lettori a cui verranno forniti gli stessi paper e nello stesso ordine precedente. Dopo un certo numero di generazioni (non noto a priori), si dovrebbe notare l'emergenza di un insieme di lettori ideali, ovvero quei lettori che hanno lo score maggiore. In questo modo si dovrebbe poter avere una conferma delle previsioni teoriche (il lettore ideale è il lettore che valuta correttamente i paper) oppure si noterebbe un problema (la strategia adottata dal lettore ideale non è quella voluta) a cui porre rimedio.

Bibliografia

- [1] AUML. <http://www.auml.org>
- [2] Bernhard Bauer. UML Class Diagrams Revisited in the Context of Agent-Based Systems.
- [3] Bernhard Bauer, Jörg P. Müller, James Odell. Agent UML: A Formalism for Specifying Multiagent Interaction
- [4] Gerhard Weiss. Multiagent Systems. The MIT Press, 1999
- [5] M. Wooldridge. Multiagent Systems.
- [6] Jacques Ferber. Multi-Agent Systems, An Introduction to Distributed Artificial Intelligence. Addison – Wesley, 1999
- [7] James Odell, H. Van Dyke Parunak, Bernhard Bauer. Extending UML for Agents.
- [8] Stefano Mizzaro (2003). Quality Control in Scholarly Publishing: A New Proposal.