Abstract

L'obiettivo del lavoro di tesi è quello di realizzare un linguaggio che permetta di implementare il modello ad agenti BDI su ambienti e piattaforme differenti, ovvero dove l'ambiente di esecuzione può essere sia quello simulato che quello reale (ad esempio la JVM). Si vuole quindi realizzare un ambiente di lavoro che permette di eseguire il modello ad agenti BDI, definito da AgentSpeak e implementato in Jason, attraverso l'utilizzo del meta-simulatore Alchemist, definendo la relativa incarnazione, e sfruttando tuProlog sia per la gestione del linguaggio che per quella dello stato e dei piani dell'agente.

L'obiettivo di questo lavoro preparatorio è quello di prendere famigliarità con i due ambienti di sviluppo, Alchemist e tuProlog, al fine padroneggiarne l'utilizzo per la realizzazione di un interprete del linguaggio ad agenti che sarà definito. Inoltre, si vuole approfondire il modello di agenti BDI (Beliefs-Desires-Intentions) definito da AgentSpeak studiandone la struttura e le implementazioni già note per estrapolarne le parti importanti, come ad esempio il ciclo di ragionamento definito in Jason.

Sono stati quindi prima esaminati e approfonditi i due ambienti singolarmente e successivamente sono stati analizzati i punti di contatto, sia fra i modelli (agenti e meta-modello di Alchemist) che tra gli ambienti (Alchemist e tuProlog), sia per capire come creare la futura integrazione che per avere la cognizione dell'intero scenario si sarebbe affrontato con il progetto di tesi.

Indice

Elenco delle figure

Capitolo 1

Introduzione

In questo lavoro preparatorio vengono descritti i modelli e gli strumenti che saranno poi utilizzati nel progetto di tesi (Alchemist, tuProlog, modello agenti BDI AgentSpeak) mostrando come sono composti e quali sono i loro punti salienti. Successivamente viene descritto il lavoro che è stato fatto su ogni ambiente per conoscere sia Alchemist che tuProlog, in modo da ottenere informazioni utili per impostare in modo corretto il progetto di tesi.

1.1 Obiettivo della tesi

L'obiettivo del lavoro di tesi è quello utilizzare la definizione di agenti BDI fatta da AgentSpeak per definire un nuovo linguaggio ad agenti al quale, inoltre, si è voluto aggiungere anche una caratteristica di flessibilità. Quest'ultima è stata raggiunta grazie all'utilizzo della libreria tuProlog che ha permesso di definire un linguaggio che possa essere utilizzato da interpreti realizzati su ambienti e piattaforme differenti accomunate dall'utilizzo di questa libreria. Si vuole mostrare, inoltre, come è possibile creare un interprete all'interno del meta-simulatore Alchemist, realizzando un'opportuna incarnazione, che sfrutta il modello di agenti BDI definito da AgentSpeak e l'implementazione di Jason, in particolare relativamente al ciclo di ragionamento,

e che permette l'esecuzione di agenti, definiti tramite le teorie utilizzando il nuovo linguaggio, in un ambiente simulato.

1.2 Obiettivo attività propedeutica

L'attività descritta in questo documento ha quindi l'obiettivo di analizzare e studiare Alchemist, tuProlog e il modello ad agenti BDI di AgentSpeak recependo le informazioni necessarie per permettere di impostare il lavoro di tesi in maniera più efficace.

Per quanto riguarda Alchemist l'obiettivo è quello di imparare ad utilizzarlo, studiando: la struttura del suo meta-modello; come scrivere la configurazione della simulazione; come implementare un'incarnazione.

Anche con tuProlog l'obiettivo è di studiare le sue funzionalità e il suo utilizzo, anche attraverso la comprensione di Prolog. Il processo di acquisizione di informazioni, descritto nel documento, racconta dello studio iniziato su Prolog, per comprendere le nozioni di base, e poi proseguito sulla libreria tuProlog, attraverso il manuale.

In questa attività si è inoltre analizzato il modello ad agenti BDI proposto da AgentSpeak, sia per quanto riguarda la composizione che per il ciclo di ragionamento; per quest'ultimo è stato preso come esempio il processo definito in Jason, una delle implementazioni già note di AgentSpeak. In particolare, si è dedicato tempo all'approfondimento del ciclo di ragionamento per captarne l'utilizzo degli elementi e poter successivamente, per il progetto di tesi, riuscire a strutturare opportunamente l'interprete realizzato con Alchemist.

Si è così potuto fare un raffronto tra il modello ad agenti e il meta-modello di Alchemist, cercando i possibili modi di effettuare l'integrazione tra gli agenti e Alchemist, utilizzando come collante tuProlog.

Capitolo 2

Alchemist

Per la realizzazione dell'ambiente per il progetto di tesi è stato scelto il simulatore Alchemist, poichè fornisce già una struttura ed un meta-modello solido e utilizzabile. Nel capitolo viene mostrato Alchemist attraverso le funzionalità e ciò che lo compone. Successivamente è presentata una serie di peculiarità di questo strumento che saranno approfondite più avanti nel documento.

2.1 Descrizione Alchemist

Alchemist è un simulatore per il calcolo pervasivo, aggregato e ispirato alla natura. Esso fornisce un ambiente di simulazione sul quale è possibile sviluppare nuove incarnazioni, cioè nuove definizioni di modelli implementati su di esso. Ad oggi sono disponibili le funzionalità per:

- simulare un ambiente bidimensionale;
- simulare mappe del mondo reale, con supporto alla navigazione e importazione di tracciati in formato gpx;
- simulare ambienti indoor importando immagini in bianco e nero;
- eseguire simulazioni biologiche utilizzando reazioni in stile chimico;

- eseguire programmi Protelis, Scafi, SAPERE (scritti in un linguaggio basato su tuple come LINDA).

2.1.1 Meta-modello

Il meta-modello di Alchemist può essere compreso osservando la figura 2.1.

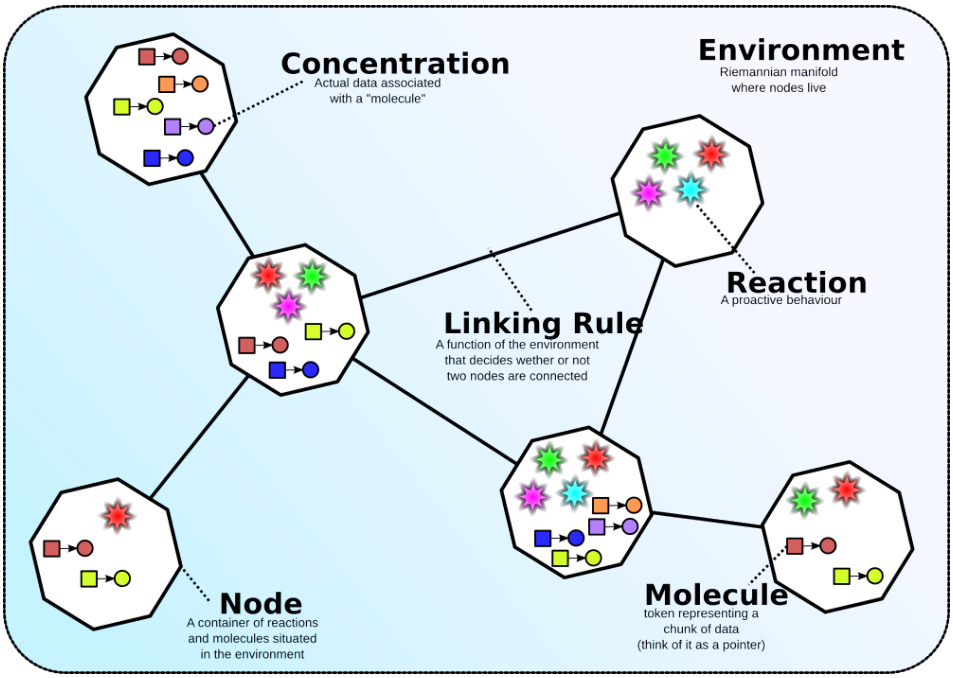

Figura 2.1: Illustrazione meta-modello di Alchemist

L'***Environment*** è l'astrazione dello spazio ed è anche l'entità più esterna che funge da contenitore per i nodi. Conosce la posizione di ogni nodo nello spazio ed è quindi in grado di fornire la distanza tra due di essi e ne permette inoltre lo spostamento.

È detta ***Linking rule*** una funzione dello stato corrente dell'environemnt che associa ad ogni nodo un ***Vicinato***, il quale è un entità composta da un nodo centrale e da un set di nodi vicini.

Un ***Nodo*** è un contenitore di molecole e reazioni che è posizionato all'interno di un environment.

La ***Molecola*** è il nome di un dato, paragonabile a quello che rappresenta il nome di una variabile per i linguaggi imperativi. Il valore da associare ad una molecola è detto ***Concentrazione***.

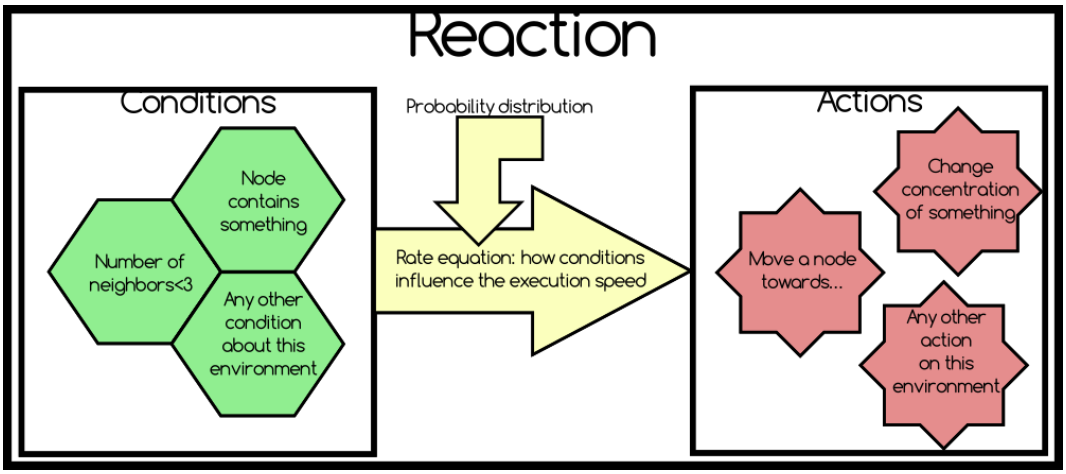

Figura 2.2: Illustrazione modello reazione di Alchemist

Una ***Reazione*** è un qualsiasi evento che può cambiare lo stato dell'environment ed è definita tramite una distribuzione temporale, una lista di condizioni e una o più azioni.
La frequenza con cui avvengono dipende da:

- un parametro statico di frequenza;
- il valore di ogni condizione;
- un'equazione di frequenza che combina il parametro statico e il valore delle condizioni restituendo la frequenza istantanea;
- una distribuzione temporale.

Ogni nodo contiene un set di reazioni che può essere anche vuoto.

Per comprendere meglio il meccanismo di una reazione si può osservare la figura 2.2.

Una ***Condizione*** è una funzione che prende come input l'environment corrente e restituisce come output un booleano e un numero. Se la condizione non si verifica, le azioni associate a quella reazione non saranno eseguite. In relazione a parametri di configurazione e alla distribuzione temporale, una condizione potrebbe influire sulla velocità della reazione.

La ***Distribuzione temporale*** indica il numero di eventi, in un dato intervallo di tempo, generati da Alchemist e che innescano la verifica delle condizioni che possono portare alla potenziale esecuzione delle azioni.

Un'***Azione*** è la definizione di una serie di operazioni che modellano un cambiamento nel nodo o nell'environment.

In Alchemist un'incarnazione è un'istanza concreta del meta-modello appena descritta e che implementa una serie di componenti base come: la definizione di una molecola e del tipo di dati della concentrazione, un set di condizioni, le azioni e le reazioni. Incarnazioni diverse possono modellare universi completamente differenti.

2.2 Aspetti principali in Alchemist

Alchemist è uno strumento molto esteso e che può offrire tantissime possibilità se lo si conosce e si è in grado di padroneggiarlo. La conoscenza iniziale di questo ambiente era però praticamente nulla e, quindi è stato necessario impiegare del tempo per riuscire a padroneggiare i meccanismi di base, tra cui l'utilizzo delle classi delle entità del meta-modello e la scrittura della configurazione di una simulazione.

Gli aspetti principali di Alchemist sono la forte adattabilità del meta-modello sul quale è costrutito, il numero di implementazioni o astrazioni base già disponibili e la grande personalizzazione delle simulazioni.
Il meta-modello fornisce una struttura molto solida sulla quale è possibile realizzare ambiti applicativi anche molto diversi tra loro.

Inoltre, Alchemist, come citato poco fa, fornisce già una serie di implementazioni o astrazioni di classi relative a entità del meta-modello che permettono di avviare lo sviluppo di un'incarnazione in modo molto più rapido.
Per quanto riguarda le simulazioni, queste sono realizzate tramite una configurazione che consiste in una mappa definita tramite il linguaggio YAML. La mappa è composta da diverse sezioni, ognuna caratterizzata da una specifica keyword, ed è altamenta configurabile: questo permette all'utente di testare tantissimi aspetti della simulazione.

Capitolo 3

tuProlog

tuProlog è stato scelto per essere utilizzato nel progetto di tesi per via della sua capacità di operare su ambienti differenti. Grazie a questo strumento e alla sua flessibilità, sarà possibile integrare le due parti, l'interprete e il linguaggio, su uno dei possibili ambienti di sviluppo, ovvero Alchemist. Infatti, proprio grazie al fatto che la libreria tuProlog supporta la programmazione multi-paradigma, è possibile utilizzare vari ambienti per la realizzazione di un'interprete che sfrutti il linguaggio che si andrà a definire nel lavoro di tesi.

3.1 Descrizione tuProlog

tuProlog è un interprete Prolog per le applicazioni e le infrastrutture Internet basato su Java. È progettato per essere facilmente utilizzabile, leggero, configurabile dinamicamente, direttamente integrato in Java e facilmente interoperabile. tuProlog è sviluppato e mantenuto da 'aliCE' un gruppo di ricerca dell'Alma Mater Studiorum - Università di Bologna, sede di Cesena. È un software Open Source e rilasciato sotto licenza LGPL.

3.1.1 Caratteristiche tuProlog

tuProlog è un sistema per applicazioni e infrastrutture distribuite ed è intenzionalmente strutturato con un core minimale, per essere configura-

to sia staticamente che dinamicamente tramite l'utilizzo di librerie e predicati [Sit. 1]. Inoltre, tuProlog supporta nativamente la programmazione multi-paradigma, fornendo un'integrazione chiara ed efficiente tra Prolog e i principali linguaggi orientati agli oggetti, ad esempio Java.

tuProlog ha diverse caratteristiche e qui di seguito verranno illustrate solo alcune di esse, ovvero quelle utilizzate all'interno di questo lavoro. Il motore tuProlog fornisce e riconosce i seguenti tipi di predicati:

- predicati built-in: incapsulati nel motore tuProlog;

- predicati di libreria: inseriti in una libreria che viene caricata nel motore tuProlog. La libreria può essere liberamente aggiunta all'inizio o rimossa dinamicamente durante l'esecuzione. I predicati della libreria possono essere sovrascritti da quelli della teoria. Per rimuovere un singolo predicato dal motore è necesssario rimuovere tutta la libreria che contiene quel predicato;

- predicati della teoria: inseriti in una teoria che viene caricata nel motore tuProlog. Le teorie tuProlog sono semplicemente collezioni di clausole Prolog. Le teorie possono essere liberamente aggiunte all'inizio o rimosse dinamicamente durante l'esecuzione.

Prolog

Prolog è un linguaggio di programmazione che adotta il paradigma della programmazione logica. È impiegato in molti programmi di intelligenza artificiale.

Prolog si basa sul calcolo dei predicati (più precisamente di quelli del primo ordine). L'esecuzione di un programma Prolog è comparabile alla dimostrazione di un teorema mediante regola di inferenza. I concetti fondamentali sono l'unificazione, la ricorsione in coda e il backtracking, o monitoraggio a ritroso.

3.2 Aspetti principali in tuProlog

tuProlog, come detto precedentemente, è un interprete semplificato di Prolog. Le difficoltà iniziali nell'utilizzo di questo strumento erano dovute alla carenza di conoscenza di Prolog e della libreria in generale.

tuProlog è uno strumento molto utile perchè riesce ad adattasi all'ambiente sul quale viene utilizzato. In questo caso, per il progetto di tesi, è stato scelto Alchemist e l'adattabilità di tuProlog si è rivelata davvero efficiente poichè in poco tempo si è riusciti ad implementare la gestione della teoria dell'agente.

tuProlog è uno strumento che ha la possibilità di essere utilizzato in maniera differente in base all'ambito applicativo, poichè mette a disposizione una serie di predicati nativi (o built-in) oltre che delle librerie per la gestione dei vari paradigmi supportati.
Una peculiare modalità di utilizzo di tuProlog, fornita per il paradigma orientato agli oggetti, è la registrazione di istanze di classi all'interno di varibili inserite nella teoria. In questo modo è possibile modificare le proprietà dell'istanza o invocare i suoi metodi utilizzando la variabile tuProlog direttamente dalla teoria.
Questa funzionalità verrà utilizzata nello sviluppo del progetto di tesi per integrare l'ambiente dell'interprete, che sarà Alchemist, con l'agente e il linguaggio che verrà definito nel lavoro di tesi.

Capitolo 4

Agent speak

AgentSpeak è un linguaggio orientato agli agenti basato sulla programmazione logica e sul modello BDI (Belief-Desires-Intention). All'interno del capitolo viene descritto cos'è il modello BDI e come funziona AgentSpeak, ed in particolare il ciclo di ragionamento, esemplificato da quello di Jason.

4.1 Agenti BDI con AgentSpeak

Il modello BDI consente di rappresentare le caratteristiche e le modalità di raggiungimento di un obiettivo secondo il paradigma ad agenti. Gli agenti BDI forniscono un meccanismo per separare le attività di selezione di un piano, fra quelli presenti nella sua teoria, dall'esecuzione del piano attivo, permettendo di bilanciare il tempo speso nella scelta del piano e quello per eseguirlo.

I **beliefs** sono quindi informazioni dello stato dell'agente, ovvero ciò che l'agente sa del mondo [Bib. 2] il suo insieme è chiamato 'belief base' o 'belief set'.

I **desires** rappresentano tutti i possibili piani che un agente potrebbe eseguire [Bib. 2]. Rappresentano ciò che l'agente vorrebbe realizzare o portare a termine: i *goals* sono desideri che l'agente persegue attivamente ed è quin-

di bene che tra loro siano coerenti, cosa che non è obbligatoria per quanto riguarda il resto dei desideri.

Le **intentions** identificano i piani a cui l'agente ha deciso di lavorare o a cui sta già lavorando e a loro volta possono contenere altri piani [Bib. 2].

Gli **eventi** innescano le attività reattive, ovvero la caratteristica di proattività degli agenti, come ad esempio l'aggiornamento dei beliefs, l'invocazione di piani o la modifica dei goals.

4.2 Definizione AgentSpeak

AgentSpeak è un linguaggio di programmazione basato su un linguaggio del primo ordine con eventi e azioni [Bib. 1]. Il comportamento degli agenti è dettato da quanto definito nel programma scritto in AgentSpeak. I beliefs correnti di un agente sono relativi al suo stato attuale, all'enviroment e agli altri agenti. Gli stati che un agente vuole determinare sulla base dei suoi stimoli esterni e interni sono i desideri [Bib. 1]. L'adozione di programmi per soddisfare tali stimoli è detta intenzioni.

4.2.1 Composizione

Un'agente in AgentSpeak è formato da una 'belief base' e da una serie di piani opportunamente programmati. La 'belief base' è il contenitore dello stato dell'agente, dove sono presenti tutte le informazioni che esso ha in riferimento a se stesso e all'ambiente. Questo set di belief può essere modificato in modo continuo dalle azioni scatenate nel ciclo di ragionamento. I piani sono sequenze di azioni o goal che permettono all'agente di reagire a situazioni che avvengono nell'ambiente o internamente.
Qui di seguito viene descritto quali sono le fasi del ciclo di ragionamento dell'agente e in che modo viene definito il suo operato. La sequenza di esecuzione di questa iterazione è fondamentale per la realizzazione di un agente e quindi importante comprenderla per realizzare un'architettura appropriata.

4.2.2 Ciclo di ragionamento

Il ciclo di ragionamento è il modo in cui l'agente prende le sue decisioni e mette in pratica le azioni. Esso è composto di otto fasi: le prime tre sono quelle che riguardano l'aggiornamento dei belief relativi al mondo e agli altri agenti, mentre altre descrivono la selezione di un evento che permette l'esecuzione di un'intenzione dell'agente.

a. Percezione ambiente

La percezione effettuata dall'agente all'interno del ciclo di ragionamento è utilizzata per poter aggiornare il suo stato. L'agente interroga dei componenti capaci di rilevare i cambiamenti nell'ambiente [Bib. 2] e di emettere dati consultabili utilizzando opportune interfacce.

b. Aggiornamento beliefs

Ottenuta la lista delle percezioni è necessario aggiornare la 'belief base'. Ogni percezione non ancora presente nel set viene aggiunta e al contrario quelle presenti nel set e che non sono nella lista delle percezioni vengono rimosse [Bib. 2]. Ogni cambiamento effettato nella 'belief base' produce un evento: quelli generati da percezioni dell'ambiente sono detti eventi esterni; quelli interni, rispetto agli altri, hanno associata un'intenzione.

c. Ricezione e selezione messaggi

L'altra sorgente di informazioni per un agente sono gli altri agenti presenti nel sistema. L'interprete controlla i messaggi diretti all'agente e li rende a lui disponibili [Bib. 2]: ad ogni iterazione del ciclo può essere processato solo un messaggio. Inoltre,può essere assegnata una priotità ai messaggi in coda definendo una funzione di prelazione per l'agente.
Prima di essere processati i messaggi passano all'interno di una funzione di selezione che definisce quali messaggi possano essere accettati dall'agen-

te [Bib. 2]. Questa funzione può essere implementata ad esempio per far ricevere solo i messaggi di un certo agente.

d. Selezione evento

Gli eventi rappresentano la percezione del cambiamento nell'ambiente o dello stato interno dell'agente [Bib. 2], come il goal. Ci possono essere vari eventi in attesa ma in ogni ciclo di ragionamento può esserne gestito uno solo, il quale viene scelto dalla funzione di selezione degli eventi che ne seleziona uno dalla lista di quelli in attesa. Se la lista di eventi fosse vuota si passa direttamente alla penultima fase del ciclo di ragionamento [Bib. 2], ovvero la selezione di un'intenzione.

e. Recupero piani rilevanti

Una volta selezionato l'evento è necessario trovare un piano che permetta all'agente di agire per gestirlo. Per fare ciò viene recuperata dalla 'Plan Library' la lista dei piani rilevanti, verificando quali possano essere unificati con l'evento selezionato [Bib. 2]. L'unificazione è il confronto relativo a predicati e termini. Al termine di questa fase si otterrà un set di piani rilevanti per l'evento selezionato che verrà raffinato successivamente.

f. Selezione piano appplicabile

Ogni piano ha un contesto che definisce con quali informazioni dell'agente può essere usato. Per piano applicabile si intendeno quelli che, in relazione allo stato dell'agente, possono avere una possibilità di successo. Viene quindi controllato che il contesto sia una conseguenza logica della 'belief base' dell'agente [Bib. 2]. Vi possono anche essere più piani in grado di gestire un evento ma l'agente deve selezionarne uno solo ed impegnarsi ad eseguirlo.
La selezione viene fatta tramite un'apposita funzione che inoltre tiene conto dell'ordinamento dei piani in base alla loro posizione nel codice sorgente oppure dell'ordine di inserimento. Quando un piano è scelto, viene creata

un'istanza di quel piano che viene inserita nel set delle intenzioni [Bib. 2]: sarà l'istanza ad essere manipolata dall'interprete e non il piano nella libreria.

Ci sono due possibili modalità per la creazione di un'intenzione e dipende dal fatto che l'evento selezionato sia esterno o interno [Bib. 2]. Nel primo caso viene semplicemente creata l'intenzione, altrimenti viene inserita un'altra intenzione in testa a quella che ha generato l'evento, poichè è necessario eseguire fino al completamento un piano per raggiungere tale goal.

g. Selezione intenzione

A questo punto, se erano presenti eventi da gestire, è stata aggiunta un'altra intenzione nello stack. Un agente ha tipicamente più di un'intenzione nel set delle intenzioni che potrebbe essere eseguita, ognuna delle quali rappresenta un diverso punto di attenzione [Bib. 2]. Ad ogni ciclo di ragionamento avviene l'esecuzione di una sola intenzione, la cui scelta è importante per come l'agente opererà nell'ambiente.

h. Esecuzione intenzione

L'intenzione, scelta nello step precedente, non è altro che il corpo di un piano formato da una sequenza di istruzioni, ognuna delle quali, una volta eseguita, viene rimossa dall'istanza del piano. Terminata l'esecuzione un'intenzione, quest'ultima viene restituita al set delle intenzioni a meno che non debba aspettare un messaggio o un feedback dell'azione [Bib. 2]: in questo caso viene memorizzata in una struttura e restituita una volta ricevuta la risposta. Se un'intenzione è sospesa non può essere selezionata per l'esecuzione nel ciclo di ragionamento.

Scambio di messaggi

Lo scambio di messaggi è la comunicazione standard che avviene tra agenti per comunicare tra loro e operare in base al contenuto ricevuto. La comunicazione definita da AgentSpeak utilizza tre parti. La prima è la coda dei

messaggi in input, ovvero una lista contenente tutti i messaggi che il sistema o interprete riceve e che sono destinati all'agente. La seconda è la coda dei messaggi di output che si allunga ogni volta che l'agente vuole inviare un messaggio ad un altro agente. L'ultima è una struttura all'interno della quale vengono memorizzate le intenzioni che sono sospese dall'esecuzione poichè aspettano una risposta dal canale di comunicazione dei messaggi.
L'interprete è il mezzo per il quale i messaggi trasmessi. Esso infatti ha il compito di recuperare tutti i messaggi nella coda in uscita di ogni agente e successivamente recapitarli. Per la consegna viene recuperato l'agente destinatario di ogni messaggio e poi quest'ultimo viene posizionato nella coda di quelli in input dell'agente, in modo tale che possa recuperarne il contenuto al prossimo ciclo di ragionamento.

Capitolo 5

Percorso di approfondimento

In questo capitolo sono descritti i processi evolutivi di apprendimento dei due strumenti principali per lo sviluppo del lavoro di tesi, tuProlog e Alchemist. Inoltre viene descritto anche come è stato progettato il processo di integrazione tra il meta-modello di Alchemist e il modello ad agenti.

5.1 Approfondimento di tuProlog

Per quello che riguarda lo studio di tuProlog per prima cosa sono stati recuperati i materiali forniti durante le lezioni e i laboratori, dai quali sono stati estratti i concetti principali, non solo di tuProlog ma anche di Prolog. Le informazioni recuperate sono state poi raffrontate con la versione '3.0.0' del documento 'Manuale tuProlog' attraverso il quale è stato possibile approfondire ulteriori aspetti di tuProlog.
Sono state quindi estratte le informazioni relativamente ai predicati (libreria, teoria, built-in), alle entità della sintassi (atomi, variabili, operatori, ...) e configurazione del motore. Inoltre, si sono apprese ulteriori nozioni riguardo agli unificatori e agli operatori di conoscenza, ad esempio per inserire o recuperare informazioni dalle teorie o librerie.
Inoltre, è stato molto utile approfondire lo studio delle librerie, alcune delle quali sono state poi utilizzate per la gestione della teoria dell'agente. In

particolare, si è compreso come gestire la creazione e la decomposizione di termini e strutture e il recupero e la gestione delle clausole (utilizzate poi per la gestione dei fatti e delle regole presenti nelle teorie).

Una parte certamente di risalto è da attribuire ad una funzionalità di tuProlog, già citata in precedenza, che consente una gestione dell'integrazione fra i due strumenti selezionati. Questo è possibile poichè tuProlog supporta la programmazione multi-paradigma e, attraverso l'apposita libreria OOLibrary, consente di manipolare oggetti Java anche tramite una teoria.
La funzionalità specifica che verrà poi utilizzata nella tesi è la possibilità di registrare all'interno di variabili della teoria tuProlog oggetti istanziati lato Java. In questo modo sarà possibile invocare funzioni o manipolare le proprietà di un certo oggetto in relazione all'implementazione presente nella teoria caricata nel motore tuProlog. Questo apre ad una serie di scenari che possono facilitare il design dell'architettura ed inoltre agevolare l'implementazione del modello ad agenti.

5.2 Approfondimento di Alchemist

Il processo di apprendimento di Alchemist è stato un po' più lungo rispetto a quello di tuProlog per via del numero di parti da cui è composto.

La prima fase è stata dedicata allo studio del meta-modello sul quale questo ambiente di simulazione si sviluppa; sono state analizzate le varie entità per capire il loro ruolo e il loro comportamento durante l'esecuzione della simulazione.

Dopo il modello ci si è dedicati ad un'analisi relativa al materiale già presente all'interno di Alchemist. Partendo dalla documentazione, sono state analizzate le interfacce di ogni entità per approfondire le caratteristiche e studiarne le peculiarità da poter sfruttare in fase di design per il progetto di tesi.
Successivamente è stata effettuate un'indagine sulle incarnazioni già presenti all'interno di Alchemist, in modo da capire in quali modi è stato utilizzato

questo simulatore e in che modo sfruttare a pieno il modello.
Proprio in questa fase, si è iniziato a pensare ai possibili punti di contatto tra il modello ad agenti e il meta-modello proposto da Alchemist. La descrizione del ragionamento e delle conclusioni è riportata nella sezione 5.2.1.

5.2.1 Mapping modelli

È necessario capire quale risulta il migliore modo, in termini di performace e espressività, per unire i due modelli. In questa fase si vuole quindi pensare come realizzare sul meta-modello fornito da Alchemist il modello ad agenti cercando eventuali incongruenze o opportunità per massimizzare il risultato.

Si è partiti analizzando le entità del meta-modello e per ognuna è stato posto l'interrogativo sul fatto che potesse essere un'agente. Fin da subito sono state ritenute inadatte l'Environment e la Molecola: il primo perchè è esso stesso lo spazio e non avrebbe potuto rappresentare lo spazio degli agenti, mentre le molecole perchè forniscono un livello di dettaglio troppo elevato e non hanno una struttura che consente di contenere lo stato dell'agente.

Le entità rimaste da analizzare sono quindi il Nodo e la Reazione. Mappando il Nodo come agente ne deriva che l'Environment corrisponderà allo spazio degli agenti mentre, all'interno dell'agente, le Molecole e le Concentrazioni potranno essere utilizzate per gestire la 'belief base' e le reazioni saranno riferite ai piani, utilizzando le Condizioni come clausola per scatenare le Azioni. Questo tipo di mapping consente di realizzare simulazioni di sistemi non complessi, in cui agenti allo stesso livello operano e comunicano tra loro.

Posizionando l'agente nella Reazione, quindi più internamente rispetto al precedente mapping, il Nodo sarà quindi un contenitore di agenti e l'Environment lo spazio nel quale si muovono i gruppi di agenti. Ogni agente avrà il riferimento ad una Condizione e ad una Azione: quest'ultima conterrà il ciclo di ragionamento dell'agente mentre la Condizione, sempre vera, ne determinerà la clausola di esecuzione. Utilizzando questa seconda ipotesi sarà possibile realizzare simulazioni di sistemi complessi, nei quali dei nodi, che

potrebbero essere dispositivi mobili (ad esempio cellulari), si muovo nello spazio ed ognuno dei quali al suo interno contiene un gruppo di agenti che possono interagire sia internamente che esternamente.

Dopo aver analizzato le due possibili alternative presentate, è stato scelto il mapping in cui l'agente è posizionato nella Reazione poichè permette una maggiore espressività e un'apertura verso più scenari applicativi.

5.2.2 Configurazione di una simulazione

Successivamente ci si è dedicata alla comprensione della struttura della configurazione di una simulazione. Per poter scrivere una simulazione è necessario per prima cosa imparare le nozioni base di YAML, poichè il documento che il simulatore si aspetta in input è una mappa definita proprio tramite questo linguaggio. La struttura di una simulazione contiene una serie di keyword, che possono essere obbligatorie o opzionali, tra cui:

- incarnation, per definire l'incarnazione
- environment, per definire l'ambiente
- network-model, per definire la linking-rule
- displacement, per definire la disposizione e la tipologia di nodi e reazioni

le quali, in base al tipo, contengono un valore oppure una mappa innestata nella quale specificano altri parametri.

Per effettuare i test delle problematiche analizzate durante lo svolgimento dell'attività propedeutica sono state scritte simulazioni molto basilari, dove ad esempio era presente un nodo con uno solo agente (tranne nel caso dello scambio di messaggi).

Un esempio di configurazione è quello presentato nel Codice sorgente 5.1.

```
1 incarnation: agent
2
3 network-model:
```

```
4   type: ConnectWithinDistance
5   parameters: [10]
6
7 displacements:
8   - in: {type: Circle, parameters: [1,0,0,2]}
9     programs:
10      -
11        - time-distribution: 1
12          program: "some_agent"
```

Codice sorgente 5.1: Esempio configurazione di una simulazione

In questo caso si vuole eseguire una simulazione che utilizzi l'incarnazione ad agenti. L'environment non è espresso e sarà considerato quello di default (ovvero Continuous2DEnvironment), mentre è specificata la keyword 'network-model' utilizzata per la linking-rule: sfruttando la classe 'ConnectWithinDistance' e passandogli un certo parametro si vuole che i nodi, la cui distanza è minore di quella indicata dal parametro, appartengano allo stesso vicinato.
Nell'ultima parte viene invece specificata la disposizione dei nodi e delle reazioni o azioni in essi contenute. Nel caso mostrato, verrà creato, all'interno di un cerchio di centro (0,0) e di raggio 2, un solo nodo al cui interno sarà presente una reazione, identificata opportunamente dal parametro 'some_agent', la quale avrà come distribuzione temporale 1, ovvero verrà eseguita una volta ad ogni trigger da parte dello scheduler di Alchemist.

5.3 Integrazione di modelli e strumenti

Il passo successivo è stato quello di iniziare a prendere confidenza con il simulatore provando ad implementare alcuni problemi basilari che sarebbero tornati utili in futuro. Alcuni dei problemi affrontati sono:

1. agente che effettua una computazione elementare

2. agente che effettua una computazione condizionata

3. agente che è in grado di spostarsi nello spazio

4. scambio di messaggi tra agenti (in questo punto è stato unito tuProlog)

Nel fronteggiare i vari quesiti sono state prodotte implementazioni di piccole parti che sono servite come punto di partenza per la realizzazione del progetto di tesi.

Si è iniziato implementando l'incarnazione attraverso la creazione del nodo e di una reazione, la quale al suo interno contiene un'azione. Quest'ultima è l'entità che viene innescata, se le condizioni si verificano, e attraverso il metodo *execute* (invocato dal simulatore) esegue una serie operazioni definite nell'implementazione dell'azione.

I primi due quesiti sono risultati complessi non tanto per la loro natura ma per lo scoglio iniziale nell'approccio ad Alchemist. Successivamente si è passati allo spostamento del nodo, sul quale è stato effettuato un ragionamento per permettere al nodo di spostarsi in ogni direzione e della distanza esatta percorsa nel tempo della simulazione. La soluzione prevede l'utilizzo di variabili per velocità, direzione e tau (tempo della simulazione dell'ultimo aggiornamento) con i quali è costruito un cerchio che ha come centro l'ultima posizione del nodo e come raggio la distanza calcolata con tempo (differenza tra tau attuale e quello dell'ultimo aggiornamento) e velocità. Sulla circonferenza, in base all'angolo definito dalla direzione, verrà preso il punto della nuova posizione del nodo.
Dopo aver realizzato i primi tre quesiti descritti nel precedente elenco, si è passati alla risoluzione dell'ultima problematica per avere conoscenza dell'intera situazione, consentendo di avere cognizione dell'intero problema e del supporto che possono fornire gli strumenti descritti in questo documento. Per la realizzazione si è utilizzato il motore tuProlog all'interno della quale è stata caricata una semplice teoria che mandava indietro il messaggio ricevuto (l'esempio realizzato è stato il Ping Pong).

La parte di integrazione di tuProlog è stata effettuata importanto la libreria 'alice.tuprolog' in Alchemist e poi utilizzandone il motore, sia per caricare le teorie sia per l'interrogazione di quest'ultime. La parte di registrazione di oggetti Java all'interno di variabili tuProlog verrà inserita direttamente durante l'implementazione del progetto di tesi.

Conclusioni

Il tempo speso per questa fase di preparazione alla prova finale è stato molto utile poichè ha permesso di conoscere in modo approfondito gli strumenti e di padroneggiarli. In questo documento è stata descritta un'evoluzione a partire dai problemi iniziali riscontrati con l'approccio a nuovi strumenti di sviluppo, passando poi per una fase di analisi e studio che, infine, ha portato alla risoluzione di semplici problemi (come lo scambio di messaggi o lo spostamento di un nodo), i quali hanno consentito di prendere ancora più famigliarità con Alchemist, tuProlog e il modello ad agenti. Le problematiche affrontate non sono state scelte a caso ma selezionate pensando a situazioni precise che si sarebbero dovute affrontare per la realizzazione del progetto per la tesi.

Vi sono ancora problemi aperti, soprattutto legati alle integrazioni, quali:

- organizzazione struttura classi
- la gestione del ciclio di ragionamento dell'agente
- utilizzo risorse tuProlog come teoria dell'agente

che saranno affrontati nella prossima fase di analisi, durante il design, prima di iniziare l'implementazione dell'interprete.

Alcune delle parti sviluppate nella risoluzione dei sotto-problemi saranno prese come spunto per ottimizzazioni e la realizzazione del progetto di tesi.

Bibliografia

[Bib. 1] AgentSpeak(L): BDI Agents speak out in a logical computable language, Anand S. Rao, 1996

[Bib. 2] Programming multi-agent systems in AgentSpeak using Jason, Rafael H. Bordini, Jomi Fred Hubner, Michael Wooldridge, 2007

35

Sitografia

[Sit. 1] http://apice.unibo.it/xwiki/bin/view/Tuprolog/

[Sit. 2] http://alchemistsimulator.github.io