ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA

Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

INTEGRAZIONE TRA GAME ENGINE E SISTEMI MULTI-AGENTE

Attività Propedeutica alla Prova Finale

Studente:
LUCA PASCUCCI
Matricola:
763205

ANNO ACCADEMICO 2018–2019

Indice

Sommario

Con lo sviluppo della tecnologia per creare ambienti virtuali più realistici, complessi e dinamici sta aumentando l'interesse sulle Game Engine (GE), le quali si rivelano sempre più importanti in numerosi ambiti, permettendo lo sviluppo di applicazioni moderne e videogiochi con relativa facilità.

Nell'ambito di ricerca, in particolare nel contesto dei Sistemi Multi-Agente (MAS), sono state recentemente utilizzate come supporto alla definizione dell'ambiente e come mezzo abilitante per la coordinazione.[19]

In questa attività propedeutica alla tesi si analizzerà lo stato dell'arte attuale delle integrazioni tra MAS e GE, ricercando nuove tecnologie utilizzabili con lo scopo finale di realizzare una prima architettura di integrazione, sviluppata successivamente in fase di tesi, utilizzabile per diversi scenari di associazione e comunicazione tra il mondo delle GE ed il mondo dei MAS, lasciando entrambi separati senza modificare le loro astrazioni e funzionalità.

Capitolo 1

Background

1.1 Stato dell'arte

Come premessa al lavoro svolto, si presenta una prima introduzione ai Sistemi Multi-Agente (MAS), alle Game Engine (GE) ed alle integrazioni già realizzate, spiegando brevemente le astrazioni presenti nei MAS. Si vedrà un breve excursus storico sulle GE e le diverse tipologie di soluzioni già realizzate.

1.1.1 Sistema Multi-Agente

La crescente complessità nell'ingegnerizzazione dei sistemi software ha portato alla necessità di modelli e astrazioni in grado di rendere più facile la loro progettazione, lo sviluppo e il mantenimento. In questa direzione, la computazione orientata agli agenti viene in aiuto agli ingegneri ed informatici per costruire sistemi complessi, virtuali o artificiali, permettendo una loro agevole e corretta gestione.[22]

In particolare, la ricerca e le tecnologie per MAS hanno introdotto nuove astrazioni per affrontare la complessità durante la progettazione di sistemi o applicazioni composte da individui che non agiscono più da soli, ma all'interno di una società. Le tecnologie e i modelli agent-oriented sono attualmente diventati una potente tecnica in grado di affrontare molti problemi che vengono alla luce durante la progettazione di sistemi computer-based in termini di entità che condividono caratteristiche quali l'autonomia, l'intelligenza, la distribuzione, l'interazione, la coordinazione, etc.

L'ingegnerizzazione dei MAS si occupa, infatti, di costruire sistemi complessi dove più entità autonome- chiamate agenti- cercano di raggiungere in maniera proattiva i loro scopi sfruttando le interazioni tra di essi (come una società), e con l'ambiente circostante. Questo modello può essere visto come un paradigma general-purpose, il quale prevede l'utilizzo di tecnologie agent-oriented in diversi scenari applicativi. [33]

Un MAS fornisce agli sviluppatori e ai designer tre astrazioni principali:

- Agenti: Le entità autonome che compongono il sistema. Sono in grado di comunicare e possono essere intelligenti, dinamici, e situati;
- Società: Rappresenta un gruppo di entità il cui comportamento emerge dall'interazione tra i singoli elementi;
- Ambiente: Il "contenitore" in cui gli agenti sono immersi e con il quale questi ultimi possono interagire, modificandolo. La caratteristica degli agenti di essere situati nell'ambiente in cui si trovano permette loro di percepire e produrre cambiamenti su di esso.

1.1.1.1 Piattaforme agent-oriented

Le piattaforme che sono state prese in esame per la programmazione agent-oriented combinano le caratteristiche dichiarative e imperative e sono le seguenti:

- Jason: framework software che interpreta un'estensione del linguaggio AgentSpeak[1] [6]
- JADE[2]: framework software pienamente implementato su Java[3] [2]

La strada intrapresa per il progetto di tesi ha portato all'utilizzo di Jason, dato che l'architettura BDI, che verrà illustrata nella sezione 2.2.1, si presta bene al dotare di intelligenza personaggi virtuali.

[1] Logic-based programming
[2] JAVA Agent DEvelopment
[3] Java-based programming

1.1.2 Game Engine

Le GE sono framework utilizzati per supportare la progettazione e lo sviluppo di giochi. Il termine "Game Engine" nacque a metà degli anni '90 in riferimento a giochi sparatutto in prima persona (FPS) come il popolare "Doom" progettato con una separazione ragionevolmente ben definita tra i suoi componenti software principali (come il sistema di rendering grafico tridimensionale, il sistema di rilevamento delle collisioni o il sistema audio) e le risorse artistiche, i mondi e le regole di gioco che comprendevano l'esperienza del giocatore.

La maggior parte delle GE sono realizzate con cura e messe a punto per eseguire un gioco particolare su una particolare piattaforma hardware. L'avvento di hardware per computer sempre più veloce e schede grafiche specializzate, insieme a algoritmi di rendering e strutture di dati sempre più efficienti, sta cominciando ad ammorbidire le differenze tra i motori grafici di diversi generi. È ora possibile utilizzare un motore sparatutto in prima persona per creare un gioco di strategia, ad esempio. Tuttavia, esiste ancora il compromesso tra generalità e ottimizzazione. Un gioco può sempre essere reso più impressionante perfezionando il motore in base ai requisiti e ai vincoli specifici di una determinata piattaforma di gioco e/o hardware.[10]

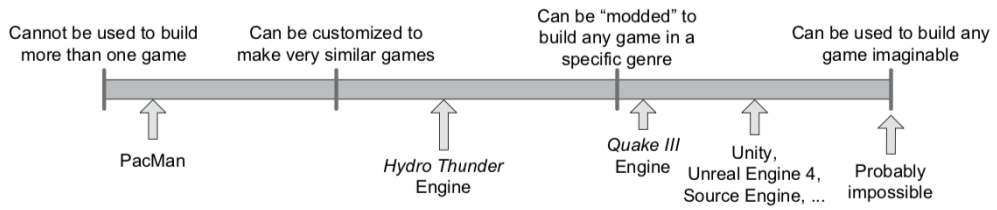

Figura 1.1: Game Engine reusability gamut [10]

Le GE moderne sono strutture general-purpose multipiattaforma orientate verso ogni aspetto della progettazione e dello sviluppo del gioco, come il rendering 2D/3D delle scene di gioco, i motori fisici per la dinamica ambientale (movimenti, dinamica delle particelle, rilevamento delle collisioni, prevenzione degli ostacoli, ecc.), suoni, script comportamentali, intelligenza artificiale dei personaggi e molto altro.

Come esempio significativo che rappresenta la gamma di piattaforme disponibili, nella sezione 2.2.4 verrà esaminata una delle più popolari GE - Unity[28] - con l'obiettivo di:

- rilevare quelle astrazioni e quei meccanismi che hanno più probabilità di avere una controparte nel MAS, o almeno quelli che sembrano fornire un supporto nel riformulare le astrazioni mancanti del MAS;
- evidenziare le opportunità per colmare le lacune concettuali / tecniche che ostacolano l'integrazione dei due mondi.

1.1.3 Integrazione

Sono già presenti esempi di integrazione tra GE e MAS che concentrano la propria attenzione su obiettivi specifici a livello tecnologico, piuttosto che sulla creazione di un'infrastruttura orientata agli agenti basata sul gioco per scopi generici. Per esempio:

- QuizMASter [3] concentrato sull'astrazione degli agenti collegando gli agenti MAS ai personaggi dei motori di gioco, nel contesto dell'apprendimento educativo;
- CIGA [31] considera sia la modellazione degli agenti che quella dell'ambiente, per agenti virtuali generici in ambienti virtuali;
- GameBots [16] concentrato sull'astrazione dell'agente, ma considera ancora l'ambiente fornendo al contempo un framework di sviluppo e un runtime per i test di sistemi multi-agente in ambienti virtuali;
- UTSAF [25] si concentra sulla modellistica ambientale nel contesto di simulazioni distribuite in ambito militare[4].

Sebbene rappresentino chiaramente esempi di integrazione (parzialmente) riuscita di MAS in GE, i lavori sopra elencati presentano alcune carenze rispetto all'obiettivo che perseguiamo in questo documento.

Solamente CIGA rappresenta un'eccezione che riconosce il divario concettuale tra MAS e GE, e propone soluzioni per affrontarlo (anche se a livello

[4]Gli agenti vengono considerati, ma solo come mezzo di integrazione tra diverse piattaforme di simulazione, non nel contesto del GE sfruttato per il rendering di simulazione

tecnologico). L'unico strato preso in considerazione nel perseguimento dell'integrazione è quello tecnologico - nessun modello, nessuna architettura, nessun linguaggio. All'interno di QuizMASter, UTSAF e GameBot (in una certa misura) l'integrazione è specificamente realizzata per obiettivo dove la maggior parte degli approcci fornisce ai programmatori alcune astrazioni per trattare con agenti e ambiente, nessuna attenzione viene data alle astrazioni sociali.[19]

1.2 Situazione iniziale

I lavori precedentemente svolti, che hanno contribuito alla definizione di questo percorso, utilizzano due approcci nettamente separati per l'integrazione MAS e GE:

1. Integrazione delle caratteristiche dei MAS all'interno della GE;
2. Realizzazione di un middleware, come layer software, per collegare l'ambiente MAS con GE.

1.2.1 MAS all'interno di GE

Il primo punto è stato realizzato implementando due modelli tipici dei MAS:

- Il modello Beliefs, Desires, Intentions (BDI) per la programmazione degli agenti [24];
- Un modello di coordinazione degli agenti tramite spazio di tuple e primitive Linda [7][1].

Il cuore pulsante di entrambi i lavori risiede nell'uso intensivo di un interprete Prolog fatto ad hoc per Unity, UnityProlog [11]. Questo interprete dispone di molte funzionalità per estendere l'interoperabilità di Prolog con i GameObject. Dal momento che è stato progettato per essere usato in maniera specifica con Unity, nasce con delle primitive che permettono di accedere e manipolare GameObject e i relativi componenti direttamente da Prolog. UnityProlog introduce tuttavia alcune limitazioni da tenere bene in considerazione [24], anche se allo stato attuale è l'unica versione di Prolog del quale è stato dimostrato il corretto funzionamento:

- Un interprete per Prolog non sarà mai performante quanto lo può essere un compilatore e questo può rappresentare un problema per simulazioni di MAS più grandi.

- Utilizza lo stack C# come stack di esecuzione, quindi la tail call optimization non è ancora supportata.

- Non supporta regole con più di 10 subgoal, quindi a fronte di una regola complessa con tanti goal da controllare, è necessario frammentare la regola in questione in sotto regole con non più di 10 subgoal per ognuna.

1.2.2 MAS e GE separati

Il secondo percorso si differenzia dal primo per la scelta di lasciare separati GE da MAS realizzando un canale di comunicazione tra i due ambienti. É stata introdotta una terminologia per contraddistinguere le entità realizzate sul GE (GameObject) e su MAS (agenti), rispettivamente definite "corpi" e "menti" virtuali. [9]

Fondamentalmente un corpo deve eseguire azioni e, a seguito di determinati eventi, deve trasmettere le proprie percezioni alla mente, quest'ultima, invece, deve elaborare le percezioni per decidere quali azioni far svolgere al proprio corpo. Per rendere possibile questa comunicazione è stato progettato e implementato un sistema middleware definito secondo il seguente schema.

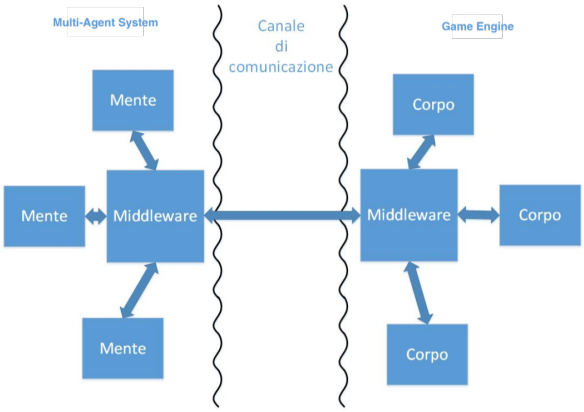

Figura 1.2: Il middleware viene suddiviso in due parti, poste sui due lati del canale di comunicazione. [9]

Dalla figura si può notare la separazione del middleware nei due sistemi, motivato dalle diverse tecnologie utilizzate dai due ambienti. Questa divisione vincola la realizzazione di una nuova parte di middleware in caso di utilizzo di un diversa tipologia di GE e/o MAS.

Il protocollo di comunicazione tra le entità è stato realizzato utilizzando messaggi strutturati. Da una parte, le menti devono definire quale azione deve compiere il relativo corpo (es. "muoviti in avanti", "ruota", "prendi", ecc.), dall'altro i corpi devono far sapere alle relative menti le proprie percezioni dell'ambiente circostante (es. "mi ha toccato un'entità", "sono alle coordinate 23,12,-6", ecc.).[9]

Il progetto di tesi è stato realizzato basandosi sul secondo approccio, ma differenziandosi per scelte architetturali e tecnologiche.

Capitolo 2

Nuova integrazione

2.1 Astrazioni principali

La situazione attuale di integrazione dei due sistemi presenta dei limiti. Per le motivazioni sopra elencate, in questo elaborato si cerca di definire un'infrastruttura utilizzabile per diversi scenari di associazione e comunicazione tra il mondo delle GE ed il mondo dei MAS, lasciando entrambi separati senza modificare le loro astrazioni e funzionalità.

Prima di proseguire con la trattazione è opportuno definire brevemente alcuni concetti che saranno utilizzati da qui in avanti, quali quelli di entità, mente, corpo, azione e percezione. In seguito, verrà quindi proposto uno schema per illustrarne la struttura.

2.1.1 Entità

Con "entità" generalmente viene inteso un insieme di elementi dotati di proprietà comuni dal punto di vista dell'applicazione considerata.[32] Concettualmente, in questo dominio, l'entità viene intesa come oggetto divisibile in due parti, mente e corpo, che collegate riescono a trasmettersi informazioni, utilizzate dalla mente per raggiungere i propri obiettivi e dal corpo per diventare "attivo" nell'ambiente in cui si trova.

2.1.2 Mente

La nozione di "mente" può essere caratterizzata da alcuni punti chiave fondamentali:

- autonomia
- interazione
- obiettivi

In altre parole, una mente può essere pensata come un componente software autonomo che interagisce con l'ambiente per svolgere i propri compiti. I punti sopra elencati rendono facile l'associazione della mente al concetto di Agente, spiegato nella sezione 1.1.1, poiché questa entità del Sistema Multi-Agente (MAS) ingloba astrazioni simili a quelle illustrate nella sezione 2.2.1.

2.1.3 Corpo

"Corpo", poi, è un termine generico che indica qualsiasi porzione limitata di materia, cui si attribuiscono, in fisica, le proprietà di estensione, divisibilità, impenetrabilità.[32] In questa trattazione è associabile alla nozione di GameObject di Unity, spiegata nella sezione 2.2.4, utilizzata per avere una rappresentazione fisica dell'entità da realizzare.

2.1.4 Azione

Nel suo significato più generale, un'"azione" è intesa come attività od operazione posta in essere da un determinato soggetto.[32] In questo studio, si considera come "azione" un certo gesto richiesto dalla mente che può essere associato ad una operazione eseguita dal corpo, ad esempio, nel caso di un'azione del tipo *"vai a (posizione)"*, richiesta dalla mente, corrisponde il movimento del corpo nell'ambiente verso la posizione indicata.

2.1.5 Percezione

La "percezione" è un atto cognitivo mediato dai sensi con cui si avverte la realtà di un determinato oggetto e che implica un processo di organizzazione e interpretazione.[32].

In questo lavoro, la percezione si collega ad una certa sensazione rilevata dal corpo ed inviata alla mente per portarla a conoscenza di questa nuova informazione, ad esempio, nel caso del raggiungimento della posizione richiesta in precedenza, il corpo trasmette la percezione *"arrivato (posizione)* che informa la mente del completamento dell'operazione.

Esiste inoltre, da parte del corpo, la possibilità di inviare percezioni "libere" ossia non associate a risposta di un'azione inviata dalla mente. Un semplice esempio è il contatto del corpo con una qualsiasi altra entità nell'ambiente che corrisponde all'invio di una percezione del tipo *"toccato(nome_entità)"*.

2.1.6 Struttura entità

Nella figura sottostante viene rappresentata la struttura di una generica entità, dove:

- Il corpo esegue azioni e, in risposta a queste ultime, oppure, a seguito di determinati eventi esterni, trasmette le proprie percezioni alla mente.

- La mente elabora le percezioni per decidere quali azioni far svolgere al proprio corpo.

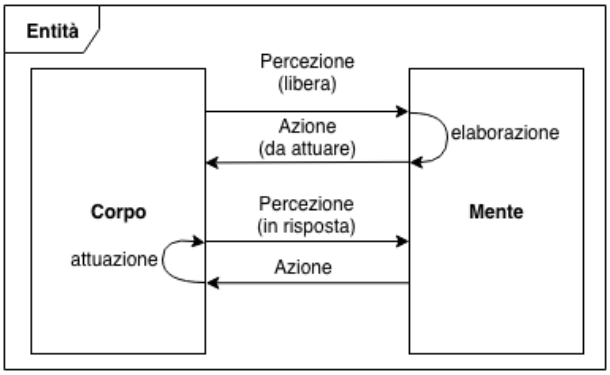

Figura 2.1: Struttura di una generica entità

2.2 Stack Tecnologico

Di seguito vengono illustrate le principali tecnologie prese in esame per la realizzazione di un middleware di collegamento dei due sistemi precedentemente definiti, stabilendo anche quale Sistema Multi-Agente (MAS), Game Engine (GE), framework e tecnologia di collegamento sono stati presi come principale riferimento.

2.2.1 JaCaMo

JaCaMo è un framework per la programmazione orientata agli agenti che combina tre tecnologie già affermate e sviluppate da diversi anni.

Un sistema multi-agente JaCaMo o, equivalentemente, un sistema software programmato con JaCaMo è definito da un'organizzazione Moise di agenti BDI autonomi basati su concetti come ruoli, gruppi, missione e schemi, implementati tramite Jason, che lavorano in ambienti condivisi distribuiti basati su artefatti, programmati in CArtAgO.

Ognuna delle tre tecnologie indipendenti che compongono il framework ha il proprio set di astrazioni, modelli di programmazione e meta-modelli di riferimento; per questo motivo in JaCaMo è stato realizzato un meta-modello globale, con l'obiettivo di definire le dipendenze, le connessioni, i mapping concettuali e le sinergie tra le differenti astrazioni rese disponibili da ogni livello.[5]

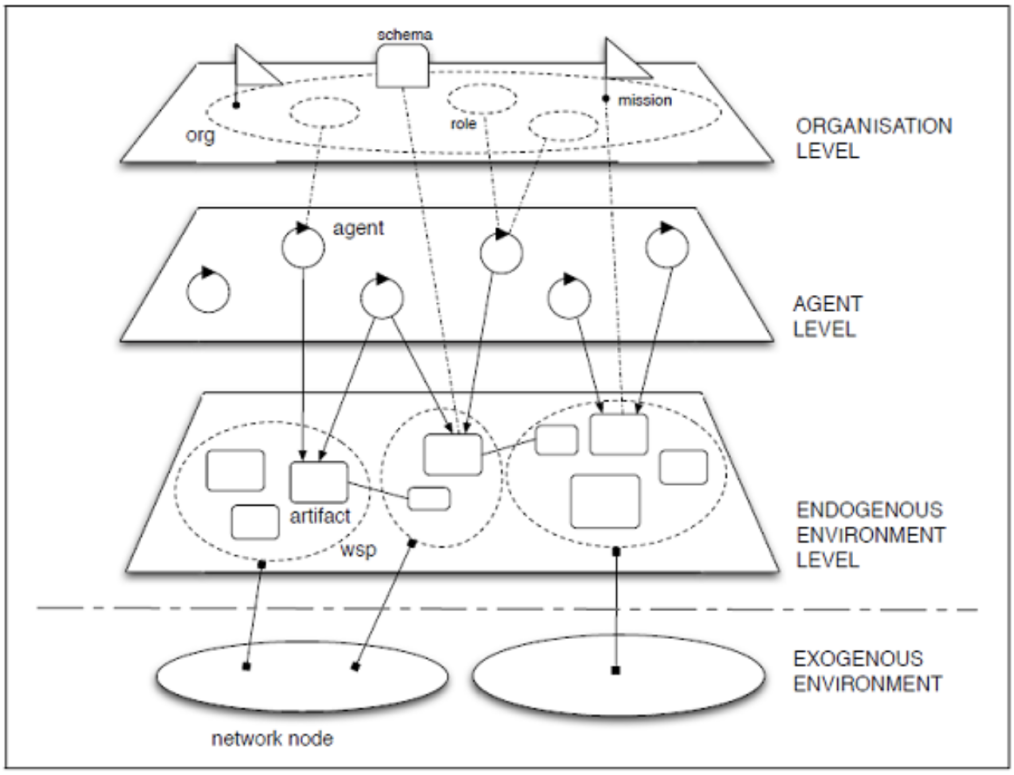

Figura 2.2: Livelli che compongono il framework JaCaMo[5]

Per apprendere questo framework, che non era stato affrontato durante il corso degli studi, sono stati utilizzati il sito [20] ed il paper di riferimento [5].

Jason

Jason è un interprete di AgentSpeak che implementa la semantica operazionale del linguaggio e fornisce una piattaforma di sviluppo per multi-agent systems, con molte funzionalità user-customisable. Il linguaggio e la piattaforma sono stati illustrati ed utilizzati, durante il corso di laurea, nell'insegnamento Sistemi Autonomi.

Le astrazioni appartenenti agli agenti, correlate al meta-modello Jason, sono principalmente ispirate all'architettura BDI sulla quale Jason è radicato. Quindi un agente è un'entità composta da un insieme di "beliefs", che

rappresentano lo stato corrente e la conoscenza dell'agente sull'ambiente in cui si trova, una serie di "goals", che corrispondono a compiti che l'agente deve eseguire / ottenere e una serie di "plans" ossia azioni (external action or internal action), innescate da eventi, che gli agenti possono comporre, istanziare ed eseguire dinamicamente per compiere i "goals".[6]

CArtAgO

Per quanto riguarda l'ambiente, ciascuna istanza di CArtAgO[1] è composta da una o più entità workspace. Ogni workspace è formato da un insieme di artefatti, che forniscono un insieme di operazioni e proprietà osservabili, definendo anche l'interfaccia di utilizzo degli artefatti.

L'esecuzione dell'operazione potrebbe generare aggiornamenti delle proprietà osservabili e degli eventi osservabili specifici. L'ultima entità relativa all'ambiente è il "manual" ed è utilizzata per descrivere le funzionalità fornite da un artefatto.

CArtAgO è basato sul meta-modello A&A (Agents & Artifacts) gli agenti, come entità computazionali, compiono qualche tipo di attività che mira a uno scopo e gli artefatti, come risorse e strumenti costruiti dinamicamente, sono usati e manipolati dagli agenti per supportare/realizzare le loro attività[26]. CArtAgo, come in precedenza Jason, era già stato parzialmente illustrato ed utilizzato durante il corso di Sistemi Autonomi, seppur in via superficiale. Gli approfondimenti necessari sono stati effettuati in autonomia facendo riferimento al materiale sullo stesso sito di JaCaMo.

Artefatto L'artefatto è un'entità reattiva, non autonoma, stateful, riutilizzabile, controllabile ed osservabile. Modellano strumenti, risorse e porzioni di ambiente agendo da strumenti mediatori di azioni e interazioni sociali tra partecipanti individuali e lo stesso ambiente.[21]

[1] Common ARTifact infrastructure for AGents Open environments

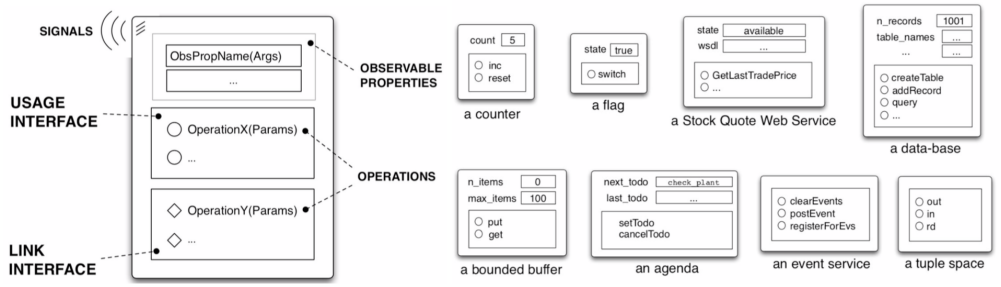

Figura 2.3: Struttura artefatto con relativi esempi

La modalità di interazione tra artefatto ed agente viene riepilogato nell'immagine sottostante.

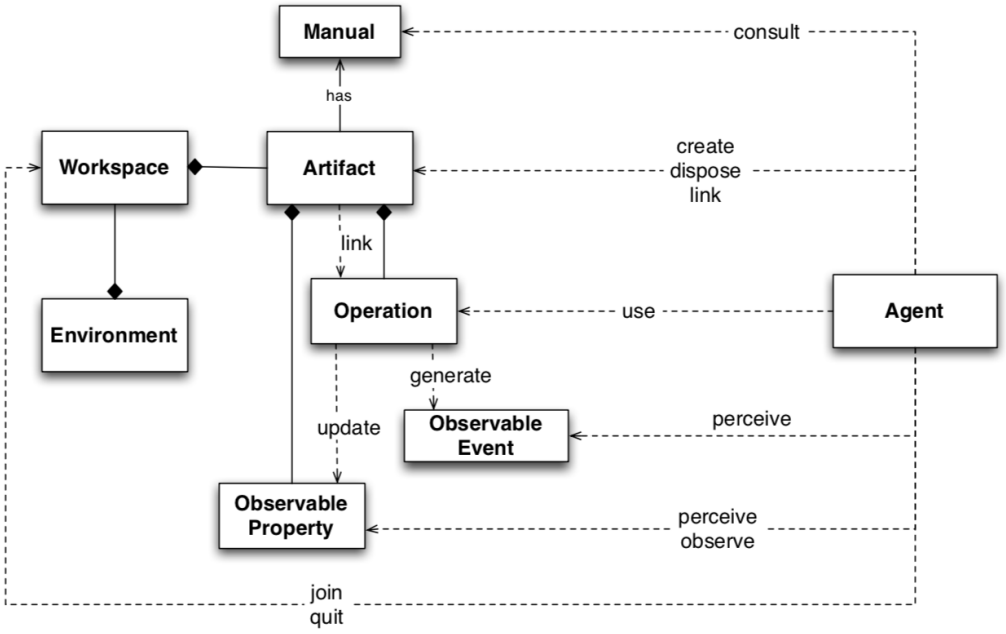

Figura 2.4: Interazione tra agente ed artefatto

Moise

Moise è un meta-modello organizzativo per MAS basato sulle nozioni di ruoli, gruppi e missioni. Abilita un MAS ad avere specifiche esplicite per la

sua organizzazione. Queste specifiche sono usate sia dagli agenti per ragioni inerenti la loro organizzazione, sia da una piattaforma organizzativa che si assicuri che gli agenti seguano le specifiche.

Moise permette di definire una gerarchia di ruoli con autorizzazioni e missioni, da assegnare agli agenti. Questo permette ai sistemi con un'organizzazione forte, di guadagnare proprietà di apertura (essenzialmente, la proprietà di lavorare con un numero e una diversità di componenti che non e imposta una volta per tutte) e adattamento.

Motivazioni

Come caso di studio si è deciso di utilizzare questo framework perché combina le tre tecnologie (di cui sopra) già affermate da diversi anni per lo studio e la produzione di software multi-agente rendendolo adatto alla realizzazione della mente di una generica entità. Jason e l'architettura DBI si prestano bene al dotare di intelligenza personaggi virtuali e CArtAgO è adatto ad integrare GE nei MAS come dimensione di ambiente.

2.2.2 Play Framework

Play è un framework lightweight, stateless e asincrono per la creazione di applicazioni e servizi Web. È stato costrutito utilizzando Scala e Akka e mira a fornire gli strumenti per la realizzazione di applicazioni altamente scalabili con consumo minimo di risorse (ad esempio CPU, memoria, thread).[15]

Play incorpora un HTTP Server integrato (quindi non è necessario un server di applicazioni Web separato come in molti Web Framework Java), un modello per la realizzazione di applicazioni baste su servizi RESTful e mette a disposizioni strumenti per la gestione di Form, protezione CSRF[2] e meccanismi di instradamento. Per semplificare il suo utilizzo fa largo uso del pattern Model-View-Controller, comune e facilmente utilizzabile, fornendo paradigmi di programmazione concisi e funzionali.

[2]Cross-Site Request Forgery

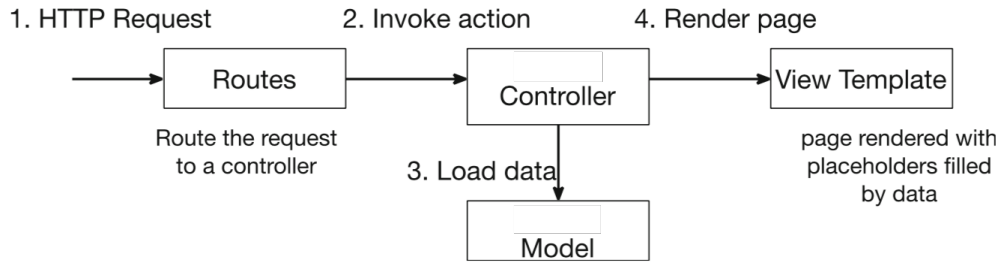

Figura 2.5: Struttura MVC di un'applicazione realizzata con Play[12]

Lo stack nelle Web Application nel mondo Java Enterprise è basato su una tecnologia che si è evoluta nel corso degli anni e richiede diversi elementi (strati) per funzionare. E' molto probabile che le molteplici tecnologie che comprendono questo stack rendano anche l'implementazione di semplici applicazioni problematica e soggetta a errori poiché ogni tecnologia deve essere integrata con successo con la successiva, spesso basandosi su file di configurazione o convenzioni standard.[12]

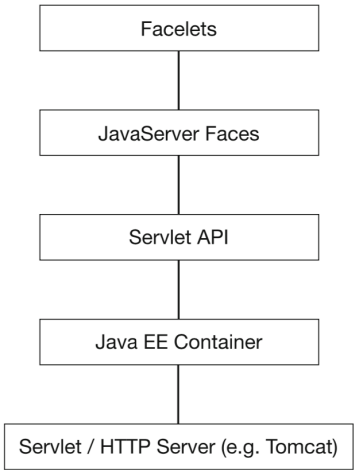

Figura 2.6: Architettura a strati JavaEE[12]

Il framework Play è stato progettato per diminuire lo stack sopra illustrato richiedendo l'utilizzo di un solo server HTTP per funzionare.

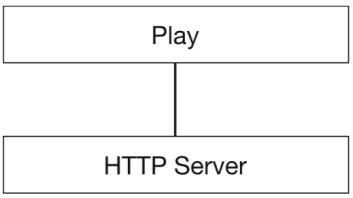

Figura 2.7: Architettura a strati Play framework[12]

Lo strato Play, in figura, è formato da una serie di componenti che includono:

- HTTP Server: componente che riceve la richiesta HTTP da un client e restituisce un risultato basato sulle informazioni fornite nella richiesta;
- Router: determina dove instradare la richiesta, pertanto fornisce un file di configurazione dei percorsi disponibili nell'applicativo;
- Sistema di templating HTML dinamico: Utilizza pagine standard in HTML e le popola con dati generati dinamicamente dall'applicazione;
- Console integrata: Per semplificare l'utilizzo di Play, viene fornita una suite di strumenti che possono essere utilizzati per creare, aggiornare e distribuire l'applicazione Play. Questi strumenti sono accessibili e gestiti dalla console;
- Persistent framework: Funzionalità utili per accesso a database.

Per apprendere al meglio questo framework, dato che non è rientrato tra gli argomenti affrontati durante il corso di studi, sono stati utilizzati il sito [15], la documentazione [14] e gli esempi disponibili [13].

Akka - Modello ad attori

Akka è un toolkit per la creazione di applicazioni altamente distribuite, concorrenti, event-driven, tolleranti ai guasti. Play framework utilizza il modello ad attori presente in Akka, dove l'attore è l'entità principale, per aumentare il livello di astrazione e fornire una piattaforma per la realizzazioni di applicazioni simultanee e scalabili [17]. Il modello ad attori, all'interno

del corso di studi, era già stato illustrato ed utilizzato nell'insegnamento di Sistemi Distribuiti.

Il modello ad attori (che risale al 1973) si basa sull'idea di avere attori simultanei indipendenti che ricevono e inviano messaggi asincroni e che svolgono un comportamento basato su questi messaggi. Gli attori possono mantenere il proprio stato e comportamento. Tuttavia, idealmente solo i dati immutabili vengono scambiati tra di loro, pertanto ogni attore è indipendente da tutti gli altri ed esegue solo alcuni calcoli o elaborazioni basati su un messaggio ricevuto da esso.

L'idea chiave alla base del modello ad attori è che la maggior parte dei problemi come concorrenza, deadlock, corruzione dei dati, derivino dalla condivisione dello stato. Pertanto, nel mondo degli attori non esiste uno stato condiviso (come una coda concorrente produttore-consumatore). Al contrario, i messaggi vengono inviati tra attori e questi messaggi vengono messi in coda in una casella di posta in modo simile ai messaggi di posta elettronica. Gli attori rispondono quindi ai messaggi appena arrivano a loro.[12]

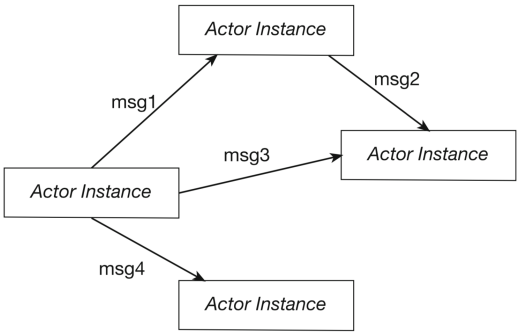

Figura 2.8: Comunicazione tra attori attraverso messaggi asincroni[17]

Motivazioni

Nel presente lavoro è stato utilizzato questo framework perché permette di realizzare un componente separato dal Sistema Multi-Agente e dalla Game Engine, ma allo stesso tempo facilmente collegabile a questi, attraverso richieste HTTP e/o canali WebSocket.

2.2.3 WebSocket

Il protocollo WebSocket (WS) consente la comunicazione bidirezionale tra un client a un host remoto instaurando un canale di comunicazione utilizzabile da entrambi sia in scrittura che in lettura. Il modello di sicurezza utilizzato è l'origin-based security model comunemente usato dai browser web. Il protocollo consiste in una hand-shake di apertura seguita da il successivo invio/ricezione di una serie di messaggi strutturati, stratificata su una connessione TCP[3] persistente. L'obiettivo di questa tecnologia è fornire un meccanismo per le applicazioni basate su browser che necessitano di comunicazione bidirezionale in tempo reale con server che non si basano sull'apertura di più connessioni HTTP [8]. Per comprendere struttura e funzionalità del protocollo WebSocket è stato utilizzato lo standard di riferimento [8] e le documentazioni delle librerie [27][23], entrambe RFC6455 compliant.

Introduzione delle WebSocket

Storicamente, la creazione di applicazioni Web che richiedono la comunicazione bidirezionale tra client e server (ad esempio applicazioni di messaggistica istantanea e giochi) ha portato ad un abuso di HTTP utilizzato per operazioni di polling (verifica ciclica) verso il server con lo scopo di controllare gli aggiornamenti, inviando notifiche upstream come chiamate HTTP distinte [18].

Il protocollo HTTP è stato inteso fin dalla prima versione ideata da Tim Berners-Lee come metodo per recuperare risorse remote in maniera semplice: una richiesta per ogni pagina Web, ogni immagine o l'invio di dati da rendere persistente. Con il passare degli anni però, all'incirca intorno al 2004, lo sviluppo di applicazioni Web subì una forte accelerazione dovuta all'introduzione di una nuova tecnologia, Ajax, che grazie all'utilizzo di Javascript fu in grado di creare e gestire richieste HTTP asincrone tramite funzioni di callback dedicate.

Seguendo l'evoluzione delle applicazioni Web molte applicazioni prevedevano una user experience orientata al real-time. Esempi possono essere applicazioni di chat, videogames multiplayer o sistemi di notifiche, tutte applicazioni che la sola tecnologia Ajax (o simili, come connessioni HTTP

[3]Transmission Control Protocol

persistenti COMET) poteva simulare solo in parte con sistemi di polling poco performanti e complessi da implementare.

La soluzione arrivò quando ci si rese conto che la risposta a questi problemi risiedeva effettivamente nel protocollo stesso: HTTP sfrutta a livello di rete il protocollo TCP/IP, connection-oriented, usata in altri contesti singolarmente per connessioni full-duplex. Nacque così il protocollo WebSocket con un ottimo tempismo considerando l'avvento contemporaneo di HTML5 e altre tecnologie dedicate all'open-source che contribuiranno successivamente alla diffusione del protocollo.[4]

Principali caratteristiche

Ecco le caratteristiche principali del protocollo WebSocket:

- **Bidirezionali**: quando il canale di comunicazione è attivo, sia il client che il server sono connessi ed entrambi possono inviare e ricevere messaggi;

- **Full-duplex**: i dati inviati contemporaneamente dai due attori (client e server) non generano collisioni e vengono ricevuti correttamente;

- **Basati su TCP**: il protocollo usato a livello di rete per la comunicazione è il TCP, che garantisce un meccanismo affidabile (controllo degli errori, re-invio di pacchetti persi, ecc) per il trasporto di byte da una sorgente a una destinazione;

- **Client-key handshake**: All'apertura di una connessione, il client invia al server una chiave segreta di 16 byte codificata con base64. Il server aggiunge a questa un'altra stringa, detta magic string, specificata nel protocollo ("258EAFA5-E914-47DA-95CA-C5AB0DC85B11"), codifica con SHA1 e invia il risultato al client. Cosi facendo, il client può verificare che l'identità del server che ha risposto corrisponda a quella desiderata;

- **Sicurezza origin-based**: Alla richiesta di una nuova connessione, il server può identificare l'origine della richiesta come non autorizzata o non attendibile e rifiutarla.

- **Maschera dei dati**: Nella trama iniziale di ogni messaggio, il client invia una maschera di 4 byte per l'offuscamento. Effettuando uno XOR bit a bit tra i dati trasmessi e la chiave è possibile ottenere il messaggio originale. Ciò è utile per evitare lo sniffing, cioè l'intercettazione di informazioni da parte di terze parti.

Motivazioni

Il protocollo appena illustrato è adatto a far comunicare corpo e mente per la peculiarità di instaurare un canale duraturo e bidirezionale nel quale entrambe le parti possono scambiarsi informazioni (azioni, percezioni). A vantaggio di questa tecnologia sono presenti librerie, ben documentate e realizzate seguendo lo standard RFC6455, facilmente integrabili su JaCaMo e Unity (illustrata nella sezione successiva) e nativamente inserite nel framework Play.

- **websocket-sharp**: Implementazione C# del protocollo WebSocket client/server[27]
- **project tyrus**: Implementazione Java dello standard JSR 356[4][23]

2.2.4 Unity

Unity è una Game Engine (GE) cross-platform sviluppata da Unity Technologies, utilizzata per la creazione di videogiochi (sia 2D che 3D) e simulazioni, che supporta la distribuzione su una larga varietà di piattaforme (PC, console, dispositivi mobili, etc.). Fornisce astrazioni che contribuiscono ad estendere il suo utilizzo tra gli sviluppatori e programmatori, rendendola una dei GE più utilizzate per produrre in maniera veloce ed efficace applicazioni e giochi.[28]

Inoltre, questa GE supporta molte funzionalità facili da utilizzare e sfruttabili per creare giochi realistici e simulazioni immersive, come un intuitivo editor real-time, un sistema di fisica integrato, luci dinamiche, la possibilità di creare oggetti 2D e 3D direttamente dall'IDE o di importarli esternamente, gli shader, un supporto per l'intelligenza artificiale (capacità di evitare gli ostacoli, ricerca del percorso, etc.), e cosi via. Per apprendere le funzionalità

[4] Java API for WebSocket, conforme al protocollo RFC6455

messe a disposizione da questa GE è stato fatto uso della documentazione ufficiale [29] e dei tutorial [30] messi a disposizione dalla stessa compagnia.

Le funzionalità principali messe a disposizione del designer sono:

- **GameObject**: La classe base per tutte le entità presenti su una scena di Unity: un personaggio controllabile dall'utente, un personaggio non giocabile, un oggetto (2D/3D). Tutto ciò che è presente sulla scena è un GameObject.
- **Script**: Codice sorgente applicato a un GameObject, grazie al quale è possibile assegnare a quest'ultimo comportamenti e proprietà dinamiche. Gli script vengono eseguiti dal game loop di Unity, che in maniera sequenziale esegue una volta ogni script, durante ogni frame del gioco. Non esiste concorrenza. Il comportamento è il risultato della logica definita nello script attraverso funzioni e routine. Le proprietà equivalgono a variabili che possono essere valorizzare nello script oppure definite dall'IDE grafico.
- **Component**: Elemento, proprietà speciale assegnabile ai GameObject. A seconda del tipo di GameObject che si desidera creare è necessario aggiungere diverse combinazioni di Components. I Components basilari riguardano la fisica (Transform, Collider,..), l'illuminazione (Light) e la renderizzazione del GameObject(Render). É possibile istanziare runtime Components attraverso gli script.
- **Coroutine**: Una soluzione alla sequenzialità imposta agli script, grazie al quale è possibile partizionare una computazione e distribuirla su più frame, sospendendo e riprendendo l'esecuzione in precisi punti del codice.
- **Prefab**: Rappresentazione di un GameObject complesso, completo di Script e Component, istanziabile più volte runtime. Le modifiche della struttura, proprietà e componenti del Prefab si propagheranno a tutti i GameObject collegati allo stesso presenti nella scena di gioco.
- **Event e Messaging System**: sistema ad eventi molto utile per far comunicare tra loro diversi GameObject. Questi sistemi sono formati tipicamente da eventi e listener. I listener si sottoscrivono ad eventi di un certo tipo; quando l'evento si verifica, viene notificato a tutti i listener in ascolto dello stesso tipo attraverso l'invio di un messaggio.

Motivazioni

In questo studio si è deciso di utilizzare questa Game Engine dato che è l'ambiente adatto alla realizzazione del corpo di una generica entità. In secondo luogo, anche per mantenere un collegamento con i lavori precedentemente realizzati dai colleghi[7][24][1][9]. Un'ulteriore motivazione per la quale è stato usato Unity riguarda la presenza di librerie, ben documentate, per integrare la tecnologia WebSocket.[27].

Capitolo 3

Design architetturale

Di seguito viene raffigurata la struttura del sistema che si realizzerà successivamente in fase di tesi.

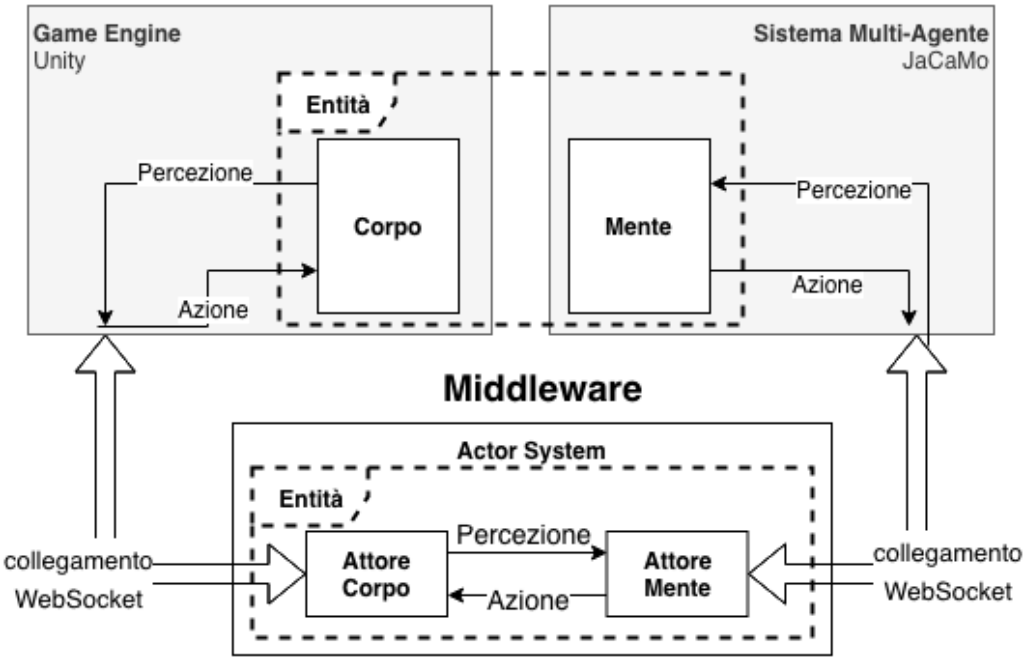

Figura 3.1: Design architetturale

Dallo schema è possibile notare come vengono utilizzate le tecnologie precedentemente illustrate:

- La Game Engine Unity è adatta ad ospitare il corpo di una generica entità;
- JaCaMo è adatto ad ospitare la mente di una generica entità;
- Il framework Play viene utilizzato per realizzare il middleware che collega i due sistemi;
- La reale connessione viene creata attraverso il protocollo WebSocket integrabile a JaCaMo, Unity e nativamente supportato nel framework Play.

Questa soluzione permette a mente e corpo di scambiarsi informazioni, anche se computazionalmente risiedono in sistemi separati, lasciando intatte funzionalità e astrazioni presenti nel Sistema Multi-Agente e nella Game Engine.

Con Unity viene "fisicamente" realizzato il corpo di una generica entità: utilizzando il concetto di GameObject ed attraverso i componenti, quali Collidere, Rigibody e script contenenti il collegamento WebSocket, è possibile dotare il corpo di funzionalità adatte a percepire l'ambiente. Ad esempio, in caso di contatto/urto con un diverso oggetto in scena, esso è capace di avvertire l'evento ed inviare alla mente una percezione del tipo *"toccato(nome_entità_toccata)"*.

Attraverso Jason viene realizzata la mente dell'entità utilizzando il concetto di agente unito ad un artefatto. Ciò permette di rappresentare il corpo dell'entità all'interno del MAS, interfacciandosi alla GE tramite l'artefatto che contiene la connessione WebSocket. Ad esempio, se la mente vuole far eseguire al corpo una determinata azione è sufficiente che utilizzi l'artefatto collegato per inviare un'azione del tipo *vai_a (posizione)*;

3.1 Scenario d'esempio

Per meglio comprendere l'architettura di sistema appena descritta, e le interazioni tra i suoi componenti costituenti, si prende a riferimento uno scenario d'esempio chiamato "recycling robots". Come si può intuire dal nome, la scena contiene dei robot, i quali hanno il compito di riciclare la spazzatura presente nell'ambiente portandola in un bidone.

Il compito generale di un robot è divisibile in un ciclo di sotto-obiettivi, ad esempio:

1. Cercare la spazzatura;
2. Andare verso la spazzatura trovata;
3. Prendere la spazzatura appena raggiunta;
4. Cercare il bidone
5. Andare verso il bidone trovato
6. Riciclare la spazzatura

Utilizzando il formalismo di Jason, i sotto-obiettivi elencati sono associabili a dei *"plans"* di un agente. Al di fuori del primo plan (Cercare la spazzatura), normalmente attivato dal *"goal"* principale presente nell'agente, i successivi possono essere attivati da una percezione ricevuta dall'agente. Ad esempio, in risposta alla ricerca del bidone è possibile notificare la scoperta di un bidone nelle vicinanze e, di conseguenza, fare in modo che l'agente utilizzi il plan "Andare verso il bidone trovato".

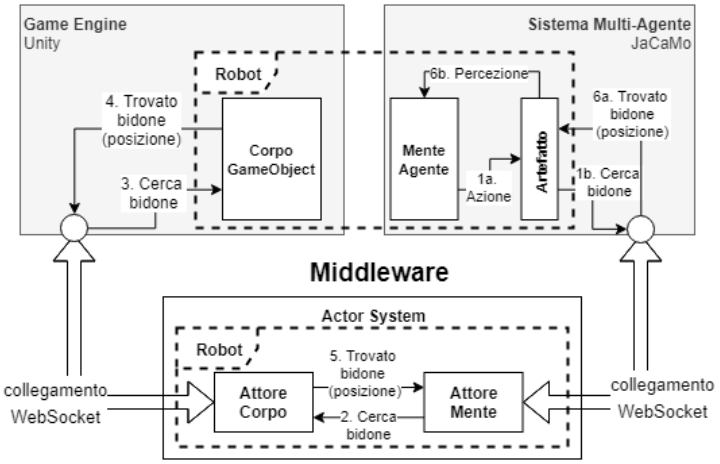

Figura 3.2: Esempio di comunicazione tra mente e corpo

L'immagine mostra il flusso ordinato di interazioni atteso per l'esempio appena descritto. La mente per svolgere il plan *"Cercare il bidone"* vuole

inviare al proprio corpo l'azione *"Cerca bidone"*. Il trasferimento del messaggio avviene attraverso l'artefatto personale dell'agente[1], in possesso del canale WebSocket collegato al middleware, dove è presente un'operazione che invia il messaggio al middleware.

L'attore mente presente nel middleware alla ricezione delle infomazioni da inviare al corpo, inoltra le stesse all'attore corpo che è l'unico possessore del riferimento all'entità corpo presente su Unity e, quindi, in grado di inoltrare il messaggio al corpo "reale". Quando su Unity l'entità corpo (GameObject) riceve il messaggio, attraverso il canale WebSocket, attua l'azione a lui richiesta e risponde alla mente inviando la percezione generata.

A questo punto la percezione viene mandata al middleware, più precisamente all'attore "corpo" che a sua volta la inoltrerà all'attore "mente", che la invierà poi all'artefatto collegato, sempre tramite WebSocket.

L'artefatto, alla ricezione della percezione, aggiunge quest'ultima alle sue proprietà osservabili che, automaticamente, aggiorneranno la BeliefBase dell'agente. L'ultimo passaggio rappresenta il punto cruciale per completare il collegamento tra corpo e mente dato che in questa maniera l'agente ha ricevuto la percezione dal proprio corpo.

Si intende inoltre lasciare aperta la possibilità, da parte del corpo, di inviare percezioni non come reazione ad azioni eseguite dalla mente, dato che il collegamento WebSocket, una volta effettuato, rimane attivo per tutta la durata di vita dell'entità. Ad esempio, in caso di contatto con un oggetto nella scena Unity, il corpo deve essere in grado di mandare una percezione del tipo *"toccata(nome_ oggetto,posizione)"* alla propria mente senza bisogno di stimoli.

Per concludere, il lavoro di attività propedeutica ha permesso di acquisire la necessaria conoscenza e dimestichezza con i concetti e le tecnologie da utilizzare per la successiva realizzazione del middleware in fase di tesi.

[1] previa associazione dei due

Elenco delle figure

Bibliografia

[1] A. Bagnoli. *Game Engines e MAS: Spatial Tuples in Unity3D*. PhD thesis, Alma Mater Studiorum - Università di Bologna, 2018.

[2] F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing multi-agent systems with JADE*, volume 7. John Wiley & Sons, 2007.

[3] J. Blair and F. Lin. An approach for integrating 3d virtual worlds with multiagent systems. In *2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications*, March 2011.

[4] BNG. WebSockets: A Guide. http://buildnewgames.com/websockets/.

[5] O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, and A. Santi. Multi-agent oriented programming with jacamo. *Science of Computer Programming*, 78(6), 2013. Special section: The Programming Languages track at the 26th ACM Symposium on Applied Computing (SAC 2011) & Special section on Agent-oriented Design Methods and Programming Techniques for Distributed Computing in Dynamic and Complex Environments.

[6] R. H. Bordini, H. J. Fred., and M. J. Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. Wiley, 2007.

[7] M. Cerbara. *Stato dell'arte della progettazione automatica di programmi per robot*. PhD thesis, Alma Mater Studiorum - Università di Bologna, 2017.

[8] I. Fette and A. Melnikov. The websocket protocol. RFC 6455, RFC Editor, December 2011. http://www.rfc-editor.org/rfc/rfc6455.txt.

[9] M. Fuschini. *Tecnologie ad Agenti per Piattaforme di Gaming: un caso di studio basato su JaCaMo e Unity.* PhD thesis, Alma Mater Studiorum - Università di Bologna, 2017.

[10] J. Gregory. *Game Engine Architecture.* CRC Press, 3. edition, 2019.

[11] I. Horswill. UnityProlog: A mostly ISO-compliant Prolog interpreter for Unity3D. https://github.com/ianhorswill/UnityProlog, Aug. 2015.

[12] J. Hunt. *A Beginner's Guide to Scala, Object Orientation and Functional Programming.* Springer International Publishing, Cham, 2018.

[13] L. Inc. Play examples. https://developer.lightbend.com/start/?group=play, 2018.

[14] L. Inc. Play framework documentation. https://www.playframework.com/documentation/2.7.x/Home, 2018.

[15] L. Inc. Play: The high velocity web framework for java and scala. https://www.playframework.com/, 2018.

[16] G. A. Kaminka, M. M. Veloso, S. Schaffer, C. Sollitto, R. Adobbati, A. N. Marshall, A. Scholer, and S. Tejada. Gamebots: a flexible test bed for multiagent team research. *Communications of the ACM*, 45(1), 2002.

[17] I. Lightbend. Akka: toolkit for building highly concurrent, distributed, and resilient message-driven applications for java and scala. https://akka.io/, 2019.

[18] S. Loreto, P. Saint-Andre, S. Salsano, and G. Wilkins. Known issues and best practices for the use of long polling and streaming in bidirectional http. RFC 6202, RFC Editor, April 2011. http://www.rfc-editor.org/rfc/rfc6202.txt.

[19] S. Mariani and A. Omicini. Game engines to model MAS: A research roadmap. In C. Santoro, F. Messina, and M. De Benedetti, editors, *WOA 2016 – 17th Workshop "From Objects to Agents"*, volume 1664 of *CEUR Workshop Proceedings*, pages 106–111. Sun SITE Central

Europe, RWTH Aachen University, 29–30 July 2016. Proceedings of the 17th Workshop "From Objects to Agents" co-located with 18th European Agent Systems Summer School (EASSS 2016).

[20] Olivier Boissier and Rafael H. Bordini and Jomi F. Hübner and Alessandro Ricci and Andrea Santi. JaCaMo Project. http://jacamo.sourceforge.net/.

[21] A. Omicini, A. Ricci, and M. Viroli. Artifacts in the a&a meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3), Dec 2008.

[22] A. Omicini and F. Zambonelli. MAS as complex systems: A view on the role of declarative approaches. In J. A. Leite, A. Omicini, L. Sterling, and P. Torroni, editors, *Declarative Agent Languages and Technologies*, volume 2990 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin Heidelberg, May 2004. 1st International Workshop (DALT 2003), Melbourne, Australia, 15 July 2003. Revised Selected and Invited Papers.

[23] Oracle Corporation. Project Tyrus. https://tyrus-project.github.io/index.html, 2017.

[24] N. Poli. *Game Engines and MAS: BDI & Artifacts in Unity*. PhD thesis, Alma Mater Studiorum - Università di Bologna, 2017.

[25] P. Prasithsangaree, J. Manojlovich, S. Hughes, and M. Lewis. Utsaf: A multi-agent-based software bridge for interoperability between distributed military and commercial gaming simulation. *SIMULATION*, 80(12), 2004.

[26] A. Ricci, M. Viroli, and A. Omicini. Cartago: A framework for prototyping artifact-based environments in mas. In D. Weyns, H. V. D. Parunak, and F. Michel, editors, *Environments for Multi-Agent Systems III*, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[27] STA. websocket-sharp: A C# implementation of the WebSocket protocol client and server. http://sta.github.io/websocket-sharp/, Oct. 2010.

[28] U. Technologies. Unity. https://unity.com/, 2019.

[29] U. Technologies. Unity documentation. https://docs.unity3d.com/Manual/index.html, 2019.

[30] U. Technologies. Unity learn. https://learn.unity.com, 2019.

[31] J. van Oijen, L. Vanhée, and F. Dignum. Ciga: A middleware for intelligent agents in virtual environments. In M. Beer, C. Brom, F. Dignum, and V.-W. Soo, editors, *Agents for Educational Games and Simulations*, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[32] Vocabolario Treccani. Treccani. http://www.treccani.it/vocabolario/.

[33] F. Zambonelli and A. Omicini. Challenges and research directions in agent-oriented software engineering. *Autonomous Agents and Multi-Agent Systems*, 9(3):253–283, Nov. 2004. Special Issue: Challenges for Agent-Based Computing.