EDITH
Prototipo per l'Ambiente di Lavoro nell'Industria 4.0

Enrico Gnagnarella
enrico.gnagnarella@studio.unibo.it

Matteo Scucchia
matteo.scucchia@studio.unibo.it

Febbraio 2019

Oggigiorno alcune grandi aziende come SAP Sistemi, stanno investendo sullo sviluppo di nuove tecnologie per l'industria 4.0, che mirano all'automazione assistita per il personale di quello che può essere un ambiente industriale. Ad esempio viene sempre più automatizzata l'assistenza nelle mansioni quotidiane degli operai all'interno dei magazzini. Mediante dispositivi wearable, i lavoratori vengono seguiti e indirizzati nello svolgimento dei loro compiti, risultando più efficienti e permettendo al sistema di essere distribuito e, nella quasi totalità, autonomo.

EDITH è un sistema di controllo dell'esecuzione dei compiti, distribuito e autonomo che consente di migliorare la produttività degli operai e quindi dell'intero magazzino, coordinando il carico di lavoro tra il personale, in base alla disponibilità dello stesso.

Indice

Elenco delle figure

1 Obiettivi e requisiti

1.1 Obiettivi

Il sistema mira ad automatizzare la distribuzione del carico di lavoro in un magazzino, inoltrando in modo autonomo le azioni da svolgere ai dipendenti. L'obiettivo è quello di creare, mediante un coordinamento tra agenti autonomi e personale umano, un sistema che permette di indirizzare, in maniera totalmente automatizzata, azioni da eseguire ad operai specifici in base alla loro disponibilità e produttività. Consentendo a tali operai di accettare l'ordine e inviare un feedback una volta che il compito viene portato a termine. In questo modo sarà possibile creare un ranking degli operai più produttivi e modificare di conseguenza l'algoritmo che inoltra gli ordini ai lavoratori, prediligendo quelli con punteggio maggiore in classifica.

Il sistema sarà basato su SPADE, per il quale quindi è necessario un approfondimento specifico. Tale framework verrà utilizzato per la gestione degli agenti e della loro comunicazione mediante server XMPP.

1.2 Requisiti

I requisti funzionali del sistema sono:

- Implementare un sistema dinamico, in modo da poter aggiungere ordini o entità senza dover interrompere il servizio;
- Implementare una comunicazione tra agenti che avverrà mediante messaggistica istantanea, servizio fornito dal server XMPP;
- Rendere il sistema quanto più possibile real-time, implementando gli agenti in modo che siano in grado di notificare eventuali cambiamenti in modo istantaneo gli altri agenti distribuiti.

I requisiti non funzionali, ma comunque necessari al fine del raggiungimento degli obiettivi sono:

- Studiare il framework SPADE e le astrazioni che tale middleware fornisce al fine di creare un sistema ad agenti distribuito e comunicante mediante server XMPP;
- Creare un server XMPP locale per permettere lo scambio di informazioni tra gli agenti, riservandoci la possibilià di utilizzare un server XMPP pubblico in caso di complicazioni;
- Fornire un'interfaccia al lavoratore, che può essere GUI o a riga di comando, per l'esecuzione di un ordine;
- Utilizzare un approccio test-driven, cominciando da subito a sviluppare test adeguati per monitorare il corretto funzionamento del sistema.

Il sistema aiuterà la collaborazione fra seguenti attori:

- Leader User, l'amministratore del sistema. Egli provvederà all'inserimento nel sistema degli ordini da svolgere e degli operai semplici;
- User, l'operaio semplice, il quale dopo aver ricevuto l'ordine tramite l'interfaccia apposita, avrà il compito di svolgerlo al suo meglio.

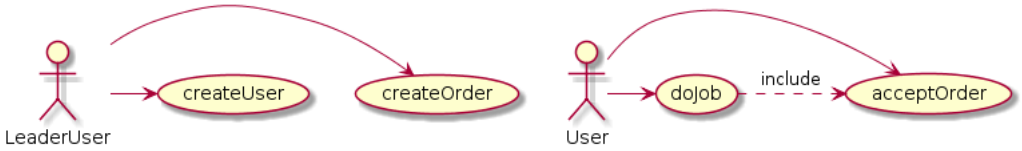

Figura 1: Diagramma dei Casi d'Uso.

1.3 Risultati attesi

Il risultato atteso è quello di avere un'applicazione che permetta la distribuzione dei compiti in modo autonomo. Saranno disponibili modalità di interfacciamento grafiche semplici ed intuitive che permetteranno all'amministratore di inserire nuovi ordini e nuovi lavoratori, permettendo allo stesso di monitorare lo stato del sistema, e agli operai di accettare ed eseguire gli ordini in arrivo, oltre che monitorare le proprie statistiche. Il gruppo si propone l'obiettivo di creare un sistema scalabile, che permetta l'inserimento di nuove entità senza dover interrompere il funzionamento del sistema stesso.

1.4 Scenari

EDITH è, come precedentemente accennato, un sistema di controllo dell'esecuzione dei compiti, distribuito e autonomo che consente di migliorare la produttività degli operai e quindi dell'intero magazzino, coordinando il carico di lavoro tra il personale, in base alla disponibilità dello stesso.

Per questi motivi, il sistema si colloca facilmente in qualsiasi ambiente lavorativo, il quale prevede la gestione di un magazzino e l'ottimizzazione di tale processo. Il sistema, come risultato del suo funzionamento, produce una maggiore efficienza nell'esecuzione degli ordini e quindi maggiore efficacia del processo aziendale in cui il sistema andrebbe a posizionarsi.

Il sistema prevede l'interfacciamento con due tipologie di utenti. In primis, vediamo l'amministratore di sistema, da noi chiamato *Leader user*, il quale disporrà di una semplice GUI, eseguibile su un qualsiasi device dotato di sistema operativo e un interprete Python, che permetterà di inserire nuovi operai nel sistema e aggiungere gli ordini da eseguire, assegnandogli un tempo minimo e una priorità d'esecuzione. Gli altri utenti

che usufruiranno del sistema sono gli operai, che d'ora in poi saranno denominati semplicemente *User*. Loro sono gli esecutori degli ordini, infatti ognuno disporrà di una piccola GUI, anch'essa eseguibile su un qualsiasi device dotato di interprete Python, che permetterà di ricevere gli ordini dal sistema, di eseguirli in un tempo maggiore al tempo minimo stabilito e terminarli, comunicando al sistema la rinnovata disponibità a ricevere ulteriori ordini.

EDITH si occuperà di tutta la gestione degli ordini e dello smistamento degli stessi fra i vari operai in modo efficiente, applicando determinate politiche basate sul ranking di produttività degli operai e sulla loro disponibilità.

1.5 Politica di autovalutazione

Il nostro progetto è stato sviluppato, attraverso il linguaggio di programmazione ad alto livello Python[6] e mediante il framework SPADE[1]. Non essendo mai venuti a contatto prima d'ora con tali sistemi, abbiamo progettato e sviluppato il sistema solo dopo aver preso conoscenza degli strumenti con i quali saremmo andati a lavorare.

Il progetto rispetta gli standard e le best-practice per quanto riguarda il linguaggio di programmazione e abbiamo approfondito e sviscerato al meglio la poca documentazione disponibile su SPADE. Tutto questo per realizzare un sistema ad agenti, che sfruttasse tutte le potenzialità che il framework offre, pur essendo ancora acerbo nel suo sviluppo e potenzialmente molto espandibile.

L'efficacia del progetto può essere verificata analizzandolo da due diversi punti di vista: il rispetto dei requisiti funzionali e dei requisiti non funzionali.

Il progetto è funzionante e rispecchia pressochè in toto i requisiti funzionali che ci eravamo prefissati. Il sistema è stato realizzato mediante l'uso di diverse tecnologie e si comporta correttamente nella sua interezza, anche se può presentare comportamenti indesiderati legati principalemente all'uso delle tencologie in questione come riportato nella Sezione 8.1. Gli agenti di fatto interagiscono nella maggior parte dei casi come dovrebbero e i behaviour sono stati implementati in modo tale da rendere la propagazione delle informazioni tra gli agenti distribuiti pressochè real-time.

Per quanto riguarda i requisiti non funzionali, è chiaro che lo sviluppo di tale progetto ci ha costretto ad uno studio approfondito di SPADE, delle diverse soluzioni XMPP disponibili e anche dello stesso Python.

2 Analisi dei requisiti

La fase di analisi dei requisiti è stata fondamentale, in quanto ci ha permesso di studiare a fondo le tecnologie da utilizzare per implementare al meglio il nostro progetto.

Il dubbio iniziale riguardava la scelta del framework da utilizzare per sviluppare il sistema ad agenti. Questa indecisione riguardava:

- JADE, JAVA Agent DEvelopment framework.
- SPADE, Smart Python Agent Development Environment.

2.1 Scelta della tecnologia

Il framework JADE, risultava inizialmente essere più appetibile in quanto era situato in un'ambiente di programmazione a noi già noto, inoltre eravamo già entrati in contatto con alcuni aspetti di questa tecnologia. Proprio per la facilità di un primo approccio a questa tecnologia, abbiamo preferito sfruttare questo progetto come una metafora per addentrarci a mondi a noi ancora sconosciuti. Infatti oltre al framework SPADE mai utilizzato, il progetto sarebbe stato vantaggioso per apprendere e conoscere meglio Python e le tecnologie XMPP. Nel paragrafo successivo verrà mostrato in modo più dettagliato il framework scelto.

2.2 Il framework SPADE

SPADE è un framework scritto in Python, a supporto della programmazione di Multi-Agents Systems. La comunicazione tra agenti è basata su protocollo XMPP, in quanto offre un'ottima architettura che permette agli agenti di comunicare in modo strutturato, risolvendo le principali problematiche che sorgono nella fase di progettazione di una piattaforma, come l'autenticazione degli utenti o la creazione di canali di comunicazione [1]. Il protocollo XMPP era precedentemente chiamato Jabber e si tratta di un protocollo basato su XML che implementa meccanismi di instant messaging e presence notification. SPADE è una piattaforma che rispetta interamente gli standard FIPA e inoltre mette a disposizione un'interfaccia web grafica per la gestione degli agenti.

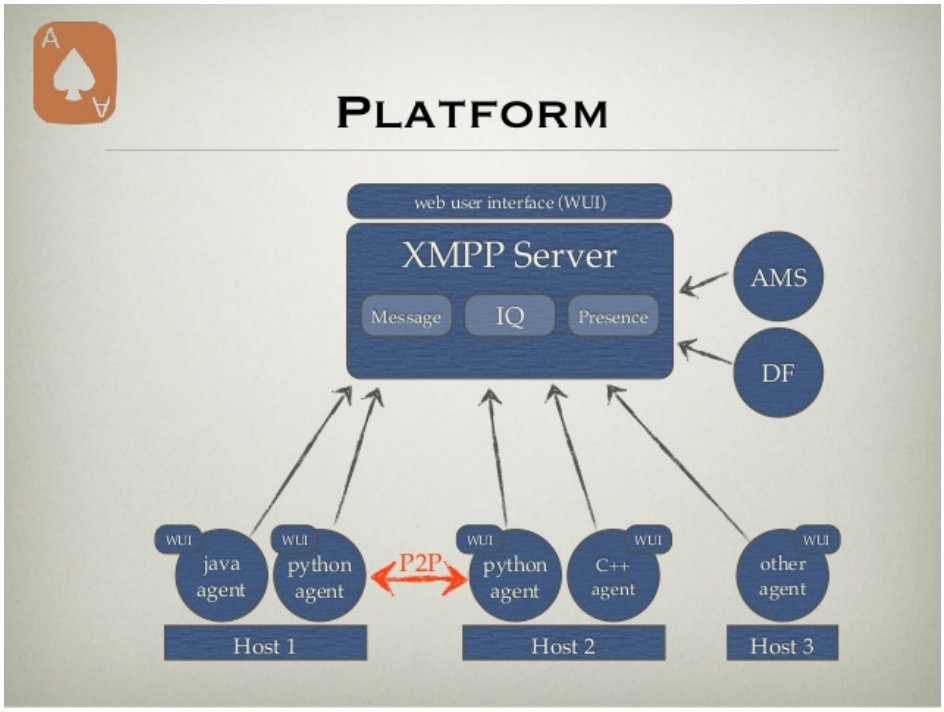

Figura 2: Piattaforma SPADE [4]

2.2.1 Agenti

Gli agenti sono l'entità principale di SPADE. Essi contengono dei meccanismi a supporto della connessione e dello scambio di messaggi oltre che un insieme di behaviors, ovvero i comportamenti che l'agente assume durante il suo ciclo di vita.

Le piattaforme ad agenti condividono queste caratteristiche, ma nello specifico SPADE le declina nel seguente modo:

- **Behaviors**: sono task che l'agente può eseguire in accordo con pattern forniti dal framework, ma con la possibilità di combinarli, sviluppandone di nuovi. Ogni agente può implementare più behaviours e possono esserne eseguiti più di uno in contemporanea. Ogni behaviour mantiene una propria mailbox per la ricezione dei messaggi;

- **Meccanismi di connessione al server**: ogni agente per poter essere registrato al server XMPP necessita di un JID(Jabber ID), un identificatore univoco composto dal nome dell'agente e dal dominio del server, scritto nella forma `nome@dominio`. SPADE fornisce meccanismi per la registrazione degli agenti al server e per la loro autenticazione. Una volta registrato un agente, viene creata tra l'agente stesso e il server un canale di comunicazione persistente;

- **Dispatcher**: ogni agente è munito di un dispatcher, un componente che smista i messaggi ricevuti dall'agente all'insieme di behaviours dello stesso che lo possono ricevere, in accordo con il loro Template.

2.2.2 Behaviours

I behaviours di un'agente vengono eseguiti ciascuno su un thread differente. I due comportamenti base che possono assumere gli agenti di SPADE sono CyclicBehaviour, un behavior che viene eseguito ciclicamente, e OneShotBehaviour, un behavior che viene eseguito una sola volta. Esistono poi behaviors più complessi, come FiniteStateMachineBehaviour che combina i due precedenti. Esso infatti estende CyclicBehaviour ed implementa degli stati, come una macchina a stati finiti, ognuno dei quali estende OneShotBehaviour. Altri behavior complessi sono:

- PeriodicBehaviour: estende CyclicBehaviour e permette di schedulare un comportamento, assegnandoli un periodo. Il suddetto comportamento viene eseguito ciclicamente, attenendo dopo ogni esecuzione un tempo pari al periodo;
- TimeOutBehaviour: estende OneShotBehaviour, è quindi eseguito una volta, ma dopo che è trascorso un determinato periodo di tempo;
- WaitingBehaviour: come nella sincronizzazione dei threads, questo comportamento permette di attendere la conclusione di un altro behaviour prima di essere eseguito.

2.2.3 Template

I Template sono il metodo con cui SPADE è in grado di eseguire il dispatch dei messaggi ricevuti dall'agente ai behaviours. Quando si aggiunge un behaviour all'agente è infatti possibile associargli un Template, ovvero una serie di attributi per cui, se corrispondono a quelli del messaggio ricevuto, lo stesso viene recapitato alla mailbox del behaviour. Tale funzionalità è molto comoda quando un agente può ricevere messaggi da molti agenti differenti e deve gestirli in modo differente a seconda del mittente.

2.2.4 Presence notification

Le presence notifications sono una peculiarità di SPADE, infatti gli agenti sono in grado di mantenere una lista di contatti sempre aggiornata e questo è possibile grazie alla tecnologia di messaggistica istantanea realizzata tramite server XMPP.

Ogni agente è dotato di un PresenceManager, un oggetto che mantiene aggiornato lo stato di presenza dell'agente. Il PresenceManager nello specifico gestisce:

- Lo stato: può quindi essere impostato come disponibile o non disponibile a seconda delle circostanze, specificando in che modo è disponibile l'agente;
- Lo status: una descrizione in linguaggio naturale riguardante le informazioni sulla presenza dell'agente;

- La priorità: definisce l'importanza della connessione al server XMPP.

Ogni agente possiede un handler chiamato AvailabilityHandler che mediante callbacks permette di gestire un contatto quando questo risulta disponibile o non disponibile.

Un meccansimo importante è quello della **sottoscrizione**, ovvero è possibile mandare informazioni specifiche riguardo alla propria presenza ad altri agenti, richiedendo una sottoscrizione. Come per la disponibilità, è presente un handler dedicato chiamato SubscriptionHandler che permette di gestire mediante callbacks un contatto che richiede di essere sottoscritto o rimosso dalla sottoscrizione.

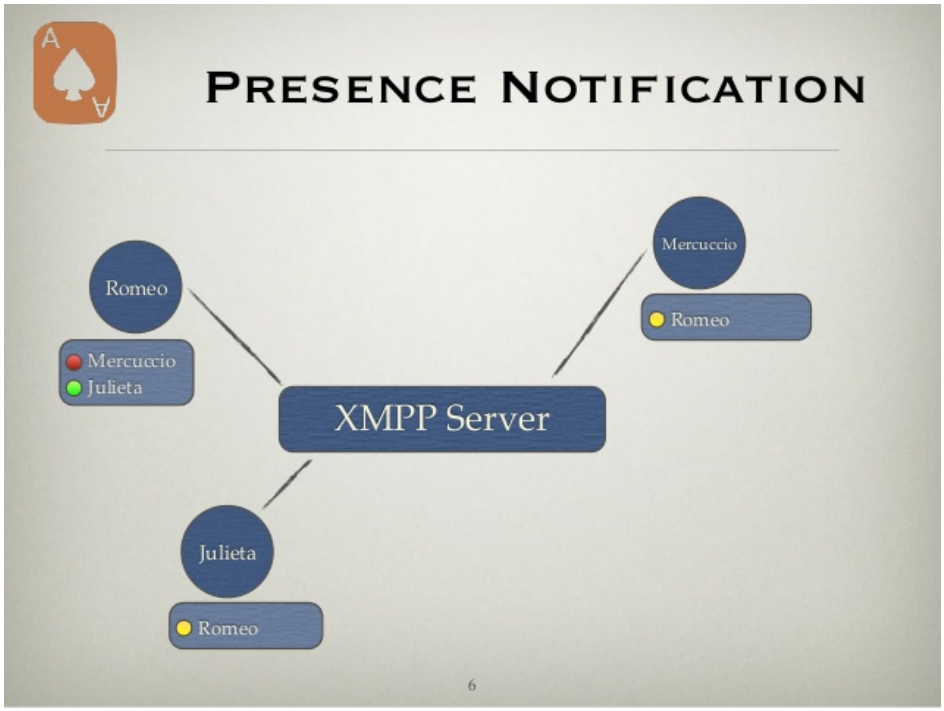

Figura 3: Presence notification [4]

3 Design

In questa sezione si analizza il sistema EDITH sotto le tre dimensioni principali. In primo luogo si discuteranno quali sono i concetti e le entità rappresentate. Secondariamente, si andrà ad analizzare il comportamento delle singole entità ed infine verranno presentate le modalità specifiche con cui queste interagiscono.

3.1 Struttura e comportamento

EDITH è un sistema agent-based, per cui ogni compomente principale del sistema è stato strutturato come un agente. Ogni agente sarà quindi dotato di tutti i metodi che il framework fornisce per la definizione degli agenti, implementerà i behaviour necessari al fine di rispettare i requisiti funzionali e farà uso dei meccanismi di SPADE per la comunicazione con gli altri agenti.

Il sistema basa il suo funzionamento sull'interazione dei seguenti agenti:

- Central Order Agent
- Highway Agent
- GUI Agent
- Statistical Agent
- Temporal Agent

Ogni agente svolge un preciso compito e interagisce con uno o più agenti differenti, in modo da permettere un corretto funzionamento del sistema. All'interno del sistema sono presenti una sola istanza di ogni agente sopra elencato, ad eccezione del GUI Agent. Per questo agente, il numero di istanze corrisponde al numero di operai al lavoro e può variare durante l'esecuzione.

3.1.1 Central Order Agent

Questo agente ha lo scopo principale di ricevere tutti gli ordini inseriti da parte dell'amministratore, memorizzarli internamente e ordinarli in base alla priorità di esecuzione assegnatagli. Di seguito, l'agente provvede a distribuire gli ordini all'Highway Agent, in modo che quest'ultimo possa poi smistarli fra gli operai. Il Central Order Agent mantiene al suo interno anche uno storico di tutti gli ordini inoltrati dalla messa in funzione del sistema.

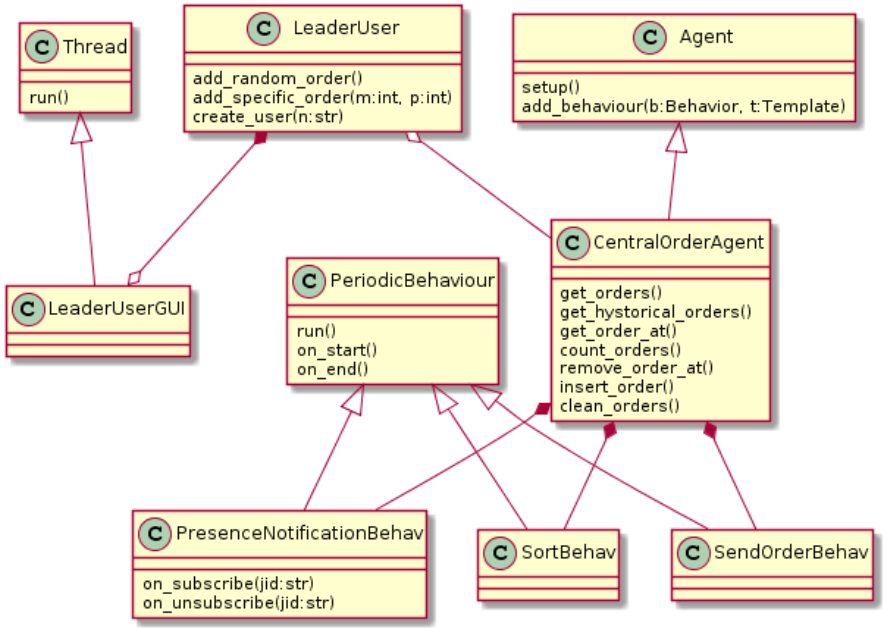

Figura 4: Central Order Agent - Diagramma delle Classi

3.1.2 Highway Agent

Questo agente, una volta ricevuti gli ordini dal Central Order Agent, effettua la distribuzione fra gli operai, quindi nello specifico fra i GUI Agent. L'Highway Agent smista gli ordini secondo due criteri in questa versione del progetto, che sono i seguenti:

- Simple Switching, tecnica che prevede che gli ordini vengano assegnati uno dopo l'altro al primo operaio disponibile.
- Complex Switching, tecnica per cui si analizza il ranking degli operai, definito dall'efficienza nell'eseguire gli ordini, e la loro disponibilità a svolgere il lavoro.

In più mantiene la disponibilità degli operai e il ranking sempre aggiornato, in quanto comunica costantemente con gli agenti dediti a questi scopi.

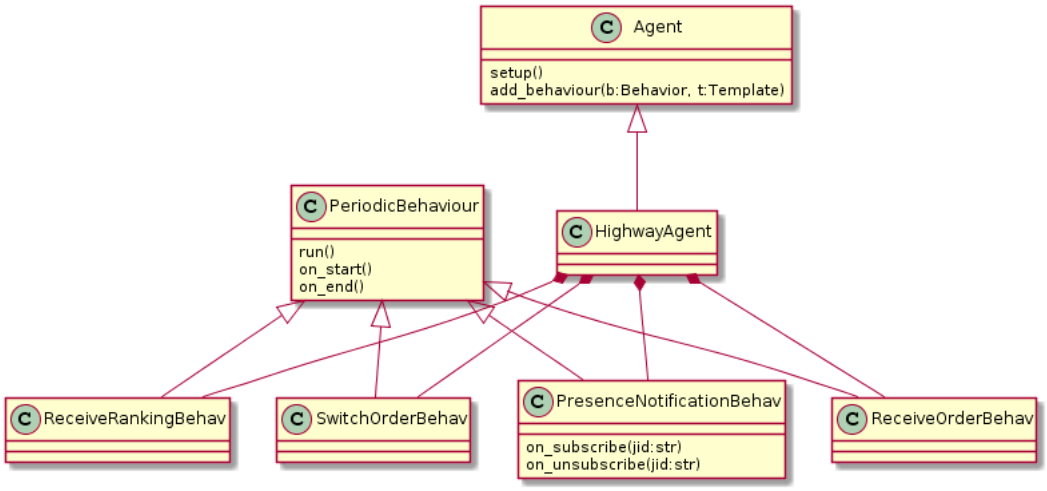

Figura 5: Highway Agent - Diagramma delle Classi

3.1.3 GUI Agent

Questo agente ha un compito piuttosto semplice, quello di permettere all'utente di eseguire gli ordini, ma allo stesso tempo una rete di comunicazione complessa in quanto deve comunicare con lo Statistical Agent, l'Highway Agent e il Temporal Agent. Il GUI Agent rappresenta la componente di controllo della GUI vera e propria, facendo un paragone con il pattern MVC si potrebbe dire che un GUI Agent rappresenta il Controller di una vera e propria GUI, che rappresenta la parte di View. Un GUI Agent riceve gli ordini dall'Highway Agent e aggiorna il suo stato. La GUI vera e propria mostra a schermo le componenti grafiche a seconda dello stato del rispettivo agente, permettendo l'interfacciamento con l'utente. L'utente tramite questo agente, avrà poi modo di accettare un ordine in arrivo, eseguirlo e notificare la terminazione una volta eseguito. Il GUI Agent provvederà poi alla distribuzione dei dati agli altri agenti sopra elencati, in modo da aggiornare il ranking e la rinnovata disponibilità ad accettare nuovi ordini.

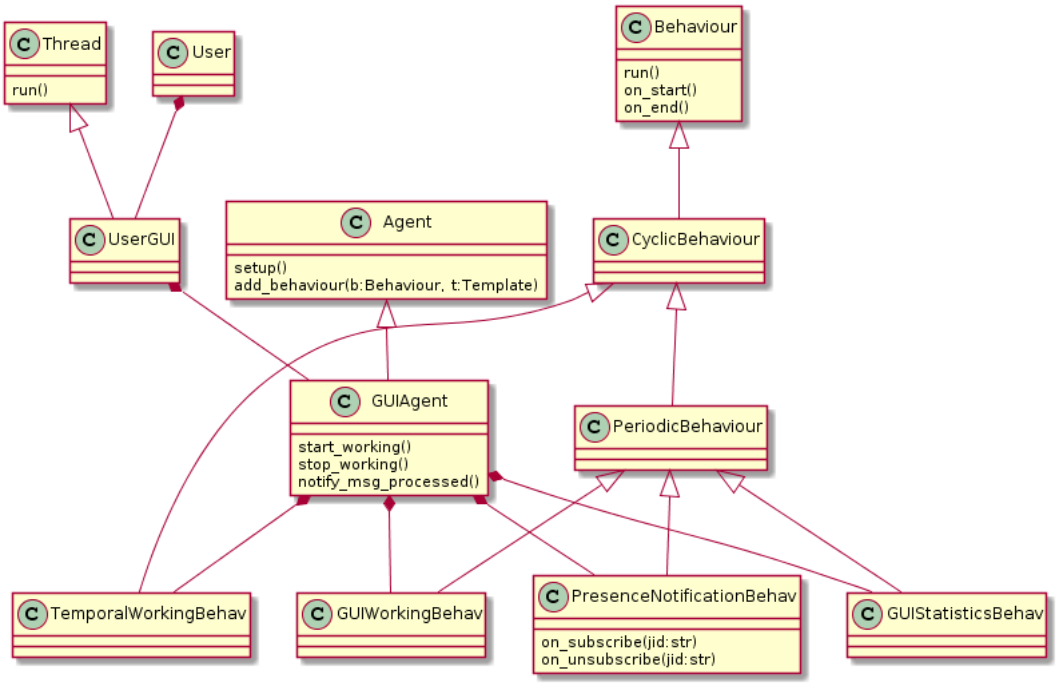

Figura 6: GUI Agent - Diagramma delle Classi

3.1.4 Statistical Agent

Questo agente è dedicato all'elaborazione statistica del sistema. Esso viene costantemente informato quando un ordine è stato eseguito e calcola quindi uno score sulla base dell'efficienza di esecuzione. Successivamente stila una classifica, un ranking, dei GUI Agents, e quindi degli User, che sono più produttivi. Inoltra poi il ranking all'Highway Agent e ad ogni GUI Agent manda la specifica posizione nel ranking oltre che altre statistiche aggregate come il tempo medio di esecuzione degli ordini, lo score totale, il numero di ordini eseguiti e quelli non correttamente eseguiti.

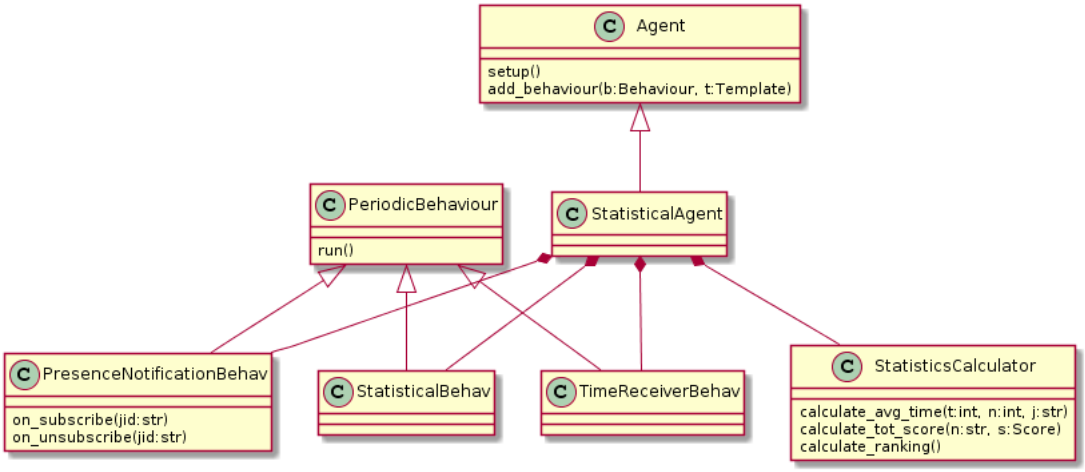

Figura 7: Statistical Agent - Diagramma delle classi.

3.1.5 Temporal Agent

Questo agente svolge la funzione di timer del sistema. Esso mantiene il tempo in due diversi formati: un formato timestamp e un formato intero in secondi. Il formato timestamp viene utilizzato dallo Statistical Agent per mantenere uno storico preciso del ranking, permettendo di capire come questo è cambiato nel tempo, mentre il formato intero viene utilizzato dai GUI Agents. Quando un utente accetta l'ordine, il GUI Agent corrispondente e il Temporal Agent si sottoscrivono, in questo modo quest'ultimo inizia a mandare messaggi al GUI Agent ogni secondo, permettendo quindi a questo di aggiornare il tempo di esecuzione dell'ordine.

Facendo riferimento ai contenuti del corso, il Temporal Agent modella sia il tempo fisico, che il tempo logico del sistema. Il tempo fisico è dato dal timestamp, che fa riferimento al servizio UTC, mentre il tempo in secondi è un tempo logico, che serve solamente per ordinare gli eventi.

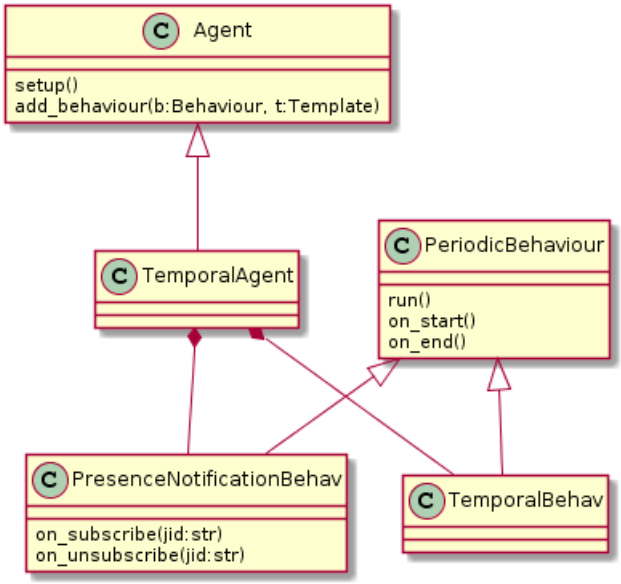

Figura 8: Temporal Agent - Diagramma delle classi.

3.1.6 Modularizzazione delle entità

Risalendo ad un livello di astrazione superiore a quello appena trattato, abbiamo strutturato il progetto in package, raggruppando fra loro le classi con aspetti comuni. Come si può vedere nella figura sottostante, sono stati definiti i seguenti package:

- Agent, contente l'implementazione di tutti gli agenti presenti nel sistema e citati nel paragrafo 3.1.
- Gui, racchiude le implementazioni delle due interfacce grafiche sviluppate per gli utenti del sistema.
- User, raccoglie le classi che modellano lo User e il LeaderUser.
- Statistics, qui sono contenute tutte le classi necessarie per la gestione delle statistiche e che sono utilizzate dallo Statistacal Agent e non solo.
- Utils, qui sono racchiuse tutte le utility necessarie al funzionamento di EDITH.

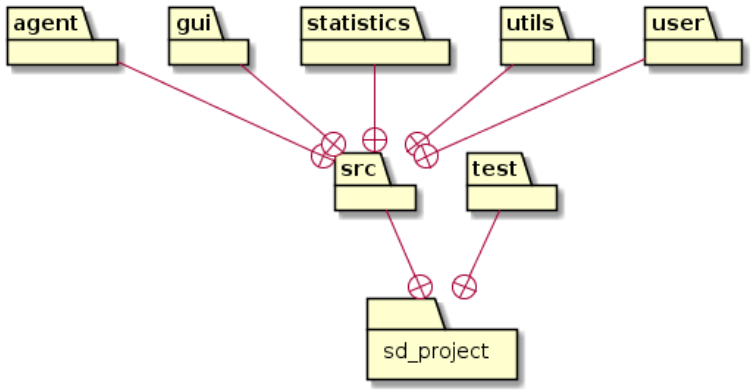

Figura 9: Diagramma dei Package

3.2 Behaviours

I behaviours implementati dagli agenti del progetto sono pressochè tutti di tipo periodico, poichè sono task che devono essere eseguiti ripetutamente per tutto il tempo in cui il sistema è attivo. L'unico behaviour non periodico è implementato dalla classe TemporalWorkingBehav appartenente al GUIAgent, che attua un comportamento ciclico. Un comportamento ciclico si comporta sostanzialmente come uno periodico, ma quest'ultimo viene eseguito ciclicamente a distanza di un tempo prefissato, mentre il comportamento ciclico viene eseguito continuamente. Tale classe implementa infatti la ricezione da parte del GUIAgent attivo, in stato di esecuzione dell'ordine, del tempo inviatogli dal TemporalAgent. Per cercare di rendere quindi l'inizio della comunicazione più reattiva ed evitare il rischio di non considerare nel conteggio il primo secondo di lavoro, rendendo il sistema più impreciso, si è deciso di implementare un behaviour ciclico che quindi lo rende pressochè sempre attivo, con un timeout interno che lo mantiene in attesa della ricezione del messaggio.

Di seguito riportiamo i diagrammi di Attività di alcuni behaviours che ci sembrano particolarmente degni di nota.

3.2.1 Central Order Agent Behaviours

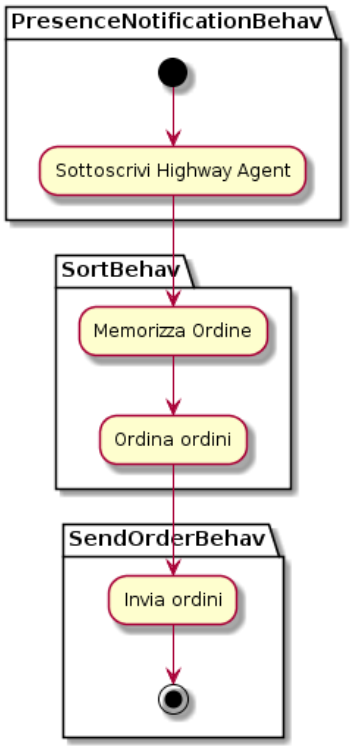

Figura 10: Central Order Agent - Diagramma delle Attività

3.2.2 Highway Agent Behaviours

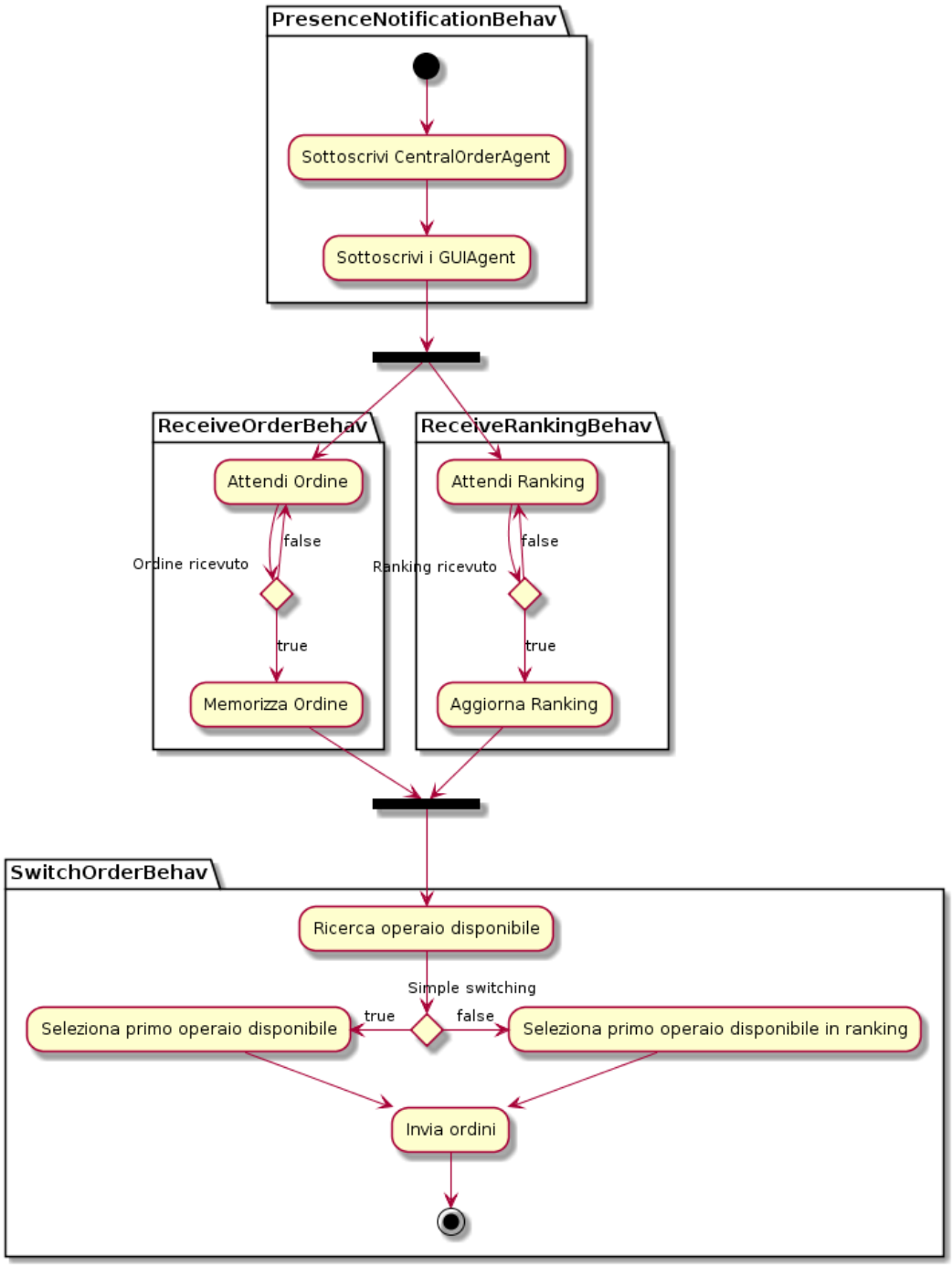

Figura 11: Highway Agent - Diagramma delle Attività

3.2.3 GUI Agent Behaviours

Per semplicità di visualizzazione, in questo diagramma non è modellata la gestione delle sottoscrizioni al Temporal Agent.

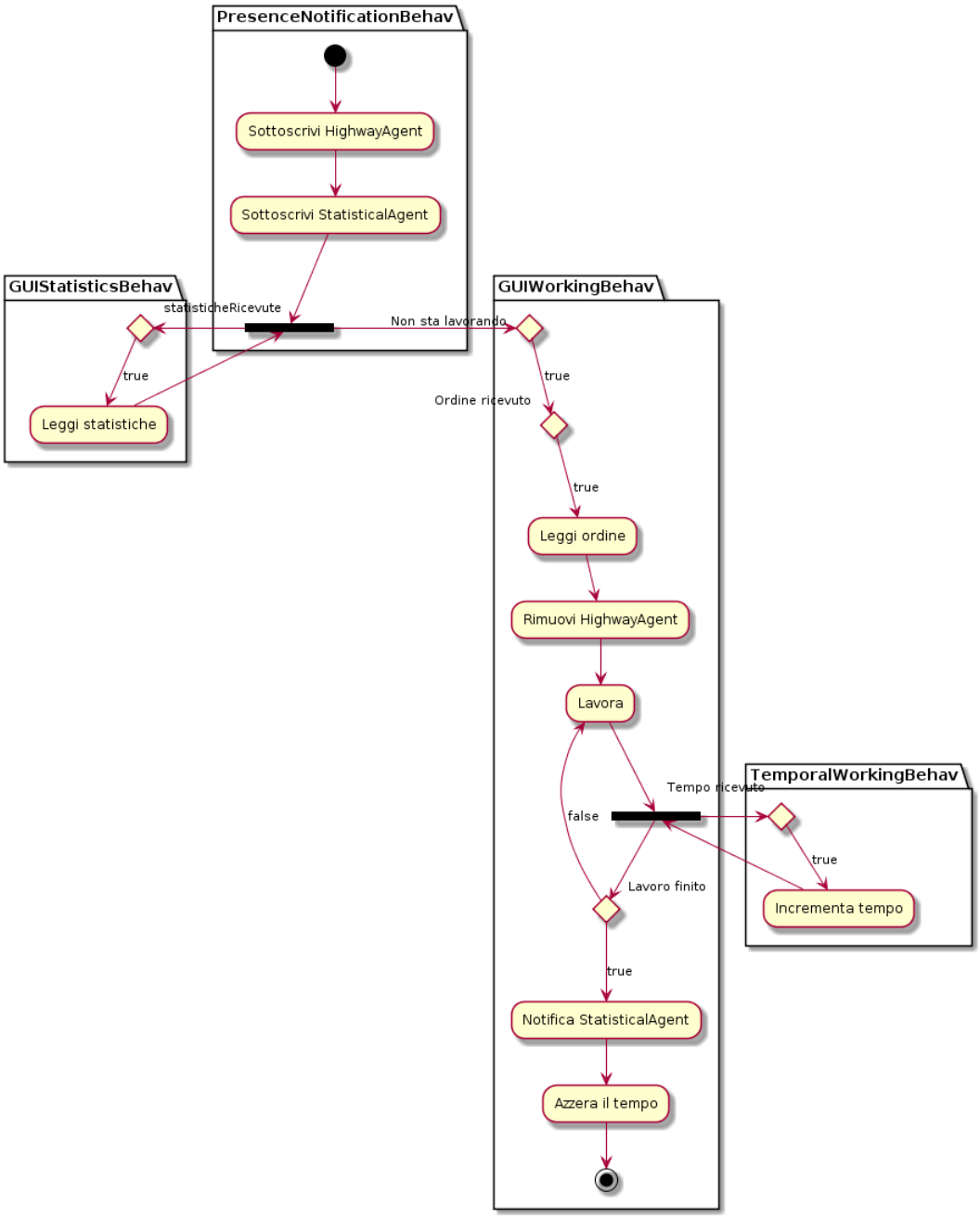

Figura 12: GUI Agent - Diagramma delle Attività

3.2.4 Statistical Agent Behaviours

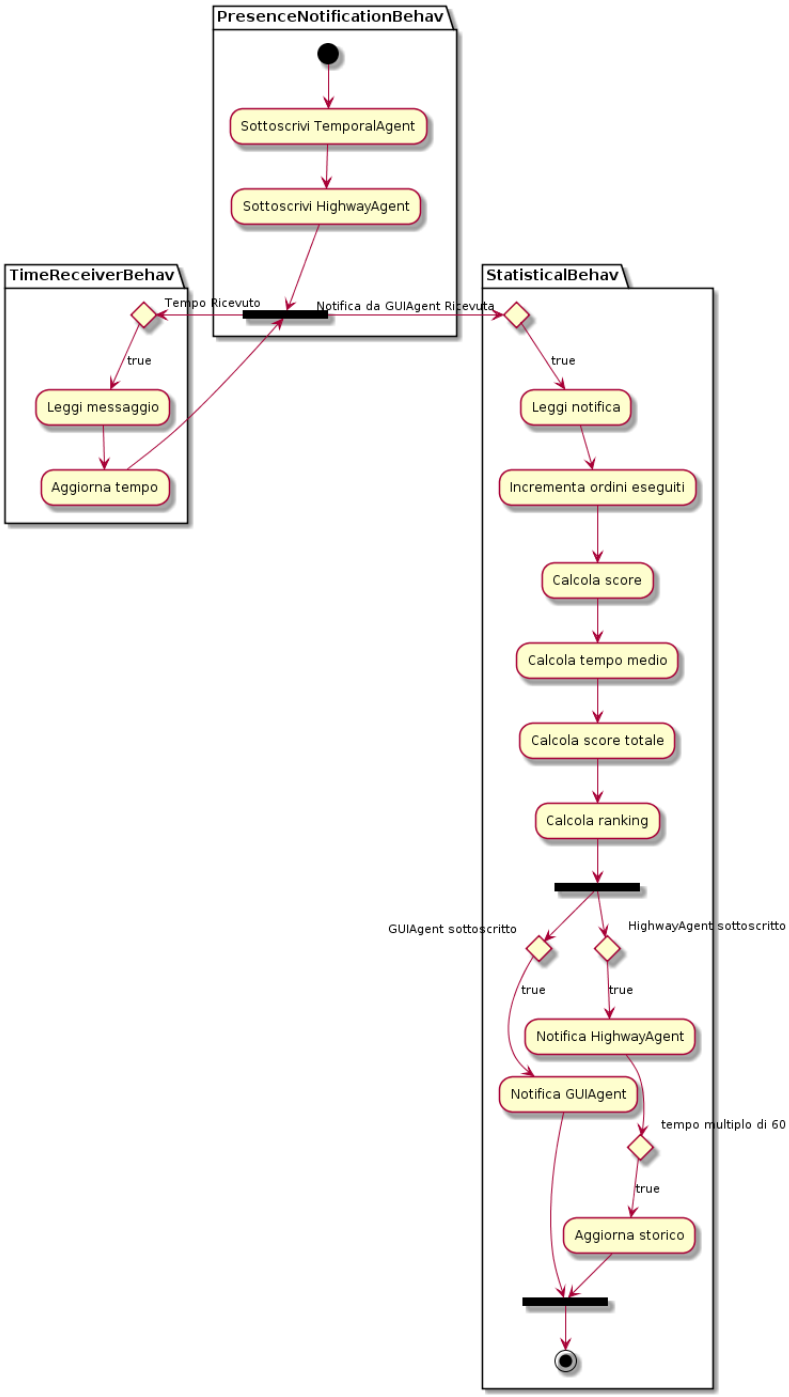

Figura 13: Statistical Agent - Diagramma delle Attività

3.2.5 Temporal Agent Behaviours

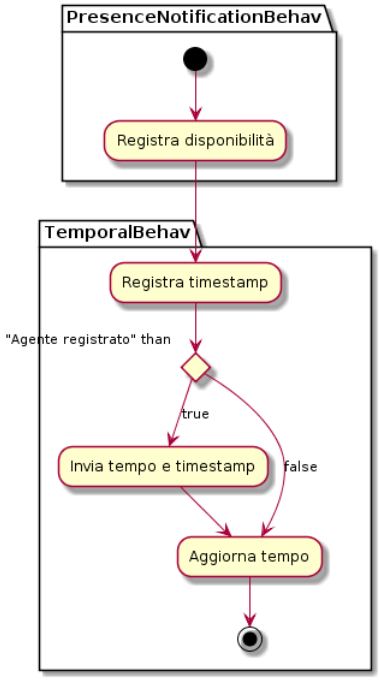

Figura 14: Temporal Agent - Diagramma delle Attività

3.3 Interazioni

Gli agenti interagiscono fra loro, sfruttando le astrazioni di messaggistica fornite dal framework SPADE. Per rendere possibile tutto ciò, sono state utilizzate le Presence Notification, illustrate dettagliatamente nel Capitolo 2.2.4. Queste permettono la sottoscrizione fra i vari agenti, ovvero ogni agente può sottoscriversi ad un'altro in modo da rendersi disponibile ai messaggi che esso spedisce. Dopo la sottoscrizione attraverso i metodi forniti dal framework, ogni agente dispone di una *Contact List*, nel quale sono presenti tutti gli agenti attualmente sottoscritti. L'agente a questo punto è a conoscenza di tutti gli utenti con il quale poter scambiare dei messaggi. Questo fase di sottoscrizione è stata implementata in tutti gli agenti, che necessitavano di scambiare messaggi fra loro.

La fase successiva prevede l'utlizzo della classe *Message* di SPADE per lo scambio dei messaggi, dove abbiamo abbinato l'uso dei *Template*. Questi ultimi sono uno strumento del framework, che ci è servito per reindirizzare i messaggi destinati ad un agente al behaviour interessato, in quanto gli agenti possono dover scambiare messaggi con più agenti contemporaneamente. Il loro funzionamento è spiegato in dettaglio nel Capitolo 2.2.3.

Nello schema sottostante è mostrato come avviene l'inizializzazione e la creazione degli utenti nel sistema, del loro avatar virtuale e della loro interfaccia, a seguito dell'attività del LeaderUser. A supporto della creazione degli agenti viene utilizzata AgentFactory, descritta in dettaglio nella Sezione 4.2.

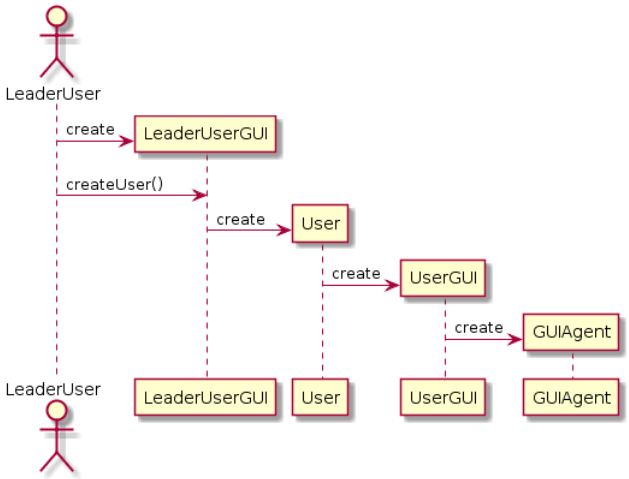

Figura 15: Diagramma di sequenza della creazione degli utenti.

Le interazioni fra i vari agenti sono di seguito riportate in modo dettagliato. Tale diagramma rappresenta come interagiscono gli agenti e gli attori del sistema, una volta che gli utenti sono stati creati e il sistema stesso è avviato. Per una questione di leggibilità, il comportamento del Temporal Agent è stato modellato solo parzialmente, in quanto

esso in realtà invia ciclicamente ogni secondo un aggiornamento del tempo allo Statistical Agent, cosa che di seguito non è rappresentata. Inoltre per come è stato costruito il sistema, tutti gli agenti che si scambiano i messaggi sono sottoscritti nella reciproca lista di contatti, ma nella figura sotto si modella solo la sottoscrizione del GUI Agent con i corrispettivi agenti, poichè è l'unica che viene modificata dinamicamente durante l'esecuzione.

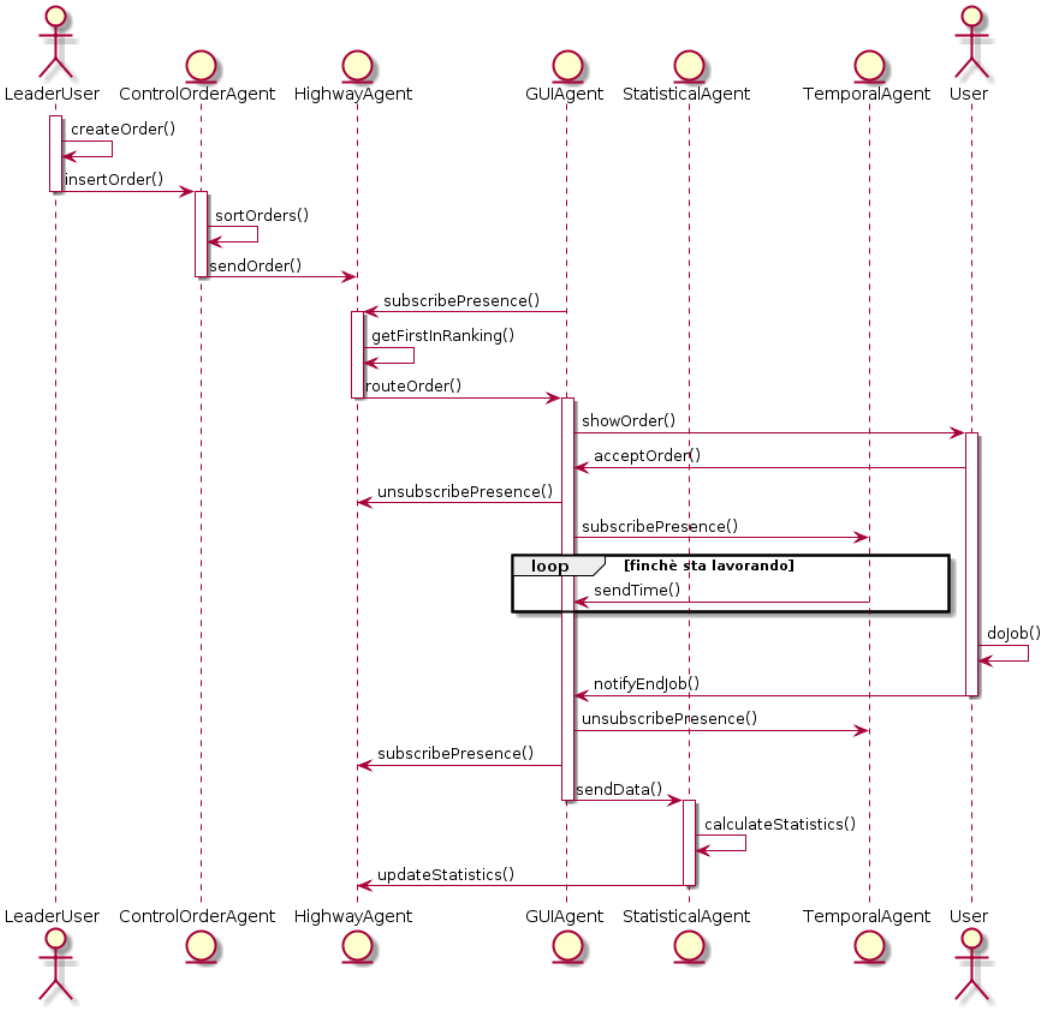

Figura 16: Diagramma di sequenza delle interazioni fra agenti ed attori.

4 Dettagli Implementativi

Il codice prodotto per il progetto è stato correttamente documentato, cercando di attenersi il più possibile agli standard della PEP 257 - Docstring Convention [5]. Ogni dettaglio e specifica implementativa è quindi consultabile direttamente sul codice sorgente.

4.1 Struttura del progetto

Per quanto riguarda la struttura del progetto, riportiamo le seguenti scelte implementative come degne di nota:

- User e GUI: abbiamo implementato l'utente come una normale classe Python, il quale però si compone di una GUI, chiamata UserGUI, la quale a sua volta si compone di un GUI Agent. Questa ci è sembrata la soluzione concettualmente più corretta;
- LeaderUser e GUI: in analogia con l'utente, il LeaderUser è una classe Python che si compone di una GUI, ma in questo caso per rendere più semplice l'interazione tra LeaderUser, GUI e Central Order Agent abbiamo deciso che anche la GUI avesse in aggregazione un LeaderUser. Inizialmente la logica era invertita, ovvero era la LeaderGUI che si componeva di un LeaderUser, cosa che abbiamo trovato concettualmente sbagliata. L'implementazione finale ci sembra un giusto compromesso fra le cose;
- Ordine: gli ordini sono modellati da una classe Python che mantiene il tempo minimo di esecuzione dell'ordine e la priorità dell'ordine, intesa anche come fattore di punteggio. In questo modo la simulazione dell'esecuzione di un ordine viene realizzata mediante un utente che, dopo aver accettato lo stesso, attende un tempo superiore al tempo minimo per poter poi confermare la corretta esecuzione del compito. Se infatti l'utente conferma la terminazione dell'esecuzione dell'ordine in un tempo inferiore al tempo minimo, lo score per quell'ordine sarà negativo, andando a penalizzare l'utente.

4.2 Design Pattern

Per rendere il progetto allo stato dell'arte sono stati introdotti due diversi design pattern. Nello specifico si è fatto uso di:

- pattern Factory: per l'istanziazione degli agenti si fa uso di tale pattern, che implementa i factory method, ognuno dei quali permette la creazione di una particolare tipologia di agente;
- pattern Strategy: all'interno dello Statistical Agent viene utilizzata una classe esterna, lo Statistics Calculator, a cui sono delegati i calcoli per ottenere i dati statistici aggregati;

5 Autovalutazione / Validazione

Per la validazione del progetto non sono stati utilizzati specifici criteri formali, bensì un'insieme di test automatizzati da noi prodotti, essendo un requisito quello di essere test driven. Sono stati sviluppati test per costruire una base di conoscenza sul funzionamento degli strumenti offerti da SPADE e riusati per valutare il funzionamento corretto delle primitive implementate.

I test sono contenuti nella classe TestAgent e sono stati creati durante tutte le fasi di sviluppo del progetto . Questi test verificano il funzionamento dei singoli agenti nelle loro interazioni altri agenti.

Nello specifico i test verificano la corretta inizializzazione delle diverse tipologie di agenti e il funzionamento del meccanismo di sottoscrizione e scambio di messaggi.

6 Istruzioni per il deployment

Di seguito sono riportate le operazioni necessarie per poter avviare il sistema.

6.1 Incompatibilità con sistema operativo MacOS

Il progetto è stato realizzato utilizzando macchine Windows. Solo arrivati alla fine e testandolo anche su altri sistemi operativi ci siamo accorti che tale sistema è incompatibile con MacOS. Il problema sta nel fatto che le GUI sono modellate come Thread a parte, differenti dal Main Thread, mentre in MacOS una qualsiasi modifica all'interfaccia grafica deve avvenire sul Main Thread stesso [3]. Questo comportamento del sistema operativo di Apple fa sì che vengano lanciate eccezioni che portano all'immediato crash del sistema.

6.2 Agenti di 616.pub

Per il corretto funzionamento del sistema è necessario che gli agenti si sottoscrivano, in modo da potersi inviare i messaggi solo nel momento in cui risultino connessi fra loro. La lista dei contatti però viene mantenuta anche quando il sistema viene fermato e riavviato, quindi gli agenti risultano già "in contatto" anche se di fatto non lo sono. Di conseguenza gli handler della sottoscrizione non vengono richiamati, causando un comportamento indesiderato. É quindi necessario pulire manualmente la lista dei contatti direttamente dal server 616.pub. Per farlo è necessario accedere dall'interfaccia degli agenti con credenziali `nome@616.pub` e password `edith`. Il link per poter accedere all'intefaccia web è `https://616.pub/web/`.

I JID predisposti per gli agenti del progetto sono:

- centralorderagent@616.pub;
- highwayagent@616.pub;
- statisticalagent@616.pub;

- temporalagent@616.pub;
- guiagent1@616.pub;
- guiagent2@616.pub;

6.3 Esecuzione del progetto

L'esecuzione del software prodotto necessita dei seguenti requisiti:

- Aver installato sulla propria macchina un sistema operativo compilato a 64bit o nel caso di Windows è supportata anche l'architettura a 32 bit.
- Aver installato Python nella versione 3.7.6, ovvero l'ultima release disponibile della versione 3.7.x.
- Disporre del framework SPADE; installabile attraverso *pip* (Package Installer for Python).

Essendo un progetto Python, non necessita di compilazione in quanto il linguaggio è interpretato. Per eseguirlo perciò basta avviare lo script Main.py attraverso l'interfaccia a linea di comando, con il seguente comando:
`py sd-project-gnagnarella-scucchia-1920/src/Main.py`.

6.4 Server XMPP

Non è necessaria l'installazione di un server locale, poichè si è fatto uso del server pubblico *616.pub*, per via delle motivazioni riportate nella Sezione 8.1.2.

7 Esempi d'uso

Come mostrato nella Figura 1 sono due i principali casi d'uso del sistema:

1. La creazione di nuovi utenti che possano partecipare al processo produttivo;
2. L'inserimento di nuovi ordini che vengano accettati dagli utenti e conseguentemente portati a termine.

All'avvio del sistema si aprirà l'interfaccia utente dell'amministratore, dalla quale sarà possibile creare nuovi utenti e inserire gli ordini.

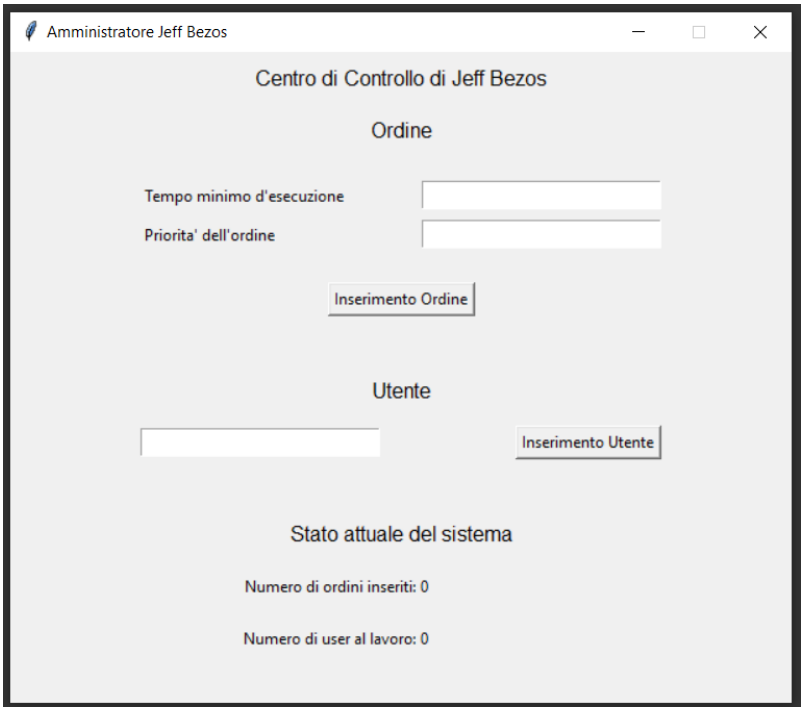

Figura 17: Interfaccia utente dell'amministratore.

Inserendo un nome e cliccando su *"Inserimento Utente"* verrà automaticamente creato un nuovo lavoratore.

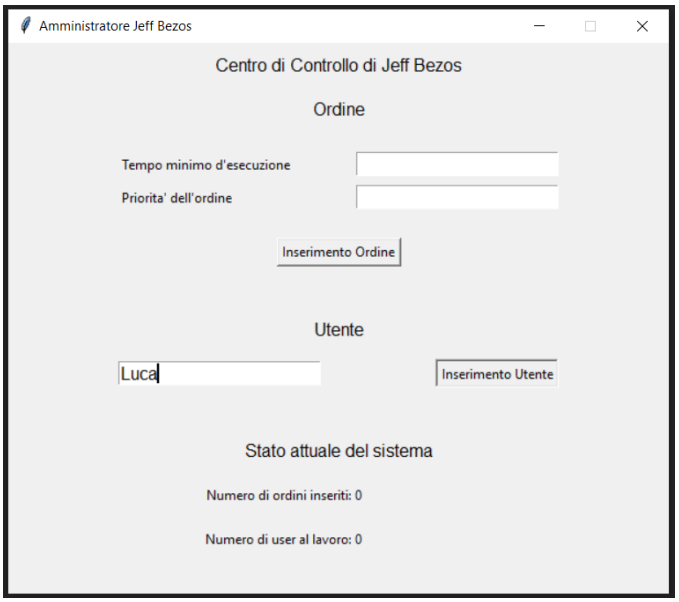

Figura 18: Inserimento utente.

Quando un utente viene creato, si avvia la sua interfaccia utente personale, nella quale inizialmente sarà visualizzato che non sta eseguendo nessun ordine, il pulsante per confermare la fine del lavoro è disabilitato e le statistiche sono tutte inizializzate a zero.

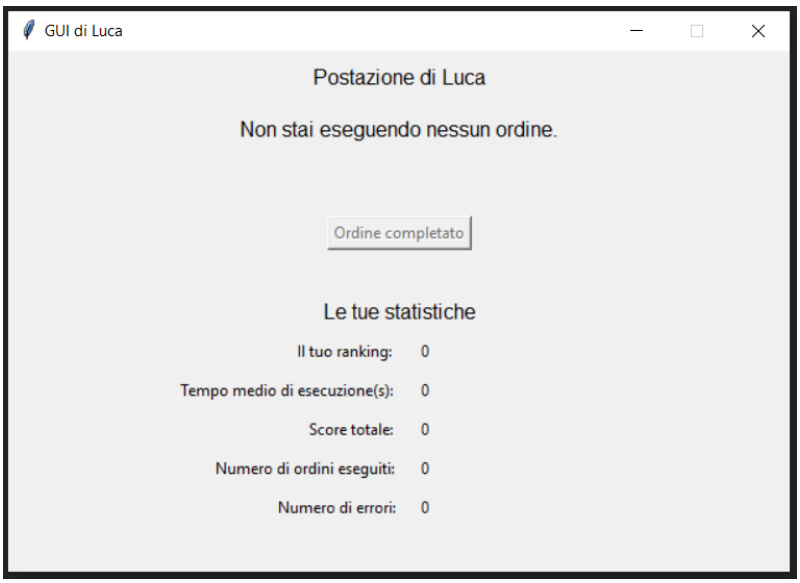

Figura 19: Interfaccia utente.

Quando l'amministratore vuole inserire un nuovo ordine è necessario riempire i campi *"Tempo minimo di esecuzione"* e *"Priorità dell'ordine"* e cliccare sul pulsante *"Inserimento Ordine"*.

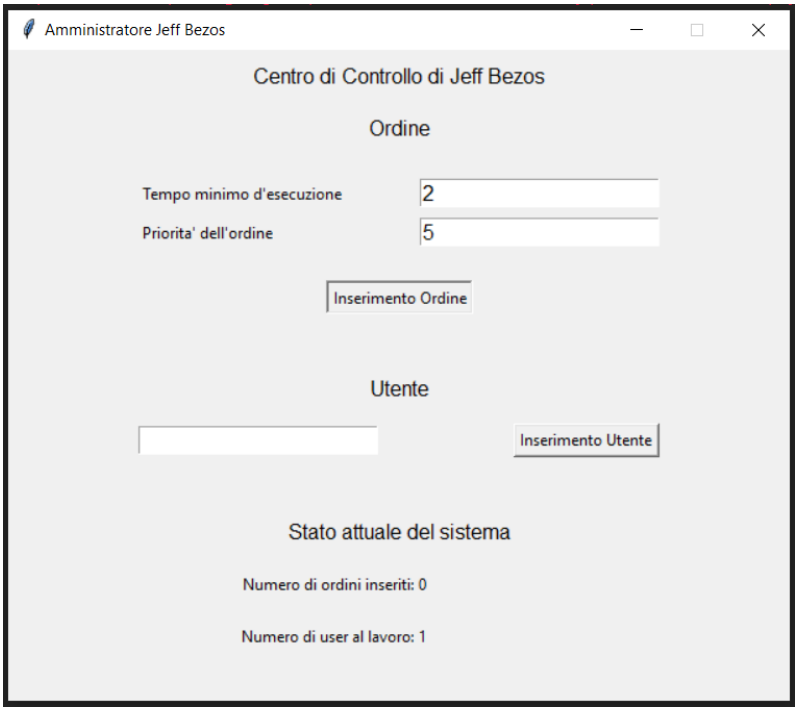

Figura 20: Inserimento ordine.

Una volta che l'ordine è inserito, esso viene inoltrato al lavoratore, il quale sarà notificato da una dialog. Cliccando il pulsante *"OK"* accetterà l'ordine.

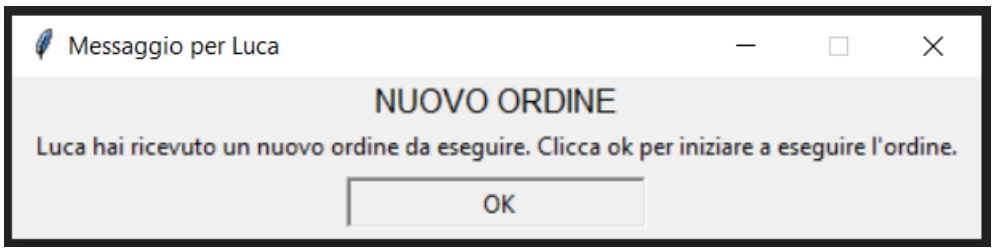

Figura 21: Accettazione dell'ordine.

Nel momento in cui l'utente accetta l'ordine, inizierà il conteggio del tempo, che viene visualizzato a schermo. Una volta terminato il compito, è necessario premere il pulsante *"Ordine completato"* per notificare la fine del lavoro.

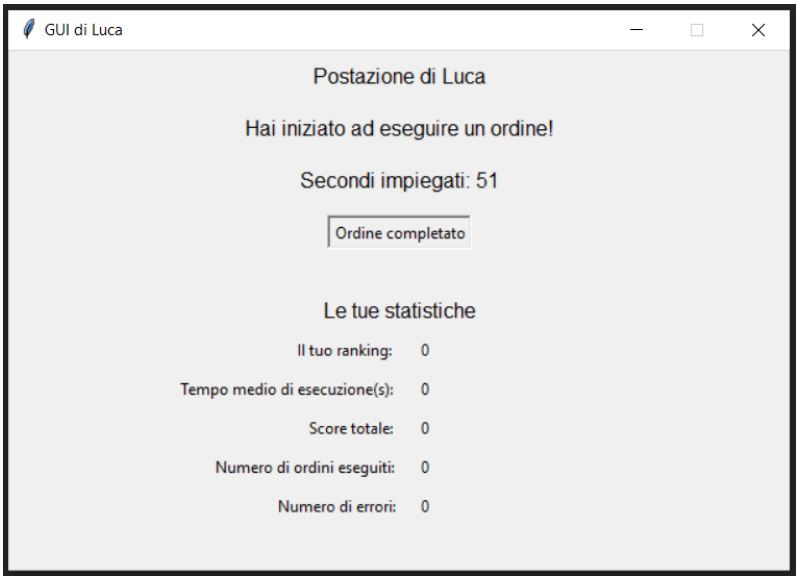

Figura 22: Completamento dell'ordine.

A questo punto l'interfaccia utente del lavoratore si aggiornerà, mostrando le nuove statistiche dell'operaio e lo stato di inattività, ovvero che non sta eseguendo nessun compito.

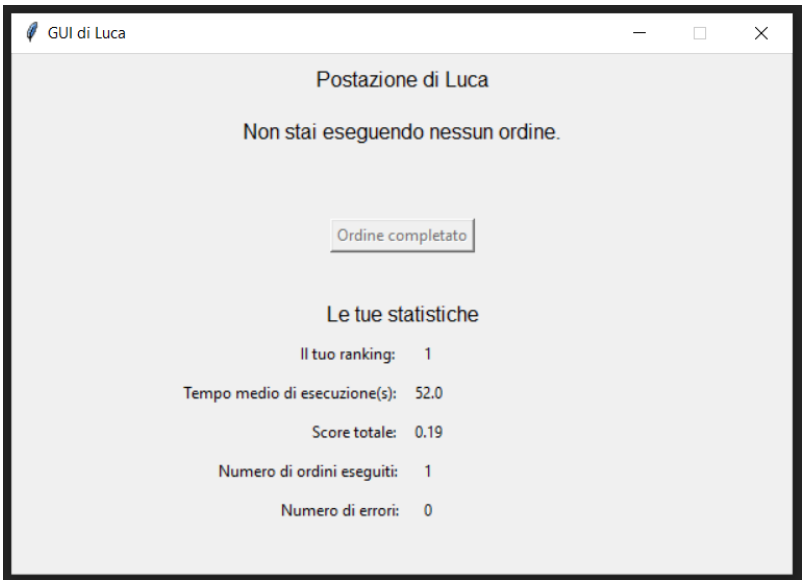

Figura 23: Aggiornamento dell'interfaccia utente.

8 Conclusioni

La realizzazione del progetto è stata indirizzata al raggiungimento degli obiettivi prefissati inizialmente. Giunti alle conclusioni ci riteniamo soddisfatti del lavoro svolto pur non essendo riusciti a coprire nell'interezza i traguardi che avevamo stabilito. Il risultato ottenuto ha soddisfatto ugualmente le nostre aspettative pur non essendo stato un percorso liscio e privo di intoppi.

La realizzazione di un sistema ad agenti distribuito con il framework SPADE ci ha visto impegnati nei seguenti passaggi:

- Studio e documentazione: prima di definire in modo dettagliato il progetto e il suo funzionamento, siamo stati impegnati nel comprendere al meglio come doveva essere strutturato un sistema ad agenti distribuito e come esso dovesse funzionare;
- Studio di SPADE: una volta apprese le conoscenze sopra indicate, siamo passati allo studio del framework SPADE, il quale è stato scelto a seguito di varie considerazioni fra i vari disponibili, come argomentato nel Capitolo 2. La scelta del linguaggio è stata obbligata dalla scelta di SPADE, infatti tale framework lavora in Python. L'apprendimento di questo linguaggio non è stato immediato e per comprendere al meglio quelli che sono i suoi costrutti e formalismi abbiamo investito del tempo;
- Implementazione: finalmente appresi i linguaggi e le tecnologie necessarie ci siamo concentrati sull'implementazione del sistema. Siamo partiti dal modello che avevamo idealizzato e abbiamo proseguito a sviluppare in modo incrementale i vari agenti.

Ci teniamo però a concludere mostrando le criticità riscontrate durante lo sviluppo di questo progetto, che ci hanno limitato nella realizzazione del sistema, impegnando molto del nostro tempo alla risoluzione di problemi esterni piuttosto che all'implementazione del sistema stesso.

8.1 Principali criticità

Come detto, siamo soddisfatti del lavoro svolto, poichè abbiamo abbondantemente superato il tempo di lavoro indicato e l'impegno da parte di entrambi è sempre stato costante. Tuttavia l'utilizzo di determinate tecnologie ha ostacolato le possibilità di dedicare più tempo all'implementazione del sistema vero e proprio e di seguito citiamo i casi principali.

8.1.1 Criticità nell'uso di SPADE

La principale criticità nell'uso del framework è sicuramente stata la scarsità della documentazione e del materiale reperibile online. Si trovano infatti vari paper che spiegano e analizzano la piattaforma SPADE, ma dal punto di vista dell'implementazione del codice il materiale online lascia un po' a desiderare. I problemi principali sono sorti nella gestione delle Presence Notification. É nostra personale opinione che da questo punto

di vista la documentazione presenti numerose lacune. Infatti la lista dei contatti di ogni agente mostra attributi di cui non è spiegato il significato e che quindi è difficile coglierne il comportamento. Tutto ciò che è stato appreso e implementato rispetto alle Presence Notification è stato fatto "a tentativi", provando e cercando di capire autonomamente quale fosse il loro comportamento. Un ulteriore dettaglio che ci ha lasciato perplessi è che ogni agente, nonostante rimuova dalla sottoscrizione altri agenti, tali agenti rimangano comunque nella sua lista dei contatti. Abbiamo capito di fatto che ogni agente mantiene una lista di roster items a lui connessi e, in linea teorica è possibile rimuoverli, ma in linea pratica ciò non è stato possibile. Infatti quando provammo a forzare manualmente la rimozione di un roster items dalla lista, questa operazione impiegava un tempo superiore alla scadenza di un timeout, il chè portava al lancio di una eccezione.

CAP Theorem In generale le varie problematiche riscontrate con le Presence Notification, pensiamo siano la reale applicazione del *CAP Theorem*, affrontato a lezione.

Il CAP Theorem dimostra come in un sistema distribuito, sia impossibile la presenza delle tre seguenti proprietà contemporaneamente:

- Consistenza, la coerenza dei dati, che indica che ogni nodo veda gli stessi dati.
- Disponibilità, cioè la garanzia che ogni richiesta riceva una risposta.
- Tolleranza ai guasti, ovvero la tolleranza di partizione.

Al più possono essere presenti soltanto due delle tre sopracitate. Nel nostro caso si denota la presenza di dati consistenti e di conseguenza vengono rispettate le proprietà di consistenza e tolleranza, mentre viene violata la disponibilità. Questo causa le problematiche sopra evidenziate e di conseguenza ci si imbatte, durante l'esecuzione in comportamenti anomali. Per esempio, l'aggiunta di un ulteriore utente durante l'esecuzione del sistema, e il successivo inserimento di ordini non porta ad una ricezione real-time dell'ordine, in quanto lo user deve prima sottoscriversi con gli altri agenti. Questo implica una perdita di tempo da parte di EDITH.

Più in generale, la ricezione e l'invio dei messaggi non risulta mai immediato, per quanto il sistema sia stata sviluppato per essere il più performante e efficiente possibile.

Salvataggio temporaneo delle credenziali Un ulteriore problema che ha richiesto molto tempo per essere compreso è il fatto che SPADE salva temporanemanete le credenziali come se fossero in memoria cache e di conseguenza la registrazione di agenti con credenziali differenti in esecuzioni successive del programma, ravvicinate nel tempo, portavano ad un errore di autenticazione poichè SPADE continuava di fatto ad utilizzare le informazioni salvate precedentemente.

Infine la documentazione e i paper trovati riguardo a SPADE ci hanno fatto sorgere dubbi riguardo al fatto che probabilmente non stessimo utilizzando il framework giusto. In vari paper online infatti si parla di Directory Facilitator, ACLMessage e altri concetti coerenti con lo standard FIPA, ma che nella versione di SPADE che abbiamo utilizzato non sono esplicitamente presenti. Abbiamo notato infatti che sulla pagina di SPADE di

GitHub sono presenti nove branches differenti e che l'implementazione esplicita di tali concetti non appartiene al ramo master.

8.1.2 Criticità nell'uso di Server XMPP

Per la realizzazione del progetto sono state testate diverse soluzioni XMPP. Inizialmente il progetto è stato implementato facendo uso di un server pubblico, *616.pub* che permette facilmente la registrazione In-Band degli agenti e la loro connessione e comunicazione. Da subito però siamo venuti in contatto con problemi la cui causa non era affatto intuitiva. Come detto SPADE nel caso di registrazione ravvicinata di agenti con credenziali differenti dava errori di autenticazione. Spesso questo comportamento però si aveva anche a distanza di tempo e il motivo lo abbiamo scoperto quando abbiamo provato a registrare manualmente nuovi agenti dal server stesso tramite l'interfaccia web. Esso infatti non permette la registrazione di in numero superiore di utenti da uno stesso indirizzo IP in un periodo ravvicinato di tempo.

Come server locali sono stati testati sia *Ejabberd* che *Prosody IM*.

Ejabberd Ejabberd è compatibile con sistemi operativi Windows, Linux e MacOS [2]. Sono stati riscontrati diversi problemi con la creazione di un server XMPP locale mediante l'uso di Ejabberd:

- L'utente admin non veniva registrato durante l'installazione, si è rivelato quindi necessario crearlo manualmente o dalla bash di Ejabberd con privilegi elevati in caso di sistema operativo Windows, o direttamente modificando il file di configurazione *ejabberd.yml* in caso di sistema operativo MacOS;
- La registrazione In-Band degli agenti è consentita solamente se si modifica il file di configurazione *ejabberd.yml*, aggiungendo all'attributo *mod_register*, l'attributo *IP_access* settato a *all*, che permette quindi ai client non registrati di autoregistrarsi al server;

L'idea di utilizzare Ejabberd è stata abbandonata in seguito all'implementazione delle Presence Notification, infatti, nonostante l'attributo *mod_roster* fosse stato settato a *all* nel file di configurazione, per permettere la registrazione dei roster items durante la sottoscrizione, tale comportamento non avveniva. Non siamo riusciti a trovare una soluzione a tale problema.

Prosody IM Prosody IM è incompatibile con il sistema operativo Windows, è stato necessario quindi installare un sottosistema Linux. Anche in Prosody IM sono state modificate diverse impostazioni nel file di configurazione, per permettere la registrazione In-Band degli agenti e la sottoscrizione dei contatti. Tuttavia anche con questo strumento sono stati rilevati dei problemi. Il sistema infatti non era in grado di funzionare poichè ad un certo momento dell'esecuzione il server disconneteva i client automaticamente, senza riportare nessun errore o motivazione particolare, come se ci fosse un timeout

oltre il quale un client viene disconnesso. Ovviamente in questo modo gli agenti essendo sconnessi non erano più attivi e in grado di scambiarsi informazioni.

Sono queste le motivazioni che ci hanno fatto ripiegare sull'uso del server pubblico *616.pub*.

8.1.3 Problema nell'uso di Unittest

Unittest è un framework ispirato a JUnit per la scrittura di test in Python [7]. Abbiamo utilizzato questo strumento per la scrittura di test, che si è dimostrato efficace. É stato riscontrato un problema riguardo alla gestione dei Thread, infatti terminati i test, nonostante questi passassero, viene lanciata un'eccezione riguardo al fatto che dei thread rimangono attivi nonostante i test fossero conclusi. Anche qui abbiamo provato diverse soluzioni:

- rendere i thread daemon;
- forzare l'arresto dei thread al termine dell'esecuzione;
- provare a eseguire manualmente i test, senza l'uso della funzione *unittest.main()*.

Tutte queste soluzioni si sono rivelate vane e anche qui la ricerca online non ha prodotto nessun tipo di risultato per permettere la correzione dell'errore.

8.2 Sviluppi futuri

Tra i possibili sviluppi futuri figura la possibilità di aggiungere funzionalità al sistema, come:

- Creazione o utilizzo di un sistema di criptazione e memorizzazione delle password per gli agenti;
- Ordini piu complessi, con specifici compiti da eseguire.
- Dispozione di un maggior di Highway Agent su sedi separate, in modo che ci sia un solo gestore centrale degli ordini e il lavoro di smistamento sia a sua volta suddiviso, in caso di grandi quantità di ordini.
- Gestione dello storico del ranking con grafici a disposizione delle varie GUI in modo che ogni User possa vedere in che momento è stato più produttivo e che il LeaderUser possa controllare l'andamento e la produttività dell'intero magazzino nel tempo.

Considerando altre modifiche che però vanno a modificare aree di maggior dimensioni all'interno del progetto, potrebbe essere interessante un upgrade del sistema per permettere l'integrazione con dispositivi mobili o wearable. Risulterebbe infatti molto più comodo per gli operai, essere dotati di un visore per la realta aumentata, che oltre a mostrare l'ambiente attorno e gli ordini da eseguire, possa assistere gli operai per svolgere tali ordini nel modo più efficiente possibile.

8.3 Conoscenze acquisite

Questo progetto ha arricchito notevolmente il nostro bagaglio di conoscenze. Innanzitutto abbiamo imparato a padroneggiare Python come linguaggio di programmazione Object-Oriented. Nota positiva in quanto attualmente è posizionato nella top ten dei linguaggi più utilizzati e risulta essere alla base di molti algoritmi di AI, che potremmo dover trattare in futuro in altri ambiti e corsi.

Indubbiamente abbiamo appreso molto del funzionamento di SPADE, le astrazioni chiave e la loro gestione. Di notevole rilevanza sono state le esperienze con i diversi server XMPP che ci hanno dato l'opportunità di mettere le mani in modo approfondito all'interno di strumenti e tecnologie a noi sconosciute.

Analizzando il progetto con uno sguardo più generale, senza concentrarsi sul singolo framework utilizzato, abbiamo esteso le nostre competenze riguardo ai sistemi ad agenti, che per noi risulta essere fondamentale visto l'importante ruolo che ricopriranno nel futuro della programmazione. É stato infatti grazie a questo progetto che abbiamo potuto capire appieno gli aspetti del corso per quanto riguarda questo nuovo paradigma di programmazione.

Riferimenti bibliografici

[1] Javier Palanca, Sergio Alemany. *SPADE Documentation*. URL: https://spade-mas.readthedocs.io/en/latest/foreword.html.

[2] *Ejabberd*. URL: https://www.ejabberd.im/.

[3] Apple Inc. *Main Thread Checker*. URL: https://developer.apple.com/documentation/code_diagnostics/main_thread_checker.

[4] Javier Palanca. *SPADE: Agents based on XMPP*. URL: https://www.slideshare.net/JavierPalanca/spade-agents-based-on-xmpp-82102493.

[5] *PEP 257 – Docstring Conventions*. URL: https://www.python.org/dev/peps/pep-0257/#one-line-docstrings.

[6] *Python*. URL: https://www.python.it.

[7] *unittest - Unit testing framework*. URL: https://docs.python.org/3.7/library/unittest.html.