ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Scuola di Ingegneria e Architettura
Campus di Cesena
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

INTEGRAZIONE ARCHITETTURALE DI PERSONAL ASSISTANT AGENT BASATI SU MODELLO BDI CON SERVIZI COGNITIVI: UN CASO DI STUDIO IN AMBITO OSPEDALIERO

Elaborata nel corso di:
Ingegneria dei Sistemi Software Adattivi Complessi

Tesi di Laurea di:
LORENZO PELLEGRINI

Relatore:
Prof. ALESSANDRO RICCI

Co-relatore:
Dott.ssa SARA MONTAGNA

ANNO ACCADEMICO 2016–2017
SESSIONE III

PAROLE CHIAVE

Trauma Tracker
Smart Hospital
Cognitive Computing
Machine Learning
Sanità

Alla mia famiglia

Indice

Introduzione **xi**

1 Personal Assistant Agent in Ambito Healthcare – Il progetto Trauma Tracker **1**
1.1 I Personal Assistant Agents 1
1.2 Applicazione in ambito Healthcare: il Personal Medical Digital Assistant 3
1.2.1 Obiettivo di un PMDA 3
1.2.2 Inquadramento nel contesto Smart Hospital 4
1.3 Il progetto Trauma Tracker 6
1.3.1 Obiettivi 6
1.3.2 Stato attuale del progetto 8
1.4 La generazione di avvisi e suggerimenti 9
1.4.1 Ruolo e utilità di un sistema per la generazione di avvisi 9
1.4.2 Generazione di avvisi basata su regole 11
1.4.3 Opportunità di evoluzione: generazione basata su informazioni storiche 12
1.4.4 Obiettivo: estensione architetturale del PMDA per sfruttare servizi cognitive 13

2 Il PMDA di Trauma Tracker e architettura BDI **17**
2.1 Architettura del sistema 17
2.2 Struttura del PMDA e tecnologie impiegate 19
2.3 Il modello BDI 21
2.4 Generazione di avvisi basati su regole: potenzialità e limiti . 22
2.4.1 Il sistema attuale 23
2.4.2 Potenzialità e limiti di un sistema basato su regole . 26

3 Cognitive Services: applicazioni in ambito Healthcare e mercato **29**
3.1 Tecnologie Cognitive nell'ambito medicale 29
3.2 Ambiti toccati dalla letteratura 31
3.3 Machine Learning 32
3.4 Piattaforme Cognitive 38

4 Estensione del Trauma Assistant Agent con Cognitive Services **51**
4.1 Obiettivi del progetto 52
4.2 Requisiti 53
4.3 Analisi 54
4.4 Architettura del sistema 55
4.5 Il Cognitive Service 57
4.5.1 Cos'è il Cognitive Service 57
4.5.2 Ruolo e funzionalità attese 57
4.5.3 Architettura del Cognitive Service 58
4.6 Modellazione del trauma 62
4.6.1 I dati rilevanti per un algoritmo predittivo 63
4.6.2 Le informazioni a disposizione e le sorgenti dati . . . 64
4.6.3 Rappresentazione della conoscenza 64

5 Tecnologie impiegate **67**
5.1 Vert.x 68
5.1.1 Architettura del framework 68
5.1.2 Inquadramento all'interno del progetto 70
5.2 Docker 70
5.2.1 Funzionalità e API offerte 71
5.2.2 Inquadramento all'interno del progetto 74
5.2.3 La libreria Docker Client 74
5.3 RabbitMQ 75
5.3.1 Inquadramento all'interno del progetto 75
5.4 MongoDB 76
5.4.1 Inquadramento all'interno del progetto 76

6 Implementazione prototipale **77**
6.1 RxVertxUtils 77
6.1.1 Problematiche relative allo stile di programmazione . 78

6.1.2 CPS vs Promises vs Reactive Programming 79
6.1.3 Bridging dal sistema Callback-based a un sistema basato su RxJava 2 80
6.2 InitializationUtils 81
6.2.1 InitializationContext 82
6.2.2 Tracciabilità degli errori di startup e configurazione . 83
6.3 Gestione delle comunicazioni e l'Event Bus Bridge 84
6.3.1 Bridging dei messaggi tramite RabbitMQ 85
6.3.2 Bridging dinamico basato sul destinatario 86
6.3.3 Uso del sistema di acknowledgment e dei meccanismi di persistenza 87
6.4 Le componenti del sistema 88
6.4.1 Aspetti comuni 88
6.4.2 TT Cognitive 89
6.4.3 Cognitive Manager 90
6.4.4 Practice Manager 90
6.4.5 Application Manager 91
6.5 Libreria di supporto per lo sviluppo di algoritmi 92
6.6 Integrazione con il sistema esistente 93
6.7 Configurabilità e deployment 94
6.7.1 Configurazione 94
6.7.2 Deployment 95
6.7.3 Entry points 97
6.8 Test 97
6.8.1 Configurazione degli elementi e dell'ambiente di test . 98
6.8.2 Unit test 99
6.8.3 Integration test 99

7 Validazione ed evoluzione **101**
7.1 Validazione 101
7.2 Evoluzione 103

Introduzione

L'ambito sanitario è oggetto di cambiamenti legati alla pervasiva adozione di tecnologie informatiche in grado di supportare i processi legati alla gestione della sanità. Diverse strutture si stanno attrezzando in tal senso con l'obiettivo di aumentare l'efficienza generale incrementando il tasso di successo nei trattamenti e diminuendo i costi.

Su questa strada vanno le iniziative inerenti agli Smart Hospital. Queste prevedono una informatizzazione degli ospedali e dei sistemi per la loro gestione al fine di incrementare l'efficienza generale del personale sanitario, facilitare la gestione dell'ospedale e migliorare il rapporto dei pazienti con la struttura. Questa pionieristica visione prevede che vengano messe in campo le più disparate tecniche informatiche con l'obiettivo di supportare l'operato del personale sanitario, specialmente per quanto concerne il processo decisionale e la gestione del flusso di lavoro.

La tesi si inserisce nel contesto di Trauma Tracker, un progetto di Smart Hospital nato dalla collaborazione tra il Trauma Center dell'ospedale Maurizio Bufalini e la facoltà di Ingegneria e Scienze Informatiche dell'Università di Bologna, sede di Cesena. Scopo del progetto è quello di sviluppare un Personal Assistant Agent in grado di supportare i medici durante la gestione dei traumi fornendo loro un'ampia gamma di funzionalità.

Tra di queste è presente la generazione di avvisi e suggerimenti il cui obiettivo è quello di guidare l'operato del Trauma Team richiamandone l'attenzione qualora venissero individuate situazioni anomale oppure opportunità circa azioni che è possibile intraprendere. In Trauma Tracker è attualmente presente un sistema per la generazione di avvisi basata su regole. Questo presenta una espressività limitata per cui si intende ampliare le capacità di ragionamento dell'Assistant al fine di considerare informazioni storiche mantenute nell'archivio delle pratiche. Tra le tecniche considerate a tal fine figurano il Machine Learning e il Cognitive Computing.

L'obiettivo della tesi è quello di identificare e sviluppare un'estensione architetturale del Trauma Assistant Agent (TAA) in modo che possa integrare la generazione di avvisi basati su regole definite dagli esperti del dominio, codificate come di piani nel modello per agenti cognitivi BDI (Belief-Desire-Intention), con la generazione di suggerimenti elaborati e forniti in modo asincrono da servizi cognitivi in rete, in continua interazione con il TAA.

La tesi si apre con una descrizione del progetto Trauma Tracker seguita da un'esplorazione riguardante le tecniche in questione e il modello BDI adottato nello sviluppo dell'Assistant. Successivamente viene esposto il progetto riguardante lo sviluppo della piattaforma con cui sarà possibile integrare nell'Assistant le capacità fornite da algoritmi predittivi.

Capitolo 1

Personal Assistant Agent in Ambito Healthcare – Il progetto Trauma Tracker

Il progetto Trauma Tracker ha come obiettivo quello di costruire un sistema in grado di supportare il personale sanitario durante la gestione dei traumi. Nato come strumento in grado di automatizzare la produzione dei rapporti finali riguardanti le pratiche, questo si è evoluto per includere differenti funzionalità in grado di supportare maggiormente l'operato del Trauma Team.

In questo capitolo viene fornita una panoramica delle principali funzionalità, della forma con cui viene fornita l'assistenza e della opportunità individuata per l'evoluzione della funzionalità di generazione di avvisi e suggerimenti, argomento su cui è incentrata la presente tesi.

1.1 I Personal Assistant Agents

Il Personal Assistant Agent è un agente software il cui obiettivo è assistere i propri utenti ottenendo e mostrando loro informazioni utili, interfacciandosi a servizi, eseguendo azioni al posto loro, supportando il processo comunicativo e decisionale e in generale aiutando gli assistiti svolgendo mansioni da questi esplicitamente richieste o la cui necessità è individuata dal contesto di riferimento per cui è stato pensato l'assistente.

I Personal Assistant Agent variano per:

- Contesto di riferimento

  Sono diversi gli assistenti software in commercio. Gli esempi più famosi sono dati da quelli comunemente forniti come servizio tramite smartphone, tablet e Personal Computer. Questa tipologia di assistente vede come contesto l'insieme delle informazioni personali dell'assistito, la sua posizione, le sue azioni e in generale tutto ciò che può essere percepito tramite i sensori presente da dispositivi connessi. Un'altra tipologia di contesto è l'ambito lavorativo e industriale. Quest'ultima tipologia di assistenti ovviamente prevede un insieme di funzionalità differenti da quelli facenti parte della famiglia precedentemente esposta. Va comunque considerato che tutti questi condividono il fatto di trarre grande vantaggio dall'uso di informazioni di contesto al fine di arricchire l'esperienza d'uso e migliorare il servizio offerto.

- Servizi offerti

  I servizi offerti variano a seconda della tipologia dell'assistente e del contesto di riferimento. Si possono individuare:

  – Servizi "concierge-type", tipici degli assistenti per smartphone, che supportano l'utente eseguendo per loro, sotto esplicita richiesta, un ristretto insieme di azioni quali l'inserimento di un elemento a calendario, la riproduzione di brani musicali, la ricerca di informazioni tramite un motore di ricerca, eccetera.

  – Servizi eseguiti automaticamente e autonomamente, cioè operazioni effettuate senza una esplicita richiesta da parte dell'utente. La scelta di quando e come intraprendere determinate azioni è dettata dal contesto in cui si trova l'assistito e dalle informazioni in possesso dall'Assistant. In questo caso l'assistente mostra un comportamento "proattivo". [26]

  I servizi "concierge-type" non offrono molto se non il supporto dato dall'automatismo fornito dall'assistente: le stesse azioni, che devono essere esplicitamente richieste, potrebbero essere eseguite dall'utente di persona. Questa forma di interazione è anche denominata "direct manipulation". Al contrario, nel secondo caso si ha una forma di

supporto denominata "indirect management". In questa l'assistito viene coinvolto in un processo di collaborazione continua. [20]

- Utenza

  Non è scontato che l'utente dell'Assistant sia una singola persona. In alcuni contesti l'assistenza deve essere fornita a un team di persone che possono interagire con l'assistente in contemporanea e in maniera contrastante.

1.2 Applicazione in ambito Healthcare: il Personal Medical Digital Assistant

Un Personal Medical Digital Assistant rappresenta una maniera elegante e potenzialmente molto utile per rendere disponibili al medico informazioni riguardo ai pazienti e al contesto in cui sta operando. Oltre al semplice rendere disponibili informazioni in tempi e modi utili, un assistente è in grado di raccogliere ed elaborare informazioni e ricevere comandi, andando a costruire un rapporto di interazione con l'assistito.

Come evidente dal nome, un PMDA è un assistente software che prevede una utenza costituita da solo assistito fornendo a questo servizi legati all'ambito medico.

1.2.1 Obiettivo di un PMDA

Un PMDA è quindi un software context-aware e non intrusivo che permette di rende più efficiente ed efficace l'operato del personale sanitario. Questo può essere modellato come un agente in grado di comunicare con la persona assistita per fornire automaticamente informazioni adeguate al contesto identificato e per rendere disponibile in un qualsiasi momento l'accesso a informazioni richieste.

Ne deriva che il Medical Assistant, agendo su base contestuale, focalizza il proprio operato sulle le informazioni riguardanti al paziente esaminato presentando risultati inerenti alla sua storia e stato attuale, cioè quell'insieme di informazioni che definiscono il contesto in cui sta operando l'assistito.

Oltre a questo l'assistente è in grado di riportare informazioni e avvenimenti importanti inerenti a contesti differenti ma comunque sotto la respon-

sabilità dell'assistito. Questo apre alle potenzialità discusse successivamente e oggetto della tesi.

1.2.2 Inquadramento nel contesto Smart Hospital

Il PMDA si inserisce nella famiglia delle applicazioni per Smart Hospital, la cui visione è sovrapponibile a quella delle Smart Cities e delle Smart Industry sia in termini di obiettivi di massima sia dal punto di vista delle tecnologie abilitanti e delle problematiche da affrontare. Nello specifico si intende creare un ambiente che integri componenti hardware e software che supportino il personale e i pazienti nella gestione del "sistema ospedale".

Al fine di ottenere questa condizione è necessaria una pervasiva informatizzazione del sistema informativo che garantisca un'adeguata circolarità delle informazioni. La sola informatizzazione degli applicativi gestionali classici non è sufficiente: da uno Smart Hospital ci si attende di poter sfruttare elementi smart in grado di comunicare dati realtime e di intraprendere azioni che coinvolgano l'ambiente in cui si trovano. A questi vanno aggiunti strumenti che permettano di sfruttare al massimo il contenuto informativo disponibile.

1.2.2.1 Visione e obiettivi

Gli obiettivi di un'applicazione di Smart Hospital consistono nel miglioramento nella gestione di risorse, tempi, risultati dei trattamenti e della qualità della vita dei pazienti durante la loro permanenza.

In particolare, è possibile individuare aree come la coordinazione degli interventi e del personale, la comunicazione ai medici di informazioni realtime riguardanti i pazienti, il tracciamento automatico di eventi, il supporto al processo decisionale, l'identificazione di situazioni anomale, l'analisi dei dati e automazioni di vario genere al fine di liberare il personale medico da incombenze non legate alla cura del paziente. Il PMDA si va a inserire proprio in questo contesto: si tratta di una forma di supporto al medico i cui obiettivi sono quelli appena esposti.

Oltre al supporto del personale sanitario, lo Smart Hospital può includere sistemi per supportarne la governance. Allo stesso tempo possono essere previsti sistemi atti a supportare l'utenza garantendo strumenti informatici con cui facilitare l'approccio al sistema sanitario, garantendo trasparenza

nella gestione delle pratiche e migliorando la qualità della vita durante la degenza.

1.2.2.2 Tecnologie abilitanti e problematiche

Vi sono diverse problematiche dal punto di vista tecnologico. In primo luogo va affrontato l'aspetto inerente ai dati. Risulta di primaria importanza che gli applicativi gestionali e gli EHR siano integrati al "sistema ospedale" al fine di garantire un'adeguata circolarità delle informazioni.

Il secondo passo consiste nell'inserimento nell'ambiente di elementi smart, che includono sensori, schermi, dispositivi indossabili e altri elementi del mondo dell'Internet of Things. A questo va aggiunta la possibilità di sfruttare i dati provenienti dai macchinari, spesso dotati di interfacce proprietarie datate o difficilmente accessibili. In generale gli Smart Hospital devono disporre di elementi in grado di ricevere e trasmettere informazioni realtime riguardo allo stato del paziente e agli eventi registrati nei diversi ambienti ospedalieri.

Infine, sia i dati storici che quelli ottenuti in tempo reale devono essere raccolti e memorizzati da sistemi informatici adeguati e in grado di supportare algoritmi di varia natura finalizzati all'analisi dello stato del paziente e dell'organizzazione ospedaliera.

Le problematiche di tipo organizzativo insorgono proprio nel contesto della gestione dei dati. Gli elementi critici riguardano la gestione della privacy e della sicurezza delle informazioni. I dati, memorizzati e analizzati in un Centro Elaborazione Dati, devono essere mantenuti al sicuro e deve essere progettato un sistema per il controllo degli accessi stringente. Tuttavia sarebbe un errore considerare questi aspetti solamente nell'ottica dell'amministrazione delle informazioni mantenute all'interno dei CED degli istituti sanitari. Gli aspetti di sicurezza in un Smart Hospital devono pervadere anche le componenti hardware e software di sensori e applicativi posti nell'ambiente. Dal punto di vista software questo significa curare la memorizzazione e la trasmissione dei dati effettuata dai vari oggetti smart, che includono elementi del mondo IoT, smartphone, tablet e indossabili di varia natura. In aggiunta alle problematiche inerenti al software, particolare attenzione va posta nella progettazione della sicurezza fisica di tali dispositivi. Un ulteriore aspetto riguarda la creazione e messa in sicurezza della infrastruttura di rete locale, spesso approntata proprio in occasione

dell'integrazione di aspetti di Smart Hospital in strutture esistenti ancora non dotate di collegamenti via cavo o wireless.

La possibilità di utilizzare le informazioni raccolte apre a diverse strade: i dati raccolti possono essere utilizzati per ottenere statistiche, condurre retrospettive sul lavoro svolto, eccetera. Dal punto di vista della privacy intervengono elementi normativi stringenti. A tal fine vi è la necessità di identificare ed eliminare identificativi e altri dati personali, oppure approntare sistemi che isolino in partenza la componente informativa che permetta l'identificazione del paziente. Altro aspetto, che tocca entrambe queste problematiche organizzative, riguarda la possibilità di far circolare informazioni sensibili tra diversi ospedali dello stesso stato. Questo aprirebbe alla possibilità di rendere minimi i tempi per l'ottenimento di informazioni urgenti riguardanti un paziente di un'altra regione, oltre all'effetto benefico nella gestione della burocrazia sanitaria e, per quanto riguarda l'utente, una maggior accessibilità e trasparenza nella gestione del proprio profilo sanitario. In un'ottica più generale sarebbe anche possibile pensare a protocolli appositamente regolamentati per comunicare informazioni rilevanti a istituti di cura esteri qualora fosse confermata la necessità di rendere immediatamente disponibili dati riguardanti un paziente. [7]

1.3 Il progetto Trauma Tracker

Di seguito viene presentato Trauma Tracker, un progetto che si muove nell'ambito degli Smart Hospital nato da una collaborazione tra l'ospedale Maurizio Bufalini di Cesena e la facoltà di Ingegneria e Scienze Informatiche dell'Università di Bologna, sede di Cesena, e che prevede lo sviluppo di un sistema a supporto del personale del Trauma Center.

1.3.1 Obiettivi

L'obiettivo del progetto Trauma Tracker è quello di costruire un Medical Assistant Agent in grado di supportare il personale del Trauma Center durante il trattamento di un trauma. Durante gli interventi l'agente è in esecuzione su dispositivi mobili quali smartphone, tablet e indossabili in grado di raccogliere ed elaborare dati, visualizzare informazioni utili, notificare il personale in caso di anomalie, eccetera.

La prima problematica presa in considerazione per lo sviluppo del sistema è stata quella del tracciamento degli eventi. Durante la gestione di un trauma vengono somministrati farmaci, eseguite manovre, effettuate diagnosi dello stato del paziente. Questi avvenimenti vanno a formare una sequenza di eventi da memorizzare in un apposito report che sarebbe da redigere sul momento. Questo nella pratica risulta impossibile per cui la procedura diventa quella di completare la gestione del caso per compilare il report solo successivamente con annessi ovvi problemi di precisione e completezza. La necessità di compilare tali rapporti è in primo luogo dettata da vincoli normativi ma a questo si aggiunge l'utilità di avere informazioni utili su cui basare le fasi di retrospettiva. Avere a disposizione la precisa sequenza temporale degli eventi avvenuti durante la gestione dei traumi può supportare il Trauma Team nell'individuazione di potenziali aree di miglioramento permettendo loro di isolare gli elementi critici.

Trauma Tracker, al fine di tenere traccia di tutti gli eventi, mette a disposizione interfacce con cui il team può inserire input manualmente al fine di registrare l'esecuzione di manovre, la somministrazione di farmaci, inserire indicatori di sintesi riguardanti lo stato del paziente oltre al poter registrare audio, clip video e fotografie che verranno inserite nel report finale. È inoltre possibile ottenere in tempo reale i dati relativi ai parametri vitali direttamente dai macchinari. Per ogni evento viene automaticamente rilevato e memorizzato l'orario e l'ambiente all'interno del quale è stato registrato.

Buona parte di queste operazioni può essere svolta utilizzando smartphone, tablet o wearable. Nello specifico, allo stato attuale, vengono utilizzati un tablet Android e smartglass non invasivi in grado di visualizzare informazioni a schermo. La possibilità di usare quest'ultima famiglia di dispositivi, che integrano una fotocamera e un microfono, ha permesso di sviluppare la funzionalità inerente alla trasmissione in tempo reale di quanto sta accadendo al fine ottenere supporto remoto da specialisti.

L'insieme dei dati raccolti, resi disponibili all'agente in tempo reale, apre alla possibilità di sfruttare diverse tecniche per rilevare situazioni anomale oppure formulare consigli. Avvisi e suggerimenti possono essere inviati direttamente al Trauma Leader per richiamarne l'attenzione.

1.3.2 Stato attuale del progetto

Per via della stretta collaborazione con gli utenti finali e dell'interesse da loro mostrato per i risvolti pratici del progetto, feedback riguardanti le funzionalità sviluppate sono ottenuti con buona frequenza e i risultati sono valutati sul campo. I rilasci vengono effettuati periodicamente secondo un piano iterativo.

Allo stato attuale l'app Android in cui viene eseguito il PMDA supporta l'apertura di una nuova pratica, l'impostazione del responsabile, il tracking degli eventi e il collegamento con la posizione in cui avvengono. Ogni accadimento viene registrato tramite un'apposita interfaccia grafica ed è possibile inserire quali farmaci sono stati somministrati (comprese le quantità), quali manovre sono state eseguite, indicatori numerici che riassumono lo stato del paziente. La versione attuale vede anche l'implementazione di un sistema di avvisi basato su regole proposte dal Team. Risulta implementato l'invio del report finale al servizio web in esecuzione nel backend. Dal punto di vista della sensoristica ambientale, dei Beacon vengono utilizzati per identificare la stanza del Trauma Center. Il sottosistema di posizionamento risulta ben funzionante.

Per quanto riguarda l'integrazione con gli smart glasses, che verranno utilizzati dal Trauma Leader, è stato sviluppato un sistema per la visualizzazione di notifiche ed è già stata approntata una versione prototipale della trasmissione in remoto del flusso video.

L'integrazione del sistema con i macchinari esistenti utilizzati per la lettura di parametri vitali risulta essere in fase di sviluppo. È stato approntato un gateway (TT Vital Signs Monitor Service) che permette di raccogliere i dati inviati dai sistemi embedded direttamente connessi ai sensori oppure ottenuti da servizi intermediari. È prevista a breve l'integrazione con il restante sistema in modo che tali informazioni risultino disponibili al PMDA e ai rimanenti applicativi in tempo reale.

Il panel amministrativo, sotto forma di web app, permette la visualizzazione della lista dei traumi ed è già stata sviluppata la possibilità di visualizzare un insieme di statistiche attraverso alcuni grafici predefiniti. In futuro potrebbe venire aggiunta la possibilità di modificare report per inserire valori disponibili solo successivamente.

Il modulo per la produzione dei report è completo. Con la possibilità di ottenere e memorizzare le informazioni relative ai parametri vitali sarà

possibile aggiungere un tracciato o snapshot dei parametri vitali, insieme ad altre informazioni che risulteranno integrabili nelle successive versioni del sistema.

1.4 La generazione di avvisi e suggerimenti

La generazione di avvisi e suggerimenti rappresenta una delle funzionalità che si intende approntare per il sistema Trauma Tracker. Avvisi e suggerimenti sono accomunati dal fatto di essere generati sotto forma di notifiche e dall'intento di migliorare la gestione complessiva dei traumi di cui l'assistito è responsabile. L'unica differenza sta nella tipologia del contenuto: dagli avvisi ci si aspettano informazioni circa situazioni che richiedono attenzioni immediate mentre dai suggerimenti ci si attende di ricevere consigli che possono essere ignorati o presi in considerazione con l'ottica di modificare il corso delle azioni nel medio-lungo periodo. Queste notifiche sono inviate all'assistito in maniere opportune decise in collaborazione con gli esperti del Trauma Team e il loro contenuto può essere vario potendo spaziare dalle informazioni circa la situazione di uno specifico paziente alle operazioni richieste nella gestione concorrente di diversi traumi, da informazioni quali lo sforamento di soglie relative a singoli parametri vitali a situazioni complesse rilevate mediante la considerazione di più elementi relativi allo stato del paziente.

1.4.1 Ruolo e utilità di un sistema per la generazione di avvisi

I medici del Trauma Center devono muoversi tra diversi pazienti al fine di gestire più situazioni concorrenti. Un meccanismo che invii degli avvisi qualora sia stata rilevata una situazione anomala, la necessità di effettuare una manovra oppure somministrare un farmaco può risultare utile per richiamare l'attenzione del personale. Un sistema di questo genere può portare alla riduzione dei tempi complessivi nella gestione di un caso, maggior precisione temporale nella somministrazione di farmaci e nell'esecuzione delle manovre previste e in generale a una maggiore probabilità di successo nel trattamento del trauma.

1.4.1.1 Come avviene la gestione di un trauma

Il contesto di riferimento è rappresentato dalla gestione dei traumi, un ambito particolarmente complesso dal punto di vista medico e allo stesso tempo interessante da un punto di vista informatico, probabilmente tra i più emblematici per quanto riguarda le possibili applicazioni che ben si sposano con la vision dello Smart Hospital. La gestione di un trauma comporta diverse problematiche legate alla varietà delle situazioni e la velocità con cui devono essere prese ed attuate le decisioni.

Il primo contatto con il personale sanitario avviene solitamente al Pronto Soccorso oppure all'arrivo dell'ambulanza e dell'auto medica. Da questo momento i parametri vitali devono essere tenuti sotto controllo e vengono messe in campo procedure atte a stabilizzare la condizione del paziente.

Una volta arrivato in ospedale e individuato un trauma effettivo il paziente viene trasferito al Trauma Center. Da questo momento il paziente si trova nelle cure di un Trauma Team. Il team è guidato da un Trauma Leader che non opera in prima persona ma che dirige le operazioni prendendo le decisioni che ritiene più opportune al fine di stabilizzare la condizione. Questo include la scelta dei farmaci da somministrare, manovre da effettuare, l'uso di macchinari specifici, il trasferimento in reparti differenti, eccetera.

In alcune situazioni si potrebbero avere più casi che necessitano di attenzioni. Se questi risultano essere in un numero troppo elevato il carico di lavoro diventa difficile da gestire per via dei frequenti cambi di contesto e della confusione che può venirsi a creare negli ambienti del Trauma Center. In particolar modo per ogni paziente sono da considerare molteplici variabili e, considerata la situazione caotica, può risultare complesso tenere traccia degli eventi passati, dei limiti temporali dettati dalla somministrazione di farmaci e dall'esecuzione di manovre e in generale delle peculiarità delle singole situazioni.

Lo scopo del team è quello di stabilizzare la condizione del paziente nel più breve tempo possibile limitando i danni. Risultano di particolare importanza gli interventi effettuati sul paziente a un'ora dall'avvenuto trauma, periodo di tempo universalmente conosciuto come "golden hour", per cui l'ottimizzazione dei tempi risulta essere un elemento critico.

In definitiva gli elementi critici nella gestione di un trauma sono: tempi stringenti, concomitanza di diversi traumi, caoticità della situazione, forti differenze tra i diversi casi.

1.4.1.2 Requisiti e modalità per la generazione di avvisi e suggerimenti

I macchinari comunemente utilizzati per rilevare i parametri vitali dispongono solitamente della possibilità di emettere avvisi sonori. Tuttavia, in situazioni caotiche come quelle che possono venirsi a formare all'interno del Trauma Center, questa soluzione risulta inutile se non addirittura dannosa per via del fatto che le notifiche vengono emesse continuamente e per tutti i pazienti. Va inoltre sottolineato come gli avvisi vengano emessi solo quando il parametro misurato risulti essere sopra o sotto una certa soglia preimpostata. In altre parole, attraverso un sistema di questo genere, non sarebbe possibile incrociare la conoscenza riguardante più di un parametro vitale, dei farmaci precedentemente somministrati, della situazione del paziente, eccetera.

Avendo a disposizione un sistema informatico in grado di raccogliere i dati riguardanti i parametri vitali e gli eventi che si sono susseguiti sarebbe possibile produrre avvisi specifici e certamente più utili da inviare al personale attraverso un canale ottimale. In particolare, sarebbe possibile notificare il Trauma Leader recapitando gli avvisi e le informazioni sui parametri vitali attraverso smartphone, tablet, dispositivi indossabili e monitor posti nell'ambiente. In questa maniera sarebbe possibile veicolare, oltre agli avvisi, informazioni aggiuntive quali suggerimenti riguardanti casi simili, procedure che sarebbe possibile mettere in campo, farmaci consigliati, informazioni riguardanti il flusso di lavoro.

Per fare questo deve essere tenuta traccia di tutti gli eventi che riguardano il trattamento del paziente. Una volta messo in campo un sistema di questo genere vi è la necessità di approntare una componente in grado di elaborare ciò che sta accadendo e quindi visualizzare i risultati ottenuti nella maniera ritenuta più opportuna.

1.4.2 Generazione di avvisi basata su regole

Avvisi e suggerimenti devono rispondere all'esigenza pratica di migliorare le performance del team e le possibilità di successo nella gestione di un trauma. Risulta di grande importanza decidere quali sono le notifiche (e il relativo contenuto informativo) che più di tutte possono impattare positivamente su come viene gestito un trauma. A partire dalle informazioni che il Trauma

Team ha definito maggiormente utili è stato formulato un insieme regole che guidano la generazione degli avvisi e suggerimenti.

L'elaborazione delle informazioni raccolte durante la gestione di un trauma è quindi attualmente affidata a un sistema basato su regole. Questo verrà esposto in dettaglio successivamente ma risultano da subito evidenti le limitazioni inerenti alla capacità espressiva di un sistema così costituito. Oltre alla complessità insita nella formulazione di regole che coinvolgono una grande quantità di variabili, queste sarebbero comunque legate all'esperienza del personale che le ha individuate e potrebbero non tenere in considerazione informazioni che invaliderebbero la generazione della relativa notifica.

Alla mancanza di capacità espressiva si uniscono complicazioni relative all'implementazione del corpus di regole nell'agente software che incapsula la logica di generazione degli avvisi.

Questi elementi hanno portato a considerare diversi metodi per la risoluzione delle problematiche appena esposte e in particolar modo è stata individuata un'opportunità legata all'ampliamento delle capacità di inferenza dell'Assistant: la generazione di avvisi e suggerimenti basati sulle numerose informazioni storiche mantenute nel Trauma Center.

1.4.3 Opportunità di evoluzione: generazione basata su informazioni storiche

L'Assistant può essere evoluto per considerare le informazioni contenute nei report delle pratiche gestite dal Trauma Team al fine di identificare situazioni simili a quella in cui sta operando l'assistito. Avvisi e suggerimenti possono essere generati a partire da una analisi della situazione attuale rapportando questa a pattern riscontrati nel trattamento di altri traumi. Il risultato atteso da questo approccio sono notifiche in grado di veicolare informazioni supportate da base statistica certa, prive del bias dovuto all'esperienza personale e in grado di considerare, a partire da pattern precedentemente sconosciuti e rivelati con apposite tecniche, situazioni particolarmente complesse oppure rare.

In questa maniera è possibile ottenere una predizione circa la situazione in cui può venirsi a trovare il paziente, informazioni inferite a partire dallo stato attuale o addirittura dalla sequenza degli eventi che si sono succeduti, consigli circa le successive manovre da eseguire, avvisi inerenti ad azioni

controindicate, pronostici riguardanti i tempi necessari per vedere l'effetto di manovre e sostanze, eccetera.

Questa strategia per la generazione di avvisi vede una capacità espressiva decisamente maggiore di quella legata al solo utilizzo di regole. Il PMDA finale potrebbe integrare entrambe le strategie e addirittura le regole potrebbero far uso di informazioni inferite con l'ausilio di un modello basato su dati storici.

1.4.4 Obiettivo: estensione architetturale del PMDA per sfruttare servizi cognitive

L'obiettivo è quello di sfruttare la conoscenza pregressa memorizzata. Al fine di raggiungere questo obiettivo il Medical Assistant può essere integrato con strumenti provenienti dal mondo del Cognitive Computing e del Machine Learning. L'assistente potrebbe eseguire e visualizzare i risultati di algoritmi di classificazione, predizione degli outcome, analisi di immagini, ... che possono risultare utili durante il processo decisionale. Tali tecniche possono essere sfruttate per creare servizi che possano produrre informazioni utili nel contesto in cui si sta muovendo l'assistito. Queste possono poi essere sfruttate per la generazione di avvisi e suggerimenti.

1.4.4.1 Vision

Il PMDA è un agente con limitate capacità di inferenza legate alla programmazione ricevuta. Gli agenti basano le proprie azioni su di un insieme di comportamenti codificati utilizzando astrazioni e tecnologie differenti. La base di conoscenza viene creata, interrogata e manipolata seguendo una programmazione definita dallo sviluppatore, cosa che porta con sé le classiche problematiche legate allo sviluppo del software e in particolar modo limita la capacità espressiva a quanto conosciuto dallo sviluppatore o dagli esperti circa il dominio da modellare e il contesto in cui si troverà l'agente. Questo trova come diretta conseguenza la scelta di costruire il sistema a regole sopra descritto con tutte le relative problematiche e limitazioni.

Attraverso l'uso di tecniche cognitive si intende sopperire a queste mancanze con il fine ultimo di ottenere un personal Assistant Agent dotato di capacità di inferenza altrimenti non possibili.

1.4.4.2 Potenzialità generali

La possibilità di sfruttare un modello costruito a partire da dati storici permette di espandere fortemente le capacità di un Personal Assistant Agent fornendogli strumenti per mettere in atto comportamenti più espressivi.

Dal punto di vista puramente teorico la possibilità di integrare tecniche cognitive nella gestione del ragionamento di un assistente apre alla possibilità di avere assistenti:

- In grado di espandere la propria base di conoscenza applicando l'esperienza codificata in un modello costruito tramite operazioni di Learning alle informazioni in quel momento in suo possesso circa il contesto in cui si trova.
- In grado di raffinare, durante il suo ciclo di vita, le proprie capacità di ragionamento. Questo è possibile se l'assistente viene dotato di tecniche che gli permettono di aggiustare la propria rappresentazione del mondo senza dover ricevere una riprogrammazione. Di queste fanno parte le tecniche di Learning per rinforzo.

La principale potenzialità è quindi data da una più ricca capacità di inferenza. Questa permette all'assistente di espandere la propria base di conoscenza con informazioni altrimenti non deducibili dal contesto. La nuova conoscenza può essere sfruttata per modificare il comportamento dell'assistente portando a scelte diverse riguardo alle azioni da intraprendere.

1.4.4.3 Potenzialità nella gestione dei traumi

Come precedentemente descritto si intende applicare queste tecniche alla generazione di avvisi e suggerimenti. Queste possono essere utilizzate, previo un allenamento su dati storici, per predire situazioni future, suggerire azioni da compiere, verificare se le manovre effettuate o le sostanze somministrate hanno avuto l'esito sperato, classificare lo stato del paziente, eccetera.

La logica che considera la conoscenza prodotta dai diversi algoritmi predittivi può essere codificata all'interno dell'assistente. In questo è possibile inserire condizioni di guardia, individuate dagli esperti del dominio, che controllino la generazione degli avvisi in modo che i risultati ottenuti non appaiano in momenti non opportuni o in situazioni in cui questo potrebbe essere controproducente di fatto unendo l'esperienza del personale del Trauma Team alla conoscenza dedotta dai dati storici.

Una situazione limite è data dall'uso dei risultati ottenuti al fine di mostrare direttamente al personale informazioni che si ritiene essere correlate al contesto corrente lasciando al Trauma Team il compito di identificare quali sono le informazioni rilevanti. Questo può essere utile per generare suggerimenti ma potrebbe essere controproducente per quanto riguarda la produzione di avvisi.

Capitolo 2

Il PMDA di Trauma Tracker e architettura BDI

Trauma Tracker è un sistema composto dal Personal Medical Digital Assistant, componente all'interno del quale è implementata la logica dei servizi offerti, e da un insieme di componenti strutturali che hanno l'obiettivo di garantire che l'Assistant abbia accesso ai dati provenienti dai sensori e allo stesso tempo permettere che questo possa compiere le azioni per cui è stato programmato.

In questo capitolo viene descritta l'architettura del sistema, le tecnologie impiegate e viene effettuata una panoramica del modello Belief-Desidre-Intention utilizzato per definire la logica dell'Assistant e delle modalità con cui questo paradigma è stato impiegato nell'ambito della generazione di avvisi e suggerimenti.

2.1 Architettura del sistema

Allo stato attuale sono quattro i macro-elementi fondamentali di Trauma Tracker:

- Il Personal Medical Digital Assistant. L'agente è reso disponibile sotto forma di un'app in esecuzione su di un tablet Android. L'Assistant è composto da un insieme agenti BDI sviluppati nel linguaggio Jason e integrati con il contesto di esecuzione Android attraverso il framework Jaca-Android. L'assistente rappresenta la componente utilizzata dal

team per ricevere e inviare informazioni riguardanti gli eventi e lo stato del paziente. L'applicativo Android include anche i servizi sviluppati per interagire con i dispositivi indossabili.

- Un insieme di servizi pervasivi in esecuzione su dispositivi disposti nell'ambiente.

  Attualmente questa componente è formata da un insieme di beacon che permettono di identificare l'ambiente del Trauma Center in cui sta operando il Team. La posizione viene identificata automaticamente dall'agente sfruttando la tecnologia di comunicazione locale Bluetooth.

- Un insieme di servizi web in esecuzione nella rete locale.

  Questi si occupano di raccogliere e memorizzare i dati oltre che a renderli disponibili alle app che ne fanno richiesta. Si tratta di un elemento che svolge il ruolo di backend per l'intero sistema. I dati provengono sia dall'PMDA che dai macchinari utilizzati per rilevare i parametri vitali.

- Un insieme di web apps.

  Queste si interfacciano ai servizi web al fine di ottenere i dati da visualizzare agli utenti, ovvero i membri del Trauma Team. Le app includono una Dashboard, all'interno della quale è possibile visualizzare lo storico dei traumi e i relativi dettagli, e un sistema di analisi in cui è possibile visualizzare statistiche. Il team utilizza queste interfacce per controllare la situazione attuale e per ottenere informazioni sui casi passati al fine di eseguire retrospettive.

La logica del PMDA è stata opportunamente separata dall'ambiente di esecuzione, risultando di fatto completamente indipendente esso e pertanto l'Assistant è considerabile una componente a sé stante nel sistema, cioè indipendente dall'applicazione Android. La logica che descrive i servizi offerti è completamente incapsulata dagli agenti che compongono il PMDA e questo rende particolarmente agevole la manutenzione e evoluzione delle funzionalità offerte.

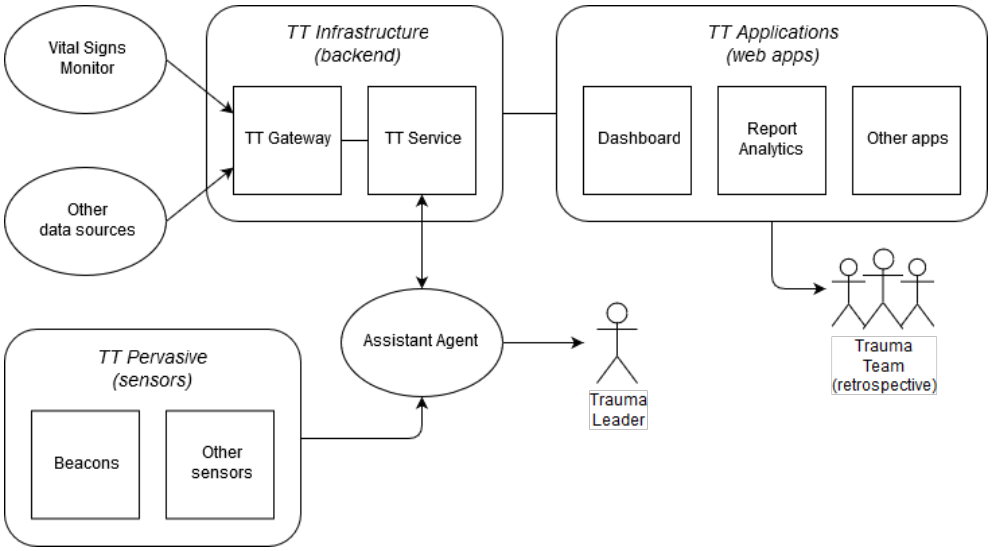

Figura 2.1: Architettura ad alto livello di Trauma Tracker

2.2 Struttura del PMDA e tecnologie impiegate

Il PMDA è costituito da un insieme di agenti sviluppati secondo il modello BDI e posti in un Multi-Agent System. Questi operano in maniera indipendente l'uno dall'altro e l'interazione con l'ambiente avviene tramite l'interfacciamento con artefatti ambientali. Questi artefatti, il cui funzionamento è descritto dettagliatamente in seguito, permettono all'agente di ricevere eventi e accedere a informazioni circa il contesto del trauma e allo stesso tempo mettono a disposizione la logica con cui effettuare le azioni richieste dall'agente. Gli agenti non interagiscono direttamente tra loro ma le azioni effettuate possono essere catturate dagli artefatti e quindi riportate agli altri agenti.

Entrando nel dettaglio relativamente alla struttura del PMDA, oltre ad alcuni agenti minori utilizzati per le operazioni di avvio e orchestrazione, è possibile individuare i seguenti agenti principali:

- Il Tracker Agent: questo agente si occupa di tenere traccia di tutti gli eventi registrati durante la gestione di un trauma al fine di produrre

un report finale dettagliato. Si tratta del principale agente di Trauma Tracker e realizza la funzionalità originariamente prevista per il sistema.

- Il Warning Generator Agent: il suo scopo è quello di generare avvisi e suggerimenti che verranno mostrati al Trauma Leader.

Il framework di riferimento per lo sviluppo degli agenti è JaCaMo: questo è specificamente studiato per lo sviluppo di agenti BDI e prevede Jason come linguaggio di programmazione utilizzabile per esprimere i piani previsti per l'agente, Cartago come sistema per programmare artefatti ambientali e Moise come sistema per l'orchestrazione di MAS.

Di seguito una essenziale panoramica delle altre tecnologie utilizzate nel progetto, che includono:

- Android: rappresenta il sistema di riferimento per l'esecuzione del PMDA. Il tablet su cui l'agente è in esecuzione deve potersi connettere alla WLAN dell'ospedale in modo da poter comunicare con il backend. Il dispositivo deve supportare il protocollo Bluetooth Low Energy al fine di garantire la possibilità di individuare la propria posizione nell'ambiente.

- Beacon e Bluetooth: i beacon vengono utilizzati per stabilire la posizione dell'agente e quindi del paziente all'interno degli ambienti del Trauma Center. La rilevazione della posizione avviene sfruttando la tecnologia Bluetooth, attraverso la tecnica di advertising prevista dal protocollo BLE.

- Occhiali per la Realtà Aumentata: questi dispositivi includono tecnologie per la visualizzazione non intrusiva di elementi utili al Trauma Leader. Sono presenti una fotocamera e un microfono utilizzabili per registrare elementi utili da inserire nel report finale oppure per creare uno streaming video visualizzabile da remoto. Sono diversi i dispositivi compatibili con queste esigenze per cui non è ancora stato stabilito il modello di riferimento. Attualmente viene utilizzato un Vuzix m300.

- Vert.x e protocolli Web: Vert.x è un framework per lo sviluppo di applicativi reattivi per Java Virtual Machine. Attraverso tale framework risulta semplice creare microservizi web RESTful ed è stato

utilizzato per l'implementazione dei servizi che formano il backend del sistema. Ne segue che le comunicazioni tra le componenti avvengono utilizzando protocolli web.

- Angular: framework di riferimento per lo sviluppo degli applicativi web quali la dashboard.

- Sistemi embedded di varia natura: alcuni dei macchinari che misurano i parametri vitali presentano i dati su interfacce che richiedono sistemi in grado di interfacciarvici direttamente. I dati raccolti vengono poi inviati a una componente del backend denominata "Gateway".

2.3 Il modello BDI

Il PMDA è stato sviluppato secondo il modello per agenti intelligenti "Beliefs-Desires-Intentions", modello ispirato alla teoria del "Human Practical Reasoning" del filosofo Michael Bratman. [2]

Gli agenti BDI presentano una architettura comune basata su tre principali componenti:

- Beliefs: descrivono tutto ciò che l'agente conosce riguardo sé stesso e il contesto in cui si trova. Si parla di Belief e non di Facts in quanto l'agente ha una visione parziale riguardo l'ambiente in cui si sta muovendo, cosa tipica negli agenti software e più in generale dei sistemi distribuiti. Di fatto questo insieme di informazioni definisce lo stato dell'agente.

- Desires: rappresentano i desideri dell'agente. Questi sono rappresentati da obiettivi "finali" o di lungo termine e possono entrare in contrasto tra loro. Questi possono concretizzarsi in Goals, ovvero un insieme di obiettivi su cui l'agente sta attivamente operando. L'agente seleziona i futuri Goals basandosi sui Desires andando a selezionare un insieme non contrastante di obiettivi.

- Intentions: rappresenta ciò che l'agente ha scelto di fare. Questo è rappresentato da un insieme di azioni.

Gli agenti BDI sono entità naturalmente reattive e in quanto tali il loro ciclo di vita è guidato e influenzato da eventi. Gli eventi possono

essere registrati dall'agente che può gestirli in maniera differente in base alla natura delle tre componenti appena descritte: con "agenti BDI" non ci riferisce a una tecnologia, framework o a un linguaggio di programmazione ben definito ma piuttosto a una famiglia di paradigmi. La differenza tra questi è rappresentata dalle diverse modalità con cui viene gestita la base dei Belief, descritti e selezionati i Desire, costruite e messe in atto Intentions e infine la modalità con cui vengono gestiti gli Events.

In Trauma Tracker viene utilizzato il Procedural Reasoning System. Utilizzando questa metodologia l'agente può essere sviluppato codificando un insieme di piani. Questi sono costituiti da tre parti:

- Un obiettivo che indica i risultati ottenuti dalla sua esecuzione.
- Un contesto che definisce quali sono le condizioni ambientali che devono risultare rispettate affinché possa essere intrapreso.
- Un corpo che descrive quali sono le azioni che formano il piano.

Nel momento in cui avviene un evento la base di conoscenza dell'agente che costituisce l'insieme dei Belief viene aggiornata. Per ogni evento viene consultata la lista dei piani disponibili. Se uno di questi risulta applicabile, cioè qualora le precondizioni risultassero soddisfatte, le azioni che lo compongono vengono inserite nell'insieme delle Intentions. Queste possono consistere in:

- Operazioni che coinvolgono la Belief Base.
- Generazione di eventi interni.
- Azioni da attuare nell'ambiente.

2.4 Generazione di avvisi basati su regole: potenzialità e limiti

Il sistema Trauma Tracker ha il proprio fulcro nel PMDA. L'assistente è stato sviluppato secondo il paradigma ad agenti con il fine di mantenere facilmente leggibile, manutenibile ed evolvibile, oltre che indipendente dagli elementi tecnici della piattaforma di esecuzione, la logica che ne determina il

comportamento. Questo permette in ogni momento di espandere le funzionalità offerte in termini di capacità di percezione e ragionamento inserendo nel sistema componenti che ricoprono il ruolo di sensori e attuatori cioè, più in generale, qualsiasi elemento che permetta di espandere le capacità dell'agente software di interagire con l'ambiente. Da questo punto di vista il progetto si è evoluto apportando incrementi dal punto di vista delle capacità di percepire il contesto in cui si muove l'assistente ma non vi sono stati significative evoluzioni dal punto di vista delle capacità di ragionamento.

Di seguito viene descritto in dettaglio il funzionamento del PMDA focalizzandosi in particolar modo sulla componente delegata alla generazione degli avvisi e dei suggerimenti da mostrare in tempo reale al Trauma Leader. Successivamente viene esposto come, a partire dal sistema esistente, sia possibile aumentare le capacità dell'assistente tramite l'uso di tecnologie cognitive. Questo comporta un insieme di problematiche sia da un punto di vista tecnologico che teorico essendo richiesta l'unione di agenti, una astrazione tipica del mondo dell'ingegneria del software, con tecniche facenti parte della famiglia del Cognitive Computing e del Machine Learning.

2.4.1 Il sistema attuale

La tecnologia di riferimento per l'implementazione dell'assistente è JaCaMo, nella sua versione adattata per l'esecuzione nell'ambiente Android denominata JaCa-Android. JaCaMo è un framework costituito dall'integrazione di tre elementi:

- Jason, un linguaggio di programmazione basato sul linguaggio astratto AgentSpeak. Permette di descrivere i piani utilizzati dagli agenti che formano il PMDA, dove il "Tracker Agent" e il "Warning Generator Agent" sono i principali. Alcuni esempi di come questi piani siano stati codificati verranno mostrati in seguito.

- Cartago, un sistema per lo sviluppo di artefatti ambientali. Lo sviluppo avviene utilizzando Java. Gli artefatti svolgono il ruolo di tramite al fine di veicolare la conoscenza dall'ambiente verso l'agente e viceversa. Questo include anche l'attuazione delle azioni che coinvolgono l'ambiente previste dai piani dell'agente.

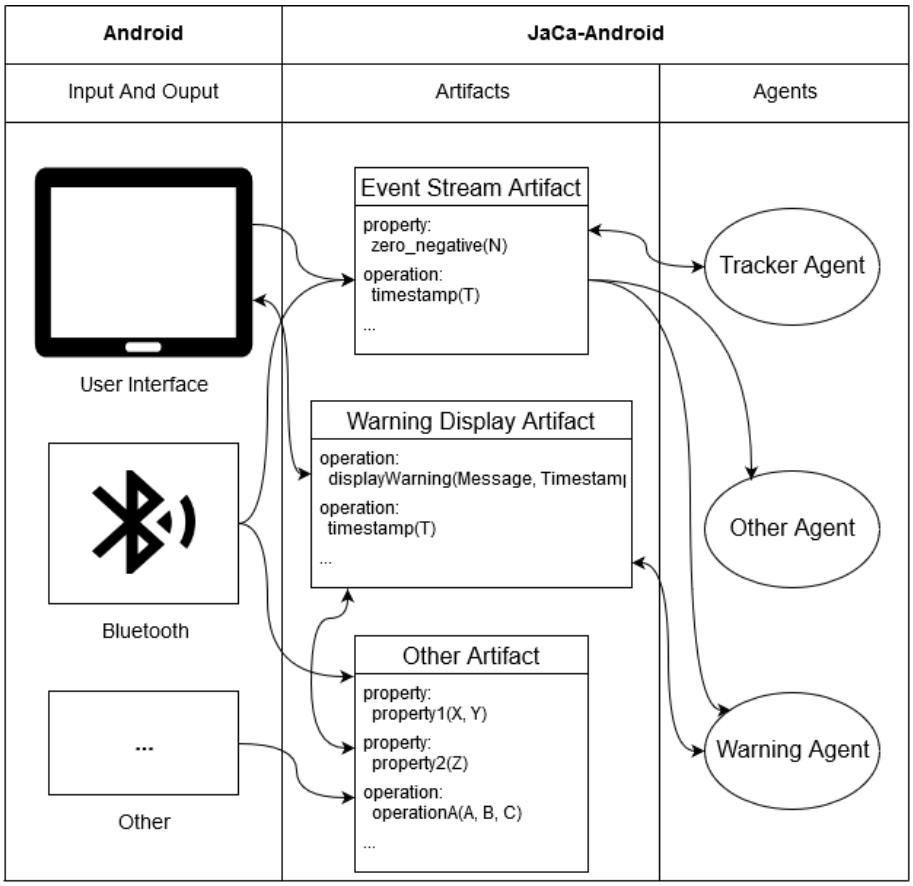

Figura 2.2: Architettura dell'applicativo Android.

- Moise, un sistema per utilizzabile per orchestrare gli agenti che partecipano ad un MAS. Questa componente non è utilizzata nel progetto Trauma Tracker e non è attualmente prevista da JaCa-Android.

Come appena accennato gli artefatti costituiscono il tramite con il quale l'agente si rapporta con l'ambiente. In fase di bootstrap l'agente identifica oppure inizializza gli artefatti di cui avrà bisogno. Il framework JaCa-Android si occupa di mappare gli elementi dell'artefatto, rappresentati da un insieme di proprietà, all'interno della Belief Base dell'agente. Tali proprietà degli artefatti sono creabili ed eliminabili dinamicamente e l'esecuzione di ognuna di queste azioni comporta quindi una modifica della base di cono-

scenza dell'agente. Sempre tramite gli artefatti l'agente può venir notificato di eventi che possono portare alla messa in esecuzione di nuovi piani e allo stesso modo l'agente può indicare le azioni da eseguire sull'ambiente esterno richiamandole per nome. Il metodo Java che implementa l'azione è cercato a runtime tra tutti gli artefatti connessi all'agente.

2.4.1.1 Formulazione e provenienza delle regole: l'esperienza sul campo

Il PMDA è in grado di generare e visualizzare opportunamente avvisi e suggerimenti circa il contesto in cui si sta muovendo il Trauma Team. Questi segnali hanno l'obiettivo di supportare il personale in una qualsiasi maniera ritenuta positiva. Ne segue che gli avvisi non devono essere ridondanti e tantomeno visualizzati in tempi e modi non opportuni.

Al fine di identificare contenuti e forma degli avvisi, il primo passo è consistito nel chiedere ai componenti del team quali fossero le forme di supporto che gli avrebbero permesso di semplificare la gestione dei casi. In questo contesto il team ha delineato un insieme di problematiche comuni legate alla gestione dei tempi e modi nell'esecuzione di manovre ricorrenti. A partire dalle problematiche identificate è stato dedotto un insieme di regole dettate dall'esperienza accumulata dal personale. Queste sono composte da due parti:

- Un insieme di condizioni che devono essere vere per poter generare l'avviso. Queste si riferiscono a dati quali: l'avvenuta esecuzione o meno di una manovra durante la gestione del trauma, l'avvenuta somministrazione di farmaci e sostanze, i parametri vitali attuali del paziente, eccetera.

- L'avviso da visualizzare.

2.4.1.2 La generazione degli avvisi

Allo stato attuale la generazione di avvisi e suggerimenti si basa sul suddetto insieme regole, codificate manualmente sotto forma di un insieme di piani che descrivono un agente sviluppato secondo il paradigma Beliefs-Desires-Intentions.

Il corpus di regole formulato dal Trauma Team è codificato all'interno del "Warning Generator Agent". Di seguito alcuni esempi seguiti da una breve spiegazione:

Per ogni regola viene descritto il messaggio da mostrare e le condizioni che devono verificarsi. Trattandosi di un sistema reattivo i piani possono venire attivati solamente nel momento in cui viene registrato un evento. In Trauma Tracker gli eventi sono costituiti da un payload e da un timestamp. La conoscenza viene codificata in stile Prolog e similmente gli eventi vengono gestiti specificando un pattern e un insieme di condizioni che devono risultare vere affinché venga messo in atto il relativo piano. La regola 6, la più semplice delle due, è da leggere come: se avviene l'evento "uscito dalla Shock Room" e tra i Belief compare "è presente una frattura" e non compare l'evento "eseguita profilassi antibiotica", allora aggiungi alla Belief Base *warning("checkABT", T)*, dove T è il timestamp dell'evento originale. Con l'aggiunta di quest'ultimo elemento alla base di conoscenza viene attivato il piano per la gestione degli eventi *warning*, che prevede venga richiamata l'azione *displayWarning* implementata in Java in un artefatto.

Per regole che coinvolgono un solo evento legato a diverse condizioni la codifica delle regole è immediata e facilmente leggibile. Al contrario, se condizioni ed eventi coincidono anche solo parzialmente allora la codifica delle regole diventa complesso, come nel caso della regola 2. In questo caso la trasfusione di sangue zero negativo, la somministrazione di fibrinogeno e di Tranex sono da considerarsi sia eventi che condizioni. In altre parole non conta l'ordine di queste azioni, ma solamente il fatto che siano state eseguite tutte e tre. Per poter implementare la regola è quindi necessario gestire tutti e tre gli eventi, in maniera separata. Ovviamente il piano correlato ha lo stesso corpo, costituito unicamente da *warning("activatePTM", T)*.

2.4.2 Potenzialità e limiti di un sistema basato su regole

Il sistema appena descritto presenta le potenzialità e i limiti intrinsechi di un sistema basato su regole. Queste possono essere codificate a partire da una loro descrizione in linguaggio naturale e a questo va aggiunto che, nel caso specifico di Trauma Tracker, grazie alla scelta tecnologica di adottare Jason come linguaggio di riferimento, queste risultano facilmente leggibili anche sotto forma di codice.

Tabella 2.1: Regole

<table>
<tr><td colspan="3">+event(roomOut("Shock-Room"), T)
: fractures(true) & not event(drug("antibiotic-prophylaxis", _),_)
<- +warning("checkABT", T).

+warning("checkABT", T)
<- displayWarning("Activate Antibiotic Prophylaxis?", T).</td></tr>
<tr><td>Regola 6</td><td>If there is a fracture, when exiting from the Shock,Room without having started the antibiotic prophylaxis.</td><td>Message: “Activate Antibiotic Prophylaxis?”.</td></tr>
<tr><td colspan="3">+event(drug("zero-negative", _),T)
: zero_negative(3) & event(drug("tranex",_),_) &
event(drug("fibrinogen",_),_)
<- +warning("activatePTM",T).

+event(drug("tranex",_),T)
: zero_negative(3) & event(drug("fibrinogen",_),_)
<- +warning("activatePTM",T).

+event(drug("fibrinogen",_),T)
: zero_negative(3) & event(drug("tranex",_),_)
<- +warning("activatePTM",T).

+warning("activatePTM", T)
<- displayWarning("Activate PTM?", T).</td></tr>
<tr><td>Regola 2</td><td>When administered 3 unit of zero negative blood and,tranexamic acid and fibrinogen.</td><td>Message: “Activate Massive Transfusion Protocol,(MTP)?”.</td></tr>
</table>

Quella che viene codificata è l'esperienza del team accumulata nella gestione di molti e diversi traumi. L'obiettivo raggiunto è quello di fornire assistenza nella gestione dei casi riportando suggerimenti circa situazioni comuni dove è stata riscontrata la possibilità che vi siano rallentamenti o errori. Il risultato, già verificato sul campo, è un miglioramento nel trattamento dei casi sotto forma di tempi di gestione diminuiti. A questo potrebbero aggiungersi miglioramenti, da verificare, sotto forma delle probabilità successo e complicazioni prevenute dovuti alla gestione tempestiva delle manovre.

Un sistema basato su regole ha limiti imposti dalla natura e forma delle regole. Le regole sono ottenute direttamente dagli esperti del dominio e potrebbero:

- Non considerare aspetti che renderebbero il suggerimento generato ridondante o addirittura controproducente. È il caso in cui le condizioni per una certa regola risultino rispettate ma queste non considerino casi specifici per cui viene generato un suggerimento che, se seguito, porterebbe a risultati negativi.

- Non coprire diversi casi nell'ambito della stessa manovra. In questo modo si ottiene una visione parziale della procedura da seguire che potrebbe portare situazioni anomale nel futuro.

- Sono limitate all'esperienza e visione personale degli esperti. Le regole possono essere condizionate da una visione parziale che non trova riscontro con quanto descritto dai dati storici.

- Regole complesse sono difficilmente esprimibili, leggibili e implementabili. Risulta difficile se non impossibile implementare regole basate sull'analisi del susseguirsi degli eventi senza perdere di generalità andando a scrivere logica applicativa ad hoc. Le regole attuali sono infatti basate sulla sola conoscenza dello stato attuale del paziente e delle manovre effettuate durante la gestione del trauma.

Queste problematiche possono essere affrontate attraverso metodi differenti e che non fanno uso di regole codificate manualmente. Tra questi fanno figurano tecniche che fanno riferimento al Machine Learning e al Cognitive Computing.

Capitolo 3

Cognitive Services: applicazioni in ambito Healthcare e mercato

La tesi ha come riferimento l'ambito sanitario e in particolar modo l'area della gestione dei traumi. Negli ultimi anni si è potuto assistere ad una spinta rivoluzionaria incentrata sulla progettazione e attuazione di metodologie a supporto del paziente e ai medici che ricadono nell'ampia famiglia delle tecniche di intelligenza artificiale. Già nei primi pionieristici progetti queste hanno restituito risultati positivi ottenuti anche grazie alla massiva disponibilità di nuove e riscoperte tecniche nell'ambito della computer science.

Di seguito vengono esposti questi approcci in particolar modo focalizzandosi sulla famiglia delle tecniche di Cognitive Computing e di Machine Learning e di come queste stiano apportando un contributo significativo in diverse aree della medicina.

3.1 Tecnologie Cognitive nell'ambito medicale

L'ospedale rappresenta un ambiente ricco di dati. Classicamente gli istituti mantengono un archivio storico di record, spesso compartimentati per reparto, riguardo a visite ed esami eseguiti sui pazienti insieme a report

che descrivono i risultati ottenuti. A questi dati sarebbe possibile e potenzialmente molto utile aggregare informazioni ottenute in tempo reale da sensori e macchinari per fare in modo che sia possibile reagire a cambiamenti improvvisi oppure per tenere una traccia più dettagliata dello stato del paziente.

Sebbene questi dati, anche se opportunamente resi disponibili da un sistema che possa garantire un adeguato livello di circolarità delle informazioni, possano essere analizzati manualmente da un medico, rimarrebbe comunque impossibile sfruttarli appieno. Sono diversi i fattori che rendono sconsigliabile o impraticabile scaricare interamente sui medici il peso dell'analisi di tale massiccia mole di dati. Si consideri che al di fuori delle visite programmate un medico non ha la possibilità di verificare lo stato di salute del paziente e in particolar modo non sarebbe possibile analizzare i dati trasmessi in tempo reale dai sensori posti su ogni paziente che ha in cura. Altri fattori comprendono l'esperienza personale e la possibilità di tenere in considerazione svariati fattori concomitanti ma riguardanti aree di specializzazione differenti. A questi si aggiunge la possibilità che il medico compia errori dovuti a disattenzioni oppure che il tempo che un medico può dedicare al singolo paziente sia troppo limitato rispetto alla complessità del caso (si consideri il caso della gestione dei traumi, in cui il medico è chiamato a gestire situazioni frenetiche).

Considerate queste problematiche, al fine di sfruttare la conoscenza presente negli ospedali e supportare i medici sia per rendere più veloci che più precise le diagnosi e la formulazione di decisioni, grande attenzione sta venendo posta verso il mondo del Cognitive Computing e del Machine Learning. Queste tecniche, e in generale tutto ciò che riguarda l'intelligenza artificiale, vedono in questi anni una rinnovata esplosione dal punto di vista della ricerca e degli investimenti, di cui una buona parte è proprio diretta all'area bio-medicale.

Attraverso dataset estratti da dati presenti in ospedali di tutto il mondo, l'uso della letteratura scientifica, la collaborazione di centri con grande esperienza in particolari aree della medicina e lo sfruttamento di opportune tecniche di learning, è possibile costruire sistemi che incapsulano l'esperienza pratica delle best practices di specifici settori. Da un certo punto di vista questi sistemi possono essere usati come una modalità per riutilizzare e distribuire expertise. L'unione dell'esperienza umana e delle potenzialità degli algorimi di learning aprono alla possibilità di costruire sistemi che variano

da strumenti atti a supportare studenti e medici alle prime armi a sistemi complessi che supportino medici esperti in aree di particolare difficoltà oggettiva.

L'obiettivo dei sistemi automatici impiegati in area medica non è quello di sostituire in alcuna maniera il medico e in generale il personale sanitario. Cognitive Computing e Machine Learning rimangono degli strumenti ad alto potenziale che permettono di formulare consigli e avvisi da sottoporre al personale esperto, che prenderà le decisioni che ritiene più opportune in totale libertà.

3.2 Ambiti toccati dalla letteratura

Gli ultimi anni rappresentano un periodo d'oro per il Machine Learning, e in particolar modo il Deep Learning, dovuto alla disponibilità di hardware specifico sufficientemente potente per l'esecuzione degli algoritmi, dalla spinta data da nuove scoperte in questo ambito e dagli investimenti attratti dai risultati promettenti ottenuti in applicazioni pratiche. Aziende storiche e nuove realtà stanno esplorando nuove occasioni di business relativamente ad applicazioni pratiche e solitamente piuttosto specifiche in ambito industriale, smart city, social, commerciale, eccetera.

Così come per sta accadendo in altri ambiti, è esplosa la ricerca relativa alle possibili applicazioni di tecniche di Machine Learning e Cognitive Computing anche per l'ambito bio-medicale. Insieme agli sforzi di ricerca stanno crescendo gli investimenti attratti dal grande valore del settore.

La ricerca si sta focalizzando in particolar modo su alcuni settori specifici della medicina. Fermo restando che le possibilità relative all'applicazione di queste tecniche rappresentano ancora oggetto di indagine, alcune aree sono stati aggredite fin da subito. Le caratteristiche che accomunano i domini trattati sono: presenza di problemi comuni e adatti alla risoluzione con le tecniche in questione, essere aree a particolare valore per gli istituti di medicina e quindi con alto valore di business e infine la presenza di una buona quantità di dati storici su cui eseguire tecniche di addestramento. L'obiettivo delle tecniche presentate è quello di fornire nuovi strumenti, eventualmente integrandoli con sistemi esistenti quali gli Electronic Health Records, che supportino l'attività dei medici e del personale sanitario nell'esecuzione delle proprie mansioni. Queste tecniche sono infatti sfruttate per costrui-

re sistemi specifici e in letteratura non sono presenti espliciti riferimenti all'integrazione di queste con forme di Personal Assistant Agents.

Come già accennato questi sistemi sono focalizzati nella risoluzione di problemi molto specifici ma allo stesso comuni all'interno dell'ambito biomedicale. In alcuni casi sono state le aziende informatiche, conducendo progetti atti ad esplorare le opportunità date dalle tecniche in questione, a individuare gli ambiti e quindi le problematiche a maggior valore e proporre soluzioni innovative alle aziende sanitarie. In altri casi la ricerca è stata spinta dagli stessi ospedali, solitamente al fine di risolvere problemi classici difficilmente affrontabili con altre tecniche.

Gli ambiti presi in considerazione dalla letteratura riguardano principalmente l'oncologia, la nefrologia, il trattamento e la prevenzione del diabete, la predizione di malattie cardiovascolari, la creazione di nuovi farmaci, la genetica. Le tecniche presentate riguardano l'analisi dei record contenenti risultati di esami o visite specialistiche, ricoveri, serie storiche di questi e altri eventi, immagini sia bidimensionali che tridimensionali, parametri vitali ottenuti in tempo reale, eccetera.

Nell'ambito della fase esplorativa è stato possibile individuare i risultati ottenuti attraverso l'applicazione di algoritmi di Machine Learning presenti in letteratura e i conseguimenti delle prime applicazioni di tecniche di Cognitive Computing.

3.3 Machine Learning

Il Machine Learning è un campo della computer science che si occupa dello studio di tecniche che permettano a una macchina di imparare a risolvere problemi senza venire programmate esplicitamente per risolverlo. In questo ambito rientrano quell'insieme di algoritmi che, dato un dataset di training composto da un insieme di record, riescono a costruire un modello del dominio al fine di poter eseguire predizioni inerenti a elementi della popolazione mai sottoposti all'algoritmo di learning.

Famiglie di tecniche comunemente utilizzate includono alberi di decisione, regole associative, reti bayesiane, macchine a vettori di supporto, clustering, reti neurali artificiali. È quest'ultima famiglia di tecniche, oggetto di studio del Deep Learning, a star ricevendo la spinta maggiore in termini di ricerca e investimenti per via della comprensibilità, facilità di applicazio-

ne e di risoluzione di problemi complessi e, fattore non meno importante, ampia disponibilità nel domino pubblico di algoritmi e framework necessari per la loro applicazione. Tuttavia, come esposto in seguito, SVM e relative varianti risultano ancora essere tra le tecniche meglio performanti quando i dati trattati non comprendono immagini.

3.3.0.1 Il mercato del Machine Learning

In generale è possibile individuare due differenti approcci che le aziende informatiche hanno deciso di adottare per quanto riguarda il Machine Learning: l'applicazione diretta delle tecniche in questione e quindi la costruzione dello strumento finale da consegnare al committente oppure la creazione di supporti per l'esecuzione degli algoritmi.

Nella prima direzione vanno piccole, medie aziende e startup che collaborano direttamente con gli istituti sanitari o le industrie e solitamente si occupano di risolvere problemi specifici.

La seconda direzione è quella intrapresa dalle grandi aziende. Fermo restando che è comune che dipartimenti di queste aziende si occupino anche di aspetti relativi alla risoluzione di problematiche specifiche (specie in collaborazione con realtà importanti, anche al fine di guadagnarne in visibilità), il business inerente al Machine Learning in questo caso è legato alla creazione di hardware, software e soluzioni cloud dedicate.

Dal punto di vista hardware già da tempo sono presenti sul mercato soluzioni basate su schede video atte a migliorare le performance degli algoritmi di Deep Learning. A questi si aggiungono dispositivi più recenti e compatti con le stesse finalità, ad esempio sotto forma di chiavette USB. Questa famiglia di algoritmi ha trovato applicazioni pratiche che anche nell'ambito dell'elettronica consumer. La novità dell'ultimo periodo riguardano infatti la spinta ad integrare all'interno di processori e SoC hardware dedicato per l'accelerazione di algoritmi di Deep Learning. In generale le caratteristiche che legano le soluzioni hardware sul mercato sono due: la capacità di eseguire gli algoritmi in questione molto più velocemente rispetto a quanto possibile utilizzando un sistema di elaborazione sequenziale e l'ottimizzato consumo energetico. [17] [10]

Dal punto di vista software sul mercato sono presenti framework e librerie, solitamente disponibili in forma gratuita e open ma in alcuni casi anche a pagamento, che includono algoritmi utilizzabili out-of-the-box già otti-

mizzati anche per l'uso di hardware dedicato. Lo sviluppo di queste librerie vede l'appoggio di una comunità che si è venuta a creare negli ultimi anni e che si è concentrata su Python e R come linguaggi di programmazione di riferimento. Esempi di queste librerie sono TensorFlow e PyTorch, anche se è possibile implementare gli algoritmi manualmente utilizzando librerie di più basso livello come NumPy.

Da questo punto di vista l'opportunità di business per le grandi aziende consiste nell'offrire soluzioni cloud PaaS che includono le suddette librerie e framework e altri strumenti proprietari per il Machine Learning, eventualmente già integrati con servizi inerenti al mondo dei Big Data offerti dallo stesso provider. Per alcune aree di mercato, incluso l'ambito Healthcare, queste aziende offrono soluzioni cloud-based dedicate ma comunque aspecifiche rispetto a quanto prodotto da aziende specializzate nel settore. Le aree toccate da queste soluzioni riguardano solitamente l'analytics e più raramente includono sistemi predittivi costruibili tramite procedure guidate e che fanno uso di meccanismi black-box.

3.3.0.2 Training

Un elemento che accomuna tutti questi algoritmi è l'essere basati su di una fase di training il cui successo è condizionato dalla disponibilità di una buona quantità di dati per l'addestramento. Oltre al dover essere dati di buona qualità, essi devono possibilmente coprire tutte le casistiche riguardanti i membri della popolazione analizzata.

Il training, impostato da sviluppatori umani, rappresenta la parte più onerosa nella creazione di un prodotto di Machine Learning in termini di tempi, potenza computazionale e risorse umane: una volta completato l'addestramento l'esecuzione del codice inerente al modello ottenuto risulta solitamente veloce. Gli obiettivi di questi algoritmi comprendono la classificazione, la regressione e il clustering con un tipo di apprendimento che può essere supervisionato, non supervisionato e per rinforzo.

Compito degli sviluppatori è quello di analizzare i dati disponibili, estrarne dei data set, eseguire il preprocessing ed eseguire una riduzione di dimensionalità, gestire la fase di addestramento, validare il modello ed infine integrare la soluzione ottenuta all'interno del software finale. Come appena accennato il successo dipende fortemente dai dati resi disponibili dall'istituto collaborante: se i dati con cui eseguire l'addestramento sono in quantità

insufficiente, non presentano feature importanti, non coprono tutte le casistiche possibili oppure non sono stati raccolti in maniera corretta allora diventa complesso se non impossibile proseguire con tecniche di Machine Learning. Al contrario, se i dati disponibili risultano qualitativamente e quantitativamente buoni allora è possibile ottenere buoni risultati. La grande promessa del Deep Learning consiste nel poter ottenere da un algoritmo semplice ma addestrato su molti dati performance migliori rispetto a quanto possibile tramite l'applicazione di tecniche complesse anche se studiate ad hoc.

3.3.0.3 Approcci all'ambito medico

Sono diversi gli approcci con cui vengono affrontati i problemi inerenti all'ambito medico. Questi dipendono fortemente dai dati utilizzabili e dagli obiettivi, che solitamente consistono nella predizione e individuazione di malattie insieme al relativo stato di avanzamento, la famiglia specifica e la gravità. Comunemente negli ospedali sono disponibili record relativi a esami di laboratorio, frequenza e risultati di visite specialistiche, prescrizioni, immagini di radiografie o risonanze e altri eventi importanti nel trattamento della malattia considerata.

Gli approcci che si trovano in letteratura utilizzano sia algoritmi classici come gli alberi decisionali, le macchine a vettori di supporto e le reti bayesiane, sia le reti neurali artificiali.

Di seguito una breve panoramica delle tecniche e dei risultati ottenuti nelle aree più esplorate dalla letteratura. Da notare come non siano presenti ricerche significative per quanto riguarda la gestione dei traumi e in generale per tutte quelle aree della medicina in cui i medici potrebbero richiedere supporto in tempo reale.

Insufficienza cardiaca L'area delle malattie cardiovascolari sta ricevendo grande attenzione. Queste rappresentano la principale causa di morte in Italia (41% dei decessi [4]) e nei paesi sviluppati. In particolare è possibile individuare nella letteratura una certa attenzione per la predizione dei casi di Insufficienza Cardiaca. Questa ha una probabilità di insorgenza nella popolazione sopra i quaranta anni pari al 20% [1] e rappresenta la causa del 3-5% delle ammissioni ospedaliere comportando costi per la sanità pari che si aggirano su una media del 2% del budget del sistema sanitario dei paesi

sviluppati [23]. Nei pazienti a cui viene diagnosticata, la mortalità risulta essere pari al 50% nei cinque anni successivi. Si tratta pertanto di una condizione particolarmente studiata sia per quanto riguarda la sua incidenza sulla aspettativa e qualità della vita dei pazienti, sia per quanto riguarda gli aspetti economici correlati al suo trattamento.

I lavori pertanto si incentrano sull'applicazione di tecniche di Machine Learning al fine di:

- Diagnosticare l'Insufficienza Cardiaca.
- Classificare il tipo di Insufficienza Cardiaca.
- Stimare la severità della condizione.
- Predire destabilizzazioni, re-ospedalizzazioni e mortalità.

Le principali tecniche utilizzate includono, in ordine di frequenza d'uso: Support Vector Machines, Random Forest, Neural Networks, C4.5, kNN, Hidden Markov Models. Nella maggior parte dei casi le feature utilizzate includono la Hearth Rate Variability, End Systolic Volume Index, altri dati vitali raccolti dagli strumenti e, in particolar modo per la predizione di destabilizzazioni, re-ospedalizzazioni e morte, informazioni riguardanti visite, numero e tempo intercorso tra le riammissioni, report delle dimissioni, eccetera.

I dataset comunemente utilizzati includono sia dati di pazienti sani che quelli di casi dove la malattia è stata effettivamente riscontrata. Quasi tutte le pubblicazioni trattano l'applicazione delle tecniche sopra citate su dataset piuttosto esigui, solitamente contenenti cartelle cliniche di meno di cento pazienti. In alcuni casi sono state impiegate tecniche generative per la costruzione di istanze aggiuntive ma la maggior parte dei dati proviene dagli EHR di diversi ospedali. [23] [21]

Diabete Mellito Diversi studi sono stati condotti al fine di creare sistemi in grado di predire, utilizzando tecniche di Machine Learning e Data Mining, l'insorgenza del diabete, di complicazioni e la sequenza di farmaci da somministrare. A questo si aggiungono studi atti a individuare i fattori genetici e ambientali che più influenzano l'insorgenza e il decorso della malattia. Gli algoritmi di classificazioni più utilizzati risultano essere SVM, ANN e gli alberi di decisione.

Segmentazione La segmentazione risulta essere un'area ad alto potenziale. Questa è solitamente svolta attraverso tecniche di Deep Learning ed è applicabile in diversi ambiti della medicina dove la nefrologia risulta essere la più attiva in quanto studi ed applicazioni pratiche, in particolar modo nell'individuazione del volume del rene colecistico.

Pneumologia La ricerca è particolarmente attiva anche nell'ambito della pneumologia. In questa area è da segnalare il progetto CheXNet, una Convolutional Neural Network a 121 strati in grado di individuare diverse malattie che colpiscono i polmoni con una performance, misurata in F1, maggiore di quella del radiologo medio.

3.3.0.4 Conclusioni

In alcuni casi gli algoritmi presentano un'accuratezza maggiore di quella ottenuta da esperti umani.

In generale le tecniche di Machine Learning possono essere usate in ogni contesto in cui sia possibile sfruttare una buona quantità di dati. Permangono problematiche relative alla disponibilità di questi dovute all'assenza di dati storici memorizzati elettronicamente e al costo che presenta la creazione di nuovi dataset specificamente realizzati per permetter l'addestramento di algoritmi. Solo una minima parte di questi dataset viene pubblicata.

In conclusione, utilizzare tecniche di Machine Learning significa estrarre expertise dai dati. Le performance degli algoritmi sono tanto migliori quanto è maggiore e varia la quantità di dati a disposizione. Una volta completato l'addestramento i risultati possono essere utilizzati anche da istituti diversi, di fatto portando a distribuire l'expertise raccolto nelle realtà che rappresentano le best practices del settore. Ne segue che i medici e i centri con meno esperienza nel trattamento dei casi coperti possono trarre particolare vantaggio da questi strumenti.

A seguito della fase esplorativa si è giunti alla conclusione che le tecniche della famiglia del Machine Learning sono adatte all'ambito di Trauma Tracker: durante la gestione della pratica viene prodotto un gran numero di dati strutturati ed è presente un corposo archivio di informazioni su di cui basare la fase di addestramento.

Un approccio di più ampio respiro che prevede l'uso combinato di diverse tecniche della computer science, tra le quali figurano il Machine Learning, il Text Mining e la Visione Artificiale, è dato dal Cognitive Computing.

3.4 Piattaforme Cognitive

"Cognitive computing refers to systems that learn at scale, reason with purpose and interact with humans naturally. Rather than being explicitly programmed, they learn and reason from their interactions with us and from their experiences with their environment." [18]

Questo il modo con cui IBM presenta il Cognitive computing. Si tratta di un nuovo approccio alla creazione di sistemi di supporto all'uomo di tipo cooperativo e interattivo. L'obiettivo, particolarmente ambizioso, consiste nel permettere di superare ostacoli umani dati dalla impossibilità di analizzare in tempi brevi una grande quantità di dati rappresentati nelle forme più disparate al fine di ottenere una conoscenza di sintesi utilizzabile. I sistemi che si rifanno a questo paradigma sono improntati più all'utilizzo pratico che alla creazione di intelligenze artificiali.

"The success of cognitive computing will not be measured by Turing tests or a computer's ability to mimic humans. It will be measured in more practical ways, like return of investment, new market opportunities, diseases cured and lives saved." [18]

Come deducibile dall'obiettivo di massima, il Cognitive Computing non rappresenta una particolare tecnica ma bensì un approccio strutturato. Alla base vi è l'integrazione di un insieme di tecniche inerenti a diverse aree della computer science, compreso il Machine Learning, al fine ottenere l'effetto desiderato.

In seguito ai primi risultati ottenuti dal progetto Watson di IBM, diverse altre compagnie si sono affacciate a questo mondo proponendo i propri prodotti. Questi hanno in comune il fatto di essere resi disponibili sotto forma di "piattaforme cognitive", ovvero piattaforme rese disponibili come servizio cloud e che possono essere sfruttate da applicativi di diversa natura. Nel valutare se il Cognitive Computing sia da considerarsi una tecnica adatta al contesto Trauma Tracker è stata svolta un'esplorazione dell'ambito e in particolar modo sono state individuate le applicazioni di queste tecniche in ambito medico.

3.4.0.1 Vision

Cognitive computing is "... an emerging research topic inspired by a vision of how the unification [of machine learning and naturalistic input processing] could lead to a new generation of computing systems enabling genuine human machine collaboration" [22]

In base alla vision del Cognitive Computing, questa nuova famiglia di sistemi è destinata a rivoluzionare il mondo della computer science. In particolare ci si riferisce all'alba di una nuova era, la "Cognitive era", contrapposta alle due ere precedenti: la Tabulating e la Programming Era.

La Tabulating Era, per la quale viene individuato un periodo di riferimento che va dal 1900 al 1940, rappresenta l'era della nascita dei primi sistemi di elaborazione. In questo periodo i computer consistevano in elaboratori limitati dal fatto di poter eseguire le operazioni di input e ouput per mezzo di schede perforate. Si tratta quindi di sistemi meccanici.

L'era successiva, denominata Programming Era, copre gli anni successivi fino ai giorni nostri. L'elaborazione in questa era è svolta da calcolatori elettronici programmabili. In questa categoria rientrano tutti quei sistemi che vanno dagli elaboratori creati prima dell'invenzione del transistor ai computer moderni. Caratteristica principale dei sistemi di elaborazione di quest'era è la programmabilità, ovvero la necessità di una programmazione esplicita per giungere agli obiettivi prefissati.

La terza era, di cui IBM ne individua l'inizio nell'anno 2011 per ragioni che verranno esposte di seguito, è denominata Cognitive Era. Le caratteristiche di questa nuova classe di elaboratori sono da ricercare nella visione di J.C.R. Licklider espressa negli anni sessanta. Nel paper "Man-Computer Symbiosis", Licklider asserisce che la simbiosi tra uomo e computer, concretizzata in una forma di interazione cooperativa, sia da considerarsi uno sviluppo naturale per la computer science. In particolare vengono individuati due scopi per questi sistemi:

"

1. To let computers facilitate formulative thinking as they now facilitate the solution of formulated problems, and

2. To enable men and computers to cooperate in making decisions and controlling complex situations without inflexible dependence on predetermined programs...

Preliminary analyses indicate that the symbiotic partnership will perform intellectual operations much more effectively than man alone can perform them." [19]

Da questa visione deriva che i sistemi in questione devono supportare l'uomo formando una collaborazione interattiva ovvero devono essere in grado di dare risposte e fornire supporto in maniera sensata anche in caso di domande e problematiche per cui il sistema non sia stato addestrato preventivamente.

L'obiettivo di questi sistemi non è quello di sostituire l'esperto umano nelle sue mansioni ma bensì supportarlo nel prendere decisioni che richiederebbero una conoscenza e una capacità di analisi dei dati al di fuori delle capacità umane. A tal fine i sistemi di Cognitive computing sono definiti come "probabilistici", da intendere come la capacità di selezionare più di una soluzione e presentare queste con il relativo grado di confidenza in modo da permettere all'uomo di scegliere quella che ritiene migliore.

IBM ha individuato 5 "core capabilities" per i sistemi cognitivi: [18]

1. Creano una profonda interazione con l'uomo.

   Questa facoltà riguarda il poter analizzare tutto ciò che riguarda una persona, come il suo stato d'animo, il posto in cui si trova, informazioni dai dispositivi wearable e la cronologia della navigazione, imparando ad agire nel migliore dei modi al fine per ottenere una interazione della miglior qualità possibile.

2. Scalano ed elevano l'expertise.

   Questi sistemi dovrebbero svolgere il ruolo di companion a supporto degli esperti che lavorano in settori in cui la conoscenza è in forte espansione. Questo porta all'ulteriore vantaggio di rendere disponibili a chiunque le best practices del settore. Ne deriva che questa classe di sistemi debba essere in grado di assimilare la letteratura scientifica, inclusa la comprensione dei termini tecnici tipici del dominio analizzato.

3. Infondono cognizione in prodotti e servizi.

   Secondo questa visione i sistemi cognitivi aprono la strada alla creazione di prodotti e servizi in grado di percepire, ragionare e imparare riguardo ai loro utenti e al mondo intorno ad essi.

4. Permettono processi e operazioni cognitive.

   Questa facoltà riguarda la maniera con cui può essere gestito il business di un'azienda. In particolare si tratta di utilizzare questi sistemi per incrementare l'efficienza analizzando l'ambiente, il contesto e il flusso di lavoro e supportando il processo decisionale analizzando una grande quantità di dati.

5. Intensificano esplorazione e scoperte.

   Alla base del Cognitive Computing vi è la promessa di poter analizzare una grande quantità di informazioni in modo da scoprire pattern e opportunità impossibili da rivelare utilizzando sistemi di ricerca tradizionale.

Al fine di coprire obiettivi così generalisti il sistema deve risultare particolarmente flessibile e adattabile ai possibili diversi contesti d'uso e settori.

Necessità derivate Al tal fine viene indicata la necessità di hardware e software dedicato. In particolare vi è la necessità di sistemi che rendano efficiente elaborare una grande quantità di dati in tempi brevi. Sempre secondo la vision, lo sviluppo di piattaforme per l'esecuzione, algoritmi e hardware per questi sistemi dovrebbe avvenire in concerto. Al momento vengono utilizzati sistemi usati anche in ambito Deep Learning, in particolar modo le schede video.

Dal punto di vista dell'interazione con l'utente gli strumenti abilitanti iniziano in questo periodo a diventare pervasivi nella vita di tutti i giorni sotto forma di smartphone, automobili, componenti per smart home, eccetera.

Dal punto di vista software vi è la necessità di orchestrare metodologie diverse da loro in modo da poter coprire i formati con cui gli umani memorizzano e scambiano tra loro la conoscenza al fine di ottenerne una versione unificata e potervi ragionare. I sistemi cognitivi dovrebbero prevedere tecniche per il Natural Language Processing, l'elaborazione di immagini e segnali audio, la gestione di dati strutturati e Big Data. A questi vanno aggiunti i più disparati algoritmi utilizzabili per estrarre e trattare la conoscenza, che costituirebbero il core del sistema.

3.4.0.2 Potenzialità

Al di là della effettiva fattibilità nel costruire nell'immediato futuro un sistema che copra tutti questi aspetti, le potenzialità del Cognitive Computing risultano immense.

La famiglia dei problemi risolvibili include tutte quelle situazioni in cui vi è la necessità di memorizzare e ragionare su dati in una quantità tale da risultare non gestibile dal singolo essere umano. Risulta fondamentale che i sistemi cognitivi siano in grado di eseguire inferenze su di una base di conoscenza costruita a partire dagli stessi dati generati e utilizzati nella comunicazione tra umani, cioè dati non strutturati. In prima battuta è possibile concentrarsi su informazioni veicolate dal testo in formato digitale, inclusa la letteratura scientifica. A questo sarebbe poi possibile affiancare l'estrazione di informazioni da immagini e audio, oltre che da Big Data di altra natura quali iterazioni sociali, posizione, eccetera.

Assistenza dell'esperto umano Un esempio a dimostrazione delle potenzialità del Cognitive Computing è rappresentato dall'ambiente intelligente. Si tratta di un ambiente in cui esperti umani, tramite schermi, microfoni e oggetti smart, possono interagire con un assistente virtuale basato sul paradigma del Cognitive Computing in modo da ottenere supporto durante lo svolgimento del proprio lavoro. Una applicazione pratica è stata creata da IBM nel Cognitive Environments Lab in cui ha costruito il sistema "Mergers and Acquisition". Tramite sensori che permettono il riconoscimento di immagini e l'isolazione delle voci dei partecipanti, Celia (Cognitive Environments Laboratory Intelligent Agent) ascolta le conversazioni al fine di identificare bisogni informativi e comandi. I risultati vengono visualizzati in tempo reale su di una serie di schermi. I partecipanti possono muovere gli oggetti virtuali utilizzando le possibilità touch offerte dagli schermi oppure attraverso degli strumenti chiamati "wand". Il fine ultimo è quello di offrire un ambiente in cui sia possibile prendere decisioni informate ottenendo dati in tempi brevi e presentandoli in maniera ottimale. [8]

In generale il Cognitive Computing ha come scopo quello di supportare esperti umani, da soli o in gruppo, nello svolgimento delle proprie mansioni. L'idea di un assistente personale o di un ambiente intelligente può essere generalizzata a qualsiasi ambito, incluso quello medicale. Risulta evidente

come vi siano forti intersezioni tra la vision alla base del Personal Medical Digital Assistant e quella del Cognitive Computing.

Al fine di valutare la fattibilità nell'integrare queste tecnologie nel progetto Trauma Tracker è stata condotto un approfondimento circa la piattaforma ritenuta più significativa in questo contesto: Watson di IBM.

3.4.0.3 Watson

Lo scopo del progetto Watson è quello di applicare la vision del Cognitive Computing al fine di creare un prodotto commerciabile flessibile da applicare ad ambiti particolarmente ricchi di informazioni da elaborare, in particolare medicina e genetica. Oltre alla sua storia e il funzionamento di massima, vengono esposte alcune delle sue applicazioni in modo da permettere al lettore di farsi un quadro delle potenzialità pratiche di questa tecnologia.

Storia L'idea che ha portato alla creazione di Watson risale al 2004 quando viene individuata una sfida particolarmente impegnativa: progettare un sistema in grado di competere con i campioni di Jeopardy!, un quiz televisivo in cui i concorrenti sono chiamati a rispondere a domande e risolvere puzzle particolarmente complessi per via della ampia gamma dei temi trattati e dalla complessità con cui vengono posti i quesiti. Probabilmente il progetto non nacque con l'intento di creare un sistema dalla così larga portata e generalizzazione prevista dalla vision del Cognitive Computing. Tuttavia appare evidente come il problema da affrontare preveda una soluzione con forti sovrapposizioni con tale vision.

L'idea viene accolta dal management di IBM che nel 2007 assembla un team di quindici ricercatori per sviluppare il sistema. L'accuratezza dei risultati del sistema migliora negli anni e nel 2010 è già in grado di competere con partecipanti umani.

La prima apparizione degna di nota risale al 2011 quando, come concorrente di una serie di tre puntate di Jeopardy!, riesce a battere i campioni Brad Rutter e Ken Jennings, i partecipanti con più vincite in denaro e con più presenze rispettivamente. La vittoria è data dall'abilità nel rispondere alle sole domande di cui si conosce la risposta, in una percentuale tale da permettere di superare il montepremi accumulato dagli altri concorrenti e rispondendo prima di questi. Tramite un'analisi delle partite passate, il team di Watson ha potuto delineare il profilo dei migliori giocatori e quindi tarare

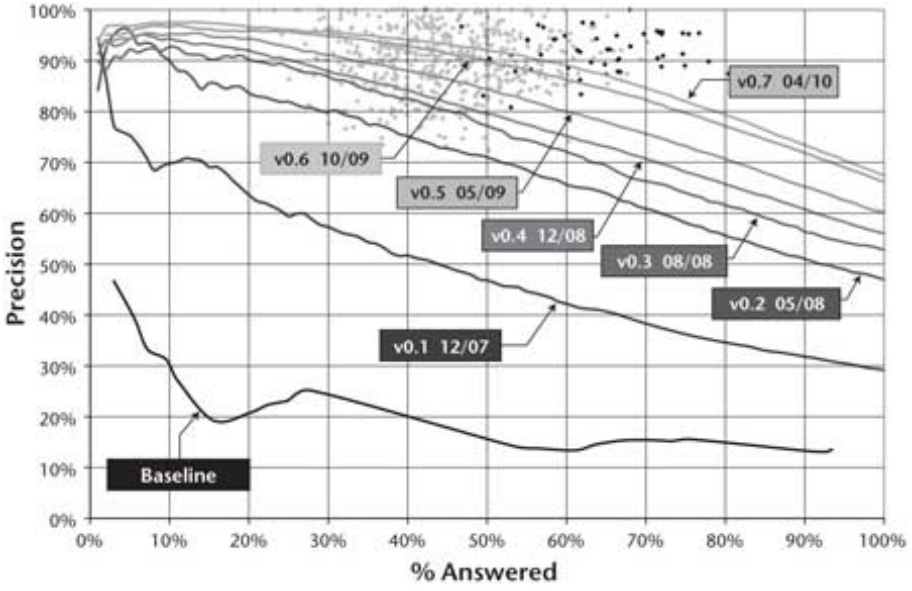

Figura 3.1: I risultati ottenuti con PIQUANT, un sistema di Question Answering di IBM considerato tra i migliori del 2007 (baseline) contro i risultati ottenuti nelle diverse versioni di Watson. [9]

l'intervallo di confidenza per il quale Watson avrebbe dovuto prenotare la risposta. Questa e altre strategie che considerano ulteriori informazioni quali la tipologia della domanda, il suo valore e la situazione della partita sono state impiegate per vincere le puntate. Watson ha partecipato alla competizione scollegato da internet, ma con una base di conoscenza particolarmente ampia che si stima occupasse circa 4 TB.

Il sistema che forma il core di Watson è DeepQA. Gli algoritmi alla base, sviluppati in maniera particolarmente specifica al fine di vincere Jeopardy!, sono stati continuamente migliorati e generalizzati per potersi confrontare con le sfide della Text REtrieval Conference. Vi era in fatti un forte interesse da parte di IBM nel poter applicare Watson a domini differenti. Dall'opera di generalizzazione è risultato che le ultime versioni di DeepQA possono essere applicate sia a Jeopardy! che alle sfide della TREC. Eventuali operazioni di adattamento atte a migliorare l'accuratezza delle risposte quando il sistema è impiegato in un dominio specifico possono comunque portare a performance migliori.

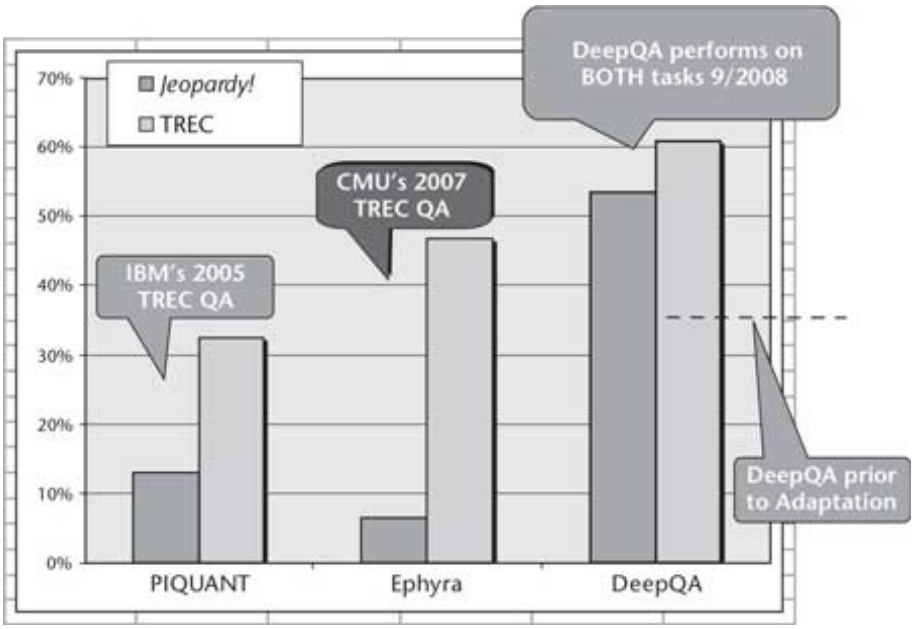

Figura 3.2: Una comparazione delle performance dei sistemi PIQUANT, Ephyra e DeepQA nel rispondere ai quesiti di Jeopardy! e alle sfide del TREC. [9]

Caratteristiche Watson è un sistema di Question Answering che incorpora tecniche di Natural Language Processing, Information Retrieval, Knowledge Representation, Automated Reasoning e Machine Learning. Il sistema è attualmente fornito sotto forma di servizio cloud facente parte della piattaforma Bluemix.

Watson è in grado di processare testo, audio, immagini e dati non strutturati per fornire diverse funzionalità. In particolare Watson mette a disposizione un insieme di API specifiche per servizio: i più semplici consistono nella traduzione, la classificazione e analisi di frammenti di testo, la conversione di documenti, la conversione di testo in voce e viceversa. I servizi più complessi includono conversazione, analisi vocale al fine di evidenziare il sentimento dell'interlocutore, riconoscimento di oggetti, analisi della personalità e, il servizio considerato più importante, il sistema di ricerca cognitiva "Discovery".

Quest'ultimo in particolare integra metodi e strumenti per: [16]

- L'ingestione, conversione, arricchimento e normalizzazione dei dati
- La loro memorizzazione e indicizzazione
- L'esecuzione di interrogazioni
- L'esplorazione e la visualizzazione del corpus e dei risultati delle interrogazioni

Considerando che i risultati ottenuti vengono utilizzati da esperti umani, il sistema di visualizzazione dei risultati risulta essere una componente fondamentale di Discovery. Questo permette sia di esporre i risultati in diverse forme, come una visualizzazione a grafo che lega la risposta agli elementi correlati, sia di visualizzare gli elementi che hanno portato a formulare quella risposta e i relativi pesi nella formulazione del punteggio di confidenza.

Funzionamento Nonostante la sua natura di sistema chiuso, sono disponibili diverse pubblicazioni che trattano il funzionamento interno di Watson. È possibile individuare due componenti principali: il sistema di ingestione e memorizzazione dei dati e DeepQA. A questo si aggiungono gli strumenti per la presentazione dei risultati e l'insieme di API accessibili programmaticamente.

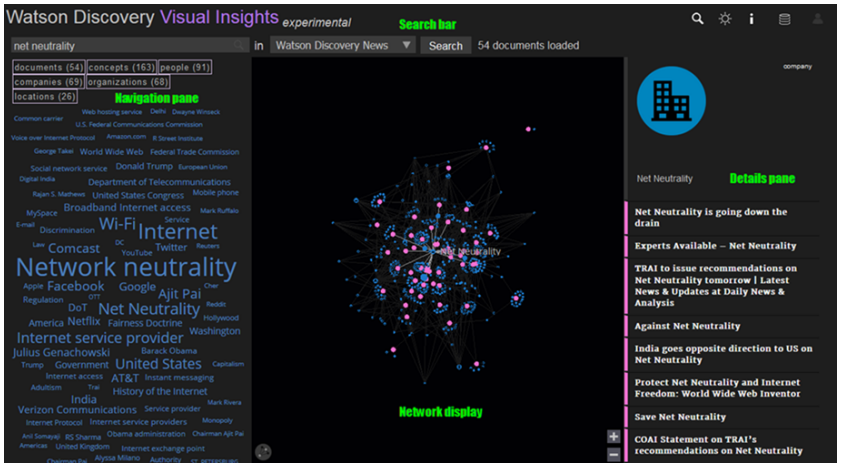

Figura 3.3: La schermata di Discovery Visual Insights, uno degli strumenti utilizzabili per la visualizzazione dei risultati delle interrogazioni. [15]

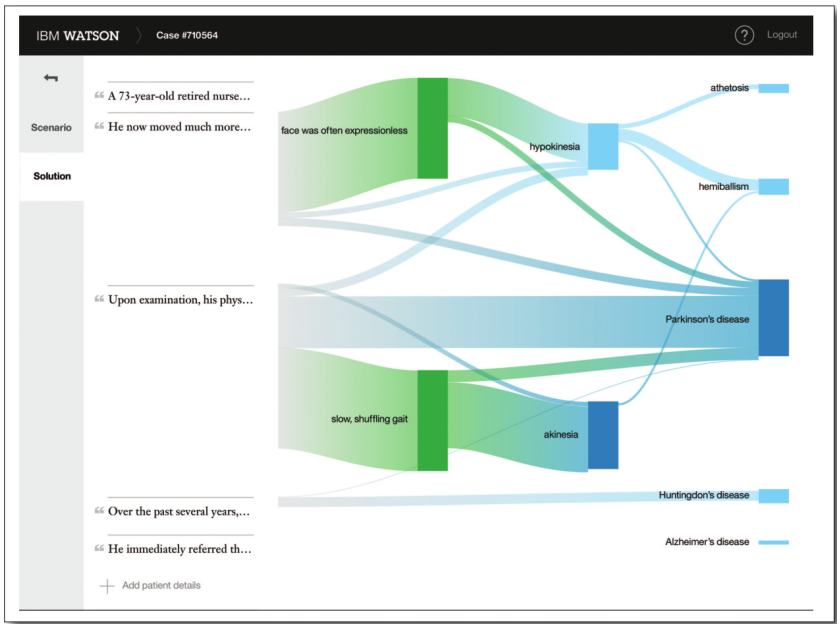

Figura 3.4: Un esempio di visualizzazione delle risposte candidate insieme ai relativi elementi a supporto. [22]

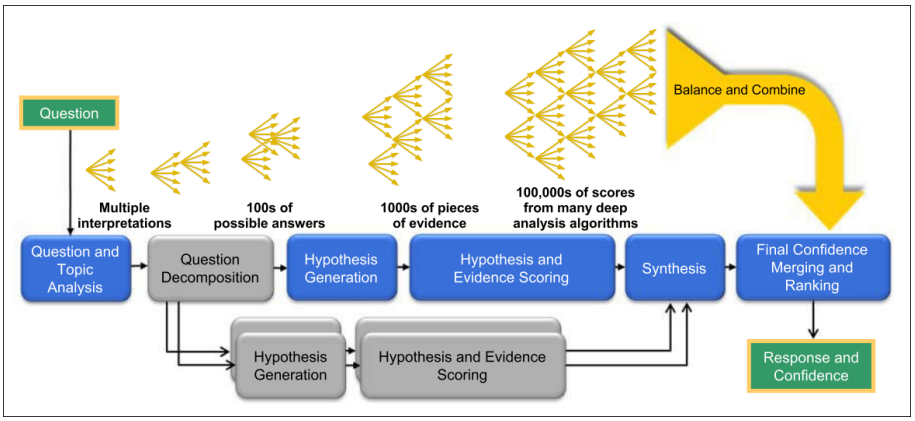

Figura 3.5: Una rappresentazione grafica della pipeline di Watson. [11]

La base di conoscenza è costituita da un insieme di dati, il corpus, costituito da dati non strutturati, in prevalenza letteratura scientifica, news, post di social network. Oltre ai semplici dati non strutturati Watson può sfruttare ontologie di dominio qualora disponibili, anche se questo non è strettamente necessario. Risulta di fondamentale importanza che i documenti coprano la conoscenza del dominio in cui opererà il sistema, almeno nella parte oggetto delle interrogazioni da parte degli esperti umani. Essendo difficile individuare le informazioni da considerare, che in ogni caso non devono risultare sovrabbondanti al fine di ottenere un sistema efficiente e accurato, devono essere individuate diverse coppie di domande e risposte attraverso le quali valutare se aggiungere o rimuovere elementi al corpus.

Internamente DeepQA è costituito da una pipeline.

1. Nel momento in cui viene effettuata una interrogazione, Watson applica tecniche di Natural Language Processing al fine di decomporre la domanda. Questa infatti può presentare richieste, collegamenti e indizi collegati tra loro che potrebbero comportare interrogazioni da risolvere indipendentemente.

2. Sempre attraverso tecniche di NLP vengono individuate le feature rilevanti della risposta attesa, degli indizi, dei vincoli, eccetera. Le feature vengono utilizzate da un motore Prolog per generare pattern di ricerca.

3. Lo step successivo consiste nella ricerca di risposte candidate. La generazione di candidati è principalmente effettuata attraverso tecniche di Text Mining e Web Semantico (Triple Stores, SPARQL, ...). L'obiettivo di questa fase è quello di generare, tra i possibili candidati, la risposta corretta. A tal fine il numero di candidati generati risulta essere particolarmente alto.

4. Una volta individuati i candidati il sistema esegue un filtraggio, basato su algoritmi di assegnazione del punteggio non computazionalmente onerosi, che mantiene i primi cento classificati.

5. Per ogni candidato viene eseguita una ricerca di prove a supporto. Anche in questo caso viene fatto uso di tecniche di Text Mining, NLP e l'uso di Triple Stores.

6. Le prove a supporto vengono analizzate da algoritmi di assegnazione dei punteggi. Questi risultano l'essere il cuore di DeepQA. In questa fase viene calcolata la confidenza della risposta rispetto alle tassonomie, la posizione geospaziale e temporale, l'affidabilità della fonte, correttezza di genere e numero, relazioni logiche, numero di volte in cui è utilizzata nello stesso contesto nel corpus, eccetera.

7. I risultati con i relativi punteggi normalizzati vengono resi disponibili.

Dopo aver individuato i dati con cui formare il corpus, Watson deve essere addestrato su diverse coppie di domande e risposte inerenti al dominio trattato. Questo serve a creare il modello con cui verranno pesati gli algoritmi di assegnazione dei punteggi. [9]

Progetti e applicazioni reali I primi e principali progetti in cui Watson è stato utilizzato riguardano l'ambito medicale.

WellPoint ha deciso di adottare questa piattaforma per di migliorare la velocità di gestione delle richieste di trattamento con lo scopo di generare suggerimenti utili al personale sanitario. Il sistema viene pertanto utilizzato come un sistema a supporto delle decisioni. [12] [14]

Il Memorial Sloan Kettering Cancer Center ha adottato Watson per di ottenere un sistema in grado di revisionare i casi di oncologia con il fine di supportare i medici attraverso la formulazione di consigli personalizzati riguardanti i trattamenti da attuare. [25] [24] [13]

Altri ambiti di applicazione includono la chimica farmaceutica. In questo ambito Watson è stato utilizzato per esplorare la ricerca scientifica. L'obiettivo era quello di individuare enzimi correlati al cancro e per identificare modalità per il riutilizzo di composti usati in ambito famaceutico. [3]

A partire da questo ambito Watson è stato adottato da diverse aziende che operano in diversi ambiti come quello delle telecomunicazioni, il supporto alla vendita e post-vendita, raccomandazione musicale, eccetera.

3.4.0.4 Conclusioni

L'ambito medico rappresenta un mercato particolarmente gettonato per quanto riguarda l'applicazione di tecniche cognitive. È stato possibile individuare diversi progetti reali riguardanti diversi settori della medicina che vedono l'uso di piattaforme cognitive e in particolar modo di Watson. La vision del Cognitive Computing ben si sposa con quella dei Personal Assistant Agent e sarebbe compatibile nell'ottica della generazione di avvisi e, in particolar modo, suggerimenti da mostrare ai medici del Trauma Team.

In generale non sono state individuate applicazioni specificamente pensate per l'ambito della gestione dei traumi e, come già riscontrato nell'esplorazione riguardante le tecniche di Machine Learning, non sono state individuate applicazioni reali in ambito medico riguardo aree che richiedono una forma di supporto in tempo reale: i principali progetti riguardano la formulazione di piani di cura con una prospettiva temporale di lungo termine.

Capitolo 4

Estensione del Trauma Assistant Agent con Cognitive Services

La componente progettuale della tesi si inserisce nell'ambito del progetto Trauma Tracker, in particolare nella parte riguardante la generazione di avvisi e suggerimenti da mostrare al Trauma Leader.

La logica dell'assistente a supporto dei medici è racchiusa nel Personal Medical Digital Assistant, un agente sviluppato secondo il paradigma Beliefs-Desires-Intentions. Un modulo per la generazione degli avvisi basato su regole è già integrato nel sistema. Tali regole sono staticamente codificate nel codice che descrive il comportamento dell'agente. Questo è stato sviluppato secondo il Procedural Reasoning System che prevede che il comportamento dell'agente venga codificato in un insieme di piani definiti dal programmatore. Le regole sono formulate dal personale del Trauma Center, che rappresentano gli utenti finali del sistema, e sono quindi basate sull'esperienza pratica. Queste riguardano principalmente la generazione di notifiche in grado di segnalare procedure non correttamente terminate o manovre non eseguite in tempi ottimali ma non contemplano elementi di predizione.

4.1 Obiettivi del progetto

Al fine di superare tale staticità e aumentare le capacità di inferenza del PMDA risulta necessario creare un sistema in grado di integrare il mondo degli agenti con quello delle tecniche di Cognitive Computing e Machine Learning. In particolar modo questa componente deve risultare dinamica, facilmente evolvibile o sostituibile, di facile integrazione con il sistema esistente e i cui risultati debbano essere basati sull'osservazione di casi passati.

La prima forma di espansione delle capacità cognitive dell'agente consiste nell'integrazione di algoritmi in grado di ottenere, previo un addestramento su dati storici, previsioni che possano rivelarsi utili al personale medico. Una seconda fase potrebbe essere costituita dall'uso di algoritmi in grado di considerare informazioni apprese durante l'uso del sistema. Di questa famiglia fanno parte quelle tecniche che non richiedono l'intervento umano quali tecniche di learning per rinforzo o sistemi cognitivi che permettano di integrare dinamicamente nuovi dati nella propria base di conoscenza.

Il Cognitive Computing risulta essere un approccio con grande potenzialità e meritevole di un'esplorazione più approfondita anche in considerazione del fatto che le prime applicazioni reali riguardano proprio l'ambito medicale. Risulta tuttavia difficile approcciarsi ai prodotti presenti sul mercato. Durante una prima fase focalizzata sull'esplorazione delle pratiche e tecniche comunemente messe in campo in ambito sanitario è stato studiato il funzionamento di Watson e del relativo modello di business. Nel valutare se adottare o meno questa tecnologia sono stati riscontrati ostacoli quali la complessità organizzativa e tecnologica insiti nella creazione di un progetto che includa questa componente, i relativi costi e infine l'alta probabilità che un progetto in tal senso richieda una collaborazione con esperti esterni. L'integrazione con sistemi di questa famiglia è stata ritenuta rimandabile a fasi successive.

Come emerso dalla letteratura diverse problematiche in ambito medicale vengono attualmente risolte tramite tecniche di Machine Learning. In particolar modo sarebbe possibile approntare un sistema che elabori sia i dati storici del paziente che quelli ottenuti durante la gestione del trauma con l'obiettivo di individuare situazioni da segnalare.

Al fine di tenere aperte tutte le possibilità è stato deciso di creare un modulo in grado di integrare queste e altre tecniche al sistema esistente.

L'architettura deve farsi carico di ricevere gli eventi (inclusi i dati relativi ai parametri vitali) riguardanti la gestione di un trauma, elaborare i dati e restituire risultati che andranno ad aggiungersi alla base di conoscenza del PMDA.

4.2 Requisiti

Il modulo deve integrarsi con l'ecosistema aperto che costituisce la componente di backend, mantenere traccia di tutti gli eventi al fine di supportare qualsiasi forma di elaborazione si voglia mettere in campo in futuro, prevedere diverse fonti di dati e le interfacce con cui comunicare con esse, gestire più pazienti contemporaneamente, supportare diverse e concomitanti tecnologie di elaborazione e infine restituire quanto ottenuto in una forma compatibile con il sistema di rappresentazione della conoscenza dell'agente. Il risultato di ogni algoritmo, componente che varierà a seconda del problema da risolvere e verrà considerato una black-box, potrà essere usato all'interno del sistema esistente implementato nel "Warning Generator Agent", di fatto andando a costituire una espansione delle capacità di ragionamento dell'agente.

Di seguito la lista dei requisiti individuati:

- Integrazione con il sistema di backend esistente.

  Il sistema di backend esistente consiste in applicativi per Java Virtual Machine in esecuzione su sistemi Linux sviluppati utilizzando il framework "Vert.x".

- Mantenere traccia di tutti gli eventi.

  Tutti gli eventi e le informazioni utili devono essere memorizzati, nessuno escluso. Questo è necessario al fine di supportare un qualsiasi algoritmo predittivo risulterà necessario sviluppare in futuro.

- Permettere agli algoritmi predittivi di utilizzare tali dati.

  Gli algoritmi basano le proprie predizioni sulla base di conoscenza formata da tutti i dati memorizzati nell'ambito del trattamento del trauma.

- Prevedere diverse fonti dati. Sviluppare adeguati meccanismi con cui ricevere informazioni da esse.

  Questo requisito descrive la necessità di rendere il sistema il più possibile indipendente rispetto alle fonti dei dati: il modulo da produrre non dovrà essere legato alle fonti dati disponibili al momento dello sviluppo.

- Gestire più pazienti.

  Pur trattandosi di un'ovvietà è bene considerare che Trauma Tracker deve poter gestire più pazienti contemporaneamente.

- Permettere l'esecuzione di più algoritmi per paziente.

- Supportare diverse tecnologie per gli algoritmi predittivi.

  Il sistema deve supportare l'esecuzione di più algoritmi predittivi contemporaneamente. Questi potrebbero avere necessità differenti in termini di tecnologie. Questo significa che il sistema finale dovrebbe prevedere metodi per poter eseguire qualsiasi genere di algoritmo.

- Restituire i risultati in una forma compatibile con il sistema di rappresentazione della conoscenza adottato dagli agenti.

  Questo rappresenta il requisito vitale del sistema: i risultati ottenuti dagli algoritmi devono essere integrabili nella Belief Base degli agenti in modo da poter essere considerati nei piani esistenti.

4.3 Analisi

Considerate queste premesse è possibile individuare diverse problematiche e le relative soluzioni:

- Non è possibile conoscere quali saranno le fonti dati da considerare in futuro.

  Non conoscendo la natura delle fonti segue che la componente che si occupa di ottenere la lista degli eventi non potrà agire attivamente interrogando direttamente le fonti: non è possibile fare altro che attendere i dati. Va considerato che le informazioni relative agli eventi

raggiungono il backend attraverso una componente già sviluppata a cui potrebbe essere possibile interfacciarsi.

- Gli eventi memorizzati devono essere accessibili dagli algoritmi predittivi.

  La soluzione consiste nell'adottare tecnologie di memorizzazione che permettano l'interrogazione dei dati memorizzati da parte di più componenti in contemporanea.

- Non è possibile prevedere quali tecnologie verranno utilizzate per lo sviluppo degli algoritmi predittivi. Queste potrebbero risultare non installabili nel sistema finale.

  Le uniche soluzioni individuate sono: inserire vincoli tecnologici riguardanti gli algoritmi eseguibili oppure prevedere forme di virtualizzazione o containerizzazione in grado di permettere un'adeguata indipendenza tecnologica.

- Integrazione con il sistema di rappresentazione della conoscenza esistente.

  Questa forma deve essere il più possibile generale in modo da richiedere poco sforzo nell'integrazione di agenti che sfruttano forme di rappresentazione differenti. La soluzione prevede che la conoscenza venga gestita dal sistema nella forma esistente, cioè sotto forma di una teoria Prolog, per poi essere trasformata nel formato richiesto da un algoritmo di estrazione e preprocessing dei dati approntato dal creatore dell'algoritmo predittivo. Ala stessa maniera i risultati ottenuti devono essere trasformati in un fatto Prolog.

4.4 Architettura del sistema

A partire dai requisiti e dalle problematiche individuate è stato possibile individuare una architettura di sistema. Questa considera le componenti già esistenti nel sistema descrivendo come viene integrato il sotto-sistema che realizza le funzionalità richieste.

Da un'analisi ad alto livello il sistema esistente è rappresentato dal PMDA, che agisce come entità autonoma, e da un insieme di componenti che rappresentano il modo con cui questo interagisce con l'ambiente,

permettendo di fatto di costruire un contesto individuato da una singola pratica.

Come precedentemente descritto si intende integrare il mondo delle tecnologie cognitive con quello degli agenti espandendo le capacità di ragionamento dell'agente. La soluzione più naturale consisterebbe nell'integrare negli agenti esistenti la componente in grado di attuare l'unione. Questo tuttavia non risulta possibile viste problematiche di tipo tecnologico: gli agenti che costituiscono il PMDA sono in esecuzione in un ambiente Android, cosa che pone diverse limitazioni. Non è inoltre possibile spostare la logica di generazione degli avvisi sul servizio di backend in quanto si desidera che il sistema basato su regole attualmente esistente possa operare il più possibile disaccoppiato da questo. La scelta di mantenere la logica del PMDA sul dispositivo Android garantirebbe inoltre migliori prestazioni in caso di una connessione di rete scadente e permetterebbe di garantire il servizio anche in totale assenza di questa.

Dalle problematiche appena esposte deriva la decisione di implementare una componente in grado di eseguire i diversi algoritmi integrandola nel servizio di backend e non nell'applicativo Android. Una integrazione ben congegnata permetterebbe comunque di raggiungere il risultato atteso anche con una componente non in esecuzione nello stesso ambiente degli agenti BDI, senza perdita di generalità. Ovviamente insorgono problematiche di più basso livello come quelle legate alla connessione di rete oltre che alla necessità di gestire correttamente la conoscenza dall'agente in modo che sia possibile sfruttarla negli algoritmi.

La realizzazione di questa componente, denominata "Cognitive Service", è l'oggetto del progetto di tesi. Costruire un Cognitive Service in esecuzione in un ambiente diverso dall'applicazione Android in cui esegue il "Warning Generato Agent" presenta alcuni vantaggi così come alcuni svantaggi. Tra i vantaggi figurano la possibilità di far evolvere indipendentemente dall'app Android il servizio e gli algoritmi, la possibilità di utilizzare qualsiasi tecnologia necessaria nell'implementazione dei modelli, le migliori prestazioni possibili dati dalla possibilità di usufruire di una potenza di calcolo e capacità di memorizzazione non disponibili in un dispositivo mobile, la possibilità di adottare migliori sistemi per la gestione dei fallimenti e infine l'eliminazione della problematica relativa alla limitata autonomia di questi dispositivi. Tra gli svantaggi figurano la necessità di far comunicare le componenti attraverso una connessione di rete, cosa che può influire sulla

qualità del servizio, e l'aumentata complessità data alla necessità di creare una apposita interfaccia per mettere in comunicazione gli agenti e il Cognitive Service. Va comunque considerato che diversi dei dati relativi ai sensori vengono inviati al PMDA proprio dal servizio di backend per cui queste problematiche non impattano in maniera negativa sulla struttura e qualità del servizio finale attualmente offerti.

4.5 Il Cognitive Service

Di seguito viene descritto il ruolo e la struttura del Cognitive Service, componente che si integra al backend del sistema Trauma Tracker e la cui realizzazione costituisce l'oggetto del progetto.

4.5.1 Cos'è il Cognitive Service

Il Cognitive Service è un sotto-sistema integrato nel servizio di backend esistente denominato "TT Infrastructure" e ha come obiettivo quello di permettere l'esecuzione di qualsiasi algoritmo predittivo ritenuto necessario nella generazione di avvisi e suggerimenti a partire dalla conoscenza accumulata dal PMDA. Tale conoscenza rappresenta tutto ciò che è conosciuto di un trauma e deriva da dati provenienti da svariati sensori, dall'input manuale di informazioni del personale, da sorgenti dati quali EHR o altri repository che includono informazioni inerenti al paziente.

4.5.2 Ruolo e funzionalità attese

Il Cognitive Service permette di costruire e mantenere aggiornata una copia della conoscenza accumulata nel PMDA integrandosi al sistema di backend esistente e in particolar modo alla componente "TT Service".

Questa conoscenza può essere sfruttata dai più disparati algoritmi come base per produrre nuove informazioni utilizzabili dal PMDA. In particolar modo si prevede di utilizzare le informazioni prodotte nel "Warning Generator Agent" ma di fatto questa potrebbe essere utilizzata anche in altre componenti del sistema.

Il Cognitive Service si fa carico di mandare in esecuzione i diversi algoritmi indipendentemente dalla loro natura e dai loro requisiti tecnici. Il Service fornisce a questi le modalità per interrogare la base di conoscenza e

rendere disponibili i risultati prodotti in modo da poter essere utilizzati dal PMDA. Lo sviluppatore dell'algoritmo deve unicamente occuparsi di stabilire quali sono i dati necessari, trasformarli dalla forma di rappresentazione Prolog-based adottata e infine restituire il risultato, lasciando al sistema il compito di attuare l'integrazione con gli agenti BDI che compongono il PMDA.

Da questo punto di vista il Cognitive Service, pur mantenendo come obiettivo originale quello di attuare l'integrazione tra agenti software e tecniche cognitive, rappresenta una piattaforma di cloud privato che può essere utilizzata per effettuare generiche elaborazioni nel contesto di una singola pratica.

4.5.3 Architettura del Cognitive Service

Dai requisiti precedentemente esposti è possibile individuare l'architettura risolutiva esposta nella figura REFS. Di seguito una descrizione delle componenti e le tecnologie individuate per la loro implementazione:

- TT Cognitive

  Questa componente, sviluppata in Java utilizzando il framework Vert.x, si integra al modulo TT Service che riceve le informazioni dalle diverse fonte dati. TT Service, modulo già esistente, deve essere modificato al fine di rendere disponibili eventi e dati relativi a tutti i pazienti su un apposito Event Bus messo a disposizione dal suddetto framework.

- Cognitive Manager

  Tiene traccia delle pratiche aperte e il riferimento ai Practice Manager. Il Cognitive Manager inoltra le informazioni riguardanti lo specifico paziente ottenute dall'Event Bus al relativo Practice Manager.

- Practice Manager

  Questa componente mantiene la base di conoscenza riguardante la gestione di un singolo trauma e gestisce l'esecuzione degli algoritmi predittivi. Questi possono interrogare il Practice Manager al fine di ottenere le informazioni necessarie e per inviare i risultati dell'elaborazione. La base di conoscenza verrà gestita compatibilmente con il metodo di rappresentazione della conoscenza utilizzato degli agenti Jason, ovvero tramite un motore Prolog.

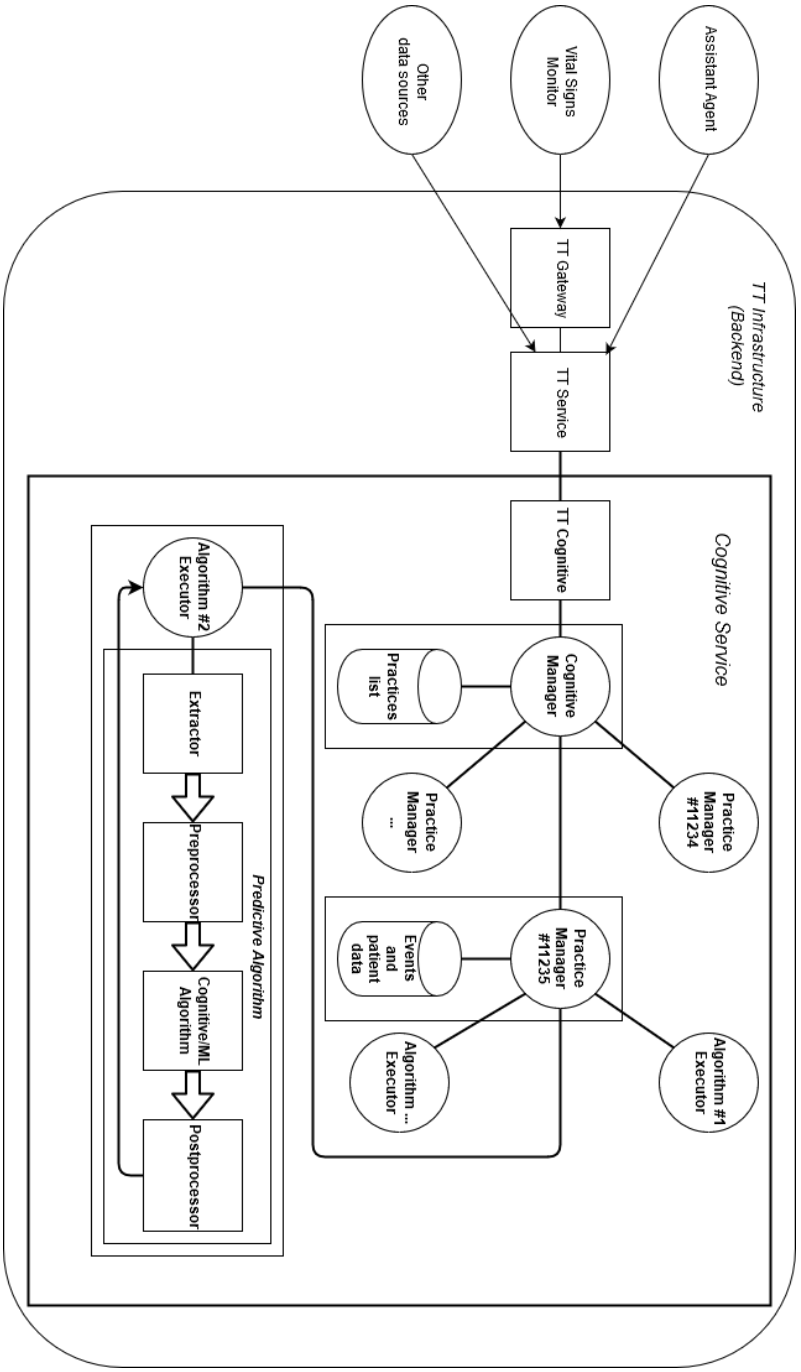

Figura 4.1: L'architettura proposta per il sistema finale.

- Algorithm Executor

  L'Executor si occupa di mandare in esecuzione, tramite il sistema per la gestione di container Docker, l'algoritmo predittivo e catturarne l'output da inviare al Practice Manager. A tal fine questa componente espone interfacce per permettere al Predictive Algorithm di ottenere le informazioni necessarie per l'elaborazione e comunicare il risultato.

- Predictive Algorithm

  Gli algoritmi, essendo mandati in esecuzione sfruttando il sistema a container di Docker, possono essere basati su qualsiasi tecnologia disponibile per l'ambiente Linux. Le immagini Docker potranno essere strutturate liberamente dai programmatori che hanno approntato gli algoritmi. È stata individuata la possibilità di costruire una libreria che supporti gli sviluppatori nell'integrazione con il Cognitive Service. Questa si articola in:

  - Estrazione dei dati. La libreria, data una lista di interrogazioni (nel formato previsto da Prolog) da effettuare, si occupa di interrogare la base di conoscenza al fine di rendere disponibili i dati per effettuarne il preprocessing.
  - Preprocessing. In questa fase è necessario trasformare quanto ottenuto dalle interrogazioni per renderlo compatibile con quanto atteso in input dall'algoritmo predittivo. La libreria a supporto rende disponibili i risultati ottenuti nella fase di estrazione sotto forma di Stream.
  - Invio dell'output. La libreria si occupa di raccogliere il risultato dell'algoritmo e inviarlo all'Algorithm Executor.

Questa architettura soddisfa tutti i requisiti espressi precedentemente. Il risultato dell'algoritmo predittivo viene trasformato in formato Prolog risultando compatibile con il metodo di rappresentazione della conoscenza utilizzato in JaCa-Android e quindi dal PMDA e, considerata la capacità espressiva del Prolog, potrebbe essere opportunamente trasformata per risultare compatibile con altre forme di rappresentazione della conoscenza. Il risultato degli algoritmi raggiunge l'agente sotto forma di evento.

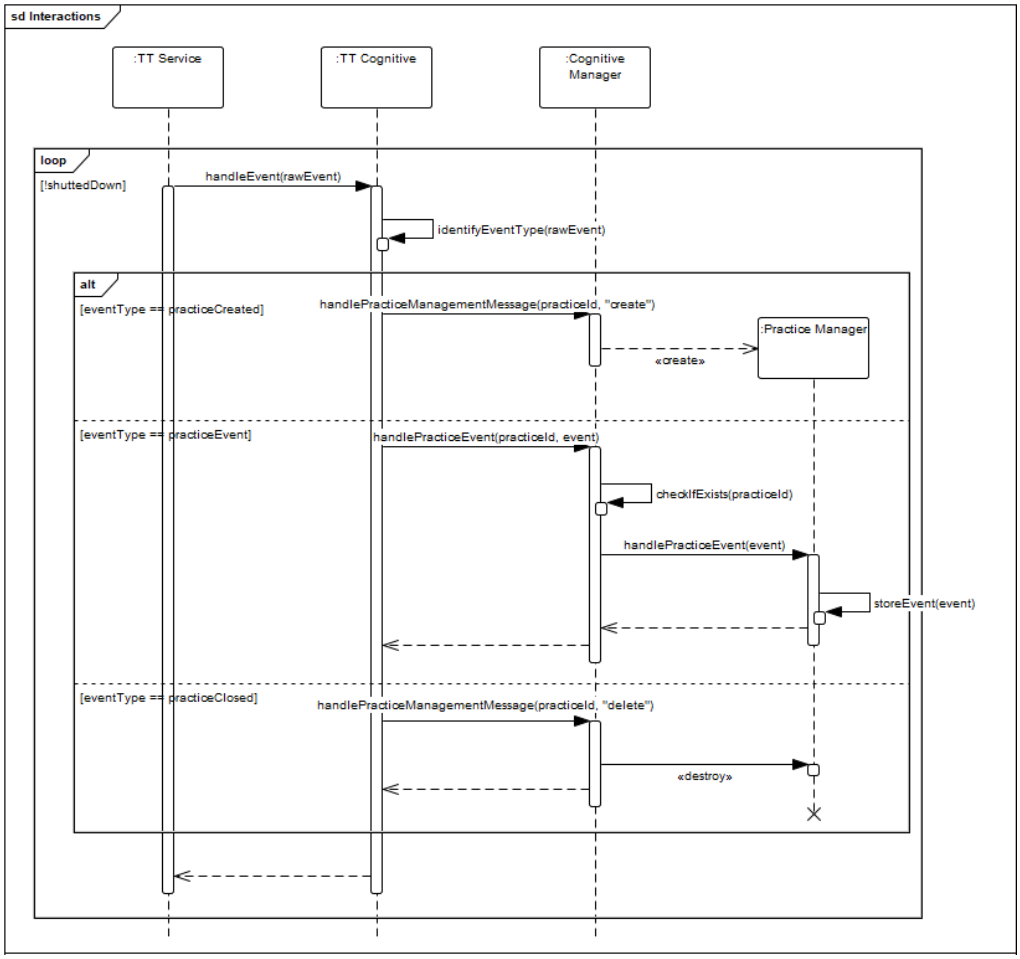

Figura 4.2: Interazioni attese tra le componenti durante il ciclo di vita di una pratica.

4.6 Modellazione del trauma

Il PMDA si muove nel contesto dalla gestione di un trauma. La conoscenza è costituita da informazioni personali inerenti al paziente, la lista degli operatori e dei responsabili, dati provenienti da sensori di diversa natura (parametri vitali, identificazione dell'ambiente ospedaliero, eccetera), una lista di azioni effettuate dal Trauma Team, l'insieme degli input manualmente inseriti che descrivono ulteriori aspetti circa lo stato del paziente o della pratica.

Sono diversi i problemi approcciabili tramite tecniche di Machine Learning e di Cognitive Computing. Per gli algoritmi sviluppati a risoluzione di uno specifico problema è di interesse solamente un sottoinsieme dei dati a disposizione. In generale i dati raccolti che vanno a formare la base di conoscenza del PMDA sono di diversa natura e non tutte le sorgenti dati che verranno integrate nel sistema finale sono state definite o implementate. È tuttavia possibile delineare un modello di massima, estrapolabile dal ciclo di vita di una pratica, con cui rappresentare la conoscenza accumulata per un trauma.

Il trauma inizia con l'accoglienza e identificazione del paziente. In questa fase viene aperta la pratica e vengono inserite informazioni di gestione come l'orario di apertura, il responsabile del procedimento insieme a una prima valutazione dello stato del paziente che permette una migliore gestione dei casi. Grazie all'identificazione del paziente è possibile ottenere, ove ciò è reso possibile dal sistema gestionale dell'ospedale, informazioni provenienti dagli Electronic Health Records. Queste informazioni sono utili in ogni situazione ma possono rivelarsi veramente preziose nel caso in cui il paziente presenti trascorsi clinici o condizioni particolari. Le informazioni identificate fin qui descrivono lo "stato iniziale" della pratica.

Da questo momento si susseguono un insieme di eventi. Questi possono consistere in azioni effettuate dal personale (esecuzione di manovre, somministrazione di sostanze), l'aggiornamento delle letture dei sensori e l'inserimento di input manuali riguardanti valutazioni circa lo stato del paziente. Il trauma si conclude con un'azione finale con cui viene chiusa la pratica e vengono inserite alcune informazioni conclusive. Questi vanno a formare lo "stato finale".

4.6.1 I dati rilevanti per un algoritmo predittivo

I dati più rilevanti per un algoritmo predittivo certamente includono:

- Il valore dei parametri vitali in un certo istante. Queste informazioni possono essere particolarmente utili per predire come potrebbe evolversi lo stato oppure per eseguire una classificazione della condizione del paziente.
- Informazioni anagrafiche del paziente e la conoscenza di patologie o altre condizioni pregresse. Questi dati possono aiutare a discriminare la tipologia di manovre effettuabili e sostanze somministrabili: alcune azioni potrebbero essere controindicate in pazienti con certe patologie.
- Le manovre effettuate durante la gestione del trauma.

Sarebbe tuttavia possibile, oltre che al considerare una singola fotografia dello stato del paziente, includere la sequenza temporale degli eventi per effettuare predizioni. Potrebbe essere utile valutare le ultime rilevazioni di un parametro vitale oppure basare alcune elaborazioni su statistiche descrittive circa il suo andamento, oppure ancora considerare l'ordine con cui sono state effettuate certe manovre. Infatti, mentre alcuni problemi possono essere risolti tramite l'analisi dello stato attuale del paziente e senza considerare eventi precedenti, per altri è impossibile ignorare elementi quali le manovre precedentemente effettuate e le sostanze somministrate. Altri problemi inoltre potrebbero essere risolvibili solamente considerando l'andamento nel tempo di una o più variabili o addirittura il tempo intercorso tra alcuni eventi.

Risulta anche difficile definire in maniera univoca lo "stato" del paziente in un certo momento: includere nella descrizione dello stato solamente i parametri vitali è quantomeno limitativo. Vanno infatti considerate manovre effettuate in precedenza e che hanno un effetto al momento attuale. Si consideri a titolo di esempio l'intubazione. Anche il fatto di aver iniettato in precedenza un farmaco può entrare a far parte dello stato del paziente dal momento in cui questa sostanza ha un qualche tipo di effetto al momento e lo avrà anche nel futuro.

Pur avendo definito in qualche maniera un concetto di "stato" attuale rimane irrisolta la problematica alla base: lo stato del paziente non è causalmente legato allo stato precedente e agli eventi registrati a meno di non

includere nello stato informazioni non disponibili e non rilevabili in alcuna maniera. In buona parte il paziente si comporta come una "black box" il cui comportamento e stato interno sono inesplorabili.

Sta al creatore dell'algoritmo decidere quali sono le informazioni rilevanti tra quelle disponibili e creare un proprio modello per lo specifico problema da risolvere. Per capire quali sono le possibilità offerte in termini di informazioni utilizzabili è necessario individuare i dati trattati dal sistema, le sorgenti dati e come la conoscenza è attualmente rappresentata.

4.6.2 Le informazioni a disposizione e le sorgenti dati

Le informazioni attualmente a disposizione sono:

- Gli input forniti dal Trauma Team attraverso l'app Android. Questi riguardano le manovre effettuate, le sostanze somministrate inclusi i dosaggi, valutazioni circa lo stato del paziente solitamente sotto forma di indici.

- Le rilevazioni dei diversi parametri vitali provenienti da macchinari. Sono stati approntate componenti che, integrandosi con questi, riescono a estrarre i dati e inviarli alla "TT Infrastructure". Questi dati vengono poi inviati al PMDA.

- Informazioni provenienti da sensori integrabili direttamente con l'app Android. Grazie a questi è possibile stabilire la posizione del paziente nell'ambiente ospedaliero. Questa informazione può essere rilevante anche ai fini della generazione dei suggerimenti tanto da essere già parte delle regole formulate dal Trauma Team.

In futuro, con le successive evoluzioni del progetto, potrebbero essere disponibili le informazioni ottenute tramite EHR.

4.6.3 Rappresentazione della conoscenza

Le informazioni provenienti dalle diverse sorgenti dati vengono accumulate nella base di conoscenza del PMDA. In questa le informazioni sono memorizzate e interrogabili attraverso un modello molto simile a quello Prolog.

Da qui la naturale scelta di mantenere, all'interno del Cognitive Service, questa forma per rappresentare la conoscenza che quindi viene codificata come teoria Prolog. Sia gli eventi che le conoscenze pregresse (quali i dati provenienti da cartelle cliniche e le informazioni inserite all'apertura della pratica) vengono registrate come un insieme di fatti.

In base all'algoritmo da sviluppare sarà necessario definire quali sono i dati necessari (feature) e quindi interrogare opportunamente la base di conoscenza per ottenerli. Questa operazione può essere fatta periodicamente oppure ogni qual volta viene registrato un nuovo evento. Effettuare delle interrogazioni su questa teoria Prolog e restituire risultati che verranno inseriti in questa è di fatto equivalente a interagire con la base di conoscenza del PMDA espandendo le capacità di ragionamento degli agenti. La teoria rappresenta una copia della Belief Base sia dal punto di vista delle informazioni contenute sia dal punto di vista della forma con cui viene rappresentata la conoscenza.

Considerato che il sistema sarà chiamato a gestire elaborazioni di natura diversa, che in questa fase non risultano del tutto prevedibili, è d'obbligo mantenere e mettere a disposizione tutte le informazioni prodotte. Questo porta a una base di conoscenza che si espande finché la pratica non viene chiusa, momento in cui di fatto viene eliminata. La progettazione e implementazione di criteri per il filtraggio e la pulizia di questa, operazioni che dovrebbero considerare quali sono le informazioni le richieste dai diversi algoritmi, costituiscono un interessante spunto per una successiva evoluzione del Cognitive Service.

Capitolo 5

Tecnologie impiegate

In questo capitolo vengono esposte le principali tecnologie impiegate per lo sviluppo del progetto. La decisione di impiegare queste è stata dettata dai requisiti precedentemente esposti. Questo insieme di librerie, strumenti e framework condividono il fatto di essere ampiamente conosciute tanto da rappresentare uno standard quando si tratta di risolvere alcune famiglie di problemi ricorrenti. Queste tecnologie vedono una grande comunità di utilizzatori e sviluppatori impegnati nella loro evoluzione, cosa che porta ad avere garanzie circa la loro stabilità.

Il sistema è basato su di una architettura in cui le singole componenti del Cognitive Service, algoritmi predittivi inclusi, sono modellate come microservizi. In questa ottica e considerati i requisiti di sistema sono state individuate tecnologie che, come successivamente esposto, perfettamente supportano la creazione (framework Vert.x) e gestione (attraverso un sistema di containerizzazione) di questa tipologia di servizi. Vi sono ovviamente altre ragioni che hanno portato a favorire queste tecnologie.

La scelta di adottare il framework Vert.x per lo sviluppo della componente di backend tiene in considerazione il fatto che questo viene utilizzato nelle altre componenti già in uso nel progetto Trauma Tracker. In questo modo viene favorita una uniformità dal punto di vista delle tecnologie impiegate. Oltre a questo le operazioni di integrazione con il sistema esistente risulterebbero semplificate. Come già accennato, il framework rappresenta un'ottima soluzione per lo sviluppo di microservizi e in generale per la creazione di sistemi software con componenti loosely coupled.

Un sistema di containerizzazione rappresenta la naturale soluzione al

problema dell'indipendenza tecnologica degli algoritmi predittivi dall'ambiente di esecuzione offerto dal backend. Questo sistema è stato preferito a uno basato su macchine virtuali per motivi legati all'efficienza, facilità d'uso e in generale perché presenta caratteristiche che si addicono perfettamente alla risoluzione della problematica da affrontare. Questa strategia permette di gestire il deployment di microservizi in maniera ottimale rendendo agevole la creazione e distruzione di istanze di questi, l'automazione nella gestione del loro ciclo di vita e infine garantendo un overhead decisamente basso in termini di risorse computazionali. La tecnologia di containerizzazione scelta è Docker per via della sua popolarità, stabilità e facilità di configurazione e uso.

5.1 Vert.x

Vert.x è un framework per Java Virtual Machine per lo sviluppo di applicativi basato su un modello a event loop. Grazie a questo è possibile sviluppare in maniera particolarmente agevole sistemi basati su microservizi.

Vert.x è un framework poliglotta in quanto supporta l'esecuzione di applicativi scritti in una grande varietà di linguaggi quali Java, Scala, Kotlin, JavaScript, Groovy, Ruby e altri. I linguaggi di programmazione maggiormente supportati sono quelli JVM-based: applicativi scritti in JavaScript vengono eseguiti all'interno dell'ambiente di esecuzione Nashorn integrato nella JVM e pertanto sono soggetti a diverse limitazioni in termini di librerie utilizzabili, livello del linguaggio supportato, eccetera. [6]

Non devono stupire le similarità tra NodeJS e Vert.x in termini di stile di programmazione e modello per la gestione della concorrenza: Vert.x, il cui progetto era inizialmente denominato "Node.x", nasce infatti con il dichiarato intento di costruire un framework alternativo ma poliglotta in grado di eseguire sulla JVM.

5.1.1 Architettura del framework

Al fine di poter comprendere i motivi del successo di Vert.x è necessario entrare nel merito identificando le funzionalità offerte per lo sviluppo delle applicazioni, lo stile di programmazione di riferimento e in particolar modo il modello di concorrenza e i sistemi offerti a supporto della modularità.

Questi elementi possono fare la differenza nella scelta di un framework da utilizzare quando il sistema da sviluppare è scomponibile in parti semplici. Va comunque precisato che Vert.x è un framework generalista per cui può essere utilizzato per sviluppare applicativi di tutt'altra natura.

I servizi, chiamati "Verticle", vengono mandati in esecuzione in una istanza Vert.x. Ogni istanza di Vert.x supporta l'esecuzione di più Verticle. Per ogni JVM possono coesistere più istanze di Vert.x ma gli applicativi in esecuzione su Vert.x differenti non possono comunicare tra loro a meno che non venga eseguita una configurazione apposita oppure vengano usati meccanismi legati al linguaggio di programmazione utilizzato.

5.1.1.1 Modello di concorrenza

La logica dei Verticle è implementata secondo il modello a event loop. Da questo punto di vista Vert.x è molto simile a NodeJS, anche per la maniera con cui vengono richiamati i metodi asincroni ed eseguiti gli handler che ricevono i risultati delle elaborazioni o delle operazioni di I/O. Vert.x, A differenza di NodeJS, permette di definire esplicitamente operazioni bloccanti che verranno eseguite da un flusso di controllo dedicato servito da un apposito thread pool.

Vi è tuttavia una differenza tra i due modelli di concorrenza: mentre il codice applicativo in esecuzione in NodeJS viene eseguito da un solo flusso di controllo, in Vert.x sono presenti più flussi in esecuzione contemporaneamente. Per applicativi semplici quali microservizi ciò non rappresenta un problema in quanto il modello di concorrenza, denominato "Multi-Reactor Pattern", prevede che sia lo stesso flusso di controllo che ha lanciato l'operazione asincrona ad eseguire il codice dell'"handler" per la gestione del risultato. Da questo deriva che una richiesta possa essere servita completamente nei confini dello stesso flusso di controllo.

La scelta di implementare un sistema con più flussi porta a evidenti vantaggi legati all'uso delle risorse computazionali disponibili permettendo una migliore scalabilità verticale. Eventuali problemi nella gestione della concorrenza possono presentarsi qualora vi sia necessità di accedere a strutture dati condivise.

Al fine di garantire una corretta integrazione con Vert.x, per diverse tecnologie popolari quali MongoDB e RabbitMQ sono state creati wrapper basati sulle librerie ufficiali. Queste presentano un modello solitamen-

te incompatibile con quello richiesto da Vert.x. I wrapper permettono di trasformare il modello offerto in modo da garantire compatibilità con il framework.

5.1.1.2 Modularità e Event Bus

I servizi non hanno riferimenti agli altri Verticle in esecuzione all'interno della stessa istanza di Vert.x. Le comunicazioni tra i Verticle avvengono indirettamente tramite la propagazione di eventi gestita da un apposito Event Bus. Questo permette di costruire sistemi modulari con componenti loosely coupled. Altre forme di comunicazione, quali l'uso di campi statici o oggetti condivisi, sono da considerarsi anti-pattern.

Vert.x prevede inoltre un sistema di clustering. Questo permette di mettere in comunicazione diverse istanze di Vert.x al fine di creare sistemi distribuiti. Costruire un cluster comporta che gli eventi inviati attraverso l'Event Bus siano visibili al di fuori dei confini del Vert.x in cui vengono prodotti. Questo sistema permette di ottenere sistemi le cui componenti possono essere messe in esecuzione anche su più elaboratori connessi in rete.

Maggiori informazioni circa il funzionamento e le caratteristiche dell'Event Bus sono disponibili nel capitolo inerente all'implementazione.

5.1.2 Inquadramento all'interno del progetto

Come già accennato Vert.x rappresenta la tecnologia di riferimento per i sistemi di Trauma Tracker già implementati. La modularità offerta dal framework per lo sviluppo dei Verticle e il sistema di comunicazione basato su Event Bus, opportunamente orchestrati con le tecnologie successivamente esposte, hanno permesso di sviluppare il progetto in maniera indipendente e l'integrazione ha richiesto modifiche minime al sistema esistente.

5.2 Docker

Docker è un sistema per l'automatizzazione del deployment di applicativi. Il sistema sfrutta le funzioni di containerizzazione offerte dal kernel Linux al fine di isolare i singoli applicativi permettendo loro di eseguire in un

ambiente definito dal creatore dell'applicativo potenzialmente radicalmente differente da quello offerto dal sistema ospitante.

5.2.1 Funzionalità e API offerte

Docker è un progetto open source nato nel 2013 che è diventato in breve tempo uno standard nella gestione di servizi cloud PaaS e per il packaging e rilascio di applicativi. In relazione a quest'ultimo aspetto i sistemi basati su container stanno andando parzialmente a mitigare il pervasivo uso delle macchine virtuali, che classicamente rappresentano lo strumento utilizzato per garantire la compatibilità del software prodotto quando viene consegnato e installato nel sistema finale.

Impiegare una macchina virtuale presenta problemi quali i prerequisiti in termini di hardware, la necessità di configurare il sistema operativo ospitante e l'hypervisor, la necessità di installare e configurare il sistema ospite, l'amministrazione dell'intero sistema, un considerevole footprint per l'esecuzione del sistema virtualizzato oltre a problematiche legate alle performance dei sistemi virtualizzati, la loro configurazione di rete, eccetera.

Anziché fare affidamento alla virtualizzazione, Docker sfrutta le capacità del kernel Linux (e recentemente anche di Windows) di isolare i processi. Questo isolamento, chiamato "containerizzazione", riguarda la memoria, il file system, le variabili d'ambiente, la rete, gli account utente e più in generale qualsiasi risorsa gestita dal sistema operativo e normalmente visibile dai processi. Quello che si ottiene è un ambiente di esecuzione in cui solamente il kernel è condiviso.

Da questo deriva che tramite Docker è possibile eseguire su un qualsiasi sistema Linux (in grado di supportare la libreria "libcontainer") applicativi che normalmente richiedono distribuzioni Linux, pacchetti, configurazioni d'ambiente, file system differenti da quello del sistema operativo ospitante. Questo permette di sviluppare, trasportare nel sistema finale e infine mandare in esecuzione applicativi che eseguiranno nello stesso ambiente di esecuzione del sistema in cui sono stati sviluppati e testati ma senza il footprint e i tempi richiesti da una macchina virtuale: un container docker può essere mandato in esecuzione in pochi millisecondi e la memoria impiegata è quella occupata dai processi in esecuzione al suo interno.

5.2.1.1 Astrazioni offerte

Parte del successo di Docker è dato dalla facilità con cui è possibile definire, costruire, trasportare e riutilizzare gli ambienti di esecuzione per gli applicativi.

L'ambiente di esecuzione per un container è dato da una "immagine". Queste rappresentano una istantanea del file system e della configurazione del sistema in cui verranno eseguiti gli applicativi. Le immagini vengono costruite a partire da un Dockerfile. Il Dockerfile inizia solitamente con una dichiarazione "FROM otherimage" che definisce qual è l'immagine di partenza da cui ereditare la composizione del file system e le configurazioni. A questa seguono una lista di azioni da eseguire sul file system e sulle configurazioni di sistema racchiuse nell'immagine. Dopo l'esecuzione di ogni azione Docker memorizza i cambiamenti apportati andando a formare un nuovo strato. Le azioni espresse nel Dockerfile solitamente riguardano file da copiare dal sistema del creatore all'interno dell'immagine, la configurazione delle variabili d'ambiente, comandi da eseguire (ad esempio apt-get), dichiarazioni relative alle porte esposte e a cartelle del file system montabili sul sistema ospitante.

Una volta eseguite tutte le azioni espresse nel Dockerfile l'immagine creata è utilizzabile per creare istanze di container. Per facilitare la sua gestione questa può essere etichettata con un nome e un "tag" ed eventualmente caricata sul Docker Hub o su di un repository privato.

Un punto di forza di Docker consiste nella grande varietà di immagini ufficiali, aggiornate e liberamente utilizzabili presenti sul repository pubblico. Queste includono immagini base delle distribuzioni Linux più comuni, applicativi quali DBMS, broker di messaggi, server web, eccetera. Ad esempio, per poter installare e mandare in esecuzione una istanza di MongoDB è sufficiente eseguire da terminale il comando "docker run mongo".

I container possono essere connessi tra di loro attraverso delle reti. Ogni rete ha un id univoco e un nome user-firendly deciso dal creatore e permette ai processi in esecuzione nei container di comunicare tra di loro. La visibilità è garantita da un servizio DNS che risolve i nomi dei container in indirizzi IP interni. Questo risulta particolarmente utile nel caso in cui il sistema sia composto da più applicativi in esecuzione in più container. Un banale esempio è dato dalla situazione in cui un web server è in esecuzione in un primo container mentre in un secondo si trova un DBMS. Oltre alle

comunicazioni intra-Docker i container possono dichiarare, nel proprio Dockerfile, delle porte che è possibile esporre. Su queste porte sono solitamente in ascolto i servizi offerti dal container. È possibile configurare opportunamente Docker affinché queste porte vengano esposte su una scheda di rete dell'host e quindi aprendo alla possibilità di rendere accessibili i container da internet.

I container possono dichiarare dei volumi. Questi sono percorsi prefissati del file system su cui il container memorizza i propri dati e che è possibile legare a un'area di memoria permanente nel sistema ospitante. Questo rappresenta il sistema con cui ottenere la persistenza dei dati. I dati memorizzati dai processi in esecuzione nel container, se non memorizzati in un apposito volume, non sopravvivono all'eliminazione del container.

Queste sono solo alcune delle funzionalità offerte da Docker. La facile comprensibilità dei meccanismi alla base, delle astrazioni offerte e la facilità di utilizzo lo rendono una tecnologia ideale per la gestione del rilascio di applicativi. Di seguito vengono esposti i metodi con cui è possibile interagire con una istanza Docker e come questo si comporta in sistemi non Linux.

5.2.1.2 API di Docker

Docker rimane in esecuzione come demone ed è accessibile via terminale tramite il comando "docker" oppure attraverso delle API REST documentate da una specifica OpenAPI (Swagger). Quest'ultima rappresenta il metodo con cui interagire programmaticamente con una istanza Docker mentre il comando "docker" dovrebbe esclusivamente venir utilizzato da operatori umani in quanto non restituisce risultati machine-readable. L'endpoint delle API è solitamente accessibile utilizzando la Unix Socket "/var/run/docker.sock" oppure configurando il demone per esporre l'endpoint in maniera differente.

5.2.1.3 Docker su sistemi non Linux

Al fine di utilizzare Docker su sistemi non Linux sono presenti strumenti quali "Docker for Windows" e "Docker for Mac". Gli installer di questi creano una virtual machine invisibile all'utente basata sulla distribuzione "boot2docker", un sistema operativo dal peso di 27 MegaByte in grado di avviarsi in circa 5 secondi che ha come unico scopo quello di ospitare una

istanza di Docker. Questa macchina virtuale può venir eseguita su una istanza di Hyper-V o VirtualBox, a seconda delle esigenze.

A parte questo non vi sono particolari differenze con la versione per Linux: i comandi inviati da terminale tramite "docker" vengono diretti al demone presente sulla macchina virtuale e l'endpoint REST viene esposto all'IP della macchina virtuale a cui è possibile accedere tramite una scheda di rete virtuale.

5.2.2 Inquadramento all'interno del progetto

L'ambiente sui cui verrà installato il sistema potrebbe non disporre degli strumenti richiesti per la esecuzione degli algoritmi (librerie, servizi, ...). Come già accennato in fase di analisi dei requisiti, forzare lo sviluppo degli algoritmi predittivi in maniera che vangano utilizzate solamente le tecnologie presenti nel sistema finale oppure, viceversa, installare nel sistema finale qualsiasi strumento richiesto dai diversi algoritmi rappresentano entrambe soluzioni impraticabili.

Da qui la scelta di adottare Docker per la loro esecuzione. Questo permette grande libertà nel loro sviluppo e allo stesso tempo consente di mantenere il sistema finale pulito. L'unico requisito per il sistema finale diventa quello di aver installato Docker, operazione che può essere effettuata una tantum, garantendo un sistema facile da configurare e manutenere.

Conseguentemente a questa scelta è stato deciso di utilizzare Docker per gestire l'esecuzione di quasi tutte le componenti del progetto, eliminando ulteriori requisiti tecnologici minori riguardanti il sistema finale e semplificando l'integrazione con il sistema esistente. Ulteriori informazioni riguardo questo aspetto sono presentate nel capitolo riguardante l'implementazione.

5.2.3 La libreria Docker Client

Al fine di utilizzare le funzionalità di Docker è stata utilizzata la libreria Docker Client sviluppata da Spotify. La libreria esegue un wrapping delle API REST eseguendo le necessarie operazioni di object mapping delle richieste e delle risposte semplificando particolarmente l'interazione con il servizio Docker. Oltre a questo la libreria supporta i diversi sistemi di autenticazione e criptazione delle comunicazioni supportati Docker, compreso l'uso di certificati.

5.3 RabbitMQ

RabbitMQ è uno dei più famosi Message Oriented Middleware. Il protocollo di riferimento è rappresentato da AMQP 0-9-1 ma sono supportate diverse versioni di questo oltre che ad altri protocolli quali MQTT e STOMP.

Le principali caratteristiche sono:

- Supporto a diversi livelli di affidabilità: RabbitMQ può essere configurato in maniera tale da garantire al mittente che i messaggi siano stati sicuramente memorizzati in una memoria persistente, che siano stati inviati al sistema di I/O del sistema operativo oppure semplicemente che siano stati ricevuti ma la loro memorizzazione avverrà quando possibile. Questo e altre configurazioni permettono di decidere qual è il compromesso tra prestazioni e affidabilità migliore per una applicazione che fa uso di questo sistema.
- Diverse astrazioni per il routing dei messaggi: RabbitMQ permette di definire delle vere e proprie topologie per quanto riguarda i canali di comunicazione. Questo significa che vengono supportati sia i sistemi più semplici, basati su singole code di messaggi, che quelli più complessi, basati su multiplexing/demultiplexing su più code, oltre che alla ricezione di messaggi dato un pattern.
- Clustering e federation: RabbitMQ permette creare cluster che permettano di garantire ridondanza dei dati e alta disponibilità agli applicativi che utilizzano il servizio anche quando gli elaboratori che formano il cluster sono connessi attraverso una connessione di rete non affidabile.

5.3.1 Inquadramento all'interno del progetto

RabbitMQ viene utilizzato per sopperire ad alcune carenze del sistema di Event Bus di Vert.x. L'uso di un MOM permette di garantire che i messaggi vengano consegnati anche quando la controparte non è disponibile per la ricezione garantendo un miglior disaccoppiamento, che le comunicazioni non vengano perse grazie all'uso dei meccanismi di conferma della memorizzazione e infine che i singoli messaggi vengano considerati consegnati solamente dopo che le operazioni relative alla loro gestione siano state interamente completate grazie al sistema di acknowledgment esplicito.

5.4 MongoDB

MongoDB, o semplicemente Mongo, è un DBMS documentale della famiglia NoSQL che consente di memorizzare documenti in formato JSON. Si tratta di un progetto open source nato nel 2007 divenuto famoso in poco tempo spinto in particolar modo dalla recente popolarità della piattaforma NodeJS Mondo è oramai uno standard de facto per la memorizzazione di dati in applicativi che non necessitano di un database relazionale.

Questo rappresenta la soluzione ideale quando si cercano garanzie circa la stabilità, facilità d'uso e la ricchezza di librerie utilizzabili per interfacciarsi ad esso. È disponibile per Vert.x la libreria ufficiale "Vert.x MongoDB Client" che permette di interrogare il DBMS e ottenerne i risultati in maniera compatibile con il modello di concorrenza del framework.

5.4.1 Inquadramento all'interno del progetto

Mongo viene utilizzato per memorizzare in maniera persistente la lista delle pratiche e le informazioni ricevute dalle diverse fonti dati in maniera tale che nessun dato venga perso in caso di fallimenti. La persistenza dei dati ottenuta grazie a questo, appositamente orchestrata con il sistema di acknowledgment esplicito dei messaggi fornito da RabbitMQ, permettono di gestire le operazioni inerenti ai singoli messaggi almeno una volta. Le operazioni sono state sviluppate in maniera tale da non avere effetti negativi sullo stato del sistema qualora venissero ripetute più volte rispettando la proprietà di idempotenza.

Capitolo 6

Implementazione prototipale

In questo capitolo vengono mostrati i dettagli relativi all'implementazione incluse le scelte effettuate, come sono state risolte le problematiche individuate, il funzionamento delle singole componenti e come queste interagiscono.

Il capitolo si apre con la descrizione di alcune librerie e strumenti costruiti intorno Vert.x al fine di complementare le potenzialità offerte dal framework con le caratteristiche richieste dal sistema finale. In seguito viene esposta l'architettura interna delle componenti insieme ai dettagli implementativi più rilevanti per ognuna di queste. Viene anche mostrato il sistema di configurazione e deployment con una descrizione delle diverse configurazioni possibili e di come queste possano influenzare il funzionamento del sistema. Infine vengono descritti i test di unità e integrazione utilizzati per verificare il corretto funzionamento del sistema.

6.1 RxVertxUtils

Vert.x nasce come framework fortemente legato a NodeJS. Questi condividono il modello di gestione della concorrenza basato sull'idea che le operazioni bloccanti debbano venir risolte in maniera asincrona, cioè senza bloccare il flusso di controllo chiamante. NodeJS, così come le librerie e i framework pensati per esso, offrivano al programmatore delle interfacce callback-based con cui definire la logica di gestione dei risultati delle operazioni asincrone. In questo modello degli "handler" vengono passati come parametro, solitamente l'ultimo, alle operazioni asincrone solitamente sotto forma di

funzioni anonime (lambda). La lettura e comprensione del codice dovrebbe risultare semplificata dal momento che il risultato viene utilizzato nelle linee di codice immediatamente successive alla chiamata asincrona. Quando l'operazione è terminata l'handler viene richiamato dal flusso di controllo che gestisce l'event loop con il risultato ottenuto. Vert.x fa sua questa idea per cui è stato previsto che il programmatore richiami i metodi offerti passando come ultimo parametro una istanza di "Handler<AsyncResult<T>>". Risulta vitale l'uso delle "Lambda expressions" messe a disposizione da Java 8 anche se questo non mitiga i problemi esposti in seguito.

6.1.1 Problematiche relative allo stile di programmazione

Nel tempo questo stile di programmazione, denominato "Continuation Passing Style", ha mostrato problematiche relative alla organizzazione e quindi comprensibilità del codice. Queste problematiche a cui ci si riferisce con il nome "callback hell" includono:

- La difficoltà nel comprendere e manutenere il codice dovuto alla frammentazione degli handler che rende complesso capire come e quando la computazione viene svolta. Problematica denominata "asynchronous spaghetti", nome che richiama il problema generale dello "spaghetti code".

- Difficile organizzazione del codice dovuto all'annidamento di chiamate asincrone, cosa che mina profondamente la leggibilità del codice e rende complesse sia le operazioni di sviluppo che di manutenzione. Questa problematica è chiamata "pyramid of doom", nome dovuto all'aspetto che presenta il codice quando sono presenti più chiamate a operazioni asincrone annidate.

Mentre i linguaggi della famiglia ECMAScript si sono evoluti al fine di far fronte ai problemi appena citati introducendo astrazioni quali le Promise, Vert.x è rimasto fedele al Continuation Passing Style: attualmente le principali librerie per JavaScript e linguaggi derivati offrono al programmatore la possibilità di concatenare Promise invece di utilizzare un callback per ricevere il risultato e questa duplice modalità non è disponibile nel framework Vert.x.

6.1.2 CPS vs Promises vs Reactive Programming

Come intuibile dal nome, le Promise consistono in una promessa, un risultato sconosciuto che deve ancora essere elaborato. Queste permettono di incapsulare le operazioni asincrone permettendo di trattarne il valore risultante anche se questo non è ancora disponibile. I metodi con cui trattare le promise variano a seconda della libreria che ne fornisce l'implementazione. ECMASCript 2015 (ES6), nella specifica "Promise/A+", definisce le operazioni più comuni permettendo interoperabilità tra le diverse implementazioni. Queste operazioni sono il concatenamento (then) e la gestione degli errori (catch). La promise può avere due stati: risolta o rifiutata. Quando risolta i callback concatenati tramite il metodo then vengono richiamati con il risultato dell'operazione asincrona. Questi a loro volta possono restituire un'altra Promise oppure un valore. In quest'ultimo caso i valori vengono gestiti come Promise già risolte. Se le operazioni asincrone incontrano un errore allora la Promise viene rifiutata e l'errore può viene catturato da un callback concatenato tramite il metodo catch. Questo permette di risolvere entrambe le problematiche sopra esposte:

- La problematica dello "spaghetti code" viene risolta in quanto al programmatore risulta trasparente il modo e l'ordine con cui vengono risolte le promise: una promise può essere già risolta o in attesa che la relativa operazione asincrona completi nel momento in cui questa viene utilizzata.

- Il problema relativo alla "pyramid of doom" viene fortemente mitigato dal momento che il concatenamento dei callback rimpiazza l'annidamento tipico del Continuation Passing Style.

Mentre le Promise sono divenute lo standard nella restituzione e gestione dei risultati delle operazioni asincrone, queste risultano essere astrazioni meno generali rispetto a quanto offerto dal Reactive Programming. Nello specifico le Promise rappresentano singoli valori, hanno un comportamento prefissato uguale per tutte, le operazioni con cui manipolarle sono standardizzate, poche e limitate al funzionamento di base. Le Promise risultano quindi meno flessibili rispetto alle astrazioni offerte da librerie della famiglia ReactiveX.

Nel CPS il callback passato come parametro alle operazioni asincrone potrebbe venir richiamato più volte e con valori differenti qualora l'operazione in questione preveda che vi siano più risultati resi disponibili in tempi diversi. Questa potenza espressiva non è disponibile utilizzando Promises.

Il Reactive Programming unisce le potenzialità delle Promise aggiungendo la possibilità di gestire flussi di dati emessi in maniera asincrona anche attraverso ricche librerie che rendono disponibili al programmatore operatori in grado di risolvere le problematiche più comuni. Sotto il nome ReactiveX sono raccolti più progetti open source che rappresentano l'implementazione per diversi linguaggi di programmazione di una specifica comune per lo sviluppo di librerie a supporto di questo stile di programmazione.

6.1.3 Bridging dal sistema Callback-based a un sistema basato su RxJava 2

Al fine di evitare le problematiche relative al CPS è stata approntata una libreria, denominata "RxVertxUtils", per trasformare le chiamate alle interfacce callback-based di Vert.x nelle astrazioni tipiche del reactive Programming. Quello implementato in Vert.x rappresenta un meccanismo di basso livello catturabile completamente con le astrazioni rese disponibili dalla libreria RxJava. Nel progetto è stata utilizzata la versione 2 di questa libreria che, a differenza di RxJava 1, permette di gestire in maniera efficace le differenze tra operazioni con risultati singoli (Single) e operazioni che comportano flussi di dati (Observable).

Vert.x prevede che i callback siano una istanza di "Handler<X>". Questi rappresentano dei consumer che accettano un parametro il cui tipo è X. Nelle operazioni asincrone questo tipo è solitamente un "AsyncResult<T>" e ne rappresenta il risultato. "AsyncResult<T>" può essere utilizzato per ottenere il risultato, di tipo T, oppure il "Throwable" che rappresenta l'errore che ha portato al fallimento dell'operazione.

Le operazioni di base implementate nella libreria RxVertxUtils, a partire dalle quali è stato possibile basare l'insieme di strumenti successivamente sviluppati consistono in "vertxHandlerToResultSingle" e "vertxHandlerToSuccessSingle". Il primo è un metodo che accetta un "Consumer<Handler<AsyncResult<T>>>" e restituisce un "Single<T>". Il consumer è solitamente una funzione anonima che ha l'obiettivo di richiamare una delle operazioni offerte da Vert.x o dalle librerie annesse passando

l'handler ricevuto in parametro. Questo verrà quindi utilizzato internamente per ricevere il risultato dell'operazione asincrona e risolvere il Single¡T¿ restituito. I Single rappresentano l'equivalente in RxJava2 di una Promise. Il secondo è del tutto equivalente a quello appena esposto ma viene utilizzato quando non è necessario utilizzare il risultato ottenuto ma solo per valutare se l'operazione è stata completata con successo. Questo è utile in particolar modo per gestire quelle operazioni Vert.x che richiedono un callback "Handler<AsyncResult<Void>>" e che quindi, non potendo essere istanziati oggetti Void, restituiscono un risultato nullo in caso di successo. RxJava2 non permette che vengano utilizzati valori nulli per cui in caso di successo il "Single<Object>" restituito viene risolto con un oggetto generico.

A partire da queste operazioni base sono stati sviluppati metodi di utilità atti a semplificare alcune operazioni ricorrenti che riguardano elementi del framework Vert.x quali l'Event Bus e la gestione del ciclo di vita dei Verticle.

```
public static Single<Object> sendEventBusMessage(
                                    Vertx vertx,
                                    String channelName,
                                    Object message) {
   return RxVertxUtils.
        <Message<Object>>vertxHandlerToSuccessSingle((handler) -> {
    vertx.eventBus().send(channelName, message, handler);
   });
}
```

Listing 6.1: Esempio di uso del metodo di utilità "vertxHandlerToSuccessSingle". Il single restituito viene risolto con un successo se è stato possibile inviare il messaggio ed è stata restituita una risposta.

6.2 InitializationUtils

Nel momento in cui i Verticle vengono inseriti in una istanza di Vert.x questi devono essere eseguire alcune operazioni prima che questi possano essere considerati operativi. Per Verticle semplici queste operazioni riguardano solamente l'inizializzazione delle strutture dati ma per altri si rende neces-

sario inizializzare il sistema di memoria, ripristinare lo stato prima di un crash, verificare i canali di comunicazione, eccetera. A tal fine Vert.x mette a disposizione la classe AbstractVerticle che semplifica la gestione del ciclo di vita dei Verticle.

Al fine di poter gestire l'inizializzazione e lo spegnimento deve essere effettuato l'override dei metodi "start(Future<Void>)" e "stop(Future<Void>)". Questi accettano una future che è possibile risolvere asincronamente permettendo ai Verticle di effettuare delle operazioni asincrone prima che questi vengano definitivamente considerati in funzione.

Queste operazioni sono potenzialmente complesse, da eseguire asincronamente ma sequenzialmente e solitamente per ognuna di queste operazioni deve essere prevista una relativa azione di pulizia da eseguire nel momento in cui il Verticle viene rimosso dal sistema.

Mentre alcune operazioni legate allo spegnimento, come quelle che riguardano la pulizia delle strutture dati, potrebbero essere lasciate da svolgere al garbage collector, altre devono essere condotte obbligatoriamente. Queste includono la chiusura dei client per Mongo e RabbitMQ, l'eliminazione degli Event Bus consumer, la chiusura e pulizia di file e in generale qualsiasi di qualsiasi risorsa potenzialmente in grado di sopravvivere alla chiusura del Verticle. Va infatti ricordato che la eliminazione di un Verticle potrebbe non corrispondere all'eliminazione dell'istanza Vert.x e tantomeno alla chiusura della JVM.

Al fine di organizzare al meglio queste operazioni, rendere tracciabili gli errori e permettere al manutentore di ottenere velocemente un feedback riguardante l'avanzamento di queste operazioni è stata creata una apposita libreria "InitializationUtils".

6.2.1 InitializationContext

Nel progetto sono diverse le componenti, anche non rappresentate da Verticle, che richiedono vengano seguiti specifici passi per la loro inizializzazione ed eliminazione. Queste componenti sono rappresentate dalle classi che implementano l'interfaccia "LazyInitializable". Per queste l'inizializzazione non viene effettuata nel costruttore e nemmeno richiamando metodi di "start" o "stop". LazyInitializable infatti prevede il solo metodo "getInitializationContext()" che restituisce un "InitializationContext".

Un InitializationContext rappresenta una lista di passi da seguire per effettuare l'inizializzazione e l'eliminazione di quella componente. Gli InitializationContext sono annidabili e vanno a formare i nodi intermedi di un albero le cui foglie sono rappresentate dagli "InitializationStep". I singoli passi sono rappresentati da una operazione asincrona da effettuare in fase di inizializzazione e una da effettuare in fase di spegnimento o rollback. Gli InitializazionContext e gli InitializationStep hanno un nome che descrive rispettivamente la componente da inizializzare e l'operazione effettuata.

Tramite il sistema di annidamento degli InizializationContext è possibile esplicitare quali sono le operazioni complessive da eseguire per ogni Verticle incluse quelle che riguardano la inizializzazione e spegnimento di eventuali sotto-componenti.

6.2.2 Tracciabilità degli errori di startup e configurazione

Nel momento in cui una componente deve essere inizializzata viene creato un albero che ha come nodo padre un InitializationContext. Di questo nodo viene eseguita la lista delle operazioni da effettuare. La fase di inizializzazione termina con un successo se tutte le operazioni sono andate a buon fine. Nel caso in cui venisse rilevato un errore in una qualsiasi operazione, l'inizializzazione verrebbe arrestata e verrebbe messo in atto un meccanismo di rollback al fine di eliminare le modifiche apportate. Questo consiste nell'effettuare le operazioni di spegnimento per le sole operazioni che hanno avuto successo.

Le operazioni di spegnimento di una componente procedono similmente ma nel caso di fallimento di una operazione la procedura continua con quelle successive.

I metodi di utilità presenti nella libreria InitializationUtils semplificano la definizione delle operazioni e la loro esecuzione. Nel progetto vengono estensivamente utilizzati due metodi:

- "initialize()" prevede come input l'InitializationContext e opzionalmente flag utilizzabili per controllare la produzione della traccia visibile prodotta. Questo metodo restituisce un "Single<Object>" che viene risolto con un successo nel caso in cui l'inizializzazione sia andata a buon fine. In caso di errore viene riportata l'eccezione che ha causato il fallimento.

```
[IsolatedCognitiveManager] Initializing
|--Data structures initialization
|--[CognitiveManagerMediumImpl] Initializing
|--|--Configuration fetch
|--|--Outgoing application manager operations bridge initialization
|--|--Outgoing elaboration results bridge initialization
|--|--Incoming practice management messages bridge initialization
|--|--Incoming trauma events bridge initialization
|--|--Incoming results bridge initialization
|--|--Incoming application manager operation results bridge initialization
|--|--Event bus listener for incoming events initialization
|--|--Event bus listener for incoming practice management messages initialization
|--|--Event bus listener for incoming results initialization
|--|--Event bus listener for incoming application manager results initialization
|--|--Dynamic practice data structures initialization
|--[CognitiveManagerMediumImpl] ***Initialization completed***
|--[PracticeMetadataInMemoryStorage] Initializing
|--|--Initialization
|--[PracticeMetadataInMemoryStorage] ***Initialization completed***
|--Restored practices stream setup
|--Message streams consumers setup
[IsolatedCognitiveManager] ***Initialization completed***
```

Figura 6.1: Esempio della traccia visualizzata a seguito dell'inizializzazione di una componente.

- "shutdown()" opera nella maniera complementare a "inizialize()" eseguendo tutte le operazioni di spegnimento previste. Anche questo restituisce un "Single<Object>" risolto con una semantica uguale a quella di "initialize()". In caso di errori viene riportata solamente la prima eccezione riscontrata.

6.3 Gestione delle comunicazioni e l'Event Bus Bridge

Vert.x mette a disposizione un sistema di comunicazione basato su Event Bus. Questo permette di gestire comunicazioni publish/subscribe, punto a punto e request-response. I messaggi vengono inviati a un "address". Questo non rappresenta l'indirizzo di un Verticle ma è paragonabile al nome di una coda di messaggi. Solitamente questo nome è scelto liberamente dal programmatore. Un Verticle può registrarsi a un address esplicitando l'handler che riceverà i messaggi. Ai messaggi è possibile rispondere per

trasmettere informazioni oppure semplicemente affinché il mittente abbia una conferma del fatto che è stato ricevuto. Una delle caratteristiche del sistema di Event Bus è infatti quella di non garantire che i messaggi vengano consegnati, mettendo a disposizione un sistema best-effort e event-based piuttosto che message-based.

Questo può portare a problemi nei casi in cui si rende necessario che le comunicazioni siano persistenti. Per far fronte a questa problematica e sopperire questa mancanza è stato deciso di costruire un Event Bus Bridge basato su RabbitMQ. Esiste già una implementazione di un bridge per AMQP ma questo non garantisce una effettiva integrazione con il sistema offerto da Vert.x richiedendo venga utilizzato esplicitamente per gestire le comunicazioni e quindi apparendo tutt'altro che trasparente al programmatore. [5]

6.3.1 Bridging dei messaggi tramite RabbitMQ

RabbitMQ può essere configurato in modo tale da offre garanzie circa la persistenza dei messaggi e il fatto che questi vengano processati almeno una volta attraverso un sistema di acknowledgment e ri-accodamento implicito. In caso di comunicazione punto-a-punto gli address che rappresentano la destinazione degli eventi sono direttamente accomunabili alle singole code di messaggi previste in AMQP. Considerate le garanzie offerte da RabbitMQ, un messaggio inviato attraverso questo bridge può essere considerato come recapitato nel momento in cui viene memorizzato dal broker. Comunicazioni del tipo request-response possono essere messe in atto sfruttando un'ulteriore di coda di messaggi per gestire le risposte.

L'implementazione del bridge è data dalla classe "RabbitMQEventBusBridge" e sfrutta l'implementazione del client RabbitMQ di Vert.x. Questa a sua volta è basata sul client ufficiale di RabbitMQ per Java e permette di gestire le risposte in maniera compatibile con il modello di concorrenza del framework.

Il Verticle invia e riceve messaggi interfacciandosi con l'Event Bus offerto da Vert.x. Da questo punto di vista il programmatore non si deve curare della eventuale presenza di un bridge, che appare a lui del tutto trasparente. Il bridge si occupa di registrare un "Event Bus Consumer" per ognuno degli address utilizzati per i messaggi in uscita e ricevendo in locale i messaggi inviati dal Verticle. Questi messaggi vengono poi inviati al broker RabbitMQ

in una o più code, realizzando di fatto un'operazione di multiplexing. Allo stesso modo il bridge registra dei "RabbitMQ Consumer" per ricevere i messaggi entranti e inoltrarli su uno o più address dell'Event Bus offerto da Vert.x realizzando un'operazione di demultiplexing.

Il codice del Verticle risulta quindi immutato in presenza di un bridge e lavora in maniera indipendente da questo. Per sua natura il bridge richiede che in fase di inizializzazione dell'istanza Vert.x vengano dichiarate le code utilizzate e se queste verranno usate per inviare o ricevere messaggi. Questa operazione può quindi essere svolta in maniera differente in base alla decisione di accorpare o separare Verticle su istanze di Vert.x differenti senza aver la necessità di modificare la logica applicativa dei servizi.

6.3.2 Bridging dinamico basato sul destinatario

Vi sono dei casi in cui alcuni Verticle potrebbero venirsi a trovare in esecuzione nella stessa istanza di Vert.x pur condividendo la necessità di utilizzare il bridge al fine di comunicare con servizi esterni. I messaggi inviati direttamente sull'Event Bus raggiungono tutti i destinatari presenti all'interno della stessa istanza di Vert.x per cui si potrebbero ottenere messaggi duplicati nel caso in cui questi venissero inviati e ricevuti utilizzando anche il broker RabbitMQ. Al fine di evitare questa problematica sono stati creati due ulteriori bridge:

- Il "Local Bridge" mantiene le funzionalità di multiplexing/demultiplexing del RabbitMQ Bridge ma a differenza di questo utilizza il sistema di Event Bus nativo e non duplica i messaggi inviati a un certo "address": infatti questi verrebbero comunque ricevuti dai Verticle presenti nella stessa istanza di Vert.x.

- L'"Hybrid Bridge" orchestra il RabbitMQ e il Local Bridge eseguendo un routing dei messaggi basato sulla componente di destinazione. Se la componente in questione è in esecuzione nella stessa istanza di Vert.x allora il messaggio viene inviato attraverso il Local Bridge. Al contrario, se il Verticle destinatario è in esecuzione su una istanza esterna, il messaggio viene passato al RabbitMQ Bridge. L'Hybrid Bridge configura i canali in entrata seguendo lo stesso principio.

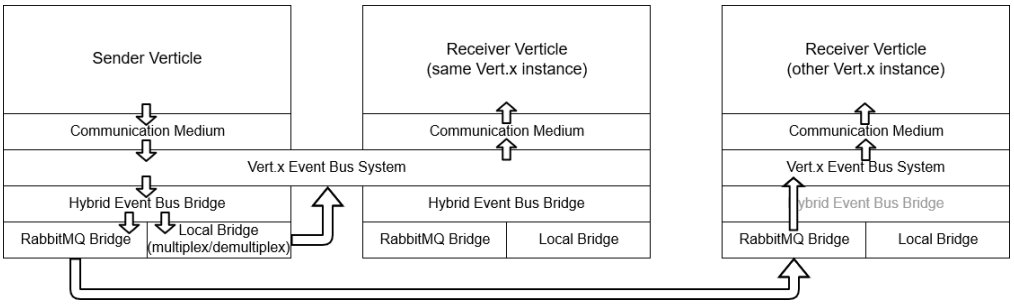

Figura 6.2: Il sistema di bridging dinamico. RabbitMQ Bridge viene utilizzato solamente per gestire le comunicazioni tra Verticle in esecuzione in istanze di Vert.x differenti.

6.3.3 Uso del sistema di acknowledgment e dei meccanismi di persistenza

Considerato che le comunicazioni sono mediate da un MOM che garantisce persistenza e ri-accodamento in caso di mancato acknowledgment, le comunicazioni sono considerate ricevute se il broker ha memorizzato il messaggio oppure, per comunicazioni non mediate dal bridge, se sono state completate tutte le operazioni per la gestione del messaggio. Se un Verticle incontra una eccezione nella gestione di un messaggio allora viene risposto a questo con un errore. In questo caso il bridge invia un "Nack" al broker indicando che il messaggio non è stato gestito e deve essere ri-accodato. Nel caso di comunicazioni locali il Verticle mittente riceve l'errore che può gestire in diverse maniere:

- Ritrasmettere l'errore nella catena di messaggi. Solitamente l'invio di un messaggio è parte delle operazioni da effettuare nella gestione di un messaggio precedentemente ricevuto. Se l'invio fallisce allora è necessario agire come per una qualsiasi operazione fallita e comunicare il fallimento nella gestione del messaggio originale. Questa rappresenta la gestione di base degli errori ed è facilitata dal pervasivo uso delle astrazioni offerte da RxJava.

- Ignorarlo. Alcuni errori sono semplicemente ignorabili nel caso di operazioni best-effort.

- Non gestirlo e andare in crash. In quest'ultimo caso il crash è solo temporaneo e viene recuperato dal demone Docker che riavvia in automatico l'istanza del container su cui era in esecuzione il Verticle. La speranza è che le condizioni dell'ambiente di esecuzione siano tali che, al momento della re-inizializzazione, l'errore non si verifichi nuovamente. In caso di errori di programmazione questo si traduce in un ciclo che è possibile interrompere utilizzando un Circuit Breaker.

6.4 Le componenti del sistema

Di seguito vengono descritte le componenti del sistema, ne viene identificato il ruolo e il funzionamento con particolare riferimento alla forma e modo con cui avvengono le comunicazioni tra di esse e tra queste e il sistema esistente. Le componenti presentano una struttura interna comune con poche differenze riconducibili al loro ruolo e quindi alla diversa logica applicativa.

6.4.1 Aspetti comuni

Le componenti hanno una struttura comune così sintetizzabile:

- Ogni componente prevede che la logica venga implementata nel relativo Verticle. Questi sono una implementazione della classe "LazyInitializableVerticle" che gestisce l'inizializzazione del Verticle a partire da un InitializationContext dato.

- Tutte le comunicazioni sono gestite da un Communication Medium che si occupa di serializzare, deserializzare e trasformare in un oggetto del modello i messaggi da inviare e quelli ricevuti. I medium hanno una implementazione diversa a seconda del Verticle a cui fanno riferimento ma hanno una struttura interna comune. Il Verticle può ricevere i messaggi in entrata registrandosi ai diversi "Observable<TTMessage<X>>" messi a disposizione dal Medium. Un TTMessage riporta il contenuto del messaggio già deserializzato ed eventualmente trasformato in un oggetto del modello. Oltre a questo TTMessage permette di indicare il successo o il fallimento nella gestione del messaggio.

- Eventuali bridge vengono inizializzati prima dei Verticle e sono configurati a livello di istanza Vert.x. Queste operazioni sono solitamente eseguite negli entry point.

- I Verticle che lo prevedono eseguono le operazioni di inizializzazione del motore di memorizzazione persistente dei dati e immediatamente successivamente ricaricano l'eventuale stato pregresso. Questa operazione viene effettuata prima che venga inizializzato il Communication Medium in modo da poter ripristinare lo stato interno prima di cominciare a gestire le comunicazioni.

- I Verticle che incapsulano la logica sono tutti rappresentati da classi astratte che prevedono vengano implementati i metodi che forniscono l'istanza del Communication Medium, dello Storage Manager, strategie legate funzionalità specifiche che riguardano il Verticle e infine altri elementi di configurazione.

- I messaggi scambiati sono nella forma di Struct Prolog. Questa scelta è stata dettata nella facilità nel gestire la forma di rappresentazione della conoscenza già adottata dal PMDA.

Gli elementi comuni del modello, in particolar modo i messaggi, sono racchiusi in un package "model". Ogni singola componente e le sue sottoparti, comprese le classi che costituiscono il modello interno, lo Storage Manager e il Communication Medium, sono contenute in package separati.

Considerata questa struttura comune è possibile focalizzarsi sulle differenze inerenti alla logica applicativa, implementata negli appositi Verticle che portano il nome delle componenti.

6.4.2 TT Cognitive

La componente TT Cognitive rappresenta il punto di ingresso e uscita dei dati. Questa si interfaccia direttamente con il sistema di backend esistente e deve essere mandata in esecuzione nella stessa istanza di Vert.x del Verticle TT Service.

Si occupa di ricevere dal sistema esistente i dati relativi al trauma in corso e di inoltrarli al Cognitive Manager. I risultati dell'elaborazione effettuata dagli algoritmi predittivi sono inviati al TTService che deve inoltrarli

ai PMDA. È presente una sola istanza del TT Cognitive nel sistema. I tracciati previsti sono:

- Un canale dal nome prefissato da cui ricevere informazioni riguardanti le operazioni di gestione (apertura e chiusura delle pratiche) e gli eventi per tutte le pratiche. Queste comunicazioni hanno rispettivamente formato: "practice(Practiceid, Operation)" e "evt(Event, Practiceid)" dover Practiceid è una string, Operation può essere "create" o "delete" e Event è un qualsiasi evento proveniente dal PMDA, dai sensori, eccetera. item Un canale dal nome prefissato su cui inviare i risultati delle elaborazioni. I risultati hanno formato "result(Result, Practiceid)".

6.4.3 Cognitive Manager

Il Cognitive Manager si occupa di ricevere i messaggi appena descritti, memorizzare in maniera persistente la lista delle pratiche, gestire l'inizializzazione e spegnimento dei Practice Manager relativi alle diverse pratiche, inoltrare al TT Cognitive i risultati delle elaborazioni provenienti dal Practice Manager. È presente una sola istanza del Cognitive Manager nel sistema. Questi i canali di comunicazione gestiti:

- Un canale su cui ricevere le operazioni di gestione delle pratiche
- Un canale da nome prefissato su cui ricevere gli eventi per tutte le pratiche
- Un canale dal nome prefissato su cui ricevere i risultati delle elaborazioni. I risultati provengono dai diversi Practice Manager.
- Diversi canali su cui inviare ai diversi Practice Manager gli eventi relativi alle pratiche da loro gestite. Il nome dei diversi canali è costruito secondo un pattern prefissato.

6.4.4 Practice Manager

Il Practice Manager si occupa di gestire gli eventi relativi a una sola pratica ricevendoli e memorizzandoli in maniera persistente. Gli eventi memorizzati

formano una teoria Prolog a cui gli algoritmi predittivi possono accedere inviando interrogazioni al Practice Manager.

Sono presenti diverse istanze del Practice Manager nel sistema, una per ogni pratica aperta. Quando una pratica viene chiusa gli eventi memorizzati nella memoria persistente vengono cancellati e l'istanza viene terminata. Di seguito i canali previsti:

- Un canale su cui ricevere gli eventi provenienti dal Cognitive Manager.
- Un canale su cui ricevere i risultati delle elaborazioni. Il nome del canale è costruito secondo un pattern prefissato.
- Un canale su cui ricevere le interrogazioni riguardanti gli eventi memorizzati. Il nome del canale è costruito secondo un pattern prefissato.
- Diversi canali su cui inviare i risultati delle interrogazioni, uno per ogni Algorithm Manager. Il nome dei canali è costruito secondo un pattern prefissato.

6.4.5 Application Manager

L'Application Manager gestisce gli applicativi in esecuzione nel sistema. Questi corrispondono a diversi container Docker al cui interno è in esecuzione una JVM. In questa viene mandata in esecuzione una istanza di Vert.x che a sua volta contiene il Verticle in cui è implementata la componente. Per i differenti applicativi è previsto un identificatore, il nome dell'immagine Docker da utilizzare, il comando per lanciare l'eseguibile, l'identificativo nell'applicativo padre. La lista degli applicativi è salvata in memoria persistente.

L'Application Manager è sempre in esecuzione nel sistema e deve poter amministrare il demone Docker. Per questo il Verticle che lo implementa è in esecuzione, insieme al TT Cognitive, nel backend esistente e non all'interno di un container. Le componenti del sistema possono interagire con l'Application Manager al fine di lanciare o eliminare gli applicativi figli. Quando un applicativo viene eliminato anche i relativi figli subiscono la stessa sorte.

I container sono configurati in maniera tale che Docker li riavvii nel caso questi incontrino un errore o venga riavviato il sistema ospite. Tra questi figurano anche l'istanza del broker RabbitMQ e del DBMS Mongo. Tutti

i container sono collegati in una rete Docker per cui possono comunicare tra loro. Ogni elemento di Docker relativo al sistema quali i container, le immagini e la rete sono etichettati con un apposito "label". I "label" sono un sistema previsto da Docker che può essere utilizzato per facilitare l'amministrazione del sistema semplificando le operazioni di ricerca.

I canali previsti sono:

- Un canale dal nome prefissato su cui ricevere le operazioni da eseguire (applicazioni da lanciare, rimuovere, ottenimento e impostazione della lista delle applicazioni figlie del richiedente)
- Diversi canali su cui inviare i risultati delle operazioni, uno per ogni richiedente. Il nome dei canali è costruito secondo un pattern prefissato.

6.5 Libreria di supporto per lo sviluppo di algoritmi

Al fine di poter gestire agevolmente il ciclo di vita di un algoritmo è stata creata una libreria di supporto. Gli algoritmi devono interfacciarsi al sistema esistente per mezzo dei suddetti canali di comunicazione e, al fine di poter elaborare i dati relativi alla pratica, devono poter eseguire interrogazioni riguardanti la base di conoscenza accumulata dal Practice Manager. Al fine di semplificare la gestione e sollevare l'implementatore dell'algoritmo da aspetti riguardanti l'integrazione con il sistema esistente è stata creata la classe astratta "AbstractAlgorithm<T>". Questa consiste in un Verticle che include la logica di integrazione con il sistema esistente e lascia allo sviluppatore il compito di implementare l'algoritmo definendo i metodi "setupQueries" e "executeAlgorithm".

- Nel metodo "setupQueries(Observer<Optional<T>>)" devono essere impostate le interrogazioni da inviare al Practice Manager. Per ogni interrogazione, registrata richiamando il metodo della classe base "setupQuery(String)" viene restituito un Observable. Questo rappresenta un flusso di risultati per quella specifica interrogazione. La libreria si occuperà di ripetere periodicamente, con un intervallo deciso dallo sviluppatore, le interrogazioni e renderne disponibili i risultati. I

diversi risultati possono essere processati e infine uniti per formare il modello dei dati richiesto dall'algoritmo. Il modello deve essere inviato all'Observer passato come parametro al metodo. Nel caso non siano ancora disponibili i dati richiesti per l'esecuzione, l'Observer può essere richiamato con un Optional vuoto.

- Nel metodo "executeAlgoritm(T model, SingleObserver<String>)" deve essere implementata la logica dell'algoritmo. Questo può includere l'esecuzione di applicativi esterni. Nel momento in cui il risultato risulta disponibile, il SingleObserver ricevuto come parametro deve essere richiamato con una rappresentazione Prolog di questo. Sarà la libreria a occuparsi di trasmetterlo al sistema. Ovviamente il formato del risultato deve essere concordato in maniera tale da essere utilizzabile all'interno del PMDA.

6.6 Integrazione con il sistema esistente

Al fine di ottenere una effettiva integrazione devono essere considerati due aspetti: l'integrazione con il software presente nel sistema di backend esistente e la capacità di eseguire nell'ambiente di esecuzione previsto.

Per quanto riguarda il primo aspetto, l'integrazione con il TT Service presente è svolta a livello di framework Vert.x. TT Cognitive rappresenta la componente in cui avviene l'integrazione. Questa e il TTService sono Verticle che devono coesistere nella stessa istanza di Vert.x. Al fine di ricevere gli eventi necessari per il funzionamento degli algoritmi, il TTService esistente deve inoltrare su un apposito address dell'Event Bus gli eventi relativi alla apertura e chiusura di una pratica, i dati provenienti dai sensori, alle azioni registrate dal personale del Trauma Team ed eventuali informazioni provenienti da altre sorgenti (cartelle cliniche, informazioni anagrafiche, ...). Allo stesso modo i risultati delle elaborazioni verranno resi disponibili dal TT Cognitive su un apposito address. Il TT Service deve raccogliere questi dati e inviarli al PMDA. Come precedentemente accennato, nella stessa istanza Vert.x deve essere mandato in esecuzione anche l'Application Manager. L'integrazione è facilitata dall'uso delle utility presenti nella classe "TTEntryPoint". Il metodo "getTTInitializationContext()" di questa ha il preciso scopo di facilitare il deployment dei due Verticle in questione. L'InitializationContext restituito include i passi necessari per il deployment e

l'esecuzione di operazioni preliminari quali la messa in esecuzione del broker RabbitMQ e del DBMS Mongo.

A tal fine è necessario soffermarsi sul secondo aspetto relativo all'integrazione con il sistema esistente: quello relativo all'ambiente di esecuzione richiesto. L'ambiente richiesto deve avere a disposizione:

- Un Java Runtime Environment.
- Un demone Docker. Non è obbligatorio che l'ambiente su cui esegue la JVM sia lo stesso su cui esegue Docker in quanto questo può essere in esecuzione su un altro ambiente, come un server in remoto o una virtual machine (situazione comune nel caso venga utilizzato "Docker for Windows" o "Docker for Mac"). L'importante è che il demone sia accessibile dall'Application Manager.

6.7 Configurabilità e deployment

Il sistema di configurazione permette di modificare il comportamento delle componenti in maniera da garantire la corretta integrazione nell'ambiente di esecuzione. Grazie a questo è possibile modificare parametri quali il nome dei canali, delle "Collection" di Mongo, il "label" applicato agli oggetti creati in Docker, il nome dei container, eccetera. Di seguito vengono esposte le possibili configurazioni e come queste vengono utilizzate nelle fasi di deployment e esecuzione.

6.7.1 Configurazione

Compatibilmente con il sistema adottato da Vert.x, la configurazione è rappresentata da un JsonObject. Questo deve essere passato al Verticle al momento in cui viene eseguito il suo deployment all'interno di una istanza di Vert.x.

La configurazione è in parte identica per tutti e in parte variabile a seconda della componente mandata in esecuzione. La parte fissa è costituita da:

- Nomi dei canali da utilizzare o pattern per crearli a partire dall'identificatore della pratica o dell'algoritmo.

- Identificatori degli applicativi o pattern per crearli a partire dall'identificatore della pratica o dell'algoritmo.
- Configurazione del client RabbitMQ.
- Configurazione del client Mongo.
- Configurazione del client Docker.
- Nome e versione delle immagini da utilizzare per quanto riguarda le istanze di RabbitMQ e Mongo.

La parte variabile è costituita da:

- Identificatore della pratica. Utilizzato dalle componenti Practice Manager, Algorithm Manager e dall'Abstract Algorithm.
- Identificatore dell'algoritmo. Utilizzato dall'Algorithm Manager e dall'Abstract Algorithm.

6.7.2 Deployment

Le operazioni di deployment devono essere effettuate prima della messa in esecuzione del sistema e hanno come scopo quello di configurare l'ambiente Docker.

Le operazioni effettuate sono:

- La creazione delle immagini Docker degli algoritmi. Queste immagini verranno utilizzate per creare i container in cui eseguiranno gli algoritmi.
- La creazione dell'immagine Docker in cui è memorizzato il Jar del sistema. Questa immagine viene utilizzata per creare istanze del Cognitive Manager, Practice Manager e dell'Algorithm Manager.
- L'ottenimento delle immagini di RabbitMQ e Mongo dal repository ufficiale.
- La creazione della rete Docker utilizzata delle componenti per comunicare.

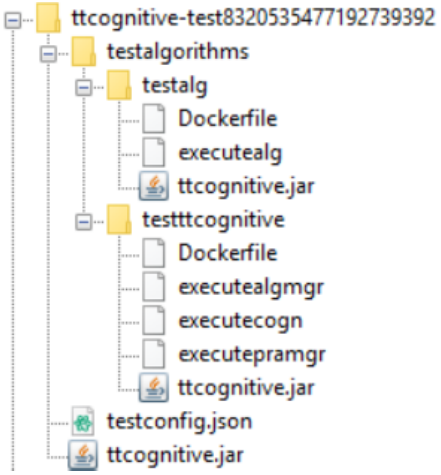

Figura 6.3: La struttura richiesta per il deployment. L'eseguibile da cui lanciare il Deployer è contenuto nella root (ttcognitive.jar) insieme al file di configurazione.

Queste operazioni, svolte dal Deployer, sono da eseguire ogni qual volta viene rilasciata una nuova versione del sistema o degli algoritmi in modo da aggiornare le immagini memorizzate.

Per le immagini da creare devono essere comprese tutte le risorse necessarie per garantire il successo del sistema di assemblaggio. Il necessario deve essere posto in una cartella che porta il nome dell'algoritmo (una cartella dal nome definito nella configurazione per quanto riguarda l'applicativo di sistema). In ognuna di queste cartelle deve essere quindi presente un apposito Dockerfile che descrive la struttura dell'immagine. Le cartelle si devono trovare all'interno del percorso definito nella configurazione passata al Deployer. Una volta completata la loro costruzione, le immagini vengono etichettate con il loro nome e saranno utilizzabili dall'Application Manager per lanciare istanze delle componenti richieste.

Le istanze di RabbitMQ e Mongo non vengono lanciate in questa fase ma vengono create nel momento in cui viene mandato in esecuzione il sistema. Ne segue che la fase di deploy non modifica il sistema ospitante e tantomeno attiva alcuna componente.

6.7.3 Entry points

Nel sistema è possibile identificare i seguenti entry points, contenuti nel package it.unibo.disi.pslab.traumatracker.cognitive.entrypoint:

- Deployer. Questo entry point accetta un solo parametro che consiste nella configurazione da utilizzare. Il parametro deve essere un oggetto JSON valido. In caso contrario questo viene interpretato come il percorso del file contenente la configurazione.

- Algorithm. Questo entry point viene utilizzato per lanciare una istanza dell'algoritmo e accetta un solo parametro che consiste nella configurazione da utilizzare. L'algoritmo deve essere implementato nella classe it.unibo.disi.pslab.traumatracker.cognitive.algorithmlib.AlgorithmImpl oppure in qualsiasi altra classe (con una minima modifica dell'entry point).

- EntryPoint. Questo entry point viene utilizzato dall'Application Manager per lanciare le componenti previste dal sistema e accetta due parametri:

  – Il nome del Verticle da inizializzare. I valori validi sono "COGNITIVE_MANAGER", "PRACTICE_MANAGER", "ALGORITHM_MANAGER". In alternativa questo parametro può avere valore "deploy" oppure "algorithm". In questo caso vengono richiamati i precedenti due entry point.

  – La configurazione, in formato JSON, da passare al relativo Verticle (o agli entry point precedenti).

6.8 Test

Di seguito viene esposto il metodo adottato per condurre i test e una descrizione di questi. I test si dividono in test di unità e test di integrazione. Per quanto riguarda i test di unità sono state testate le singole componenti configurate in modo tale da utilizzare il sistema di Event Bus senza bridge e senza memorizzare in memoria persistente i dati. Al contrario i test di integrazione si svolgono in un ambiente creato appositamente in maniera

automatica dove vengono testate tutte le funzionalità del sistema. L'unico requisito per condurre i test è la disponibilità di una istanza di Docker.

I test sono stati scritti utilizzando il framework Vert.x Unit che a sua volta si interfaccia con JUnit.

6.8.1 Configurazione degli elementi e dell'ambiente di test

I test utilizzano una configurazione simile a quella utilizzata nel sistema finale. Risultano differenti:

- L'indirizzo del demone Docker, del broker RabbitMQ e del DBMS Mongo.
- Il nome delle immagini, dei container e della rete Docker e la "label" con cui questi sono identificati, compresi il nome dei container delle istanze di RabbitMQ e Mongo.
- Il nome dei canali di comunicazione.
- I nomi delle "Collection" in cui vengono memorizzati i dati su Mongo.

Al fine di eseguire correttamente i test è necessario modificare in maniera opportuna almeno le configurazioni degli elementi al primo punto. Questo è fattibile modificando le stringhe presenti nella classe "TestConfigurations". I test prevedono dei controlli riguardanti queste configurazioni per cui JUnit potrebbe riportare test ignorati nel caso in cui l'ambiente di esecuzione non risultasse essere quello atteso. I test di integrazione devono essere lanciati utilizzando l'apposito task "test" di Gradle. Questo vede come prerequisito il task "shadowJar" in quanto il Jar prodotto da questo viene utilizzato nei test di integrazione: utilizzare l'esecutore integrato nell'IDE porta a un fallimento nel controllo delle precondizioni.

I singoli test prevedono delle operazioni di pulizia che eliminano le tracce della loro esecuzione dall'istanza di Docker. Il sistema in cui viene eseguito il test non viene modificato. I file creati per l'esecuzione dei test di integrazione vengono memorizzati in una cartella temporanea.

6.8.2 Unit test

Ogni componente da testare viene mandata in esecuzione su di una istanza di Vert.x. Insieme a questa, nella stessa istanza, si trovano Verticle che rappresentano le altre componenti del sistema. Questi sono istanze delle classi "TestReceiveVerticle" e "TestSendVerticle" e vengono utilizzate per inviare e ricevere eventi. I test iniziano inviando messaggi alla componente sotto test e i messaggi ricevuti in risposta da questa vengono utilizzati per verificare che il comportamento sia quello atteso.

Questi test sono contenuti nelle classi "AlgorithmManagerUnitTest", "AlgorithmUnitTest", "CognitiveManagerUnitTest", "PracticeManagerUnitTest", "TTCognitiveUnitTest".

Oltre ai test di unità riguardanti le componenti del sistema è stato costruito un test della libreria con cui l'Application Manager si interfaccia con il client Docker. Quest'ultimo infatti è un client generico e le operazioni messe a disposizione sono bloccanti. La libreria in questione permette di utilizzare questo client in maniera compatibile con il modello di concorrenza di Vert.x incapsulando le operazioni bloccanti e restituendo i risultati sotto forma di Single. Il test è contenuto nella classe "DockerUtilsTest".

6.8.3 Integration test

I test di integrazione prevedono operazioni preliminari atte a creare un ambiente di esecuzione identico a quello previsto nel sistema finale, con l'unico requisito che una istanza di Docker sia disponibile.

Il test di integrazione "CompleteIntegrationTest" prevede i seguenti passi:

1. Verifica del funzionamento del sistema di deployment. Questo viene eseguito al fine di creare le immagini e la rete Docker richiesti in fase di esecuzione. Viene anche eseguita la compilazione dell'immagine relativa a un algoritmo di test utilizzando il Jar prodotto dal task "shadowJar" di Gradle. Nel progetto deve essere pertanto implementato l'algoritmo di test. Lo stesso Jar viene utilizzato per creare l'immagine utilizzata per l'lanciare le istanze delle componenti previste dal sistema. I file utilizzati per costruire le immagini vengono creati in una cartella temporanea.

2. Verifica del funzionamento dell'entry point TTEntryPoint. Questo è l'entry point utilizzato per mandare in esecuzione le componenti "TT Cognitive" e l'"Application Manager".

3. Il sistema manda autonomamente in esecuzione il Cognitive Manager sotto forma di container Docker.

4. Viene mandato in esecuzione un "TestSendVerticle" e un "TestReceiveVerticle" nella stessa istanza di Vert.x del "TT Cognitive". Questi Verticle rappresentano il sistema esistente (TT Service).

5. Il test vero e proprio inizia. Attraverso i Verticle di test vengono inviati al "TT Cognitive" i messaggi che indicano l'apertura di una pratica e gli eventi previsti per il funzionamento dell'algoritmo di test.

6. Vengono ricevuti i risultati dell'algoritmo di test.

7. Viene eliminata la pratica.

8. Vengono inviati ulteriori eventi relativi alla stessa pratica. Ci si attende che il sistema ignori questi messaggi. Non devono essere ricevuti ulteriori risultati dall'algoritmo in quanto l'istanza di questo deve essere stata eliminata.

9. Viene verificato che non vi siano altri container in esecuzione oltre a quello del "Cognitive Manager", l'istanza di RabbitMQ e quella di Mongo.

10. A prescindere dal risultato viene eseguita la pulizia del sistema, eliminando tutte le immagini, container e la rete Docker creati in fase di avvio del test.

Il "Complete Integration Test" può anche essere lanciato nel sistema finale al fine di verificare che la configurazione dell'ambiente sia quella corretta.

Capitolo 7

Validazione ed evoluzione

In questo capitolo viene fornita una valutazione circa il prototipo sviluppato in termini di aderenza ai requisiti, performance, scalabilità, manutenibilità ed evolvibilità. A tal fine sono ripercorsi i requisiti e per ognuno di questi è descritto il grado di soddisfacimento raggiunto e gli eventuali limiti della soluzione proposta.

Vengono infine elencati alcuni spunti per l'evoluzione della componente sviluppata.

7.1 Validazione

Di seguito sono elencati i requisiti individuati per il Cognitive Service e una descrizione di come e a quale livello questi sono stati soddisfatti.

- Integrazione con il sistema di backend esistente.

  La componente creata, essendo stata sviluppata utilizzando lo stesso framework di quella esistente e con lo stesso linguaggio di programmazione, si integra con il TT Service andando a coesistere nella stessa istanza di Vert.x. L'unico requisito tecnico aggiuntivo richiesto per l'esecuzione del Cognitive Service è quello della disponibilità di una istanza di Docker nel sistema finale.

- Mantenere traccia di tutti gli eventi.

  Tutti gli eventi e le informazioni utili sono memorizzati, nessuno escluso. La base di conoscenza per ogni trauma è mantenuta separata da

quella degli altri, cresce durante il ciclo di vita della pratica e viene eliminata nel momento in cui questa viene chiusa.

- Permettere agli algoritmi predittivi di utilizzare tali dati.

  Gli algoritmi possono accedere alle informazioni riguardanti la pratica interrogando la base di conoscenza mantenuta dall'istanza del Practice Manager adibita a quella pratica.

- Prevedere diverse fonti dati. Sviluppare adeguati meccanismi con cui ricevere informazioni da esse.

  La componente sviluppata non è dipendente da alcuna sorgente dati. Nuove sorgenti possono essere integrate in maniera del tutto indipendente inviando, attraverso il TT Service esistente o attraverso un altro Verticle in esecuzione nella stessa istanza di Vert.x, le informazioni e gli eventi provenienti da queste all'"address" predefinito.

- Gestire più pazienti.

  Il Cognitive Service può gestire più pratiche.

- Permettere l'esecuzione di più algoritmi per paziente.

  Il Cognitive Service può gestire più algoritmi per pratica.

- Supportare diverse tecnologie per gli algoritmi predittivi.

  Grazie alla decisione di utilizzare Docker gli algoritmi possono essere sviluppati con qualsiasi tecnologia per ambienti Linux. Il Cognitive Service non supporta attualmente tecnologie di altri ambienti e non è possibile accedere ad hardware dedicato in grado di accelerare l'esecuzione delle elaborazioni.

- Restituire i risultati in una forma compatibile con il sistema di rappresentazione della conoscenza adottato dagli agenti.

  Grazie alla scelta di adottare un sistema Prolog-based per la rappresentazione della conoscenza, forma compatibile con quella prevista in Jason, è possibile integrare i risultati ottenuti dagli algoritmi nella Belief-base del "Warning Generator Agent".

7.2 Evoluzione

A partire dai requisiti appena esposti e dalle limitazioni evidenziate è possibile individuare alcuni spunti circa l'evoluzione di questa componente.

Pulizia e filtraggio della base di conoscenza Allo stato attuale il Cognitive Service mantiene una copia di tutti gli eventi registrati per una pratica. Queste informazioni non vengono in alcun modo selezionate o eliminate andando quindi a creare una base di conoscenza in continua espansione. Questa copia viene eliminata nel momento in cui viene chiusa la pratica.

Un passo successivo potrebbe essere quello di, considerate le esigenze degli algoritmi, progettare un meccanismo con cui filtrare i dati inutilizzati o obsoleti in modo da mantenere contenuta la dimensione di questa copia. Queste operazioni dovrebbero essere implementate in maniera tale da non richiedere una loro reimplementazione ogni qual volta viene aggiunto, modificato o rimosso un algoritmo.

Attualmente gli algoritmi eseguono le loro elaborazioni con una frequenza fissata. Sarebbe possibile, nel contesto di questa espansione, approntare un metodo con cui segnalare la necessità di ricalcolare il risultato di un certo algoritmo nel momento in cui viene registrata un'informazione da esso considerata.

Motore Prolog e scalabilità Il motore attualmente adottato per mantenere e interrogare la base di conoscenza è "tuProlog". Sarebbe possibile introdurre librerie Prolog di utilità o addirittura permettere di definire codice esterno (ad esempio scritto in Java) richiamabile durante le interrogazioni. Queste estensioni dovrebbero avere l'obiettivo di semplificare la definizione delle interrogazioni e migliorare le prestazioni.

Le interrogazioni attualmente vengono eseguite su un'unica copia mantenuta da una istanza (per pratica) del Practice Manager. Un'altra estensione utile al fine di migliorare le prestazioni del sistema consisterebbe nello sviluppare un meccanismo di replica e bilanciamento del carico.

Sistema di containerizzazione Il requisito relativo alla possibilità di supportare qualsiasi tecnologia richiesta per l'esecuzione di algoritmi è in realtà stato solo parzialmente raggiunto: nelle più recenti versioni di Doc-

ker è possibile creare ed eseguire container Windows. Nella versione attuale del Cognitive Service non è possibile gestire questa tipologia di container. Va considerato che questa estensione potrebbe non essere necessaria in quanto quasi tutti gli strumenti utilizzati in ambito Machine Learning sono disponibili per ambienti Linux.

Oltre a questo sarebbe possibile, con minime modifiche e sfruttando la modalità Swarm integrata in Docker, creare un cluster in grado di eseguire i container richiesti al fine di ottenere prestazioni migliori e permettere forme di gestione dei fallimenti.

Attualmente non è prevista la possibilità di sfruttare eventuale hardware dedicato collegato al sistema ospitante al fine di accelerare l'elaborazione dei dati. Questa problematica potrebbe essere risolta con alcune modifiche all'Application Manager.

Memorizzazione dei dati per algoritmo Al fine di implementare forme di apprendimento per rinforzo sarebbe possibile espandere il Cognitive Service permettendo agli algoritmi di memorizzare informazioni in grado di sopravvivere alla chiusura di una pratica.

Questa estensione non è necessaria in assenza di apprendimento con rinforzo ma potrebbe esserlo nel caso in cui il Cognitive Service venisse utilizzato anche per scopi differenti da quello di gestire l'esecuzione di algoritmi predittivi.

Conclusioni

Il Personal Medical Digital Assistant sviluppato nell'ambito del progetto Trauma Tracker è in grado di supportare i medici del Trauma Team fornendo funzionalità che vanno ben oltre al semplice tracciamento degli eventi.

Un assistente in grado di percepire le informazioni riguardanti il paziente e le azioni effettuate durante il trattamento del trauma sarebbe in grado di richiamare, quando necessario, l'attenzione del Trauma Leader attraverso la visualizzazione di avvisi e suggerimenti. La strategia con cui generare tali notifiche deve essere progettata tenendo a mente che le informazioni riportate ai medici devono risultare utili al fine di ridurre i tempi di gestione delle pratiche e migliorare le probabilità di successo.

Allo stato attuale la generazione di avvisi e suggerimenti è affidata a un sistema che si basa su di un insieme di regole definite dal Trauma Team e questo comporta una capacità espressiva limitata. Come alternativa a questo sono state individuate le tecniche della famiglia del Machine Learning e del Cognitive Computing. Queste permetterebbero di classificare e predire la condizione del paziente anche grazie all'analisi dei dati storici mantenuti negli archivi digitali del Trauma Center con il fine di generare avvisi di maggiore utilità.

Al fine di integrare queste tecnologie con il Personal Assistant attualmente esistente è stato necessario sviluppare un Cognitive Service in grado di supportare elaborazioni di vario genere. L'obiettivo di questo è quello di permettere l'esecuzione di algoritmi basati sui dati mantenuti nella base di conoscenza del PMDA andando poi a integrare in questa i risultati ottenuti. Quello che si ottiene è una espansione delle capacità di ragionamento dell'assistente. Tale piattaforma, attualmente allo stato prototipale, potrà anche essere utilizzata per supportare elaborazioni di natura differente.

A parte la verifica relativa all'attinenza con i requisiti, non è stato possibile validare completamente l'architettura del Cognitive Service. A tal fine

ci si attende che in futuro, nell'ambito di successivi lavori inerenti all'implementazione e quindi integrazione di algoritmi predittivi di varia natura e basati su tecnologie differenti, si possa confermare la bontà dell'architettura proposta.

Ringraziamenti

Ringrazio il relatore, Prof. Alessandro Ricci, e la correlatrice, Dott.ssa Sara Montagna, per la fiducia mostrata verso di me nell'affidarmi l'esplorazione di un ambito di particolare interesse, innovatività e complessità e il progetto che ne è scaturito.

Un ringraziamento particolare va alla mia famiglia, che in questi cinque anni di studi universitari ha saputo supportarmi e sopportarmi nei momenti di difficoltà e in particolar modo durante il periodo in cui ho lavorato a questa tesi.

Ringrazio i miei amici per essermi stati vicino anche in quei, purtroppo frequenti, momenti in cui risultavo un po' assente. Senza di loro questi anni non sarebbero stati neanche lontanamente così felici.

Bibliografia

[1] Heart disease and stroke statistics—2016 update: A report from the american heart association. *Circulation*, 2016.

[2] M. Bratman. Intention, plans, and practical reason. 1987.

[3] Y. Chen, J. E. Argentinis, and G. Weber. Ibm watson: how cognitive computing can be applied to big data challenges in life sciences research. *Clinical therapeutics*, 38(4):688–701, 2016.

[4] Direzione generale della digitalizzazione, del sistema informativo sanitario e della statistica. Relazione sullo stato sanitario del Paese 2012-2013. *Ministero della Salute*, pages 138–143, 2014.

[5] Eclipse Foundation. Vert.x amqp bridge. `http://vertx.io/docs/vertx-amqp-bridge/java/`.

[6] Eclipse Foundation. Vert.x core manual. `http://vertx.io/docs/vertx-core/js/`.

[7] European Parliament, Council of the European Union. Directive 2011/24/eu of the european parliament and of the council of 9 march 2011 on the application of patients' rights in cross-border healthcareg, 2011. `http://data.europa.eu/eli/dir/2011/24/oj`.

[8] R. Farrell, J. Lenchner, J. Kephart, A. Webb, M. Muller, T. Erickson, D. Melville, R. Bellamy, D. Gruen, J. Connell, et al. Symbiotic cognitive computing. *AI Magazine*, 37(3), 2016.

[9] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. Murdock, E. Hyberg, J. Prager, et al. The ai behind watson—the technical article. *The AI Magazine*, 2010.

[10] M. Gurman. Apple is working on a dedicated chip to power ai on devices, 2017. https://www.bloomberg.com/news/articles/2017-05-26/apple-said-to-plan-dedicated-chip-to-power-ai-on-devices.

[11] R. High. The era of cognitive systems: An inside look at ibm watson and how it works. *IBM Corporation, Redbooks*, 2012.

[12] IBM. Wellpoint and ibm announce agreement to put watson to work in health care, 2011. https://www-03.ibm.com/press/us/en/pressrelease/35402.wss.

[13] IBM. Memorial sloan-kettering cancer center: Ibm watson helps fight cancer with evidence-based diagnosis and treatment suggestions, 2013. https://www-935.ibm.com/services/multimedia/MSK_Case_Study_IMC14794.pdf.

[14] IBM. Wellpoint, inc.: Ibm watson enables more effective healthcare preapproval decisions using evidence-based learning, 2013. https://www-935.ibm.com/services/multimedia/WellPoint_Case_Study_IMC14792.pdf.

[15] IBM. Watson discovery visual insights, 2017. https://console.bluemix.net/docs/services/discovery/visual-insights.html.

[16] IBM. About discovery, 2018. https://console.bluemix.net/docs/services/discovery/index.html#about.

[17] G. B. Jim Spohrer. Cognition as a Service: An Industry Perspective. *AAAI*, pages 71–86, 2015.

[18] J. Kelly. Computing, cognition and the future of knowing. *Whitepaper, IBM Reseach*, 2015.

[19] J. C. R. Licklider. Man-computer symbiosis. *IRE Transactions on Human Factors in Electronics*, HFE-1(1):4–11, March 1960.

[20] P. Maes. Agents that reduce work and information overload. In *Readings in Human–Computer Interaction*, pages 811–821. Elsevier, 1995.

[21] K. Ng, S. R. Steinhubl, C. deFilippi, S. Dey, and W. F. Stewart. Early detection of heart failure using electronic health records. *Circulation: Cardiovascular Quality and Outcomes*, 2016.

[22] J. Spohrer and G. Banavar. Cognition as a service: an industry perspective. *AI Magazine*, 36(4):71–86, 2015.

[23] E. E. Tripoliti, T. G. Papadopoulos, G. S. Karanasiou, K. K. Naka, and D. I. Fotiadis. Heart failure: Diagnosis, severity estimation and prediction of adverse events through machine learning techniques. *Computational and Structural Biotechnology Journal*, 15(Supplement C):26 – 47, 2017.

[24] B. Upbin. Ibm watson hits daily double fighting cancer with memorial sloan kettering, 2012. `https://www.forbes.com/sites/bruceupbin/2012/03/22/ibm-watson-hits-daily-double-fighting-cancer-with-memorial-sloan-kettering/`.

[25] B. Upbin. Ibm's watson gets its first piece of business in healthcare, 2013. `https://www.forbes.com/sites/bruceupbin/2013/02/08/ibms-watson-gets-its-first-piece-of-business-in-healthcare/`.

[26] N. Yorke-Smith, S. Saadati, K. L. Myers, and D. N. Morley. The design of a proactive personal agent for task management. *International Journal on Artificial Intelligence Tools*, 21(01):1250004, 2012.