ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

IL PROGETTO SMART SHOCK ROOM: STUDIO E SVILUPPO DI UN AMBIENTE OSPEDALIERO INTELLIGENTE BASATO SULL'INTEGRAZIONE DI APPROCCI AD AGENTI, ASSISTENTI VOCALI E DIGITAL TWIN

Tesi in
PERVASIVE COMPUTING

Relatore
Prof. ALESSANDRO RICCI

Presentata da
MANUEL BOTTAZZI

Corelatore
Dott. Ing. ANGELO CROATTI

Anno Accademico 2019 – 2020
Sessione I

Per Aspera Sic Itur Ad Astra

Alla mia famiglia,
fatta di legami di sangue
e legami di cuore

Indice

Introduzione

"*The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.*" scrisse Weiser nel 1991 in [26] ragionando sul futuro degli elaboratori. Nonostante siano passati quasi 30 anni le idee che emergono da quell'articolo sono quanto mai attuali, anzi hanno trovato negli ultimi anni la loro massima diffusione. Oggi è comune vedere oggetti della vita quotidiana contenenti sensori, attuatori o dispositivi con capacità computazionali e la maggioranza delle persone dispongono di uno smartphone con una potenza di elaborazione sempre maggiore. Nell'ultimo periodo questo genere di tecnologie si sta espandendo dai semplici oggetti in ambiti sempre più grandi, fino a coprire intere città.

In quest'ottica è quindi inevitabile che la tecnologia si infiltri in ogni aspetto della vita umana, dal lavoro all'abitazione, dalla mobilità alla gestione dell'energia fino alle strutture ospedaliere, fornendo un supporto sia agli utilizzatori che agli addetti ai lavori. Il progetto di tesi presentato in questo documento si inserisce in questo contesto andando ad analizzare i punti di forza dell'applicazione di un'architettura capace di integrare diversi paradigmi e modelli tecnologici per la modellazione e la realizzazione di un ambiente intelligente in ambito ospedaliero, in grado di fornire un supporto al lavoro di un'equipe di medici esperti durante il trattamento di pazienti con traumi gravi. In questo scenario, particolarmente critico, il vantaggio che sistemi di questo genere possono portare è ancora più evidente, e per testarne al meglio l'efficienza verrà realizzato un prototipo di applicazione sfruttando le caratteristiche dell'architettura sopra evidenziata.

In questa tesi sarà quindi presentato il progetto Smart Shock Room e verranno analizzate nel dettaglio le funzionalità che un ambiente intelligente può fornire ad un ambiente di questo tipo e con quali modalità. Per fare questo sarà inevitabilmente necessario confrontarsi e collaborare con l'equipe medica che si troverà ad agire in quell'ambiente. Una volta evidenziati i requisiti che tale ambiente dovrà soddisfare e messo in luce il modello del dominio in esame,descritti nel secondo capitolo, si procederà nel terzo capitolo ad un'analisi delle possibili modalità di integrazione tra diversi modelli e paradigmi tecno-

logici al fine di ottenere un'architettura ottimizzata per l'implementazione del sistema in maniera efficiente ed efficace.

Nel quarto capitolo verrà trattata l'implementazione del prototipo del sistema, con particolare attenzione alle scelte tecnologiche fatte e ai requisiti specifici che sono stati scelti per l'implementazione. In questa fase verrà anche gestito uno degli aspetti più importanti di questo genere di applicazioni che riguarda le interfacce e le modalità di interazione con l'utente. Nell'ambito specifico del caso di studio infatti sarà necessario prevedere modalità di interazione multimodali che rendano possibile l'utilizzo hands-free del sistema.

Infine nel quinto capitolo verranno presentati i risultati dei test effettuati sul sistema per valutarne le performance e i risultati della validazione con il team di medici dell'ospedale al fine di poter verificare la bontà del risultato stesso, mostrando come il sistema ottenuto presenti un buon grado di efficienza e risponda alle richieste del team di esperti del dominio ed allo stesso tempo risulti valido in termini di architettura. Infatti la possibilità di integrare diversi approcci ha permesso di modellare ogni parte del domino secondo il paradigma che più è risultato adatto e quindi di ridurre notevolmente il lavoro complessivo grazie ad una modellazione il più possibile semplice e lineare.

Il progetto presentato in questo documento rappresenta quindi la prima base di un sistema più ampio nell'ottica dei sistemi pervasivi, diffuso e integrato per tutta la struttura ospedaliera, in cui diversi ambienti possano interagire tra loro e con gli utenti umani che si trovano al loro interno al fine di ottenere un supporto specifico e diffuso, presentando inoltre una possibile architettura per permettere la modellazione di tale sistema ed una valida scelta tecnologica per la sua implementazione.

Capitolo 1

Sistemi Pervasivi e Smart Hospital

Viviamo in un periodo di grande espansione tecnologica e l'IT e le sue applicazioni stanno sempre più trovando spazio in ogni aspetto della vita quotidiana. Negli ultimi anni questo sviluppo tecnologico ha portato le tecnologie informatiche ad uscire dall'ambito classico ed estendere il loro influsso sull'ambiente stesso che ci circonda. Testimonianza di questo è l'enorme aumento nella diffusione di tecnologie Internet of Things e di oggetti Smart, che hanno generato in Italia un mercato da 6.2 miliardi di euro nel solo 2019, con un aumento del 24 % sull'anno precedente [3]. Sempre in questo studio emerge come i settori in maggiore crescita sono legati alle Smart home (+40%), trainato in particolare dal boom degli assistenti vocali, le Smart factory dell'Industria 4.0 (+40%) e le Smart city (+32%). Queste tecnologie permettono sempre più di fornire supporto agli utenti e funzionalità digitali all'ambiente stesso, aprendo nuove possibilità verso un ambiente intelligente, o *Smart Environment*. Questo rappresenta quindi un ambito in forte espansione, in cui il lavoro di ricerca trova ampio spazio di manovra.

In questo lavoro di tesi verrà analizzata la possibilità di integrazione di alcune tecnologie innovative per lo sviluppo di Smart Enviroment e la relativa applicazione per la realizzazione di un'applicazione. In seguito ad alcune richieste ricevute dall'equipe medica dell'ospedale Maurizio Bufalini, con il quale il dipartimento di informatica della nostra università collabora, è stato scelto come ambito specifico per l'applicazione da sviluppare il dominio dell'healthcare. Il caso di studio, descritto in seguito, riguarderà lo sviluppo di uno Smart Environment per l'ambiente della Shock Room, un ambiente ospedaliero dedicato alla gestione dei traumi.

1.1 Sistemi Pervasivi e Smart Environment

L'enciclopedia Treccani definisce pervadere come "*Infiltrarsi e diffondersi in un ambiente compenetrandolo*" [1]. L'idea che le tecnologie informatiche possano entrare nell'ambiente fisico in modo da fornire funzionalità specifiche emerge per la prima volta in [26]. In questo articolo Weiser definisce l'idea di *Ubiquitous Computing*, in cui centinaia di dispositivi digitali piccoli ed economici sono fisicamente distribuiti nell'ambiente fisico, fungendo da sensori, attuatori, display e simili, fornendo un comportamento complessivo dato dalla loro interazione. Già in questa prima idea emerge l'importanza della fusione di tali dispositivi con l'ambiente, in modo che l'interazione con l'utente avvenga in maniera naturale. Come vedremo successivamente questo aspetto risulta quanto mai importante anche oggi. Un'altra proprietà che caratterizza tali sistemi è la presenza di *Context Awareness*, ossia la capacità del sistema di raccogliere informazioni sull'ambiente in cui è in esecuzione ed adattare il proprio comportamento di conseguenza.

1.1.1 Smart Environment

Il termine Smart Environment viene definito in [12] come "*un piccolo mondo in cui diverse tipologie di smart device lavorano continuamente per rendere più confortevole la vita ai suoi abitanti*". Più formalmente può essere definito, sempre secondo la stessa fonte, come un ambiente capace di acquisire informazioni riguardo all'ambiente stesso ed applicarle, adattandole alle entità che lo abitano, per migliorare la loro esperienza all'interno dell'ambiente. Le capacità e le funzionalità specifiche perciò dipendono da cosa l'utente desidera ottenere dall'ambiente stesso, ma in generale comportano l'automazione di uno o più compiti che di solito vengono svolti in tale ambiente. L'obiettivo finale è quindi quello di collegare dispositivi computazionali ad attività ed ambienti quotidiani.

Ogni Smart Environment presenta alcune caratteristiche specifiche:

- *Controllo remoto dei dispositivi*: Il sistema dovrebbe prevedere il controllo remoto e/o automatico dei dispositivi presenti nell'ambiente, permettendone l'utilizzo senza la necessità di accesso fisico agli stessi.

- *Comunicazione tra dispositivi*: I dispositivi presenti nel sistema costituiscono un ambiente connesso, comunicando tra di loro sfruttando le diverse tecnologie disponibili per condividere dati ed informazioni ed accedere a informazioni e risorse esterne accessibili tramite la rete Internet per migliorare la loro capacità di svolgere task.

[1] `http://www.treccani.it/vocabolario/pervadere/`

- *Acquisizione di informazioni tramite reti di sensori*: La rete di sensori diffusa nell'ambiente permette di raccogliere informazioni e adattare lo stato del sistema in base alle condizioni ambientali e dei suoi utilizzatori.

- *Integrazione con Smart Device*: La presenza nell'ambiente di dispositivi Smart permette di integrare nell'ambiente un insieme di funzionalità e servizi offerte da tali dispositivi.

- *Predittività e Decision-Making*: Lo smart environment può prendere decisioni oer automatizzare il suo comportamento in modo da ottenere uno specifico obiettivo.

- *Accessibilità tramite la rete*: L'ambiente presenta dispositivi che possono essere gestiti e controllati tramite la rete, sfruttando standard condivisi.

Computer Supported Cooperative Work

Il Computer Supported Cooperative Work (CSCW), come descritto in [10], studia come le attività collaborative di un gruppo di umani e la loro collaborazione può essere supportata da un sistema informatico. Esso può supportare gli umani nell'ambito lavorativo in diversi modi, come ad esempio offrendo mezzi di comunicazione, migliorando le possibilità di monitoraggio oppure riducendo la complessità delle attività di coordinazione tra gli attori interessati. I sistemi di questo tipo coprono due aspetti fondamentali:

- *Gestione dell'interdipendenza*: Gli attori si trovano ad agire su un ambiente condiviso su cui le loro azioni portano effetti. Affinché essi possano collaborare è quindi necessario che siano a conoscenza dei cambiamenti ambientali e della presenza degli altri attori in modo da programmare le proprie azioni di conseguenza, in quanto il risultato di una azione modifica il contesto anche degli altri attori.

- *Spazio di informazione condiviso*: Gli attori devono poter monitorare l'ambiente e la sua evoluzione in seguito alle azioni effettuate e devono poter accedere ad oggetti condivisi con altri. Sarà quindi necessario avere un sistema capace di identificare e tracciare gli oggetti.

Questi aspetti sono importanti da tenere in considerazione anche quando si progetta uno Smart Environment in cui più persone si trovano a collaborare per raggiungere un obiettivo comune. Questo è comune in ambienti progettati per un ambito lavorativo.

1.1.2 Tecnologie Abilitanti

Lo sviluppo di Smart Environment nasce nell'ambito dei sistemi distribuiti e dell'Internet of Things e si basa su un insieme di tecnologie di base, abilitanti per lo sviluppo di questo tipo di sistemi. Per poter funzionare avranno bisogno ad esempio di protocolli per la comunicazione e il networking, algoritmi distribuiti, framework e middleware dedicati, strumenti per la gestione di sensori ed attuatori, elaborazione di immagini e speech recognition.

Internet of Things

Internet of Things (IoT), prendendo la definizione presente in [18], è un "*sistema di oggetti fisici che possono essere scoperti, monitorati, controllati e con cui si può interagire tramite dispositivi elettronici, che comunicano tramite diverse interfacce di rete e possono essere eventualmente collegati a Internet*". Si parla quindi di una rete di oggetti fisici a cui vengono fornite funzionalità aggiuntive e capacità computazionali in modo tale da renderlo un *oggetto aumentato*. Questi oggetti presentano sensori ed attuatori per interagire con il mondo fisico e interfacce di comunicazione con il quale possono comunicare con altri dispositivi. Tramite questa rete è possibile una nuova tipologia di interazione con l'ambiente, il quale fornisce nuove funzionalità tramite questi oggetti connessi.

Le Thing di IoT sono delle più svariate forme e variano da oggetti piccoli e semplici come tag RFID ed oggetti indossabili fino a sistemi complessi come una Smart Car. La caratteristica comune è quella di poter accedere ad essi tramite la rete Internet, in maniera diretta o indiretta. Questa è la maggiore potenzialità di questo sistema ma ad oggi anche il suo più grande limite, in quanto mette in campo tanti diversi protocolli di comunicazione che vanno a limitare l'interoperabilità e la possibilità di collegare dispositivi diversi.

Protocolli per IoT

La scelta del protocollo di comunicazione più adatto dipende da diversi fattori, tra cui:

- La distanza tra i dispositivi da collegare,
- Il tipo di collegamento (fisico o wireless),
- Il consumo energetico,
- L'ampiezza di banda richiesta,
- La latenza consentita.

Il numero di protocolli disponibili è molto elevato, ed ognuno presenta specifiche caratteristiche e peculiarità per cui sono da preferire in certe applicazioni piuttosto che altre. Alcuni dei protocolli maggiormente utilizzati sono BlueTooth e BLE, AMQP, CoAP, ZigBee, NFC, MQTT, ...

Web of Things

Una interessante estensione di IoT è Web of Things, che si prefigge come obiettivo quello di realizzare una rete di oggetti intelligenti sfruttando i protocolli e le tecnologie tipiche del World Wide Web, in particolare il protocollo HTTP. Sfruttando questi protocolli si rende più semplice l'accesso e la programmazione di questi sistemi, usando linguaggi e protocolli ampiamente diffusi. L'uso di un linguaggio comune rende inoltre più semplice l'estensibilità e l'interoperabilità tra sistemi, migliorando deploy, manutenzione ed estensione del sistema. Inoltre il protocollo HTTP permette un disaccoppiamento tra i vari dispositivi, rendendo di fatto il sistema aperto, in quanto la comunicazione avviene tramite la web API esposta dal singolo oggetto connesso.

1.2 Smart Healthcare nelle Smart City

Il paradigma IoT trova applicazione in vari ambiti della vita quotidiana, non solo negli ambienti domestici ma anche in industria e nelle città. Già da qualche anno infatti si sta diffondendo sempre di più il concetto di Smart City e con essa l'idea di applicare le moderne tecnologie negli ambiti della pianificazione urbana e della vita pubblica.

1.2.1 Smart City

Il concetto di Smart City, come descritto in [19], indica un insieme di strategie da mettere in campo per creare uno sviluppo economico sostenibile ed una qualità della vita più alta per i cittadini di una città, combinando le tecnologie dell'informazione con il capitale umano e sociale degli abitanti insieme con investimenti in nuove infrastrutture e servizi per il cittadino.

Per rendere possibile questo tipo di sviluppo la tecnologia informatica offre un contributo fondamentale a diversi livelli. Infatti per poter funzionare in maniera efficiente un sistema del genere necessita di strumenti e sensori embedded nell'ambiente per l'acquisizione diffusa di dati ed informazioni e di reti che rendano tali dati accessibili ed interconnessi, in modo che possano essere raccolti, condivisi ed utilizzati dalle persone e dalle istituzioni della città. Inoltre acquista via via maggiore importanza l'applicazione di algoritmi intelligenti ed AI per il supporto nell'analisi dei dati e lo sviluppo di modelli predittivi,

ad esempio negli ambiti della gestione del traffico o dei flussi dell'energia. La figura 1.1, tratta da [19], rappresenta la tipica architettura di un sistema IoT per una Smart City, capace di collegare ed integrare diversi ambiti ed aspetti dell'ambiente cittadino per migliorare lo sviluppo di tutti i settori.

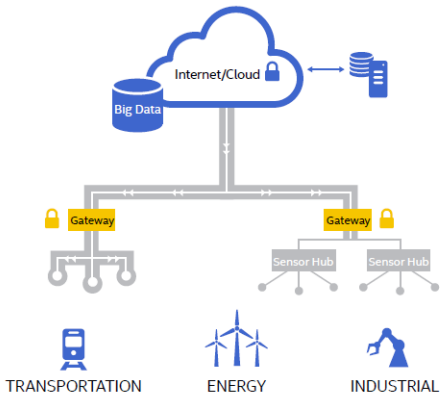

Figura 1.1: Architettura IOT di una smart city

Vista questa necessità di integrazione ed interazione lo sviluppo di una Smart City non si può limitare ad un solo aspetto della vita della città, ma deve invece interessare molteplici ambiti. In particolare, secondo l'agenzia di consulenza Frost & Sullivan, si può definire Smart una città che ha una pianificazione o dei progetti attivi in almeno cinque degli otto ambiti da loro evidenziati, mostrati in figura 1.2.

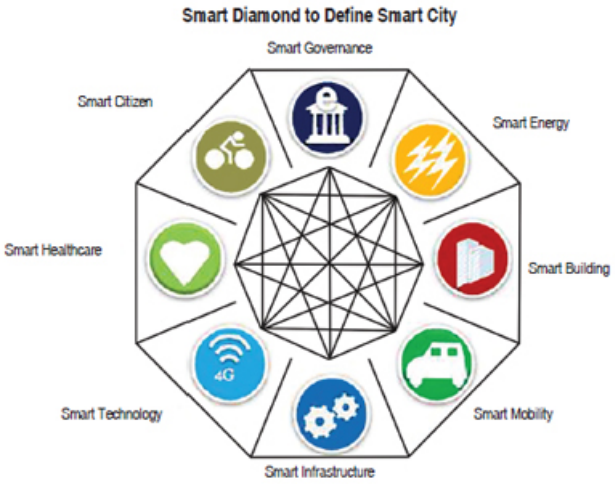

Figura 1.2: Ambiti di una smart city (tratta da Frost & Sullivan [5])

1.2.2 Smart Healthcare

Questo lavoro di tesi si concentrerà in uno specifico di questi ambiti, quello dello Smart Healthcare. In [19] viene descritto come obiettivo dello Smart Healthcare la capacità di permettere migliori diagnosi, supportare il personale nel trattamento delle malattie e i pazienti per permettere loro di scegliere in maniera più consapevole ed intelligente riguardo la loro salute. Questo aspetto sta diventando man mano sempre più importante dati i sempre maggiori problemi nella gestione della salute pubblica data dall'aumento della popolazione. Infatti, come descritto in [37], il numero di medici non aumenta in maniera proporzionale alla popolazione e questo aumenta il rischio di ricevere un trattamento non adeguato. L'applicazione di tecnologie ICT nell'ambito della cura della persona porta grandi vantaggi, permettendo di mantenere e aumentare l'efficienza e l'accuratezza del trattamento nonché la sostenibilità per medico e paziente. Questo è possibile sfruttando reti di sensori, IOT e dispositivi avanzati, strumentazioni mediche Smart e wearable tra loro interconnesse, in grado di elaborare i dati medici dei pazienti.

Ad oggi esistono svariati esempi di applicazioni di questo tipo di tecnologie nell'ambito della cura della persona, alcune delle quali verranno approfondite a titolo di esempio nei prossimi paragrafi. Analizzando la letteratura gli ambiti principali in cui ad oggi le tecnologie informatiche trovano applicazione nella cura della persona sono i seguenti:

- Prevenzione ed informazione.
- Monitoraggio e Supporto del paziente.
- Raccolta e monitoraggio di dati biometrici.
- Accesso remoto ad esami e referti.
- Supporto alla diagnostica e al lavoro dei medici.

1.2.3 Smart Hospital

L'ospedale è il luogo in cui avviene principalmente il processo di cura dei pazienti e in cui ogni giorno un gran numero di persone, tra tecnici, infermieri, medici e pazienti, si trovano a transitare e lavorare. Viene quindi naturale pensare, nell'ottica di ottenere e migliorare la Smart Healthcare sopra descritta, di fornire questo ambiente con delle funzionalità digitali capaci di ottenere ed integrare informazioni e di fornire supporto agli utilizzatori del servizio offerto dall'ospedale stesso.

L'ospedale rappresenta un sistema molto complesso, con svariati servizi da gestire e controllare. L'applicazione di tecnologie IoT e più in generale

di tecnologie informatiche permettono una maggiore integrazione tra i vari settori e un accesso facilitato alle informazioni a tutti i livelli. La figura 1.3 mostra alcuni dei servizi che l'applicazione di queste tecnologie possono fornire ai pazienti e al personale sanitario.

Figura 1.3: Ambiti di uno Smart Hospital, tratto da `http://www.idforward.it/smart-hospital/`

Il risultato dell'applicazione di queste tecnologie è quello di ottenere un ecosistema integrato, veloce ed efficiente, capace di migliorare il servizio offerto all'utente in termini di tempo necessario e di qualità, lasciando così una migliore impressione del servizio offerto.

Inoltre lo sviluppo tecnologico mette in luce sempre nuovi scenari di applicazione, rendendo oggi possibile pensare a tecnologie di monitoraggio per i pazienti anziani, meccanismi di assistenza personale al personale medico, integrazione di sistemi di Intelligenza Artificiale per il supporto alle decisioni e alla diagnostica e tanto altro, come descritto ad esempio in [21].

Un esempio: Careggi Smart Hospital

Uno degli esempi di Smart Hospital presenti nell'ambito della sanità italiana è quello del policlinico Careggi di Firenze. Come descritto in [24], all'interno di questa struttura sono attivi diversi sistemi che permettono di gestire la posizione di ambulatori e personale, ricavando in caso di necessità la posizione del singolo medico. Ad essi è affiancata una applicazione mobile per il pubblico che offre diversi servizi oltre all'orientamento e alla ricerca. Tramite essa è infatti possibile accedere alle proprie informazioni e ai propri referti medici

tramite l'interazione con il sistema informativo dell'azienda ospedaliera, consultare e chiamare numeri utili, visualizzare la modulistica e pagare ticket e anche trovare un alloggio o un punto di ristoro nei pressi dell'ospedale. L'idea è quindi quella di fornire agli utenti un unico punto d'accesso per interagire con i servizi messi in campo dall'ospedale.

Gestione delle emergenze negli Ospedali Intelligenti

Un altro ambito in cui l'applicazione di approcci tecnologici innovativi porta un contributo notevole è quello relativo alla gestione delle emergenze all'interno di ambienti come gli Smart Hospital. Essi infatti possono fornire un grande supporto in questi casi in cui i fattori chiave sono il tempo e l'accesso alle informazioni, due aspetti che possono essere supportati dal mezzo tecnologico. Questo è evidenziato da alcuni studi, come ad esempio [38], in cui viene descritto come in alcuni casi il personale medico spende fino al 50 % di tempo in più rispetto a quello effettivamente necessario per diagnosi e trattamento a causa di inefficienze nel passaggio di informazioni, che in casi di emergenza può fare la differenza tra la vita e la morte del paziente.

In tale studio viene proposto un sistema, *SmartER*, che permette di velocizzare le procedure tramite un servizio web per la visualizzazione delle informazioni relative al paziente, sempre accessibili tramite un QRCode sul suo polso, affiancato ad un sistema di IA per supportare i medici nel processo decisionale.

Un altro esempio rilevante è *SHES*, *Smart Hospital Emergency System*, presentato in [6]. Questo sistema è ideato per migliorare l'efficienza del servizio di emergenza fornendo in automatico una serie di informazioni relative al paziente che chiama il centralino per le emergenze sfruttando anche le informazioni raccolte tramite i sensori presenti nello smartphone. Queste informazioni possono poi essere visualizzate dagli esperti dell'ospedale tramite un servizio web.

1.3 Sistemi Pervasivi nell'Healthcare

Viene definito Pervasive Healthcare l'applicazione di metodi e tecnologie tipiche dei sistemi pervasivi all'healthcare e alla cura della persona. Come descritto in [7] questo ha l'obiettivo di ottenere ambienti smart integrati, con l'obiettivo di migliorare le qualità della cura e diminuire i costi per il sistema sanitario. Una architettura tipica per applicazioni di questo tipo è quella mostrata in figura 1.4, in cui si nota come il livello applicativo del sistema pervasivo si va ad interfacciare con il sistema informativo aziendale e genera informazioni accessibile tramite un'interfaccia web. Per questi sistemi infatti l'aspetto di integrazione è fondamentale.

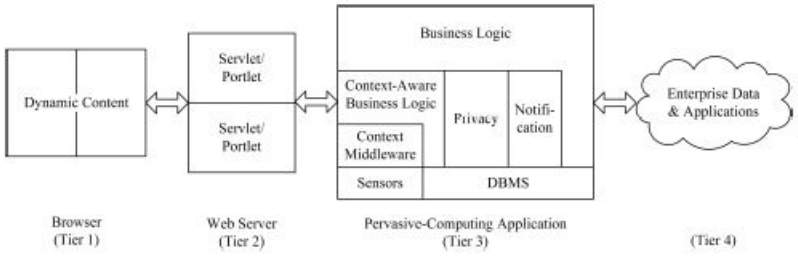

Figura 1.4: Architettura generale di un sistema pervasivo nell'healthcare, tratto da [7]

1.3.1 Smart Healthcare e Digital Twin

L'idea dei Digital Twin nasce in ambito industriale nell'ottica dell'industria 4.0, ma sta trovando ampia diffusione anche nell'ottica delle Smart City. Tale modello propone di modellare ogni entità con un'entità digitale che rappresenta la copia di una certa entità fisica, con la quale è strettamente collegata e ne condivide il comportamento e lo stato. In questo modo ogni entità è modellata come unione delle due parti. Questo argomento verrà ripreso ed approfondito nella sezione 3.3. Questo modello trova oggi un gran numero di applicazioni, come visibile in letteratura, e trova anche alcune applicazioni interessanti nell'ambito healthcare.

Internet of Health Things (*IoHT*)

Come definito in [14] Internet of Health Things (IoHT) propone l'integrazione di mondo digitale e mondo fisico tramite l'inserimento nei dispositivi medici di capacità di connessione alla rete. In questo modo si possono sfruttare i vantaggi dei dispositivi IoT nei dispositivi medici, permettendo lo scambio di informazioni e la possibilità di fornire capacità aggiuntive. Questa idea riprende il modello dei Digital Twin e ne rappresenta una applicazione specifica nel mondo dell'healthcare.

Esempi di Digital Twin in Healthcare

Cardio Twin

Il progetto *Cardio Twin*, presentato in [27], propone un'architettura per la realizzazione di un Digital Twin del cuore umano, tale da permettere un monitoraggio continuo e puntuale al fine di ridurre il rischio e gli effetti di malattie cardiovascolari come gli infarti sui pazienti interessati. Questo progetto è particolarmente interessante poichè l'oggetto mappato come Digital Twin non è

un componente meccanico ma una parte biologica di un individuo. Il sistema si pone come obiettivo quello di raccogliere dati dai sensori presenti sul paziente, collegati tramite bluetooth ad uno smartphone, e tramite tecniche di Machine Learning incrociarli con i dati presi dalle cartelle cliniche, dai social e da altri sensori esterni in modo da ottenere informazioni più dettagliate. Tali informazioni potranno essere usate da un medico per monitorare il paziente o effettuare una diagnosi, o addirittura per allertare i soccorsi in caso di emergenza.

DT per Healthcare Personalizzata

Rivera mostra in [36] come l'avanzamento tecnologico sta portando la medicina ad livello di precisione e personalizzazione che fino a pochi anni fa era impensabile. Con le tecnologie e la grande quantità di dati oggi disponibili è infatti possibile effettuare monitoraggi continui e predizioni accurate, migliorando l'efficacia del trattamento sul paziente e riducendo i costi per le strutture ospedaliere. Data la grande mole di dati e la complessità risultante in tale articolo viene mostrato come i Digital Twin rappresentano una modalità ottimale per permettere questo tipo di personalizzazione, in quanto rispondono alla necessità di collegare la componente fisica dell'ambiente con la parte computazionale del sistema. Tali entità possono portare verso una versione più personalizzata della medicina, aiutando nella raccolta e nell'analisi dei dati ma anche simulando il risultato che un certo trattamento avrebbe sul paziente, tramite la generazione di modelli accurati e fedeli, diversi da paziente a paziente.

1.3.2 Sistemi ad Agenti per l'Healthcare

Un altro modello tecnologico che oggi sta trovando molteplici applicazioni in ambito medico ed healthcare è quella dei sistemi ad agenti o multiagente. Sistemi di questo tipo presentano entità computazionali con un certo grado di autonomia e di flessibilità, e quindi in grado di gestire al meglio la grande complessità del dominio medico in questione. Essi trovano inoltre ampia diffusione nella gestione dell'interazione con l'utente o in sistemi di supporto alle decisioni, in collaborazione con tecniche di intelligenza artificiale. Questo argomento verrà approfondito successivamente nella sezione 3.2.

Isern riporta una serie di esempi di applicazioni di sistemi di questo genere nell'ambito dell'healthcare in una systematic literature review del 2016, [22], da cui emerge innanzitutto una duplice classificazione delle applicazioni in questo dominio. In base al tipo di utilizzatore i sistemi possono essere suddivisi in:

- *Strumenti per l'organizzazione ospedaliera*, i più comuni, che hanno l'obiettivo di supportare lo scambio di informazioni e rendere più efficienti i processi interni tramite simulazioni e teleassistenza.

- *Strumenti per i professionisti*, spesso nella forma di assistenti personali, hanno l'obiettivo di elaborare informazioni in modo da supportare il personale sanitario nel loro lavoro giornaliero.

- *Strumenti per i pazienti*, che hanno cioè l'obiettivo di fornire un servizio specifico al paziente. Questo varia moltissimo nella tipologia e comprende dall'informazione al monitoraggio alla generazione e simulazione di processi di cura personalizzati.

Una ulteriore classificazione possibile è in base al tipo di applicazione. Tale documento evidenzia 7 differenti tipologie: *Monitoraggio dei dati*, *Sistemi di monitoraggio ed allerta automatica*, *Meccanismi per sicurezza e privacy*, *sistemi di supporto alle decisioni*, *Sistemi di scheduling e pianificazione di attività*, *Ambienti di simulazione* e *Piattaforme di gestione dell'assistenza generale*. Nello studio in questione vengono analizzati diversi lavori, 133 per la precisione, classificandoli secondo questi parametri, mettendo in luce come l'applicazione di sistemi multiagente nell'ambito dell'healthcare abbia delle potenzialità in quanto permette di gestire l'intrinseca complessità di tali sistemi con un buon risultato in termini di tempo di sviluppo, costi e performance, principalmente grazie alla flessibilità e modularità offerta.

Sistemi Conversazionali in Ambito Medico

Una particolare applicazione dei sistemi ad agenti che sta trovando ampia diffusione è quelle dei sistemi conversazionali. Come descritto in [25] essi sono sistemi digitali in grado di interagire e conversare con l'utente umano usando il linguaggio naturale. Da tale documento emerge come siano presenti svariati esempi di applicazioni di sistemi conversazionali in diversi settori specifici della cura del malato e del supporto al lavoro del personale medico, e come da una prima ricerca su motori specializzati siano emersi ben 6036 risultati relativi ad articoli e pubblicazioni. Questo numero può dare un'idea di quanto queste tecnologie stiano impattando il settore.

Sempre citando [25], in generale i sistemi conversazionali si possono suddividere in base al tipo di dialogo che permettono o al mezzo con cui vengono resi disponibili, ma la suddivizione più significativa è quella per tipologia:

- *Sistemi Text-based*, in cui l'utente interagisce tramite un'interfaccia testuale con il sistema stesso.

- *Embodied Conversational Agent*, in cui l'agente unisce al dialogo la presenza di sembianze umane simulate, in modo che la comunicazione possa avvenire anche attraverso canali non verbali (e.g. movimenti ed espressioni facciali).

- *Assistenti vocali/virtuali*, tipologia mista rispetto alle precedenti, in quanto presentano tratti tipici dell'essere umano, quali la comunicazione verbale e la possibilità di comunicare in modo più o meno empatico ma non presentano una rappresentazione virtuale vera e propria, tipica invece degli agenti embodied. Un esempio classico di agenti di questa tipologia sono gli assistenti che si trovano integrati nei moderni smartphone (e.g. Siri, Google Now, Cortana, ...) o in appositi dispositivi (e.g Amazon Echo, Google Home, ...)

Le applicazioni di tali sistemi in ambito medico sono le più svariate, anche se quelle che trovano un utilizzo più diffuso riguardano il supporto alla salute mentale del paziente, sistemi di monitoraggio ed autodiagnosi e supporto alla ricerca di informazioni. Inoltre sono presenti un gran numero di applicazioni che supportano l'utilizzatore nel cambiamento di una certa abitudine poco salutare in un'ottica di prevenzione. Il principale punto critico che emerge riguarda la difficoltà nell'utilizzo di questo genere di applicazioni a causa di interfacce non intuitive e troppo complesse. Uno dei requisiti fondamentali del progetto stesso riguarda la necessità di interagire velocemente e facilmente con il sistema, per cui anche questo aspetto sarà tenuto in grande considerazione.

Quello che emerge anche da questa ricerca è che la maggior parte delle applicazioni sono orientate al paziente, lasciando uno spazio molto limitato ad applicazioni dedicate al supporto del personale medico, con solo qualche esempio nell'ambito della formazione e della simulazione. Questo rappresenta un grande limite, in quando il beneficio che tali applicazioni potrebbero portare in un ambiente di lavoro in cui l'interazione hands-free risulta spesso l'unica fruibile da parte del personale operante è veramente grande, come cercheremo di dimostrare nel progetto di tesi presentato in questo documento. Anche in Isern [22] emerge che solo il 17.5% degli studi presi in esame è rivolto al personale sanitario. Questo mette in luce ancora di più l'importanza che può andare a ricoprire questo progetto di tesi, in quanto si inserisce in un ambito specifico ed ancora non molto esplorato.

Il Sistema Trauma Tracker

Un ulteriore esempio significativo della tecnologia ad agenti applicata nell'ambito dell'healthcare è rappresentato dal sistema Trauma Tracker. Questo sistema, che si pone nell'ambito del supporto al personale medico, è presentato nel dettaglio in [28] e permette di tracciare e monitorare il processo di cura di un paziente traumatizzato, annotando eventi, spostamenti e procedure, producendo così una documentazione accurata e precisa della sequenza temporale degli eventi e della posizione in cui essi sono avvenuti.

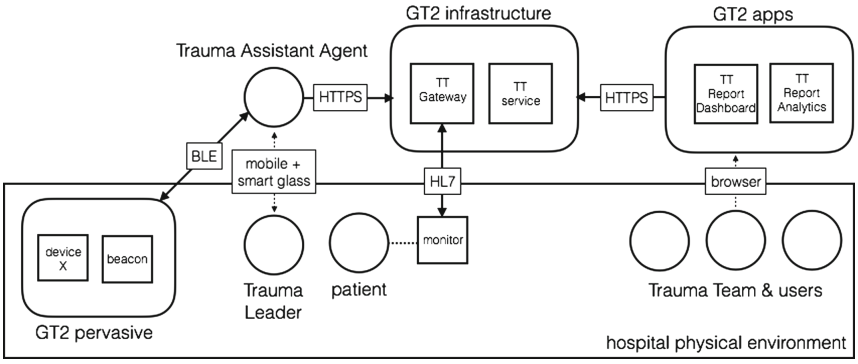

Figura 1.5: Architettura e protocolli usati da Trauma Tracker, tratto da [13]

Tale sistema basa una parte fondamentale del suo funzionamento su un agente, *Trauma Leader Assistant Agent* (*TLAAgent*), con cui il trauma leader, il responsabile del team di soccorso, interagirà mediante un tablet e/o degli smart glass durante la gestione del trauma. Tramite esso il trauma leader potrà annotare operazioni ed informazioni ritenute importanti ottenendo come risultato un report dettagliato. L'uso di questo sistema porta notevoli vantaggi all'equipe medica, che in assenza di tale sistema deve annotare gli avvenimenti a mano su carta. Questo comporta quindi un inevitabile spreco di tempo. È stato inoltre dimostrato dall'articolo sopracitato come la qualità dei dati raccolti sia migliore, permettendo di avere accesso ad informazioni altrimenti non disponibili, e possa fornire supporto all'equipe medica sia in fase acuta, fornendo informazioni utili in procedure critiche, sia in fase di revisione del caso.

1.4 Il Progetto Smart Shock Room

Da una collaborazione tra il dipartimento di Ingegneria e Scienze Informatiche (*DISI*) dell'università di Bologna e il personale medico dell'ospedale Maurizio Bufalini di Cesena, in particolare con i reparti di terapia intensiva e medicina d'urgenza è nata l'idea di realizzare all'interno dell'ospedale un ambiente intelligente capace di supportare l'equipe medica durante il trattamento dei pazienti che hanno subito un trauma. Questo ambiente, la *Smart Shock Room*, rientra nell'ambito di un progetto di collaborazione più ampio che punta alla digitalizzazione e allo sviluppo tecnologico di diversi aspetti del processo di cura offerto ai pazienti e dell'ospedale stesso, in modo da fornire un ambiente all'avanguardia. All'interno di tale collaborazione è stato sviluppato anche il sistema Trauma Tracker, presentato nella sezione precedente.

1.4.1 La Shock Room e il Trauma Team

Innanzitutto è necessario chiarire il contesto nel quale è nata l'idea del progetto Smart Shock room ed in cui esso si andrà a posizionare.
L'ospedale Bufalini di Cesena è quello designato dall'AUSL locale per il trattamento dei pazienti che presentano traumi gravi e presenta un gruppo designato altamente specializzato in grado di rispondere a queste situazioni di emergenza, il Trauma Team. L'ambiente privilegiato in cui si svolge la cura di questi pazienti è quello della Shock Room, una stanza dedicata in cui è possibile trovare tutta la strumentazione necessaria per poter agire in modo efficace ed efficiente sul paziente. In tale contesto il tempo a disposizione è una risorsa quantomai preziosa e da qui è nata l'idea di fornire un supporto al Trauma Team per riuscire ad ottimizzare il loro operato affiancandogli strumenti tecnologici avanzati.

1.4.2 Obiettivo della Tesi

L'obiettivo che si propone questa tesi è duplice. Il primo obiettivo risponde ad una richiesta di realizzare un sistema, e sarà quindi necessaria quindi la progettazione e lo sviluppo di una soluzione tecnologica adeguata ai requisiti della Shock Room, in modo da fornire funzionalità digitali all'ambiente tramite un sistema distribuito e pervasivo. Per fare questo sarà necessario valutare con attenzione quali possono essere le necessità di un simile ambiente e delle persone che ci operano all'interno. Il sistema così emerso non dovrà solo funzionare ma dovrà offrire un supporto al lavoro dei medici e degli infermieri del Trauma Team, in modo che possano ottimizzare il lavoro aumentando l'efficacia e la qualità del processo di cura e minimizzando il tempo perso a svolgere attività che possono essere svolte in automatico.

È possibile tuttavia ottenere un sistema del genere utilizzando svariate tecnologie, con un'applicazione più o meno complessa. Sarà quindi il secondo obiettivo di questo lavoro quello di valutare l'efficacia dell'applicazione di alcune tecnologie innovative e della loro integrazione, applicandole al caso di studio proposto, in modo da poterne mettere in luce gli aspetti positivi e le criticità che potranno emergere. Come visto alcune tecnologie molto promettenti in ambito healthcare sono rappresentate dai sistemi multiagente e dai Digital Twin, che verranno pertanto selezionate come candidati ideali per questa integrazione. Data la natura del progetto verrà inoltre integrata una modalità di gestione hands-free tramite un assistente vocale in modo da migliorare l'usabilità durante le operazioni di soccorso.

Capitolo 2

Il Progetto Smart Shock Room

L'idea di rendere Smart un ambiente di lavoro come quello della Shock Room si inserisce all'interno di un progetto più ampio che vede l'inserimento nelle procedure e nei protocolli dell'ospedale di applicazioni e sistemi tecnologici per supportare il lavoro dell'equipe medica e fornire un migliore servizio ai pazienti. In quest'ottica nasce ad esempio il progetto TraumaTracker, presentato nel paragrafo 1.3.2, che segue il paziente nelle diverse fasi della gestione di un trauma. Uno di questi passaggi è quello del trattamento nella Shock Room, che quindi è da subito sembrata la candidata ideale per diventare un ambiente aumentato, data la criticità delle operazioni che in essa avvengono.

2.1 Il Progetto

Le possibilità messe in campo da uno Smart Environment, come visto in precedenza, possono essere svariate ed è quindi stato necessario circoscrivere il campo di azione ad alcune specifiche funzionalità, pensando al sistema in modo tale da renderlo il più possibile flessibile ed estensibile in un ottica di future integrazioni. Mantentendo la volontà di produrre un risultato utile si è voluto partire dai medici stessi e da quelle che sono le necessità di cui avrebbero avuto bisogno nella loro azione sul paziente in quello specifico ambiente. Il sistema infatti dovrà per definizione essere situato ed adattato alle necessità dell'ambiente fisico nel quale dovrà funzionare. In seguito ad un dialogo con un gruppo di medici interessati a tale progetto è emerso quali fossero gli aspetti più critici ed urgenti da risolvere, in cui l'applicazione del nuovo sistema avrebbe potuto portare i maggiori benefici e si è deciso di realizzare come prima istanza del progetto un sistema in grado di gestire la visualizzazione e la manipolazione di dati provenienti dalle varie fonti presenti, come ad esempio i diversi strumenti medici ed apparecchiature biomedicali presenti nella stanza, i dati diagnostici ed un insieme di eventi provenienti dall'ambiente stesso. Il sistema si occuperà

dell'integrazione e dell'elaborazione di tali dati al fine di visualizzarli nella forma più adatta in un display designato. Questo porterà auspicabilmente ad un miglioramento del trattamento stesso andando ad ottenere dati accurati nel momento in cui essi risultano necessari, sfruttando inoltre le capacità computazionali per ottenere dati aggregati in maniera autonoma. In questo modo il personale medico potrà spostare la propria attenzione sul processo di cura con la minor distrazione possibile ed avere al contempo accesso ad informazioni che allo stato attuale è difficile ottenere, facendo così risparmiare una grande quantità di tempo ed energie. In seguito ad un sopralluogo è infatti emerso che il problema principale allo stato attuale riguarda proprio l'accesso ai dati, per cui è prevista la presenza di diversi schermi e di svariati punti di accesso, con informazioni sparse in punti diversi della stanza o dell'ospedale stesso e non in grado di interagire tra di loro.

2.1.1 Scenari Applicativi

Dal dialogo con l'equipe del Trauma Team sono emersi diversi scenari ideali in cui il progetto Smart Shock Room può supportarli nella gestione degli eventi traumatici:

Supporto Preospedaliero

La gestione del trauma di un paziente inizia prima ancora che egli arrivi fisicamente all'interno dell'ospedale. Infatti nel momento in cui viene ricevuta una chiamata destinata alla Shock Room il team viene preallertato dalla centrale. Già in questa prima fase sarebbe utile poter visualizzare alcune informazioni relative al paziente man mano che esse vengono misurate o recuperate dalla squadra di soccorso sul posto ed allo stesso tempo sarebbe utile conoscere la posizione effettiva del mezzo in arrivo e una stima del tempo necessario in modo da potersi preparare in maniera efficace.

Gestione del Trauma

Una volta che il paziente arriva in ospedale e viene portato nella Shock Room iniziano le procedure di cura effettive ed alcune delle informazioni precedentemente visualizzate perdono di importanza mentre nuove necessitano di essere visualizzate. In questa fase l'equipe si trova a dover visualizzare i parametri misurati dagli strumenti presenti nella stanza, cambiando spesso da un dato all'altro a seconda della necessità del momento. Può inoltre essere utile conoscere alcune informazioni relative alla storia clinica del paziente.

Esami Diagnostici

Durante il processo di cura vengono spesso eseguiti esami diagnostici quali radiografie, effettuate in loco, o TAC, effettuate nell'apposita stanza. L'equipe medica vorrebbe visualizzare il risultato di tali esami direttamente nel display presente nella stanza in modo da coordinare le procedure di gestione del paziente in base al risultato degli esami.

Gestione degli Eventi

Per tutto il processo di gestione del trauma accadono svariati eventi e l'equipe vorrebbe che il sistema potesse gestire in automatico alcuni di essi, andando ad adattare i dati visualizzati in relazione all'evento stesso. Questo può voler dire visualizzare un certo parametro quando esso supera una certa soglia oppure visualizzare alcuni specifici parametri durante l'esecuzione di una certa procedura. Anche determinati eventi fisici, quali ad esempio l'utilizzo di un certo numero di sacche di sangue dall'emoteca devono produrre effetti sul sistema, come nel caso specifico quello di avvisare il centro trasfusionale.

2.2 Raccolta dei Requisiti

2.2.1 Requisiti Richiesti

I requisiti del sistema sono stati raccolti mediante interviste con diversi esponenti del Trauma Team dell'ospedale Bufalini, a cui è seguito un sopralluogo sul posto per prendere visione degli spazi e degli strumenti tecnologici disponibili. In questa fase è stato inoltre molto importante ricevere una descrizione dettagliata di come viene gestita al momento la Shock Room, in modo da evidenziare gli aspetti critici e far emergere diverse possibili funzionalità che possano essere gestite dal progetto Smart Shock Room.
Di seguito sono elencati i requisiti emersi, suddivisi per categorie.

1. REQUISITI GENERALI DI FUNZIONAMENTO

R1: Possibilità di visualizzare dati di varie sorgenti e dispositivi in una unica posizione

R2: Possibilità di scegliere quali dati visualizzare in qualsiasi momento

R3: Possibilità di mantenere dati fissi sullo schermo per tutta la durata delle operazioni

R4: Integrazione di vari dispositivi biomedicali differenti con una modalità di interazione comune

R5: Integrazione delle informazioni sugli eventi provenienti dal sistema e da TraumaTracker

R6: Possibilità di visualizzare dati clinici storici relativi al paziente

R7: Possibilità di vedere il flusso dei dati nel tempo e la loro variazione

R8: Possibilità di ricerca, filtraggio ed accesso alle informazioni in qualunque momento

R9: Possibilità di recuperare in ogni momento dati non più visualizzati

R10: Possibilità di replicare / condividere le informazioni in altri ambienti dell'ospedale

R11: Possibilità di contattare in automatico altri reparti in seguito ad eventi specifici

R12: Possibilità di interazione con il sistema in modalità Hands-free

2. SPECIFICHE OPERAZIONI RICHIESTE

O1: Possibilità di annotare in maniera vocale i farmaci somministrati

O2: Possibilità di annotare in maniera vocale le manovre effettuate

O3: Possibilità di chiedere in maniera vocale dati aggregati sul paziente (e.g. quantità di farmaco somministrata)

O4: Possibilità di monitorare e controllare il tempo globale dall'ingresso in Shock Room

O5: Possibilità di monitorare e controllare il tempo relativo a specifiche procedure tempo-dipendenti

O6: Possibilità di cambiare i dati visualizzati in automatico, senza bisogno di interazione esplicita, in seguito ad azioni ed eventi specifici :

- O6.1: Cambio di visualizzazione in base allo stato e alla posizione del paziente (e.g. diversa nel PreH, nella Shock Room, in fase di esami diagnostici, ecc.)
- O6.2: Cambio in seguito ad azioni specifiche (e.g. in fase di intubazione visualizzare saturazione e livello di CO_2)

O6.3: Cambio in seguito ad eventi ambientali specifici (e.g. allerta al centro trasfusionale dopo aver aperto la teca del sangue un certo numero di volte)

O7: Possibilità di ricevere feedback vocali sulle azioni automatiche effettuate

3. REQUISITI DI INTERFACCIA

I1: Possibilità di interazione hands-free con il sistema

I2: Interazione solo quando necessaria

I2.1: Sistema reattivo e sempre pronto

I2.2: Possibilità di avere dati visualizzati in automatico in seguito a specifiche azioni / eventi

I3: Possibilità di personalizzare l'interfaccia con cui vengono visualizzati i dati rispetto al singolo medico che dirige le operazioni

4. REQUISITI NON FUNZIONALI

Dal dialogo con il personale medico sono inoltre emersi come requisiti non funzionali:

N1: Necessità di interazione facile ed intuitiva

N2: Processo veloce e con il minimo overhead possibile.

In questo ambito infatti il tempo è la risorsa critica e il sistema deve dare un supporto permettendo al personale di lavorare senza perdere ulteriore tempo a causa del sistema e poter lavorare meglio sul paziente. È quindi necessario che il sistema semplifichi il processo complessivo.

2.2.2 Dati Richiesti

Durante il dialogo con l'equipe medica sono emerse anche una serie di dati ed informazioni che sono necessarie per la gestione del trauma e che al momento vengono utilizzate dal team sotto diverse forme. Esse rappresentano le informazioni che il sistema dovrà andare a gestire ed elaborare. Di seguito è riportato l'elenco, non esaustivo e completo, delle informazioni richieste:

1. Dati Biometrici del Paziente

- Parametri Vitali (anche pre-ospedalieri):
  - Saturazione di ossigeno nel sangue (SpO_2)
  - Pressione Arteriosa
  - Frequenza Cardiaca
  - Temperatura
- Dati Biometrici:
  - Livello CO_2
  - Emogasanalisi (EGA)
  - Tromboelastometria Rotazionale (ROTEM)
- Dati Diagnostici:
  - Immagini radiografiche
  - TAC
  - Referti di esami

2. Dati Temporali

- Tempo dall'arrivo nella stanza
- Tempo dall'inizio di una procedura tempo-dipendente
- Report Overview e dati scelti di TraumaTracker

3. Dati Ambientali

- Posizione dell'ambulanza / elicottero in arrivo
  - Stima del tempo di arrivo
  - Avviso quando si trova in prossimità
- Numero di sacche di sangue disponibili ed utilizzate

2.2.3 Aspetti Critici

Durante la raccolta dei requisiti sono emerse alcune problematiche rilevanti di cui la progettazione del sistema dovrà tenere conto:

- Essendo il sistema situato dovrà tenere conto della configurazione fisica della stanza in cui avverrà il deploy. Allo stato attuale è già presente uno schermo in grado di supportare la visualizzazione delle informazioni, che tuttavia si trova in una posizione sfavorevole essendo dietro al paziente e coperto da altri strumenti che ostruiscono parzialmente la visuale ed è inoltre di dimensioni abbastanza ridotte.
  Per questo sarà necessario valutare come visualizzare i dati in maniera efficiente e valutare quali dati visualizzare contemporaneamente. Si suggerisce inoltre nel lungo periodo di spostare lo schermo in una posizione più consona ed utilizzare uno schermo più ampio per facilitare la visualizzazione.

- Ogni dispositivo diagnostico e biomedicale presente nella stanza, nonché i sistemi di gestione delle informazioni relative al paziente sono gestite in maniera differente e presentano una modalità di acceso ai dati non uniforme. Inoltre non è possibile nella maggior parte dei casi agire direttamente su di essi in quanto rappresentano sistemi proprietari.
  Ogni sorgente andrà quindi gestita separatamente e specificamente per poter essere integrata nel sistema.

2.3 Analisi dei Requisiti

Dai requisiti raccolti emerge l'idea di un sistema fortemente situato e reattivo, capace di adattarsi alle necessità degli utenti e agli eventi che possono avvenire durante il processo di cura del trauma del paziente. Per questo sarà importante che possa rispondere alle richieste dell'equipe del Trauma Team con un certo grado di autonomia. Le richieste che arriveranno al sistema saranno di diverse tipologie e potranno facilmente evolvere nel tempo, per cui sarà richiesto un certo grado di estensibilità.

Il sistema si troverà ad interagire con macchinari differenti e diverse tecnologie, eventualmente posizionati in diversi punti dell'ospedale.

Inoltre dalle richieste emerge la necessità di interagire con il sistema contemporaneamente con l'azione del Trauma Team sul paziente, per cui sarà auspicabile avere la possibilità di interagire con il sistema in maniera multimodale, in modo da permettere di interagire a seconda della modalità preferita a seconda del momento e della situazione corrente. Sarà inoltre estremamente

importante che il sistema offra la possibilità di una interazione *hands-free*, fondamentale in scenari di lavoro come questi in cui il personale medico si trova occupato in altre mansioni che richiedono la loro attenzione e l'utilizzo delle mani. Tale modalità dovrà consentire una interazione rapida ed il più possibile semplice e facile da apprendere per l'equipe, per cui l'interfaccia relativa andrà studiata nel dettaglio.

2.3.1 Casi D'Uso

Andando ad analizzare i principali requisiti funzionali del sistema emergono alcuni specifici casi d'uso relativi al sistema:

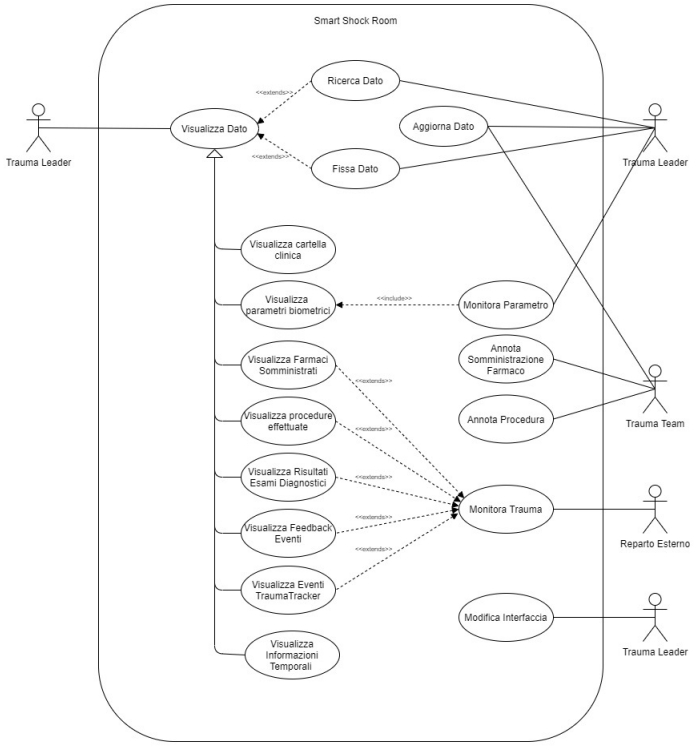

Figura 2.1: Diagramma dei casi d'uso

A partire dai casi d'uso si possono osservare differenti scenari di utilizzo del sistema, a seconda di come l'utente interagisce con esso.

C1: *Visualizza Dato* fa riferimento ai requisiti R1 e R2 e rappresenta il caso generale e più comune di utilizzo del sistema, in cui l'utente richiede al

sistema un qualsiasi dato affinché esso venga visualizzato in output. Da esso discendono per generalizzazione le varie tipologie di dati che l'utente può richiedere:

C1.1: *Visualizza Cartella Clinica* fa riferimento a R6, ossia alla possibilità di visualizzare i dati clinici storici del paziente attraverso l'accesso ai database ospedalieri, che risultano essere utili in alcuni frangenti come ad esempio la scelta di un medicinale in base alle eventuali allergie del paziente.

C1.2: *Visualizza Parametri Biometrici* fa riferimento a R4 e rappresenta una delle funzionalità principali del sistema, ossia la possibilità di richiedere e visualizzare parametri misurati dai vari strumenti presenti nella stanza. In questo modo l'equipe può scegliere cosa visualizzare e vederlo direttamente sul display centrale senza dover controllare sullo specifico dispositivo ogni volta. Un esempio di questa attività può essere la visualizzazione del valore attuale del battito cardiaco del paziente.

C1.3 e 1.4: *Visualizza Farmaci e Procedure* fanno riferimento, insieme ai casi *C6* e *C7*, ai requisiti O1, O2 e O3, relativi ad azioni specifiche eseguite sul trauma di cui risulta utile tenere traccia ed avere informazioni a riguardo. L'equipe medica ha infatti fatto presente l'importanza di conoscere la quantità di un certo farmaco somministrato e allo stesso tempo quanto sia estremamente importante poter ricostruire a posteriori i farmaci somministrati e le procedure effettuate in una fase di ricostruzione dell'intervento. Questo aspetto è in parte gestito anche dal sistema TraumaTracker.

C1.5: *Visualizza Risultati Esami Clinici* fa riferimento a R4 e risulta particolarmente importante in quanto questa operazione è quella che al momento fa perdere la maggiore quantità di tempo poichè richiede di appuntare codici identificativi spesso diversi per ogni esame effettuato e la ricerca manuale nei diversi archivi. Possono essere ad esempio visualizzati gli esiti degli esami radiografici o della TAC.

C1.6 e 1.7: *Visualizza Eventi e TraumaTracker* fanno entrambi riferimento a R5 e si differenziano nell'origine degli eventi mostrati. *C1.6* infatti è legato agli eventi ambientali della stanza o generati dal sistema stesso, mentre *C1.7* è specifico rispetto alle informazioni fornite dal sistema TraumaTracker con cui il sistema della Shock Room si dovrà integrare. Questi aspetti risultano importanti in quanto permettono all'equipe di essere a conoscenza di specifici eventi che

possono risultare particolarmente importanti e che richiedono un intervento specifico.

C1.8: *Visualizza Informazioni Temporali* fa riferimento a O4 e O5 e alla necessità di monitorare le tempistiche durante le operazioni. L'equipe medica ha infatti descritto come in alcune procedure specifiche le tempistiche siano fondamentali per la buona riuscita del trattamento e la possibilità di monitorare e ricevere avvisi relativi al tempo di esecuzione di una certa procedura può essere un aiuto importante.

C2: *Ricerca Dato* fa riferimento ai requisiti R8 e R9 e risponde alle necessità dell'utente ricercare e filtrare i dati a seconda delle necessità specifiche. Infatti non sarà possibile tenere ogni informazione sullo schermo ed ottenere solo le informazioni più rilevanti sarà fondamentale. Un esempio di questa attività può essere la visualizzazione del valore massimo del battito cardiaco del paziente.

C3: *Fissa Dato* fa riferimento a R3 e alla possibilità di scegliere se tenere un certo dato fisso sul display in modo da visualizzare sempre il valore aggiornato di tale informazione o di visualizzarlo solo temporaneamente. Sarà chiaramente possibile modificare questa scelta e rimuovere un dato anche dopo averlo fissato.

C4: *Aggiorna Dato* fa riferimento a O6 ed ha una duplice valenza. Infatti gli utenti possono andare a modificare alcune informazioni manualmente ma anche il sistema stesso è in grado di aggiornare automaticamente i valori mostrati per permettere un monitoraggio più dettagliato.

C5: *Monitora Parametro* fa riferimento a R7 e permette di seguire lo sviluppo temporale di uno specifico parametro, eventualmente ricevendo avvisi in caso venga raggiunta una certa soglia.

C6 e 7: *Annota Farmaco e Procedura* come detto fanno riferimento rispettivamente a O1 e O2 e riguardano la possibilità di salvare nel sistema informazioni relative ad operazioni effettuate sul paziente che richiedono di essere tracciate e monitorate.

C8: *Monitora Trauma* fa riferimento a R10 e R11, ossia alla duplice possibilità di monitorare alcune informazioni sul trauma in corso in altri reparti dell'ospedale che possono essere interessati all'andamento e anche alla possibilità di notificare e avvisare un reparto esterno in seguito ad alcuni eventi specifici in maniera completamente automatica. Questo in particolare può voler dire visualizzare tutti o solo alcuni dei dati presenti nel sistema e a cui fanno riferimento i vari sottocasi di C1.

C9: *Modifica Interfaccia* fa riferimento a O3, O6 e I3. Anche questa operazione è duplice:

- Il Trauma Leader può scegliere quali dati mostrare e come modificare a piacere la propria interfaccia del display. Questa operazione di solito avviene a freddo, non durante la fase di emergenza, in quanto non pregiudica la possibilità di mostrare qualunque dato ma piuttosto l'ordine e la posizione di default in cui questi dati vengono mostrati.
- Il sistema stesso può modificare l'interfaccia del display in base ad eventi specifici oppure in base alla fase di gestione del trauma. Per esempio infatti la visualizzazione dovrà essere differente durante la fase preospedaliera (PreH) e durante la gestione vera e propria del trauma.

2.3.2 Modello del Dominio

Tramite l'analisi dei requisiti e relativi casi d'uso è stato possibile modellare in maniera chiara il dominio del sistema, usando una modalità Domain-driven, in modo da avere una migliore comprensione del problema generale. In tale modello emergono le varie entità presenti e come esse interagiscono tra loro in modo da poter descrivere il processo complessivo. Per fare questo è stato utilizzato il formalismo offerto da UML ed in particolare dal diagramma delle classi.

Osservando i requisiti ed i casi d'uso è possibile ottenere una prima rappresentazione intuitiva del dominio, che è possibile utilizzare come un primo modello del dominio, con un elevato livello di astrazione.

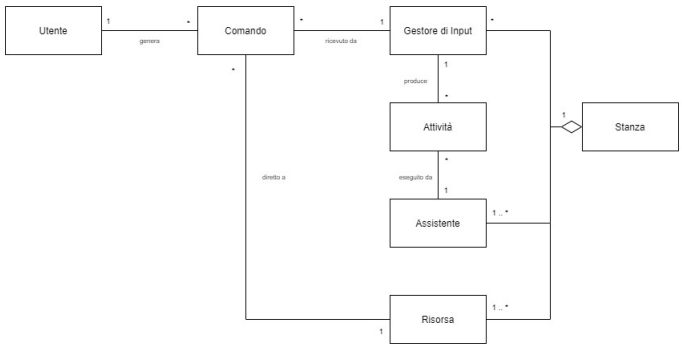

Figura 2.2: Rappresentazione del dominio ad alto livello

2.3.2.1 Entità

Da tale rappresentazione emergono le principali entità coinvolte nel sistema, tuttavia è necessario procedere ad una analisi più approfondita per specificare le caratteristiche delle singole entità. Infatti alcune di esse possono essere specializzate in diverse tipologie specifiche che emergono dall'analisi dei requisiti forniti. A seguire ogni entità verrà descritta nel dettaglio e per ognuna di esse verranno presentati alcuni esempi emersi in maniera specifica dai requisiti presentati dal Trauma Team:

- *Utente*
- *Comando*
- *Gestore di input*
- *Attività*
- *Stanza*:
  - *Assistente*
  - *Risorsa*

Utente

La prima categoria di entità presenti nel dominio è relativa agli utenti del sistema. Essi rappresentano i membri del team che lavorano alla gestione del trauma e che utilizzano le funzionalità del sistema per svolgere il proprio lavoro e trarre beneficio dallo strumento tecnologico. In base allo stato attuale ed ai requisiti raccolti emergono diversi ruoli all'interno del Trauma Team e a seconda del ruolo ricoperto dal membro le funzionalità richieste al sistema possono essere diverse. Per questo posso ulteriormente suddividere queste entità a seconda del ruolo che essa rappresenta:

- *Trauma Leader*: È il responsabile della gestione del trauma in corso, svolge un ruolo di coordinazione e direzione dell'intero Trauma Team ed è anche colui che si occupa della gestione del sistema Trauma Tracker, attualmente utilizzato all'interno della procedura di gestione dei traumi nella struttura in esame. Egli si occuperà specificamente della richiesta delle informazioni da visualizzare sullo schermo, anche se ha comunque la possibilità di aggiungere informazioni al sistema. Le principali azioni che si troverà a richiedere al sistema sono le seguenti:
  - Richiedere informazioni su eventi relativi al trauma in corso

  - Richiedere informazioni sulla storia clinica del paziente
  - Richiedere valori misurati al momento sul paziente
  - Richiedere informazioni ambientali alla stanza

- *Membro del Trauma Team*: Membro attivo del team che si occupa della gestione del trauma, e come tale ha accesso a tutte le informazioni relative ad esso. A differenza del Trauma Leader la sua interazione con il sistema è più legata all'aggiunta di informazioni, anche se al bisogno può chiedere la visualizzazione di dati specifici. Le principali azioni che si troverà a richiedere al sistema sono le seguenti:

  - Registrare eventi nel trauma in corso
  - Registrare la somministrazione di farmaci
  - Registrare le procedure effettuate

- *Membro Esterno*: Membro esterno al Trauma Team che comunque è interessato al monitoraggio del trauma in corso. Potrebbe anche trattarsi di qualcuno che è fisicamente in un altro luogo dell'ospedale ma che necessita di ricevere aggiornamenti o informazioni riguardo al trattamento del paziente. Esso può ovviamente solo visualizzare informazioni senza poter inserire nuovi dati e senza interferire con la visualizzazione nella Shock Room. Un esempio di questa categoria di entità potrebbe essere il reparto di terapia intensiva, il quale può avere interessa nel monitorare lo stato del paziente in cura.

Comando

L'intenzione dell'utente è rappresentata dall'entità Comando. In base alle richieste effettuate, espresse esplicitamente dal requisito R12, dovranno essere gestite almeno due diverse tipologie di comandi, quelli impartiti tramite una comune interfaccia grafica e quelli effettuati tramite interazione vocale. Inoltre esistono diverse tipologie di comandi che l'utente può impartire a seconda del tipo di operazione che egli intende richiedere al sistema:

A - *Comandi di Visualizzazione*
Tali comandi possono essere a loro volta suddivisi in *visualizzazione semplice*, in riferimento al caso d'uso C1 e *visualizzazione Aggregata*, in riferimento al caso d'uso C2. Essi rappresentano l'utilizzo principale del sistema e quindi la maggior parte di comandi impartiti al sistema rientreranno in questa categoria. Il loro scopo ultimo è quello di ottenere la visualizzazione di una certa informazione come output.

B - *Comandi di Annotazione*
Tali comandi si riferiscono ai casi d'uso C6 e C7. Sono comandi che hanno lo scopo di andare ad aggiungere un qualche tipo di informazione al sistema

C - *Comandi di Monitoraggio*
Tali comandi fanno riferimento ai casi d'uso C5 e C8. Sono comandi che hanno lo scopo di richiedere al sistema il controllo di un parametro o di un evento, in modo da poter ricevere una notifica o lo svolgimento di un'azione automatica quando esso raggiunge un valore di soglia o si verifica.

D - *Comandi di Azione*
Tali comandi fanno riferimento ai casi d'uso C3 e C9. Sono comandi che hanno l'obiettivo di modificare lo stato di un'entità o dell'ambiente stesso

Ogni comando inoltre farà riferimento ad una specifica risorsa come bersaglio, sulla quale dovrà avvenire l'effetto desiderato.

Gestore di Input

Gli utenti necessiteranno di entità specifiche per interagire con il sistema, tramite i quali i comandi da esso generati possono essere direzionati alla specifica entità che si occuperà della loro esecuzione. È stato utile specificare la presenza di queste entità in quanto dai requisiti emergono diverse tipologie di gestori di input che sarà necessario gestire e che non sono esattamente intercambiabili tra loro. Infatti alcuni permetteranno di gestire tutti gli aspetti del sistema mentre altri saranno specializzati su alcune parti specifiche:

- *Gestore di Input Ambientale*: Entità specializzata sull'ambiente e sulla stanza stessa, permette di gestire l'interazione con il sistema per aspetti ad essi legati.

- *Gestore di Input del Trauma*: Entità tramite la quale l'utente può interagire con la parte relativa alla gestione del trauma e al sistema TraumaTracker.

- *Gestore di input multiplo*: Entità che permette l'interazione con tutte le componenti del sistema.

Attività

I gestori degli input hanno il compito di acquisire ed elaborare i comandi degli utenti in modo che essi possano essere effettivamente eseguiti dalle entità relative alla stanza di cui sono bersaglio. Quindi quello che produrranno è una lista di attività che rimarranno in attesa di essere eseguite. Non è tuttavia necessaria una rappresentazione esplicita di queste entità in quanto derivano direttamente dai comandi, i quali possono essere usati direttamente per richiedere una certa attività al sistema. Inoltre l'utilizzo esplicito può creare confusione ed ambiguità con le attività effettive eseguite dal sistema.

Stanza

I comandi prodotti dagli utenti hanno come destinatario una stanza che si occuperà dell'esecuzione degli stessi, attuando l'intenzione dell'utente. Tale entità rappresenta l'ambiente della Shock Room, e contiene a sua volta una serie di entità che rappresentano la parte centrale del sistema stesso. Dato il modello in esame è possibile che siano presenti più stanze, ognuna con le sue entità e i suoi assistenti, rendendo di fatto possibile una estensione del sistema e delle sue funzionalità. I componenti principali sono:

1 - Assistenti

Tali entità rappresentano la parte del sistema che si occupa dell'effettiva esecuzione dei comandi richiesti dagli utenti, facendo allo stesso tempo da mediatore tra le diverse entità del sistema. Infatti per poter completare le richieste degli utenti sarà necessario interagire con diverse risorse e sorgenti di informazioni, le quali tuttavia non interagiscono mai tra di loro direttamente. Esistono due tipologie di Assistente presenti nell'ambito della Shock Room:

Room Assistant

Room Assistant rappresenta la stanza stessa ed il suo sistema di coordinazione e gestione dei comandi. Tramite esso l'utente può interagire con le risorse presenti nell'ambiente. Dai requisiti e dai casi d'uso emergono quali dovranno essere le attività che questa entità dovrà essere in grado di supportare e svolgere:

- Attività di gestione del trauma
  - Gestione eventi temporali (*in riferimento al caso d'uso C1.8*)
- Attività di gestione della diagnostica

  - visualizzazione parametri biometrici (*rif. casi d'uso C1.2 e C2*)
  - visualizzazione esami diagnostici (*rif. caso d'uso C1.5*)
- Attività di gestione dell'ambiente
  - Visualizzazione feedback di eventi ambientali (*rif. caso d'uso C1.7*)
  - gestione operazioni su risorse ambientali
  - gestione operazioni su risorse cliniche (*rif. requisito R4*)
- Attività di gestione del sistema
  - Inizio intervento
  - Cambio di interfaccia del display (*rif. caso d'uso C9*)
  - Elenco delle attività disponibili
  - Elenco dei dati disponibili
  - Chiusura intervento

Trauma Assistant

Trauma Assistant è un'entità che si occupa della gestione del trauma in corso e le attività relative ad esso, similmente a come Room Assistant si occupa delle attività relative all'ambiente. Dai requisiti e dai casi d'uso emergono quali dovranno essere le attività che questa entità dovrà essere in grado di supportare e svolgere:

- Attività di gestione del paziente
  - visualizzazione cartella clinica (*rif. caso d'uso C1.1*)
  - gestione dati anagrafici ed identificativi (*rif. casi d'uso C1.1 e C1.5*)
- Attività di gestione del trauma
  - gestione procedure effettuate (*rif. casi d'uso C1.3 e C6*)
  - gestione farmaci somministrati (*rif. casi d'uso C1.4 e C7*)
  - gestione eventi TraumaTracker (*rif. caso d'uso C1.6*)
- Attività di gestione del sistema
  - Inizio intervento
  - Elenco delle attività disponibili
  - Elenco dei dati disponibili

  - Chiusura intervento

Essendo presente questa entità risulta ovvio inserire nel dominio del sistema un'entità che rappresenti in modo esplicito il trauma stesso con cui questa entità può andare ad interagire al fine di fornire informazioni relative allo stesso e ai relativi eventi.

2 - Risorse

Nella stanza sono presenti alcune risorse che contengono le informazioni necessarie per completare i comandi impartiti dagli utenti. In seguito ad una approfondita analisi sono emerse due diverse categorie di entità di questo tipo:

- **Sorgenti di Informazioni**: Le sorgenti di informazioni rappresentano le entità che contengono o misurano i dati che possono essere richiesti dall'equipe medica. Su di esse non è possibile effettuare nessuna operazione tramite il sistema se non quella di ottenere dati ed informazioni. Esistono diverse sotto categorie che fanno parte di questa tipologia di entità, con la caratteristica comune di essere entità in sola lettura. In particolare possiamo evidenziare due tipologie principali:
  - *Sorgente Clinica*: Entità che rappresentano i vari dispositivi di misurazione biometrici e clinici che sono in grado di fornire la lettura di un determinato parametro o di una certa informazione. Un esempio di questa entità potrebbe essere quello relativo al dispositivo per la misurazione della saturazione di ossigeno nel sangue.
  - *Sorgente Ambientale*: Entità che rappresentano i vari dispositivi Smart e IoT presenti nella stanza in grado di fornire informazioni rispetto al proprio stato. Un esempio di un'entità di questo genere è quella relativa all'emoteca, il cui dispositivo fisico è equipaggiato con un sensore in grado di fornire informazioni riguardo il numero di sacche di sangue prelevate ed utilizzate.
- **Risorse**: Le risorse sono entità simili alle sorgenti di risorse, infatti come esse il loro obiettivo è quello di fornire informazioni in esse contenute o da esse misurate. Tuttavia questa categoria di entità permette di svolgere operazioni aggiuntive su di esse, oltre alla semplice lettura dei dati, con cui è possibile modificare il loro stato. Anche in questo caso sono presenti alcune sotto categorie :
  - *Risorsa Clinica*: Entità che fanno riferimento ad un qualunque strumento biomedicale o clinico che permette la modifica o il settaggio oltre che la lettura del valore misurato.

- *Risorsa Ambientale*: Entità che si riferiscono ai vari dispositivi Smart e IoT presenti nella stanza che possono essere controllati da remoto attraverso il sistema. Un banale esempio di questa tipologia di entità potrebbe essere la rappresentazione di una lampadina smart che permette di controllare l'illuminazione dell'ambiente tramite il sistema.
- *Display*: Entità che si occuperà della visualizzazione del risultato prodotto in output dall'elaborazione della richiesta dell'utente. Potranno essere presenti una o più di queste entità, eventualmente anche in luoghi fisici diversi dalla Shock Room stessa.

Per evitare ambiguità tra i due termini chiameremo ***Ambiente*** il caso più generale, e intendendo invece per ***Risorse*** questa ultima categoria di entità.

3 - Paziente

Tale entità rappresenta il paziente fisico che è in cura nella Shock Room ed è in grado di fornire informazioni relative ad esso ed alla sua identità.

Modello del Dominio

Rappresentando esplicitamente queste entità si ottiene il modello del dominio dettagliato, rappresentato in figura 2.3

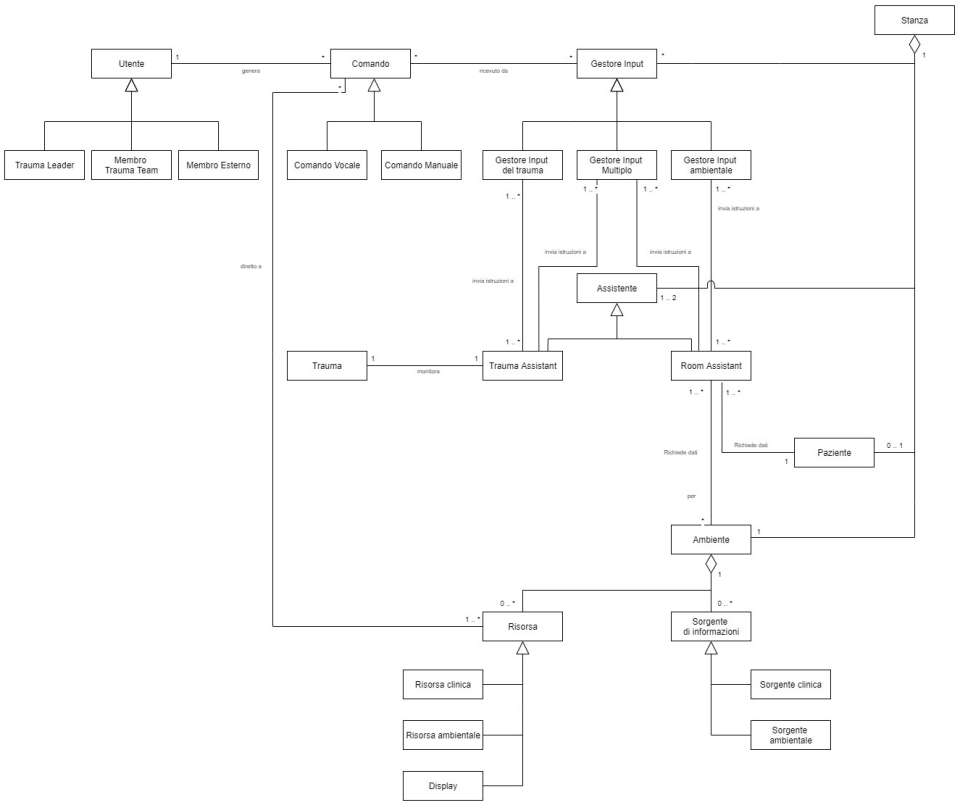

Figura 2.3: Modello del dominio

2.3.2.2 Modalità di Interazione

La parte che emerge meno chiaramente dal diagramma delle classi è quella relativa alle modalità di interazione tra le diverse entità presenti nel sistema, per cui è opportuno andare ad approfondire questo aspetto che risulta fondamentale nella modellazione di un sistema di questo genere.

Interazioni tra Entità

Andando ad analizzare il modello ci sono tre elementi particolari che rappresentano le interazioni tra diversi componenti del sistema:

I1: *L'entità Comando*: media la comunicazione tra Utenti e Gestori di Input.

I2: *Le richieste di attività*: mediano la comunicazione tra Gestori di Input e Assistenti.

I3: *Le richieste di informazioni*: mediano la comunicazione tra gli Assistenti, il trauma, l'ambiente e il paziente.

Mantenendo il formalismo proposto da UML verranno mostrate, tramite Activity Diagram e Sequence Diagram, le interazioni complessive tra le entità del sistema in relazione ai principali comandi che l'utente può richiedere al sistema.

Come descritto sopra, nel paragrafo 2.3.2.1 nella parte relativa ai comandi, esistono quattro tipologie principali di attività che l'utente può richiedere. Per ognuna di esse verrà presentato il relativo diagramma:

Comando A1: Visualizzazione di Informazioni Semplice

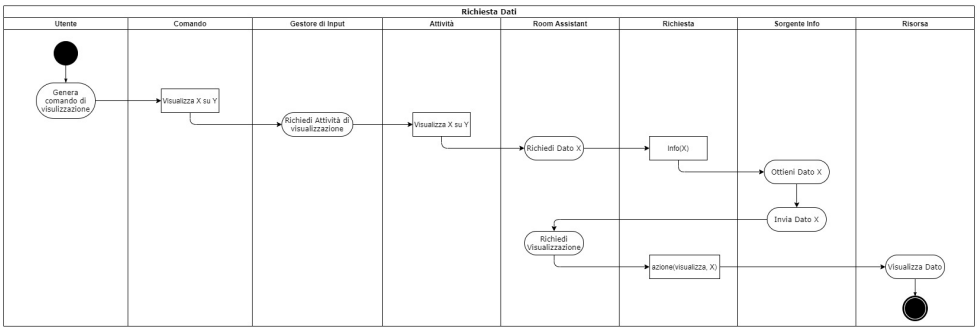

Figura 2.4: Visualizzazione di informazioni semplice

Il diagramma in figura 2.4 rappresenta il caso più generale e l'attività principale per il sistema. Essa infatti rappresenta il suo funzionamento base. Quello che emerge già da questo diagramma è il ruolo di mediatore che svolge Room Assistant tra le richieste dell'utente e l'accesso alle risorse e alle informazioni. Risulta infatti evidente come l'utente non comunichi direttamente con le risorse, ed anzi potrebbe anche non conoscere quali di esse siano disponibili, ed è l'Assistente che si occupa della comunicazione con esse. Alcuni esempi di questo comando possono essere la visualizzazione del valore della saturazione del paziente, del suo battito cardiaco o dei dati relativi alla sua cartella clinica.

Esempio: Visualizzazione del valore della saturazione del paziente
Per entrare più nel dettaglio viene mostrato nella figura sottostante il processo dettagliato relativo ad un esempio di applicazione di tale attività tramite il formalismo dei diagrammi di sequenza. Lo scenario dell'esempio è il seguente: *Il Trauma Leader vuole conoscere il valore attuale della saturazione dell'ossigeno nel paziente attualmente in cura. Per farlo invierà il comando tramite uno dei dispositivi di input. L'entità relativa ad esso convertirà il suo comando in una richiesta rivolta al Room Assistant, il quale prima andrà a recuperare il valore della saturazione chiedendolo direttamente all'entità relativa al dispositivo che la misura e successivamente richiedendo la visualizzazione di tale valore sul display della Shock Room.*

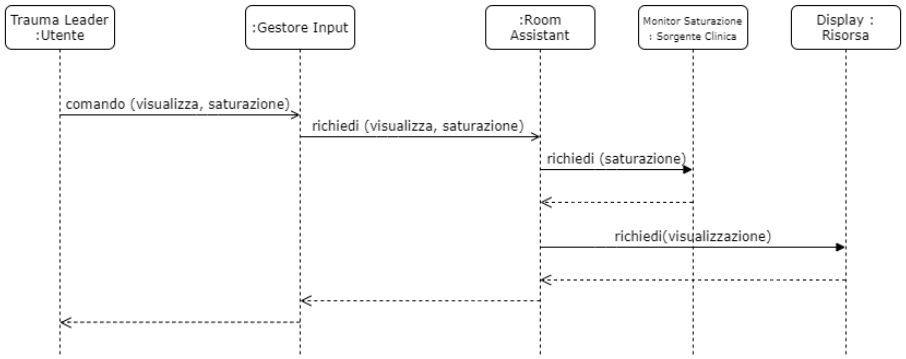

Figura 2.5: Visualizzazione del parametro 'saturazione'

Comando A2: Visualizzazione di Informazioni Aggregate
Una variante della attività precedente riguarda la richiesta di informazioni aggregate, che sono il risultato di una specifica ricerca da parte dell'utente. Tale operazione di aggregazione dei dati avviene in parte per opera della risorsa stessa ed in parte ad opera dell'assistente che ne ha richiesto il valore, a seconda della tipologia di dato richiesto.

Comando B: Annotazione di informazioni

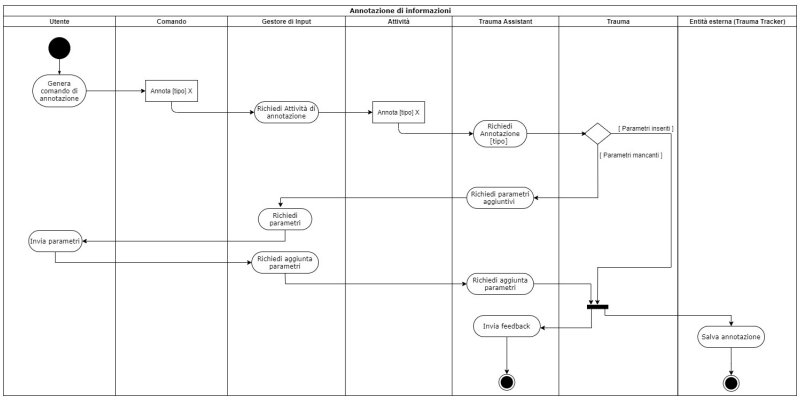

Figura 2.6: Annotazione di informazioni

Un'altra attività comune nell'utilizzo del sistema è quella relativa all'annotazione di informazioni relative al trauma in corso, che possono essere farmaci somministrati o operazioni effettuate sul paziente. In questo caso si tratta quindi di una operazione di scrittura di informazioni rispetto alla sola lettura del caso precedente. Come in esso tuttavia risulta evidente il ruolo di mediatore che svolge in questa operazione l'entità Trauma Assistant, la quale si andrà ad interfacciare con l'entità relativa al trauma ed attraverso di essa con l'entità esterna che gestisce il registro degli eventi relativi al trauma tramite il sistema TraumaTracker.

Comando C: Monitoraggio

Un'ulteriore attività che l'utente può richiedere al sistema è quella di monitoraggio dei dati e dei parametri del paziente. In base ai requisiti forniti esistono due tipologie di operazioni relative a questa attività: Nel primo caso l'operazione richiesta è molto simile alla semplice visualizzazione ma richiede di rinnovare i valori mostrati ad intervalli periodici. Tale operazione è descritta nel dettaglio in figura 2.7. In alternativa è possibile tramite questa tipologia di attività ricevere una notifica quando il parametro in questione raggiunge una certa soglia. Tale operazione infatti è molto simile alla precedente, con la sola differenza di non aggiornare la visualizzazione periodicamente ma solo al determinarsi di uno specifico evento.

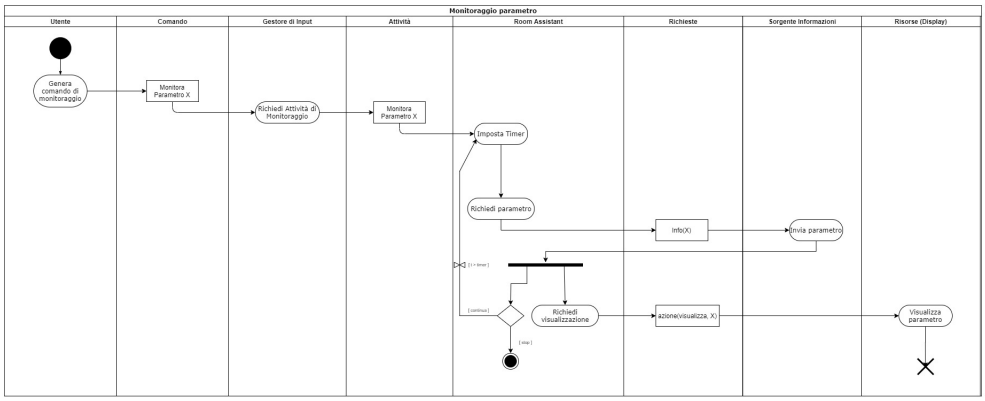

Figura 2.7: Attività di monitoraggio

Comando D: Azioni sul Sistema

Infine l'ultima tipologia di attività che l'utente può richiedere al sistema è quella relativa alle azioni sullo stesso. Esse producono come risultato una modifica nello stato delle entità o dell'ambiente stesso. La più comune di queste operazioni riguarda la modifica dell'interfaccia del Display.

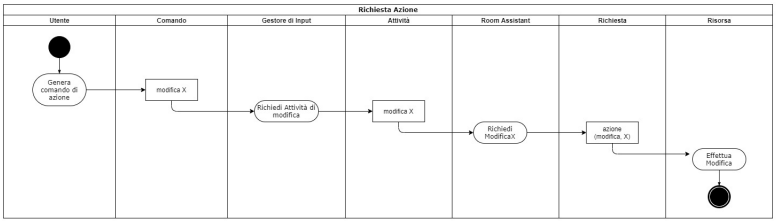

Figura 2.8: Attività azione sul sistema

Capitolo 3

Verso un'Architettura Integrata

Come già detto in precedenza l'obiettivo di questo lavoro di tesi non è solamente lo sviluppo di un sistema che risponda ad una richiesta ed implementi una lista di requisiti, ma vuole anche andare ad esplorare le possibilità e i vantaggi portati dall'applicazione di un insieme di approcci tecnologici e concettuali innovativi alla modellazione (e successivamente all'implementazione) del sistema stesso. Per cui per poter sviluppare l'architettura del progetto Smart Shock Room è necessario prima riflettere sull'integrazione dei paradigmi che la andranno a comporre.

3.1 Modello Architetturale

Esistono in letteratura svariati modelli e paradigmi adatti allo sviluppo di un sistema pervasivo con le caratteristiche del progetto in esame, ognuno dei quali presenta inevitabilmente punti di forza e debolezze. In questo progetto di tesi si è scelto di non analizzare il modello per selezionare tra le alternative disponibili gli approcci migliori in termini di efficienza e diffusione, ma piuttosto di scegliere alcuni degli approcci ad oggi più innovativi e promettenti a livello di ricerca, in modo da valutarne l'applicazione e l'integrazione, per potere così valutare il processo di design e sviluppo tramite essi. Per questo motivo la scelta è ricaduta sul paradigma ad agenti, ed in particolare sui sistemi multiagente (*MAS*) insieme con il modello dei Digital Twin. Ad essi si affianca inoltre la gestione degli input tramite un assistente vocale, ritenuta utile ed adatta al tipo di applicazione richiesta per il sistema, nonché risposta ad una specifica richiesta espressa nei requisiti dal team committente.

Inoltre, nonostante in letteratura esistono svariati esempi di applicazione di questi approcci, è difficile trovarne uno in cui siano applicati insieme, in sinergia, per lo sviluppo di un unico sistema, in particolare nell'ambito dell'healthcare. Per questo la scelta risulta particolarmente interessante in modo

da poter valutare l'efficacia di tale approccio per la modellazione e lo sviluppo di sistemi di questo tipo.

L'idea è quindi quella di sviluppare un'architettura capace di rappresentare in modo uniforme l'intero sistema della Smart Shock Room, in cui gli agenti che abitano un MAS sono in grado di interagire direttamente con le entità fisiche presenti nell'ambiente tramite la loro rappresentazione digitale, il loro Digital Twin. In questo modo gli agenti potranno interagire con il mondo fisico e viceversa. Ovviamente non esiste una integrazione nativa tra i due approcci, e per rendere possibile l'interazione si è dimostrata particolarmente adatta, anche a livello concettuale, l'idea di artefatto, espressa nel modello Agent & Artifact.

Visto il modello del dominio del sistema la scelta di tali modelli risulta particolarmente adatta per diverse motivazioni:

- Alcune delle entità presenti nel dominio sono la rappresentazione di elementi dell'ambiente fisico della Shock Room, per cui risulta particolarmente adatta la loro rappresentazione secondo il modello Digital Twin. In questo modo si ottiene un gap minimo tra il modello e l'architettura e il processo di mappatura risulta quasi immediata.

- Il modello presenta anche entità con un comportamento intrinsecamente reattivo, mentre alcuni dei requisiti richiedono che le stesse possano agire con un certo grado di proattività. Questa particolarità rende ottima la scelta del paradigma ad agenti.

La figura 3.1 mostra in maniera informale l'idea del sistema Smart Shock Room che emerge da un'organizzazione architetturale di questo tipo:

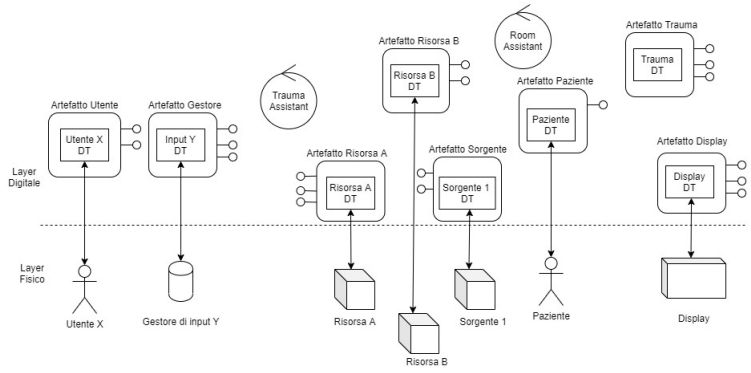

Figura 3.1: Smart Shock Room con Agenti e Digital Twin

3.2 Il Paradigma ad Agenti

Il paradigma ad agenti rappresenta una moderna alternativa al classico paradigma di programmazione Object-Oriented ed esprime il massimo della sua potenzialità nella rappresentazione di entità attive.

3.2.1 L'Astrazione di Agente

Per definizione un agente è un'entità computazionale proattiva ed autonoma. Etimologicamente il termine stesso agente deriva dal latino *agens*, colui che agisce, ed è indicativo di come tale entità possa effettuare azioni senza bisogno dell'intervento di una entità esterna che lo richieda. Tale proprietà prende il nome di *proattività*. L'autonomia può invece essere definita in termini computazionali come l'incapsulamento del flusso di controllo all'interno dell'entità stessa. Conseguenza di questa definizione è che l'agente non avrà interfacce in quanto non può essere controllato o invocato da entità esterne. Tuttavia l'agente si troverà ad interagire con altre entità nel suo ciclo di vita. Infatti egli vive all'interno di un ambiente sul quale esso può agire e che spesso è condiviso con altre entità.

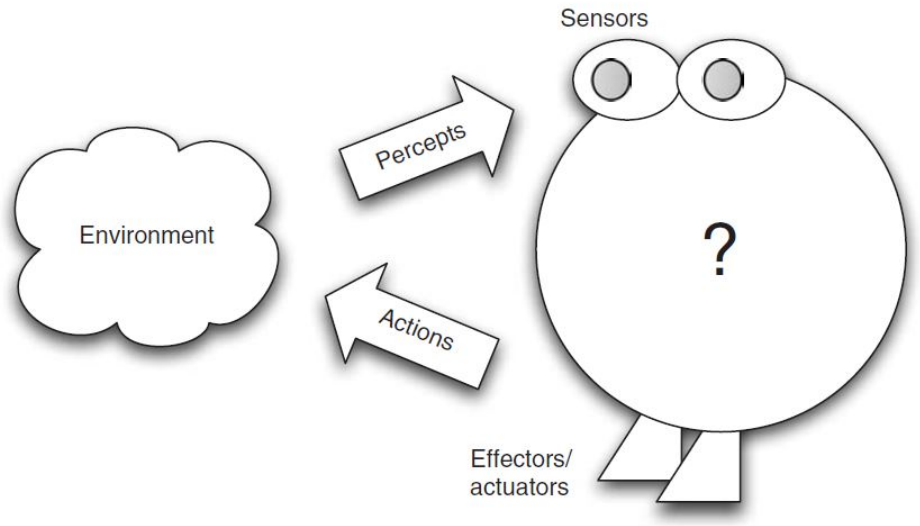

Figura 3.2: Rappresentazione di un agente, tratto da [8]

Da questa proprietà fondamentale che è l'autonomia derivano le altre proprietà tipiche degli agenti:

- *Situatedness*: L'agente vive all'interno di un ambiente con cui interagisce e che condiziona le sue azioni in base alle proprie caratteristiche specifiche. Inoltre le azioni dell'agente sono in grado di provocare cambiamenti nello stato dell'ambiente stesso.

- *Reattività ai cambiamenti*: Essendo il comportamento dell'agente legato all'ambiente è necessario che esso possa percepire i cambiamenti che avvengono nello stesso e che possa adattarsi ad essi.

- *Capacità sociali*: L'ambiente in cui vive un agente è nella maggior parte dei casi condiviso con altri agenti con il quale l'agente può comunicare scambiando informazioni e conoscenza.

Nella figura 3.3, tratta da [23], viene mostrata l'organizzazione di un tipico sistema multiagente in cui un certo numero di agenti, organizzati in gruppi (denominati *società*), vivono all'interno di un ambiente condiviso in cui ogni agente ha una specifica sfera di influenza, ovvero una parte di ambiente che è in grado di controllare.

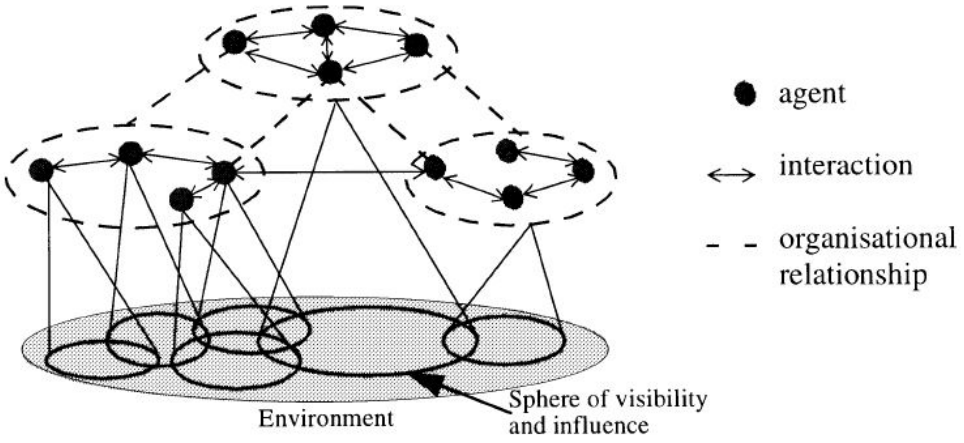

Figura 3.3: Struttura di un sistema multi-agente (MAS)

Come descritto in [39], quella fornita fino a qui è una prima definizione di agente che identifica quelli che sono chiamati *Agenti Deboli*. In questa definizione ricadono infatti tutte le entità che hanno le quattro caratteristiche fondamentali sopra elencate (*autonomia*, *proattività*, *reattività al cambiamento*, *abilità sociali*). Sempre dallo stesso documento emerge tuttavia una seconda definizione, quella di *Agente Forte*. Essa deriva dagli studi relativi all'AI e va ad aggiungere a queste quattro proprietà fondamentali una componente

mentale o cognitiva; secondo tale definizione un agente possiede conoscenza del mondo, desideri ed intenzioni, con un comportamento *goal-oriented*.

Da tale definizione traggono origine gli agenti cognitivi ed il modello *beliefs-desires-intentions* (BDI), con cui sono state mappate le entità del modello del dominio.

3.2.2 Il Modello BDI

Il modello Beliefs-Desires-Intentions trae ispirazione dalla teoria del *Human Pratical Reasoning* espressa da Bratman in [9].

Secondo questo modello gli agenti presentano tre componenti principali:

- *Beliefs*: Rappresentano la conoscenza dell'agente, ciò che conosce rispetto al suo stato e allo stato dell'ambiente circostante in cui vive. È da notare come questa conoscenza risulti spesso parziale rispetto al complessivo stato del mondo.

- *Desires*: Rappresentano gli obiettivi a lungo termine dell'agente. Tra di essi l'agente ne sceglierà uno o alcuni per la realizzazione, a cui associa delle Intenzioni per ottenere tali risultati.

- *Intentions*: Rappresentano ciò che l'agente ha scelto di fare per raggiungere i suoi obiettivi. Esse sono quindi un insieme di azioni.

Su queste tre componenti si basa il ciclo di controllo che definisce il comportamento dell'agente, espresso in pseudo-codice in figura 3.4, tratta da [8].

Durante l'esecuzione l'agente controllerà ciclicamente l'ambiente tramite i suoi sensori per poter aggiornare i suoi Belief correnti. Successivamente, in base alle proprie Intention e Belief aggiornerà i Desire e da essi ricaverà le Intention correnti. Sulla base di esse potrà quindi generare un piano di esecuzione per poterle soddisfare, che verrà successivamente messo in esecuzione nel ciclo interno (righe 9-22). L'esecuzione è gestita tramite un ciclo in modo da poter gestire eventuali cambiamenti nell'ambiente e modificare il piano di azione durante l'esecuzione stessa. Infatti l'agente dopo aver eseguito una singola azione presa dal piano andrà nuovamente ad osservare l'ambiente ed eventualmente a riconsiderare il piano di azione se questo dovesse risultare in qualche maniera utile al raggiungimento dell'obiettivo.

L'implementazione effettiva di questo modello dipende da come vengono gestite le funzioni presenti all'interno di questo ciclo di controllo, le quali modificano il comportamento effettivo dell'agente. BDI infatti è un modello che è stato implementato in diversi framework e linguaggi di programmazione, ognuno con le sue caratteristiche specifiche.

```
1.  B ← B0;          /* B0 are initial beliefs */
2.  I ← I0;          /* I0 are initial intentions */
3.  while true do
4.    get next percept ρ via sensors;
5.    B ← brf(B, ρ);
6.    D ← options(B, I);
7.    I ← filter(B, D, I);
8.    π ← plan(B, I, Ac);  /* Ac is the set of actions */
9.    while not (empty(π) or succeeded(I, B) or impossible(I, B)) do
10.     α ← first element of π;
11.     execute(α);
12.     π ← tail of π;
13.     observe environment to get next percept ρ;
14.     B ← brf(B, ρ);
15.     if reconsider(I, B) then
16.       D ← options(B, I);
17.       I ← filter(B, D, I);
18.     end-if
19.     if not sound(π, I, B) then
20.       π ← plan(B, I, Ac)
21.     end-if
22.   end-while
23. end-while
```

Figura 3.4: Ciclo di controllo di un agente BDI

3.2.3 Modellazione dell'Ambiente in un Sistema ad Agenti: L'Astrazione di Artefatto

Gli agenti per interagire con l'ambiente circostante necessitano di entità che esprimano e descrivano la modalità di interazione. Concettualmente parlando tali entità possono essere rappresentate come *Artefatti*.

Modello A&A

Il metamodello Agents and Artifacts (A&A), presentato da Ricci, Viroli ed Omicini in [34] descrive il concetto di Artefatto partendo dall'osservazione delle attività umane e traendo ispirazione da Activity Theory e Distributed Cognition.

In base al modello descritto in tale studio gli artefatti sono entità usate per costruire l'ambiente (o *working context*) per gli agenti. Citando [34], "*Gli artefatti sono entità non goal-oriented progettate in maniera esplicita per incarnare e offrire una certa funzione, la quale è usata dagli agenti per raggiungere i loro obiettivi individuali o sociali.*"

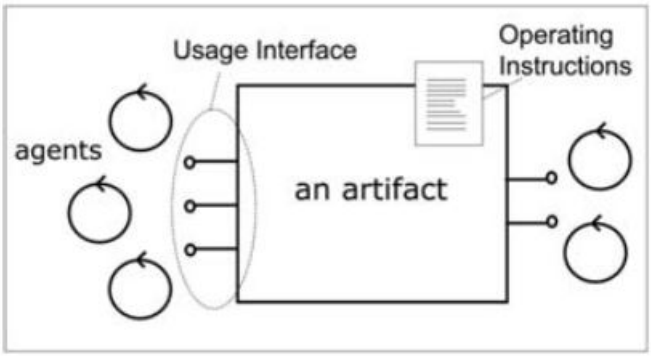

Figura 3.5: Rappresentazione astratta di un artefatto, tratta da [34]

Essi rappresentano quindi un'interfaccia verso un certo elemento ambientale che è in grado di fornire una specifica funzionalità, permettendo all'agente di ottenere un certo risultato in maniera trasparente rispetto all'effettiva implementazione dell'elemento stesso. In questo modo è possibile modellare la parte dell'ambiente del MAS che non è direttamente modellabile come entità proattiva in modo che risulti accessibile agli agenti. Per questo si presta molto bene all'applicazione nel contesto di questo lavoro di tesi come strumento concettuale per rappresentare un'interfaccia con l'ambiente e rendere possibile l'integrazione tra il paradigma ad agenti ed il modello dei Digital Twin.

3.3 Il Modello Digital Twin

Non esiste in letteratura una definizione univoca di Digital Twin, ma possiamo prendere in maniera intuitiva le definizioni fornite da Qi e Tao in [31], in cui si definisce Digital Twin "*un modello virtuale di un oggetto fisico creato in modo da simulare il suo comportamento nell'ambiente fisico*" oppure la definizione presentata in [15], secondo cui un Digital Twin è semplicemente "*la replica digitale di una entità, vivente o non vivente, che permette una trasmissione senza interruzione dei dati tra il mondo fisico e quello virtuale*". Da tali articoli emergono inoltre gli aspetti fondamentali dei Digital Twin. Ogni Digital Twin ha tre elementi fondamentali:

- L'entità fisica situata nell'ambiente fisico (Physical Asset)
- L'entità virtuale situata nell'ambiente virtuale (Digital Asset)
- I dati che legano le due entità e rappresentano la connessione tra i due ambienti

I Digital Twin rappresentano infatti "*un mapping dinamico bidirezionale tra gli oggetti fisici e il loro modello virtuale*", come descritto in [31].

Questo comporta la possibilità di ottenere effetti sul mondo virtuale agendo sulle entità del mondo fisico e viceversa, ottenendo un certo grado di integrazione tra i due livelli. Tramite questa integrazione il livello digitale permette non solo di rappresentare ma anche di aumentare ed arricchire la controparte fisica fornendogli nuove funzionalità. Questo porta verso un'ottica di fusione tra i due livelli, ottenendo un sistema in cui i due livelli sono completamente integrati e complementari, come ad esempio quello descritto nel modello Mirror World in [32].

3.3.1 Integrazione tra gli Approcci

Ognuno degli approcci descritti è adatto a modellare sistemi con caratteristiche specifiche: se gli agenti sono adatti nell'applicazione in sistemi in cui è richiesto un certo grado di autonomia, i Digital Twin si prestano bene al caso in cui sia necessario rappresentare nel sistema entità fisiche. Nel caso in esame tuttavia il sistema presenta caratteristiche miste: infatti sebbene la parte fisica sia preponderante in un sistema pervasivo di questo genere, che proprio per definizione è situato e legato all'ambiente, esso presenta una parte computazionale forte, che dovrà inoltre interagire con le entità dell'ambiente stesso. Per cui viene naturale l'idea di rappresentare entrambe le parti ognuna con l'approccio più adatto alla sua natura, riuscendo però a garantire una modalità di interazione tra le due componenti. Per garantire questo è possibile utilizzare gli artefatti, che da una parte permettono agli agenti di interagire con il sistema e dall'altra possono essere facilmente collegati con i Digital Twin tramite la definizione di una apposita interfaccia, facendo da wrapper. Si ottiene così un sistema in cui le entità fisiche sono mappate nei corrispondenti Digital Twin, e la parte computazionale è gestita in maniera autonoma da un sistema multiagente, in grado di utilizzare una serie di artefatti per ottenere funzionalità specifiche sfruttando il collegamento con i Digital Twin e quindi con la loro controparte fisica. In questo modo gli agenti possono interagire con gli elementi fisici dell'ambiente e viceversa, fornendo elementi di pervasività al sistema.

3.4 Architettura Integrata per il Progetto Smart Shock Room

Per ottenere l'architettura del sistema Smart Shock Room si è proceduto ad effettuare un mapping delle entità del modello del dominio secondo i principi sopra esposti.

Digital Twin nella Smart Shock Room

Un buon numero di entità presenti nel modello del dominio fanno riferimento ad elementi fisici presenti nell'ambiente della Shock Room, per cui possono essere mappati in maniera quasi automatica nei rispettivi Digital Twin. La controparte fisica sarà inoltre in alcuni casi fornita di sensori ed attuatori con i quali il sistema può interagire tramite la parte digitale dell'entità.

Le entità che possono essere mappate come Digital Twin sono:

- *Gli Utenti*
- *I Gestori di Input*
- *Il paziente*
- *Il trauma in corso*
- *Le Risorse*
- *Le Sorgenti di informazioni*

Inoltre è interessante vedere come, applicando la definizione, di fatto il sistema Smart Shock Room nel suo insieme può essere considerato un Digital Twin dell'intera Shock Room fisica.

Agenti nella Smart Shock Room

Nel modello del dominio sono presenti gli Assistenti, *Room Assistant* e *Trauma Assistant*, che si dovranno occupare dell'effettiva esecuzione dei comandi impartiti dagli utenti e dalla gestione degli eventi provenienti dall'ambiente. Data la natura reattiva, ed in parte proattiva, di tali entità risulta semplice rappresentarli come agenti, rappresentando la parte attiva del sistema. Anche gli utenti, in quanto rappresentazione degli utilizzatori umani, intrinsecamente proattivi, possono essere rappresentati come agenti.

Artefatti nella Smart Shock Room

Rappresentando l'interfaccia per gli agenti verso l'ambiente per ogni Digital Twin saranno presenti uno o più artefatti in grado di fornire alcune specifiche funzionalità, facendo da wrapper per i Digital Twin.

3.4.1 Architettura ad Alto Livello

Il sistema di gestione della Smart Shock Room prenderà il nome di *ASTRA (Augmented Spaces for TRauma Assistance) Room Manager*.

Il risultato di un primo mapping tra il modello del dominio e l'architettura del sistema ha come risultato la definizione dei macrocomponenti del sistema, mostrati in figura 3.6. Tale rappresentazione mostra l'organizzazione del sistema ad alto livello

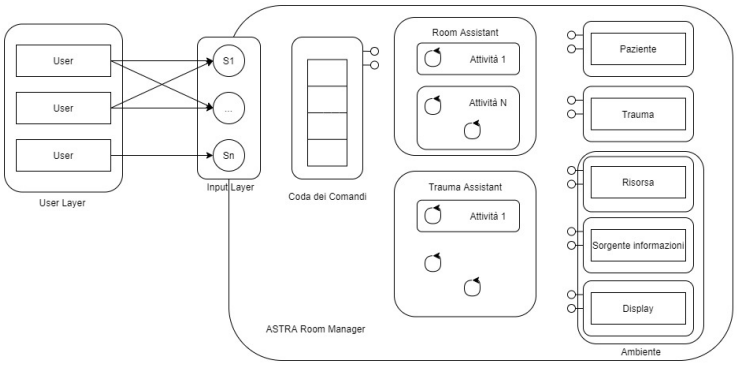

Figura 3.6: Macrocomponenti Architetturali

3.4.2 User Layer

In questo livello risiedono le entità relative agli utenti, la controparte digitale degli utenti umani del sistema, i membri del Trauma Team che agiscono nella Shock Room. Data la natura proattiva delle entità che essi rappresentano, tali entità sono mappate nel sistema come agenti software. Queste entità rappresentano la sorgente dei comandi da impartire al sistema, e logicamente non fanno parte del sistema di gestione della Shock Room o della stanza stessa, in quanto ne rappresentano gli utilizzatori.

Essendo agenti tali entità non presentano interfacce di input. Tali agenti avranno nei loro Beliefs:

- un *codice identificativo* univoco e il *nome* dell'utente che rappresentano.
- il *ruolo* ricoperto al momento dall'utente, che servirà a stabilire la tipologia di Utente che in questo momento esso ricopre e di conseguenza quali tipi di comandi può richiedere.

Ogni agente avrà poi uno o più piani utili per la generazione di comandi da impartire al sistema ed offrirà inoltre la possibilità di modificare il proprio ruolo. L'idea è infatti che ogni utente umano abbia un unico agente software che ne è la rappresentazione nel layer digitale e che esso possa, in una futura estensione del sistema, seguirlo in diversi ambienti dell'ospedale, dove lo stesso utente umano può avere ruoli differenti e quindi possa richiedere operazioni differenti.

Comandi

I comandi sono entità che rappresentano l'azione che l'utente vuole svolgere tramite il sistema o su di esso. Tali entità rappresentano uno scambio di informazioni tra entità fisiche e/o digitali. Ogni comando generato avrà:

- un *tipo*, che rappresenta quale genere di operazione verrà richiesta al sistema tramite esso;
- un *bersaglio*, nel caso esso venga diretto verso una specifica entità. Un valore vuoto di tale proprietà indica il bersaglio di default, decido dal sistema stesso;
- un *richiedente*, un riferimento al codice identificativo dell'utente che ha generato il comando;
- uno o più *parametri* che ne specificano i dettagli e forniscono informazioni aggiuntive. Essi variano a seconda della tipologia del comando stesso.

Come espresso nel modello il sistema al momento gestisce quattro tipologie di comandi (*Visualizzazione*, *Monitoraggio*, *Annotazione*, *Azione*), con le relative tipologie: `visualisation`, `monitoring`, `annotation` e `action` .

In ASTRA i comandi sono rappresentati come oggetti JSON. Tale formato rappresenta una scelta ideale in quanto è un formato testuale appositamente studiato per lo scambio di informazioni ed è indipendente dalle tecnologie scelte, come descritto in [1], lasciando così la massima libertà all'implementazione.

A titolo di esempio un ipotetico comando per la visualizzazione del valore della saturazione del paziente sarebbe il seguente:

```
{
  "command" : {
      "type" : "visualisation",
      "target" : "",
      "issuer" : "U_1357",
      "params" : [
        {"value" : "spO2"}
      ]
  }
}
```

3.4.3 Input Layer

In questo livello risiedono le entità che si occupano della gestione dei comandi in input. Tali entità fungono da punto di ingresso per il sistema ed hanno il compito di ricevere ed elaborare i comandi provenienti dagli utenti. Questo layer è composto da componenti con due ruoli distinti, gli *Input Handler* che si occupano della gestione degli input provenienti dagli utenti e sono legati a dei dispositivi specifici e i *Command Manager* che si occupano della loro elaborazione e dalla creazione delle richieste da sottoporre al sistema affinchè possano essere effetivamente completati.

Input Handler

Come emerge dal modello i gestori di input non sono associati al singolo utente ma sono condivisi a livello di ambiente tra i vari utilizzatori presenti. Queste entità sono legate a specifici dispositivi fisici, di cui rappresentano il digital layer in un'ottica di Digital Twin e ne rappresentano il sistema di gestione. Per ogni tipologia di Input prevista dal modello (Trauma, Ambientale e Multiplo) verrà previsto un apposito dispositivo. Essi avranno diversa natura per poter garantire la multimodalità richiesta al sistema.

Command Manager

I comandi raccolti dagli Input Handler sono inviati ad un apposito componente che si occuperà della raccolta e aggregazione di tali informazioni per poter generare le richieste di Attività che saranno in seguito elaborate dal sistema per l'effettiva esecuzione di operazioni.

Dovranno essere presenti almeno due diversi componenti, specializzati sui diversi tipi di comandi. Uno di essi in particolare gestirà le attività relative al trauma (*Trauma Command manager*) e l'altro quelle relative alla stanza e all'ambiente (*Room Command manager*). Nello specifico saranno presenti 1

Room Command Manager per ogni stanza e 1 Trauma Command Manager per ogni trauma. Infatti è possibile che coesistano più ambienti diversi e più traumi nello stesso momento.

A livello architetturale questi componenti che ricevono i comandi sono rappresentati da microservizi con interfaccia REST. Tramite queste API gli Input Handler potranno interagire con essi per la gestione dei comandi ricevuti, ottenendo un certo grado di disaccoppiamento tra i due livelli. Ad essi sarà collegato un artefatto che si occuperà dell'aggregazione delle informazioni ricevute e permetterà agli agenti di interagire con essi.

Servizi REST

REST sta per *Representational State Transfer* ed indica un modello architetturale basato sul protocollo HTTP. Tale modello permette di ottenere interoperabilità e scalabilità ed è descritto nel dettaglio da Fielding in [17]. In base a questo modello la comunicazione avviene secondo il modello client-server tramite una interfaccia uniforme che stabilisce esattamente quali operazioni sono consentite. La comunicazione è stateless ed è possibile accedere alle risorse in maniera uniforme tramite URL e metodi HTTP, specificati dall'interfaccia del server.

Modello a Microservizi

Il modello a microservizi, come descritto in [29], è un modello architetturale secondo cui il sistema è composto da una serie di servizi piccoli, specializzati ed autonomi che collaborano tra loro per ottenere un risultato complessivo. Questo garantisce al sistema modularità e scalabilità e permette un disaccoppiamento maggiore tra i componenti dello stesso.

Command Manager REST API

Andando ad applicare questo modello al progetto ASTRA è possibile definire l'interfaccia che presenteranno i due tipi di Command Manager. Infatti i due componenti presentano la stessa interfaccia nonostante gestiscano comandi di tipi diversi. Anche la rappresentazione interna dei comandi è uniforme e deriva direttamente dai comandi ricevuti dagli utenti, il cui formato è definito in 3.4.2.

Il modello, descritto in figura 3.7, mostra tale rappresentazione. Lo stato di un comando può variare tra `pending`, `in_processing`, `completed` e `failed` a seconda se il comando sia in elaborazione, sia stato preso in carico o sia già stato completato. Diventerà `failed` nel caso in cui non sia possibile portarlo a termine per qualche motivo, ad esempio perchè viene richiesta una operazione non supportata dal sistema.

Al fine di valutare le performance dell'applicazione vengono anche memorizzati il timestamp del momento della creazione del comando, quello della

presa in carico e quello del suo completamento, in modo da poter valutare il tempo complessivo per l'elaborazione del comando.

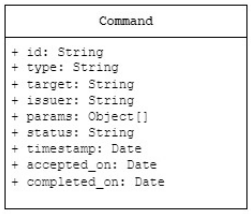

Figura 3.7: Rappresentazione dei Comandi

Data tale rappresentazione l'interfaccia del relativo servizio sarà la seguente:

- `POST` *'/api/commands/'*: permette l'aggiunta di nuovi comandi affinchè essi vengano elaborati.
- `GET` *'/api/commands/:command_id'*: permette di ottenere uno specifico comando per monitorarne lo stato.
- `GET` *'/api/commands/pending'*: permette di ottenere la lista dei comandi ricevuti ma non ancora presi in carico dal sistema.
- `GET` *'/api/commands/in_processing'*: permette di ottenere la lista dei comandi in elaborazione non ancora completati.
- `GET` *'/api/commands/completed'*: permette di ottenere la lista dei comandi completati.
- `GET` *'/api/commands/failed'*: permette di ottenere la lista dei comandi falliti in seguito ad un errore.
- `PUT` *'/api/commands/:command_id'*: permette di modificare uno specifico comando.
- `PUT` *'/api/commands/:command_id/status'*: permette di modificare lo stato di uno specifico comando.
- `DELETE` *'/api/commands/:command_id'*: permette di eliminare uno specifico comando.

Ad ognuno dei servizi Command manager sarà inoltre collegato uno specifico artefatto che permetterà di effettuare query ed operazioni sui comandi inviati al sistema. Esso presenta l'interfaccia descritta nella figura 3.8.

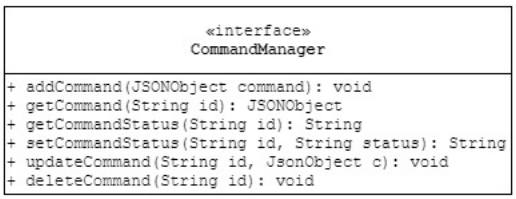

Figura 3.8: Interfaccia dell'artefatto `CommandManager`

Coda dei Comandi

Quando i due Command Manager ricevono un comando da uno degli utenti lo elaborano in modo tale che possa essere inserito nel sistema. Essi infatti dovranno essere salvati in attesa che il sistema proceda con la loro esecuzione. Ogni comando dovrà essere gestito una sola volta ed in base alla tipologia di richiesta presenterà un certo grado di criticità. Per poter gestire in maniera efficace tali comandi essi sono quindi organizzate in differenti code con priorità, ognuna delle quali sarà relativa ad uno specifico Command Manager e presenterà un artefatto tale per cui gli agenti della stanza vi possano interagire. Sono infatti gli agenti stessi che si occupano della scelta dei comandi dalla coda in base alle proprie capacità specifiche, in quanto essi non sono equivalenti tra di loro. Questo garantisce la massima estendibilità ed apertura del sistema, in quanto è possibile l'ingresso nel sistema di nuovi agenti senza dover agire sui Command Manager.

Per aumentare l'efficienza di questo meccanismo è possibile gestire le code tramite un pattern *Publish/Subscribe*. Come descritto in [16] tale pattern permette a entità dette *Subscriber* (nel caso in esame gli agenti) di esprimere il loro interesse verso uno specifico evento o un pattern di eventi in modo da essere notificati ogniqualvolta tale evento (l'aggiunta di un comando ad una specifica coda), generato da altre entità dette *Publisher* (i Command Manager), si verifica. In questo modo si può ottenere una comunicazione asincrona e garantire disaccoppiamento tra le diverse entità. Applicando questo pattern al sistema ASTRA ogni agente potrà quindi rimanere in ascolto per tutti i comandi che esso è in grado di gestire, automatizzando il processo di selezione degli stessi.

Artefatti CommandQueue

Affinché gli agenti possano interagire con le code dei comandi ognuna di esse è rappresentata come un artefatto. Esso fornirà nella sua interfaccia di utilizzo metodi per l'aggiunta di comandi alla coda e la loro estrazione. Di conseguenza l'interfaccia risultante sarà la seguente:

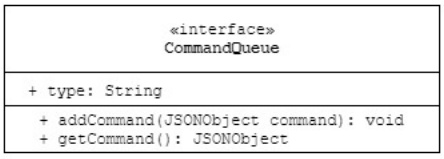

Figura 3.9: Interfaccia dell'artefatto `CommandQueue`

Data la modalità di interazione tra le code e gli agenti ogni volta che viene aggiunto un comando ad una delle liste essa emetterà uno specifico evento, in modo che gli agenti interessati siano notificati della nuova attività disponibile.

3.4.4 Agent Layer

Gli agenti costituiscono la parte attiva del sistema che si occupa dell'effettivo svolgimento delle operazioni necessarie per completare i comandi richiesti dagli utenti. Logicamente questo livello è composto da due agenti che rappresentano gli Assistenti del modello del dominio, Room Assistant e Trauma Assistant. Come emerge dal modello stesso tali agenti saranno specializzati in modo da poter gestire un insieme complementare di attività. La struttura dei due agenti sarà tuttavia simile; nella loro base di conoscenza conterranno informazioni riguardanti le risorse attualmente disponibili all'interno del sistema. Presenteranno inoltre alcuni piani con cui gestire l'arrivo di nuovi comandi e il loro svolgimento. Infatti, in generale, questi agenti interagiscono con le code dei comandi rimanendo in ascolto di specifici eventi. Ogni agente è in ascolto soltanto delle code per cui ha competenza. Quando uno di questi eventi viene scatenato l'agente interessato svolge un piano, dipendente dal tipo e dal contenuto del comando ricevuto, che porta al suo completamento. Per poter completare il comando l'agente, nella maggior parte dei casi, necessita di interagire con gli artefatti che rappresentano le risorse, descritti in dettaglio nel paragrafo 3.4.5. Una volta completate le operazioni necessarie viene notificato il completamento del comando al Command Manager in modo da poter fornire un feedback all'utente se esso lo richiede.

3.4.5 Environment Layer

Questo ultimo livello raggruppa gli artefatti con cui gli agenti interagiscono per poter completare le operazioni richieste dai comandi. Questo è parte dell'ambiente in cui gli agenti vivono e rappresentano le diverse entità, fisiche e non, presenti nell'ambiente della Shock Room. Le entità appartenenti a questo insieme possono inoltre interagire con entità esterne per il loro funzionamento, come ad esempio l'entità relativa al trauma che si andrà ad interfacciare con TraumaTracker Service per accedere ai dati relativi al trauma in corso o ancora l'entità relativa al paziente che potrà accedere alle informazioni della sua storia clinica interfacciandosi con il sistema informativo dell'ospedale. In questo insieme ricadono diverse tipologie di artefatti, ognuna delle quali avrà delle caratteristiche specifiche ed un diverso insieme di operazioni che saranno in grado di fornire agli agenti.

Paziente

Questo artefatto rappresenta il paziente ed ha il compito di raccogliere e fornire diverse informazioni relative alla sua identificazione. Infatti nel corso del processo di cura vengono generati diversi codici identificativi relativi ad esempio alle diverse tipologie di indagini diagnostiche effettuate, che possono essere memorizzati e gestiti tramite questa entità. Nella figura 3.10 è mostrata l'interfaccia di tale artefatto:

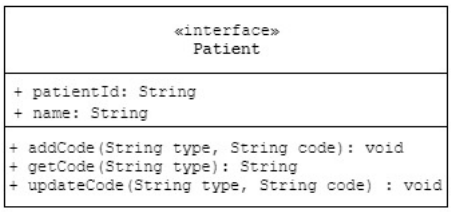

Figura 3.10: Interfaccia dell'artefatto `Patient`

Trauma

Questo artefatto rappresenta il trauma in corso ed avrà il compito specifico di gestirne gli eventi e le operazioni effettuate. Questo componente si interfaccerà con TraumaTracker Service per garantire l'interazione tra i due sistemi. Inizialmente consentirà di gestire eventi, somministrazione di farmaci e manovre effettuate, lasciando tuttavia aperta la possibilità di integrare nuove

funzionalità di TraumaTracker in futuro. La figura 3.11 mostra l'interfaccia di tale artefatto.

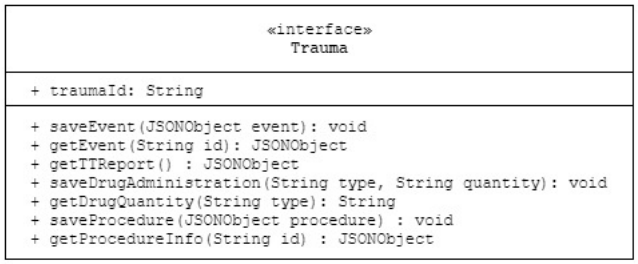

Figura 3.11: Interfaccia dell'artefatto `Trauma`

Elementi Ambientali

Questa categoria di artefatti rappresenta gli strumenti e gli oggetti fisici presenti nella Shock Room con cui il sistema può interagire. A livello architetturale essi sono mappati come Digital Twin, nella forma di microservizi con interfaccia REST. In questo modo gli artefatti fungono da wrapper, fornendo interfacce per gli agenti per permettergli di raggiungere i servizi offerti dalle singole entità. Logicamente questi artefatti possono essere ulteriormente suddivisi in due sottocategorie, textitSorgenti di informazioni e *Risorse* in base al tipo di operazioni che permettono di eseguire, come espresso in relazione ad essi nel modello del dominio. Ogni entità avrà la sua specifica interfaccia, legata alle caratteristiche proprie dell'entità fisica a cui è collegata, ma in generale presenteranno un codice identificativo univoco e uno o più metodi per accedere ai dati. Inoltre sarà possibile monitorare in maniera continua il valore di tali informazioni sfruttando le proprietà osservabili esposte dagli artefatti.

Sorgenti di Informazioni Questi artefatti permettono di accedere in sola lettura ad informazioni misurate o contenute nella controparte fisica a cui i relativi Digital Twin sono collegati.

A titolo di esempio viene descritta l'interfaccia relativa al pulsiossimetro. Tale dispositivo fisico permette di conoscere il valore della saturazione dell'ossigeno nel sangue del paziente. Immaginando che tale dispositivo abbia delle funzionalità Smart esso è rappresentato nel mondo digitale da un Digital Twin che avrà la forma di un microservizio che espone una interfaccia REST. Essa

presenterà un metodo che permetterà di accedere al valore misurato dal dispositivo fisico in quel momento. Inoltre presenterà altri metodi per conoscere lo stato del dispositivo. Collegato a questo Digital Twin ci sarà un artefatto, il quale presenta uno stato osservabile relativo al valore della saturazione, in modo che gli agenti possano essere sempre aggiornati sul valore attuale, oltre ad un metodo per ottenerne il valore istantaneo. Per funzionare questo artefatto farà una o più richieste HTTP all'API esposta dal microservizio per ottenere il valore richiesto. Il risultato di tale organizzazione è mostrato nella figura 3.12.

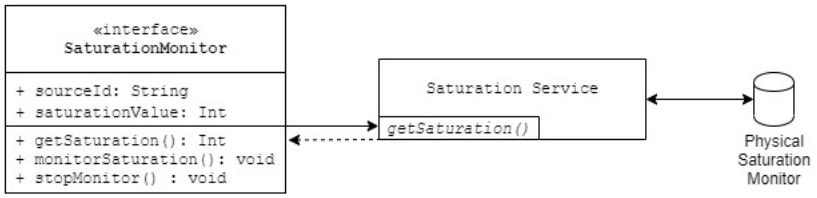

Figura 3.12: Esempio di sorgente di informazioni: Pulsiossimetro

Risorse Questa categoria di artefatti, a differenza delle sorgenti di informazioni, permettono non solo di accedere in lettura alle informazioni in essi contenute ma anche di agire direttamente sul loro stato, permettendo agli agenti di produrre effetti sullo stato del mondo tramite essi. A titolo di esempio per questa categoria viene presentata l'interfaccia relativa al Display.

Anche in questo caso esiste un Digital Twin relativo al display fisico presente nella stanza, mappato come un microservizio con relative API REST. Tale interfaccia offrirà metodi per richiedere la visualizzazione di informazioni, per accedere allo stato del display ma anche per modificare il layout delle informazioni visualizzate. Per poter interagire con esso gli agenti useranno un Artefatto specifico per questa entità. L'organizzazione complessiva è mostrata nella figura 3.13

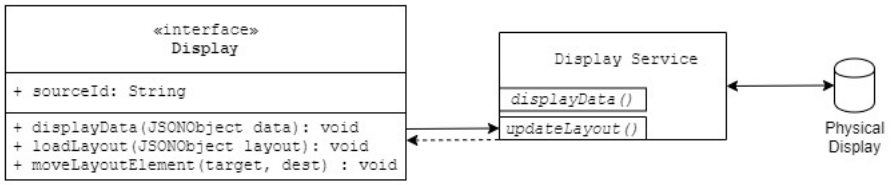

Figura 3.13: Esempio di risorsa: Display

3.4.6 ASTRA Room Manager

L'unione di questi elementi compone l'architettura complessiva del sistema, rappresentata nella figura 3.14, da cui emerge chiaramente l'idea dell'integrazione tra i diversi approcci di sviluppo.

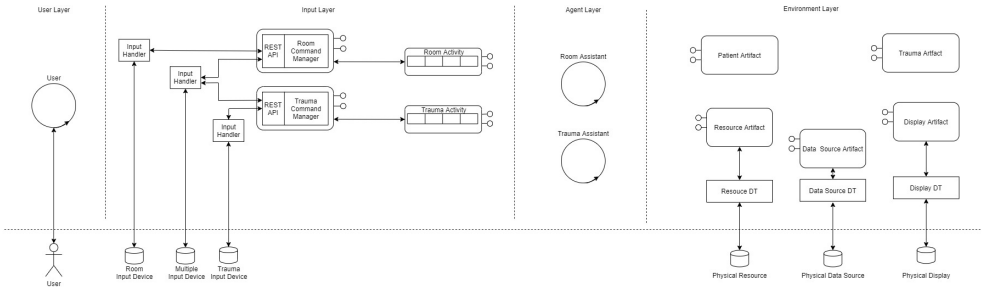

Figura 3.14: Architettura di dettaglio

Capitolo 4

Il Sistema ASTRA Room Manager

Basandosi sull'architettura sviluppata nel capitolo precedente è stato ideato e realizzato un prototipo del sistema per permettere di testarne le funzionalità e le performance. Nelle prossime pagine verranno descritte nel dettaglio le caratteristiche di tale prototipo e verranno inoltre presentate le modalità e le tecnologie scelte per la sua implementazione.

4.1 Gestori di Input

Nell'ambito del prototipo l'utente umano si trova ad interagire con il sistema tramite dei dispositivi fisici di input che, per garantire la multimodalità richiesta nei requisiti, saranno di due diverse tipologie. In particolare sono previsti:

- Un gestore di input legato alla stanza, sempre presente all'interno dell'ambiente fisico della Shock Room, che si occupa di tutti gli aspetti relativi alla gestione della stessa e dell'ambiente. Nel caso specifico del prototipo questo sarà un dispositivo di input grafico.
- Un gestore di input vocale, che permette l'interazione con entrambe le componenti del sistema e garantirà l'interazione hands-free, permettendo di impartire comandi senza la necessità di utilizzare le mani o di spostare lo sguardo dalle operazioni in corso.

Quest'ultimo è quello che richiederà una maggiore attenzione in quanto richiederà una attenta progettazione dei comandi vocali che sarà in grado di gestire, in modo da poter trovare il giusto compromesso tra complessità delle operazioni consentite e semplicità di utilizzo. Questo sarà fondamentale per

permettere al sistema di funzionare in maniera efficiente ed allo stesso tempo permettere agli utenti un utilizzo semplice ed una curva di apprendimento non troppo ripida. Come descritto in [30] la possibilità di avere un'interfaccia di comunicazione vocale (*Vocal User Inteface* o *VUI*) risulta molto importante perché la voce rappresenta la modalità di interazione preferita dall'uomo e rispetto ad altre modalità garantisce una maggiore velocità ed intuitività.

4.1.1 ASTRA Android Room Controller

Nel prototipo il gestore di input relativo alla stanza è stato implementato nella forma di una applicazione Android che sarà mandata in esecuzione in un apposito tablet posizionato nell'ambiente della Shock Room.

Funzionalità

Questa applicazione al momento permette di gestire solamente la visualizzazione di informazioni relative al caso in corso nel display della Shock Room. Sarà possibile selezionare quale informazione visualizzare da una lista dei dati disponibili e in quale posizione del display visualizzarla. Sarà inoltre possibile per ogni dato scegliere la modalità di visualizzazione (valore singolo o monitoraggio).

Al fine di migliorare l'usabilità e ridurre lo spreco di tempo l'applicazione manterrà lo schermo sempre attivo, limitando così l'overhead dovuto ad un eventuale sblocco del dispositivo in caso di necessità.

Modello dell'Applicazione

Tale sistema presenta una struttura molto semplice, rappresentata nella figura 4.1. Tramite l'entità `DisplayPosition` sarà possibile definire una parte del layout dell'applicazione in modo che sia coerente con l'effettiva organizzazione del display fisico su cui verranno visualizzati i dati. In questo modo sarà possibile adattare l'applicazione al layout scelto. Esistono quattro tipologie di `DisplayPosition`:

- `ScreenPanel`: Rappresenta il livello più esterno in cui inserire altri elementi, non permette la visualizzazione diretta di dati
- `ParentPanel`: Rappresenta il contenitore per altri DisplayPosition, non permette la visualizzazione diretta di dati.
- `BasePanel`: Rappresenta l'elemento base in cui visualizzare i dati.

- `SystemPanel`: Rappresenta elementi gestiti in automatico dal sistema ASTRA, in cui verranno visualizzati dati in automatico e in cui l'utente non può interagire.

L'entità `PatientData` contiene invece le informazione relative ai dati di cui è possibile richiedere la visualizzazione.

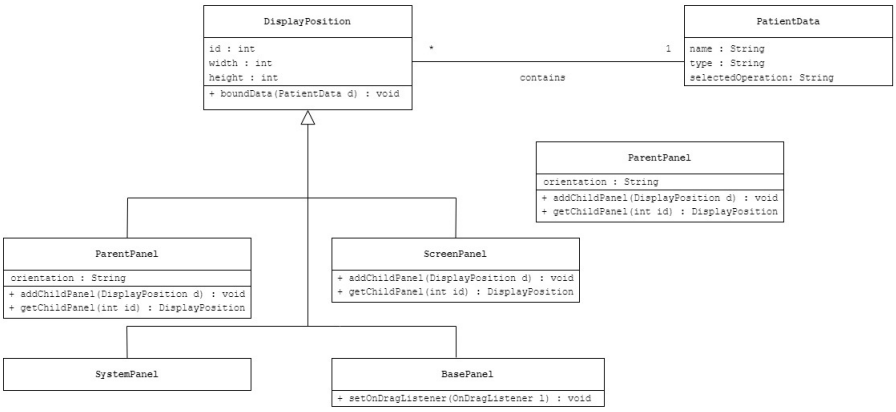

Figura 4.1: Modello di Android Room Controller

Implementazione

L'applicazione è implementata per il sistema operativo Android usando il linguaggio Java. È realizzata come una singola Activity. Durante l'inizializzazione viene generata la lista di `DisplayPosition` che definiscono il layout del display e la lista di `PatientData` che rappresenta i dati che possono essere visualizzati. Inoltre si abilita lo schermo a rimanere sempre attivo tramite il comando:

```
getWindow()
    .addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
```

Terminata l'inizializzazione è possibile utilizzare l'applicazione per richiedere la visualizzazione di informazioni sul display della Shock Room. Questa operazione è svolta tramite un meccanismo drag & drop: è possibile selezionare una delle voci da visualizzare dall'elenco e trascinarla nella posizione specifica in cui si vuole visualizzarla. Quando si rilascia l'elemento verrà inviata una richiesta HTTP al servizio relativo al Room Command Manager contenente le informazioni relative al comando richiesto e alla posizione nella quale lo si

vuole mostrare. Questa è infatti ricavata dall'elemento in cui si è rilasciato il dato.

```
case DragEvent.ACTION_DROP:

    // Gets the item containing the dragged data
    ClipData.Item item = event.getClipData().getItemAt(0);
    String dataName =
        item.getIntent().getStringExtra("patientDataName");
    String dataType =
        item.getIntent().getStringExtra("patientDataType");
    String dataOp =
        item.getIntent().getStringExtra("patientDataOperation");

    PatientData draggedData = new PatientData(dataName, dataType);
    draggedData.setSelectedOperation(dataOp);

    ...

    if( view instanceof LinearLayout ) {
        LinearLayout c = (LinearLayout) view;
        //get the info about screen position
        DisplayPosition p = this.screenPanel.getChildByID(c.getId());
        p.boundData(draggedData);
        // send HTTP POST Request
        sendCommandRequest(p);
    }

    return true;
```

Listato 4.1: Drop Operation

È inoltre possibile selezionare il tipo di visualizzazione tramite una Dialog che si può visualizzare cliccando su uno degli elementi presenti nella lista. Questo modificherà il tipo di comando che verrà richiesto al sistema per quell'elemento.

Il repository relativo a questa parte del progetto è disponibile all'indirizzo `https://github.com/manuBottax/ASTRA-AndroidRoomManager`.

NOME	ID	TIPOLOGIA
Dati Anagrafici del Paziente	patient_details	Visualizzazione
Frequenza Cardiaca	heart_rate	Visualizzazione, Monitoraggio
Pressione Arteriosa	blood_pressure	Visualizzazione, Monitoraggio
Saturazione	spO2	Visualizzazione, Monitoraggio
Temperatura	temperature	Visualizzazione, Monitoraggio
Livello CO2	CO2_level	Visualizzazione, Monitoraggio
Emogas analisi	ega	Visualizzazione, Monitoraggio
ROTEM	rotem	Visualizzazione, Monitoraggio
TAC Cerebrale	tac	Visualizzazione
ECG	ecg	Visualizzazione
Esami del sangue	blood_tests	Visualizzazione
RX torace	chest_rx	Visualizzazione
Tempo Totale	total_time	Visualizzazione, Monitoraggio
Tempo Stimato di arrivo	eta	Visualizzazione, Monitoraggio
Sacche di sangue utilizzate	used_blood_unit	Visualizzazione, Monitoraggio

Tabella 4.1: Parametri disponibili nell'applicazione

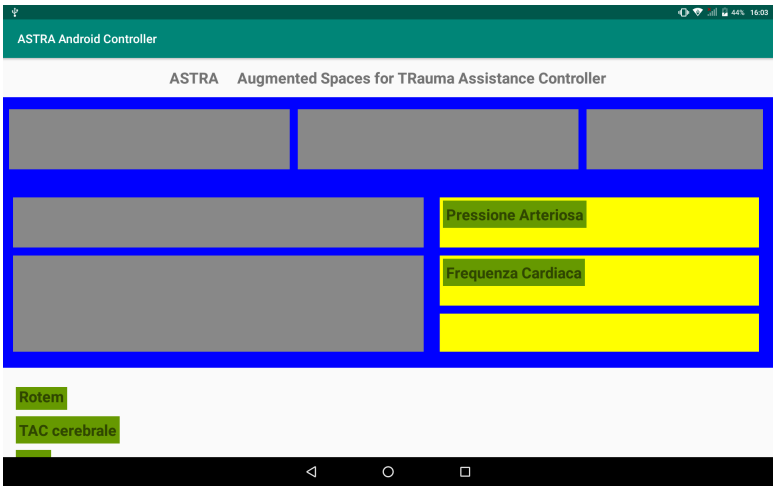

Figura 4.2: Layout dell'applicazione

Sviluppi Futuri

L'applicazione rappresenta uno stato prototipale e diverse funzionalità non sono effettivamente implementate. Per garantire un corretto funzionamento nell'ambiente reale ed un pieno raggiungimento dei requisiti richiesti sarà necessario implementare:

- *Recupero dinamico dei dati disponibili.* Al momento l'applicazione lavora con un set predefinito di dati che è possibile visualizzare. In futuro questo elenco dovrà essere ottenuto ed aggiornato periodicamente in base ai dati effettivamente messi a disposizione dal sistema.

- *Collegamento dinamico con il layout del display.* Al momento infatti il layout è definito a compile-time e non si adatta ad eventuali cambiamenti del layout se non tramite una modifica manuale. Dato che la modifica del layout è una funzionalità prevista del sistema è necessario che questa applicazione possa aggiornare dinamicamente la propria organizzazione interna in modo da rimanere coerente con tale rappresentazione.

- *Funzionalità di gestione.* Al momento l'applicazione permette solamente di richiedere la visualizzazione di informazioni. In futuro dovrebbe offrire anche funzionalità di gestione in relazione alla modifica dello stato delle risorse collegate, ad esempio la modifica del layout del display.

4.1.2 ASTRA Voice User Interface

L'interfaccia vocale rende possibile interagire con il sistema utilizzando tutti i comandi consentiti da esso. È infatti importante che l'utente possa interagire con il sistema in modalità hands-free indipendentemente da quale tipo di attività dovrà svolgere poiché in questo ambiente di utilizzo è possibile, anzi è molto probabile, che il sistema venga utilizzato contemporaneamente allo svolgimento di altre azioni che necessitano dell'attenzione e delle mani dell'utilizzatore. Per questo motivo è importante che il sistema fornisca un'interfaccia intuitiva e semplice da utilizzare ma che possa anche essere robusta nell'utilizzo pratico.

Quella che è stata realizzata è un'interfaccia vocale del tipo *Command-and-Control*, che richiede una azione specifica da parte dell'utente per indicare al sistema l'intenzione di parlare. Come descritto in [30] questa modalità è ancora la più comune per quanto riguarda le interfacce vocali ma si adatta bene al caso in esame in quanto è adatta a situazioni in cui il sistema non sa quando l'utente parlerà nuovamente ed evita che il sistema ascolti dialoghi in cui non è coinvolto.

Gestione dei Comandi Vocali

Ogni comando impartito ad un assistente vocale genera un flusso di informazioni che parte dall'utente e viene interpretato dal sistema fino a ritornare all'utente sotto forma di feedback sul risultato. Questo processo cambia poco da una tecnologia all'altra e viene descritto in dettaglio in [11]. L'esecuzione

inizia quando l'utente pronuncia una *wake word* o *wake phrase*, che permette di attivare l'assistente vocale. Questo è possibile tramite un riconoscimento vocale effettuato su un buffer di parlato che il dispositivo elabora localmente. Una volta che l'assistente vocale è attivo ascolta la frase fornita dall'utente e la invia tramite la rete ad un componente che si occupa del *Natural Language Processing* (NLP).

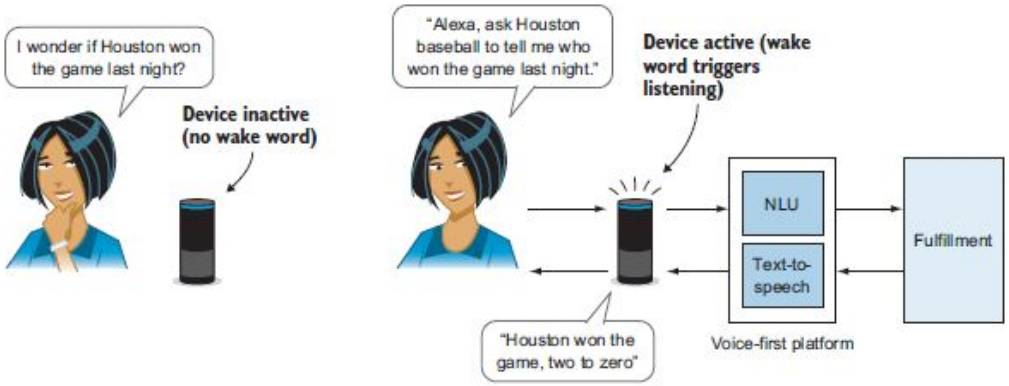

Figura 4.3: Flusso di elaborazione di un comando vocale, tratto da [11]

Questo processo consente ad un computer di interagire con comandi in linguaggio naturale e si basa su *Natural Language Understanding* per estrarre informazioni utili dalle frasi in linguaggio naturale e trasformarle in testo. Allo stato attuale questa operazione, molto complessa, è fornita come servizio direttamente dai vendor dei vari assistenti vocali e non c'è la necessità di implementarla nelle singole applicazioni.

Una volta completata questa operazione di riconoscimento il software deve elaborare la richiesta per capire cosa dovrà eseguire per rispondere al comando dell'utente. Questo è possibile tramite specifiche *Skill* o *Action*, che rappresentano le applicazioni sviluppate per l'esecuzione di compiti specifici tramite gli assistenti vocali.

Ogni Skill consente un certo insieme di operazioni, che prendono il nome di *Intent*. Quello che fa il sistema è semplicemente collegare una specifica frase dell'utente ad uno specifico Intent. Tuttavia dato che il linguaggio naturale prevede molteplici variazioni per dire la stessa cosa il sistema non si limita ad una singola frase, ma permette di specificare alcune frasi di esempio (*Sample Utterance*) con cui viene costruito un modello statistico usato per il fare il mapping tra la frase ricevuta dall'utente e l'intent necessario. Infine è possibile che la frase contenga dei parametri da usare nell'esecuzione dell'input, che è possibile specificare nelle frasi di esempio tramite *Slot*.

Il codice che specifica le operazioni da eseguire per ogni Intent è contenuto in un componente chiamato *fulfillment*, di solito eseguito su piattaforme serverless, come ad esempio *AWS Lambda*, che eseguono il codice on demand. Una volta completata l'esecuzione viene inviato al dispositivo un feedback per l'utente, che verrà convertito con un *Text-to-Speech* (TTS) in modo da essere letto dal dispositivo. Tale feedback è specificato usando il linguaggio *Speech Synthesis Markup Language (SSML)* e permette di specificare oltre al contenuto anche il tono e la modalità di lettura della frase.

Skill ASTRA Room Controller

Per il prototipo del sistema è stata realizzata una Skill per Alexa in grado di gestire i comandi degli utenti. Lo sviluppo di skill per Alexa è possibile tramite *Alexa Developer Console* [1]. La skill realizzata è di tipologia *custom*, in modo da permettere la personalizzazione dei vari elementi a seconda delle necessità specifiche ed è impostata per avere l'italiano come linguaggio di default. Ogni Skill Alexa è composta da due elementi fondamentali:

- Il *modello di interazione*, che rappresenta la VUI della skill e permette di specificare le modalità di interazione con essa.
- Il *fulfillment*, che rappresenta il codice eseguito dalla skill in risposta ai comandi.

Modello di Interazione

Il modello di interazione di una skill è composto da diversi elementi, come descritto nella figura 4.4. L'*invocation name* indica il nome della skill e deve corrispondere esattamente a quello pronunciato dall'utente per invocarla. Nel caso del prototipo esso corrisponde a *'ASTRA Room Controller'*.

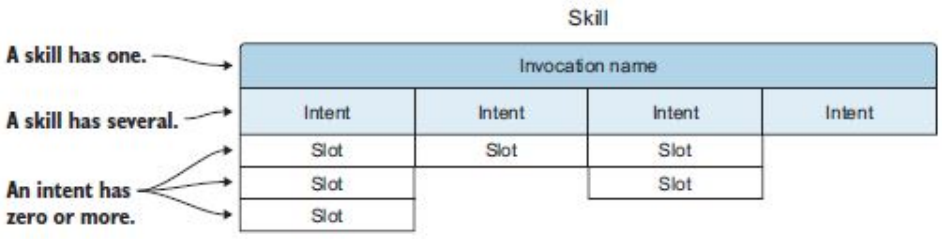

Figura 4.4: Modello di interazione di una Skill Alexa, tratto da [11]

Una volta aperta la skill è possibile richiedere al sistema diversi comandi, ad ognuno dei quali è associato un Intent ed alcune frasi di esempio per la

[1] `https://developer.amazon.com/alexa/console/ask`

sua richiesta. La suddivisione è effettuata in base alla tipologia di comando richiesto.

1 - *VisualisationRequestIntent*

Questo Intent permette di richiedere la visualizzazione di informazioni relative al paziente in cura. Presenta due parametri che è possibile specificare, *data_type*, che indica il tipo di dato che si vuole visualizzare (obbligatorio) e *position*, che indica in quale slot del display si intende visualizzare tale informazione.

Per consentire all'equipe medica di utilizzare il sistema in maniera fluida sono state inserite alcune frasi di esempio con cui richiamare questo Intent. Sono accettate le seguenti frasi:

- `[ Visualizza | Mostra | Fammi vedere ]` *`data_type`*
- `[ Visualizza | Mostra | Fammi vedere ]` *`data_type`* `del paziente`
- `[ Visualizza | Mostra | Fammi vedere ]` *`data_type`* `in` *`position`*
- `[ Visualizza | Mostra | Fammi vedere ]` *`data_type`* `del paziente in` *`position`*

data_type può assumere i valori di tutti i parametri che il sistema permette di gestire. In seguito all'analisi dei requisiti sono stati individuati 17 possibili valori. Ad ogni valore corrisponde un codice identificativo che è usato all'interno del sistema per riferirsi a quel tipo. Ognuno di essi presenta inoltre un elenco di sinonimi possibili. Ad esempio per richiedere la frequenza cardiaca, è possibile utilizzare *'frequenza cardiaca'*, *'polso'* o *'battito'*.

È possibile vedere l'elenco completo dei valori possibili e dei relativi sinonimi nella tabella 4.2.

position è un valore numero che indica l'indice della posizione nel display. Nel layout proposto nel prototipo tale valore assume un valore da 1 a 7, anche se l'utilizzo di altri numeri non genera errori ma viene corretto a livello di codice. Nel caso questo parametro non sia indicato il valore di default è 4.

2 - *MonitorRequestIntent*

Questo Intent permette di richiedere il monitoraggio di informazioni relative al paziente in cura. Questo comporta un aggiornamento periodico dei dati visualizzati in modo da avere sempre un valore aggiornato. I parametri che è possibile richiedere sono un sottoinsieme di quelli visualizzabili, in quanto per alcuni di essi non ha senso la procedura di monitoraggio essendo sono dati stabili. In particolare nella tabella 4.3 sono elencati tutti i parametri monitorabili. Può essere invocato tramite le seguenti frasi :

NOME	ID	SINONIMI
nome	patient_details	dettagli del paziente,dettagli,dati,id,informazioni,info,dati del paziente
frequenza cardiaca	heart_rate	polso,battito
pressione	blood_pressure	pressione arteriosa
saturazione	spO2	spo2
temperatura	temperature	
livello CO2	CO2_level	CO2
emogas analisi	ega	EGA
rotem	rotem	
RX torace	chest_rx	lastre torace, lastra torace
TAC	tac	
esami del sangue	blood_tests	esami
ECG	ecg	elettrocardiogramma
Condizioni Iniziali del paziente	patient_initial_condition	
tempo procedura	procedure_time	
tempo totale	total_time	tempo gestione trauma
tempo di arrivo	eta	tempo mancante all'arrivo,tempo mancante,eta,tempo all'arrivo
Trauma Tracker	tt_report	report di trauma tracker,report trauma tracker,report
Sacche di sangue utilizzate	used_blood_unit	sacche di sangue,unità di sangue utilizzate,unità di sangue ,unità utilizzate

Tabella 4.2: Parametri disponibili per la visualizzazione

NOME	ID	SINONIMI	PERIODO DI AGGIORNAMENTO
frequenza cardiaca	heart_rate	polso,battito	1 sec
pressione	blood_pressure	pressione arteriosa	5 sec
saturazione	spO2	spo2	1 sec
temperatura	temperature		30 sec
livello CO2	CO2_level	CO2	1 sec
emogas analisi	ega	EGA	1 sec
rotem	rotem		1 sec
tempo procedura	procedure_time		1 sec
tempo totale	total_time	tempo gestione trauma	1 sec
tempo di arrivo	eta	tempo mancante all'arrivo,tempo mancante,eta,tempo all'arrivo	1 sec
Trauma Tracker	tt_report	report di trauma tracker,report trauma tracker,report	5 sec
Sacche di sangue utilizzate	used_blood_unit	sacche di sangue,unità di sangue utilizzate,unità di sangue ,unità utilizzate	30 sec

Tabella 4.3: Parametri disponibili per il monitoraggio

- `Monitora` *`monitor_type`*
- `Monitora` *`monitor_type`* `del paziente`
- `Monitora` *`monitor_type`* `in` *`position`*
- `Monitora` *`monitor_type`* `del paziente in` *`position`*

3 - *DrugAnnotationRequestIntent*

Questo Intent permette di annotare nel sistema la somministrazione di un farmaco. Può essere invocato con il comando
`[Annota | Salva] somministrazione di` *`drug_name`*
drug_name può assumere qualsiasi valore. Questo Intent rappresenta un caso particolare in quanto richiede un dialogo per essere completato. Infatti una volta invocato sarà necessario specificare la quantità di farmaco, che verrà espressamente richiesta dal sistema una volta ricevuto il nome del farmaco.

4 - *DrugInfoVisualisationRequestIntent*

Questo Intent permette visualizzare la quantità totale somministrata di un

certo farmaco. Può essere invocato con il comando
`Visualizza quantità drug_name`
oppure
`Visualizza quanto drug_name è stato somministrato`

5 - *OperationAnnotationRequestIntent*
Questo Intent permette di annotare nel sistema l'inizio o la fine di una certa procedura. Può essere invocato con il comando
`[Annota | Salva] type manovra operation_name`

type indica l'inizio o la fine della procedura specificata.

6 - *StartActionIntent, EndActionIntent e PatientActionIntent*
Infine sono presenti alcuni Intent specifici che permettono di effettuare azioni specifiche che modificano lo stato del sistema. In particolare permettono di segnalare nell'ordine l'inizio e la fine della gestione di un trauma e l'arrivo del paziente in ospedale. Per invocarli possono essere utilizzate le seguenti frasi:

- *StartActionIntent*:
  `Paziente in arrivo, Crea nuovo paziente, Crea nuovo caso, Nuovo paziente, Inizia gestione trauma`
- *EndActionIntent*: `Termina gestione trauma`
- *PatientActionIntent*:
  `Annota arrivo paziente, Il paziente è arrivato, Paziente arrivato`

Fulfillment
Quando viene richiesto un Intent il codice relativo viene eseguito tramite AWS Lambda, che si occupa dell'esecuzione di codice on-demand come servizio, permettendo di sviluppare le Skill in un'ottica server-less. Ogni Intent necessita di un Handler che ne specifica le istruzioni specifiche. Nel caso specifico della Skill il codice eseguito da ogni Intent è simile in quanto il loro compito è quello di generare uno specifico comando ed aggiungerlo al sistema ASTRA tramite le API REST fornite in modo che possa essere eseguito. A titolo di esempio verrà presentato ed analizzato il codice relativo all'Intent *VisualisationRequestIntent*.

```
const Alexa = require('ask-sdk-core');
const Util = require('util.js');
const Escape = require('lodash/escape');
const requestHandler = require('then-request');
```

```
const COMMAND_SERVICE_PATH = 'http://<SERVICE-IP>:3010/api/commands';

const VisualisationRequestIntentHandler = {
    // Specify which Intent use this handler
    canHandle(handlerInput) {
        return
        Alexa.getRequestType(handlerInput.requestEnvelope) ===
            'IntentRequest' &&
        Alexa.getIntentName(handlerInput.requestEnvelope) ===
            'VisualisationRequestIntent';
    },

    async handle(handlerInput) {
        // The code executed by the handler
        ...
    }
}
```

Listato 4.2: Struttura dell'Intent Handler

Nel blocco di codice 4.2 viene mostrata la struttura generale dell'IntentHandler, in cui viene specificato a quali Intent l'Handler specifico fa riferimento. Il metodo `handle()` permette poi di specificare le istruzioni specifiche da eseguire quando esso viene richiamato. Nel caso specifico di *VisualisationRequestIntent* tale metodo svolge due compiti principali. Come prima cosa recupera i dati dalla richiesta dell'utente accedendo agli slot della richiesta ricevuta e li usa per costruire un oggetto JSON da inviare tramite una POST request a *RoomCommandManager*, il servizio che gestisce i comandi. Questa parte è visibile nel codice 4.3.

```
async handle(handlerInput) {
    // get slots data
   const slots = handlerInput
        .requestEnvelope
        .request
        .intent
        .slots;

    const data_type = slots.data_type.value;
    var position = slots.position.value;

    var id = slots.data_type.resolutions.resolutionsPerAuthority[0]
```

```
        .values[0].value.id;

    //If position is not defined by the user use a default value;
    if (! isFinite(position)){
        position = 4;
    }

    //generate Command info
    var data = {json: {
        type : "visualisation",
        category : "room",
        target : "display_sr",
        issuer : "alexa_ASTRA_controller",
        params : {
            value : id,
            position : position
        }
    }}

    // post data to CommandService
    await requestHandler('POST', COMMAND_SERVICE_PATH, data)
        .getBody('utf-8')
        .then(JSON.parse)
        .done(function (res) {
            console.log(res);
        });
    ...
}
```

Listato 4.3: Metodo `handle()` di *VisualisationRequestIntent* (parte 1)

Se questa operazione viene completata con successo la Skill prosegue con la costruzione della risposta da inviare all'utente come feedback. Tale operazione è mostrata nel blocco 4.4 e consiste nella costruzione di una frase di risposta, personalizzata in base alla richiesta, al quale è aggiunto un elemento audio. Questo è stato reso necessario poiché il timeout di default per le Skill è di 8-16 secondi, rendendo difficile l'utilizzo nel caso specifico in cui non è noto il tempo che può intercorrere tra una richiesta e la successiva, ma che presumibilmente è maggiore di questo intervallo.

```
const audioUrl =
    Util.getS3PreSignedUrl("Media/silence-long.mp3");

const speech = 'Richiedo la visualizzazione di ${data_type} del
    paziente. <audio src="${Escape(audioUrl)}"/> '

await requestHandler('POST', COMMAND_SERVICE_PATH, data)
    .getBody('utf-8')
    .then(JSON.parse)
    .done(function (res) { console.log(res); } );

return handlerInput.responseBuilder
    .speak(speech)
    .reprompt('Rimango in attesa di altri comandi. Tra poco sara
        necessario riaprire la Skill per continuare.')
    .getResponse();
```

Listato 4.4: Metodo `handle()` di *VisualisationRequestIntent* (parte 2)

Tramite questo elemento audio, silenzioso e della durata di 200 secondi, è invece possibile mantenere attiva la Skill per la gestione di richieste successive per questo periodo di tempo. Questa soluzione è stata tratta da [2].

4.1.3 ASTRA Trauma Controller

Al momento non è previsto nel prototipo l'implementazione di un gestore di input specifico per il trauma in quanto le operazioni relative ad esso possono essere integrate con il sistema di gestione di TraumaTracker, già implementato ed utilizzato dal Trauma Team nell'ambito della gestione dei traumi nella Shock Room.

4.2 Coda dei Comandi

I comandi generati dagli utenti in attesa che un agente li selezioni per l'effettiva esecuzione vengono mantenuti in una coda apposita. Per rendere efficiente e scalabile questo componente si è scelto di utilizzare un *Message Oriented Middleware* per la sua implementazione.

[2] https://github.com/alxiong/alexa-skill-with-arduino-webclient/blob/master/keep-alexa-wait-hacky.md

4.2.1 Message Oriented Middleware

Un *message-oriented middleware* (o *MOM*), come descritto in [2], è una infrastruttura software o hardware che permette di inviare e ricevere messaggi tra diversi componenti di un sistema distribuito. Il middleware crea un layer di comunicazione che permette di astrarre da dettagli di basso livello nella programmazione e permette a componenti software indipendenti, eventualmente basati su tecnologie e linguaggi differenti ed in esecuzione su piattaforme differenti, di interagire tra di loro. Un altro importante vantaggio offerto da questo tipo di middleware è la gestione della persistenza dei messaggi, rendendo possibile una comunicazione completamente asincrona tra i componenti.

4.2.2 Il Middleware RabbitMQ

RabbitMQ[3] è un *message-oriented middleware* (MOM) che implementa il protocollo *Advanced Message Queuing Protocol* (AMQP) e permette di gestire il modello di comunicazione Publish-Subscribe. RabbitMQ permette nello specifico lo scambio e il routing di messaggi tra *publisher* e *consumer* garantendo disaccoppiamento tra le entità interessate nella comunicazione. La parte server è scritta in linguaggio *Erlang*, mentre vengono fornite diverse librerie client a seconda del linguaggio specifico in cui vengono utilizzate.

Architettura

L'architettura di base di RabbitMQ è definita dalle seguenti entità:

- 1 o più *Producer*, processi che inviano messaggi.
- 1 *Exchange*
- 1 o più *Code* che contengono e mantengono i messaggi.
- 1 o più *Consumer*, processi che ricevono i messaggi.

L'idea di base di RabbitMQ è che i Producer non inviano messaggi direttamente alle code ma ad un Exchange. In questo modo i Producer non necessitano di conoscere a quale coda o a quale Consumer inviare il messaggio, garantendo disaccoppiamento tra le parti. Il ruolo centrale è quindi quello dell'entità Exchange, che riceverà i messaggi dai Producer e avrà il compito di effettuare il routing alla giusta coda. Per funzionare un Exchange deve essere collegato ad una o più code tramite un processo di *binding*. Le code si occupano inoltre della persistenza dei messaggi, che rimarranno salvati fintanto che

[3] `https://www.rabbitmq.com/`

non vengono prelevati da un Consumer e della gestione della concorrenza in modo che ogni messaggio sia elaborato una sola volta.

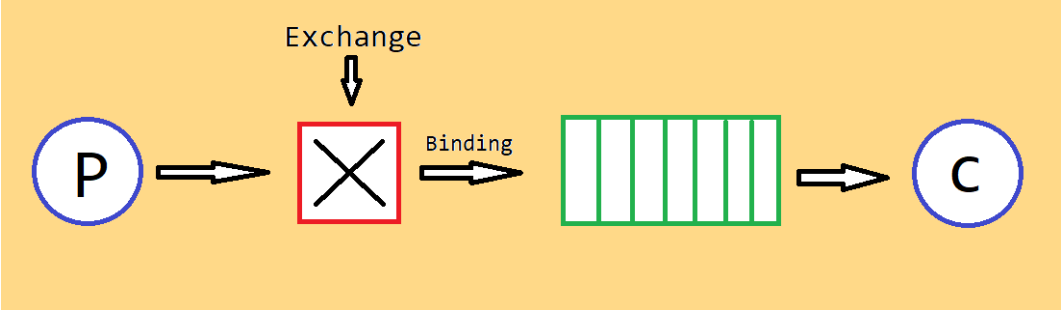

Figura 4.5: Struttura base di RabbitMQ, tratta da [4]

Modalità di Comunicazione

A seconda della tipologia di Exchange tramite questo middleware è possibile definire diverse modalità di comunicazione:

- ***Fanout***: è la tipologia di Exchange più semplice, fa il broadcast di tutti i messaggi che riceve a tutte le code a cui è collegato.
- ***Direct***: con questo tipo di Exchange un messaggio viene inoltrato a tutte le code la cui *chiave di associazione*, definita al momento del binding, corrisponde esattamente alla *chiave di routing* del messaggio.
- ***Topic***: Questo tipo di Exchange è il più potente e versatile e comprende i due precedenti. I messaggi per questo tipo di Exchange hanno come chiave di routing una lista di parole delimitate da punti. Come per Direct essi vengono inviati a tutte le code la cui *chiave di associazione* coincide. In questo caso è tuttavia possibile utilizzare delle specifiche wildcard per rendere più versatile il routing. In particolare * può essere sostituito da una singola parola, mentre # rappresenta 0 o più parole.

4.2.3 RabbitMQ nel Sistema ASTRA

Nel prototipo del sistema RabbitMQ è utilizzato per la gestione della comunicazione tra il servizio di gestione dei comandi (*CommandManager*) e le code dei comandi con le quali gli agenti vanno ad interagire, presenti sotto forma di artefatti. In questa ottica il servizio *CommandManager* rappresenta il processo Producer, mentre ognuno degli artefatti *CommandQueue* generati dagli agenti per la gestione dei comandi rappresentano un Consumer. Al fine di sfruttare al massimo le potenzialità del modello *publish-subscribe* ed

ottenere disaccoppiamento e parallelizzazione verrà usato il modello di comunicazione tramite topic offerto dal middleware. Ogni coda quindi effettuerà il binding tramite uno specifico exchange verso uno specifico insieme di messaggi. Questo meccanismo rende possibile ottenere un livello di granularità più fine nella selezione dei comandi e la creazione di più code in base alla tipologia di comando, andando ad aumentare il parallelismo e di conseguenza l'efficienza dell'intero sistema, trattandosi di processi per la maggior parte parallelizzabili. In particolare sono gestite dal prototipo 3 code distinte:

- *room_visualisation_commands_queue*, contenente tutti i comandi relativi alle richieste di visualizzazione di singoli valori. Essa fa un binding su *room.visualisation*.
- *room_monitor_commands_queue*, contenente tutti i comandi relativi alle richieste di monitoraggio. Essa fa un binding su *room.monitoring*.
- *room_action_commands_queue*, contenente tutti i comandi relativi alle richieste di azioni sulle risorse del sistema. Essa fa un binding su *room.action*.

Producer: *RoomCommandManager*

Il servizio *RoomCommandManager* si occupa della pubblicazione dei messaggi verso l'exchange. Questo è possibile sfruttando la libreria per NodeJS messa a disposizione da RabbitMQ, attraverso la quale viene definito il canale di comunicazione attraverso la quale avviene lo scambio di messaggi, come definito nel blocco di codice mostrato di seguito.

```
var amqp = require('amqplib/callback_api');

var exchangeName = 'room_command_exchange';
var channel;

amqp.connect('amqp://localhost', function(connect_error,
    connection) {
  if (connect_error) {
    throw connect_error;
  }

  connection.createChannel(function(channel_error, ch) {
    if (channel_error) { throw channel_error; }
    channel = ch;
```

```
      channel.assertExchange(exchangeName, 'topic', {durable:
          true});
    });
  });

  module.exports.publishActivity = function(msg, key) {
    channel.publish(exchangeName, key,
        Buffer.from(JSON.stringify(msg)));
    console.log("[x] Comando inviato a %s :", key);
  };
```

Consumer: *RoomCommandQueueArtifact*

Ogni istanza di questo artefatto rappresenta una coda per una determinata categoria di messaggi. Infatti a seconda della chiave di binding passata al costruttore essa riceverà messaggi solo per la categoria desiderata e li manterrà in una coda interna finché uno degli agenti non lo preleverà per l'esecuzione. Similmente al caso precedente l'implementazione sfrutta la libreria offerta dal middelware, questa volta in linguaggio Java, per specificare le operazioni da svolgere quando un messaggio è ricevuto. Il codice sottostante descrive nel dettaglio tale procedura.

```
 // Specify how to handle a new received message.
 DeliverCallback deliverCallback = new DeliverCallback() {
   public void handle(String consumerTag, Delivery delivery)
       throws IOException {

       String message = new String(delivery.getBody(), "UTF-8");
       JSONObject req = new JSONObject(message);

        //add command to local priority queue
        addCommandToQueue(req);
};

// Connect to the RabbitMQ queue
ConnectionFactory factory = new ConnectionFactory();
 factory.setHost("localhost");
 Connection connection;

try {
   connection = factory.newConnection();
```

```
    channel = connection.createChannel();
    channel.exchangeDeclare(EXCHANGE_NAME, "topic", true);
    channel.queueDeclare(queueName, true, false, false, null);
    // Connect to the exchange with the specified topics
    channel.queueBind(queueName, EXCHANGE_NAME, this.topic);
} catch (IOException e) {
    ...
} catch (TimeoutException e) {
    ...
}

 System.out.println(
     "[ " + queueName + " Queue ] Waiting for messages.");

 while(true) {
     try {
      await_time(TICK_TIME);
         channel.basicConsume(queueName, true, deliverCallback,
             new CancelCallback() {
         public void handle(String consumerTag) throws IOException
             { ... }
      });
     } catch (IOException ex){
         ex.printStackTrace();
     }
 }
```

4.3 Digital Twin e Microservizi

Gli agenti fin qui descritti rappresentano la parte attiva del sistema, ma per rendere il prototipo effettivamente funzionante ed utile in ambito pratico è necessario che tali entità possano interagire con l'ambiente fisico per ottenere dati ed informazioni o per agire direttamente su di esso. Proprio da questa possibilità di interazione si può vedere la potenza dell'integrazione tra il paradigma ad agenti e il modello dei Digital Twin per applicazioni che si interfacciano con l'ambiente fisico reale, come nel caso degli Smart Environment. Per implementare i Digital Twin nel prototipo dell'applicazione si è scelto di mappare la parte digitale come microservizi, garantendo in questo modo disaccoppiamento ed estensibilità al sistema. Il collegamento con la controparte fisica è stato invece garantito tramite l'utilizzo di WebSocket, in modo da permettere una comunicazione bidirezionale full-duplex tra le due componenti.

4.3.1 Stack MEAN: MongoDB, Expess, Angular, Node.js

Per l'implementazione dei microservizi si è scelto di utilizzare lo stack MEAN. Esso è uno stack software gratuito ed open source, particolarmente adatto allo sviluppo di applicazioni e servizi web. Come descritto in [20] questo stack è composto da 4 diversi framework, tutti basati sul linguaggio Javascript, con cui è quindi possibile sviluppare sia la parte client che la parte server. I componenti dello stack sono:

- *MongoDB* è un programma che permette di gestire database NoSQL, con cui vengono gestiti i dati in un formato JSON-like.
- *Express.js* è un framework basato su Node.js per lo sviluppo di applicazioni web.
- *Angular* è il framework per lo sviluppo del front-end.
- *Node.js* rappresenta infine il framework per lo sviluppo del server web.

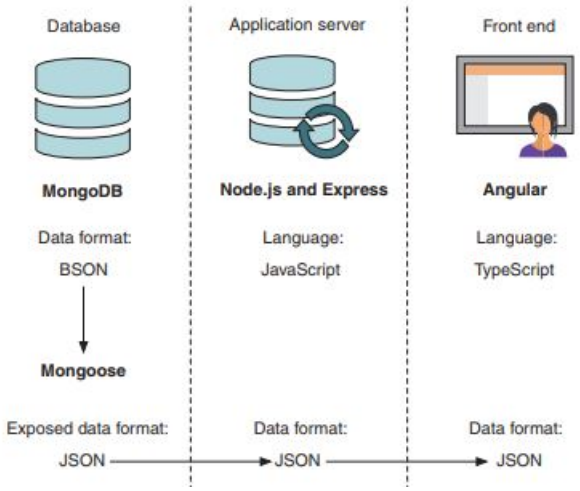

Figura 4.6: Struttura di un'applicazione web con lo stack MEAN. tratta da [20]

Node.js

Node.js rappresenta una piattaforma software con cui è possibile definire e costruire il proprio server web, su cui è possibile poi andare ad inserire la propria applicazione web. Esso gestisce infatti un server http tramite una libreria

built-in. Data questa particolarità esso rappresenta il componente fondamentale dello stack. L'utilizzo di questo framework ha diversi vantaggi, primo fra tutti la possibilità di utilizzare Javascript anche per la parte di back-end. Dal punto di vista implementativo questo framework risulta estremamente veloce e leggero, facendo un uso efficiente delle risorse di sistema. Questo è possibile poiché il server è basato su un singolo Thread, con cui il client interagisce solamente quando necessita di svolgere qualche operazione.

Node Package Manager

Un altro grande vantaggio di Node.js è la possibilità di utilizzare un Package Manager apposito, *npm*, che viene installato con il framework principale. Tramite esso è possibile installare e gestire pachetti con funzionalità aggiuntive a seconda delle necessità. Il numero di moduli disponibili è in costante crescita. Al momento della stesura di questo documento, secondo il sito `http://www.modulecounts.com/` tale numero ammonta a 1.316.330 moduli, con un incremento giornaliero medio di 859 moduli.

Express.js

Per aiutare nello sviluppo di un'applicazione web è stato inserito nello stack *Express.js*, o semplicemente *Express*, che gestisce in automatico diversi aspetti legati al routing e alla struttura dell'applicazione che si sta sviluppando. La sua feature principale è infatti quella di fornire un'interfaccia uniforme che dirige un certo URL ad uno specifico pezzo di codice. Questo risulterà fondamentale anche nell'utilizzo che ne viene fatto nel prototipo del sistema in esame in quanto permetterà di gestire in maniera semplice le API dei singoli servizi.

MongoDB

All'interno dello stack MongoDB si occupa dello storage delle informazioni. Esso gestisce database non relazionali e gestisce le informazioni sotto forma di documenti, usando il formato BSON (binary JSON). Ogni documento rappresenta una riga del database ed è composto da una serie di coppie chiave-valore. In questo modo il documento rappresenta sia la struttura che il dato stesso. Il framework fornisce inoltre la possibilità di gestire indici ed operazioni sulle informazioni. Il principale vantaggio è tuttavia quello di usare la stessa rappresentazione dei dati che viene usata all'interno degli altri componenti software dello stack. Tale framework risulta inoltre essere veloce e facilmente scalabile, adattandosi bene all'esigenza di utilizzo all'interno dei microservizi.

Angular

Angular è un framework per la creazione di interfacce per una applicazione web. Il vantaggio principale è quello di permettere di svolgere una parte più o meno consistente dell'elaborazione dalla parte del client, alleggerendo il carico sul server, che in quest'ottica è di fatto utilizzato principalmente per lo scambio e l'interazione con i dati e il database. Nel prototipo specifico sarà utilizzato principalmente per la gestione del Display fisico sulla quale visualizzare le informazioni generate dal sistema.

4.3.2 Collegamento con la Parte Fisica

Affinchè si possa parlare di Digital Twin è necessario che ci sia una parte digitale collegata con una controparte fisica e che esse condividano lo stato. Nel prototipo del sistema per rendere possibile questo collegamento si è scelto di utilizzare il meccanismo delle WebSocket, con le quali i due componenti di ogni Digital Twin possono mantenere aggiornato e coerente lo stato e scambiarsi informazioni.

4.3.3 Integrazione tramite CArtAgO

Uno degli aspetti fondamentali che si vuole testare con lo sviluppo di questa applicazione è come detto la possibilità di integrare il paradigma ad agenti con il modello dei Digital Twin per la gestione di ambienti aumentati. Affinché gli agenti possano interagire con i Digital Twin è necessario che venga creata un'interfaccia comune tra le due entità. Nell'applicazione in esame questo compito è svolto dai microservizi che rappresentano la parte digitale dei Digital Twin. Essi infatti sono collegati direttamente con la controparte fisica come descritto dal modello stesso, ma allo stesso tempo espongono delle API REST con il quale gli agenti sono in grado di interagire sfruttando le capacità messe in campo dagli artefatti prodotti con CArtAgO, descritto in 4.4.1. I diversi componenti sono quindi in grado di comunicare ed interagire tra di loro in maniera molto semplice sfruttando il protocollo HTTP.

Nei due paragrafi successivi è presentata l'applicazione dei microservizi e dei Digital Twin al prototipo del sistema ASTRA, mettendo in luce in particolare l'efficacia di questa operazione di integrazione.

4.3.4 Digital Twin nel Sistema ASTRA

Nel sistema sono presenti diverse entità che sono mappate come Digital Twin. Esse presentano una controparte digitale che prende la forma di un

microservizio con interfaccia REST collegata con la controparte fisica. Durante il periodo di sviluppo del prototipo non è stato possibile avere accesso ai veri e propri dispositivi fisici, per cui essa è realizzata come un processo computazionale che rappresenta l'oggetto fisico.

TAC

Questo Digital Twin rappresenta la macchina per la TAC presente al pronto soccorso, spesso usata per la realizzazione di esami diagnostici sul paziente della Shock Room.

Digital Assets: `TACSourceService`
Il servizio relativo alla parte digitale del Digital Twin è in ascolto sulla porta 3003. La sua interfaccia presenta solamente 3 metodi, tramite i quali è possibile aggiungere i risultati di una TAC, ottenere i risultati per un determinato paziente e ottenere lo stato della macchina fisica collegata. Quest'ultima operazione è possibile grazie al collegamento con la controparte fisica, garantito tramite una WebSocket.

Physical Assets: `TACSource`
Il processo che simula la parte fisica del Digital Twin è un'applicazione Angular che permette di interagire tramite la pagina web generata. Tramite essa, accessibile sulla porta 3002, è possibile modificare lo stato della macchina tra i tre disponibili (*"unavailable"*, *"in use"*, *"available"*) e richiedere il salvataggio di un certo risultato per un certo paziente, come se fosse la vera macchina (o il suo sistema di gestione).

Active Trauma

Digital Assets: `TACSourceService`
Il servizio relativo al trauma in corso è in ascolto sulla porta 3005. Questo servizio fornisce un gran numero di metodi relativi alla gestione delle informazioni sul trauma in corso. Essi possono essere fondamentalmente suddivisi in due categorie: i comandi relativi ai parametri vitali (temperatura, battito, pressione e saturazione) e quelli relativi ai dati generati e gestiti da Trauma Tracker.

Physical Assets: `Trauma Tracker`
Il trauma ovviamente non è un vero e proprio oggetto fisico presente nell'ambiente, e come tale non può essere colelgato direttamente. Tuttavia esso è

ben rappresentato in termini di entità computazionale tramite il sistema TraumaTracker, presentato in precedenza, che viene in questo modo integrato con il progetto ASTRA.

Display

Digital Assets: `DisplayService`

Il servizio relativo al display è in ascolto sulla porta 3001. Questo servizio fornisce diversi metodi con il quale visualizzare le diverse tipologie di dati che vengono elaborate dal sistema. Ogni volta che una richiesta verrà ricevuta varrà inviato al display fisico un messaggio tramite una webSocket contenente tutte le informazioni da stampare sullo schermo. Il servizio non si occupa della generazione del layout vero e proprio, lasciato come compito al Display, ma permette di specificare come e quali dati possono essere gestiti e visualizzati. Tramite esso è inoltre possibile gestire lo stato del display, andando a modificare il relativo layout. I tre stati possibili sono *idle*, *preH*, *active*

Physical Assets: `DisplayClient`

Sulla porta 3000 è disponibile il client che rappresenta l'applicazione web in esecuzione sul display fisico. Infatti per motivi di estensibilità e portabilità su dispositivi diversi si è scelto di implementare il display tramite una web application. Questo componente riceve messaggi direttamente dal servizio alla quale e collegato e a seconda dello stato e del messaggio ricevuto stampa a video le informazioni utilizzando una configurazione specifica per ogni tipologia di dato. Questo è stato possibile sfruttando la possibilità di iniettare dinamicamente componenti nelle pagine generate da Angular.

L'interfaccia utente del Display, seppur ancora in stato prototipale, è stata sviluppata direttamente partendo dai requisiti e dagli spunti offerti dall'equipe del Trauma Team, in base alle esigenze di visualizzazione che possono avere nel corso della gestione di un trauma. Le tre immagine sottostanti mostrano il display nei suoi tre diversi stati. Quando il display è in stato *idle* esso risulta inutilizzato e pertanto non mostra informazioni se non una pagina scura, visibile nella figura 4.7 .

Figura 4.7: Display in attesa

In stato *preH* viene visualizzato un insieme ridotto di informazioni man mano che esse vengono ricevute, quindi il layout del display risulta minimale, suddiviso in tre aree orizzontali. Esso è visibile nella figura 4.8.

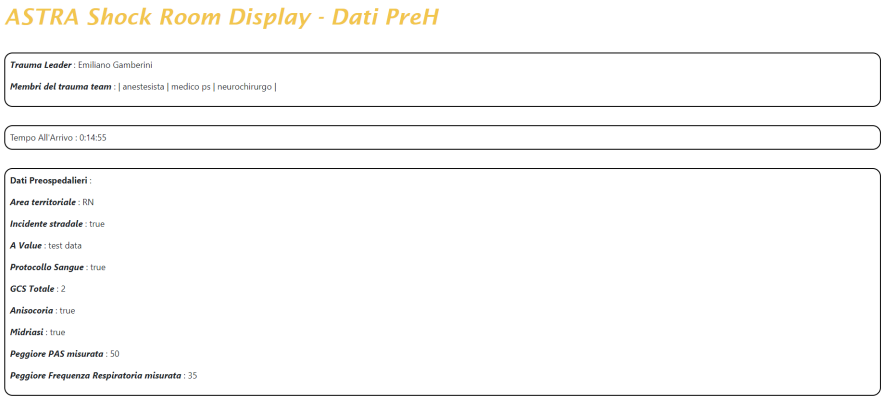

Figura 4.8: Display in modalità Trauma Preospedaliero

In stato **active** invece il layout è completo e presenta 8 aree in cui è possibile visualizzare informazioni, come visibile in 4.9. In questo particolare stato inoltre il sistema è in grado di stabilire se un dato può essere contenuto in una area limitata o necessita di maggiore spazio, come ad esempio per visualizzare un'immagine, adattando la visualizzazione di conseguenza, come visibile in 4.10

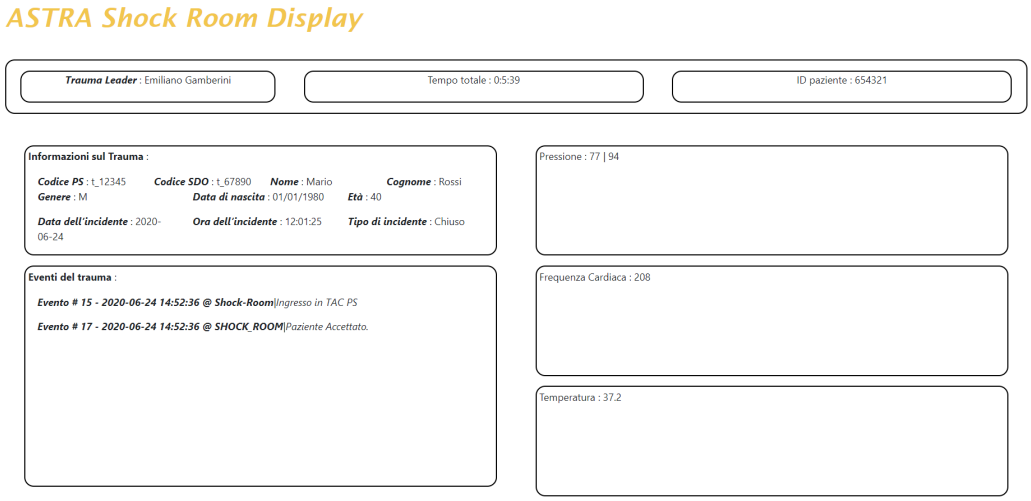

Figura 4.9: Display in modalità Trauma Attivo con dati multipli

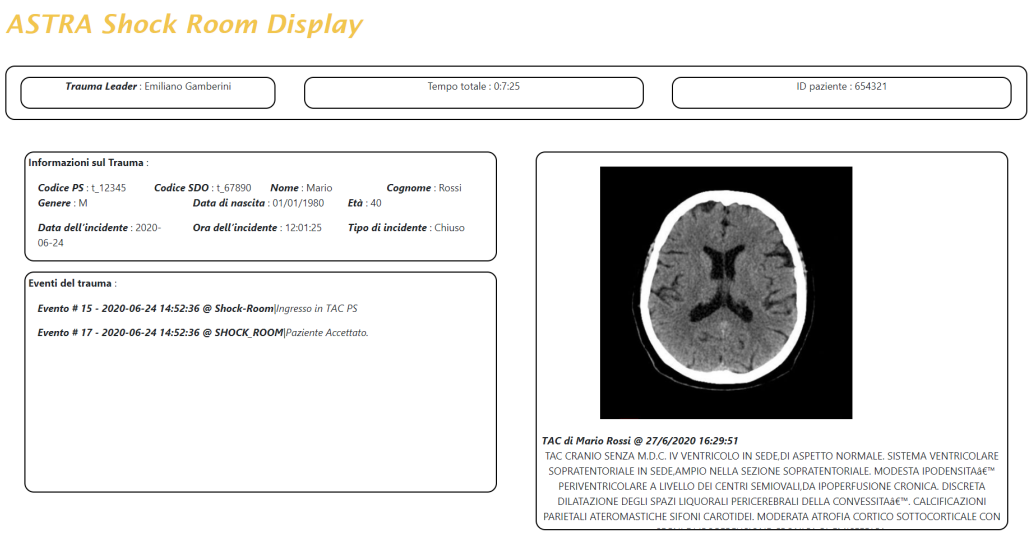

Figura 4.10: Display in modalità Trauma Attivo con immagine

RoomCommandManager

Questo servizio, attivo sulla porta 3010, permette ai gestori di input di interagire con il sistema. Esso infatti permette di aggiungere, modificare e cancellare comandi tramite i metodi della sua API, mostrata nel blocco di codice sottostante.

```
router.post('/commands', controller.postCommand);
router.delete('/commands', controller.clearCommandCollection);

router.get('/commands/pending', controller.getPendingCommand);
router.get('/commands/in_processing',
    controller.getProcessingCommand);
router.get('/commands/completed',
    controller.getCompletedCommand);
router.get('/commands/failed', controller.getFailedCommand);
router.get('/commands/:command_id', controller.getCommand);

router.put('/commands/:command_id', controller.updateCommand);
router.put('/commands/:command_id/status',
    controller.updateStatus);

router.delete('/commands/:command_id', controller.deleteCommand);
```

Ogni volta che viene effettuata la POST di un comando, oltre ad essere memorizzato, esso viene pubblicato tramite l'exchange di RabbitMQ nella coda adatta, in base al suo tipo, in modo che gli agenti possano elaborarlo.

4.4 Agenti ed Artefatti

4.4.1 Il Framework JaCaMo

JaCaMo [4] è un framework per lo sviluppo di sistemi multiagente che offre l'integrazione di tre diversi tool e linguaggi di programmazione adatti alla definizione degli agenti (*Jason*), del loro ambiente (*CArtAgO*) e della relativa organizzazione (*Moise*).

Programmazione di Agenti con Jason

Jason, descritto nel dettaglio in [8], è un linguaggio di programmazione basato sul linguaggio *AgentSpeak* che permette la definizione di agenti BDI. Tale linguaggio permette di specificare *Beliefs*, *Goals* e *Piani* di ogni agente, che verranno poi interpretati dal motore di esecuzione per produrre l'effettivo ciclo di ragionamento dell'agente. Ogni agente infatti agirà percependo costantemente l'ambiente, ragionando su come raggiungere i suoi goal in base ai piani presenti nella sua *plan library* e agendo di conseguenza modificando il suo ambiente.

Belief

I belief di un agente sono contenute nella sua *Belief Base* e sono espressi tramite una sintassi Prolog-like nella forma di predicati.

Goal

I goal rappresentano lo stato del mondo che l'agente vuole raggiungere tramite l'applicazione di piani. In Jason esistono due tipologie di goal:

- *Achievement Goal*: rappresentati con l'operatore `!` con il quale marcare un predicato. Essi sono la tipologia standard dei goal e rappresentano la volontà di rendere vero il relativo predicato.

- *Test Goal*: rappresentati con l'operatore `?`, sono utilizzati per ottenere informazioni da recuperare dalla belief base dell'agente stesso.

[4] `https://github.com/jacamo-lang/jacamo`

Piani

I piani rappresentano il know-how dell'agente, le operazioni che può svolgere sull'ambiente. Essi sono eseguiti in seguito ad un cambiamento nei Goal o nella Belief Base dell'agente. Essi sono espressi nella forma

```
triggering_event : context <- body .
```

Ogni piano viene eseguito in seguito allo scatenarsi di un *triggering_event*, che può essere una *Aggiunta* o una *Eliminazione*, soltanto se le condizioni ambientali, espresse in *context*, sono favorevoli. Se queste condizioni sono verificate il piano può essere scelto per l'esecuzione, in cui viene eseguita la sequenza di *formule* specificate in *body*. Esistono 6 tipi di formule:

- *Azioni*: rappresentano le operazioni che l'agente è in grado di svolgere sull'ambiente. Espresse come predicati.
- *Achievement Goal*: permettono di aggiungere un goal all'insieme dei goal dell'agente. Espresse con l'operatore `!`.
- *Test Goal*: permettono di aggiungere un Test Goal per recuperare informazioni necessarie all'esecuzione del piano. Usano l'operatore `?`.
- *Mental Notes*: permette la creazione di Belief per aggiungere informazioni alla base di conoscenza. Espresse con l'operatore `+`. Esse vanno espressamente eliminate quando non più necessarie con l'operatore `-`.
- *Azioni interne*: rappresentano operazione eseguite internamente all'agente che non hanno effetti sull'ambiente. Usano l'operatore `.`
- *Espressioni*: rappresentano espressioni booleane, usate per controllare il flusso del piano. Se esse non sono verificate infatti faranno fallire il piano stesso.

Programmazione di Artefatti con CArtAgO

CArtAgO (Common Artifact infrastructure for Agent Open environment), come descritto in [33], è una infrastruttura per la programmazione ed una piattaforma per l'esecuzione di ambienti Artifact-Based in un MAS. Esso si basa sul modello concettuale *A&A*, descritto in precedenza. In particolare la piattaforma fornisce:

- Una API Java-based per la programmazione di artefatti.
- Una API per agenti che fornisce un insieme di azioni per la creazione e l'interazione con gli artefatti.

- Un ambiente di esecuzione che supporta l'esecuzione e la gestione del ciclo di vita di artefatti e workspace.

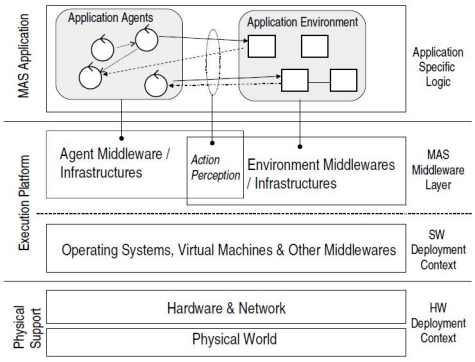

Figura 4.11: CArtAgO Abstract Architecture Model, tratto da [35].

Artefatti

Gli artefatti descritti in CArtAgO sono oggetti Java che espongono una *Usage Interface*, una lista di operazioni di cui un agente può richiedere l'esecuzione. Tali operazioni possono modificare lo stato dell'artefatto ed eventualmente produrre *eventi osservabili* che possono essere percepiti dagli agenti presenti nell'ambiente. Inoltre gli artefatti possono presentare uno *stato osservabile*, ossia una parte dello stato interno dell'agente che gli agenti possono osservare. In base al suo stato osservabile un artefatto può modificare dinamicamente l'insieme delle operazioni presenti nella usage interface.

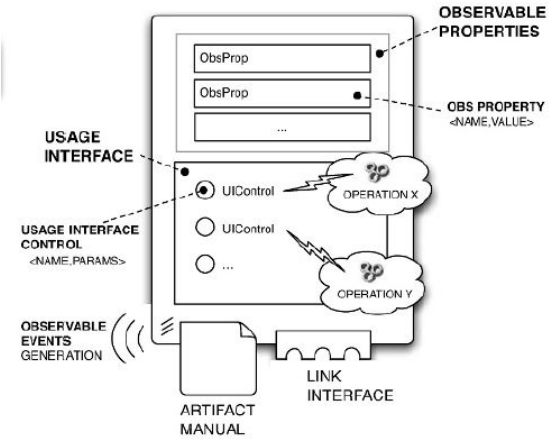

Figura 4.12: Rappresentazione astratta di un artefatto CArtAgO, tratto da [33].

In particolare ogni artefatto è un oggetto che estende `cartago.Artifact` e può presentare:

- *Variabili di stato*: Rappresentano lo stato interno, non osservabile. Dichiarati come normali variabili o oggetti Java.

- *Proprietà osservabili*: Rappresentano lo stato osservabile dell'artefatto. Sono coppie chiave-valore definite tramite il metodo `defineObsProperty()`, accessibili tramite primitive Get e Set.

- *Operazioni*: Rappresentano i metodi pubblici dell'artefatto e sono quindi le operazioni che fanno parte della sua Usage Interface. Sono metodi annotati con `@Operation` e ritornano obbligatoriamente `void`. Esse possono modificare lo stato dell'artefatto ed emettere aventi osservabili tramite la primitiva `signal()`

- *Operazioni interne*: Rappresentano i metodi privati dell'artefatto e come tali non sono direttamente richiamabili dagli agenti. Questi metodi sono annotati con `@Internal_Operation`

- *Operazioni linkate*: Rappresentano i metodi che possono essere richiamati da altri artefatti, ossia quelli che compongono la *Link Interface* dell'artefatto. uesti metodi sono annotati con `@Link`

Agenti ed Artefatti

CArtAgO offre agli agenti diverse azioni per la gestione degli artefatti e dei relativi workspace. Quelle fondamentali per l'utilizzo di un artefatto sono le seguenti 3:

- `makeArtifact()`: permette la creazione dell'istanza di un artefatto. Sono infatti gli agenti stessi ad occuparsi della creazione degli artefatti che popolano l'ambiente.

- `lookupArtifact()`: permette di identificare un artefatto per l'utilizzo.

- `focus()`: permette di osservare un artefatto per poterne visualizzare lo stato osservabile e gli eventi osservabili.

Per l'esecuzione di una operazione con un artefatto sarà sufficiente chiamarla come una normale operazione. L'agente la svolgerà con l'artefatto adatto. Per ulteriori informazioni si rimanda a [35], [33] e al sito web del progetto [5].

[5] `http://cartago.sourceforge.net/`

Gestione delle Organizzazioni con Moise

Il terzo componente di JaCaMo è Moise [6], un sistema per la gestione delle organizzazioni e l'orchestrazione di agenti all'interno di un MAS. In questa versione del progetto non sarà utilizzata.

4.4.2 JaCaMo nel Sistema ASTRA

Agenti Jason

Nel sistema sono implementati un insieme di agenti che interagiscono tra loro per garantire il completamento delle richieste degli utenti. L'agente principale, *Room Assistant*, che a livello logico è un unico agente è stato scomposto in più agenti in modo da semplificare la leggibilità e la struttura del codice. Questa scelta permette inoltre di migliorare le prestazioni del sistema, garantendo la possibilità di una parallelizzazione nell'esecuzione in quanto ognuno degli agenti è libero di agire in maniera autonoma.
Esso è quindi suddiviso in:

- `roomVisualisationAgent` che si occupa di tutti i comandi relativi alla visualizzazione di informazioni
- `roomMonitoringAgent` che si occupa del monitoraggio, tramite visualizzazione periodica, di informazioni

`roomVisualisationAgent`
L'agente Jason che si occupa della visualizzazione delle informazioni quando viene inizializzato va a cercare una serie di artefatti che gli permettono di interagire con i Digital Twin e gli altri componenti presenti nel sistema. Tra essi è presente l'artefatto relativo alla coda che contiene i comandi relativi alla visualizzazione di informazioni. Esso riceverà quindi ogni comando che viene pubblicato tramite publish-subscribe con l'argomento *room.visualisation*.

La presenza di un nuovo comando viene percepita dall'agente tramite una proprietà osservabile dell'artefatto stesso. Quando questo avviene l'agente prova a prendere in carico l'esecuzione del comando, aggiornando lo stato del comando stesso. Questa è una ulteriore modalità di verifica per evitare che un comando possa essere eseguito più di una volta.

[6] `http://moise.sourceforge.net/`

```
+ last_pending_command(Command)
   <-   -+current_command(Command);
      acceptCommand [artifact_id(QueueId)].

+ accepted_work(Command, Id, Type, DataType, Target, Position)
   <-   .println("Command Accepted : " , Id);
      !processCommand(Id, Type, DataType, Target, Position).
```

Una volta accettato può iniziare l'esecuzione effettiva, tramite l'invocazione di un piano adeguato. Tutti i comandi di visualizzazione per essere completati necessitano di richiedere una o più informazioni e poi di richiederne la visualizzazione sul display identificato come bersaglio. A seconda del tipo di dato richiesto sono disponibili diversi piani tra cui l'agente può scegliere, in modo che possa interagire con l'artefatto adatto a seconda della tipologia. Anche la visualizzazione è specifica per tipologia di dato, per cui, come è possibile vedere nel blocco di codice sottostante, sono presenti diversi piani relativi alla visualizzazione.

```
+! processCommand(CommandId, "visualisation", DataType, Target,
    Position)
   <-   .send(roomMonitoringAgent, tell, visualise_on(Position));
       !requestData(CommandId, DataType, Target, Position).

/* ----------- DATA REQUEST ----------- */

+! requestData(CommandId, "blood_pressure", Target, Position)
  <- .println("Searching for blood_pressure data");
     getBloodPressureValue(Value)
         [artifact_id(VitalParameterSourceId)];
     !displayData(CommandId, "blood_pressure", Value, Target,
         Position).

    // ... handle all supported data types

/* All other Data */
+! requestData(CommandId, DataType, Target, Position) :
    current_command(Command)
  <- .println("Command not handled, refusing ...")
     refuseCommand(Command) [artifact_id(QueueId)].

-! requestData(CommandId, DataType, Target, Position) :
```

```
      current_command(Command)
    <-    .println("Error ! Data unavailable")
       setErrorOnCommand(Command) [artifact_id(QueueId)].

  /* Display data in Target display using related artifact*/

  +! displayData(CommandId, DataType, Value, Target, Position) :
      DataType = "blood_pressure" | DataType = "spO2" | DataType =
      "heart_rate" | DataType = "temperature"
    <- .println("Got ", DataType, " value : ", Value);
       .println("Displaying ", DataType, " data " , Value, " in ",
          Target, " on ", Position);
       showBiometricData(Value, DataType, Position)
          [artifact_id(Target)]
       .println("Display request completed successfully");
       ! completeCommand.

    ...

  -! displayData(CommandId, DataType, Value, Target, Position) :
      current_command(Command)
    <- .println("Error on command display");
       setErrorOnCommand(Command) [artifact_id(QueueId)].
```

Una volta completata la visualizzazione con successo viene completata l'operazione aggiornando il comando per segnalarne il completamento.

```
  +! completeCommand : current_command(Command)
    <-    completeCommand(Command) [artifact_id(QueueId)]
       .println("Command Handling completed, waiting for a new
          command.").
```

`roomMonitoringAgent`

L'agente che si occupa del monitoraggio è simile a quello che gestisce la visualizzazione, e raccoglie tutti i comandi pubblicati con l'argomento *room.monitoring*.

La differenza fondamentale risiede nella gestione delle richieste di informazioni, che dovranno essere ripetute nel tempo in modo da visualizzare una versione aggiornata dei dati. Il periodo di aggiornamento è gestito direttamente dall'agente, che mette in attesa l'esecuzione per un certo periodo prima di inviare una nuova richiesta e richiamare lo stesso piano in modo da conti-

nuare con il monitoraggio. Questa operazione è mostrata nel blocco di codice sottostante.

```
+! monitorBloodPressure(Target, Position) :
    monitoring("blood_pressure")
  <-    getBloodPressureValue(Value)
       [artifact_id(VitalParameterId)];
      showBiometricData(Value, "blood_pressure", Position)
         [artifact_id(Target)];
      .wait(1000);
      !! monitorBloodPressure (Target, Position).
```

Nel caso venga richiesto di utilizzare uno spazio in cui è già in corso un monitoraggio è inoltre necessario sospendere tale operazione per poter visualizzare correttamente l'informazione richiesta. Questo può avvenire sia per un nuovo monitoraggio sia per una singola visualizzazione. In questo caso l'agente riceverà un messaggio dall'agente addetto alla visualizzazione, indicante la posizione da liberare.

```
+ visualise_on(Position) : use_position (Position, Type)
  <- .println("Received message from Visualisation Agent. Want
     to free position " , Position);
     .println("Position ", Position , " used by " , Type)
     ! freePosition(Position, Type);
     -visualise_on(Position).
```

`traumaEventManagerAgent`

Questo agente garantisce la gestione degli eventi relativi al trauma in corso. Tramite i suoi piani esso è in grado di gestire in maniera automatica il susseguirsi degli eventi, richiedendo la visualizzazione di determinate informazioni in momenti specifici ed inoltre modificando la struttura del layout del display conseguentemente all'avanzare della gestione del trauma. Infatti in ogni gestione di un trauma si identificano diverse fasi, in cui è necessario visualizzare informazioni diverse:

- **Fase 1 - Paziente in arrivo**: In questa fase prima fase il Trauma Team viene allertato dell'arrivo di un paziente. Durante essa sarà utile mostrare informazioni relative alla composizione del Trauma Team stesso, sul

tempo di arrivo stimato e sulle informazioni che man mano arrivano dalla fase preospedaliera (preh).

- **Fase 2 - Paziente in ospedale**: In seguito all'arrivo del paziente sarà necessario visualizzare un diverso insieme di informazioni che vengono man mano raccolte, tra cui le informazioni relative al trauma, le condizioni iniziali del paziente e la sequenza degli eventi man mano che vengono registrati.

Tutte queste informazioni vengono registrate dal sistema TraumaTracker, e vengono accedute da questo agente tramite un artefatto che rappresenta il punto di accesso verso tale sistema. Quando questo agente viene creato crea una istanza dell'artefatto e rimane in ascolto dello stato del trauma in corso tramite una proprietà osservabile dello stesso, che può assumere 3 valori: *arriving*, *active*, *done*. In base ad essi richiede la modifica del layout del display e va a visualizzare una serie di informazioni di default, come è possibile vedere nel blocco di codice sottostante. Questa operazione genera un po' di latenza, dovuta in parte anche al periodo di aggiornamento del valore della proprietà osservabile, fissato ora a 5 secondi per bilanciare tra reattività e carico della rete.

```
+trauma_status(Status) : Status = "arriving"
  <- .println("Trauma Status : Arriving patient");
     setDisplayStatus("preH") [artifact_id(DisplayId)];
     !monitorPreHETA
     !monitorPreHInfo.

+trauma_status(Status) : Status = "active"
  <- .println("Trauma Status : patient is arrived");
     setDisplayStatus("active") [artifact_id(DisplayId)];
     setPatientArrived [artifact_id(TimeMonitorId)];
     !showInitialData
     !monitorActiveTraumaTime
     !monitorActiveTraumaEvent.

+trauma_status(Status) : Status = "done"
  <- .println
      ("Trauma Status : Patient trauma handling completed");
     setDisplayStatus("idle") [artifact_id(DisplayId)];
     completeTraumaHandling [artifact_id(TraumaId)].
```

Questo agente si occupa inoltre, come è visibile nel codice, del monitoraggio del tempo trascorso e delle informazioni che arrivano durante le operazioni dal servizio TraumaTracker. Questi dati in particolare vengono aggiornati ogni 5 secondi.

Altri Agenti nel MAS

Nel sistema sono inoltre presenti altri agenti che si occupano di aspetti specifici:

- `systemInitAgent`: Questo agente si occupa della creazione degli artefatti comuni, andando a popolare l'ambiente con tutti gli elementi necessari agli altri agenti per l'esecuzione dei loro piani
- `systemStatusMonitorAgent`: Questo agente monitora lo stato degli artefatti, notificando eventuali errori e cambiamenti di stato negli artefatti, ad esempio nel caso in cui uno dei Digital Twin diventi irraggiungibile.
- `errorHandlerAgent`: Questo agente gestisce gli errori sui comandi ricevuti, gestendo la strategia per la gestione degli stessi. In particolare se un comando ha un errore nel corso della sua esecuzione esso viene segnalato e rimesso nella coda dei comandi in attesa in modo da poter riprovare la sua esecuzione. Nel caso l'errore si ripeta dopo 5 tentativi il comando viene definitivamente rifiutato.

Artefatti CArtAgO

All'interno del MAS sono inoltre definiti un certo insieme di artefatti, utilizzando il modello proposto da CArtAgO, per permettere il collegamento tra gli agenti ed i Digital Twin.

`RoomCommandQueueArtifact`

Questo artefatto contiene la coda dei comandi ricevuti dagli utenti ed è un caso particolare perchè non è collegato direttamente ad un Digital Twin. Esso infatti rappresenta un Consumer, collegato tramite publish-subscribe, al servizio *RoomCommandManager*. Questo gli permette di ricevere i messaggi tramite il topic specificato al momento della creazione dell'artefatto.

Ogni artefatto mantiene al suo interno 3 code con priorità, nelle quali vengono suddivisi i comandi in attesa, quelli che presentano errori risolvibili e quelli rifiutati. Per ognuna di queste code presenta una proprietà osservabile, che permette di accedere al messaggio in cima alla coda stessa. Questa particolarità permette al singolo artefatto di essere utilizzato da più di un agente contemporaneamente.

Le operazioni esposte da questo artefatto permettono di gestire il ciclo di vita di un comando:

- `acceptCommand()` permette di richiedere al CommandService la presa in carico di un comando per poter iniziare l'effettiva esecuzione. Questo avviene quando un agente lo seleziona dalla coda.
- `completeCommand()` permette di contrassegnare un comando come completato, questo avviene quando l'esecuzione è stata completata con successo.
- `setErrorOnCommand()` permette di segnalare un errore relativo al comando. In questo caso il suo stato non viene aggiornato nel servizio in quanto lo si ritiene un errore risolvibile e vengono messe in campo delle strategie per poterlo risolvere.
- `refuseCommand()` permette di contrassegnare un comando rifiutato. Questo avviene per due motivi principali:
  - Il sistema non supporta il comando richiesto, e quindi non è in grado di completarlo in questo stadio di sviluppo.
  - Il comando ha raggiunto il limite consentito di tentativi ed è stato quindi rifiutato, ritenendo impossibile il suo completamento a causa di qualche genere di errore.

Nel prototipo del sistema vengono gestite 3 istanze di questo artefatto, permettendo di suddividere in maniera più fine i comandi destinati agli agenti relativi alla stanza. In particolare i tre topic gestiti sono *room.visualisation*, *room.monitoring* e *room.action*, in modo da poter gestire tutti i possibili comandi rivolti verso la stanza.

`ActiveTraumaArtifact`

Questo artefatto rappresenta l'interfaccia verso il Digital Twin del trauma in corso. Tramite esso è possibile accedere alle informazioni relative al trauma che vengono generate dal sistema Trauma Tracker. Questo artefatto è collegato tramite una webSocket al realtivo Digital Twin, in modo da poter essere notificato quando viene generato un nuovo trauma. Da questo momento esso inizia a monitorare lo stato del trauma in corso esponendolo tramite la proprità osservabile `trauma_status`. In questo modo è possibile per gli agenti conoscere in quale fase della gestione si trova il paziente traumatizzato.

Le operazioni esposte permettono di accedere alle informazioni sul trauma, che coincidono con quelle generate e gestite dal sistema Trauma Tracker integrato, come visto in precedenza, al sistema ASTRA tramite il servizio

`ActiveTraumaService`. Tramite questo artefatto è quindi possibile accedere alle seguenti informazioni:

- Composizione del Trauma Team (tramite il metodo `getTraumaTeam`).
- Dati preospedalieri (tramite il metodo `getPrehInfo`).
- Informazioni sul trauma (tramite il metodo `getTraumaInfo`).
- Condizioni iniziali del paziente (tramite il metodo `getPatientInitialCondition`).
- Eventi del trauma (tramite i metodi `getEventList` e `getEvent`).

Ogni istanza di questo artefatto è in grado di gestire contemporaneamente un unico trauma.

`TACArtifact , VitalParameterArtifact e MockDataSourceArtifact`
Questi artefatti rappresentano i rispettivi Digital Twin e presentano una serie di operazioni che permettono di accedere ai dati specifici. Inoltre ognuno di essi presenta una proprietà osservabile che permette di conoscere lo stato del servizio alla quale sono collegati, per poter agire in caso di errori.

`TimeMonitorArtifact`
Questo artefatto permette di gestire lo scorrere del tempo all'interno del sistema. Ha un duplice compito, quello di mostrare il tempo restante al momento stimato dell'arrivo del paziente e, successivamente al suo arrivo, misurare il tempo trascorso nella Shock Room nella gestione del trauma. Queste informazioni possono essere ottenute tramite i metodi `getETA` e `getArrivalTime`.

`DisplayArtifact`
Questo artefatto è collegato al Display fisico tramite il suo Digital Twin e ricopre un ruolo fondamentale in quanto è necessario per poter completare con successo i comandi impartiti al sistema. Esso presenta una serie di operazioni specifiche per visualizzare le diverse tipologie di dati elaborati dal sistema. Inoltre ha un'operazione, `setDisplayStatus`, che permette di modificare lo stato del display ed il relativo layout. Questa sarà particolarmente utile per permettere di visualizzare informazioni specifiche a seconda dello stato corrente della gestione del trauma in corso.

Capitolo 5

Valutazione del Prototipo

In seguito allo sviluppo di una prima versione del prototipo sono stati effettuati alcuni test relativi alle performance e si è provveduto ad effettuare una prima valutazione del risultato prodotto con alcuni membri del Trauma Team, in modo da ottenere da loro alcuni primi feedback sul progetto. Questo ultimo test con gli esperti del dominio funge in particolare da validazione per questo lavoro di tesi.

5.1 Test delle Performance

Al fine di valutare le performance del sistema è stato sviluppato un semplice componente che permette di visualizzare informazioni e statistiche sullo stato dei comandi impartiti dagli utenti al sistema. Tramite esso è possibile conoscere:

- Il numero totale dei comandi richiesti
- Il numero di comandi in attesa di essere elaborati, in elaborazione, completati con successo e falliti
- Il tempo medio e massimo dalla ricezione all'inizio dell'elaborazione, dalla ricezione al completamento o dalla ricezione al fallimento.

Per rendere più veritiero ed affidabile il risultato esso non viene calcolato per singola sessione ma sul totale dei comandi dall'ultimo reset. In questo modo è possibile valutare un intervallo di tempo più lungo ed un numero di comandi significativo.

5.1.1 Risultati

Il test si è svolto su un insieme di comandi generati appositamente simulando un normale utilizzo del sistema fino ad ottenere un numero sufficientemente significativo di comandi per poter considerare affidabile il risultato calcolato.

Al momento della verifica risultano impartiti 163 comandi, provenienti da sorgenti miste (applicazione Android e assistente vocale). Il tempo medio per l'accettazione di un comando da parte del sistema una volta che esso è posto in una delle code è di 0.563 secondi, con un picco massimo di 1.051 secondi.

Di questi 139 risultano completati, con un tempo medio tra l'accettazione e il completamento di 0.592 secondi e un massimo di 1.626 secondi. 24 comandi risultano invece rifiutati. Questo numero è dato dal fatto che alcune informazioni che è possibile richiedere non sono ancora supportate dal sistema e per questo tali comandi vengono scartati. Per essi il tempo medio risulta molto inferiore in quanto l'agente li scarta senza procedere ad ulteriori operazioni. Il tempo medio tra l'accettazione e il rifiuto è di 0.015 secondi, con un massimo di 0.028. Nel caso invece vengano riscontrati errori nel completamento di un comando i tempi si allungano inevitabilmente in quanto il sistema effettua diversi tentativi per completare le operazioni necessarie prima di rifiutare il comando. Effettuato un test ad hoc con 10 comandi si vede infatti che la durata media è di 1.049 secondi con un picco di 2.048 secondi.

Questo risultato mostra come il sistema sia comunque efficiente ed adatto ad un utilizzo in un contesto di lavoro reale, in cui è necessario che i risultati siano visualizzati nel più breve tempo possibile. È importante notare che questo test non considera il tempo complessivo per l'esecuzione, in quanto è presente un intervallo da quando l'utente genera il comando a quando questo arriva al command manager del sistema che al momento non viene considerato, ma anche valutando questo periodo ed una eventuale latenza dovuta alla comunicazione tramite la rete il risultato rimane comunque pienamente accettabile.

5.2 Validazione con il Trauma Team e Valutazione del Risultato

Al termine dello sviluppo del prototipo è stata organizzata una dimostrazione delle funzionalità del sistema con due membri del Trauma Team, anestesisti-rianimatori dell'U.O. di Anestesia e Rianimazione Ospedale Bufalini di Cesena, afferenti al Trauma Center di AUSL Romagna, la dottoressa Costanza Martino ed il dottor Emiliano Gamberini.

5.2.1 Test di Validazione

Nel corso di tale dimostrazione sono state descritte e mostrate le diverse funzionalità del sistema sviluppato e la sua integrazione con il sistema TraumaTracker. La parte relativa all'integrazione è stata mostrata per prima, facendo vedere come il passaggio da una fase all'altra del trattamento del paziente corrisponde ad una modifica dello stato del display e delle informazioni in esso riportate. Tramite l'agente preposto è infatti possibile gestire in automatico tale tipo di eventi.

La dimostrazione è iniziata mostrando la struttura del sistema e il relativo output con lo schermo in attesa. In seguito alla simulazione dell'arrivo di un paziente traumatizzato, è stata mostrata l'attivazione e il relativo cambio di layout del display, nel quale sono mostrati i dati relativi al trauma team e dati preospedalieri, man mano che essi sono stati inseriti nel sistema tramite TraumaTracker. Successivamente è stato simulato l'arrivo del paziente in Shock Room, con conseguente cambiamento del layout del display. In questa modalità esso mostra in automatico le informazioni sul trauma e la lista degli eventi, oltre a lasciare spazio per la visualizzazione di informazioni a richiesta. A questo punto della dimostrazione sono state testate le funzionalità relative a richieste direttamente provenienti dagli utenti; sono stati quindi mostrati i diversi tipi di dati che è possibile richiedere, sia testuali che sotto forma di immagine (come nel caso della TAC) sia nella modalità di visualizzazione che di monitoraggio. Inizialmente tutti i comandi sono stati impartiti tramite assistente vocale, per mostrare il funzionamento dello stesso e presentare il vantaggio che tale approccio può offrire in termini di interfaccia con l'utente, per poi terminare la dimostrazione stessa mostrando che è possibile ottenere lo stesso effetto tramite il dispositivo di input fisico che esegue l'applicazione Android ASTRA Room Controller. In questo modo è stato possibile raccogliere qualche feedback anche relativamente alle interfacce sviluppate. Un'ulteriore aspetto di validazione interessante, soprattutto relativamente all'interfaccia vocale, sarebbe stata infatti la prova diretta del sistema da parte dei due esperti al fine di valutarne l'usabilità, che tuttavia non si è potuta svolgere nelle tempistiche richieste per cause di forza maggiore dovute alla pandemia di Covid-19, in quanto la dimostrazione organizzata non si è potuta svolgere in presenza.

5.2.2 Risultato della Validazione

Al termine della dimostrazione sono state raccolte le prime impressioni degli esperti del dominio, che si sono detti soddisfatti dal risultato ottenuto,

seppur in uno stato ancora prototipale e migliorabile sotto diversi aspetti, e lo hanno ritenuto in linea con i requisiti espressi nella fase iniziale del progetto.

In base ai loro commenti gli aspetti più critici riguardano la parte relativa all'interfaccia grafica e alla presentazione dei dati mostrati dal display, che può essere migliorata per rendere più leggibili e fruibili le informazioni, mentre non sono stati rilevati problemi significativi nella parte relativa alla gestione interna dei dati. Anche relativamente all'interfaccia vocale è stato fatto notare come sia necessario semplificare ulteriormente il funzionamento in modo da rendere meno verboso e più veloce il dialogo con l'assistente, ma che come prima versione risulta accettabile. Per fare questo sarà utile in futuro un confronto approfondito e puntuale con gli esperti del dominio, in modo da trovare direttamente con loro i termini più adatti per fare riferimento ai vari comandi ed addestrare il modello dell'assistente vocale di conseguenza.

I medici stessi hanno inoltre suggerito diversi miglioramenti possibili relativi all'interfaccia grafica, mettendo in luce come alcuni dei dati al momento visualizzati non riescono a mostrare abbastanza informazioni o lo fanno in maniera non ottimale. Per esempio è stato suggerito di modificare la visualizzazione della TAC in modo da non avere una singola immagine statica ma piuttosto una sequenza, come effettivamente registrato dalla macchina, in grado di ricostruire meglio la struttura della parte interessata. Un altro dato molto importante che sarebbe utile visualizzare è relativo alla possibilità di calcolare in automatico l'intervallo di tempo in cui un certo parametro vitale, di cui si sta effettuando il monitoraggio, rimane sopra o sotto una certa soglia predefinita, in modo da poter regolare l'attività sul paziente di conseguenza. Essi inoltre hanno messo in luce come alcuni dei requisiti richiesti non siano al momento ancora stati implementati, e nonostante questo non riduca il valore del risultato ottenuto è importante che essi vengano implementati nel sistema al più presto. I due aspetti principali che al momento risultano mancanti sono l'assenza di possibilità di personalizzazione del layout del display in base al Trauma Leader e la mancata implementazione di alcune operazioni di annotazione e gestione del trauma che al momento non è ancora possibile eseguire direttamente tramite il sistema, e che invece sarebbe molto utile poter utilizzare tramite l'assistente vocale.

Sono inoltre stati raccolti diversi aspetti aggiuntivi, inizialmente non previsti dai requisiti, che sarebbe comunque utile includere nella versione finale del progetto e che saranno esposti nel dettaglio nel paragrafo relativo agli sviluppi futuri inserito nelle conclusioni.

Conclusioni

Il lavoro sul progetto Smart Shock Room ha prodotto diversi risultati interessanti. Innanzitutto il più ovvio riguarda l'implementazione del sistema stesso, capace di gestire l'ambiente della Shock Room e di fornire una serie di funzionalità capaci di supportare le necessità dell'equipe medica del Trauma Team, permettendo di mostrare svariate informazioni relative al paziente e al trauma in corso anche grazie all'integrazione con il sistema Trauma Tracker, attualmente utilizzato nell'ospedale. Nonostante il sistema sia ancora in una prima fase prototipale dello sviluppo esso risponde ad un insieme significativo dei requisiti richiesti ed un livello di usabilità e performance adeguato, tale da permettere una valutazione positiva da parte degli esperti del dominio che hanno partecipato alla validazione dello stesso. Il risultato portato da questo lavoro tuttavia non è solamente legato all'implementazione del sistema stesso, ma ha permesso di mettere in luce una possibile architettura per permettere la modellazione di sistemi di questa tipologia. Questa architettura deriva dall'integrazione tra diversi approcci tecnologici particolarmente innovativi e promettenti, come il paradigma ad agenti e il modello Digital Twin, che si è dimostrata particolarmente adatta e funzionale alla modellazione e progettazione di un sistema pervasivo di questo genere. Questo perché la possibilità di integrare diversi approcci ha permesso di modellare ogni parte del domino secondo l'approccio che più è risultato adatto alle singole caratteristiche dell'entità e quindi di ridurre notevolmente il lavoro complessivo grazie ad una modellazione il più possibile semplice e lineare. I paradigmi scelti inoltre sono ottimali in termini di estendibilità e modularità del sistema, garantendo il massimo grado di disaccoppiamento tra i diversi componenti, in modo che sia possibile aggiungere funzionalità ed elementi al sistema in maniera semplice e veloce senza dover agire sull'architettura complessiva.

La scelta di integrare diversi approcci e quindi diverse tecnologie porta inevitabilmente con se un overhead dovuto alla necessità di utilizzare diversi linguaggi e diverse tecnologie, che quindi richiedono un maggiore tempo di apprendimento rispetto all'utilizzo di un unico approccio, ma tale svantaggio è ampiamente bilanciato dai vantaggi sopra elencati.

Si può quindi concludere che la scelta di integrare diversi approcci tecnolo-

gici si è dimostrata ottimale soprattutto a livello architetturale, in quanto permette di semplificare notevolmente il processo di mappatura del dominio, ottenendo così un'architettura più aderente all'ambiente che essa modella. Questo bilancia ampiamente il tempo necessario per l'implementazione e una efficienza leggermente minore rispetto ad una possibile architettura adhoc, rendendo in generale conveniente l'applicazione di questo tipo di modellazione a problemi specifici legati allo sviluppo di ambienti intelligenti.

Sviluppi Futuri

I principali sviluppi futuri per questo progetto rappresentano nell'immediato un miglioramento dell'interfaccia utente relativa alla presentazione dei dati e il completamento dell'implementazione delle funzionalità richieste che ancora non risultano implementate, come ad esempio la possibilità di personalizzare il layout del display in base allo specifico Trauma Leader.

Inoltre l'equipe di esperti stessa ha suggerito alcune funzionalità aggiuntive che sarebbe utile implementare ed integrare nel sistema Smart Shock Room:

- La possibilità di accedere alle lastre effettuate sul paziente mentre queste sono eseguite, senza dover aspettare che esse vengano caricate manualmente nel sistema informativo dell'ospedale come avviene ora, in modo da avere le immagini radiografiche sempre a disposizione in base al bisogno. Per poter ottenere questo risultato sarà necessario integrare nel sistema il macchinario fisico che effettua le radiografie, modellandolo con uno specifico Digital Twin in grado di gestire tali dati.

- La possibilità di gestire le informazioni mostrate sul display da remoto, in modo che qualcuno da un altro reparto, ad esempio dalla terapia intensiva, nel caso noti qualcosa di particolare nei dati mostrati possa metterlo in evidenza, fornendo un supporto al team che sta operando in Shock Room. Un'ulteriore sviluppo di questa funzionalità potrebbe essere quella di rendere possibile la comunicazione tramite streaming video tra i due ambienti in modo da far comunicare tra loro i medici.

Quest'ultimo particolare sviluppo risulta molto interessante nell'ottica dei sistemi pervasivi, in quanto porta in se l'idea di un ambiente intelligente non più limitato alla singola stanza ma diffuso e integrato per tutta la struttura ospedaliera. In quest'ottica si può già immaginare un estensione del sistema ASTRA per ottenere un ospedale Smart composto da un insieme di diversi ambienti intelligenti, ognuno dotato del suo Room Manager, in grado di comunicare tra loro e con altri sistemi per fornire funzionalità specifiche in base al tipo di utente e di ambiente in cui esso si trova. Questa rappresenta la visione finale di questo progetto, di cui il sistema sviluppato in questo documento rappresenta una primissima base, sia implementativa che concettuale.

Bibliografia

[1] Introducing json. https://www.json.org/json-en.html. Accessed: 2020-05-18.

[2] Message-oriented middleware. https://en.wikipedia.org/wiki/Message-oriented_middleware. Accessed: 2020-06-22.

[3] Osservatorio internet of things 2020, tutto sul mercato dell'iot in italia. https://www.zerounoweb.it/trends/dinamiche-di-mercato/osservatorio-internet-of-things-2020-tutto-sul-mercato-delliot-in-italia/. Accessed: 2020-05-31.

[4] Rabbitmq. https://it.wikipedia.org/wiki/RabbitMQ. Accessed: 2020-06-02.

[5] Smart cities. https://ww2.frost.com/wp-content/uploads/2019/01/SmartCities.pdf. Accessed: 2020-06-15.

[6] M Al-Khafajiy, H Kolivand, T Baker, D Tully, and A Waraich. Smart hospital emergency system via mobile-based requesting services. *Multimedia Tools and Applications*, February 2019.

[7] JP Black, W Segmuller, N Cohen, B Leiba, A Misra, MR Ebling, and E Stern. Pervasive computing in health care: Smart spaces and enterprise information systems. In *Proceedings of the MobiSys 2004 Workshop on Context Awareness*, 2004.

[8] Wooldridge M. Bordini R., Hubner J. *Programming multi-agent systems in AgentSpeak using Jason*. Wiley Series in Agent Technology. John Wiley & Sons, 2007.

[9] Michael Bratman. *Intention, Plans, and Practical Reason*. Cambridge, MA: Harvard University Press, 1987.

[10] Peter H. Carstensen and Kjeld Schmidt. Computer supported cooperative work: New challenges to systems design. In *In K. Itoh (Ed.), Handbook of Human Factors*, pages 619–636, 1999.

[11] D.A. Coates. *Voice Applications for Alexa and Google Assistant*. Manning Publications, 2019.

[12] Diane Cook and Sajal Kumar Das. *Smart environments: technology, protocols, and applications*, volume 43. John Wiley & Sons, 2004.

[13] Angelo Croatti, Sara Montagna, and Alessandro Ricci. A personal medical digital assistant agent for supporting human operators in emergency scenarios. In Sara Montagna, Pedro Henriques Abreu, Sylvain Giroux, and Michael Ignaz Schumacher, editors, *Agents and Multi-Agent Systems for Health Care*, pages 59–75, Cham, 2017. Springer International Publishing.

[14] Thierry Edoh. Internet of things in emergency medical care and services. In Hamed Farhadi, editor, *Medical Internet of Things (m-IoT)*, chapter 1. IntechOpen, Rijeka, 2019.

[15] Abdulmotaleb El Saddik. Digital twins: The convergence of multimedia technologies. *IEEE multimedia*, 25(2):87–92, 2018.

[16] Patrick Eugster, Pascal Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35:114–131, 06 2003.

[17] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

[18] Dominique Guinard and Vlad Trifa. *Building the Web of Things: With Examples in Node.Js and Raspberry Pi*. Manning Publications Co., USA, 1st edition, 2016.

[19] Robert R Harmon, Enrique G Castro-Leon, and Sandhiprakash Bhide. Smart cities and the internet of things. In *2015 Portland International Conference on Management of Engineering and Technology (PICMET)*, pages 485–494. IEEE, 2015.

[20] Simon Holmes. *Getting MEAN with Mongo, Express, Angular, and Node*. Manning Publications Co., USA, 1st edition, 2015.

[21] Andreas Holzinger, Carsten Röcker, and Martina Ziefle. *From Smart Health to Smart Hospitals*, volume 8700, pages 1–20. Springer, 2 2015.

[22] David Isern and Antonio Moreno. A systematic literature review of agents applied in healthcare. *Journal of Medical Systems*, 40, 02 2016.

[23] Nicholas R Jennings. On agent-based software engineering. *Artificial intelligence*, 117(2):277–296, 2000.

[24] Alessio Luschi, Andrea Belardinelli, L. Marzi, Francesco Frosini, Roberto Miniati, and Ernesto Iadanza. Careggi smart hospital: A mobile app for patients, citizens and healthcare staff. *2014 IEEE-EMBS International Conference on Biomedical and Health Informatics, BHI 2014*, pages 125–128, 06 2014.

[25] Bottazzi Manuel. Conversational healthcare : Review sulle applicazioni dei sistemi conversazionali in medicina. Available at `http://apice.unibo.it/xwiki/bin/view/Courses/Sa1819ProjectsConversationalhelthcareBottazzi`.

[26] Weiser Mark. The computer for the 21st century. *Scientific American*, Vol 265, No 3:94–104, 1991.

[27] R. Martinez-Velazquez, R. Gamez, and A. E. Saddik. Cardio twin: A digital twin of the human heart running on the edge. In *2019 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*, pages 1–6, 2019.

[28] Sara Montagna, Angelo Croatti, Alessandro Ricci, Vanni Agnoletti, Vittorio Albarello, and Emiliano Gamberini. Real-time tracking and documentation in trauma management. *Health Informatics Journal*, 26:146045821982550, 02 2019.

[29] Sam Newman. *Building Microservices*. O'Reilly Media, Inc., 2015.

[30] Cathy Pearl. *Designing Voice User Interfaces: Principles of Conversational Experiences*. O'Reilly Media, Inc., 1st edition, 2016.

[31] Qinglin Qi and Fei Tao. Digital twin and big data towards smart manufacturing and industry 4.0: 360 degree comparison. *Ieee Access*, 6:3585–3593, 2018.

[32] Alessandro Ricci, Michele Piunti, Luca Tummolini, and Cristiano Castelfranchi. The mirror world: Preparing for mixed-reality living. *Pervasive Computing, IEEE*, 14:60–63, 06 2015.

[33] Alessandro Ricci, Michele Piunti, Mirko Viroli, and Andrea Omicini. *Environment Programming in CArtAgO*, pages 259–288. Springer US, Boston, MA, 2009.

[34] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. Programming mas with artifacts. In Bordini R.H., Dastani M.M., Dix J., and El Fallah Seghrouchni A., editors, *Programming Multi-Agent Systems. ProMAS 2005. Lecture Notes in Computer Science*, volume volume 3862, pages 206–221, 07 2005.

[35] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. Cartago: A framework for prototyping artifact-based environments in mas. In *International Workshop on Environments for Multi-Agent Systems*, pages 67–86. Springer, 2006.

[36] Luis F. Rivera, Miguel Jiménez, Prashanti Angara, Norha M. Villegas, Gabriel Tamura, and Hausi A. Müller. Towards continuous monitoring in personalized healthcare through digital twins. In *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*, CASCON '19, page 329–335, USA, 2019. IBM Corp.

[37] Bhagya Nathali Silva, Murad Khan, and Kijun Han. Towards sustainable smart cities: A review of trends, architectures, components, and open challenges in smart cities. *Sustainable Cities and Society*, 38:697 – 713, 2018.

[38] C. Toremis and E. Karaarslan. A web technology based smart emergency room system recommendation: Smarter. In *2017 International Conference on Computer Science and Engineering (UBMK)*, pages 1117–1121, 2017.

[39] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.