

ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA  
CAMPUS DI CESENA

---

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

**SVILUPPO DI UN SISTEMA DI E-HEALTH  
PER IL TRACCIAMENTO DEI PARAMETRI  
VITALI NELLA GESTIONE DEI TRAUMI**

*Elaborato in*  
**SISTEMI DISTRIBUITI**

*Relatore*  
Prof. ALESSANDRO RICCI

*Presentata da*  
GIANLUCA SPADAZZI

*Co-relatore*  
Dott.ssa SARA MONTAGNA  
Ing. ANGELO CROATTI

---

Terza Sessione di Laurea  
Anno Accademico 2015 – 2016



# **PAROLE CHIAVE**

e-health

HL7

TraumaTracker

REST

Microservizi



*Alla mia ragazza, alla mia famiglia e ai miei amici*



# Indice

<b>Introduzione</b>	<b>ix</b>
<b>1 Informatica in campo sanitario</b>	<b>1</b>
1.1 Breve storia . . . . .	1
1.2 Importanza dell'interoperabilità . . . . .	2
1.3 Sistemi informativi sanitari . . . . .	3
1.3.1 HIS - Hospital Information System . . . . .	3
1.3.2 Basi di dati . . . . .	5
1.4 Standard sanitari . . . . .	7
1.4.1 DICOM . . . . .	7
1.4.2 OpenEHR . . . . .	9
1.5 Health Level Seven International (HL7) . . . . .	10
1.5.1 HL7 2.x e HL7 3 . . . . .	12
1.5.2 Componenti dello standard . . . . .	15
1.6 IHE - Integrating the Healthcare Enterprise . . . . .	21
1.6.1 Processo IHE . . . . .	23
1.6.2 Technical Framework . . . . .	24
1.6.3 IHE Profiles . . . . .	24
1.6.4 Connectathon . . . . .	25
1.7 Stato attuale . . . . .	25
1.7.1 Sanità digitale in Italia e in Europa . . . . .	25
1.7.2 Alcune aziende italiane . . . . .	28
<b>2 Progetto TraumaTracker e Infrastruttura GT2</b>	<b>35</b>
2.1 La gestione dei traumi . . . . .	36
2.1.1 Pervasive Health . . . . .	37
2.2 TraumaTracker . . . . .	38
2.2.1 Requisiti . . . . .	39
2.2.2 Analisi e Dispositivi utilizzati . . . . .	39
2.2.3 Architettura del sistema . . . . .	42
2.3 Infrastruttura GT2 . . . . .	47

<b>3 GatewayService</b>	<b>51</b>
3.1 HL7SubSystem . . . . .	52
3.1.1 Infinity Gateway . . . . .	53
3.1.2 Sistema simulato . . . . .	62
3.1.3 Server reale . . . . .	72
3.1.4 Test del sistema . . . . .	74
3.2 GatewayService . . . . .	85
3.2.1 Requisiti . . . . .	85
3.2.2 Analisi . . . . .	86
3.2.3 Progetto . . . . .	99
3.2.4 Test del sistema . . . . .	102
3.3 Monitors Viewer . . . . .	109
3.3.1 Requisiti . . . . .	109
3.3.2 Analisi e progetto . . . . .	110
3.4 Sviluppi futuri . . . . .	111
<b>Conclusioni</b>	<b>117</b>
<b>A Architetture e tecnologie utilizzate</b>	<b>119</b>
A.1 Microservizi . . . . .	119
A.2 REpresentational State Transfer (REST) . . . . .	121
A.2.1 Mobile computing . . . . .	125
A.3 JSON . . . . .	126
A.4 Swagger IO . . . . .	126
A.4.1 Componenti . . . . .	127
A.5 Vert.x . . . . .	130
A.5.1 Event loop e confronto con Node JS . . . . .	130
A.5.2 Architettura . . . . .	133
A.5.3 REST con Vert.x . . . . .	135
A.6 NoSQL . . . . .	135
A.6.1 MongoDB . . . . .	138
A.7 Angular . . . . .	139
A.8 HAPI - HL7 Api . . . . .	141
A.8.1 Funzionalità . . . . .	141
A.8.2 Alcuni esempi . . . . .	142
A.9 JMeter . . . . .	144
<b>Ringraziamenti</b>	<b>147</b>
<b>Bibliografia</b>	<b>149</b>

# Introduzione

L'ambito medico è uno dei più importanti settori in cui l'informatica può fornire un importante contributo al fine di rendere più efficienti e meno costosi i processi ospedalieri, fornendo strumenti per semplificare, supportare, gestire e controllare il lavoro degli operatori sanitari. All'interno delle strutture ospedaliere convivono tanti diversi sistemi informativi per la gestione dei flussi di dati clinici, diagnostici, amministrativi e non solo. Questi sistemi sono prodotti da tanti diversi produttori che spesso impiegano anche tecnologie diverse, e questo fa emergere la necessità di implementare soluzioni di interoperabilità per fare in modo che i sistemi possano interagire tra di loro scambiandosi informazioni. Per questa ragione sono nati standard utilizzati anche oggi come HL7, DICOM e non solo, oltre ad organizzazioni come IHE che promuovono gli standard dando direttive su come utilizzarli al meglio per ottenere interoperabilità.

Le tecnologie informatiche in ambito industriale negli ultimi anni hanno fatto passi da gigante, in particolare per quanto riguarda le tecnologie mobili, wearable ed hands free, che diventano sempre più potenti e che hanno aperto nuove frontiere per lo sviluppo di applicazioni e-health. Nonostante le problematiche relative alle scarse risorse computazionali ed in particolare alla scarsa durata dalla batteria, è impensabile non utilizzare questo tipo di tecnologie in ambito medico, oltre alle già numerose ed importanti applicazioni gestionali che si occupano della gestione dei dati che fluiscono all'interno dell'ospedale. E' possibile quindi acquisire automaticamente dati, supportare le operazioni degli operatori con strumenti aggiuntivi e non solo per ottimizzare quello che viene effettuato, ridurre gli sprechi e ridurre gli errori. Nonostante tutti questi vantaggi questo tipo di tecnologie non viene ancora largamente impiegato dappertutto. Per questa ragione, grazie alla collaborazione con l'ospedale Bufalini di Cesena e un gruppo di ricerca del Dipartimento di Informatica - Scienza e Ingegneria (DISI), Università di Bologna, Sede di Cesena, è in via di sviluppo il sistema TraumaTracker, che utilizza questo tipo di tecnologie per supportare l'acquisizione di dati relativi alle operazioni effettuate dal team di medici durante il processo di rianimazione a seguito di traumi, al fine di produrre dei report completi senza che debbano essere scritti a mano dagli

operatori dopo che l'intervento è stato completato, con tutti i problemi relativi a dimenticanze ed errori che ne conseguono. In questo elaborato viene presentato un servizio web sviluppato al fine di acquisire e tracciare il valore dei parametri vitali dei pazienti durante tutto il percorso della rianimazione. Il sistema si interfaccia con l'Infinity Gateway, server sviluppato da Draeger che contiene i parametri vitali che gli vengono inviati dai monitor che seguono il paziente e ne visualizzano real-time l'andamento. L'applicativo sviluppato espone quindi delle API che permettono alle applicazioni interessate di recuperare e tracciare i parametri vitali associati a un monitor. Il servizio viene quindi usato da TraumaTracker per recuperare periodicamente e/o su richiesta i parametri vitali con il fine di inserirli all'interno del report. Inizialmente, a causa dell'impossibilità di testare il sistema in un contesto reale, è stato sviluppato un server simulato che emula il funzionamento del gateway reale. Successivamente, grazie al fatto che Draeger ha fornito il software del gateway da installare insieme ad un monitor reale, è stato possibile effettuare i test del sistema in un contesto molto più simile a quello reale e quindi sviluppare un insieme di considerazioni connesse con le prestazioni ed il reale utilizzo del sistema. L'elaborato è strutturato in questo modo:

- Nel primo capitolo verrà affrontato il tema dell'informatica in ambito sanitario, soffermandosi in particolare sulla questione dell'interoperabilità, insistendo sulla sua importanza ed effettuando quindi una panoramica degli standard utilizzati per cercare di ottenerla
- Nel secondo capitolo si descrive il progetto TraumaTracker, che ha lo scopo come accennato di automatizzare la stesura dei report contenenti le operazioni svolte dagli operatori sui pazienti che arrivano al pronto soccorso. Si introduce quindi la visione dell'infrastruttura GT2, che si pone al di sopra dei sistemi già presenti all'interno dell'ospedale, e si occupa di interagire con essi in modo tale da fornire servizi alle applicazioni, semplificando il loro sviluppo
- Nel terzo capitolo si illustra lo sviluppo del servizio web utilizzato da TraumaTracker che si occupa del recupero e del tracciamento dei parametri vitali dei pazienti. Si mostra lo sviluppo di una libreria che si occupa dell'interfacciamento con l'Infinity Gateway, per poi passare allo sviluppo di un microservizio che, servendosi della libreria, fornisce delle API per le applicazioni che hanno la necessità di ottenere i parametri vitali dei pazienti, e infine allo sviluppo di un'applicazione web per visualizzare i parametri vitali dei pazienti da remoto e manipolare le informazioni di cui il sistema ha bisogno per funzionare correttamente

- In appendice si può consultare una panoramica sulle tecnologie, architetture e strumenti utilizzati per lo sviluppo del software

Il sistema sviluppato verrà sperimentato in ospedale in modo tale da completare il suo sviluppo ed assicurarsi che sia pronto per essere utilizzato in un contesto reale.



# Capitolo 1

## Informatica in campo sanitario

Con il termine *e-health* ci si riferisce solitamente a tutto l'insieme delle possibili applicazioni dell'ICT in ambito medico. Il termine è stato coniato in maniera analoga ad altri termini di questi tipi (come e-commerce ad esempio) poichè sembrava appropriato coniare un nuovo termine non appena lo sviluppo delle tecnologie ha aperto nuove possibilità in ambito medico. Una possibile definizione più accurata è la seguente: *e-health* è una disciplina che si trova tra l'informatica medica, la sanità pubblica e il business, che si occupa della fornitura di servizi di healthcare attraverso le tecnologie informatiche. Non si riferisce solo alla parte tecnica, ma anche all'idea stessa di utilizzare queste tecnologie per migliorare questi servizi [2]. La *Health Informatics* è una disciplina scientifica in continua evoluzione che si occupa della collezione, del salvataggio, del recupero, della comunicazione e dell'uso di dati relativi all'ambito medico. Questa disciplina applica metodi e tecnologie dell'informatica al problem solving, al decision making e al miglioramento dell'assistenza sanitaria in tutte le aree delle scienze biomediche [3].

### 1.1 Breve storia

Lo sviluppo dell'informatica in ambito medico iniziò negli anni 50, quando appunto si iniziarono a impiegare i computer in questo ambito. Tuttavia fino agli anni 70 (periodo in cui ci fu un rapido sviluppo delle tecnologie e dell'hardware, che costava sempre meno ed era sempre più potente) il tutto rimase rilegato a semplici esperimenti e difatti ancora il termine *health informatics* non era stato coniato. Il termine *medical informatics* e *nursing informatics* apparve infatti per le prime volte nelle pubblicazioni solo negli anni 70. Con l'arrivo dei microcomputer (in particolare all'Apple II, il primo vero personal computer), la tecnologia iniziò a diffondersi anche nelle case piuttosto che rimanere rilegata solo all'ambito business. Nel 1978 venne introdotto il programma

VisicalC, il primo vero foglio di calcolo. Questo venne ovviamente visto come una ottima opportunità in campo finanziario, e allo stesso modo venne preso in considerazione dai responsabili della gestione finanziaria dell'ambito medico. Il personal computer iniziò quindi a farsi strada più profondamente negli ospedali e in altre aree legate all'ambito medico. Inizialmente i computer venivano utilizzati solo per processare i dati, ma in seguito si iniziò anche a usare l'informatica in modo più innovativo, ad esempio sfruttando l'intelligenza artificiale per il decision making. Dal 1989 si iniziarono anche a connettere tra di loro diversi sistemi utilizzando le tecnologie di rete e i database.

## 1.2 Importanza dell'interoperabilità

Nonostante gli innumerevoli vantaggi dell'impiego dell'informatica in ambito medico, ancora oggi questa non viene largamente impiegata rispetto ad altri ambiti. La prima causa da ricercare è nel concetto di *interoperabilità*. Per interoperabilità si intende l'abilità di due o più sistemi indipendenti tra loro di interagire senza la necessità di onerose modifiche alla loro struttura o/e al loro comportamento. Lo scopo è lo scambio di informazioni tra questi sistemi in modo da raggiungere determinati obiettivi. L'interoperabilità è un aspetto fondamentale affinchè l'impiego delle tecnologie informatiche in ambito medico sia possibile su larga scala. Basti pensare al fatto che sono presenti sul mercato tanti vendor differenti che propongono i loro sistemi e macchinari per gli ospedali. Questi devono poter essere integrati tra loro e devono poter comunicare. I medici devono poter:

- Essere connessi gli uni con gli altri quando si tratta di effettuare trasferimenti di pazienti
- Essere connessi con i farmacisti per evitare errori nocivi per salute dei pazienti
- Essere connessi ai dati medici del paziente

Oltre a ciò, gli ospedali devono essere connessi tra loro per supportare il passaggio di informazioni tra di essi, e i laboratori devono essere connessi ai dati medici del paziente. Si prenda in considerazione questo esempio: un paziente si trasferisce da una città a un'altra e si reca in ospedale. Se manca l'interoperabilità, sarà costretto a ripetere gli esami e a presentare una copia cartacea della sua storia medica, in quanto dal nuovo ospedale non potranno accedere ai dati elettronici dell'altro ospedale perché gestiti in modo totalmente diverso. Oltre agli aspetti legati ai dati medici e al paziente, vanno tenuti in considerazione

anche gli aspetti economici. La popolazione aumenta e invecchia, le strumentazioni e le cure costano.. Oltre a questo il modello cartaceo è inefficiente e porta spesso ad errori. Per queste ragioni è necessario cercare di investire il più possibile nel passaggio dal modello cartaceo a quello elettronico, sia per questioni di efficienza sia per questioni appunto economiche. Senza contare che il modello elettronico permette una maggiore visione dello stato del paziente sia dal punto di vista del paziente stesso sia dal punto di vista delle organizzazioni. Per tutte queste ragioni è quindi importante lavorare sugli standard per lo scambio e la condivisione dei dati per permettere la comunicazione di applicazioni, sistemi e organizzazioni diverse. Si possono identificare tre aspetti chiave dell'interoperabilità:

- *Aspetti tecnici*: spostare le informazioni da sistema A a sistema B
- *Aspetti semanticici*: il sistema A e il sistema B devono interpretare i dati nella stessa maniera
- *Aspetti di processo*: fare in modo che i processi aziendali delle organizzazioni a cui il sistema A e il sistema B appartengano possano collaborare

Le sfide che uno standard si deve prefissare sono le seguenti:

- Come è possibile condividere dati del paziente tra organizzazioni diverse e tra diverse tecnologie?
- Come può essere identificato un paziente da diverse organizzazioni?

Bisogna inoltre anche tenere in considerazione aspetti che non riguardano le interazioni tra diverse organizzazioni, ma anche il fatto che i dati del paziente, che provengono da diverse fonti (farmacie, laboratori, ospedali..) devono essere integrati tra loro in un modello elettronico comune.

## 1.3 Sistemi informativi sanitari

### 1.3.1 HIS - Hospital Information System

HIS è l'insieme degli strumenti informatici utilizzati in ambito sanitario per gestire i flussi clinici, diagnostici e amministrativi di un ospedale. Di seguito verrà mostrata e descritta la struttura tipica di un HIS, come mostrato in figura 1.1. Si nota che i componenti principali sono appunto la diagnostica, la clinica e l'amministrazione. Oltre a questi sono presenti anche il sistema direzionale, il CUP (Centro Unico di Prenotazione) e i sistemi territoriali.

## Diagnostica

- *LIS - Laboratory Information System*: sistema informatico utilizzato per gestire le richieste dei pazienti (come l'accettazione), e per processare, memorizzare e generare report riguardo alle informazioni generate dai macchinari dei laboratori. In sistemi più evoluti permette anche di controllare delle apparecchiature, e se è ben integrato con gli altri sistemi informatici dell'ospedale, permette anche di scambiare informazioni con essi
- *RIS - Radiological Information System*: sistema informatico utilizzato per gestire i flussi di dati in radiologia, e le sue funzioni vanno dall'approccio del paziente con la struttura fino alla stesura del referto
- *AnaPat - Anatomia patologica*: sistema informatico utilizzato per gestire i flussi di dati che riguardano l'anatomia patologica, quindi ciò che riguarda biopsie, esami istologici etc

## Clinica

- *ADT - Admission and Discharge Transfer*: sistema informatico che si occupa della gestione dei dati dei pazienti (anagrafica) e dei ricoveri
- *PS*: sistema informatico che si occupa della gestione del pronto soccorso
- *Reparto*: sistema informatico che si occupa della gestione del reparto
- *Cartella clinica*: sistema informatico che si occupa della gestione delle cartelle cliniche

## Amministrazione

La parte amministrativa di HIS si occupa principalmente di:

- *Gestione personale*: sistema informatico che si occupa della gestione del personale
- *Ordini magazzino*: sistema informatico che si occupa della gestione degli ordini (esempio: farmaci)
- *Sistema economico amministrativo*: sistema informatico che si occupa della gestione finanziaria

## Altro

- *Sistema direzionale*: gestione contabilità, gestione manutenzione, gestione magazzini, fornitori, gestione budget, pianificazioni..
- *CUP - Centro Unico di Prenotazione*: è il servizio telematico che permette al cittadino di prenotare prestazioni sanitarie diagnostiche e visite mediche specialistiche
- *Sistemi territoriali*: sistemi che concernono le attività di diverse strutture sparse sul territorio

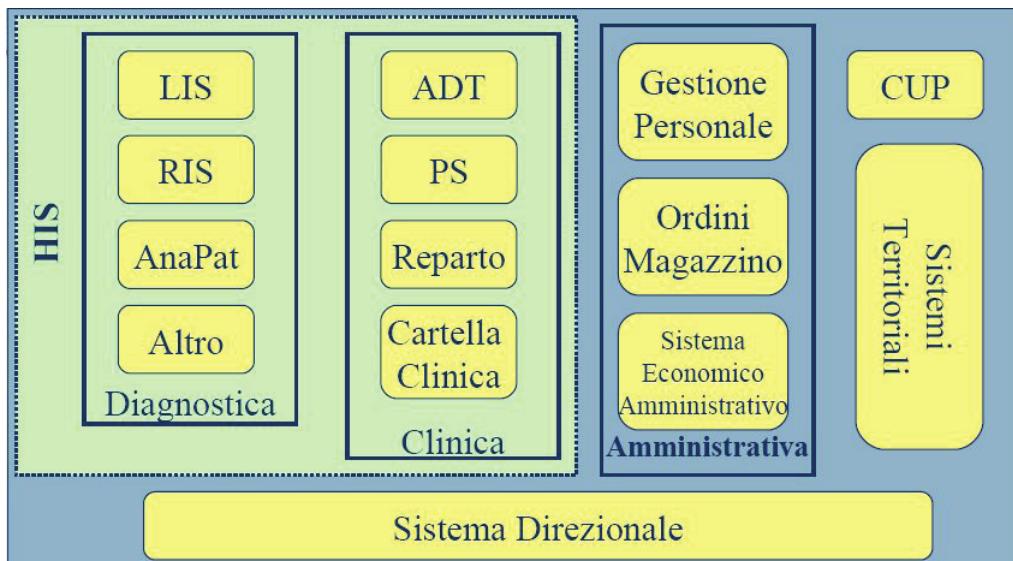


Figura 1.1: Struttura di un Hospital Information System [9]

### 1.3.2 Basi di dati

Ovviamente le basi di dati sono necessarie per memorizzare e accedere ai vari dati dei pazienti. Molti dati sono di tipo stringa o numerico, ma sono presenti anche altri tipi di dato, come audio, immagini e video. Per queste ultime tipologie vanno affrontate alcune problematiche, tra cui:

- Come rappresentare le immagini tramite dati alfanumerici
- Come indicizzare le immagini per recuperarle velocemente dai database

Ci sono diverse possibilità per affrontare questa problematica:

- *BLOB*: A tal fine, in SQL 1999, è stato introdotto il tipo di dato BLOB (Binary Large OBject), che permette di rappresentare oggetti di grandi dimensioni in formato binario. Tuttavia, nonostante sia possibile quindi salvare immagini, audio e video in un database tramite questo tipo di dato, non è possibile usare i valori inseriti come criterio di interrogazione per le query
- *Oracle Database*: Oracle Intermedia, estensione di Oracle Database, permette la gestione di contenuti multimediali come:
  - ORDImage
  - ORDAudio
  - ORDVideo
  - ORDDoc

La scelta di questo approccio ha alcuni vantaggi:

- Integrazione con sistemi Oracle (risultando quindi vantaggiosa se già si usano i loro sistemi)
- Alcuni dati relativi alle immagini vengono salvati automaticamente (come altezza, larghezza, estensione, tipo di compressione..), come mostrato in figura 1.2. Per quanto riguarda il campo *source*, è supportato attualmente l'accesso a immagini salvate nel database come BLOB, l'accesso a immagini esterne (file locali, file remoti e anche tramite URL)
- Supporto DICOM: Oracle Intermedia riconosce le immagini DICOM (descritto in 1.4.1) e quindi estrae i metadati ad esse associate. Questi metadati possono essere quindi salvati come attributi di immagini ORDImage oppure salvati direttamente in BLOB. Inizialmente il tutto veniva effettuato tramite una procedura, in seguito è stato introdotto il tipo di dato specifico ORDDicom.

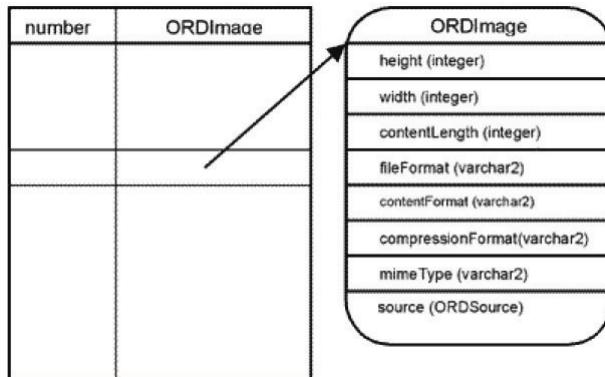


Figura 1.2: Campi di un oggetto ORDImage [9]

## 1.4 Standard sanitari

Si inizia a parlare per la prima volta di standard sanitari negli anni 60. Uno dei primi tentativi di costruzione di standard sanitari viene effettuato dalla ASTM (American Society for Testing and Materials), relativamente a standard per lo scambio di messaggi, salvataggio di dati medici e sicurezza dei sistemi sanitari. Successivamente vengono sviluppate diverse proposte di standard. Di seguito sono elencati alcuni degli standard più rilevanti attualmente:

- *Health Level 7 - HL7*: standard dedicato alla costruzione di un framework per lo scambio, integrazione, condivisione e recupero di dati sanitari, orientato sia alle questioni cliniche (ad esempio i dati dei pazienti) sia alle questioni organizzative (ad esempio ricoveri) dei servizi sanitari
- *Digital Imaging and Communication in Medicine - DICOM*: standard aperto per la gestione, archiviazione, visualizzazione e trasmissione di informazioni mediche per poter fare diagnostica di immagini mediche
- *OpenEHR*: specifica aperta che riguarda principalmente le cartelle cliniche elettroniche e la standardizzazione dei dati in esse contenute

Si andrà a effettuare una breve panoramica relativa a questi standard, ma verrà approfondito in particolare lo standard HL7, che è alla base del progetto relativo alla tesi.

### 1.4.1 DICOM

I computer non sono solo utilizzati per visualizzare immagini, ma anche per produrre immagini e modelli 3D partendo da dati raccolti tramite dispositivi

che usano tecniche come SPECT, PET, CT, MRI.. Si parla quindi di ogni branca che può avere a che fare con delle immagini. Si è reso quindi necessario lo sviluppo di uno standard per la connessione e lo scambio di immagini provenienti da dispositivi medici, dato che i produttori dei dispositivi sono tanti e diversi tra loro. Lo standard DICOM è composto da molte parti e risulta piuttosto ampio poichè deve coprire diverse branche. Ognuna di esse ha un sottogruppo che si occupa di sviluppare quella parte dello standard. Questo standard viene usato principalmente in radiologia, ma anche in cardiologia, mammografia e altri settori.

### Funzionamento

Lo standard supporta diversi tipi di immagini a seconda delle varie branche. Per trattare le immagini, DICOM, oltre ai dati dell'immagine, include altri dati importanti per l'immagine in strutture dati che sono posizionate in un header. Queste strutture dati sono chiamate Information Object Definitions (IOD), ovvero tabelle di attributi che definiscono gli information object, e contengono una descrizione, dati del paziente, nome dell'istituzione e altre informazioni (come ad esempio le procedure effettuate). Ad esempio, un information object può essere "paziente", contenendo attributi come "ID del paziente", "nome del paziente" e così via. Il supporto per immagini DICOM è stato implementato in diversi visualizzatori di immagini, in modo tale che la visualizzazione sia possibile in tutti i dispositivi e in particolare anche nei dispositivi dei pazienti a casa. Ad esempio Adobe Photoshop può visualizzare le immagini e anche leggere i dati contenuti nell'header. Oltre alla definizione di questi dati, lo standard si occupa anche di fornire servizi. I principali sono i seguenti:

- Archiviazione delle immagini
- Richiesta della lista di immagini presenti in una apparecchiatura da parte di un'altra apparecchiatura
- Trasferimento delle immagini
- Stampa delle immagini

Affinchè due dispositivi possano trasferire le immagini tra loro devono supportare gli stessi servizi e oggetti, chiamati Service-Object Pair (SOP). Un servizio è composto dalle operazioni che deve effettuare il dispositivo client e le operazioni che deve effettuare il dispositivo server. Una coppia di dispositivi con queste caratteristiche viene chiamato Service-Object Pair Class (SOP Class). Ad esempio, si supponga che due dispositivi implementano il SOP QUERY/-RETRIEVE (servizio che riguarda la richiesta di un'immagine da parte di un

dispositivo client a un dispositivo server). Il SOP definisce le operazioni che devono effettuare i due dispositivi (spiegando ad esempio che il client effettua la richiesta ed il server elabora la risposta) affinchè il tutto venga effettuato.

### 1.4.2 OpenEHR

OpenEHR è una specifica aperta che riguarda principalmente le cartelle cliniche elettroniche e la standardizzazione dei dati in esse contenute. Si vuole che il tutto sia indipendente dai diversi vendor e dalle diverse tecnologie. La terminologia medica è in costante evoluzione e quindi la standardizzazione è necessaria per poter sviluppare sistemi che siano in grado di interfacciarsi con gli altri. In un sistema openEHR i dati sono completamente interoperabili a prescindere dal linguaggio di programmazione e dalla tecnologia di database usata internamente.

#### Funzionamento

Alcuni dei problemi da affrontare sono i seguenti:

- Come fanno i vari software a conoscere la semantica dei dati? (Semantic Interoperability)
- Come costruire una cartella clinica elettronica condivisa da tutti i vari dispositivi/organizzazioni che potrebbero necessitarne l'accesso?
- Come creare sistemi che possano sopravvivere ai cambiamenti?

Per affrontare questi problemi in openEHR si utilizza un approccio a due livelli:

- *Reference model*: sono i modelli di informazioni base che definiscono come i dati sono rappresentati nella cartella clinica. In questo modo chi si occuperà di mantenere questo livello non dovrà cambiare sempre il modello se ci sono dei cambiamenti. Il modello ad esempio contiene la definizione dei tipi di dato
- *Archetype Object Model*: modelli che si possono definire con dei tool che definiscono appunto le conoscenze cliniche

Riassumendo, il Reference Model definisce i componenti base, che, mescolati tra loro, possono comporre diversi archetipi. Si possono creare archetipi per modellare diversi concetti, come ad esempio pressione sanguigna, peso, medicinali.. L'archetipo viene creato con un linguaggio chiamato ADL (Archetype Definition Language). Nei tool sono presenti generalmente interfacce grafiche

quindi non è necessario avere a che fare con il codice se si preferisce. Gli archetipi vengono usati dai *Template*, che sono il risultato delle aggregazioni di appunto determinati archetipi. I template sono in sostanza aggregazioni di più archetipi, di cui per ognuno di esso si possono mantenere alcuni dati ed eliminare altri. Ad esempio, si consideri di avere un archetipo che riguarda la pressione sanguigna. Questo definisce appunto un modello di riferimento che contiene i dati utili per la pressione sanguigna. Si può pensare di creare un template che riguarda un determinato report (ad esempio di una visita radiologica) e quindi comporlo di diversi archetipi tra cui appunto la pressione sanguigna. Grazie ai tool si possono creare template, compilarli, e eventualmente serializzarli in altri linguaggi come XML e JSON, per poi procedere ad esempio alla creazione di GUI relative al template. Infine va menzionato un ultimo aspetto. E' presente un linguaggio ad hoc che è costruito per fare query sulla base degli archetipi *AQL - Archetype Querying Language*. Questo permette di essere indipendenti dalla tecnologia di database utilizzata, lasciando al system developer il fatto di dover creare un mapping tra lo statement dell'AQL e il meccanismo nativo di querying.

## 1.5 Health Level Seven International (HL7)

La HL7 è un'organizzazione no profit dedita allo sviluppo di standard fondata nel 1987, ed è composta da membri che risiedono in oltre 55 paesi. Attualmente quasi tutti i fornitori di sistemi informativi sanitari in USA ne fanno parte e ultimamente stanno aderendo sempre più anche membri dell'Europa. HL7 si occupa di creare uno standard per lo scambio, integrazione, condivisione e raccolta di informazioni in ambito health care a supporto della gestione, consegna e valutazione dei servizi sanitari. Lo standard in questione è appunto HL7, che è utilizzato da più di 1600 membri distribuiti in 50 diversi paesi (produttori, compagnie farmaceutiche..). Il tutto nasce dalla necessità di gestire elettronicamente i dati medici, così come avviene già per tanti altri settori (come ad esempio la finanza). Non viene fornito nessun software, ma vengono fornite delle specifiche alle organizzazioni che permettono ad esse di costruire sistemi interoperabili. Level 7 si riferisce al fatto che HL7 concettualmente opera al settimo livello dello stack di protocolli ISO-OSI. Per questa ragione si occupa della definizione dei dati che devono essere scambiati, della comunicazione degli errori etc. I campi di applicazione dello standard sono diversi, tra cui:

- *Amministrazione del paziente*: ricoveri, dimissioni e trasferimenti di pazienti

- *Gestione degli ordini:* richiesta di farmaci, materiali o comunque risorse per i pazienti
- *Gestione finanziaria:* gestione delle transazioni da parte del paziente e tutti gli aspetti che le riguardano
- *Report di osservazioni:* possibilità di inviare dati clinici dei pazienti da un sistema all'altro (risultati di laboratorio, impiego di farmaci, allergie, segnali vitali, stato fisico..)
- *Gestione informazioni e cartelle cliniche:* facilitare la produzione, salvataggio e recupero di documenti accurati che contengono il resoconto dei servizi sanitari offerti al paziente
- *Logistica:* scheduling di appuntamenti
- *Cura del paziente:* riguarda la cura del paziente in ogni forma (assistenza clinica, assistenza a lungo termine, assistenza residenziale, assistenza domiciliare sanitaria, assistenza nelle scuole..)
- *Identificazione dei pazienti:* è necessario che, in caso in cui diverse organizzazioni abbiano bisogno di avere a che fare con lo stesso paziente, possano identificarlo correttamente e condividere le stesse informazioni

Lo standard ha diverse caratteristiche che lo rendono robusto e in grado di stare al passo con i tempi:

- Definisce delle specifiche che non dipendono dalle tecnologie e linguaggi di programmazione usati
- Supporta delle specifiche variazioni locali (ad esempio i segmenti HL7-Z che si possono inserire nei messaggi, ovvero segmenti che si definiscono localmente e che non fanno parte dello standard). In questo modo lo standard è flessibile. Questo è sia un punto di forza sia una debolezza. La flessibilità rende più facile l'adozione dello standard, ma la stessa flessibilità porta lo standard a non essere rigido e quindi le variazioni potrebbero limitare l'interoperabilità con altri sistemi
- E' strutturato in modo da prevedere la possibilità che insorgano nuovi requisiti nel corso del tempo

### 1.5.1 HL7 2.x e HL7 3

Attualmente viene utilizzata in particolare la versione 2.x dello standard. Precisamente le versioni 2.3 e 2.3.1. Le prime versioni (2.1 e 2.2) erano piuttosto vaghe e non erano state rilasciate specifiche formali su tutto. Questa fu una mossa vincente perché proprio la mancanza di rigidità eccessiva dello standard ne permise l'adozione in massa da parte dei vari vendor. Successivamente con la crescita dell'adozione dello standard, le versioni successive di esso furono sempre più specifiche e coprirono sempre più aree, mantenendo però la compatibilità con le versioni precedenti dello standard. La compatibilità viene mantenuta poiché gli elementi aggiunti nei messaggi vengono marcati come opzionali. Anche se si può fare, mantenere la compatibilità tra diverse versioni dello standard non è banale da implementare. Tuttavia la versione 2.x è più un framework per la negoziazione piuttosto che un vero e proprio standard. Per lasciare appunto più spazio per la personalizzazione è stato adottato un approccio 80/20, ovvero era predefinito solo l'80 per cento dell'interfaccia. Il restante 20 per cento andava definito dalle organizzazioni liberamente. Come detto, questo favorì la diffusione dello standard perché così le organizzazioni potevano continuare a essere libere nonostante l'impiego dello standard, ma allo stesso tempo fece emergere la necessità di regole più strette. HL7 3 ha introdotto un modello dei dati esplicito chiamato RIM (Reference Information Model) in cui ci sono degli oggetti legati tra di loro tramite relazioni. Ogni oggetto ha un set di attributi. Tutte le varie parti dello standard si basano sul RIM e utilizzano quindi dei modelli derivati da esso. Tutte le interazioni in ambito medico che possono accadere vengono modellati tramite sei classi principali:

- *Entity*: sono le entità che possono venire coinvolte nelle varie interazioni (pazienti, medici, organizzazioni..)
- *Act*: azioni che possono essere compiute dalle attività (ad esempio un esame specifico)
- *Role*: ruoli che le entità possono assumere (ad esempio dottore)
- *Participation*: modalità in cui una entità che assume un dato ruolo deve effettuare una data azione
- *ActRelationship*: insieme di sotto azioni degli Act
- *RoleLink*: collegamenti con altri ruoli

Si veda a riguardo la figura 1.3. Viene anche introdotto XML come formato per lo scambio di dati, in quanto all'epoca della creazione di HL7 2.x non era

ancora stato inventato questo formato, e quando viene introdotto HL7 3 è stato opportuno adottarlo per facilitare il parsing ed il controllo sintattico sui dati scambiati. XML si contrappone quindi al formato di HL7 2.x che è costituito da dei messaggi di testo formattati grazie all'uso di determinati caratteri. La versione 3 dello standard nasce quindi per creare un modello dei dati più stretto e preciso (modello 90/10). Tuttavia la sua adozione sarà molto più lenta, più costosa e per un certo periodo sicuramente le applicazioni dovranno supportare sia la versione 2.x sia la versione 3 (dato che non sono interoperabili tra loro). Attualmente si ritiene che la versione 3 di HL7 sia stata troppo complicata, complessa e costosa da utilizzare e quindi secondo l'opinione di molti ha fallito nel suo intento. Allo stato attuale si continua ad usare ancora la versione 2.x dello standard nella maggior parte dei casi. Negli ultimi tempi è in via di sviluppo lo standard FHIR (Fast Healthcare Interoperability Resources), standard che ha lo scopo di rimpiazzare la versione 3 e che si basa su REST, cercando quindi di colmare le lacune di HL7 2.x (come l'eccessiva flessibilità e modello dei dati non esplicito) ma di utilizzare tecnologie e architetture leggere (per favorire le applicazioni mobile) e semplici, in modo da permettere ai vendor di adottare il nuovo standard senza particolari problemi.

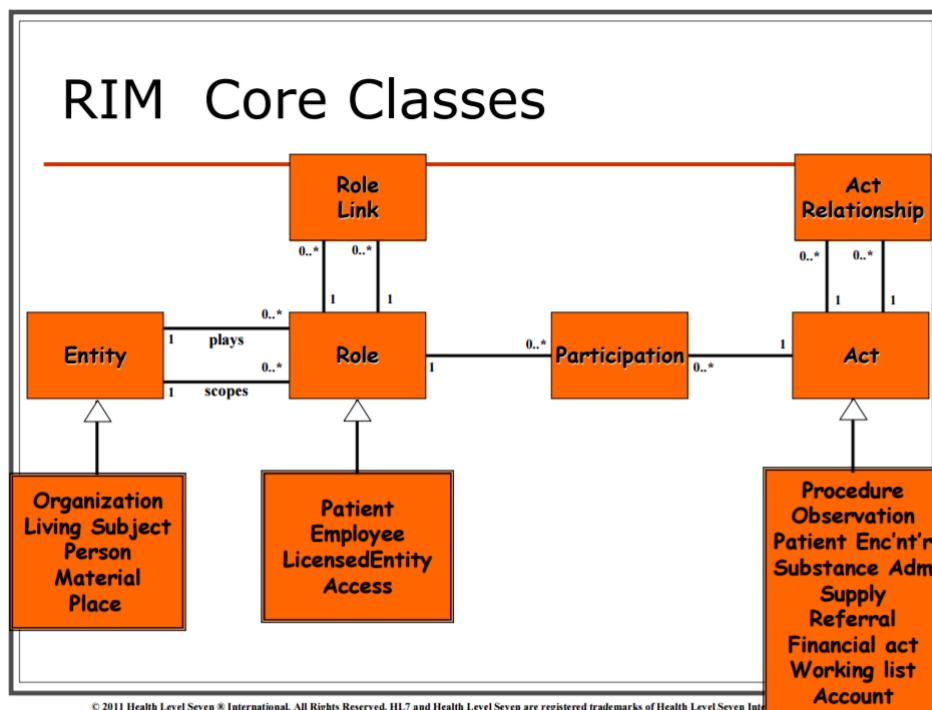


Figura 1.3: Core Clased del RIM (HL7 3) [22]

```

MSH|^~\&|AcmeHIS|StJohn|ADT|StJohn|20060307110111||ADT^A04|MSGID20060307110111|P|2.4
EVN|A04
PID|||12001||Jones^John||19670824|M|||123 West St.^Denver^CO^80020^USA
PV1||O|OP^PAREG||||2342^Jones^Bob||OP|||||||||2|||||||||||||||||20060307110111|
AL1|||3123^Penicillin||Produces hives~Rash~Loss of appetite

```

Figura 1.4: Esempio di messaggio HL7 versione 2.x [27]

```

<- <author>
  - <assignedEntity>
    <id root="2.16.840.1.113883.9876.210.3">
      extension="5332443" />
      <telecom value="tel:+1(317)630-7960" />
    - <assigneePerson>
      - <name>
        <given>Keiko</given>
        <family>Jones</family>
        <suffix>MD</suffix>
      </name>
    </assigneePerson>
  </assignedEntity>
</author>
<!-- Removed consumable -->
- <patientSubject>
  - <patient>
    <id root="2.16.840.1.113883.9876.211">
      extension="344253425" />
    + <addr>
      <telecom value="tel:213-555-4344" />
    - <patientPerson>
      <id root="2.16.840.1.113883.4.1">
        extension="333224444" />
      - <name>
        <given>George</given>
        <given>Simon</given>
        <family>Wigny</family>
      </name>
      <administrativeGenderCode code="M" />
      <codeSystem="2.16.840.1.113883.5.1" />
      <birthTime value="19740423" />
    </patientPerson>
  </patient>
</patientSubject>

```

Figura 1.5: Esempio di messaggio HL7 versione 3 [27]

## XML

Dalla versione 3 di HL7 si è incominciato ad usare XML come linguaggio di riferimento. Tutti gli standard che compongono HL7 si basano sul linguaggio XML a partire dalla versione 3. Un documento XML è formato da tre componenti (file):

- *Modello dei dati*: definisce le regole con cui è necessario strutturare il documento (chiamato solitamente Schema). Ad esempio che attributi possono esserci, di che tipo devono essere, numero di attributi figli, ordine di essi..
- *Contenuto*: l'informazione che viene creata sulla base del modello
- *Presentazione*: dai file XML è possibile derivare delle forme di presentazione come file PDF, pagine HTML e altro. Ad esempio si possono usare

degli stylesheet per passare dalla rappresentazione grezza del file xml da parte del browser ad una forma più elaborata

Il linguaggio XML è stato scelto perché permette una lettura e comprensione facile sia per gli umani sia per le macchine che possono fare delle elaborazioni.

### 1.5.2 Componenti dello standard

HL7 in realtà non è uno standard unico, ma è una composizione di diversi standard. Tutti i componenti HL7 ed il materiale fornito dall'organizzazione è organizzato in sette sezioni:

- *Sezione 1 - Primary Standards*: contiene gli standard più utilizzati dai vari produttori per costruire soluzioni di integrazione tra sistemi
- *Sezione 2 - Foundational Standards*: contiene i tool e i vari componenti di più basso livello usati per costruire i vari standard
- *Sezione 3 - Clinical and Administrative Domains*: standard per i messaggi e per i documenti clinici. Solitamente i produttori implementano per primi gli standard della sezione 1, e in seguito questi
- *Sezione 4 - EHR Profiles*: contiene gli standard utili per la gestione e costruzione delle cartelle cliniche elettroniche
- *Sezione 5 - Implementation Guides*: contiene guide per l'implementazione e documenti di supporto aggiuntivi a quelli già forniti nelle altre sezioni
- *Sezione 6: Rules and References*: contiene specifiche tecniche e linee guida per lo sviluppo del software
- *Sezione 7: Education & Awareness*: contiene gli HL7 Draft Standards for Trial Use (DSTUs) e anche delle risorse e tool utili per fornire ulteriore supporto per capire gli standard HL7 e adottarli

Di seguito vengono illustrati i componenti di HL7 più rilevanti.

#### Messaggi

Un messaggio HL7 ha lo scopo di passare informazioni di un certo evento da un sistema all'altro. Un messaggio viene inviato dopo l'avvenimento di un certo evento (trigger event). Di seguito si parlerà dei messaggi della versione 2.x dello standard, in quanto di interesse per il progetto. Lo standard definisce come i messaggi devono essere strutturati. Il messaggio si compone di segmenti in una sequenza definita che:

- Possono essere opzionali o obbligatori
- Possono essere ripetuti nel messaggio oppure possono comparire solo una volta
- Possono esserci vincoli del tipo ”se viene inserito un determinato segmento opzionale, allora devono esserci obbligatoriamente altri determinati segmenti”

Inoltre, come menzionato in precedenza, possono esserci dei segmenti personalizzati chiamati segmenti Z, al fine di permettere la modellazione di aspetti locali. Ogni segmento è costituito da campi (ripetibili all'interno del segmento) che a loro volta sono costituiti da Data Elements (componenti). Ogni messaggio deve sempre iniziare con un segmento MSH (MeSsage Header) che è sempre richiesto. Quando il messaggio è ricevuto, viene fatto il parsing di questo segmento per identificare il tipo di messaggio. Ogni segmento va terminato con il carattere ASCII 13 (carriage return). I campi contenuti nel segmento sono separati dal carattere ”|”. Ogni campo può essere composto, e nel caso lo sia, i suoi elementi sono separati dal carattere ”^”. Di seguito si illustrano alcuni dei tipi di messaggi disponibili. Ogni tipo di messaggio potrebbe essere associato a diversi trigger event.

- *ADT - Admission, Discharge and Transfer:* trasmissione di dati che riguardano ricoveri (Admission), dimissioni (Discharge) e trasferimenti (Transfer)
- *ACK:* messaggio di acknowledgement generico
- *ORU - Observation ResUlt:* messaggi riguardanti risultati di osservazioni che non sono inviati in risposta a specifiche richieste di utenti o applicazioni
- *ORM - ORder Message:* trasmissione di informazioni riguardanti ordini (farmaci, trattamenti..)
- *ORR - ORder Response:* messaggio generico di risposta agli ordini
- *PID - Patient IDentification:* messaggio contenente le informazioni del paziente

Una differenza importante in questo caso tra HL7 2.x e HL7 3 è che il formato dei messaggi di HL7 2.x è il formato pipe (figura 1.4), mentre HL7 3 usa XML (figura 1.5). I messaggi vengono scambiati in due modalità:

- *Unsolicited*: i messaggi vengono inviati da un sistema all'altro sulla base di eventi trigger. Il sistema ricevente risponde con un messaggio di tipo ACK
- *Solicited*: i messaggi vengono inviati esplicitamente da un sistema all'altro. Solitamente esiste un sistema che invia un messaggio di query a un sistema ricevente che risponde con un messaggio di risposta

I campi e i componenti contengono dei dati che possono provenire da tre fonti:

- Tabella HL7
- Tabelle definite localmente sulla base delle esigenze
- Tabelle definite esternamente con valori che riguardano decodifiche controllate (ad esempio LOINC)
- Soluzioni ibride (ad esempio elementi personalizzati localmente insieme a tabella HL7 o di altri standard)

**Esempio - PID Message** Un esempio di struttura di PID message si può trovare qui: (<https://corepointhealth.com/resource-center/hl7-resources/hl7-pid-segment>). Di seguito una legenda sulle proprietà dei campi:

- *SEQ*: numero che identifica in che posizione del segmento è un certo campo
- *LEN*: lunghezza del campo
- *DT*: tipo di dato
- *OPT*: Opzionalità del campo (O se opzionale, R se richiesto)
- *RP*: indica se il campo può essere ripetuto. Va lasciato vuoto in caso che non possa essere ripetuto, altrimenti va settato a Y
- *Element Name*: indica il valore del campo

Il messaggio quindi si compone di una sequenza di campi, tra cui come esempio Patient ID, Patient Name, Sex e così via.

**Esempio - ORU Message** Si può consultare la struttura di un messaggio ORU qui: <https://corepointhealth.com/resource-center/hl7-resources/hl7-oru-message>. Il messaggio è composto da gruppi di segmenti. Ogni gruppo può essere opzionale, obbligatorio, ripetibile (e i vincoli si applicano su tutto il gruppo o anche nelle sue parti). Si può notare come il messaggio si compone di diversi segmenti, tra cui lo stesso PID, che è un campo richiesto. E' presente anche un segmento chiamato NTE (NoTes and Comments), che è opzionale e ripetibile.

### Clinical Document Architecture - CDA

Questa parte dello standard è nata per facilitare lo scambio di documenti clinici all'interno e tra le istituzioni mediche. E' uno standard di markup che appunto definisce la struttura con cui le informazioni devono essere rappresentate. Potrebbe esserci del testo, immagini, multimedia.. I documenti CDA sono infatti codificati in XML. un documento CDA è flessibile, quindi può essere trasmesso con qualunque protocollo, sia HL7 stesso, sia anche HTTP, FTP, DICOM o altro.

- *Header:* contiene diverse informazioni sul documento:
  - Document Information: identità, appartenenza e relazioni del documento con altri documenti
  - Encounter: contesto del documento (dove si svolgono le attività, in che organizzazione, in che luogo..)
  - Service Actors: operatori coinvolti nelle attività descritte (includendo anche operatori amministrativi)
  - Service Targets: paziente coinvolto o altre persone coinvolte
- *Body:* contiene il report clinico e può essere sia un insieme di dati non strutturato (blob di testo, immagini..) [figura 1.6], sia strutturato con dei tag [figura 1.7]. Se è strutturato, si divide in sezioni. Ogni sezione può contenere:
  - Un blocco di testo che deve essere visualizzato per i lettori. E' obbligatorio
  - Delle entries: dati strutturati necessari per fare dell'ulteriore processing (utili per esempio per applicazioni di decision support).
  - Delle external reference: oggetti esterni al documento (esempio immagini)

Le sezioni possono essere innestate l'una dentro l'altra, e avere uno specifico autore e argomento (altrimenti si assume che sia tutto dello stesso autore).

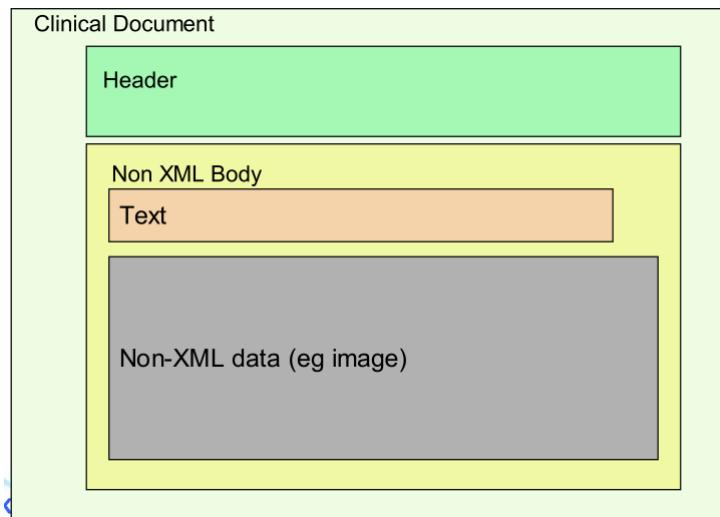


Figura 1.6: Struttura CDA con body non strutturato [19]

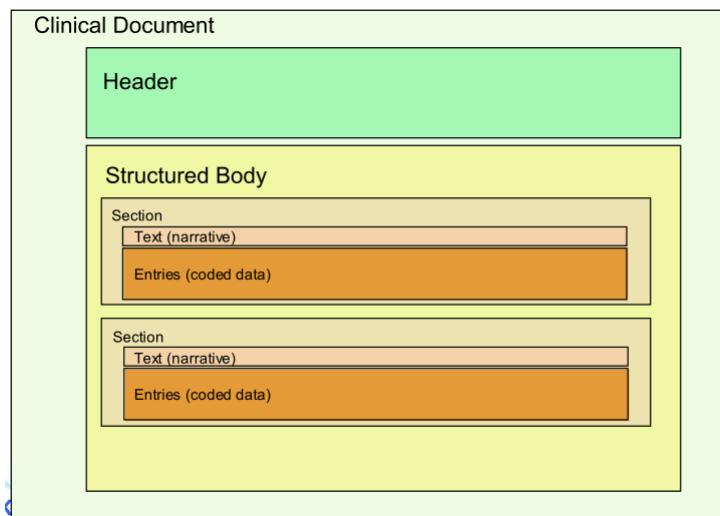


Figura 1.7: Struttura CDA con body strutturato [19]

### Continuity of Care Document - CCD

Documento che rappresenta un sommario delle informazioni sul paziente che riguardano il presente ed il passato. Contiene quelle informazioni utili

al proseguimento delle cure (quindi non tutte quante). Si appoggia su CDA, difatti risulta un’istantanea del tempo del paziente, sotto forma di CDA, contenenti informazioni cliniche, demografiche e amministrative. La sua ragione di esistenza deriva dal fatto che inizialmente erano presenti due standard: CDA e CCR. CCR (Continuity of Care Record) era uno standard basato su XML che aveva lo scopo di contenere appunto le informazioni di base sul paziente con lo scopo di facilitare il trasferimento del paziente. Per ridurre le distanze tra chi aveva adottato CDA e chi invece CCR è nato CCD. CCD usa dei template che indicano come devono essere appunto combinati e usati gli elementi CDA. Alcuni dei template sono i seguenti:

- Storia di famiglia
- Vaccini
- Parametri vitali
- Farmaci
- Problemi
- ...

Alcuni di essi devono essere obbligatoriamente inclusi, altri invece sono optional.

### **SPL - Structured Product Labeling**

Questo standard nasce da una problematica: il controllo delle informazioni sui prodotti. Solitamente le organizzazioni prendono le informazioni sui prodotti e le rielaborano per usarle per i loro scopi, portando a grosse inefficienze. Questo è problematico anche perchè ci sono diverse altre organizzazioni che potrebbero aver bisogno di questi dati (enti di regolarizzazione, centri di ricerca universitari, operatori sanitari, pubblico generico..), e hanno bisogno che questi siano il più possibile accurati e consistenti. Anche in questo caso viene usato il linguaggio XML. I goal di SPL includono:

- Rendere disponibili informazioni accurate e aggiornate delle etichette dei prodotti
- Facilitare la revisione, salvataggio e distribuzione di queste informazioni
- Permettere la lettura di queste informazioni agli umani ma anche alle macchine affinchè si possano fare delle elaborazioni

Se una organizzazione vuole adeguarsi allo standard, deve convertire le informazioni del prodotto nel formato XML richiesto e inviarlo all'organizzazione FDA (Food and Drug Administration), che rende disponibili le informazioni sui prodotti che rispettano lo standard. Un documento SPL si compone di tre parti:

- *Header*: contiene informazioni sull'organizzazione (nome, sede..) e sul prodotto in questione (nome, dosaggio..)
- *Contenuto narrativo*: contenuto narrativo leggibile dagli umani. Sono riportate le informazioni che di solito compaiono sul foglietto illustrativo (controindicazioni, effetti collaterali..)
- *Dati strutturati*: sono riportate informazioni sul farmaco come trade name, nome generico, principi attivi e inattivi , dettagli sul confezionamento.. Queste informazioni saranno usate principalmente per permettere lo scambio di dati tra computer, oltre ad analisi su di essi.

### CCOW - Clinical Context Object Workgroup

Questo standard definisce un mezzo per la coordinazione automatica e la sincronizzazione di diverse applicazioni healthcare che risiedono sullo stesso sistema. Le applicazioni che lo usano permettono all'utente di settare un contesto (insieme di soggetti clinici) usando una qualunque delle applicazioni. Una volta che il contesto è settato, tutte le altre applicazioni automaticamente si adattano al suddetto contesto. Esempio: l'utente sceglie un paziente, e automaticamente tutte le applicazioni si sintonizzano sul paziente scelto. I vantaggi di questo approccio sono evidenti in quanto per l'utente l'utilizzo delle applicazioni sarà più facile e intuitivo. Lo standard propone delle best practice da seguire per riuscire a raggiungere questo obiettivo.

## 1.6 IHE - Integrating the Healthcare Enterprise

IHE è un'iniziativa di professionisti del settore per migliorare il modo in cui i sistemi di computer in ambito medico condividono le informazioni tra di loro. IHE promuove l'uso coordinato di standard molto in voga come DICOM e HL7. Non vengono quindi definiti nuovi standard, ma viene definito come usarli. IHE è nata nel 1998 grazie alla collaborazione di HIMSS e RSNA che si sono resi conto dei problemi di interoperabilità tra i sistemi di imaging e di informazione. IHE è oggi sponsorizzata da associazioni di operatori sanitari in

tutto il mondo e ha accolto la partecipazione di molti dei principali produttori di sistemi di imaging e di informazione. Alcuni membri volontari di queste associazioni lavorano a stretto contatto con i vari produttori per identificare i problemi e gli ostacoli nell'integrazione e sviluppano di conseguenza delle soluzioni. I sistemi sviluppati in conformità con IHE hanno diverse qualità:

- Maggiore facilità di implementazione
- Comunicazione più semplice con altri sistemi IHE
- Maggiore efficienza data la maggiore semplicità nella fruizione delle informazioni

I medici, gli infermieri, gli amministratori e gli altri operatori auspicano un giorno in cui i dati dei pazienti possono essere passati in modo trasparente da un sistema all'altro sia all'interno dei reparti sia tra un reparto e l'altro, in modo che siano disponibili dove e quando servono. IHE nasce proprio per cercare di raggiungere questo obiettivo. La mancata efficienza e correttezza nella condivisione delle informazioni affligge molte figure professionali:

- *Medici*: il lavoro dei medici viene ostacolato se i sistemi non possono scambiarsi informazioni in modo efficace. Informazioni mancanti, difficili da reperire e ridondanti, non solo portano ad inefficienze, ma anche ad errori e problemi, dato che si andranno a prendere decisioni importanti sul paziente senza avere un quadro completo della situazione
- *Informatici*: l'interfacciamento dei sistemi è un compito chiave affrontato dallo staff IT. Avere a che fare con diversi vendor, e quindi avere la necessità di capire le differenze nell'implementazione dello standard da parte di essi, e infine capire come mettere tutti insieme, è un compito costoso e complesso
- *Amministratori*: chi si occupa di amministrazione deve tener conto di vincoli tecnici, fiscali, cinici e interpersonali nel compiere decisioni operative e di acquisto. Mancanza di quadri completi da questo punto di vista porta a aumento di costi, diminuzione dell'efficienza del personale, e infine anche della qualità della cura

Gli standard da soli però non bastano a risolvere i vari problemi, perchè in ognuno di essi ci sono opzioni e possibili conflitti interpretativi che portano a diverse implementazioni di essi. Inoltre nessuno standard riesce a coprire esattamente tutto quello che riguarda il dominio delle informazioni complesse e sempre in evoluzione dell'ambito medico. Quello che si fa generalmente per garantire la possibilità di scambiare informazioni tra un sistema e l'altro è

sviluppare costose e specifiche interfacce ad hoc. Questo però non è l'approccio corretto, poichè è necessario lo sviluppo di un framework per l'implementazione degli standard che indichi la direzione in modo unico, piuttosto che soluzioni ad hoc. IHE opera in tutti i settori dell'ambito medico, dato che appunto è necessario integrare le informazioni provenienti da ogni settore per aumentare l'efficienza.

### 1.6.1 Processo IHE

In modo tale da unire insieme vendor, sviluppatori e medici, viene innescato un processo annuale che si compone di quattro fasi:

- *Problem identification*: medici e esperti informatici identificano problemi nell'accesso e condivisione delle informazioni
- *Integration Profile Specification*: vengono selezionati gli standard che servono per l'integrazione e creati dei profili con delle specifiche da seguire. Le specifiche tecniche riguardanti il come implementare gli standard sono documentate nell'IHE Technical Framework
- *Implementation and testing*: i vendor implementano questi profili e testano i loro sistemi nel Connectathon, manifestazione in cui si testa l'interoperabilità dei sistemi con altri vendor
- *Integration statements and RFPs*: i vendor pubblicano gli Integration Statement, documenti che riportano i profili supportati dai loro prodotti, e li forniscono in caso di request for proposals

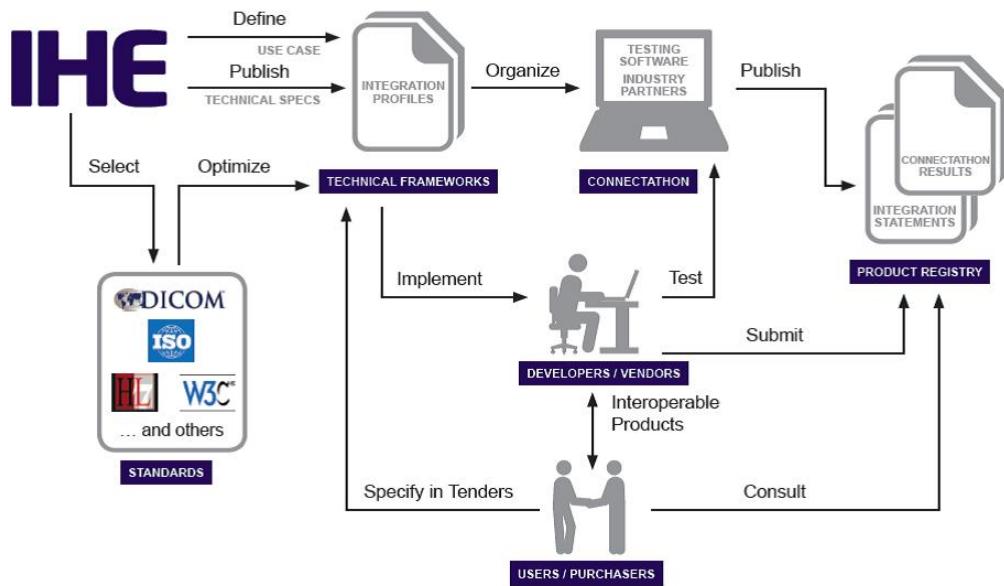


Figura 1.8: Processo IHE [4]

### 1.6.2 Technical Framework

Il Technical Framework è un documento molto dettagliato che fornisce una guida all'implementazione dei sistemi in modo da garantire la capacità di integrarsi con gli altri. Il Technical Framework definisce delle transazioni standardizzate tra sistemi chiamati attori. I Technical Framework sono tanti, vengono espansi annualmente, e ne è disponibile uno per ogni branca medica a cui può servire (cardiologia, radiologia..).

- *Actors*: i sistemi o applicazioni logiche che producono, gestiscono o lavorano sulle informazioni sono rappresentati come unità funzionali chiamate Actors. Ognuno di essi supporta uno specifico set di transazioni. Un sistema reale può supportare uno o più attori
- *Transactions*: le transazioni sono scambi di informazioni tra attori tramite standard definiti (come HL7 e DICOM). Ogni transazione è quindi associata a uno specifico standard, e oltre a ciò, ad informazioni aggiuntive come ad esempio degli use cases

### 1.6.3 IHE Profiles

I profili di integrazione organizzano insiemi di IHE Actors e IHE Transactions in modo da affrontare specifiche esigenze di assistenza del paziente. I

profili di integrazione offrono un modo conveniente per i vendor e gli utenti di fare riferimento alla funzionalità definite nel Technical Framework senza dover ribadire e riscrivere tutti i dettagli che riguardano gli attori e le transazioni. Questi profili descrivono le informazioni e il flusso di lavoro necessario e specificano gli attori e le transazioni che serviranno.

### 1.6.4 Connectathon

Il Connectathon è un evento che dura una settimana. Durante questo evento tutti i sistemi dei partecipanti vengono connessi in rete. Viene quindi simulato un ambiente ospedaliero reale, in cui coesistono diverse applicazioni di diversi produttori, i quali forniscono soluzioni diversificate per una serie di diverse attività. Quindi si va a verificare la corretta implementazione delle specifiche dei profili con una serie di test. I test riguardano lo scambio di dati secondo appunto i requisiti del profilo in esame. Le valutazioni vengono fatte da esperti indipendenti che sono scelti sulla base della loro esperienza e conoscenza del settore. Se un sistema supera i test per determinati profili, i suoi produttori possono emettere l'*IHE Integration Statement*, documento contenente i dettagli dei profili per i quali il sistema ha superato i test. Questi dettagli sono anche disponibili pubblicamente sul sito IHE così che tutti possano accedervi. Si noti che quindi IHE non certifica. Semplicemente i produttori possono dichiarare che hanno superato con successo i test relativi a certi profili.

## 1.7 Stato attuale

Fino ad ora si è parlato degli aspetti teorici ed ideali dell’interoperabilità dei dati e integrazione di sistemi. E’ quindi opportuno andare a analizzare la realtà dei fatti per capire qual è il livello attuale dell’informatizzazione degli ospedali in Italia, ma non solo. Per prima cosa si va ad illustrare la situazione generale italiana paragonata al resto dell’Europa, e in seguito si vanno a presentare alcuni prodotti disponibili sul mercato italiano.

### 1.7.1 Sanità digitale in Italia e in Europa

Le informazioni che sono riportate in questa sottosezione provengono dallo studio [5]. I progressi dei paesi europei in termini di sanità digitale vengono calcolati sulla base di quattro indicatori:

- 1) Ricerca di informazioni online sui temi della salute da parte dei cittadini
- 2) Prenotazione visite mediche via Web da parte dei pazienti

- 3) Medici di medicina generale che inviano elettronicamente le prescrizioni ai farmacisti
- 4) Medici di medicina generale che condividono i dati medici dei pazienti con altri operatori e professionisti sanitari

Ora si andrà a specificare per ogni indicatore il piazzamento dell’Italia rispetto agli altri paesi europei. I paesi europei sono 28+2 (Islanda e Norvegia).

- 1) l’Italia è solo al 27esimo posto. Nei tre mesi precedenti allo studio la percentuale di utenti che cercano informazioni sulla salute era del 46% contro la media europea del 56%. Emerge anche che nel 2007 l’Italia era pari alla media europea, ma nel periodo successivo ha perso terreno

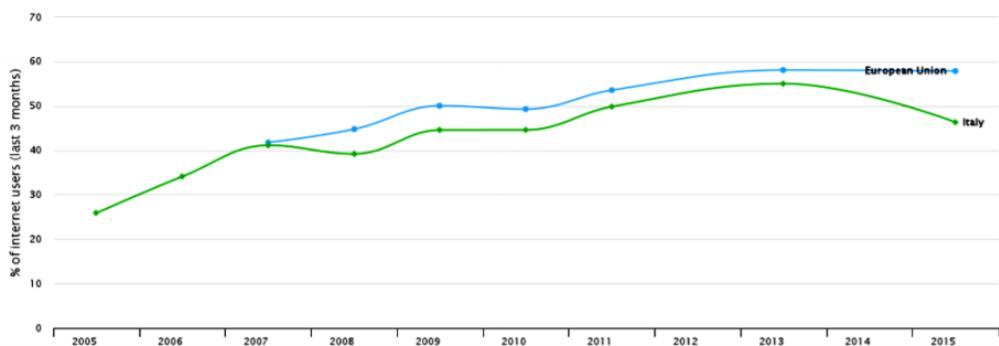


Figura 1.9: Ricerca di informazioni online sui temi della salute da parte dei cittadini [5]

- 2) La situazione in questo caso è migliore (rispetto a molti altri paesi europei). L’Italia è al 12esimo posto con il risultato del 10%. La media europea è comunque superiore (12.5%), però questo è dato dal fatto che i paesi più avanzati (Spagna, Finlandia e Danimarca) hanno ottenuto risultati di gran lunga superiori agli altri. In generale comunque nel corso del tempo l’Italia è sempre stata leggermente inferiore alla media dell’unione europea in questo indicatore

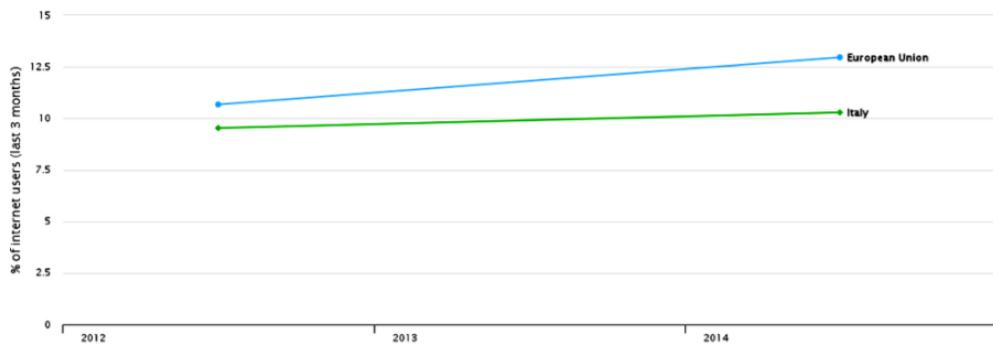


Figura 1.10: Prenotazione visite mediche via Web da parte dei pazienti [5]

- 3) L'unico dato disponibile per questo indicatore risale al 2013. L'Italia è al 17esimo posto con solo il 9%. Dato interessante è che in molti paesi il valore è tra il 90% e il 100%, o comunque di gran lunga superiore al valore raggiunto dall'Italia

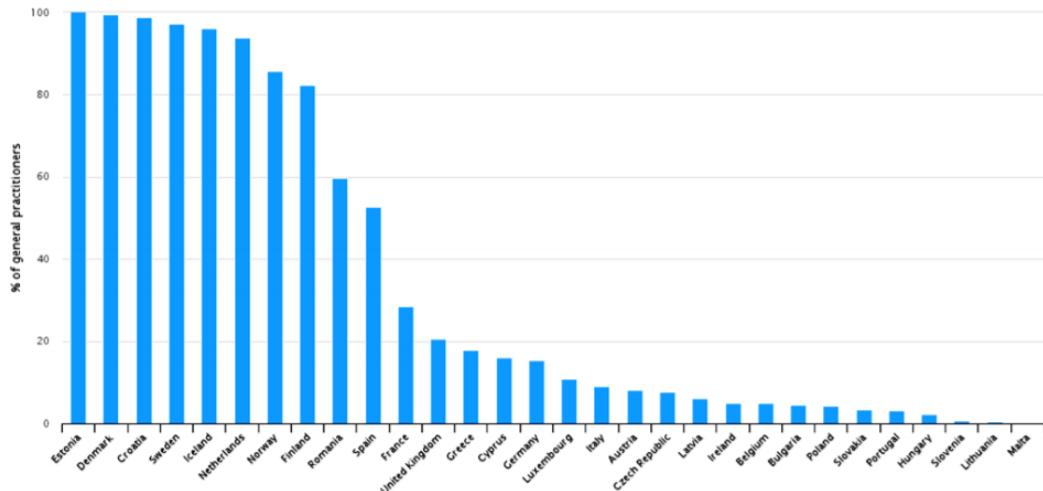


Figura 1.11: Medici di medicina generale che inviano elettronicamente le prescrizioni ai farmacisti [5]

- 4) Anche per questo indicatore l'unico dato disponibile risale al 2013. L'Italia è al 14esimo posto con il 31%. Si noti che nonostante la posizione sia a metà, molti paesi hanno riscontrato risultati di gran lunga superiori

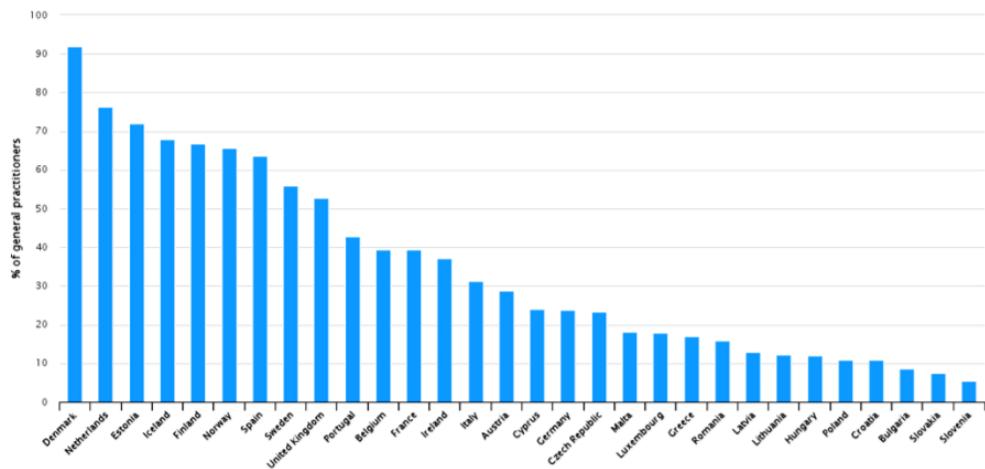


Figura 1.12: Medici di medicina generale che condividono i dati medici dei pazienti con altri operatori e professionisti sanitari [5]

Infine viene presentato un indicatore generale che fa una sintesi di tutti gli altri. L’Italia ottiene 0,26 (su un massimo di 1). I migliori paesi europei secondo lo studio sono:

- Danimarca (0,87)
- Finlandia (0,84)
- Spagna (0,72)
- Olanda (0,71)
- Svezia (0,67)

L’Italia quindi in generale ha ottenuto scarsi risultati comparati alle altre realtà europee, e rimane molto distante dalle migliori di esse. Si ritiene che gli scarsi risultati ottenuti siano causati dalla bassa spesa effettuata nel campo della sanità digitale. Nel 2015 infatti in Italia si è speso solo l’1,2% della spesa sanitaria pubblica in sanità digitale, mentre in Europa la media è del 2-3%, con punte del 4%.

### 1.7.2 Alcune aziende italiane

Sul territorio italiano sono diverse le aziende che si occupano dello sviluppo di soluzioni informatiche in ambito healthcare. Di seguito ne vengono mostrati alcuni esempi.

## NoemaLife

NoemaLife è un'azienda che opera anche a livello europeo che si occupa di sistemi software che riguardano il settore dell'healthcare. Come esempio è stato preso il software *Galileo*, che ha superato il Connechtacon di IHE su ben 13 profili. Con questi requisiti questo software si presta come buon esempio di stato dell'arte.

**NoemaLife - Galileo** Galileo è un framework applicativo che coinvolge tantissimi processi che coinvolgono le diverse organizzazioni sanitarie. E' composto da tanti moduli che possono ovviamente convididere le informazioni tra di loro. L'architettura di Galileo si può consultare in figura 1.13:

- *Patient Administration System*: moduli che si occupano della gestione dei pazienti interni (ADT centralizzata, ADT di reparti..) e esterni (pronto soccorso, CUP..), sia dal punto di vista amministrativo sia finanziario
- *Registries*: si occupa della gestione dei dati anagrafici dei pazienti e dell'associazione di codici univoci identificativi alle persone fisiche
- *Sistemi specialistici*: gestione dei flussi di alcuni processi dipartimentali importanti come la sala operatoria, oncologia, cardiologia, pronto soccorso..
- *Territorio*: moduli che si occupano di garantire la continuità della cura sul territorio (assistenza domiciliare, portale per medici e pazienti, fascicolo sanitario elettronico..)
- *EEMR - Enterprise Electronic Medical Record*: moduli per la gestione dell'EMR con interfaccia utente:
  - *Clinical Data Repository*: possibilità di accedere in tempo reale da parte del personale sanitario alla completa storia clinica dei pazienti
  - *Order Management*: sistema di gestione e monitoraggio delle richieste provenienti dalle corsie e dai servizi diagnostici
  - *Cartella clinica elettronica*: funzioni per gestire la cartella clinica elettronica (dall'accettazione alla dimissione del paziente)
  - *E-Prescribing*: funzioni dedicate alle prescrizioni farmacologiche ed alla loro somministrazione, basandosi sui costi e sulla sicurezza del paziente
  - *Galileo Nursing*: moduli per il supporto all'assistenza del paziente nelle varie fasi, ovvero dalla valutazione alla pianificazione degli interventi alla valutazione dei risultati ottenuti

- *Galileo Mobile*: interfacce ottimizzate per il mobile, e in particolare per i tablet, in modo da garantire una maggiore qualità del servizio anche in mobilità
- *Galileo eWhiteboard*: lavagna touchscreen che serve per avere sotto controllo notifiche, allarmi e urgenze

L’architettura di Galileo vanta diverse proprietà importanti:

- *Interoperabilità*: sia per quanto riguarda le componenti stesse di Galileo sia per quanto riguarda l’apertura nei confronti di sistemi esterni.
  - Interno: viene usata una Service Oriented Architecture con modello dei dati basati su HL7. I componenti di Galileo possono integrarsi tra di loro grazie alla piattaforma eHealth Application Platform che fornisce i servizi di comunicazione e una serie di componenti infrastrutturali comuni a tutte le applicazioni (resi disponibili tramite Web Services e API)
  - Esterno: si ottiene apertura verso l’esterno tramite un gateway che fornisce messaggi usando i protocolli più usati, come HL7, DICOM, XML
- *Flessibilità*: le interfacce utente sono molto flessibili in quanto possono essere personalizzate per venire incontro ai bisogni di qualunque possibile utilizzatore. Si punta anche all’implementazione di interfacce da usare in mobilità
- *Sicurezza*: i dati clinici sono ovviamente molto critici e deve essere garantito il maggiore livello di sicurezza possibile. Per fare questo sono state implementate diverse soluzioni:
  - Identificazione dei pazienti tramite codice a barre
  - Identificazione degli utenti tramite username e password, smartcard o altro
  - Accesso di emergenza
  - Uso di Secure Communication Encryption
  - Supporto per la firma digitale e per la cifratura a chiave asimmetrica
  - Tracciabilità delle azioni degli utenti
  - Gestione dei diritti d’accesso basata su profili

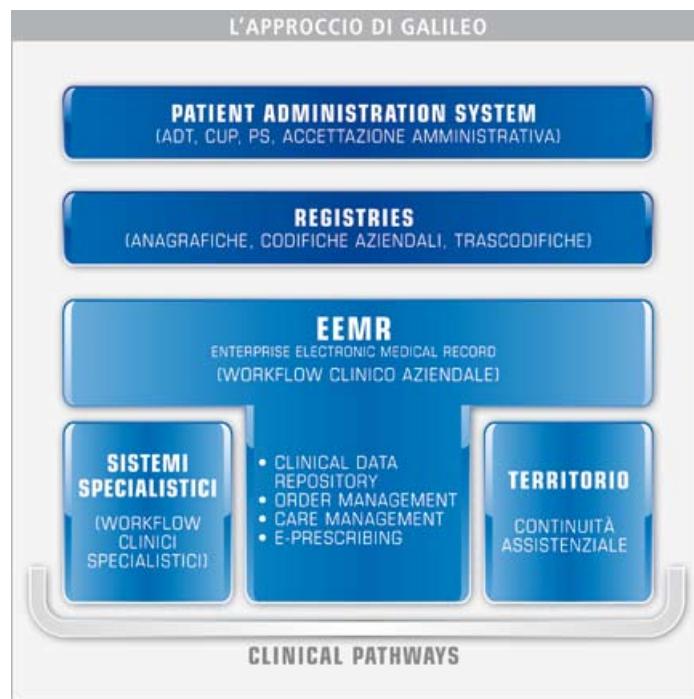


Figura 1.13: Architettura piattaforma Galileo [38]

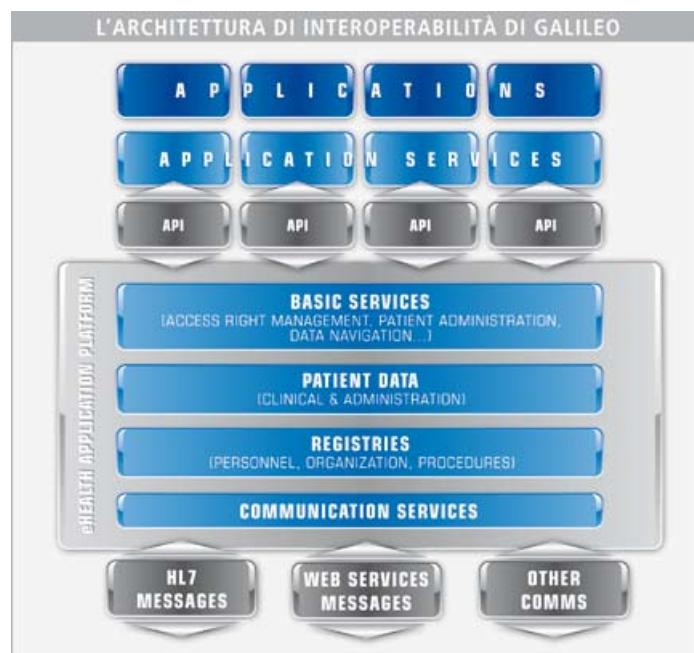


Figura 1.14: Interoperabilità piattaforma Galileo [38]

## Onit

Anche Onit produce software rivolto all'ambito sanitario, e conta clienti in tutt'Italia. Onit si occupa sia di sanità pubblica sia di sanità privata, fornendo una piattaforma ERP chiamata On.Health, che è suddivisa in moduli sulla base dell'ambito applicativo di riferimento. Il tutto è integrabile con i sistemi informativi aziendali, regionali e il FSE (Fascicolo Sanitario Elettronico). Per quanto riguarda la comunicazione con servizi esterni, si usa il protocollo SOAP (Simple Object Access Protocol), i Web Services e lo standard HL7. Il sistema, insieme a tutti i suoi moduli, ha ottenuto alcune certificazioni IHE.

**Onit - On.Health** OnHealth si compone di diversi moduli, di cui alcuni dedicati alla sanità pubblica, e uno dedicato alla sanità privata. Di seguito verranno quindi brevemente elencati i moduli del sistema. Si veda anche la figura 1.15.

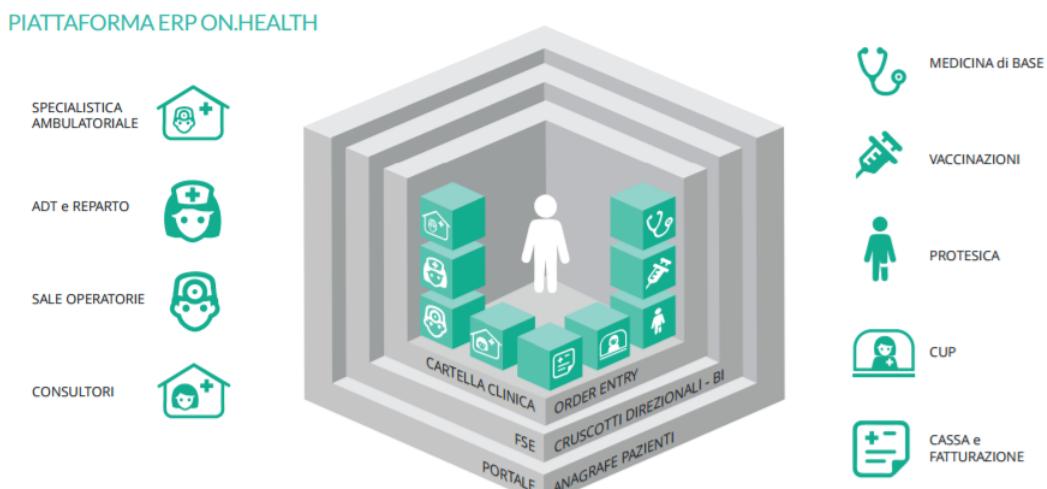


Figura 1.15: Architettura piattaforma On.Health [39]

## Onit - Sanità pubblica

- *On.Health/CUP*: modulo applicativo web based, che si occupa della gestione delle attività di programmazione (agende CUP, CIP, SOVRACUP e liste di attesa) e della prenotazione di visite e prestazioni erogabili da strutture sanitarie (sia in regime di SSN sia di Libera Professione Intramoenia)
- *On.Health/Cassa*: modulo applicativo web based che si occupa della completa gestione delle attività di fatturazione e cassa. In particolare

delle attività di registrazione degli incassi, di emissione dei documenti (fatture, ricevute, note di accredito, etc.) e di contabilizzazione delle prestazioni e beni erogati

- *On.Health/Ambulatoriale*: modulo applicativo web based che si occupa della completa gestione dell'attività specialistica ambulatoriale. In particolare dell'agenda interna e della prenotazione, accettazione, erogazione e refertazione di qualsiasi prestazione ambulatoriale
- *On.Health/Consultori*: modulo applicativo web based che si occupa della completa gestione dell'attività dei consultori. In particolare delle attività di programmazione e prenotazione, accettazione, erogazione e refertazione di qualsiasi prestazione erogabile dai consultori
- *On.Health/Vaccinazioni*: modulo applicativo web based che si occupa della gestione delle vaccinazioni (sia obbligatorie sia richieste dai pazienti). La registrazione da remoto dei singoli eventi e l'accesso è consentito anche ai MMG, PLS e personale di Pronto Soccorso.
- *On.Health/Ingegneria Clinica*: modulo applicativo web based che si occupa della completa gestione del Servizio di Ingegneria Clinica all'interno di una struttura sanitaria pubblica o privata. Si può gestire l'anagrafe della strumentazione biomedicale, tracciare e gestire le manutenzioni e verifiche effettuate e da effettuare sulle apparecchiature

I moduli, dove necessario, sono integrati con il laboratorio analisi (integrazione con il LIS) e con la diagnostica (integrazione con RIS e PACS). Ad esempio nel modulo CUP per gestire le prenotazioni appunto relative al laboratorio analisi ed alla diagnostica.

### Onit - Sanità privata

- *On.Health/Poliambulatori*: modulo applicativo web based che si occupa della completa gestione dell'attività specialistica ambulatoriale, fisioterapica e della medicina dello Sport. In particolare della gestione dell'agenda e della prenotazione, accettazione, erogazione e refertazione di qualsiasi prestazione ambulatoriale di questi ambiti

### Log 80

Log 80 è un'azienda che si occupa anche di sviluppare software per l'HealthCare che viene utilizzato da diverse strutture sanitarie italiane (tra cui l'ospedale Bufalini di Cesena). In particolare vengono sviluppati dei moduli in

tecnologie web (PHP e database MySQL) che sono rivolti ai diversi aspetti e processi dell'ambito healthcare. La lista dei moduli forniti si può vedere in figura 1.16.

+ *	Anatomia Patologica
+ *	Sanità Territoriale
+ *	Cartella Clinica Elettronica Degenze e Day Hospital
+ *	POLO 80
+ *	Gestionale Immunotrasfusionale
+ *	Gestionale Libera Professione
+ *	Gestionale Medico competente e cartella di sorveglianza sanitaria
+ *	Gestionale strutture socio assistenziale
+ *	Gestionale Poliambulatori Privati
+ *	TERA80: Gestionale Terapia Farmacologica
+ *	Cartella Clinica Ambulatoriale e Day Service
+ *	Area Oncologia
+ *	Pronto Soccorso (Comprensivo di modulo per la gestione delle liste d'attesa, auto medica, ...)
+ *	Gestione Rischio Clinico

Figura 1.16: Moduli forniti da Log 80 [40]

## Capitolo 2

# Progetto TraumaTracker e Infrastruttura GT2

All'interno dell'ospedale convivono diversi sistemi. Come descritto in precedenza sono presenti sul mercato diverse soluzioni di software, ed inoltre sono presenti anche dei produttori che si occupano non solo di software ma anche di hardware, concentrandosi sullo sviluppo di macchinari ospedalieri. I sistemi potrebbero (e dovrebbero) essere integrati tra loro utilizzando degli standard tra i quali quelli descritti nel capitolo precedente, e tramite i profili IHE possono svolgere questo compito più facilmente. Le soluzioni software principali sono rivolte alla gestione della grossa mole di dati che fluiscono nell'ospedale e nei suoi processi. Tutto questo però ai giorni nostri non è più sufficiente, in quanto la crescita ed il miglioramento di tecnologie che riguardano il mobile computing, il wearable computing, il pervasive computing, l'hands free computing ed altro non possono essere ignorate. Queste tecnologie possono e devono essere applicate all'ambito healthcare per semplificare ed ottimizzare i processi ospedalieri in contesti specifici. A questo scopo è necessario lo sviluppo di infrastrutture rivolte alla semplificazione dello sviluppo ed ottimizzazione del funzionamento di applicazioni smart che in particolare utilizzano le tecnologie menzionate e che spesso sono rivolte a contesti specifici. Difatti esistono problematiche che non possono essere ignorate:

- Scarsa durata della batteria
- Ridotte risorse computazionali

Le infrastrutture sono quindi necessarie, da una parte per fare in modo di svolgere operazioni complesse come l'interfacciamento con i sistemi e macchinari esistenti all'interno dell'ospedale in modo tale che non siano svolte dai dispositivi mobile, e dall'altra, nello stesso modo, per cercare di far consumare meno batteria possibile ai suddetti dispositivi. Oltre a ciò si aggiunge

anche la possibilità di riusare il software sviluppato, poichè se si incapsula in un servizio l’interfacciamento con un macchinario, tutte le nuove applicazioni potranno usufruire del servizio senza dover implementare nulla a riguardo, e anche di standardizzare il modo in cui i dati vengono forniti alle applicazioni. Inizialmente verrà descritto il background del progetto TraumaTracker, ovvero la gestione dei traumi, e di conseguenza come le tecnologie possono migliorarla. Successivamente si illustra il progetto TraumaTracker, sviluppato in collaborazione con l’ospedale Bufalini di Cesena, e infine si andrà a descrivere l’infrastruttura che si vuole creare al suo interno per abilitare in futuro sempre più applicazioni smart.

## 2.1 La gestione dei traumi

La cosiddetta *Trauma Resuscitation* si riferisce alle cure iniziali a cui vengono sottoposti i pazienti che hanno subito dei traumi di natura fisica. Dopo il trasporto in ospedale essi vengono trasferiti nel dipartimento d’emergenza dove viene proseguito il loro soccorso già iniziato durante il trasporto. Il team è composto da diverse figure professionali, solitamente eterogenee nelle conoscenze e specializzazioni, tra cui spicca il *Trauma Leader*, che si occupa della coordinazione delle operazioni. È molto importante documentare in modo appropriato e completo tutte le operazioni effettuate, sia al fine di migliorare il risultato finale sul paziente, sia al fine di migliorare le prestazioni del team di soccorso. Difatti le informazioni raccolte vengono analizzate per migliorare il processo di soccorso, in modo tale da capire le prestazioni di esso, i problemi che ci sono stati e così via. Quello che solitamente accade, in particolare in Europa, è che la documentazione viene scritta a mano dopo che l’intervento è concluso. I paper prodotti vengono quindi inviati ad un ufficio centrale che si occupa di inserire manualmente i dati in una banca dati. Questo processo, oltre ad essere inutilmente complesso e costoso, ovviamente porta ad avere dati incompleti, errati e quindi porta a numerosi problemi anche futuri in quanto la documentazione viene utilizzata nel decision making, nell’analisi etc. La maggior parte delle informazioni vengono infatti documentate dopo l’intervento, in quanto per la natura dello stesso, risulta complicato prendere numerose note durante il suo svolgimento in quanto è necessario perdere meno tempo possibile ed essere veloci. Oltre a questo si aggiunge anche il fatto che molte operazioni vengono effettuate nello stesso momento da diversi membri del team. Infine chi si occupa della documentazione spesso non si occupa solamente di quello ma effettua anche altre operazioni ed attività. L’impiego di tecnologie all’avanguardia può sicuramente aiutare a acquisire automaticamente delle informazioni, oltre anche a potenziare tutto il processo di soccorso

raccogliendo informazioni aggiuntive (ad esempio raccogliendo periodicamente i parametri vitali del paziente) e, migliorando la qualità e quantità dei dati raccolti, oltre a ridurre gli errori, migliorare anche le analisi successive con conseguenze che non possono che essere positive.

### 2.1.1 Pervasive Health

La pervasive health è una disciplina scientifica che si occupa dell'impiego di tecnologie e dispositivi ai fini di migliorare la qualità della cura del paziente. I contributi a questo settore provengono da diverse branche:

- *Ingegneria biomedica*: produzione di dispositivi medici, di diagnostica, protesi e così via
- *Informatica*: gli informatici lavorano con le tecnologie, protocolli, standard e dispositivi che permettono di ottimizzare l'acquisizione e l'utilizzo dei dati relativi all'ambito medico

La pervasive health è strettamente collegata con la branca dell'*ubiquitous computing/pervasive computing*, che si occupa di sviluppare e studiare sistemi che utilizzano tecnologie mobili, sensori etc. che vengono sparsi nell'ambiente in cui si opera. Il contesto in cui i dispositivi vengono posti è molto importante e, se le applicazioni sono context-aware, possono fare in modo che gli operatori abbiano le informazioni di cui hanno bisogno nel luogo e tempo in cui le necessitano. Questa disciplina si pone come obiettivo sia quello di migliorare la qualità della cura all'interno dell'ospedale e di altre strutture ospedaliere, sia quello di continuare le cure anche al di fuori di esse, monitorando costantemente il paziente anche da casa, in modo tale da essere proattivi nella decisione delle cure. Negli ultimi anni il progresso tecnologico è stato notevole, sia per quanto riguarda le tecnologie di salvataggio, elaborazione, modellazione, presentazione dei dati, sia per quanto riguarda in particolare:

- *Mobile computing*: nonostante siano ancora presenti problematiche relative alla durata della batteria, le tecnologie mobili (in particolare smartphone e tablet) sono migliorate tantissimo e offrono una potenza di calcolo ormai sufficiente per essere impiegate in contesti come l'HealthCare. I sistemi utilizzati dai medici (e anche dai pazienti) possono essere utilizzati anche in mobilità e da remoto, schermi touch e tablet possono essere utilizzati nei vari reparti per i più disparati scopi, beacon e qrcode possono essere utilizzati per la localizzazione in modo tale che le applicazioni sappiano dove stanno venendo utilizzate (per adattarsi quindi al contesto) e così via

- *Hands Free*: tecnologie hands free come gli smartglass possono sicuramente essere impiegate in questi contesti per migliorare i processi. Il grande vantaggio delle tecnologie di questo tipo è il fatto che è possibile utilizzarle senza interferire particolarmente con le operazioni da svolgere
- *Body Sensor Network*: sensori che devono essere distribuiti nel corpo dei pazienti nel modo meno invasivo possibile, al fine di raccogliere parametri vitali, effettuare monitoraggio e riscontrare eventuali anomalie non appena queste si verificano

Ai giorni nostri è impensabile non utilizzare tutte le tecnologie che si hanno a disposizione per migliorare la qualità della cura dei pazienti, sia fuori sia dentro agli ospedali. Per questa ragione è oggi possibile concepire un sistema come TraumaTracker che utilizza proprio queste tecnologie per sopperire a delle problematiche che possono causare numerosi problemi. Utilizzando dispositivi e tecnologie all'avanguardia nei vari contesti all'interno ed esterno dell'ospedale è possibile raccogliere informazioni automaticamente su quello che accade e migliorare sia i processi sia l'analisi successiva ad essi.

## 2.2 TraumaTracker

Di seguito verrà illustrato il progetto TraumaTracker, svolto in collaborazione con l'ospedale Bufalini di Cesena. Il tutto è nato da alcune necessità dei medici del pronto soccorso. Come scritto in precedenza, quando vengono portati dei pazienti al pronto soccorso con l'ambulanza, un team di medici deve prendersi carico di essi ed effettuare delle operazioni, manovre, somministrazioni di farmaci.. Successivamente i pazienti vengono smistati nei vari reparti. La problematica riguarda il fatto che a fine giornata i suddetti medici devono stilare dei report contenenti appunto le operazioni effettuate, con tutte le problematiche di dimenticanze ed errori che ne conseguono. Per questa ragione è stata comunicata la necessità di informatizzare il processo in modo tale da permettere la compilazione automatica dei report. Il progetto è stato svolto da un team composto da diverse persone, tra cui oltre a me professori, dottorandi e studenti. I requisiti sono stati stilati grazie ad alcune riunioni e contatti via mail con i medici interessati al progetto. Durante le riunioni venivano mostrati prototipi delle varie parti del sistema, in modo tale da ricevere feedback. Per questa ragione l'approccio di sviluppo è stato agile e orientato ai prototipi. Dopo la definizione dei requisiti, si andava ad implementare il tutto ed a sottoporlo poi al giudizio ed ai nuovi feedback dei medici per raffinare i prototipi.

### 2.2.1 Requisiti

Dai feedback dei medici coinvolti nel progetto sono emersi diversi requisiti:

- Generazione automatica del report dell'intervento effettuato, comprensivo di:
  - Luogo dove sono state effettuate le operazioni
  - Manovre effettuate
  - Farmaci somministrati, con quantità della dose
  - Eventuali foto
  - Eventuali note
  - Orario in cui le operazioni sono effettuate (manovre effettuate, somministrazione farmaci, richiesta parametri vitali)
  - Visualizzazione dei parametri vitali senza la necessità di dover guardare il monitor associato al paziente e inserimento automatico di essi nel report non appena vengono richiesti

Oltre a questo, si aggiunge anche una caratteristica fondamentale che il sistema deve necessariamente possedere affinché il tutto sia applicabile:

- Il sistema non deve intralciare il lavoro del medico (quindi i dispositivi che saranno impiegati non devono essere un ostacolo al compimento del lavoro da parte del medico). In particolare le mani dovranno essere necessariamente libere

Questi requisiti soddisfano la necessità della memorizzazione dei report in modo automatico. Tuttavia è possibile aggiungere altre funzionalità al sistema in modo tale che il processo venga ulteriormente migliorato:

- Possibilità di gestire i report (visualizzazione, modifica, eliminazione ed esportazione del report)
- Possibilità di visualizzare i parametri vitali riportati nei vari monitor da remoto

### 2.2.2 Analisi e Dispositivi utilizzati

Ora è necessario andare più nel dettaglio per elaborare meglio i requisiti. In base ai requisiti e la necessità che il sistema sia hands free, è necessario effettuare delle considerazioni e delle scelte tecnologie adeguate, che non possono che ricadere sulle tecnologie mobili. In particolare, va considerato che:

- Affinchè il sistema sia hands free, è necessario trovare una soluzione per permettere agli operatori di comunicare al sistema le operazioni effettuate. Come accennato, sicuramente le tecnologie mobili sono d'obbligo. Si è quindi deciso di utilizzare i comandi vocali per comunicare al sistema le operazioni che l'operatore effettua. Si è scelto quindi di sviluppare un'applicazione per smartphone che serva principalmente all'elaborazione dei comandi vocali vitali e all'invio del report al server (si veda in seguito), e di un'applicazione su smartglass per permettere la visualizzazione dei feedback sullo stato del riconoscimento, la visualizzazione dei parametri vitali e lo scatto di foto
- Per recuperare i parametri vitali, è necessario interfacciarsi con il server gateway sviluppato da Draeger. Difatti i monitor collegati con i pazienti inviano i dati dei pazienti a determinati server gateway, e per questa ragione per ottenerli è necessario comunicare con essi. I server accettano connessioni TCP e forniscono i dati in HL7. Bisognerà quindi sviluppare un'applicativo che possa richiedere i dati e possa elaborarli in HL7 per estrarre le informazioni richieste. Si è deciso di utilizzare la libreria Java HAPI per costruire l'applicativo, in modo da avere un buon supporto ad oggetti per modellare client, server e messaggi HL7
- E' opportuno nascondere la complessità dell'interfacciamento con il gateway all'applicativo che dovrà usufruire dei dati. Per questa ragione si è deciso di costruire un back end che fornisca delle API REST alle applicazioni in modo tale da fornire un'interfaccia semplificata e standard. Questo serve anche a non replicare il lavoro da svolgere in caso in cui in futuro nuove applicazioni necessitino dell'interfacciamento con il gateway. Si è deciso di costruire il server tramite Vert.x, potente framework che sfrutta il modello ad event loop e mette a disposizione numerosi strumenti aggiuntivi (gestione di task che richiedono thread, creazione di server HTTP, API REST e tanto altro)
- E' conveniente prevedere un applicativo che permetta di visualizzare i parametri vitali da remoto e permetta la manipolazione dei monitor da salvare nel sistema, in modo che le applicazioni che debbano richiedere i parametri vitali relativi ad un monitor debbano conoscere solo l'identificativo del monitor di interesse, e che sia il sistema a conoscere tutte le informazioni relative ai monitor per interrogare il gateway corretto
- E' necessario salvare i report, perciò si è scelto di salvarli in un database in modo tale da poterli recuperare e manipolare. A tal fine si è scelto di utilizzare l'approccio NoSQL (e nello specifico MongoDB) per ottenere

ottime prestazioni e per potere essere più flessibili per quanto riguarda la struttura dei dati

- E' necessario prevedere un'applicazione che permetta di manipolare i report. E' stato deciso quindi di creare un'applicazione web che possa essere quindi acceduta in maniera molto semplice da browser e utilizzata anche su mobile
- Affinchè le applicazioni non debbano interfacciarsi direttamente con il database, in modo tale da semplificarle, è conveniente sviluppare un back end che tramite delle API REST offre un'interfaccia standard e semplice alle applicazioni per manipolare i report
- Si è deciso di sviluppare le parti server del sistema a microservizi. Questo semplifica la divisione del lavoro da parte del team (in questo modo un team/persona può farsi carico di un microservizio e lavorarci indipendentemente dagli altri). Quindi si è sviluppato un microservizio per l'interfacciamento con il gateway e uno per la gestione dei report

Dopo tutte queste considerazioni, sono questi i dispositivi scelti:

- *Smartphone Android*: necessario per effettuare tutte le elaborazioni di cui il sistema ha bisogno, a partire dall'elaborazione dei comandi vocali e dall'invio dei dati raccolti alla piattaforma. Esso risulta necessario perché è impossibile effettuare operazioni pesanti su dispositivi smartglass al momento essendo meno potenti dal punto di vista computazionale. Oltre a ciò non sarebbe opportuno inserire i feedback visivi nello smartphone, perchè sarebbe un ostacolo per il medico dovere sempre guardarlo
- *Smart glass*: si è scelto di utilizzare degli smart glass in modo che gli operatori non debbano mai dover occupare le mani per operazioni che non concernono l'intervento stesso. Sfruttando dei comandi vocali, è possibile comunicare al sistema l'operazione che si sta compiendo, continuando ad avere le mani libere. Inoltre risulta molto comodo per scattare foto continuando comunque ad avere le mani libere. Essendo in seguito emersa la richiesta dei medici di potere comunque comunicare le operazioni al sistema senza i comandi vocali ma direttamente su smartphone, è stato necessario anche sviluppare un'interfaccia grafica su smartphone in modo tale da appunto poter inserire le operazioni anche manualmente. Il modello utilizzato è Vusix M-100/M-300. La comunicazione con lo smartphone avviene via BlueTooth.
- *Beacon*: sono necessari per determinare il luogo in cui le operazioni vengono effettuate

### **2.2.3 Architettura del sistema**

Di seguito viene mostrata l'architettura del sistema. I macro componenti sono i seguenti:

- App su smartphone
- App su smartglass
- Microservizio per la gestione dei report
- Microservizio per il recupero dei parametri vitali e per la gestione dei monitor salvati nel sistema
- Web app per gestire i report
- Web app per gestire i monitor salvati nel sistema e per visualizzare da remoto i parametri vitali

Si è deciso di sviluppare due microservizi separati (ReportsService e GatewayService) poichè, nonostante per il momento GatewayService venga utilizzato solo per operazioni relative a TraumaTracker, in futuro potrebbe essere condiviso da diverse applicazioni smart, e quindi è opportuno non replicare il lavoro già svolto. Di seguito verranno illustrati più nel dettaglio tutti i componenti coinvolti.

#### **SmartphoneApp**

Applicazione in esecuzione sullo smartphone Android che si occupa di:

- Riconoscere ed elaborare comandi vocali riguardo i termini forniti dai medici, e di conseguenza:
  - Memorizzare le manovre effettuate
  - Memorizzare i farmaci somministrati (con unità di misura e quantità somministrata)
  - Memorizzare i parametri vitali del paziente (con unità di misura e valore)
- Lettura dei beacon in modo da determinare la posizione dove le operazioni sono effettuate
- Interfaccia grafica per l'inserimento manuale delle operazioni effettuate in caso in cui non si vogliano utilizzare gli smartglass con gli annessi comandi vocali

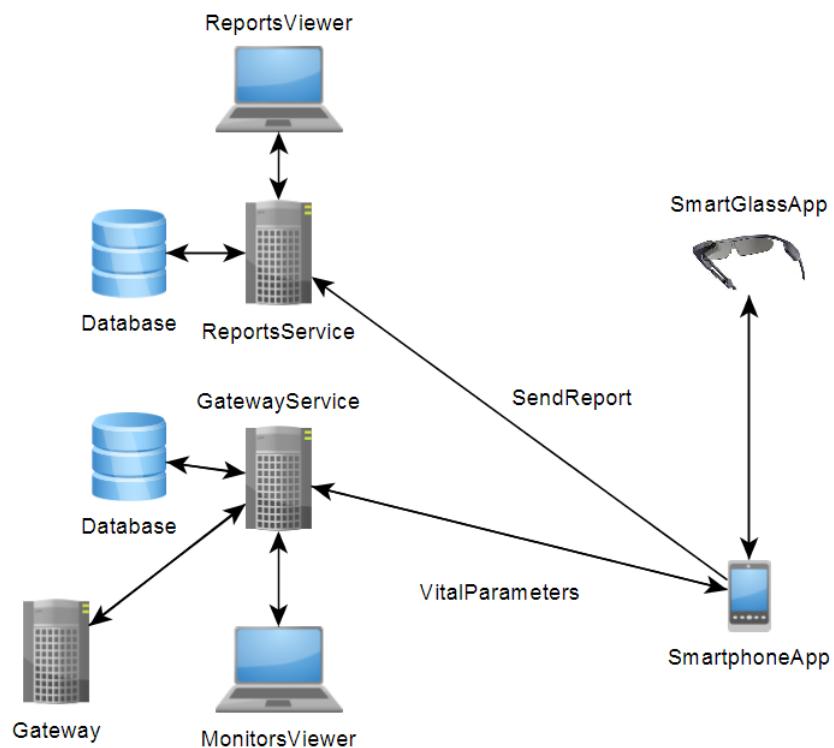


Figura 2.1: Architettura del sistema TraumaTracker

I medici hanno fornito tutti i termini di interesse al fine di implementare il riconoscimento vocale dei comandi all'interno dell'applicazione. L'intervento si compone di due possibili stati:

- *Intervento fermo*: l'intervento è fermo. È possibile iniziare la sessione tramite il comando "inizia intervento"
- *Intervento attivo*: l'intervento è attivo. La sessione si può terminare tramite il comando "fine intervento". Durante questo stato è possibile avere due stati interni:
  - Riconoscimento attivo: il riconoscimento dei termini medici è attivo. Si può mettere in pausa il riconoscimento tramite il comando "trauma pausa". Durante questo stato è possibile riconoscere i comandi medici (manovre, farmaci, parametri vitali)
  - Riconoscimento fermo: il riconoscimento dei termini medici è in pausa. Si può riprendere il riconoscimento tramite il comando "trauma ascolta"

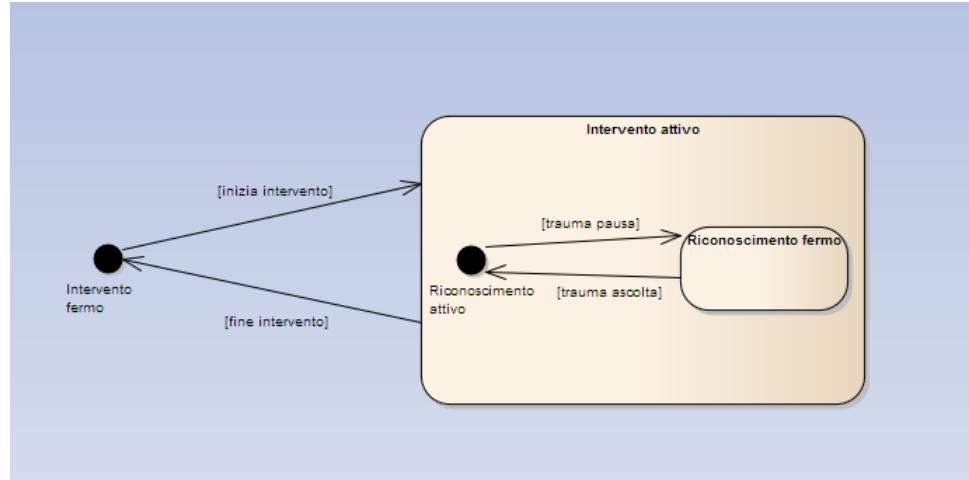


Figura 2.2: Stati dell'applicazione su smartphone

### SmartglassApp

Applicazione in esecuzione sugli smartglass. Il suo scopo è quello di fornire un'interfaccia grafica al sistema. Di seguito le sue funzionalità:

- Visualizzare feedback visivi sui comandi vocali, in particolare:

- Informare l'utilizzatore sul fatto che la modalità intervento sia attiva o meno
- Informare l'utilizzatore sul fatto che il riconoscimento vocale sia attivo o meno
- Informare l'utilizzatore sul fatto che il riconoscimento vocale sia in pausa o meno
- Informare l'utilizzatore sullo stato del riconoscimento dei termini
- Visualizzare i parametri vitali quando richiesti
- Scattare foto e visualizzare un feedback quando vengono appunto scattate

Essa comunica con lo smartphone tramite Bluetooth e riceve da esso informazioni circa lo stato del riconoscimento vocale dei comandi e le informazioni relative ai parametri vitali. Ricevendo queste informazioni, può dare feedback visivi all'operatore. L'area visibile è stata suddivisa in regioni. In ognuna di esse vengono visualizzate determinate informazioni (figura 2.3):

- *Regione 1*: la regione numero 1 occupa 1/8 dello schermo. Questa regione viene utilizzata per visualizzare più informazioni:
  - Stato dell'intervento (se attivo o no)
  - Il fatto che il riconoscimento vocale sia in pausa
  - Icona per indicare quando viene scattata una foto
- *Regione 2*: la regione numero 1 occupa 1/2 dello schermo. Viene utilizzata per visualizzare i parametri vitali
- *Regione 3*: la regione numero 1 occupa 1/4 dello schermo. Viene utilizzata per visualizzare i comandi che vengono riconosciuti
- *Regione 4*: la regione numero 4 occupa 1/8 dello schermo. In questa regione viene visualizzato lo stato del riconoscitore dei comandi vocali (ovvero se è possibile inviarne o no)

### ReportsService

Microservizio che si occupa di fornire delle API per la manipolazione dei report, in particolare per:

- Aggiunta di nuovi report

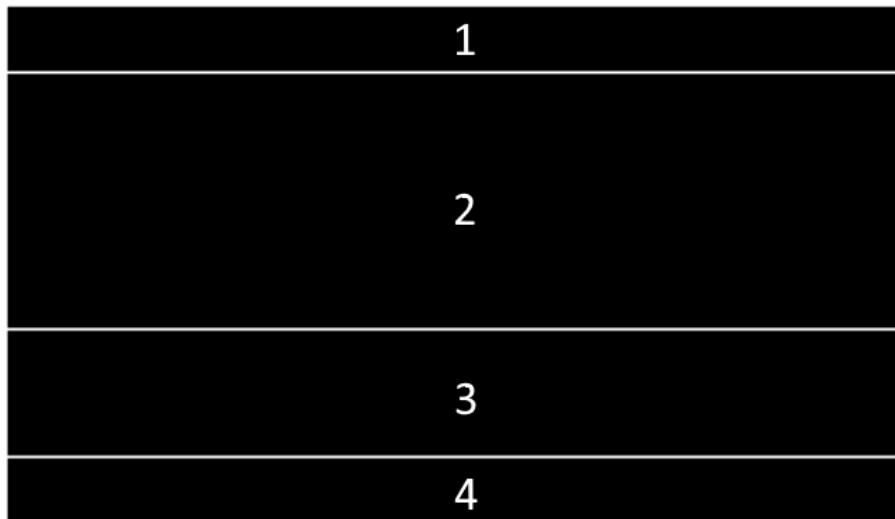


Figura 2.3: Interfaccia dell'applicazione su smartglass

- Modifica dei report
- Cancellazione dei report
- Recupero dei report

Al servizio si aggiunge anche *ReportsViewer*, applicazione web che permette di effettuare le operazioni appena nominate:

- Manipolazione dei report che il sistema ha in memoria
- Esportazione dei report anche a fini di eventuali analisi statistiche (in formato pdf e csv)

### GatewayService

Microservizio che si occupa di fornire delle API per il recupero dei parametri vitali del paziente, interrogando il gateway server di Draeger che si occupa di recuperare i dati dei monitor e di renderli disponibili alle applicazioni esterne. A tal scopo, il servizio incapsula al suo interno anche un altro componente:

- *HL7SubSystem*: componente che si occupa della connessione con il gateway di Draeger, in modo tale da richiedere i parametri vitali (restituiti in formato HL7) per poi estrarre le informazioni rilevanti

Questo microservizio deve fornire anche API per la gestione dei monitor memorizzati nel sistema (aggiunta, recupero, modifica e cancellazione di monitor).

Al servizio si aggiunge anche *Monitors Viewer*: applicazione web che permette di:

- Manipolare i monitor che il sistema ha in memoria (aggiunta, modifica e cancellazione)
- Visualizzazione dei parametri vitali rilevanti dei monitor salvati nel sistema

## 2.3 Infrastruttura GT2

Partendo dal progetto TraumaTracker, è in seguito emersa la possibilità di portare il tutto ad un livello più ampio. TraumaTracker è un'applicazione rivolta ad una singola necessità, ma in futuro potrebbe rivelarsi necessario implementare altre applicazioni smart. Oltre alle problematiche relative allo sviluppo delle applicazioni in sè, in campo medico spesso è necessario interagire e interfacciarsi con servizi, sistemi informativi ed apparati presenti all'interno dell'ospedale. Per questa ragione non avrebbe senso che ogni applicazione sviluppata debba sviluppare da zero la parte di interfacciamento ai sistemi esistenti. La visione relativa all'infrastruttura riguarda la costruzione di un livello che si ponga sopra ai sistemi esistenti e che si occupi di interfacciarsi con essi. In questo modo le applicazioni non devono interfacciarsi direttamente con i sistemi ma possono interagire con questo livello. Quindi si è pensato di sviluppare una piattaforma che permetta di fungere da punto centrale per le varie applicazioni smart all'interno dell'ospedale e che provveda a fornire loro i dati in un formato ben preciso, in modo da facilitare lo sviluppo di nuove app e l'integrazione tra esse. Ad esempio, se è necessario recuperare i parametri vitali dei pazienti tramite il gateway di Draeger, per evitare che ogni applicazione debba interfacciarsi con esso, è opportuno sviluppare ciò che serve una volta sola e inserire l'implementazione all'interno dell'infrastruttura, rendendo disponibile un'interfaccia standard per tutte le applicazioni che lo necessitino. Si è deciso di utilizzare un'approccio a microservizi, in modo tale da rendere indipendenti i vari componenti dell'infrastruttura. Tramite questo approccio ogni componente può essere sviluppato, distribuito e aggiornato indipendentemente dagli altri, senza dovere aggiornare l'intera infrastruttura quando le modifiche concernono solo una parte ristretta di essa. I microservizi utilizzeranno l'architettura REST in modo tale da fornire delle API standardizzate alle applicazioni di terze parti che avranno la necessità di consumarle. I dati saranno scambiati in formato JSON. Di seguito viene mostrata un'immagine che sintetizza i concetti appena espressi (figura 2.4).

- *Infrastruttura GT2:*

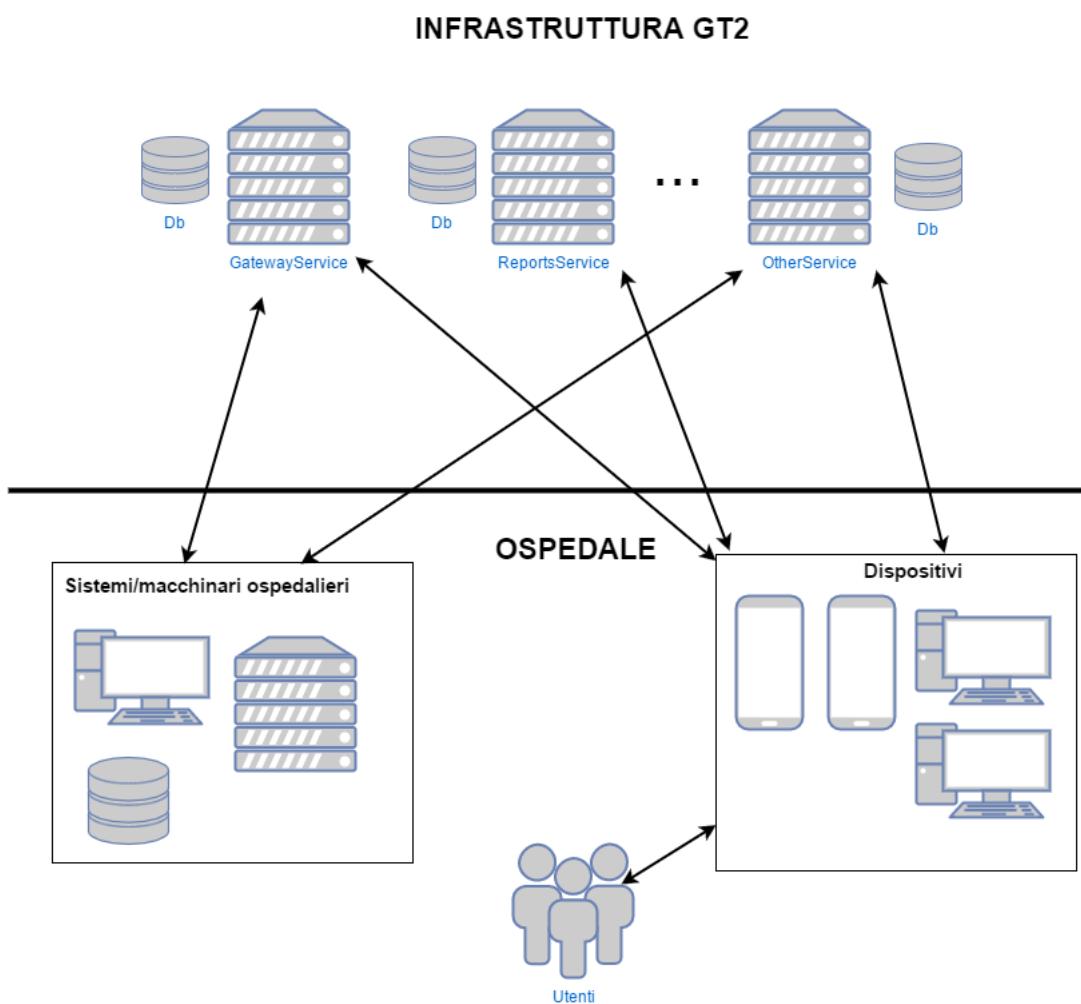


Figura 2.4: Infrastruttura GT2

- Servizi: l'infrastruttura è composta da diversi microservizi indipendenti tra loro che si occupano di fornire delle funzionalità e se necessario di interagire con i macchinari e sistemi dell'ospedale, in modo da evitare che siano le applicazioni a farlo direttamente. Per la natura dei microservizi, questi potrebbero essere sviluppati in tecnologie diverse
- Db: ogni servizio potrebbe usufruire di un database. Per la natura dei microservizi e per il fatto che essi sono indipendenti tra loro, anche i database potrebbero essere costruiti con tecnologie diverse

- *Ospedale:*

- Dispositivi: dispositivi (che possono essere sia fissi sia mobili) su cui vengono eseguite le applicazioni che vengono quindi utilizzate dagli operatori dell'ospedale (o anche dai pazienti)
- Sistemi/macchinari ospedalieri: dispositivi, macchinari e sistemi informativi che sono presenti nell'ospedale, con i quali le applicazioni potrebbero avere la necessità di interagire. Per nascondere loro la complessità dell'interazione con questi servizi, essi interagiscono direttamente con la piattaforma, che quindi si occupa di fornire i dati necessari alle applicazioni con una interfaccia standardizzata



# Capitolo 3

## GatewayService

Di seguito verrà illustrato nel dettaglio il sistema per il recupero ed il tracciamento dei parametri vitali del paziente sviluppato. Il sistema è composto da alcune parti:

- *HL7SubSystem*: applicativo che si occupa dell’interfacciamento con il gateway di Draeger per il recupero dei parametri vitali dei pazienti collegati ai monitor
- *GatewayService*: servizio che utilizza HL7SubSystem per recuperare i parametri vitali dei pazienti collegati ai monitor e renderli disponibili tramite un’interfaccia standardizzata e semplice alle applicazioni esterne, nascondendo loro la complessità dell’interfacciamento con il gateway di Draeger
- *GatewayViewer*: applicazione web che permette di manipolare i monitor salvati nel sistema (in modo tale che a fronte delle richieste ricevute GatewayService possa sapere che gateway interrogare sulla base del monitor richiesto dalle applicazioni) e di visualizzare da remoto i parametri vitali associati ai monitor

Per quanto riguarda lo sviluppo, ci si è concentrati per il momento principalmente sugli aspetti legati ai requisiti e alle funzionalità del sistema al fine di poter essere utilizzato in modo fruibile, ponendo attenzione anche ad aspetti legati alle performance. Tuttavia questo non basta per ottenere un prodotto che possa inserirsi in un contesto delicato come l’healthcare. A tal scopo bisognerà lavorare su altri punti, descritti in 3.4. Infine è opportuno anche affrontare il problema della privacy dei dati. In questo caso il servizio non colleziona, distribuisce o condivide informazioni relative ai pazienti, perciò il problema non sussiste. Il gateway di Draeger fornisce nel messaggio solamente il nome e cognome del paziente (di cui non è nemmeno certa la presenza,

in quanto il paziente viene collegato al monitor solamente all'arrivo al pronto soccorso), ma questi dati non vengono salvati (e nemmeno letti) dal servizio, che fornisce alle applicazioni solo i dati relativi ai parametri vitali.

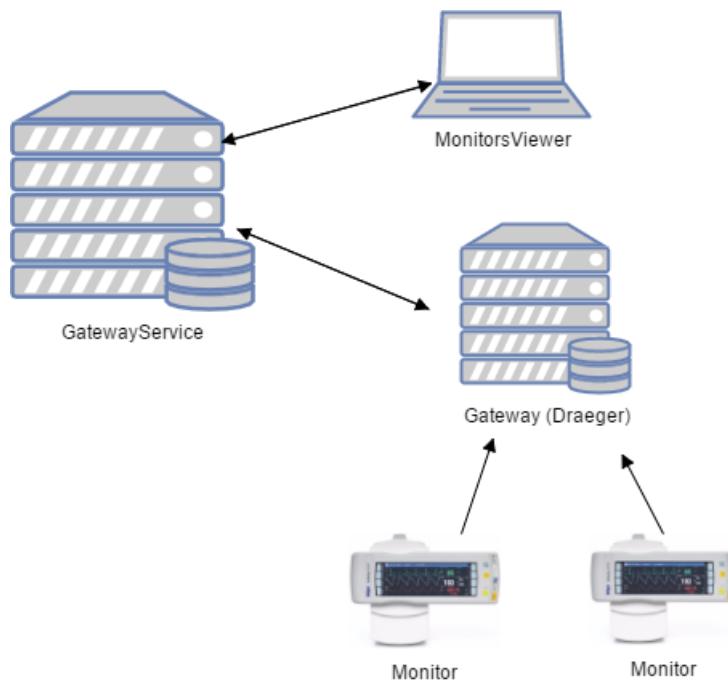


Figura 3.1: Architettura del sistema

Il software che verrà sviluppato verrà utilizzato inizialmente solamente dal progetto TraumaTracker. Il software sviluppato sarà in seguito anche posto all'interno di un quadro più ampio, ovvero l'infrastruttura GT2, che ha l'obiettivo di costruire un livello che si pone al di sopra dei sistemi software e hardware che convivono all'interno dell'ospedale.

### 3.1 HL7SubSystem

Il sottosistema sviluppato è un software che si occupa dell'interfacciamento con il gateway di Draeger. Si consideri che inizialmente il software reale del gateway non era ancora disponibile, perciò è stato necessario sviluppare un server simulato, e solo successivamente testare il funzionamento dell'applicativo con il software reale.

- *Simulatore*: sviluppo di un simulatore che debba appunto simulare la presenza di pazienti che avranno determinati parametri vitali

- *Server*: sviluppo di un server simulato che dovrà rendere disponibili a delle applicazioni client i dati dei pazienti simulati in HL7
- *Client*: sviluppo di un applicativo client che deve poter essere in grado di richiedere al server (sia simulato sia reale) i dati relativi ai parametri vitali dei pazienti, ed estrarre dai parametri ricevuti solamente quelli rilevanti (poichè il server di Draeger, alla ricezione delle richieste, invia tutti i parametri vitali del paziente all'applicativo che li richiede. Sarà infatti compito del ricevente elaborare il messaggio ricevuto per estrarne le informazioni necessarie)

Di seguito verranno approfondite le macro parti dell'applicativo e le loro interazioni. Prima di illustrare lo sviluppo è necessario analizzare l'Infinity Gateway di Draeger, con il quale il software sviluppato si dovrà interfacciare.

### 3.1.1 Infinity Gateway

Il software sviluppato si interfaccia con l'Infinity Gateway, software sviluppato da Drager. Tutte le interfacce HL7 dell'Infinity Gateway utilizzano HL7 versione 2.3 (a meno che sia specificato diversamente), implementata su socket TCP/IP, utilizzando il protocollo MLLP (Minimal Lower Layer Protocol) come protocollo di basso livello per i messaggi. Di seguito verranno descritte tutte le informazioni di rilievo focalizzandosi in particolare su quelle che concernono il progetto.

#### Il sistema

Il sistema si compone di due tipologie di elementi principali:

- *Monitor*: macchine che collezionano i dati raccolti dai sensori dei pazienti a cui sono collegate
- *Gateway server*: server che raccoglie i dati provenienti dai monitor

Ogni determinato intervallo di tempo i monitor inviano i dati raccolti sui pazienti al gateway che li rende disponibili ad applicazioni esterne tramite diverse interfacce. La rete quindi sarà composta da diversi monitor e da un gateway. Il gateway si prenderà carico della ricezione dei dati da parte dei monitor (figura 3.2).

#### Connessione

Un sistema assume il ruolo di master e l'altro lo slave. Il master inizia la connessione. Il gateway server può essere sia master sia slave, e il tutto dipende

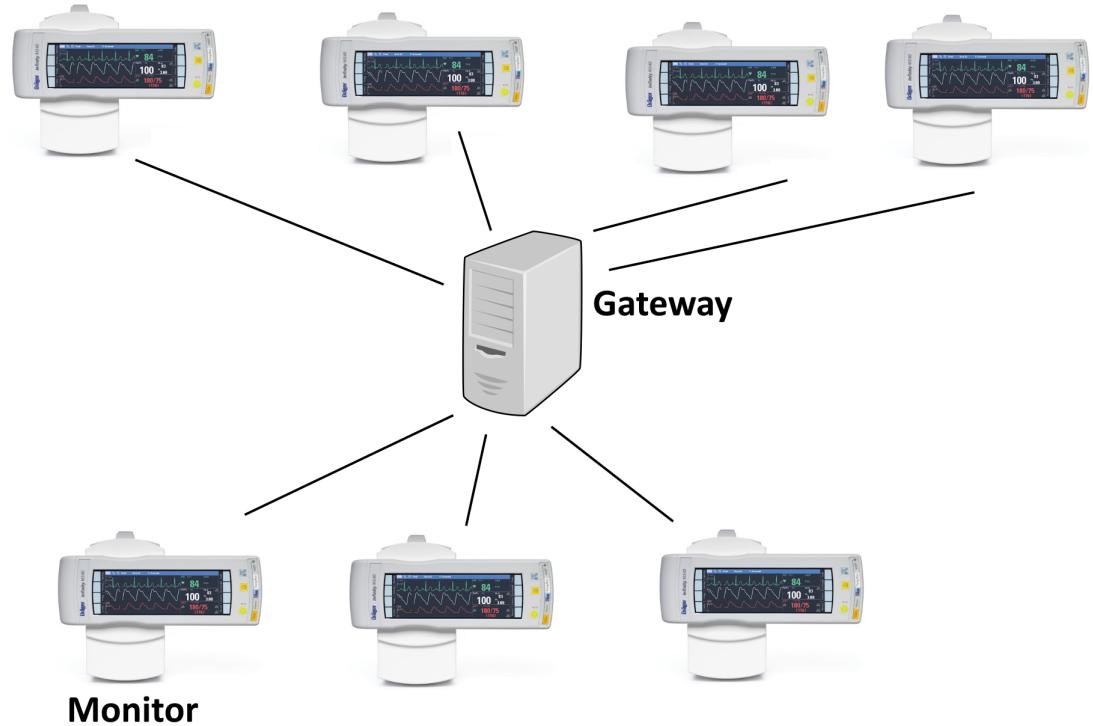


Figura 3.2: Rete costituita da gateway e monitor

dall’interfaccia utilizzata (per determinati servizi il gateway si comporterà da master e per altri da slave). La connessione si effettua specificando indirizzo IP e porta del servizio che si vuole utilizzare dell’infinity gateway. Il manuale consiglia delle porte di default e consiglia di usarle se possibile, ma non si è costretti.

**Interfacce disponibili** Il gateway ha diverse interfacce, a seconda del servizio offerto. A seconda dell’interfaccia, potrebbe essere il gateway/monitor a fornire i dati, oppure riceverli da un sistema esterno. Quella d’interesse per il progetto è ”Vital Signs solicited/unsolicited”.

Default Port	HL7 Interface
2250	ADT solicited/unsolicited
2350	Stat Lab (ASTM Access)
2450	Lab Import
2550	Vital Signs solicited/unsolicited
2650	Lab Export
2750	Order Entry Interface
2755	Order Complete Interface
2400	Schedule Inbound

Figura 3.3: Interfacce disponibili

## Messaggi

Un messaggio è composto in modo analogo ad un messaggio HL7, dato che viene appunto utilizzato il suddetto standard. Perciò è composto da:

- *Header*: header del messaggio
- *Segmenti*: un certo numero di segmenti, di cui alcuni potrebbero essere ripetuti

Il tutto varierà ovviamente sulla base del tipo di messaggio in questione.

**MLLP - Minimal Lower Layer Protocol** Il messaggio viene inviato tramite il protocollo MLLP. E’ un protocollo piuttosto basico e si sta già lavorando ad un rimpiazzo (precisamente: HL7 over HTTP). Tuttavia, dato che l’Infinity Gateway utilizza ancora HL7 versione 2, viene ancora utilizzato MLLP. Il funzionamento è il seguente:

- *Inizio del messaggio*: il messaggio deve cominciare con il carattere ASCII 0x0B (VT - Vertical Tab)
- *Fine del messaggio*: il messaggio deve terminare con il carattere ASCII 0x1C (FS - File Separator) seguito da un carriage return (0x0D - CR)
- *Segmenti del messaggio*: i segmenti che compongono il messaggio sono separati da un carriage return (0x0D - CR)

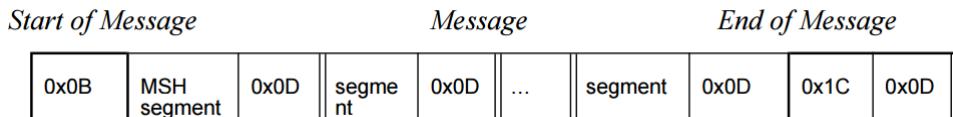


Figura 3.4: Messaggio HL7 trasmesso con MLLP

### Recupero dati vitali

E' possibile recuperare i dati vitali dei pazienti in diversi modi. Il gateway viene aggiornato con nuovi dati da parte dei monitor non appena essi variano. Queste sono le modalità disponibili:

- *Unsolicited Non-validated Vital Signs Export*: in questo caso è il gateway ad iniziare la connessione. Il gateway invia un messaggio di tipo ORU per ogni paziente ad una frequenza configurabile (tra 5 secondi e 6000 secondi). Il sistema che riceve il messaggio deve rispondere con un messaggio di tipo ACK
- *Unsolicited Validated Vital Signs Export*: simile al caso precedente, ma funziona con macchine Innovian (che riguardano la gestione dei dati per l'anestetista)
- *Unsolicited Non-validated IHE Vital Signs Export*: i dati vitali vengono inviati come nel caso precedente con messaggi ORU, però con HL7 versione 2.6 (non versione 2.3). I dati vengono inviati con nomenclature compatibili con IHE (Integrate the Healthcare Enterprise). La questione della frequenza di emissione dei messaggi rimane invariata rispetto al caso precedente, così come il fatto che sia il gateway a iniziare la connessione
- *Solicited Vital Signs Export*: in questa modalità è possibile per sistemi esterni richiedere i dati vitali. E' possibile utilizzare questa modalità solo nei gateway e non nei monitor. La richiesta avviene con un messaggio

di tipo QRY (patient QueRY). Si possono richiedere i dati sia di un solo paziente sia di pazienti multipli. I pazienti si identificano con la loro posizione all'interno della Infinity Network grazie ad una combinazione del codice di reparto e del codice del letto. Dopo la ricezione del messaggio, il gateway esporta tutti i dati ricevuti dal monitor del paziente in un messaggio ORF (il gateway riceve periodicamente i dati del paziente da parte del monitor, anche se non vengono fatte nuove misurazioni). In questo caso è il sistema esterno ad iniziare la connessione

### Solicited Vital Signs Export

La modalità solicited è quella più di interesse per il progetto, perchè consente di recuperare i dati relativi a un determinato paziente richiedendoli esplicitamente. Di seguito verrà illustrata la struttura dei messaggi relativi alla richiesta e alla risposta. Si noti che i campi dei messaggi dello standard sono tantissimi, perciò verranno riportati solamente i campi utilizzati dal gateway, dato che appunto ne vengono utilizzati solo un sottoinsieme. Le informazioni riportate qua sotto sono state ottenute dal manuale del gateway insieme e dai sample di messaggi forniti del sistema reale in funzione (che avevano leggere differenze con quanto riportato nel manuale, anche se non di rilievo).

**Richiesta** La richiesta si effettua tramite un messaggio di tipo QRY. Questo messaggio si compone di tre segmenti:

- MSH - Message Header
- QRD - Query Definition
- QRF - Query Filter

Per ognuno dei segmenti verranno descritti brevemente i campi da riportare.

Segment	Description
MSH	<a href="#">MSH Segment - Message Header</a>
QRD	<a href="#">Acknowledgment Codes</a>
QRF	<a href="#">QRF - Query Filter</a>

Figura 3.5: Messaggio QRY (richiesta parametri vitali)

- *MSH*:
  - Field Separator: il valore da inserire è ”|”

- Encoding Characters: il valore da inserire è "~~&"
- Date/Time of Message: timestamp del messaggio nella forma YYYYMM-DDHHmm. Rappresenta appunto la data in cui il messaggio viene inviato
- Message Type: indica la tipologia di messaggio. In questo caso corrisponde a "QRY^R02"
- Message Control ID: numero univoco identificativo del messaggio
- Processing ID: il valore da inserire è "P"
- Version ID: indica la versione di HL7 usata. In questo caso quindi il valore da inserire è "2.3"

- *QRD:*

- Query Date/Time: timestamp del messaggio nella forma YYYYMM-DDHHmm. Rappresenta appunto la data in cui il messaggio viene inviato
- Query Format Code: il valore da inserire è "R"
- Query Priority: il valore da inserire è "I"
- Query ID: numero univoco identificativo della query
- What Subject Filter: il valore da inserire è "RES"

- *QRF:*

- Where Subject Filter: il valore da inserire è "MONITOR"
- Other QRY Subject Filter: specifica il monitor a cui applicare la query. Per identificare un monitor, è necessario possedere il codice del reparto e il codice del lettino. Il valore da inserire nel campo deve avere questa forma: <Care Unit Label>^^<Bed Label>. E' possibile anche specificare più monitor concatenando più stringhe strutturate in questa maniera, separate con il carattere di ripetizione "~~"

**Risposta** Il gateway invia la risposta tramite un messaggio composto da diversi segmenti. Precisamente, i segmenti sono i seguenti:

- MSH Segment - Message Header
- MSA Segment - Message Acknowledgment
- QRD Segment - Query Definition

- QRF Segment - Query Filter
- PID Segment - Patient Identification
- PV1 Segment - Patient Visit
- OBR Segment - Observation Request
- OBX Segment - Observation/Result (questo campo può essere ripetuto più volte per inserire più parametri vitali)

Il blocco di segmenti PID, PV1, OBR e OBX viene ripetuto per ogni paziente nel caso in cui nella richiesta siano stati specificati più pazienti e non solo uno. Questa tipologia di messaggio è del tutto identica a un messaggio di tipo ORF, con l'aggiunta del segmento PV1.

<b>Segment</b>	<b>Description</b>
MSH	<a href="#">MSH Segment - Message Header</a>
MSA	<a href="#">MSA Segment - Message Acknowledgment</a>
QRD	<a href="#">Acknowledgment Codes</a>
QRF	<a href="#">QRF Segment – Query Filter</a>
PID	<a href="#">PID Segment - Patient Identification</a>
PV1	<a href="#">PV1 Segment - Patient Visit</a>
OBR	<a href="#">OBR Segment - Observation Request</a>
{[OBX]}	<a href="#">OBX Segment - Observation/Result</a>

Figura 3.6: Messaggio ORF (risposta contenente i parametri vitali)

- *MSH:*
  - Field Separator: il valore da inserire è ”|”
  - Encoding Characters: il valore da inserire è ”^~&”
  - Sending Application: nome dell'applicativo che invia il messaggio. Per quanto riguarda il gateway, il valore da inserire è ”INFINITY”
  - Receiving Application: nome dell'applicativo a cui è destinato il messaggio. Il valore dovrà corrispondere a quello inserito nel segmento MSH della richiesta (messaggio QRY).
  - Date/Time of Message: timestamp del messaggio nella forma YYYY-MM-DDHHmm. Rappresenta appunto la data in cui il messaggio viene inviato

- Message Type: indica la tipologia di messaggio. In questo caso corrisponde a "ORF^R04"
- Message Control ID: numero univoco identificativo del messaggio
- Processing ID: il valore da inserire è "P"
- Version ID: indica la versione di HL7 usata. In questo caso quindi il valore da inserire è "2.3"

• *MSA:*

- Acknowledgment Code: il valore del campo potrà assumere tre valori a seconda dell'esito della richiesta.
  - \* AA (Application Accept): la richiesta è stata accettata e processata
  - \* AE (Application Error): c'è stato un errore nel processare la richiesta
  - \* AR (Application Reject): la richiesta è stata rifiutata
- Message Control ID: contiene lo stesso Message Control ID del segmento MSH della richiesta (messaggio QRY)
- Text Message: campo di testo che fornisce indicazioni sullo stato della richiesta o su eventuali errori
- *QRF:* Questo segmento corrisponde esattamente al segmento QRF della richiesta (messaggio QRY).

• *PID:*

- Patient ID (Internal ID): ci sono due possibilità sulla base della configurazione del gateway. Gli ID dei pazienti possono essere interpretati come MRN (Medical Record Number) o come Patient Account Number.
  - \* MRN: viene inserito nel campo il codice identificativo del paziente
  - \* Patient Account Number: viene inserito nel campo il codice identificativo del paziente e l'indicazione che si tratta del Patient Account Number (il tutto in forma "Patient ID^AN").
- Patient Name: nome del paziente (nome, cognome e nome di mezzo se presente). I dati sono inseriti in questo formato: "<Last>^<First>^<Middle>". Se non sono presenti questi dati, il campo sarà settato a "<UNKNOWN>^^"

- Patient Account Number: ci sono due possibilità sulla base della configurazione del gateway. Gli ID dei pazienti possono essere interpretati come MRN (Medical Record Number) o come Patient Account Number
  - \* MRN: il campo è vuoto
  - \* Patient Account Number: viene inserito nel campo il codice identificativo del paziente

- *PV1:*

- Assigned Patient Location: contiene lo stesso valore del campo Other QRY Subject Filter della richiesta (messaggio QRY)

- *OBR:*

- Observation Date/Time: timestamp in formato YYYYMM- DDDHHmmss che identifica la data in cui l'osservazione è stata effettuata

- *OBX:*

- Value Type: dipende dal tipo di parametro. Se è numerico, va inserito "SN". Se invece è testuale, va inserito "ST".
  - Observation Identifier: stringa nel seguente formato:  
" <identifier1> ^ local ^ <identifier2> & <instance> ^ ^ <codingsystem> ".
    - \* <identifier1>: etichetta del parametro Drager (per esempio: HR per Heart Rate)
    - \* <identifier2>: codice del parametro (o di tipo "LOINC" o di tipo "MIB/CEN")
    - \* <instance>: istanza d'uso del codice (è un numero)
    - \* <codingsystem>: tipo di codice del parametro (o "LOINC" o "MIB/CEN")

Tutti questi valori sono reperibili in una tabella all'interno del manuale.

- Observation Value: il valore del parametro. La formattazione del campo è diversa a seconda del fatto che il parametro sia numerico o stringa.
    - \* SN: il campo contiene "= ^ <valore>"
    - \* ST: il campo contiene solo il valore

- Units: rappresenta l’unità di misura con cui il parametro è riportato. Il campo è inserito nel formato: ”<identifier>^^ISO+, dove <identifier> è il codice LOINC o MIB/CEN dell’unità di misura (ad esempio /min per o battiti per minuto)
- Observation Result Status: viene inserito nel campo il valore ”R”
- Date/Time of the Observation: timestamp in formato YYYYMM-DDHHmmss che identifica la data in cui l’osservazione è stata effettuata

### Note tecniche

- *Character set*: il gateway supporta caratteri 8-bit del character set ISO 8859-1 e caratteri 16 bit del character set Unicode (UTF-16). Di default nel gateway è impostato ISO 8859-1, ma è possibile cambiare la configurazione

### 3.1.2 Sistema simulato

Prima di interfacciarsi con il gateway reale è necessario eseguire qualche test e costruire un simulatore per emularne il funzionamento. La simulazione ha diversi vantaggi:

- Possibilità di fare dei test anche senza la presenza fisica del gateway
- Possibilità di testare in un ambiente controllato, ovvero avere il pieno controllo delle dinamiche (parametri, interazioni tra server e client..)
- Astrazione: è possibile concentrarsi solamente sugli aspetti che si ritengono rilevanti, tralasciando aspetti tecnici che non sono importanti

E’ però necessario essere accurati nella simulazione in quanto affinchè il tutto abbia senso il funzionamento reale del gateway deve essere emulato il più fedelmente possibile. Per costruire il simulatore è stata utilizzata la libreria Java HAPI (A.8). E’ quindi necessario realizzare un server e un client, dove il server dovrà simulare il gateway. Il client dovrà avere la possibilità di inviare richieste al server. I messaggi scambiati saranno ovviamente in formato HL7 in quanto è lo standard usato dal gateway reale.

### Parametri vitali

Per prima cosa è necessario analizzare i parametri vitali rilevanti. Un parametro vitale si identifica univocamente grazie ad un insieme di codici che verranno di seguito elencati:

- *Codice*: codice HL7/LOINC o MIB/CEN del parametro
- *Tipo di codice*: indica se viene utilizzato il codice di tipo HL7/LOINC o di tipo MIB/CEN
- *Tipo di dato*: indica se il parametro si rappresenta sotto forma di stringa (ST) o di numero (SN)
- *Istanza*: numero istanza del parametro
- *Label*: label del parametro

Tutti questi valori possono essere consultati nel manuale dell'Infinity Gateway. I parametri vitali verranno a breve elencati. Verranno indicati per ognuno di essi i codici identificativi, nella seguente forma: [codice, tipo di codice, tipo di dato, istanza, label]. Inoltre verrà indicata anche l'unità di misura del parametro. In figura 3.7 è possibile vedere come il tutto è stato modellato.

- Pressione diastolica [”8462-4”, ”LOINC”, ”SN”, ”1”, ”ART D”, con unità di misura ”mm(hg)”]
- Pressione sistolica [”8480-6”, ”LOINC”, ”SN”, ”1”, ”ART S”, con unità di misura ”mm(hg)”]
- Battito cardiaco [”8867-4”, ”LOINC”, ”SN”, ”1”, ”HR”, con unità di misura ”/min”]
- EtCO2 [”20824”, ”MIB/CEN”, ”SN”, ”2”, ”etCO2\*”, con unità di misura ”mm(hg)”]
- Saturazione [”2710-2”, ”LOINC”, ”SN”, ”1”, ”SpO2”, con unità di misura ”%”]
- Temperatura [”8332-9”, ”LOINC”, ”SN”, ”4”, ”Ta”, con unità di misura ”cel”]

Come ultima nota, in figura si può notare una voce aggiuntiva, ovvero ”value”. Serve solo ad indicare una stringa a piacere da associare al parametro.

## Server

E' necessario innanzitutto prevedere un sistema di simulazione. Devono venire simulati dei pazienti, i quali saranno caratterizzati da identificatore, dati anagrafici e parametri vitali. I parametri vitali dovranno essere aggiornati periodicamente. Inoltre servirà ovviamente modellare il funzionamento del



Figura 3.7: Modello parametri vitali

server (inizializzazione, ricezione connessioni e ricezione richieste da parte dei client) con un meccanismo per preparare il messaggio HL7 di risposta da inviare al client. Di seguito vengono illustrate brevemente le entità che compongono il server:

- *IPatient*: entità passiva che modella i dati dei pazienti
- *IPatientSimulator*: entità attiva che modella la simulazione, aggiornando periodicamente i dati dei pazienti
- *IServer*: entità attiva che modella il funzionamento di base del server
- *IServerWrapper*: entità attiva che incapsula IServer modellando aspetti di più alto livello
- *IGUIListener*: entità passiva che modella il listener della GUI (se si decide di usarla)
- *QRYHandler*: entità passiva che modella la creazione del messaggio di risposta per il client sulla base della richiesta ricevuta

- *ISimulatorData*: entità passiva che modella i dati relativi all simulazione.  
E' possibile utilizzare i dati di default o personalizzarli
  - Pazienti: numero dei pazienti, unità di reparto, prefisso del lettino (in modo da potere derivare il codice del lettino in base al numero di pazienti scelto)
  - Parametri vitali: valori massimi e minimi per tutti i parametri vitali di interesse
  - Intervallo di aggiornamento dei parametri vitali dei pazienti simulati
  - Latenze: range di valori di latenza dopo l'apertura della connessione e latenza invio messaggi di risposta (utile per simulare contesti reali)

In figura 3.9 si può vedere come interagiscono i componenti del Server.

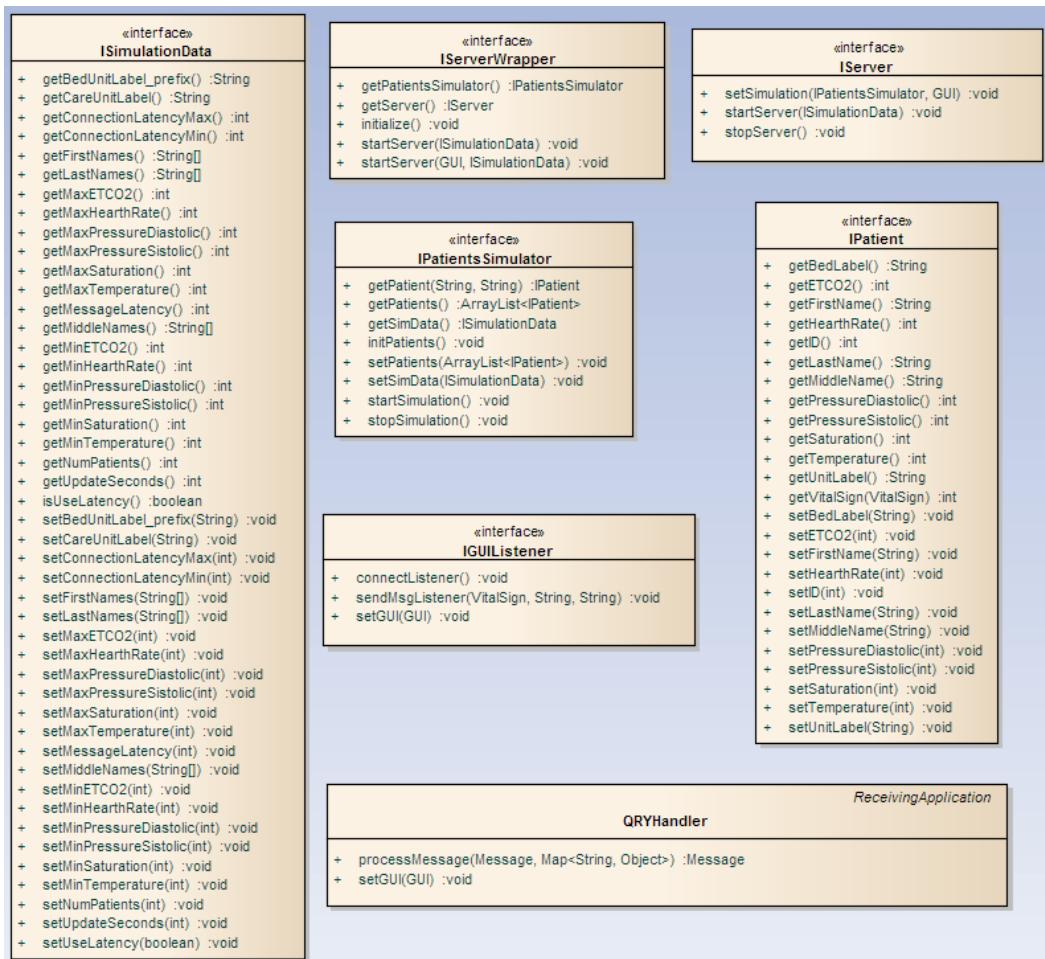


Figura 3.8: Modello server

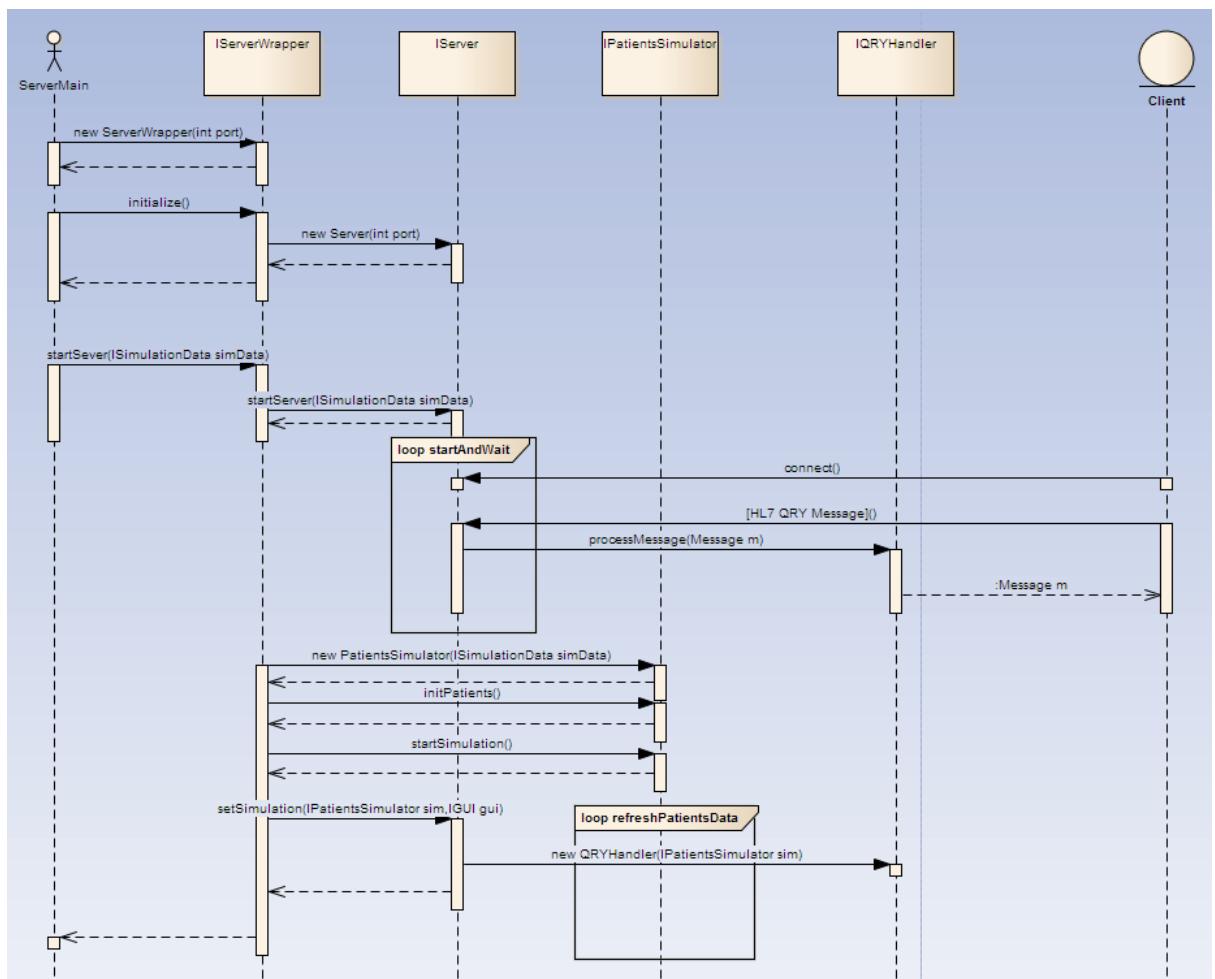


Figura 3.9: Interazione componenti server

## Client

E' necessario modellare ovviamente anche il funzionamento del client (inizializzazione, connessione al server, invio messaggi al server..) con un meccanismo per preparare il messaggio HL7 da inviare al server come richiesta e un meccanismo per processare la risposta ricevuta. E' necessario inoltre prevedere dei meccanismi per controllare se la connessione è andata a buon fine e se il messaggio ricevuto è valido e non è di errore.

- *IClient*: entità attiva che modella il funzionamento di base del client
- *IClientWrapper*: entità attiva che incapsula *IClient* modellando aspetti di più alto livello (in particolare le richieste dei dati vitali al server)
- *IGUIListener*: entità passiva che modella il listener della GUI (se si decide di usarla)
- *IQRYCreator*: entità passiva che si occupa di preparare il messaggio HL7 di richiesta al server
- *IMessageProcessor*: entità passiva che si occupa di processare la risposta ottenuta dal server e di stamparla a video o su GUI
- *IConnectionStatus*: entità passiva che modella lo stato della connessione al server
- *IRequestResult*: entità passiva che modella la risposta ricevuta dal server, contenendo informazioni sull'errore nel caso in cui ne siano riscontrati

In figura 3.11 si può vedere come interagiscono i componenti del Client.

## Gestione errori

Affinchè il client sia robusto è necessario prevedere le varie casistiche di errore e gestirle opportunamente. Sono stati individuati i seguenti scenari:

- Connessione al server
  - Il client non riesce a connettersi al server (ad esempio se il server non è raggiungibile)
- Richiesta parametri vitali al server
  - Il client invia una richiesta al server ma esso non è più raggiungibile
  - Il client invia una richiesta al server e non riceve risposta entro un timeout prefissato

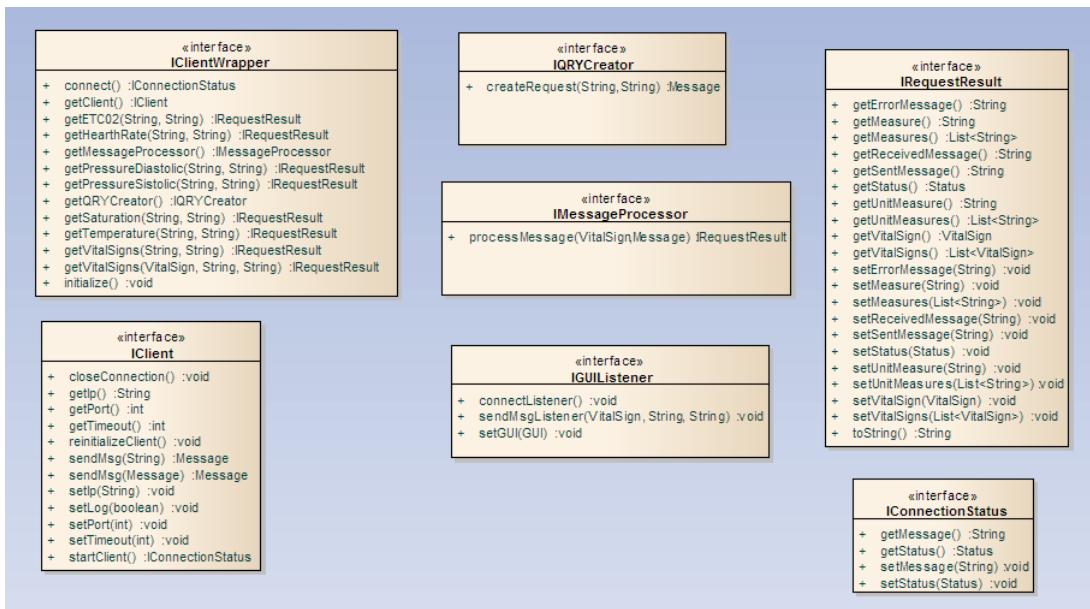


Figura 3.10: Modello client

- Il client invia una richiesta al server, ma la risposta è un messaggio di tipo ACK invece che di tipo ORF (solitamente questo indica che c'è stato qualche problema di tipologia Application Error o Application Reject)
- Il client invia una richiesta al server, e questo risponde con un messaggio di tipo ORF ma contenente un errore (di tipologia Application Error o Application Reject)
- Il client invia una richiesta al server e questo risponde comunicando che non è stato individuato il letto (in questo caso il gateway non contiene i dati del monitor richiesto)

Tutti questi casi sono stati gestiti, ed è possibile quindi accorgersi di queste tipologie di errori lato applicativo e quindi specificare il comportamento da seguire. Si veda la figura 3.12 per il modello degli errori.

## Interazione

Di seguito verrà illustrato il modello d'interazione tra client e server. Si noti che nella figura 3.13 i nomi non rispecchiano i nomi usati poi nell'implementazione. Il tutto vuole solo spiegare logicamente il funzionamento del sistema. Lato server, viene fatta partire la simulazione, inizializzato e fatto partire il server stesso. Ogni tre secondi circa vengono rinfrescati i dati relativi

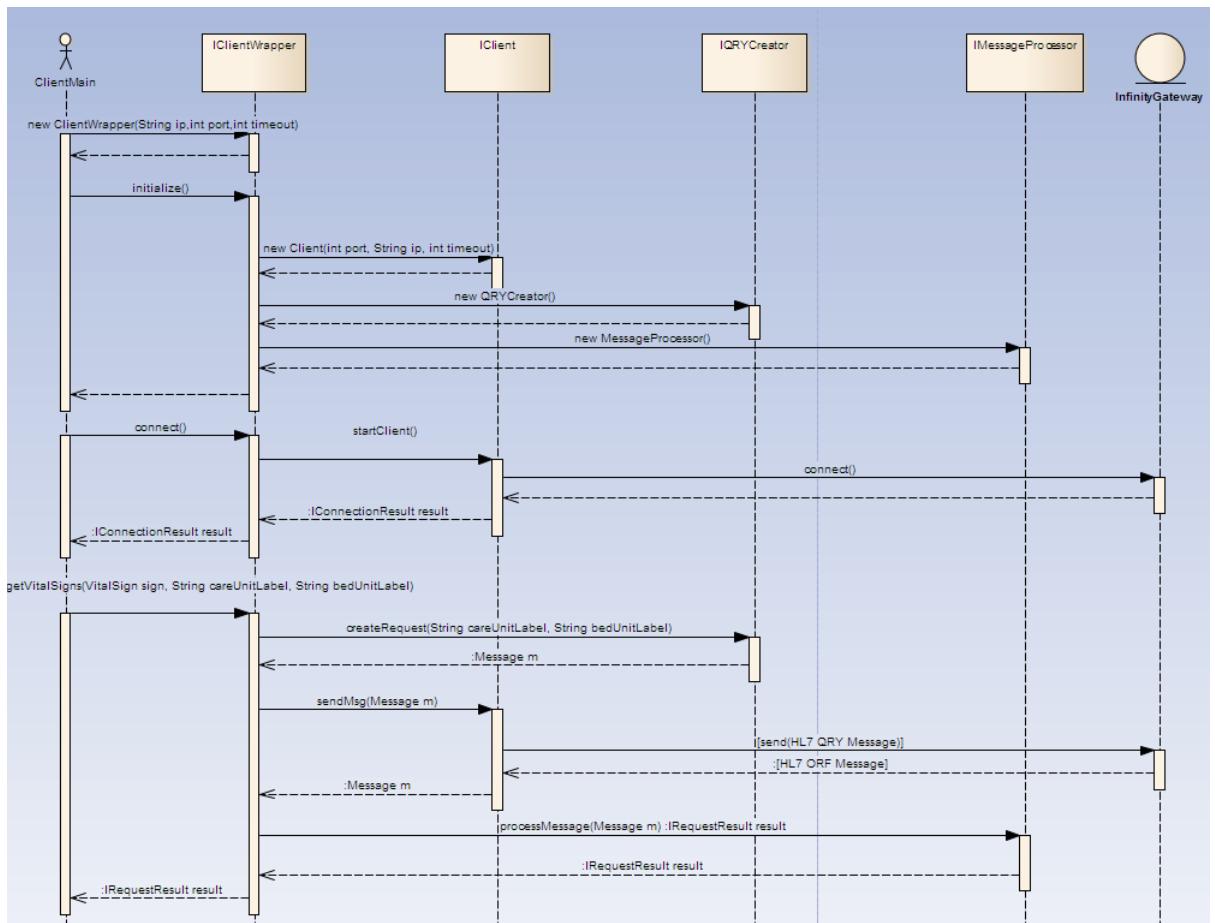


Figura 3.11: Interazione componenti client

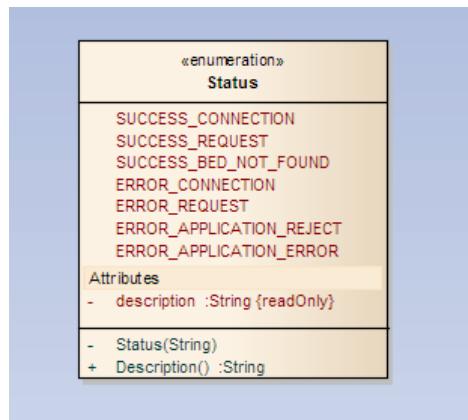


Figura 3.12: Modello errori

ai pazienti (il refresh rate è configurabile). Il server quindi entra in una fase in cui accetta connessioni da parte dei client e attende dei messaggi. Una volta ricevuta una richiesta, questa viene processata e si fornisce la risposta al client. Il client invece si inizializza e si prepara a fare delle richieste al server. Per prima cosa deve identificare il paziente collegato al monitor (se non conosce i valori del codice reparto e codice monitor a priori), quindi inviare la richiesta dei dati vitali. Una volta ricevuta la risposta dal server, sulla base del parametro vitale richiesto, il messaggio viene processato e viene estratto il valore relativo appunto al parametro richiesto.

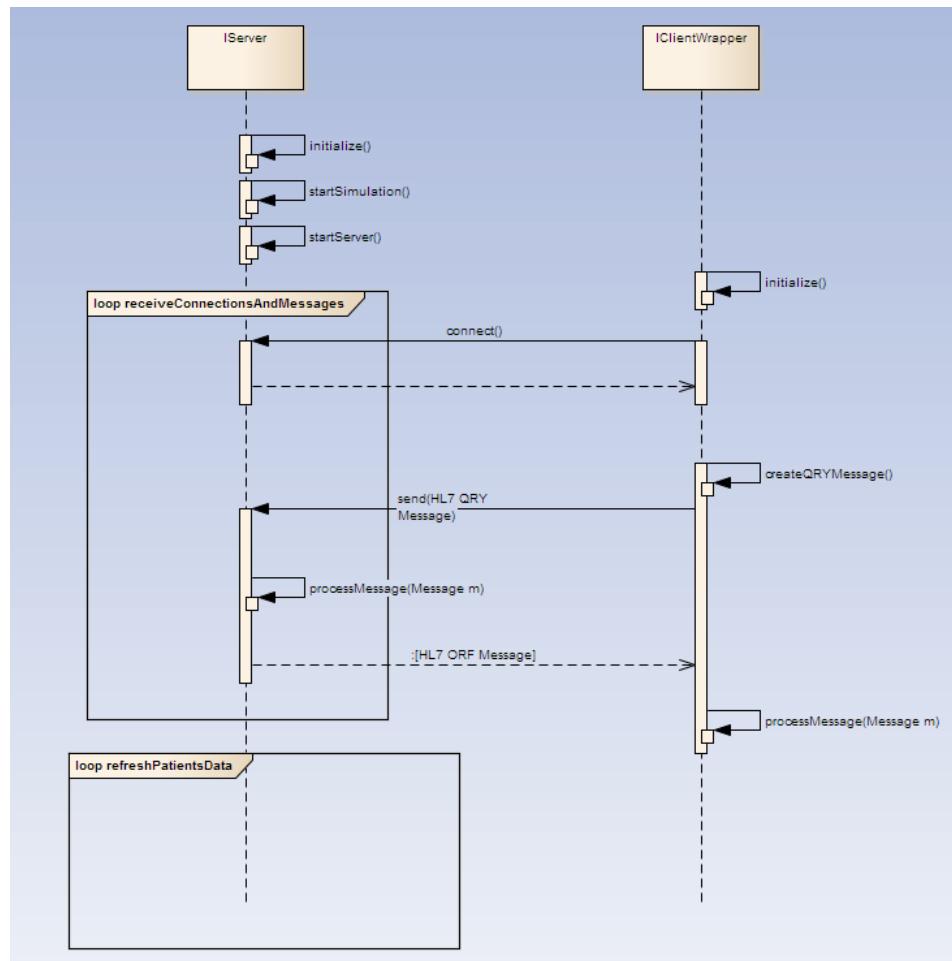


Figura 3.13: Modello interazione

## Deploy e uso

Il sistema è stato rilasciato sotto forma di libreria Jar. Per usare la libreria, basta importarla nel progetto in cui si vuole appunto utilizzarla. Di seguito si mostra un semplice esempio di codice per utilizzarla in ambiente Java.

- Client: esempio richiesta hearth rate

```
IClientWrapper init = new ClientWrapper("localhost", 2550,2000);
init.initialize();
IConnectionStatus connRes = init.connect();
if (connRes.getStatus().equals(Status.SUCCESS_CONNECTION) ==
    true) {
    // connessione avvenuta: richiesta dati al server
    IRequestResult result =
        init.getVitalSigns(VitalSign.HEARTH_RATE,"TI","SALA1");
    if (result.getStatus().equals(Status.SUCCESS_REQUEST)) {
        // la richiesta parametri vitali ha successo
        System.out.println(result.toString());
        // per stampare la misura o la unitá di misura
        result.getMeasure();
        result.getUnitMeasure();
    } else {
        // errore nella richiesta
        System.out.println(result.getErrorMessage() + "\n");
        // gestione errore
        requestHandler(result);
    }
} else {
    // errore nella connessione al server
    System.out.println(connRes.getMessage());
    //gestione errore connessione
    connectionHandler(connRes);
}
```

- Server

```
ServerWrapper init=new ServerWrapper(2550);
init.initialize();
init.startServer(10,"TI","SALA");
```

Si noti che per usare il client su Android sono necessarie alcune accortezze:

- Inserire i permessi di rete nell'Android Manifest

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
    android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

- Eseguire il client in un AsyncTask (poichè non è possibile eseguire operazioni di networking nel thread principale dedicato alla UI)

Mentre si consiglia di utilizzare il server in ambiente Java desktop.

### 3.1.3 Server reale

Per fare test con il software reale e non simulato, ci è stato fornito:

- Il software ufficiale dell'Infinity Gateway (con anche un simulatore)
- Un monitor, ovvero il dispositivo che legge i dati vitali del paziente
- Dongle USB contenente la licenza d'uso

Il software è composto da diverse parti, ma di seguito verranno introdotte solo quelle rilevanti per il progetto. Il software ha due componenti d'interesse principali:

- *Software gateway*: è il software del gateway vero e proprio che viene usato all'interno dell'ospedale. Affinchè funzioni è necessario che sia attiva la licenza
- *Simulatore*: software che appunto funge da simulatore. È possibile simulare:
  - Gateway: è possibile simulare il funzionamento (in modo parziale) di un gateway. In questo modalità si può contattare il simulatore con un sistema esterno e farsi inviare un messaggio HL7 prefissato che è possibile scegliere
  - Sistema esterno: è possibile simulare il funzionamento (in modo parziale) di un sistema esterno. In questo modo si può contattare il gateway con un messaggio definibile e leggere i messaggi di risposta

Per il funzionamento del simulatore non è richiesta la licenza.

Il software è stato installato su macchina virtuale, e i passaggi effettuati per il setup del sistema sono stati i seguenti:

- Installazione software per la virtualizzazione (VMWare in questo caso)

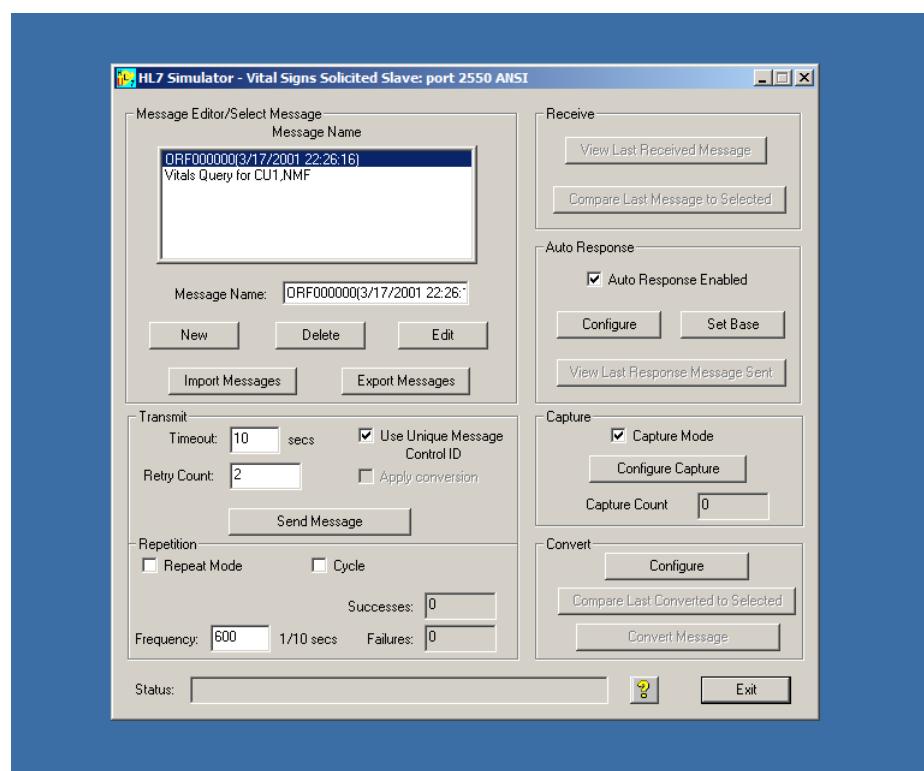


Figura 3.14: Simulatore gateway

- Configurazione network bridge per fare in modo che la macchina virtuale possa connettersi alla rete dell'host ottenendo un nuovo indirizzo ip
- Copia di una macchina virtuale (con sistema operativo Windows Server 2008) preconfigurata con il software installato e le varie impostazioni del sistema operativo già configurate (in seguito a diversi problemi nell'installazione del software derivanti dalla necessità di dover effettuare numerose configurazioni e modifiche di basso livello nel sistema operativo)
- Disattivazione del firewall sulla macchina virtuale affinchè essa fosse contattabile da macchine esterne

Infine è stato necessario creare una rete locale in cui inserire il monitor e la macchina contenente la macchina virtuale con il software del gateway, oltre ad eventuali altre macchine con installati dei sistemi esterni che devono interagire con il gateway. A questo scopo è stato usato uno switch di rete. Se si necessita dell'accesso ad Internet è possibile usare un router. Una volta che i dispositivi sono nella rete, tutto è pronto per fare le dovute prove. Il monitor può funzionare in due modalità:

- Modalità "reale": il monitor funziona normalmente, quindi legge i dati vitali provenienti dai sensori collegati al paziente
- Modalità simulata: il monitor setta dei parametri vitali casuali (molto utile per fare le prove)

In entrambi i casi, per come è configurato, il monitor invia in multicast messaggi HL7 ad intervalli regolari contenenti appunto i parametri vitali. Dopo alcuni raffinamenti del software sviluppato per leggere correttamente i dati che il gateway fornisce, si è accurato il corretto funzionamento del tutto.

### 3.1.4 Test del sistema

Di seguito verranno illustrati i passaggi per effettuare il test:

- Setup della rete per effettuare i test. Ci si deve quindi munire di uno switch di rete o di un router e collegarci via ethernet il monitor, la macchina virtuale contenente il software del gateway e la macchina su cui è installato l'applicativo che dovrà contattare il gateway (in questo caso la macchina virtuale è installata sulla stessa macchina su cui è installato l'applicativo). Si può vedere una fotografia della rete in figura 3.15.

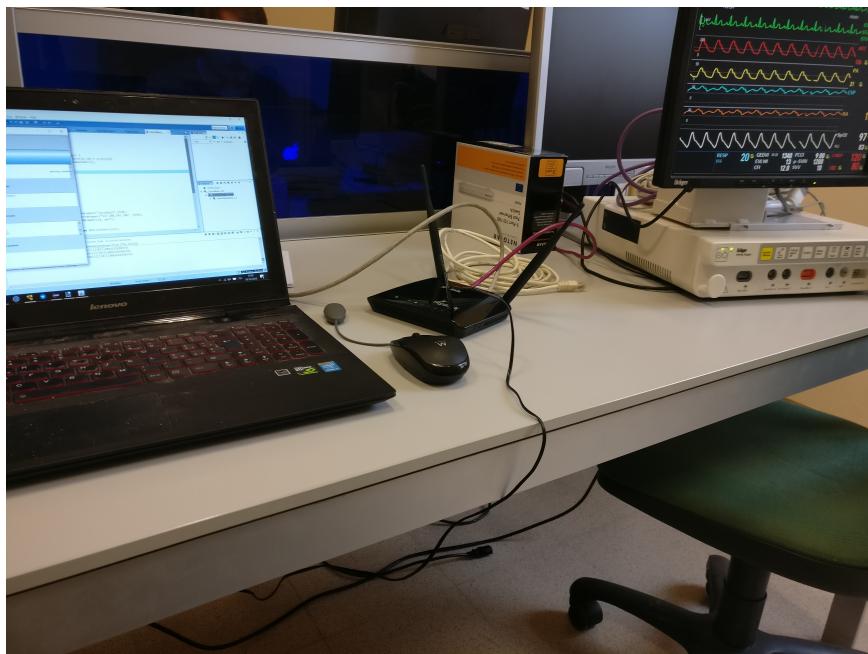


Figura 3.15: Rete locale di test

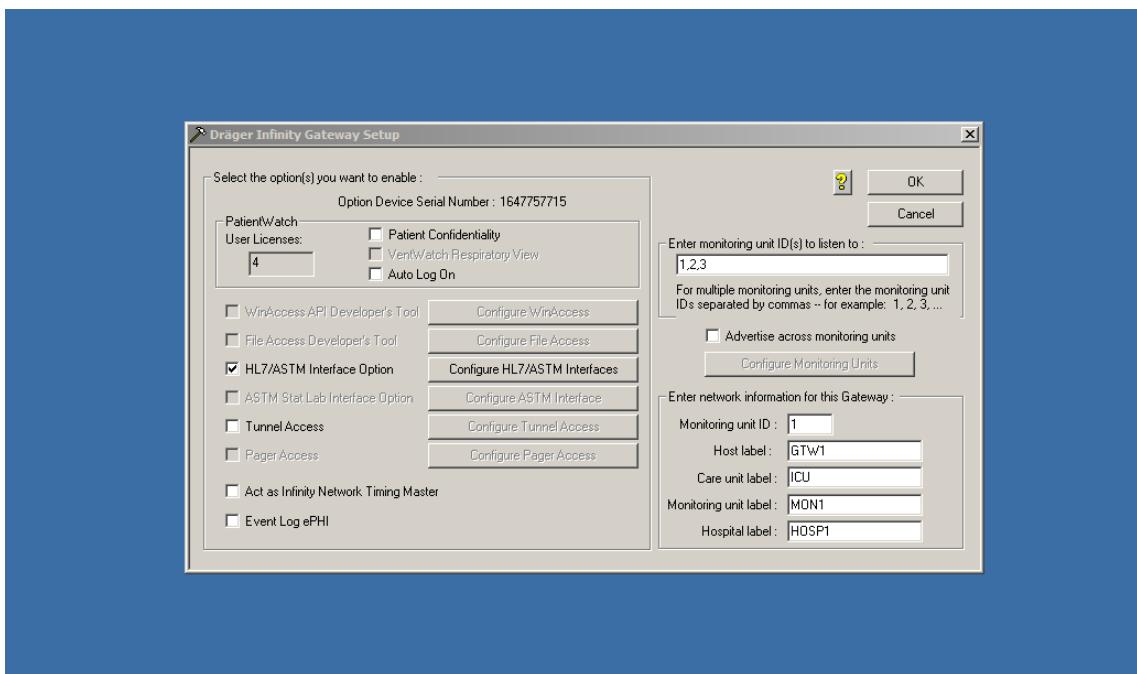


Figura 3.16: Schermata di configurazione gateway

- Ora è necessario lanciare il software del gateway (con dongle inserito dato che è necessaria la licenza). Si può vedere la schermata di configurazione iniziale in figura 3.16.
- Impostazione del monitor in modalità simulazione. Si può vedere la schermata dei parametri vitali in figura 3.17. Si noti che la schermata è identica a quella che ci sarebbe se un vero paziente fosse collegato al monitor grazie ai sensori



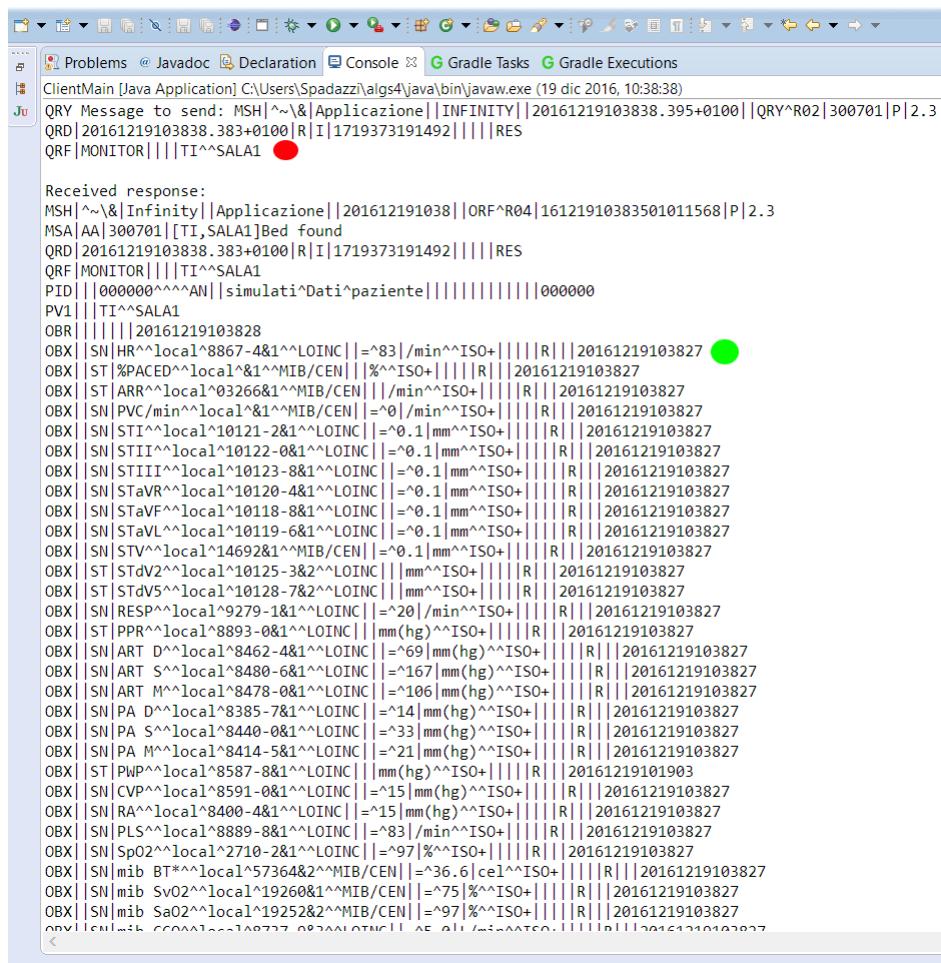
Figura 3.17: Monitor in modalità simulazione

- Il monitor quindi invierà messaggi HL7 in multicast e il gateway salverà i parametri vitali del paziente
- Ora è necessario lanciare il client HL7 sviluppato per poter inviare una query al gateway. Nella figura 3.18 si può vedere il messaggio di query inviato dall'applicativo al gateway insieme a parte del lungo messaggio di risposta ricevuto. Si può notare in figura (precisamente vicino al pallino rosso) che nella parte finale del messaggio di query sono stati indicati i seguenti valori:
  - TI: codice del reparto (Terapia Intensiva in questo caso) del monitor
  - SALA1: codice del monitor stesso

I valori servono per identificare il monitor, in modo tale che il gateway sappia che dati deve fornire. Vicino al pallino verde si può invece vedere ad esempio il valore per il parametro relativo al battito cardiaco (83 battiti al minuto), valore che poi viene correttamente estratto dal messaggio come verrà ora illustrato. Nella figura 3.19 infatti si può vedere la parte finale del messaggio ricevuto e i parametri vitali rilevanti estratti da esso. Vicino al pallino verde si può vedere come il valore relativo al battito cardiaco sia stato estratto correttamente. Si può notare la giusta corrispondenza anche nella figura 3.17 guardando la prima riga di parametri vitali (in valore è in alto a destra). Si possono anche vedere ad esempio i parametri relativi alla pressione diastolica e sistolica nella terza riga (quella rossa) che corrispondono rispettivamente a 69 e 167, come riportato nella stampa finale. Se i test si vogliono effettuare con il simulatore, basta lanciare quello al posto del gateway e configurare un messaggio da farsi inviare come risposta. Il software sviluppato è stato testato e si è appurato che i requisiti funzionali sono stati soddisfatti in quanto è possibile recuperare i parametri vitali richiesti dai medici. Per quanto riguarda considerazioni aggiuntive e problematiche riscontrate, si può consultare il paragrafo successivo. Il monitor è stato sempre lasciato in modalità simulata ma si è anche testato il funzionamento reale con la misura di SpO<sub>2</sub>. E' infatti molto semplice effettuare questo test in quanto il sensore è una semplice presa in cui inserire un dito. E' quindi possibile collegarsi al monitor tramite questo sensore senza la necessità della modalità simulata, e in questo caso notare che il gateway viene aggiornato con il valore corrente e reale del parametro (figura 3.20).

Di seguito vengono illustrati i test effettuati. I test sono stati effettuati all'interno della rete locale (la macchina che interroga al gateway si trova all'interno della stessa rete).

- Test recupero parametri vitali: si è testato che i parametri vitali venissero effettivamente recuperati in modo corretto
- E' stato misurato il tempo di apertura della connessione. La connessione viene aperta in tempi molto brevi (di media in circa 15ms)
- E' stato misurato il tempo di elaborazione di una richiesta al gateway. Il tutto si compone di:
  - Invio del messaggio al gateway
  - Elaborazione del messaggio da parte del gateway e invio risposta (parte che occupa quasi la totalità del tempo)



The screenshot shows a Java application running in an IDE. The console tab displays the following message:

```

ClientMain [Java Application] C:\Users\Spadazzi\algs4\java\bin\javaw.exe (19 dic 2016, 10:38:38)
QRY Message to send: MSH|^~\&|Applicazione||INFINITY||20161219103838.395+0100||QRY^R02|300701|P|2.3
QRD|20161219103838.383+0100|R|I|1719373191492||||RES
QRF|MONITOR||||TI^^SALA1
```

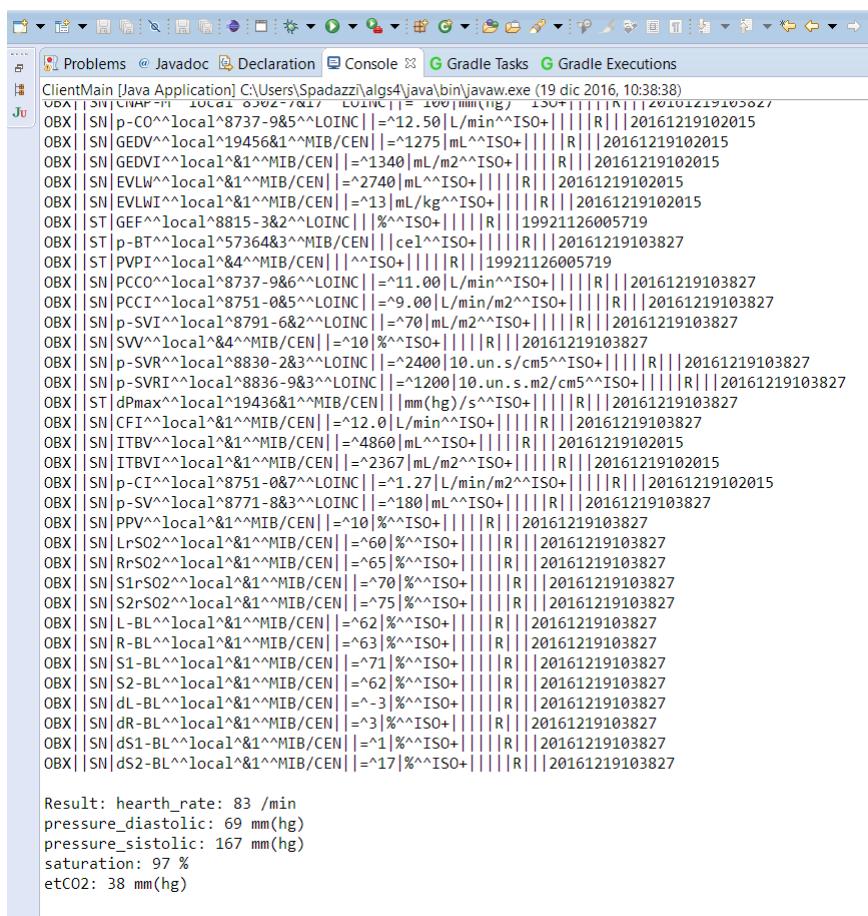
Received response:

```

MSH|^~\&|Infinity||Applicazione||201612191038||ORF^R04|16121910383501011568|P|2.3
MSA|AA|300701|[TI,SALA1]Bed found
QRD|20161219103838.383+0100|R|I|1719373191492||||RES
QRF|MONITOR||||TI^^SALA1
PID|||000000^^AN||simulati^Dati^paziente|||||||||000000
PV1||||TI^^SALA1
OBR|||||20161219103828
OBX||SN|HR^^local^8867-4&1^^LOINC||=^83|/min^^ISO+|||||R|||20161219103827
OBX||ST|%SPACED^^local^&1^^MIB/CEN|||^%^^ISO+|||||R|||20161219103827
OBX||ST|ARR^^local^032668&1^^MIB/CEN|||^min^^ISO+|||||R|||20161219103827
OBX||SN|PVC/min^^local^&1^^MIB/CEN|||=^0|/min^^ISO+|||||R|||20161219103827
OBX||SN|STI^^local^10121-2&1^^LOINC|||=^0.1|mm^^ISO+|||||R|||20161219103827
OBX||SN|STII^^local^10122-0&1^^LOINC|||=^0.1|mm^^ISO+|||||R|||20161219103827
OBX||SN|STIII^^local^10123-8&1^^LOINC|||=^0.1|mm^^ISO+|||||R|||20161219103827
OBX||SN|StaVR^^local^10120-4&1^^LOINC|||=^0.1|mm^^ISO+|||||R|||20161219103827
OBX||SN|StaVF^^local^10118-8&1^^LOINC|||=^0.1|mm^^ISO+|||||R|||20161219103827
OBX||SN|StaVL^^local^10119-6&1^^LOINC|||=^0.1|mm^^ISO+|||||R|||20161219103827
OBX||SN|STV^^local^1469281^^MIB/CEN|||=0.1|mm^^ISO+|||||R|||20161219103827
OBX||ST|STDv2^^local^10125-3&2^^LOINC|||mm^^ISO+|||||R|||20161219103827
OBX||ST|STDv5^^local^10128-7&2^^LOINC|||mm^^ISO+|||||R|||20161219103827
OBX||SN|RESP^^local^9279-1&1^^LOINC|||=^20|/min^^ISO+|||||R|||20161219103827
OBX||ST|PPR^^local^8893-0&1^^LOINC|||mm(hg)^^^ISO+|||||R|||20161219103827
OBX||SN|ART D^^local^8462-4&1^^LOINC|||=^69|mm(hg)^^^ISO+|||||R|||20161219103827
OBX||SN|ART S^^local^8480-6&1^^LOINC|||=^167|mm(hg)^^^ISO+|||||R|||20161219103827
OBX||SN|ART M^^local^8478-0&1^^LOINC|||=^106|mm(hg)^^^ISO+|||||R|||20161219103827
OBX||SN|PA D^^local^8385-7&1^^LOINC|||=^14|mm(hg)^^^ISO+|||||R|||20161219103827
OBX||SN|PA S^^local^8440-0&1^^LOINC|||=^33|mm(hg)^^^ISO+|||||R|||20161219103827
OBX||SN|PA M^^local^8414-5&1^^LOINC|||=^21|mm(hg)^^^ISO+|||||R|||20161219103827
OBX||ST|PWP^^local^8587-8&1^^LOINC|||mm(hg)^^^ISO+|||||R|||20161219101903
OBX||SN|CVP^^local^8591-0&1^^LOINC|||=^15|mm(hg)^^^ISO+|||||R|||20161219103827
OBX||SN|RA^^local^8400-4&1^^LOINC|||=^15|mm(hg)^^^ISO+|||||R|||20161219103827
OBX||SN|PLS^^local^8889-8&1^^LOINC|||=^83|/min^^ISO+|||||R|||20161219103827
OBX||SN|Sp02^^local^2710-2&1^^LOINC|||=^97|%^^^ISO+|||||R|||20161219103827
OBX||SN|mib BT*^^local^57364&2^^MIB/CEN|||=^36.6|ce1^^ISO+|||||R|||20161219103827
OBX||SN|mib Sv02^^local^19260&1^^MIB/CEN|||=^75|%^^^ISO+|||||R|||20161219103827
OBX||SN|mib Sa02^^local^19252&2^^MIB/CEN|||=^97|%^^^ISO+|||||R|||20161219103827

```

Figura 3.18: Query inviata e parte del messaggio di risposta ricevuto



The screenshot shows an IDE interface with several tabs at the top: Problems, Javadoc, Declaration, Console, Gradle Tasks, and Gradle Executions. The main area displays a large block of text representing a response message. The text is a series of lines starting with 'OBX' followed by various parameters and their values. At the bottom of this list, there is a 'Result:' section containing extracted parameters:

```

Result: hearth_rate: 83 /min
pressure_diastolic: 69 mm(hg)
pressure_sistolic: 167 mm(hg)
saturation: 97 %
etCO2: 38 mm(hg)

```

Figura 3.19: Parte finale del messaggio di risposta e estrazione parametri da esso

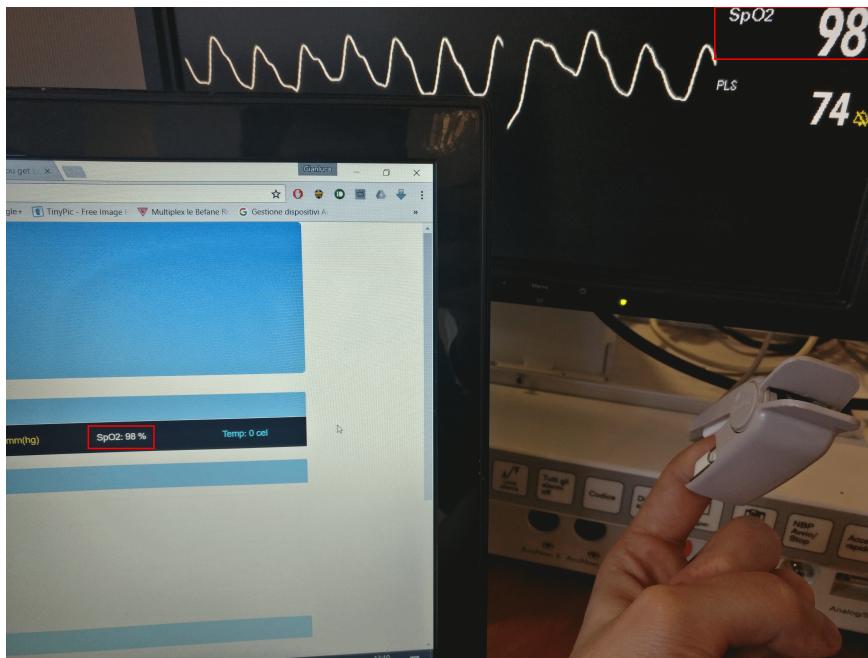


Figura 3.20: Test SpO2

- Ricezione del messaggio e estrazione dei parametri da esso

Lo scopo è quello di capire le tempistiche di funzionamento del gateway per gestire il carico quando sarà costruito il servizio. Il test è stato così strutturato:

- Invio di 100 messaggi separati da un secondo ciascuno
- Ripetizione del test per tre volte

Sono stati riscontrati i seguenti risultati:

- Il tempo medio per un messaggio è stato rispettivamente per i tre test di 187ms, 192ms e 191ms, con una media totale di 190ms.
- Quasi nella totalità dei casi i messaggi vengono processati in un tempo che oscilla tra 180ms-210ms
- Possono capitare rare volte dei valori più alti (tra 450ms-550ms)
- Più di frequente rispetto ai picchi alti, può capitare un tempo medio più basso di circa 100ms

I test sono stati quindi ripetuti anche con la macchina che interroga il gateway fuori dalla rete locale (connettendosi ad essa tramite VPN da

remoto) e i risultati sono stati del tutto simili, confermando che le latenze riguardano effettivamente il tempo di elaborazione dei dati del software di Draeger. E' da concludere quindi che il gateway di Draeger è in grado di processare mediamente 5 richieste al secondo. Tuttavia va tenuto in considerazione il problema del troncamento dei messaggi, che impedisce che richieste relative allo stesso monitor vengano effettuate a distanza inferiore di 800ms. I test con i sistemi in ospedale mostreranno se questo problema si verifica anche lì, ma per il momento è necessario considerare il problema e porvi rimedio

- E' stato misurato il tempo di invio/ricezione di un messaggio al gateway per capire quanto esso impiega a processare una richiesta nel caso in cui la connessione venga aperta immediatamente prima dell'invio del messaggio. In questo caso si nota che è presente un delay temporale piuttosto elevato dopo l'apertura della connessione prima che il messaggio venga processato dal gateway. Nel caso precedente si è assunto che la connessione all'invio del messaggio fosse stata già aperta tempo prima rispetto all'invio del messaggio (nel caso del test 3 secondi prima). In questo caso si misurano le tempistiche aprendo la connessione e subito dopo inviando il messaggio. Il test viene descritto meglio in 3.1.4
- Test troncamento messaggio (si veda 3.1.4 per maggiori dettagli sul problema): si è testato l'intervallo di tempo minimo tra un messaggio e l'altro per non fare accadere la problematica, ovvero 800ms. Si è notato che se il gateway riceve messaggi di richiesta relativi ad un monitor ad intervalli inferiori di 800ms tra l'uno e l'altro, potrebbe verificarsi in modo non deterministico il troncamento del messaggio. E' stato eseguito un test per la durata di 5 ore in cui ogni 800ms è stato inviato un messaggio al gateway ed il messaggio di risposta è stato sempre completo

Una volta che sono stati misurati i tempi di risposta nelle varie casistiche, è possibile raffinare il simulatore rendendo la simulazione più realistica. Questo è necessario poichè ci è stato fornito un solo monitor con una sola licenza d'uso, perciò è possibile testare il sistema reale solo con un monitor collegato al gateway. Se si vuole testare uno scenario in cui il sistema lavora con più monitor collegati inserendo con i tempi di latenza riscontrati nei test. Sono quindi stati inseriti i seguenti delay:

- Il server avrà una latenza compresa tra 1 secondo e 1.5 secondi (configurabile) non appena apre una nuova connessione
- Il server avrà una latenza di 190ms non appena riceve un messaggio

Oltre ai test relativi alle funzionalità e alla misurazione delle latenze, sono state testate anche le casistiche di errore descritte in ??:

- Test connessione al server non riuscita: è sufficiente non lanciare il server reale o simulato, lanciare il client e tentare comunque la connessione. E' quindi possibile verificare il fatto che il sistema si accorga che è impossibile connettersi e che quindi restituisca errore
- Test server non raggiungibile: è possibile che si verifichi che il server non sia più raggiungibile quando la connessione è già stata effettuata in precedenza. E' necessario:
  - Lanciare il server (reale o simulato)
  - Lanciare il client e connettersi al server
  - Chiudere il server
  - Tentare il recupero dei parametri vitali

Si può quindi notare che alla scadenza del timeout impostato nel client, viene restituito errore

- Ricezione di un Application Error/Reject: è possibile testare il fatto che il client riconosca la ricezione di messaggi di tipo ORF contenenti errori di tipo application reject richiedendo al server i dati relativi a un paziente non memorizzato nel sistema (si veda il punto successivo). Può succedere anche il caso in cui ci sia un errore interno del server (ad esempio un'eccezione non gestita). In questo caso non viene inviata come risposta un messaggio di tipo ORF, bensì di tipo ACK. Non potendo provocare un errore interno nel sistema di Draeger, è necessario usare il simulatore. Inserendo durante l'elaborazione della risposta un blocco di codice che provochi un'eccezione, è possibile verificare che il client si rende conto della ricezione di un messaggio di tipo ACK che contiene un errore. La libreria HAPI gestisce automaticamente l'invio da parte del server di un messaggio di tipo ACK nel caso in cui si verifichi un errore interno
- Lettino non trovato: si può testare il fatto che il client rilevi che il messaggio di risposta ricevuto indichi che il lettino non è stato trovato con i seguenti passaggi:
  - Lancio del server (simulato o reale)
  - Lancio del client e richiesta di parametri vitali specificando una unità di reparto o codice di lettino o entrambi errati

**Problematiche software Gateway** Sono state riscontrate alcune problematiche relative al software di Draeger.

- *Troncamento messaggi:* se il gateway riceve più di una richiesta di parametri vitali relativamente ad un dato monitor in un ristretto intervallo di tempo, in modo non deterministico è possibile notare che i messaggi HL7 ricevuti risultano troncati (ovvero non tutti i parametri vitali vengono riportati nel messaggio). Se i messaggi sono distanziati da un certo intervallo di tempo i messaggi risultano tutti uguali (in quanto il monitor in modalità simulazione mantiene sempre uguali i parametri vitali). Il problema potrebbe essere imputato a diversi fattori:
  - Il software è predisposto all'esportazione dei dati in formato HL7 in maniera sporadica e non si presta ad un uso pesante
  - Il monitor fornito ci potrebbe avere dei problemi interni e fornire messaggi errati ogni determinati lassi di tempo
  - Il software di Draeger potrebbe avere effettivamente dei bug

Anche se il problema potrebbe non essere presente nelle apparecchiature e software presenti nell'ospedale, fino a che non sarà possibile effettuare sperimentazioni al suo interno è necessario tenere in considerazione il problema e fare in modo che non si verifichi. Sono stati effettuati dei test in questa modalità:

- Apertura connessione
- Invio continuo di messaggi al gateway ogni determinati intervalli di tempo sempre crescenti

Sono state fatte le seguenti osservazioni:

- Se l'intervallo di tempo è inferiore a 800 ms, il problema si verifica (e più i messaggi sono ravvicinati più di frequente accade)
- Se l'intervallo di tempo è maggiore o uguale a 800 ms, il problema non si verifica mai (il test è stato eseguito per circa 5 ore e non si è mai verificato il problema)
- Se per inviare ogni messaggio viene chiusa e riaperta la connessione, il problema non si verifica mai in quanto a causa del delay presente dopo l'apertura della stessa (problema descritto nel punto successivo) il gateway elabora i messaggi con un intervallo di tempo sempre superiore a 800ms

- *Alte latenze:* sono state riscontrate latenze molto alte nel caso in cui la connessione venga aperta e chiusa molto spesso. In particolare:
  - Non appena la connessione viene aperta e un messaggio inviato immediatamente dopo, esso viene processato in media in 1-1.5 secondi al posto dei 190ms riscontrati nei test. Pare quindi che dopo l'apertura della connessione serva del tempo al gateway per processare il primo messaggio ricevuto. Quelli successivi vengono invece processati in tempi regolari
  - Se la connessione viene chiusa, immediatamente riaperta e immediatamente inviato un nuovo messaggio, la latenza cresce drasticamente, arrivando a toccare tempi inaccettabili di 3 secondi per processare una richiesta. Inserendo un tempo di attesa tra la chiusura della connessione e la sua riapertura si può notare che il tempo di processing diminuisce. Per riassestarsi sul tempo regolare di 1-1.5 secondi è necessario attendere un tempo di 2.5 secondi

A fronte di questa problematica sarà opportuno fare in modo di aprire e chiudere la connessione con il gateway il meno possibile, e di lasciarla sempre aperta in modo tale da evitare latenze che comprometterebbero la reattività del sistema.

- *Temperatura:* il monitor fornитоci non invia nel messaggio la misura della temperatura, che è richiesta dai medici. Sarà necessario verificare con le sperimentazioni sul campo se:
  - Il valore non è presente solo nel monitor fornитоci in quanto lavora in modalità simulata e invia solo un sottoinsieme dei parametri
  - E' stato usato il codice errato della temperatura seguendo il manuale dell'Infinity Gateway. In tal caso è sufficiente individuare il parametro corretto con l'aiuto di qualcuno con competente in campo medico e sostituirlo nell'enumerato contenente i parametri vitali all'interno del software

Concludendo, saranno sicuramente necessarie sperimentazioni più approfondite quando il sistema sarà testato all'ospedale in modo tale da capire innanzitutto se i problemi riscontrati si verificano e anche di verificare che i tempi di latenza sotto la rete ospedaliera siano simili a quelli riscontrati nella rete di test.

## 3.2 GatewayService

GatewayService è il microservizio che si occupa di fornire alle applicazioni interessate i dati vitali dei pazienti collegati ai monitor. E' realizzato in Vert.x (A.4) con linguaggio Java e sfrutta l'architettura REST (A.2) con linguaggio Json (A.3) per lo scambio dei dati. Inoltre incapsula al suo interno HL7Subsystem per interfacciarsi con il gateway di Draeger. In questo modo le applicazioni possono richiedere i dati al gateway con molta semplicità tramite un'interfaccia standardizzata all'interno del sistema, eliminando così la complessità legata all'operazione.

### 3.2.1 Requisiti

I requisiti dell'applicativo sono i seguenti:

- Prevedere delle API per il recupero dei parametri vitali associati a un monitor. In particolare:
  - Recupero di tutti i parametri vitali rilevanti
  - Recupero di un singolo parametro vitale
- Prevedere una modalità di tracciamento periodico dei parametri vitali (i parametri sono gli stessi di HL7SubSystem):
  - Deve essere possibile specificare una frequenza di recupero dei parametri vitali. Il servizio deve quindi raccogliere i dati e fornirli all'applicazione che ha richiesto questa modalità non appena lo richiede. Questo requisito deriva dall'esigenza di TraumaTracker di tracciare i parametri vitali con una frequenza variabile a seconda del luogo in cui l'operatore si trova. La frequenza quindi potrebbe cambiare anche durante l'intervento stesso se l'operatore si sposta
- Prevedere delle API per la manipolazione dei monitor salvati nel sistema, in particolare per:
  - Aggiunta di un nuovo monitor
  - Modifica di un monitor
  - Eliminazione di un monitor
  - Recupero di tutti i monitor

Il sistema, secondo quanto indicato dai medici, dovrà essere in grado di gestire due o tre monitor collegati ad un unico gateway nello stesso intervallo

di tempo, poichè questo è il numero di interventi contemporanei che possono essere effettuati nella stessa finestra temporale. I dati possono venire richiesti esplicitamente al sistema prima e dopo l'esecuzione di certe manovre e somministrazione di farmaci, e in modo periodico nel caso peggiore ogni due minuti.

### 3.2.2 Analisi

Per prima cosa è necessario modellare il dominio (figura 3.21).

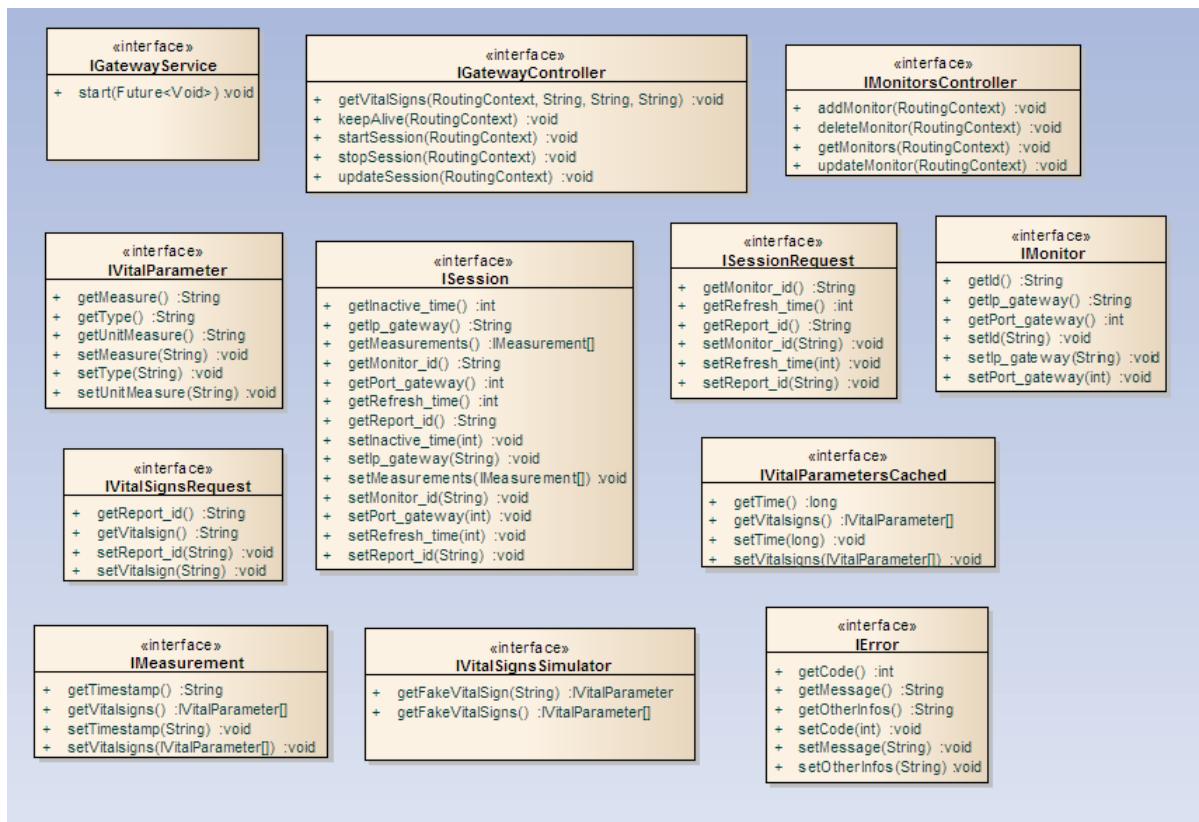


Figura 3.21: Modello del dominio (GatewayService)

- *IVitalParameter*: entità che modella il parametro vitale. Un parametro vitale è composto da tre campi:
  - Tipo: tipologia del parametro vitale (DIA, SYS, EtCO2, Temp, HR, SpO2)
  - Misura: valore della misurazione del parametro vitale
  - Unità di misura: unità di misura del parametro vitale

- *IMonitor*: entità che modella i monitor. Un monitor è composto da tre campi:
  - ID: identificativo univoco del monitor. Per semplicità, si è scelto di usare lo stesso identificativo che utilizza il sistema di Draeger. I monitor sono infatti identificati tramite due valori, che tipicamente riguardano la posizione del monitor all'interno dell'ospedale:
    - \* Codice unità reparto (CU : Care Unit label). Ad esempio, terapia intensiva (codice TI).
    - \* Codice lettino/monitor (BU: Bed Unit label). Ad esempio, SALA1
  - Questi due valori sono uniti tra di loro tramite il carattere ”\_” per formare l'identificativo, quindi si avrà la forma *CU\_BI* (ad esempio *TI\_SALA1* è l'identificativo del monitor che ci è stato fornito per fare le prove).
  - IP del gateway: indirizzo IP del gateway associato al monitor. In questo modo, dato un monitor, è possibile interrogare il gateway ad esso associato per ottenere i dati
  - Porta del gateway: numero di porta del gateway associato al monitor. In questo modo, come per il campo precedente, dato un monitor, è possibile interrogare il gateway ad esso associato per ottenere i dati
- *IMeasurement*: entità che modella una misurazione dei parametri vitali per quanto riguarda la modalità del tracciamento periodico. È composta da:
  - Timestamp della misurazione
  - Parametri vitali misurati
- *ISession*: entità che modella una sessione, ovvero tutte le informazioni che riguardano il tracciamento dei parametri vitali nella modalità di tracciamento periodico. Si compone di:
  - ID del report a cui associare il tracciamento (servirà alla applicazione di terze parti per recuperare le misurazioni effettuate)
  - ID del monitor a cui è associato il tracciamento
  - Indirizzo IP e numero di porta del gateway associato al monitor
  - Intervallo di tempo ripetuto il quale è necessario misurare i parametri

- Tempo di inattività: per evitare che sessioni restino aperte all’infinito (in caso in cui il client non richieda i dati immediatamente alla fine dell’intervento per qualunque ragione) è necessario tenere traccia di quanto tempo la sessione resta attiva. La modalità in cui tutto questo viene gestito verrà esposta più avanti
- Tutte le misurazioni effettuate (IMeasurement)

- *ISessionRequest*: modella una richiesta di apertura di sessione da parte di un client. Si compone di:

- ID del report a cui associare il tracciamento (servirà alla applicazione di terze parti per recuperare le misurazioni effettuate)
- ID del monitor a cui è associato il tracciamento
- Intervallo di tempo

Questi dati vengono utilizzati dal servizio per costruire la sessione (ISession)

- *IVitalParametersCached*: a causa delle problematiche relative al troncamento dei messaggi discusso in precedenza, è necessario prevedere dei meccanismi di caching per evitare che richieste ravvicinate relative allo stesso monitor vengano processate. Si compone di:

- Timestamp per capire quando la misurazione è stata effettuata (servirà per capire se restituire la seguente misurazione a fronte di una richiesta o no)
- I parametri vitali misurati

- *IVitalSignsSimulator*: entità che modella un piccolo simulatore per fornire parametri vitali falsi. Utile per effettuare operazioni di debugging. Permette di recuperare tutti e sei i parametri vitali di interesse o uno specifico

- *IError*: entità che modella l’errore durante l’elaborazione delle richieste ricevute. Si compone di tre campi:

- Codice errore: codice relativo all’errore. Affinchè tutto sia standardizzato viene usato il codice di stato HTTP.
- Messaggio di errore: messaggio di errore che specifica la problematica
- Informazioni aggiuntive: eventuali informazioni che possono essere utili, in particolare ai programmati delle applicazioni di terze parti (ad esempio lo stacktrace delle eccezioni, errori delle query etc)

In questo modo le applicazioni di terze parti hanno tutte le informazioni necessarie

- *IGatewayService*: entità che si occupa di inizializzare il servizio
- *IGatewayController*: entità che si occupa di elaborare le richieste relative al recupero dei parametri vitali, in particolare:
  - Recupero di tutti i parametri vitali di interesse
  - Recupero di un parametro vitale specifico
- *IMonitorController*: entità che si occupa di gestire le richieste relative alla modifica/aggiornamento delle informazioni relative ai monitor salvati nel sistema. E' importante tenere salvati i monitor che vengono utilizzati dal sistema poichè in questo modo le applicazioni di terze parti devono conoscere solamente l'identificativo del monitor e passarlo al servizio, che si occupa quindi di interrogare il monitor corretto. Questo servirà anche all'applicazione web che permetterà la visualizzazione da remoto dei parametri vitali associati ai monitor. In particolare, sono necessarie queste operazioni:
  - Recupero di tutti i monitor
  - Aggiunta di un monitor
  - Cancellazione di un monitor
  - Aggiornamento di un monitor

Ora è necessario modellare le API da fornire alle applicazioni, in modo tale da avere un quadro completo della dimensione dell'interazione. Tramite Swagger IO (A.4) è possibile costruire il modello delle API in modo dettagliato ma allo stesso tempo semplice e chiaro. In prima battuta verranno quindi modellate le API del servizio (di cui verrà mostrata per semplicità solo il modello generato, e non il codice YAML), e successivamente verrà modellata l'interazione tra le entità del dominio.

- Manipolazione monitor
  - Recupero di tutti i monitor (figura 3.22)
  - Aggiunta di un monitor (figura 3.23)
  - Modifica di un monitor (figura 3.24)
  - Eliminazione di un monitor (figura 3.25)
- Recupero parametri vitali

- Richiesta singolo parametro (figura 3.26)
- Richiesta di tutti i parametri (figura 3.27)
- Tracciamento periodico parametri vitali
  - Inizio di una sessione (figura 3.28)
  - Chiusura di una sessione (figura 3.29)
  - Modifica della frequenza di raccolta dei dati (figura 3.30)
  - KeepAlive sessione (figura 3.31)

**GET /gt2/gateway/api/monitors**

GatewayService

**Summary**  
Get all monitors

**Description**  
All monitors are returned

**Responses**

Code	Description	Schema
<b>200</b>	Monitors	<pre> [&gt; [     &gt;&gt; Monitor {         id:          &gt; string         ip_gateway: &gt; string         port_gateway: &gt; string     } ]   </pre>
<b>default</b>	Unexpected error	<pre> Error {     code:      integer     message:   string     otherInfos: string }   </pre>

**Try this operation**

Figura 3.22: Recupero di tutti i monitor

Ora si passa a descrivere le problematiche da affrontare relativamente al software da sviluppare e come sono state risolte.

**Latenza e richieste parallele** Le latenze riscontrate nei test sono state elevate (per quanto riguarda il software di Draeger). Come descritto in precedenza, il gateway può processare una richiesta in media in 190 millisecondi. Di media quindi solamente cinque richieste al secondo. Oltre a ciò, si aggiunge anche il tempo di latenza dopo l'apertura della connessione. Difatti, dai test è stato riscontrato che non appena la connessione viene aperta e inviata una query per recuperare i parametri vitali, è presente un delay iniziale

POST /gt2/gateway/api/monitors

GatewayService

**Summary**

Add a monitor

**Description**

Add a new monitor

**Parameters**

Name	Located in	Description	Required	Schema
monitor	body	Monitor to create	Yes	<pre>▼ Monitor {     id: ▶ string     ip_gateway: ▶ string     port_gateway: ▶ string }</pre>

**Responses**

Code	Description	Schema
201	The monitor has been added	
default	Unexpected error	<pre>▼ Error {     code: integer     message: string     otherInfos: string }</pre>

Try this operation

Figura 3.23: Aggiunta di un monitor

**PUT /gt2/gateway/api/monitors/{id}**

GatewayService

**Summary**  
Update a monitor

**Description**  
Update a monitor

**Parameters**

Name	Located in	Description	Required	Schema
<code>id</code>	path	ID of the monitor to update	Yes	<code>integer</code>
<code>monitor</code>	body	Monitor to update	Yes	<code>Monitor {</code> <code>    id: string</code> <code>    ip_gateway: string</code> <code>    port_gateway: string</code> <code>}</code>

**Responses**

Code	Description	Schema
<b>200</b>	The monitor has been updated	
<b>default</b>	Unexpected error	<code>Error {</code> <code>    code: integer</code> <code>    message: string</code> <code>    otherInfos: string</code> <code>}</code>

[Try this operation](#)

Figura 3.24: Modifica di un monitor

**DELETE /gt2/gateway/api/monitors/{id}**

GatewayService

**Summary**  
Delete a monitor

**Description**  
Delete a monitor

**Parameters**

Name	Located in	Description	Required	Schema
<code>id</code>	path	ID of the monitor to delete	Yes	<code>integer</code>

**Responses**

Code	Description	Schema
<b>200</b>	The monitor has been deleted	
<b>default</b>	Unexpected error	<code>Error {</code> <code>    code: integer</code> <code>    message: string</code> <code>    otherInfos: string</code> <code>}</code>

[Try this operation](#)

Figura 3.25: Eliminazione di un monitor

GET /gt2/gateway/api/monitors/{monitor\_id}/{vitalsign}

GatewayService

**Summary**

Get a vital sign.

**Description**

Get one of the vital sign of the patient attached to the specified monitor

**Parameters**

Name	Located in	Description	Required	Schema
monitor_id	path	ID of the monitor [in the form "careunitlabel_bedunitlabel"]	Yes	≥ string
vitalsign	path	Vital sign to request	Yes	≥ string

**Responses**

Code	Description	Schema
200	Vital sign requested.	<pre>  * VitalParameter {     type:      &gt; string     unitMeasure: &gt; string     measure:   &gt; string   }   * Error {     code:     integer     message:  string     otherInfos: string   }</pre>
default	Unexpected error	<pre>  * Error {     code:     integer     message:  string     otherInfos: string   }</pre>

Try this operation

The screenshot shows a detailed API endpoint definition. At the top, the URL is listed as 'GET /gt2/gateway/api/monitors/{monitor\_id}/{vitalsign}' and the service is identified as 'GatewayService'. Below this, the 'Summary' section states 'Get a vital sign.' The 'Description' section specifies 'Get one of the vital sign of the patient attached to the specified monitor'. The 'Parameters' section contains two path parameters: 'monitor\_id' (located in path, required, schema: ≥ string) and 'vitalsign' (located in path, required, schema: ≥ string). The 'Responses' section defines two possible outcomes: a successful response (status 200) with a schema that includes a 'VitalParameter' object (containing 'type', 'unitMeasure', and 'measure') and an 'Error' object (containing 'code', 'message', and 'otherInfos'), and an unexpected error response (status default) with a similar schema for the 'Error' object. A 'Try this operation' button is located at the bottom of the responses table.

Figura 3.26: Richiesta singolo parametro

**GET /gt2/gateway/api/monitors/{monitor\_id}/vitalsigns**

GatewayService

**Summary**  
Get all relevant vital signs

**Description**  
Get all of the relevant vital signs of the patient attached to the specified monitor

**Parameters**

Name	Located in	Description	Required	Schema
monitor_id	path	ID of the monitor [in the form "careunitlabel_bedunitlabel"]	Yes	string

**Responses**

Code	Description	Schema
200	Vital signs requested.	<pre>     [         *VitalParameter {             type:      &gt; string             unitMeasure: &gt; string             measure:   &gt; string         }     ]   </pre>
default	Unexpected error	<pre>     *Error {         code:      integer         message:   string         otherInfos: string     }   </pre>

[Try this operation](#)

Figura 3.27: Richiesta di tutti i parametri

**POST /gt2/gateway/api/sessions**

GatewayService Session

**Summary**  
Start a monitoring session

**Description**  
Start a monitoring session for a monitor. The vital parameters will be measured every period of time specified

**Parameters**

Name	Located in	Description	Required	Schema
sessionRequest	body	Session to open	Yes	<pre>     *SessionRequest {         report_id:      &gt; string         monitor_id:     &gt; string         refresh_time:   &gt; integer     }   </pre>

**Responses**

Code	Description	Schema
200	The session has been opened	
default	Unexpected error	<pre>     *Error {         code:      integer         message:   string         otherInfos: string     }   </pre>

[Try this operation](#)

Figura 3.28: Inizio di una sessione

The screenshot shows a Swagger API documentation page for the `Session` resource. The operation is `GET /gt2/gateway/api/sessions/{report_id}`. The `report_id` parameter is required and of type `string`. The response schema is defined by the `VitalParameter` and `Error` objects. The `VitalParameter` object contains `type`, `unitMeasure`, and `measure` fields. The `Error` object contains `code`, `message`, and `otherInfos` fields. A `Try this operation` button is at the bottom.

Name	Located in	Description	Required	Schema
<code>report_id</code>	path	Report ID of the session	Yes	<code>string</code>

**Responses**

Code	Description	Schema
<b>200</b>	The session has been closed. The measures are returned	<code>VitalParameter</code> [ - <code>type</code> : <code>string</code> - <code>unitMeasure</code> : <code>string</code> - <code>measure</code> : <code>string</code> ]  <code>Error</code> { - <code>code</code> : <code>integer</code> - <code>message</code> : <code>string</code> - <code>otherInfos</code> : <code>string</code> }
<b>default</b>	Unexpected error	

Figura 3.29: Stop di una sessione

che ritarda l'elaborazione del messaggio. Il tempo aumenta ulteriormente se la chiusura e l'apertura della connessione avviene molto spesso in intervalli di tempo ravvicinati. Inizialmente, dato che secondo i requisiti la richiesta dei parametri vitali non deve venire effettuata molto spesso, si è pensato di utilizzare HL7SubSystem aprendo la connessione all'inizio di ogni richiesta e chiudendola alla fine di ognuna di esse. Questo però si è notato che portava a tempi di latenza molto alti per i problemi descritti in precedenza. In caso in cui ci fossero state anche solo due richieste parallele per lo stesso monitor, la seconda veniva processata in oltre tre secondi, poiché dopo la chiusura della connessione e la sua conseguente riapertura, il gateway rispondeva con molto ritardo. Per evitare il problema si è pensato di attuare una migliore gestione delle connessioni, non chiudendole ad ogni richiesta. In particolare si è scelto di:

- Mantenere in memoria una lista delle connessioni aperte, in modo che:
  - Se a fronte di una richiesta il servizio ha già in precedenza aperto una connessione con il gateway associato al monitor richiesto, non la apre nuovamente e invia solo il messaggio al gateway

**PUT /gt2/gateway/api/sessions/{report\_id}**

GatewayService Session

**Summary**  
Update a monitoring session

**Description**  
Update the refresh time of a monitoring session (other data can't be modified)

**Parameters**

Name	Located in	Description	Required	Schema
report_id	path	Report ID of the session	Yes	☞ string
sessionRequest	body	Session to update	Yes	☞ <pre>*SessionRequest {     report_id: &gt; string     monitor_id: &gt; string     refresh_time: &gt; integer }</pre>

**Responses**

Code	Description	Schema
<b>200</b>	The refresh time of the session has been updated	
<b>default</b>	Unexpected error	☞ <pre>*Error {     code: integer     message: string     otherInfos: string }</pre>

**[Try this operation]**

Figura 3.30: Aggiornamento tempo di refresh di una sessione

**GET /gt2/gateway/api/sessions/{report\_id}/keepalive**

GatewayService Session

**Summary**  
Keep the session alive

**Description**  
Keep the session alive, or it will be closed automatically after a certain amount of time

**Parameters**

Name	Located in	Description	Required	Schema
report_id	path	Report ID of the session	Yes	☞ string

**Responses**

Code	Description	Schema
<b>200</b>	The inactive timer of the session has been resetted	
<b>default</b>	Unexpected error	☞ <pre>*Error {     code: integer     message: string     otherInfos: string }</pre>

**[Try this operation]**

Figura 3.31: KeepAlive di una sessione

- \* Se il tutto va a buon fine verrà ricevuta una risposta nei circa 190ms riscontrati
- \* Se non viene ricevuta una risposta entro un timeout prefissato (1 secondo) significa che è necessario considerare che la connessione è stata chiusa da gateway. In questo caso è necessario aprirla e inviare nuovamente il messaggio
- Se a fronte di una richiesta il servizio non ha in precedenza aperto una connessione con il gateway associato al monitor richiesto, viene:
  - \* Aperta la connessione
  - \* Salvata in memoria la connessione
  - \* Inviato il messaggio al gateway come nel caso precedente

Si è considerato il caso con più gateway poichè, anche se al momento all'ospedale ne è presente uno solo, in futuro potrebbe esserci la necessità di gestirne di più. Sfruttando questo modello è possibile limitare i casi in cui la latenza sia elevata. In caso di funzionamento regolare infatti sarà possibile processare fino a 5 richieste al secondo senza nessun problema. Sarà presente solo della latenza elevata quando per qualche ragione la connessione viene chiusa da parte del gateway (ma a meno di problemi questo non dovrebbe accadere).

**Troncamento dei messaggi** Come descritto nella sezione relativa ad HL7SubSystem, il software di Draeger invia dei messaggi troncati (che quindi non contengono tutti i parametri vitali di interesse) nel caso in cui arrivino più richieste ad esso relative allo stesso monitor in un arco temporale minore di circa 800ms. A prescindere dal fatto che il problema possa o non possa accadere anche all'interno dell'ospedale, conviene sfruttare la problematica per attuare delle soluzioni per sia risolvere le seguenti problematiche:

- La problematica appunto del troncamento dei messaggi
- Numero di richieste processabili in un secondo: sono solo 5 (come descritto nel paragrafo precedente). Evitando che vengano processate relative allo stesso monitor in un dato intervallo di tempo è possibile migliorare questo valore

Per farlo si è pensato di introdurre un sistema di caching che ha il seguente funzionamento:

- Non appena arriva una richiesta relativa a un monitor, essa viene processata misurando tutti i parametri vitali di interesse, e i valori vengono salvati, associando loro il timestamp

- Se nell’arco del secondo secondi successivi al salvataggio in cache dei dati devono venire processate altre richieste relative al monitor in questione, vengono restituiti i dati salvati senza misurarli nuovamente
- Se il secondo è passato i parametri verranno richiesti al gateway

Se durante i test in ospedale i tempi riscontrati saranno diversi, sarà possibile modificare molto facilmente i parametri ed effettuare del tuning per ottimizzare ulteriormente il tutto.

**Chiusura automatica delle sessioni** E’ necessario tenere in considerazione, per quanto riguarda la modalità di tracciamenti periodico dei parametri vitali, che per qualche ragione (mancanza di connettività, dimenticanza..) la sessione non venga chiusa (recuperando quindi i dati misurati) subito dopo la fine dell’intervento ma che passi del tempo. Per evitare che rimangano sessioni aperte che quindi potrebbero, a fronte di sempre nuove sessioni aperte, degradare i tempi di risposta del sistema, si è pensato di attuare la seguente soluzione:

- Decidere una soglia (configurabile) oltre la quale bisogna considerare di chiudere la sessione (quindi smettere di misurare i parametri vitali)
- Associare alle sessioni il tempo di inattività in modo tale da capire quando esso supera la soglia
- Ogni volta che trascorre la frequenza di misurazione, oltre a compiere la suddetta misurazione:
  - Si controlla il tempo di inattività: se ha superato la soglia la sessione viene chiusa
  - Se il tempo di inattività non ha superato la soglia viene sommato al tempo di inattività della sessione la frequenza di misurazione

Affinchè una sessione venga tenuta attiva anche oltre il dovuto, si è pensato di introdurre un sistema di *KeepAlive*:

- Il client può inviare una richiesta al servizio specificando un report id (identificando quindi una sessione) in modo tale da resettare il tempo di inattività di essa, di fatto prolungandola

### 3.2.3 Progetto

La prima considerazione da fare è relativa all'interfacciamento con il gateway di draeger. Il servizio dovrà necessariamente recuperare i parametri vitali, in modo tale da renderli disponibili alle applicazioni. Per questa ragione si servirà del componente HL7SubSystem per interagire con il gateway di Draeger al fine di recuperare i parametri. Ora è necessario scegliere opportunamente la tecnologia lato server per implementare il servizio. E' stato scelto Vert.x per diverse ragioni:

- Modello a event loop, che consente di avere ottime prestazioni
- Grazie all'event loop, non è necessario dovere gestire manualmente aspetti critici del multithreading che avrebbero aggiunto solo ulteriore complessità inutile
- Grande supporto per quanto riguarda librerie e client per i più utilizzati DBSM
- Supporto per la creazione di server REST e supporto per tutti gli aspetti che riguardano questa architettura

Si è scelto di usare Java perchè essendo HL7Subsystem sviluppato appunto nel medesimo linguaggio, l'integrazione dei due componenti risulta semplice ed immediata. Scegliere un altro linguaggio non avrebbe avuto quindi senso poichè in ogni caso le funzionalità del framework sarebbero state le stesse. E' necessario anche scegliere le tecnologie adatte per memorizzare le informazioni relative ai monitor. E' difatti necessario salvarli affinchè i client abbiano solamente la necessità di semplicemente specificare l'identificativo del monitor e il/i parametri vitali da richiedere. Sarà il servizio invece a interrogare il gateway corretto all'indirizzo IP e porta opportuni, sfruttando HL7Subsystem. Per memorizzare i dati è stato scelto di usare un'approccio NoSQL (A.6), in particolare con MongoDB (A.6.1). Si è scelto di usare NoSQL piuttosto che SQL per alcune ragioni:

- Maggiori prestazioni
- Le query previste non sono molto complesse e quindi non utilizzare SQL non rappresenta un problema
- Il modello dei dati potrebbe avere bisogno di cambiamenti in futuro e quindi l'approccio NoSQL risulta migliore per gestire requisiti mutevoli nel tempo, grazie al fatto che il modello dei dati non deve avere una struttura fissa

MongoDB è stato scelto perchè è il più usato, più supportato e più robusto DBMS NoSQL in circolazione. Dopo aver scelto le tecnologie con cui realizzare il servizio, è opportuno andare più a fondo e descrivere il modello di comportamento del servizio sviluppato.

### Funzionamento del servizio

Ora verrà illustrato il funzionamento del sistema nelle sue parti. Prima di tutto è necessario menzionare che ai fini del funzionamento vengono sfruttate due collezioni MongoDB:

- *monitors*: collezione che contiene i dati relativi ai monitor salvati nel sistema
- *sessions*: collezione che contiene le sessioni (che a loro volta contengono anche le misurazioni dei parametri vitali) relative al tracciamento periodico dei parametri vitali

**Richiesta parametri vitali** In questo paragrafo si illustra il funzionamento della richiesta dei parametri vitali. I passaggi da effettuare sono i seguenti:

1. Recupero dei dati relativi al monitor richiesto dal client (è necessario conoscere l'indirizzo IP e il numero di porta del gateway da interrogare)
2. Se il monitor è presente nel sistema, si procede con il recupero dei parametri vitali
  - Se è necessario prelevare i valori dalla cache, non viene interrogato il gateway
  - Se non bisogna prelevare i dati dalla cache, viene interrogato il gateway e, se l'operazione va a buon fine, vengono restituiti i dati al client

In tutti gli altri casi e nel caso in cui la query al database fallisca ritorna errore al client. Si noti che è necessario utilizzare uno dei thread di Vert.x del pool per interrogare il gateway, in quanto è importante non bloccare l'event loop

**Tracciamento periodico dei parametri vitali** Di seguito viene illustrato il funzionamento del tracciamento periodico dei parametri vitali. I dati vengono salvati all'interno della collezione *sessions*.

1. Il client invia una richiesta di apertura della sessione specificando il report ID, il monitor da tracciare e la frequenza

2. Il client dopo un certo periodo di tempo può richiedere i dati tracciati
3. Il client può inviare una richiesta per modificare la frequenza con cui raccogliere i dati
4. Il client può inviare una richiesta per prolungare la durata della sessione e evitare che essa venga terminata automaticamente dopo un certo periodo di inattività

**Tracciamento periodico dei parametri vitali - Sessione** Non appena viene aperta una sessione, i parametri vitali vengono recuperati ogni intervallo di tempo specificato. Ogni intervallo di tempo vengono performati i seguenti passi:

1. Si controlla la presenza della sessione nel database (se non esiste la sessione termina - significa che il client ha richiesto i dati e quindi la sessione è stata eliminata dal database)
2. Se esiste si controlla che non sia da chiudere perché il tempo di inattività ha superato la soglia prefissata o perché il tempo di refresh è cambiato (per quest'ultimo caso si veda 3.2.3)
3. Se la sessione non va chiusa viene aggiornato il tempo di inattività della sessione sommando la frequenza di aggiornamento
4. Vengono richiesti i parametri vitali al gateway come in 3.2.3, con la differenza che i dati non vengono ovviamente restituiti al client ma vengono salvati nel database
5. I valori dei parametri vitali vengono salvati nella sessione

In caso di errori per quanto riguarda la comunicazione con il gateway, se le query al database falliscono e se il tempo di inattività ha superato la soglia, la sessione viene terminata e la raccolta dei dati termina.

**Tracciamento periodico dei parametri vitali - Apertura sessione**

1. Si controlla che la sessione non esista già
2. Se non esiste, si controlla che il monitor richiesto sia memorizzato all'interno del sistema
3. Se il monitor esiste, viene creata la sessione e aggiunta al database
4. La sessione può quindi iniziare a raccogliere i dati

In tutti gli altri casi e se le query al database falliscono ritorna errore al client.

**Tracciamento periodico dei parametri vitali - Chiusura sessione**

1. Si controlla che la sessione esista
2. Se esiste, si elimina la sessione dal database
3. Vengono quindi restituiti al client i dati raccolti

In tutti gli altri casi e se la query al database fallisce viene ritornato errore al client.

**Tracciamento periodico dei parametri vitali - Aggiornamento sessione** E' permesso aggiornare solamente il tempo di refresh della sessione, non gli altri dati (report ID e monitor ID)

1. Si controlla che la sessione esista nel database
2. Se esiste, si controlla che si intende cambiare solo il refresh time
3. Se è così, viene aggiornata la sessione nel database e fatta partire una nuova sessione (quella vecchia verrà chiusa poichè si controllerà nella sessione (3.2.3) se il tempo di refresh è variato

In tutti gli altri casi e se le query al database falliscono ritorna errore al client.

**Tracciamento periodico dei parametri vitali - KeepAlive sessione** E' possibile azzerare il tempo di inattività delle sessioni per prolungarle ed evitare la loro chiusura automatica.

1. Si controlla che la sessione esista nel database
2. Se esiste, si azzerà il tempo di inattività della sessione

In tutti gli altri casi e se la query al database fallisce ritorna errore al client.

### **3.2.4 Test del sistema**

Di seguito verranno illustrati i test effettuati ed i risultati ottenuti relativamente al servizio sviluppato. Per effettuare i test sono state scritte delle classi apposite. Per i test di performance e carico invece è stato usato JMeter (A.9) in modo tale da avere un quadro migliore e con informazioni aggiuntive. Per effettuare il setup del sistema è necessario seguire i passaggi effettuati per HL7SubSystem, con l'aggiunta di effettuare il deploy del servizio su un'altra macchina. Il setup di test, precisamente è stato quello di avere:

- Router collegato alla rete dell'università, con connessi:
  - Portatile personale (su cui gira il servizio ed i test)
  - Un altro portatile (su cui gira il Gateway di Draeger)
  - Monitor

Per eventualmente comunicare con il software di Draeger anche da remoto è necessario effettuare due passaggi:

- Impostare il port forwarding dal router per fare in modo che le connessioni ricevute dal router vengano effettuate in realtà sul portatile contenente il gateway
- Connettersi tramite VPN alla rete dell'università

Per evitare latenze indesiderate i test sono stati comunque effettuati in rete locale e non da remoto. Prima di procedere si consideri che:

- *SC - soglia chiusura*: soglia di tempo oltre la quale la sessione viene terminata per tempo di inattività. E' stato considerato 1 minuto per fare i test (il parametro è configurabile da file di configurazione in caso in cui sia necessario variarlo - sicuramente andrà variato sulla base di quanto diranno i medici)
- *TC - tempo caching*: tempo per il quale i parametri vitali recuperati relativamente ad un monitor vengono mantenuti salvati in cache (e quindi richieste entro questo tempo successive alla prima non comporteranno l'interrogazione del gateway). E' stato considerato un tempo di 1 secondo (per coprire l'intervallo di 800ms che provoca il problema del troncamento dei messaggi), ma anche in questo caso il tempo è variabile tramite file di configurazione

Di seguito vengono quindi illustrati i test effettuati.

### Test funzionalità

- Test del recupero dei parametri vitali: è stato verificato che il recupero dei parametri vitali, sia singoli sia di tutti, sia effettuato in modo corretto. Il test è stato effettuato tramite una classe di test ma si può anche verificare il recupero di tutti i parametri tramite l'applicazione web Monitors Viewer
- Test sessione: è stato testato il funzionamento del sistema delle sessioni con una classe di test che effettua i seguenti passaggi:

- Apertura sessione con refresh time di 5 secondi
- Attesa di un dato periodo di tempo (inferiore a SC). Nel test pari a 20 secondi
- Recupero dei parametri vitali tramite chiusura della sessione

Sono stati quindi controllati i dati ricevuti, appurando che essi sono stati raccolti ogni 5 secondi come richiesto, e infine si è controllato che i dati siano stati effettivamente eliminati dal database. Il test è stato ripetuto diverse volte variando i tempi di refresh

- Test chiusura automatica sessione: è stata realizzata una classe di test che effettua i seguenti passaggi:

- Apertura sessione
- Attesa di un tempo superiore a SC (nel test 120 secondi, il doppio di SC)
- Chiusura sessione

Si è quindi controllato che i dati sono stati raccolti solo per il primo minuto e non per il secondo minuto, e che quindi la sessione è stata chiusa come per inattività

- Test ibrido: è stato testato il funzionamento delle due modalità (richiesta parametri vitali e sessione) con un test che effettua i seguenti passaggi:

- Apertura sessione con refresh time di 5 secondi
- Richiesta parametri vitali ogni 3 secondi
- Chiusura sessione dopo un tempo inferiore a SC (nel test 40 secondi)

Si è quindi controllato che i parametri vitali venissero comunque richiesti nonostante la presenza della sessione e che le due modalità di funzionamento non interferissero tra loro

- Test aggiornamento intervallo di tempo sessione: è stata realizzata una classe di test che effettua i seguenti passaggi:

- Apertura sessione
- Cambiamento tempo di frequenza tramite richiesta dopo un dato lasso di tempo
- Chiusura della sessione dopo un altro lasso di tempo

Verificando i dati ricevuti è possibile controllare che effettivamente il tempo di refresh è stato cambiato correttamente

- Test KeepAlive sessione: è stata realizzata una classe di test che effettua i seguenti passaggi:
  - Apertura sessione con un tempo di refresh di 5 secondi
  - Invio richiesta di keepalive della sessione dopo un lasso di tempo inferiore a SC (nel test 50 secondi)
  - Chiusura sessione dopo 2 minuti e 30 secondi

Verificando i dati ricevuti è possibile controllare che la sessione è stata prolungata e che quindi la raccolta dei dati è proseguita e non è stata fermata la sessione dopo 60 secondi

**Test performance - gateway reale con monitor reale** Oltre ai test che validano le funzionalità del sistema, è necessario anche andare ad effettuare dei test di performance per capire la quantità di richieste che il sistema riesce a processare. Sono stati quindi effettuati alcuni test tramite il software JMeter. Dato che si dispone di un solo monitor e di un solo gateway reale, prima di tutto è stato testato il funzionamento di questa configurazione, per poi passare alla simulazione in cui sono stati simulati più monitor, inserendo nel simulatore i tempi di delay medi ottenuti dai test di HL7SubSystem. Uno dei punti critici riguarda il processing di richieste che arrivano parallelamente da più client riguardo lo stesso monitor. Il sistema è risultato in questo caso molto performante, grazie in particolare al sistema di caching che permette di gestire tantissime richieste ravvicinate. Test effettuato:

- 10 Utenti
- Ram up time: 1 secondo (perciò ogni 0.1 secondi viene creato un utente e lanciata la sua richiesta)
- Ogni utente ripete la richiesta 5 volte

Dai risultati si nota che per quanto riguarda le richieste che non hanno comportato l'interrogazione del gateway, il tempo di risposta è stato inferiore ai 30ms (tempo massimo riscontrato, ma generalmente il tempo era compreso tra 10 e 20 ms). Solamente i primi messaggi pervenuti hanno una latenza maggiore poichè appunto il gateway deve venire interrogato e questo richiede come detto in precedenza più tempo, e 210ms in caso in cui sia già aperta (il tempo medio è leggermente superiore a quello riscontrato nei test di HL7SubSystem poichè è necessario passare attraverso al servizio che tra le altre cose effettua anche una query al database). Quindi è possibile fare alcune considerazioni:

- Se la connessione è già aperta, il primo messaggio avrà un tempo di risposta di circa 210ms. I messaggi pervenuti a massimo 210ms rispetto al primo soffriranno di una simile latenza in quanto processati dopo il primo (quindi 210ms+tempo di processing in cache). I messaggi che arrivano a distanza superiore di 210ms e inferiore a TC dopo il primo saranno processati quasi istantaneamente perché non è necessario attendere il processing del primo messaggio ma si accede direttamente alla cache
- Il caso peggiore riguarda il fatto che la connessione sia da aprire prima di interrogare il gateway. In questo caso valgono gli stessi ragionamenti del caso precedente ma sostituendo 210ms con 1-1.5 secondi. Nonostante non sia un tempo molto basso, è purtroppo inevitabile ma fortunatamente i requisiti del sistema permettono che ci sia un tempo di risposta di questo ordine di grandezza. In ogni caso una latenza così elevata verrebbe riscontrata solo a fronte del primo messaggio al boot del sistema relativo a un gateway e successivamente solo se il gateway per qualche ragione termina la connessione (ma non dovrebbe succedere a regime)
- Considerando il caso peggiore di processing in cache che è stato di 30ms (ma generalmente è anche inferiore come detto in precedenza), il sistema è in grado di processare le seguenti richieste ogni secondo:
  - Se è necessario connettersi: considerando il tempo peggiore di 1.5 secondi, non è possibile processare neanche una richiesta ogni secondo
  - Se non è necessario connettersi, in un secondo è possibile processare  $1 + ((1000 - 210) / 30) = 27,33$  richieste (deve venire processato il primo messaggio)

**Test performance - un gateway simulato con più monitor** Anche senza possedere più monitor è comunque possibile effettuare dei calcoli per capire quanto può incidere il fatto di avere più monitor da interrogare:

- Alle considerazioni precedenti va moltiplicato il valore di 210ms per il numero di monitor che bisogna interrogare
  - Se non è necessario aprire la connessione, è quindi possibile processare fino a  $(1000 / 210) = 4,76$  messaggi ogni secondo, considerando il caso in cui arrivino messaggi tutti relativi a monitor diversi

Infine va tenuto conto che è sicuramente molto raro che le interrogazioni riguardo diversi monitor o gateway avvengano nello stesso preciso istante, ma va

tenuto in considerazione come caso peggiore in quanto possibile, e data la criticità del sistema in quanto usato in un ambito importante come l’healthcare, non può essere ignorato. Ora è necessario effettuare qualche test per verificare che i tempi di risposta sono quelli specificati prima. A tal scopo è necessario utilizzare il simulatore in quanto non si dispone di più monitor con il gateway reale. E’ stato considerato il seguente scenario di test:

- Un gateway con 5 monitor diversi
- Su ogni monitor vengono effettuate 2 richieste ogni secondo (in totale quindi il servizio riceverà 10 richieste al secondo)
- Eseguire almeno 1000 HTTP Request

Per effettuare questo test è necessario impostare JMeter con 2 thread per ognuna delle 5 HTTP Request da effettuare (una per ogni monitor). Per fare in modo che ogni thread effettui una richiesta al secondo è necessario impostare un Constant Throughput Timer con 60 sample per minuto per ogni thread. I risultati sono riportati in figura 3.32:

- Il tempo medio risulta di 358ms
- La mediana è 378ms
- Il tempo minimo è molto basso (questo trova giustificazione nel fatto che il gateway simulato è sulla stessa macchina su cui risiede il servizio, quindi non saranno presenti latenze, in particolare quando vengono restituiti i dati in cache. Nella rete ospedaliera sicuramente ci saranno dei tempi di latenza superiori)
- Il tempo massimo è 1095ms
- I valori della 90%, 95% e 99% linea sono rispettivamente di 657,780,999ms. Questo significa che sono pochi i valori ad avere ottenuto alti tempi di risposta rispetto ai valori medi

Considerando che i medici hanno comunicato che per quanto riguarda le richieste di parametri vitali periodiche il caso in cui l’intervallo di tempo è più ristretto è di circa 2 minuti, anche se è possibile che i parametri vengano richiesti manualmente altre volte, lo scenario di test considerato è ampiamente più difficile di quanto potrebbe accadere nella realtà, perciò i risultati ottenuti sono soddisfacenti. Oltre a ciò si aggiunge anche il fatto che per il momento i monitor che potrebbero essere attivi contemporaneamente sono 2-3 e non 5. Si consideri infine che dato che tutte le considerazioni effettuate riguardano la rete di test, sarà necessario considerare anche degli inevitabili tempi di latenza aggiuntivi per le richieste dopo avere effettuato i test in ospedale.

Label	# Samples	Average	Median	90% Line	95% Line	99% Line
HTTP Request	1015	358	378	657	780	999
TOTAL	1015	358	378	657	780	999
Min	Max	Error %	Throughput	Received KB/sec	Sent KB/sec	
0	1095	0,00%	10,0/sec	5,13	2,21	
0	1095	0,00%	10,0/sec	5,13	2,21	

Figura 3.32: Risultati test un gateway e cinque monitor

**Test performance - più gateway simulati con più monitor** Attualmente in ospedale è presente un solo gateway, ma non è da escludere che in futuro possano esserne utilizzati altri. E' quindi opportuno predisporre il sistema al funzionamento con più di un gateway. Di default Vert.x assegna un pool di thread di dimensione pari al numero di processori. Dato che ogni gateway viene processato in un worker thread apposito, è possibile processarne parallelamente lo stesso numero. E' opportuno quindi verificare che il servizio processi parallelamente le richieste relative a gateway diverse. Per testare questo è necessario effettuare lo stesso test di prima ma aggiungendo altri 5 monitor appartenenti ad un altro gateway. Se il test avrà risultati simili al precedente significa che le richieste sono effettivamente state parallelizzate. Sono state effettuate il doppio delle richieste del caso precedente poichè in questo caso i monitor presenti sono 10 al posto di 5. I risultati (come si nota in figura 3.33 sono molto simili al caso precedente, confermando il fatto che il tutto viene eseguito in parallelo.

Label	# Samples	Average	Median	90% Line	95% Line	99% Line
HTTP Request	2030	338	366	651	785	994
TOTAL	2030	338	366	651	785	994
Min	Max	Error %	Throughput	Received KB/sec	Sent KB/sec	
0	1088	0,00%	20,0/sec	10,27	4,41	
0	1088	0,00%	20,0/sec	10,27	4,41	

Figura 3.33: Risultati test con due gateway e cinque monitor per gateway

**Note finali** Le prestazioni del sistema possono in generale essere considerate buone, ma tutto cambia quando è necessario gestire in parallelo più monitor relativi allo stesso gateway. Purtroppo le prestazioni sono vincolate dal sistema di Draeger che non consente di fare di meglio. Sarà sicuramente necessario testare il sistema con il gateway reale presente in ospedale per verificare che i tempi di latenza e di risposta siano simili a quelli riscontrati. Come ultima nota, si illustra che è possibile ottenere prestazioni migliori a scapito della freschezza dei dati, modificando il parametro relativo al TC. Difatti, aumentando

il TC, sicuramente si ottengono dati meno freschi ma prestazioni migliori, in quanto molte più richieste saranno processate evitando di chiedere i dati al gateway. Se nel caso d'uso del sistema reale sarà necessario utilizzare molti monitor, sarà sicuramente necessario effettuare il tuning di questo parametro per ottenere risultati più soddisfacenti.

### 3.3 Monitors Viewer

Monitors Viewer è una semplice web app che permette di visualizzare i parametri vitali dei monitor da remoto. Oltre a ciò, permette di gestire i monitor presenti nel sistema, permettendo anche di aggiungerli, modificarli ed eliminarli. Va detto che anche Draeger fornisce del software per visualizzare da remoto i parametri vitali, tuttavia si è scelto di sviluppare comunque questa applicazione per due ragioni:

- Lo scopo principale dell'applicazione è quello di fornire un'interfaccia grafica per l'accesso al database in cui sono salvati i dati dei monitor. Questi dati vengono utilizzati da GatewayService per interfacciarsi con il gateway corretto, sulla base del monitor richiesto dalle applicazioni di terze parti
- Non è presente uno strumento che possa permettere di raccogliere tutti i monitor di interesse in un unico punto. Il software di Draeger permette di accedere ai dati dei monitor relativi ad un gateway specificando l'indirizzo IP del server gateway. Risulta quindi più immediato e veloce avere tutti i monitor di interesse in un unico posto, sia che siano tutti relativi allo stesso gateway sia che siano relativi a gateway diversi

#### 3.3.1 Requisiti

I requisiti dell'applicazione sono i seguenti:

- Possibilità di visualizzare i monitor salvati nel sistema con i loro dati
- Manipolazione dei monitor
  - Aggiunta di un monitor
  - Modifica di un monitor
  - Eliminazione di un monitor
- Possibilità di visualizzare i parametri vitali di un dato monitor

Si aggiunge anche un requisito non funzionale, ovvero l'usabilità da smartphone: l'applicazione deve essere disponibile e utilizzabile sia da pc sia da smartphone e l'interfaccia deve adattarsi a seconda del dispositivo utilizzato.

### 3.3.2 Analisi e progetto

**Manipolazione monitor** Un monitor si modella, come detto nella parte relativa a GatewayService, con le seguenti informazioni:

- Monitor ID: codice univoco del monitor, nella forma "careUnitLabel\\_bedUnitLabel"
- Indirizzo IP del server gateway a cui il monitor invia i dati
- Numero di porta del gateway

Per manipolare i monitor, è necessario inviare delle richieste a GatewayService per accedere al database.

**Visualizzazione parametri vitali** Si vuole visualizzare i dati dei monitor. A questo scopo, in modo tale da non appesantire troppo il carico sul server, è possibile impostare la frequenza di aggiornamento dei valori a 3 secondi. In questo modo, se chi accede all'app lo desidera, è possibile fare in modo i valori si aggiornino ogni 3 secondi. Per recuperare i dati è necessario inviare delle richieste a GatewayService specificando l'ID del monitor di interesse.

**Struttura applicazione** L'applicazione è stata suddivisa in due schermate:

- *Pagina principale*: nella pagina principale viene visualizzata la lista dei monitor salvati nel sistema. E' possibile aggiungerli, modificarli ed eliminarli. Il risultato attuale si può vedere in figura 3.34 e 3.35
- *Pagina relativa al monitor*: si accede a questa schermata cliccando su uno dei monitor della schermata precedente. Vengono visualizzati:
  - I parametri vitali
  - I dati del monitor
  - Le opzioni:
    - \* Possibilità di aggiornare i parametri vitali ogni tre secondi
    - \* Possibilità di eseguire un audio non appena i parametri vengono aggiornati

- Una legenda che riporta la spiegazione delle sigle relative ai parametri vitali riportati

Si può vedere il risultato attuale in figura 3.37 e 3.35

Prima di implementare è ovviamente necessario scegliere le tecnologie di riferimento da utilizzare. Sono state in particolare fatte due scelte:

- *Angular JS*: come framework di riferimento è stato scelto Angular JS. Esso risulta uno dei framework più utilizzati e supportati per quanto riguarda le tecnologie web. Garantisce anche ottime prestazioni essendo un framework single page, e permette grazie al binding dei dati tra il modello e la vista di creare in modo molto agevole applicazioni dinamiche.
- *Bootstrap*: si è deciso di utilizzare i componenti di bootstrap per garantire che l'applicazione sia responsive e che quindi possa essere fruita in modo opportuno sia da pc sia da smartphone

Dopo lo sviluppo del software è possibile validare il suo funzionamento testando il corretto funzionamento delle varie funzionalità.

## 3.4 Sviluppi futuri

In futuro sarà sicuramente necessario affinare e migliorare il sistema, affinchè possa essere usato ogni giorno all'interno dell'ospedale. In particolare sarà necessario affrontare le seguenti questioni:

- *Nuovi requisiti*: potrebbero emergere nuovi requisiti da soddisfare (come è già successo per quanto riguarda la modalità di tracciamento dei parametri vitali, che non è stata richiesta fin dall'inizio) sia per quanto riguarda eventuali altre nuove applicazioni smart che entreranno a far parte dell'architettura GT2 e che dovranno quindi interfacciarsi con il servizio sviluppato
- *Sperimentazioni in ospedale*: nonostante ci sia stato fornito un vero monitor, è necessario sperimentare il sistema in ospedale per affinarlo in caso in cui lo scenario non sia esattamente lo stesso, in quanto potrebbero esserci diverse latenze e altri aspetti che non coincidono con la rete di test creata
- *Aspetti di sicurezza*: saranno implementate misure di sicurezza:
  - Il servizio dovrà essere sicuramente distribuito tramite il protocollo HTTPS, in modo tale che il canale di comunicazione tra esso ed i client sia sicuro

The screenshot shows the main interface of the "Monitors Viewer" application. At the top, a blue header bar displays the title "Monitors Viewer" and the text "Developed by PSLAB @ DISI, University of Bologna - Cesena". Below the header is a table titled "Monitor disponibili" (Available Monitors) with 10 rows. Each row contains a monitor name (e.g., TI\_SALA2, TI\_SALA1, ..., TI\_SALA10) and two buttons: "Modifica" (orange) and "Elimina" (red). A blue "+" button is located at the top right of the table. At the bottom, a legend section defines the abbreviations: "NOME1\_NOME2: unità di reparto" and "NOME1\_NOME2: codice monitor/lettino".

Monitor disponibili	
TI_SALA2	Modifica Elimina
TI_SALA1	Modifica Elimina
TI_SALA4	Modifica Elimina
TI_SALA3	Modifica Elimina
TI_SALA5	Modifica Elimina
TI_SALA6	Modifica Elimina
TI_SALA7	Modifica Elimina
TI_SALA8	Modifica Elimina
TI_SALA9	Modifica Elimina
TI_SALA10	Modifica Elimina

Legenda

NOME1\_NOME2: unità di reparto  
NOME1\_NOME2: codice monitor/lettino

Figura 3.34: Pagina principale applicazione web (parte 1)

Inserire i dati

TI	SALA1	137.204.107.184	2550
----	-------	-----------------	------

Annulla Salva

Monitor disponibili

TI_SALA2	Modifica	Elimina
TI_SALA1	Modifica	Elimina
test_test	Modifica	Elimina

Legenda

NOME1\_NOME2: unità di reparto  
NOME1\_NOME2: codice monitor/lettino

Figura 3.35: Pagina principale applicazione web (parte 2)

# Monitors Viewer

Developed by PSLAB @ DISI, University of Bologna - Cesena

Parametri vitali

HR: 97 /min	DIA: 86 mm(hg)	SYS: 100 mm(hg)	EtCO2: 100 mm(hg)	SpO2: 92 %	Temp: 35 cel
-------------	----------------	-----------------	-------------------	------------	--------------

Informazioni sul monitor

Care unit label: TI
Bed unit label: SALA1
Ip gateway: 127.0.0.1
Porta gateway: 2550

Figura 3.36: Pagina monitor applicazione web (parte 1)



Figura 3.37: Pagina monitor applicazione web (parte 2)

- Prevedere meccanismi di autenticazione (esempio: token based authentication) per fare in modo che solo gli utenti autorizzati possano accedere al sistema, impedendo ad utenti indesiderati di accedervi. Utilizzando il protocollo OAuth2, si possono effettuare questi macro passaggi: nel sistema saranno inseriti solo i medici autorizzati a utilizzare il servizio (quindi quelli di TraumaTracker ed altri utenti aggiuntivi che utilizzeranno altre applicazioni). Il server genererà dei token da inviare alle applicazioni che effettuano l'autenticazione. Le applicazioni dovranno inviare i token ad ogni richiesta che effettueranno al servizio, ed esso controllerà che il token sia valido e non scaduto. Si può quindi procedere creando delle API per:
  - \* Recuperare un token temporaneo da utilizzare (fornendo al servizio le credenziali del medico)
  - \* Ottenere un nuovo token se il precedente è scaduto
 Il client quindi dovrà inserire il token nell'header delle richieste che effettua al server.
- Controllare le query che vengono effettuate nel database in modo tale da evitare attacchi di injection. Per fare questo è opportuno effettuare controlli sui dati inviati dai client per evitare l'esecuzione di query malevole
- Prevedere meccanismi per evitare attacchi che cerchino di aprire un enorme numero di sessioni o cerchino di sovraccaricare il sistema
- Altre misure di sicurezza

- *Certificazioni:* una volta che il sistema sarà completato, sarà necessario considerare la possibilità che il sistema (e anche TraumaTracker stesso) debba o meno ricevere delle certificazioni prima di poter essere impiegati ogni giorno all'interno dell'ospedale



# Conclusioni

All'interno delle strutture sanitarie sono presenti molteplici sistemi, sviluppati con diverse tecnologie da produttori diversi. L'eterogeneità di essi costringe alla ricerca di soluzioni di interoperabilità che possano abilitare la comunicazione e lo scambio di informazioni. Nel corso della tesi sono stati quindi illustrati i principali standard utilizzati ai giorni nostri, con particolare enfasi su HL7 (Health Level 7), l'insieme di standard per lo scambio di dati in ambito medico più diffuso al mondo. E' stata anche illustrata IHE (Integrating the Healthcare Enterprise), organizzazione dedita alla stesura di linee guida che riguardano il corretto uso degli standard, poichè essi da soli non bastano per costruire sistemi interoperabili, ma è necessario anche che i produttori li usino, implementino ed interpretino tutti allo stesso modo. Per quanto riguarda l'interoperabilità, anche se grazie alle soluzioni sopra citate la situazione è sicuramente molto migliorata, ancora non si è raggiunto uno stato perfetto poichè l'ambito medico è molto vasto e le esigenze e differenze locali che esistono tra struttura e struttura impediscono che sia sufficiente un modello comune per tutti. Si è quindi passato all'introduzione del progetto TraumaTracker, svolto in collaborazione con l'ospedale Bufalini di Cesena. Grazie alla crescita ed al miglioramento di tecnologie come quelle mobile, wearable ed hands free, è possibile pensare a sempre nuove applicazioni smart che possano supportare gli operatori sanitari nello svolgimento del loro lavoro, e di conseguenza migliorare la qualità della cura del paziente riducendo gli errori e minimizzando gli sprechi. Tuttavia utilizzare questo tipo di tecnologie, nonostante gli innumerevoli vantaggi, comporta anche dei vincoli, in quanto spesso i dispositivi hanno scarse capacità computazionali e scarsa durata della batteria. Inoltre è opportuno supportare le applicazioni che si basano su questo tipo di tecnologie con funzionalità aggiuntive (manipolazione dei dati raccolti, interazione con sistemi esistenti..). Per tutte queste ragioni è stata introdotta l'architettura GT2, un gruppo di servizi web che interagisce con i sistemi esistenti all'interno dell'ospedale senza interferire con il loro funzionamento, in modo tale da ridurre la complessità dello sviluppo delle applicazioni ed evitare che siano loro ad interagire con i sistemi dell'ospedale. I servizi web sono stati realizzati con lo stile architettonicale REST (REpresentational State

Transfer), in modo tale da costruire servizi che espongano API alle applicazioni. Si è infine parlato dello sviluppo di GatewayService, software che si occupa del recupero e tracciamento dei parametri vitali dei pazienti collegati ai monitor tramite interrogazione del server gateway di Draeger, che raccoglie questi dati. Questo mi ha permesso di rapportarmi in modo pratico con lo standard HL7, il più usato tra gli standard per lo scambio di informazioni tra sistemi in ambito medico. La scelta di sviluppare servizi REST risulta vincente, come confermano gli ultimi sviluppi sul fronte HL7: la nuova versione dello standard in lavorazione, ovvero FHIR, che dovrà rimpiazzare la poco apprezzata versione 3, punta proprio su questo. I test effettuati al sistema confermano che esso è in grado di soddisfare i requisiti con tempi di risposta accettabili. I requisiti attuali indicano che è presente un gateway e che verranno utilizzati nello stesso momento circa 2 o 3 monitor. Il sistema è in grado di gestire fino a 5 richieste di parametri vitali relative a diversi monitor ogni secondo e quindi i requisiti sono soddisfatti, poichè i monitor sono di meno ed inoltre i parametri vitali vengono chiesti periodicamente nello scenario peggiore ogni due minuti, e qualche altra volta prima e dopo l'esecuzione di qualche manovra o somministrazione di farmaci. Le sperimentazioni in ospedale dovranno confermare se il sistema è pronto per essere utilizzato in un contesto reale oppure se sarà necessario raffinarlo e migliorarlo. In conclusione, l'esperienza di progetto ha confermato che oggi è sicuramente possibile investire nello sviluppo di sistemi smart che utilizzano queste tecnologie al fine di migliorare l'acquisizione di dati in ambito medico. Il margine di miglioramento è ancora ampio in quanto certe tipologie di sistema in determinati contesti e strutture non sono ancora stati inseriti. Inoltre, qualora lo fossero già, sempre nuovi sistemi possono essere ideati, oltre alla possibilità di migliorare quelli esistenti.

# Appendice A

## Architetture e tecnologie utilizzate

Di seguito verranno illustrate le tecnologie, architetture e modelli utilizzati all'interno del progetto, relativamente al lavoro da me svolto.

### A.1 Microservizi

L'architettura a microservizi consiste nello sviluppo di applicazioni suddivise in servizi indipendenti tra loro, ognuno dei quali ha il suo processo e le sue funzioni. La divisione dipende solitamente dalle diverse funzionalità che l'azienda deve fornire. Come accennato, i microservizi devono essere indipendenti tra loro, ovvero che ognuno di essi può essere sviluppato, aggiornato e distribuito indipendentemente dagli altri (i microservizi potrebbero essere infatti sviluppati in linguaggi diversi e interagire con database costruiti con diverse tecnologie). L'indipendenza però non vieta la possibilità di effettuare dei controlli sui servizi in maniera centralizzata. Questo approccio nasce da diverse necessità:

- *Applicazioni non monolitiche*: necessità di non avere applicazioni server monolitiche, in cui ogni cambiamento necessita dell'aggiornamento dell'intera applicazione. Con questo approccio solo le parti interessate dell'applicazione vengono aggiornate
- *Gestione più semplice*: oltre alla questione degli aggiornamenti si aggiunge anche la maggiore facilità di gestione dell'applicazione quando essa diventa molto complessa ed estesa, essendo essa divisa in servizi indipendenti
- Generalmente infatti sono due le alternative:

- *Applicazione monolitica*: si suddivide l'applicazione sulla base delle tecnologie (ad esempio la user interface, il database e la parte server). Questo porta ad avere dei team di sviluppo specializzati (ad esempio un team per il database, un team per la user interface e così via)
- *Microservizi*: si suddivide l'applicazione sulla base delle aree di business. In questo caso i team sono suddivisi sulla base dei servizi da offrire, e avranno competenze su tutte le aree di rilievo (database, user interface e parte server). In questo modo è più facile gestire i cambiamenti, poiché ogni servizio è indipendente tra loro. Questo chiaramente necessita però di avere a disposizione personale qualificato che abbia conoscenze trasversali e non specifiche solo a determinate aree
- *Scalabilità*: si aggiunge anche la maggiore facilità di gestione di problematiche relative alla scalabilità (è infatti possibile occuparsi di tali problematiche servizio per servizio piuttosto che farlo per l'applicazione intera). Per quanto riguarda la scalabilità:
  - Applicazione monolitica: se è necessario aggiungere dei server per scalare, è necessario replicare l'intera applicazione su ognuno di essi per bilanciare il carico
  - Microservizi: se è necessario aggiungere dei server per bilanciare il carico, non è necessario obbligatoriamente replicare tutti i servizi su tutti i server, ma solo quelli per cui si ha la necessità di bilanciare il carico
- *Fallimenti*: i fallimenti dei singoli servizi non dovrebbero avere impatto sugli altri, proprio perché i servizi sono indipendenti tra loro

Non sono tuttavia presenti solo vantaggi in questo approccio:

- *Gestione distribuita*: Con l'architettura a microservizi l'approccio è distribuito e non centralizzato, perciò la gestione è sicuramente più complessa, in particolare se i servizi devono interagire tra di loro (il fallimento è dietro l'angolo nei sistemi distribuiti). In questa architettura i servizi possono interagire tramite richieste HTTP oppure tramite l'utilizzo di bus di messaggi. In una applicazione centralizzata invece l'interazione tra le sue parti si applica in modo decisamente più efficiente e sicuro tramite invocazione di metodi locali
- *Problema della consistenza*: nel caso di una applicazione monolitica il database in genere è unico. Nell'architettura a microservizi sono invece solitamente presenti diversi database, e alcuni database potrebbero

essere utilizzati da più servizi. Tutto questo può portare a problemi di consistenza dei dati (se ad esempio devono essere aggiornati i dati di più microservizi, potrebbero esserci delle finestre di tempo in cui tutti i microservizi non hanno ancora completato l'aggiornamento)

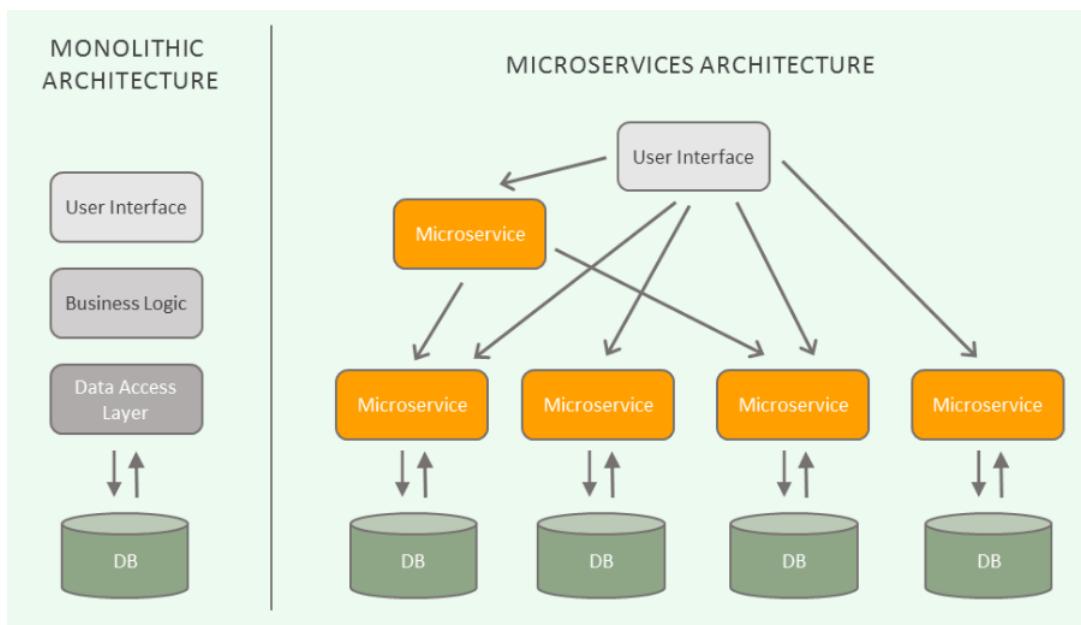


Figura A.1: Architettura a microservizi vs architettura monolitica [32]

## A.2 REpresentational State Transfer (REST)

Un servizio web è un componente software accessibile tramite la rete. Il punto di forza dei servizi web è che essi sono indipendenti dalle tecnologie impiegate. Vengono infatti definite delle interfacce per accedere a delle funzionalità che sono indipendenti dall'hardware, dal sistema operativo e dal linguaggio di programmazione utilizzato. E' per esempio possibile fornire un servizio web in Node.JS e accedervi tramite un client Android, Java, IOS e/o altro. In questa maniera inoltre i client costruiti con tecnologie diverse possono comunicare tra di loro grazie a servizi web. Esistono diversi modi per costruire servizi web, tra cui l'approccio REST (REpresentational State Transfer), in cui le risorse sono i servizi web. Grazie al fatto che in questo approccio si utilizzano protocolli internet (HTTP) e formati di messaggi standard (JSON, XML..) è possibile quindi sviluppare client e server con diverse tecnologie. Negli ultimi anni la crescita della diffusione di questo approccio è stata enorme, e difatti

attualmente questo approccio è lo standard de facto per le imprese che forniscono i loro servizi con un modello di consumazione di risorse web. È stato infatti adottato dalle maggiori compagnie al mondo, tra le quali Google, Facebook, Yahoo.. REST si può descrivere come una architettura in cui convivono componenti disaccoppiati tra loro che comunicano tramite interfacce costruite sui protocolli web standard. Di seguito vengono illustrati i quattro principi portanti di REST:

- *Identificazione delle risorse tramite URI*: le risorse sono identificate da degli URI. Esse inoltre sono organizzate come se fossero delle directory (quindi ad albero). Questo facilita anche la comprensione dei concetti da parte degli sviluppatori e non solo. Ad esempio:

`http://www.myservice.org/forum/threads/{thread_id}`

Supponiamo che ci sia un forum online. La radice sarà appunto il forum, in cui sono presenti svariati thread. Il thread che ci interessa è identificato da un id

- *Manipolazione delle risorse tramite un set di operazioni (metodi HTTP)*: le risorse possono venire manipolate tramite delle richieste HTTP. Una richiesta si compone di queste parti:

- VERB: nome del metodo HTTP
- URI: URI della risorsa
- HTTP Version: versione di HTTP da usare
- Request Header: collezione di metadati in formato chiave valore che riguardano la tipologia del messaggio e caratteristiche di esso
- Request body: contenuto del messaggio, quindi la rappresentazione di una risorsa

La risposta invece è simile alla richiesta, ma differisce per il fatto che non è presente VERB e URI ma è invece presente il Response Code (tipicamente i codici a tre cifre degli stati HTTP, per indicare appunto lo stato della richiesta, ovvero se è stata completata con successo o se ci sono stati dei problemi, e in caso quali). Di seguito vengono illustrate le richieste HTTP di base che permettono la manipolazione delle risorse:

- *PUT* (update): cambiamento di stato di una risorsa esistente. Esempio di PUT:

```
PUT /users/Robert HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
<name>Bob</name>
</user>
```

Si noti la composizione della richiesta rispetto a quanto detto prima:

- \* VERB: in questo caso è "PUT"
- \* URI: in questo caso è "/users/Robert")
- \* HTTP Version: in questo caso è "HTTP/1.1"
- \* Request Header: in questo caso è "Content-Type: application/xml")
- \* Request body: in questo caso è il codice XML
- *DELETE* (delete): eliminazione di una risorsa. Esempio di DELETE:

```
DELETE /users/Robert HTTP/1.1
Host: myserver
Accept: application/xml
```

- *GET* (read): recupero della rappresentazione di una risorsa. Si effettua una GET ad un URI specifico per recuperare direttamente una risorsa, oppure fornire dei parametri grazie ai quali il server possa restituire una determinata risorsa cercata. Esempio di GET:

```
GET /users/Robert HTTP/1.1
Host: myserver
Accept: application/xml
```

- *POST* (create): creazione di una nuova risorsa. Esempio di POST:

```
POST /users HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
<name>Robert</name>
</user>
```

Nessuno obbliga a seguire questo modello, nel senso che è possibile ad esempio effettuare l'aggiornamento di una risorsa tramite GET. Tuttavia questo approccio, oltre ad essere un'inconsistenza a livello semantico (dato che la GET è stata progettata per recuperare risorse e non per altro), porta anche a problemi pratici, in quanto i crawler o i motori di ricerca potrebbero involontariamente far partire delle modifiche alle risorse semplicemente navigando dei link. Per evitare side effect è quindi consigliato di seguire le linee guida e di utilizzare i metodi HTTP per quello per cui sono stati pensati

- *Rappresentazione delle risorse*: le risorse sono disaccoppiate dalla loro rappresentazione, in questo modo è possibile rappresentarle in diverse forme (XML, JSON, HTML, PDF etc). La rappresentazione di una risorsa quindi incapsula le informazioni relative ad essa al momento in cui avviene la richiesta
- *Interazioni senza stato*: le interazioni con le risorse sono senza stato. Lo stato viene trasferito esplicitamente nei messaggi. La mancanza dello stato è giustificata dalla necessità di costruire applicazioni scalabili in modo da avere buone prestazioni. A tal scopo le richieste dei client devono contenere tutti i dati che servono affinchè la richiesta possa venire processata (in modo tale da facilitare anche operazioni di load balancing come forwarding di richieste da un server all'altro). Il client diventa l'unico responsabile del mantenimento dello stato. Tutto questo porta anche alla semplificazione delle applicazioni lato server, in quanto dato che nelle richieste sono contenuti tutti i dati che servono, esse non dovranno conservare e elaborare dati necessari per essere allineati con lo stato dei client

In conclusione, diversi sono i vantaggi che si ottengono dall'utilizzo dell'architettura REST:

- Migliori prestazioni
- L'interazione senza stato migliora la scalabilità
- Semplificazione del codice sia lato server sia lato client (in questo caso anche un semplice browser può facilmente effettuare delle richieste)
- Facile individuazione delle risorse grazie agli URI
- Facilità nell'aggiungere nuovi tipi di rappresentazioni delle risorse senza dover rinunciare al supporto di altri tipi (esempio aggiungere il supporto ad XML oltre che a JSON)

- Standardizzazione delle interfacce e della rappresentazione delle risorse
- Possibilità di nascondere la complessità delle operazioni lato server (database multipli, più servizi, più server..) fornendo solamente delle API che ai consumatori risultano molto semplici da capire e da utilizzare

### A.2.1 Mobile computing

Un campo che trae vantaggio dai servizi web (REST in particolare) è sicuramente il mobile computing. Come noto, i dispositivi mobili sono ormai diventati pervasivi e sono posseduti da quasi tutti. I dispositivi vengono usati da chiunque in qualunque luogo per accedere appunto a servizi in mobilità. Nonostante la potenza dei dispositivi mobili stia crescendo di anno in anno, sono ancora presenti limitazioni da questo punto di vista (connessione ad Internet non sempre presente e stabile, minori prestazioni, durata della batteria limitata...). Comuni problematiche per i dispositivi mobili e i servizi web sono le seguenti:

- Prestazioni inferiori rispetto ad altre macchine
- Problemi di banda e perdita di connessione
- Risorse computazionali limitate (anche se in continua crescita nel tempo)

L'introduzione di REST ha migliorato di molto le prestazioni in ambito mobile. Un esempio di test che ha riscontrato le migliori prestazioni di REST rispetto ad altri approcci come SOAP si può trovare qui: [12]. Viene infatti riportato che usare un servizio web REST comporta tempi di risposta drasticamente inferiori rispetto a servizi web SOAP. I motivi sono da ricercare nel fatto che

- I messaggi XML SOAP sono di dimensione maggiore a causa della loro maggiore verbosità. Questo fatto può non influire di tanto su sistemi con buone prestazioni, ma si può sentire maggiormente su dispositivi mobili
- Possibilità di utilizzare meccanismi di caching. Questa possibilità consente anche di risparmiare molto traffico di rete (e anche di diminuire il carico di lavoro dei server)

Nell'articolo sono descritti dei semplici benchmark che sono stati eseguiti e hanno mostrato le migliori prestazioni di REST.

## A.3 JSON

JSON (JavaScript Object Notation) è un formato per lo scambio di dati. I suoi principali punti di forza sono il fatto che sia semplice da capire sia per gli umani sia per le macchine. La sua semplicità e la sua facilità di gestione da parte dei programmatori lo hanno reso il formato più utilizzato per lo scambio di dati. Due sono le possibilità:

- *Oggetto JSON*: set non ordinato di coppie chiave-valore

```
{
  name: "Piero",
  age: 40
}
```

- *Array JSON*: collezione ordinata di valori

```
["Piero", "Luca"]
```

I valori possono essere di tipo: stringa, numero, oggetto JSON, array JSON, booleano e null. Sono disponibili librerie per lavorare con questo formato in quasi tutti i linguaggi di programmazione. Questo formato può quindi venire utilizzato per costruire la rappresentazione delle risorse da scambiare tra i server REST e le applicazioni. Il vantaggio principale di questo formato rispetto ad altri formati molto diffusi (come ad esempio XML) è la sua sintassi molto semplice che quindi contribuisce a renderlo molto leggero, portando a migliori performance nel parsing.

## A.4 Swagger IO

Swagger IO è una specifica formale accompagnata da un grande insieme di tool, che coprono diversi aspetti (creazione di API, interfacce front end, librerie di basso livello). Lo scopo è di definire uno standard per la creazione e la documentazione di API REST indipendentemente dal linguaggio di programmazione usato. Solitamente quando si documentano delle API si usano dei tool che generano la documentazione a partire dal codice. L'approccio di Swagger è esattamente il contrario. Il grosso vantaggio di Swagger invece è che la descrizione delle API è sia comprensibile da umani sia da macchine, permettendo quindi di gestire contemporaneamente la descrizione delle API e la documentazione, potendo inoltre effettuare operazioni su di esse come la generazione di codice in diversi linguaggi di programmazione. Se si vuole definire delle API con Swagger, due sono le possibilità:

- *Approccio top down*: si utilizza lo Swagger Editor per definire le API e lo Swagger Codegen, se si desidera, per generare l'implementazione dello scheletro del server scegliendo tra tanti diversi linguaggi di programmazione
- *Approccio bottom up*: si usa se si hanno già delle API REST e si vogliono ridefinire con Swagger. Si potrebbe quindi lavorare con l'approccio top down di fatto ridefinendo le API, oppure, se le API sono scritte in un linguaggio supportato, generare la loro definizione Swagger automaticamente

Se si è un consumer di API, è possibile esplorarle usando la Swagger UI (applicazione web che permette appunto di visualizzare le informazioni sulle API), e usare quindi il Codegen per generare la libreria client per il linguaggio di programmazione desiderato.



Figura A.2: Swagger [35]

#### A.4.1 Componenti

Di seguito verranno introdotti i componenti di Swagger IO.

##### Swagger Editor

Swagger Editor è un'applicazione disponibile sia in versione web sia in versione desktop. È possibile definire delle API con uno specifico linguaggio nella parte sinistra dello schermo (YAML), mentre nella parte destra viene visualizzata una schermata con una descrizione grafica e testuale delle API che si stanno definendo. YAML è un linguaggio derivato da JSON che serve per semplificare drasticamente la sintassi. Si possono usare in ogni caso entrambi i linguaggi a seconda della preferenza del singolo. Di default viene usato YAML. Di seguito viene mostrato un semplice esempio, mostrando prima il codice YAML:

```
/store/animals:
  get:
    summary: Get all animals
    description: |
      All animals are returned
```

```
responses:
  200:
    description: Animals
    schema:
      type: array
      items:
        $ref: '#/definitions/Animal'
  default:
    description: Unexpected error
    schema:
      $ref: '#/definitions/Error'

definitions:
  Animal:
    type: object
    properties:
      ID:
        type: string
        description: Unique identifier of the animal
      name:
        type: string
        description: Name of the animal
      specie:
        type: string
        description: Specie of the animal
  Error:
    type: object
    properties:
      code:
        type: integer
      message:
        type: string
```

Ora invece in figura A.3 viene mostrata la documentazione generata:

The screenshot shows the Swagger Editor interface for a GET request to the endpoint `/store/animals`. The request is titled "GET /store/animals".

**Summary:** Get all animals.

**Description:** All animals are returned.

**Responses:**

Code	Description	Schema
<b>200</b>	Animals	<pre> [&gt; Animal {     ID: &gt; string     name: &gt; string     specie: &gt; string }]   </pre>
<b>default</b>	Unexpected error	<pre> [&gt; Error {     code: &gt; integer     message: &gt; string }]   </pre>

**Try this operation**

Figura A.3: Swagger Editor: documentazione generata

## Swagger Codegen

Swagger Codegen è un'applicazione disponibile sia in versione web sia in versione desktop. Permette di:

- Generare il codice dello scheletro del server partendo dalla definizione delle API in diversi linguaggi di programmazione (tra cui NodeJS, Asp.net, JAX-RS..)
- Generare una libreria client che si interfaccia con il server anche in questo caso scegliendo tra tanti linguaggi di programmazione e piattaforme (Javascript, Android, Java, C#..)

## Swagger UI

Applicazione web che permette di visualizzare in forma grafica e testuale le informazioni sulle API di certi servizi, sempre che ovviamente si possieda l'URL della definizione delle API. Si può anche interagire con le API per provarle. E' quindi disponibile lo Swagger Editor che permette di definire le API. Se l'organizzazione che produce le API lo desidera, può rendere disponibili a tutti le proprie API che quindi sono consultabili tramite la Swagger UI.

## A.5 Vert.x

Vert.x è un toolkit lato server costruito in Java che prende forte ispirazione da NodeJS e dal suo modello ad event loop. Di seguito vengono elencate le principali caratteristiche di Vert.x:

- *Poliglotta*: sono supportati diversi linguaggi di programmazione. Si possono quindi costruire server con Vert.x in svariati linguaggi di programmazione e non solo in Java
- *Semplice modello di concorrenza*: non è necessario che il programmatore si occupi degli aspetti legati al multi threading, poichè viene utilizzato il modello ad event loop. Se è necessario però avere a che fare con dei task bloccanti che richiedono l'utilizzo di thread, Vert.x mette a disposizione delle funzionalità per farlo in modo semplificato
- *Event bus*: è possibile fare comunicare più server tra loro grazie all'event bus, che può funzionare con meccanismo punto-punto o publish-subscribe. Grazie a questo è possibile anche ad esempio fare comunicare tra loro microservizi costruiti con tecnologie diverse

Un'applicazione Vert.x è composta da uno o più *Verticle*. Ogni verticle ha la propria istanza. Un sistema sarà quindi composto da una o più istanze Vert.x che al suo interno possiedono una o più istanze di verticle. L'istanza Vert.x è eseguita a sua volta nella sua istanza della Java Virtual Machine. I verticle come precedentemente detto possono comunicare tra loro se necessario utilizzando l'event bus.



Figura A.4: Vert.x [33]

### A.5.1 Event loop e confronto con Node JS

Prima di andare a vedere più nel dettaglio il modello event loop, è necessario andare ad introdurre anche il modello a cui si contrappone, ovvero quello del thread pool. Si potrebbe pensare infatti di creare un nuovo thread per gestire ogni richiesta che arriva ad un ipotetico web server. Questo approccio però ha delle debolezze, in particolare se i thread da creare sono tanti, dato che a livello di kernel il context switch per poter utilizzare numerosi thread ha un costo. Il modello event loop cerca di superare queste problematiche modellando tutto ad eventi ed utilizzando un singolo thread. Gli eventi sono sia la ricezione di

richieste, sia eventi generati dalle richieste stesse (ad esempio quando viene ricevuta una risposta da un database). Il pattern di riferimento è il cosiddetto *Reactor Pattern*. E' presente un thread chiamato appunto event loop che invoca gli event handler associati agli eventi. Più in dettaglio, quello che accade sono i seguenti passaggi:

1. L'applicazione genera una nuova operazione di IO inviando una richiesta ad un componente chiamato Event Demultiplexer (ED). Viene anche specificato un handler, che verrà invocato quando l'operazione viene completata. La richiesta da inviare al ED è non bloccante e ritorna immediatamente il controllo all'applicazione
2. Quando un'operazione di IO termina, l'ED inserisce un nuovo evento nella Event Queue
3. L'Event Loop itera sugli elementi della Event Queue
4. Viene invocato per ogni evento il corrispondente handler
5.
  - L'handler, non appena la sua esecuzione termina, restituisce il controllo all'Event Loop
  - Nel frattempo però, durante l'esecuzione dell'handler, nuovi eventi potrebbero essere stati inseriti nella Event Queue in caso in cui vengono richieste dall'handler stesso nuove operazioni di IO
6. Quando tutti gli elementi della Event Queue sono processati, il loop si blocca sul ED che quindi innesca un nuovo ciclo

Il modello a event loop si può dunque semplificare in questo modo:

```
loop{
    Event ev = evQueue.remove()
    Handler handler = selectHandler(ev)
    execute(handler)
}
```

Gli eventi vengono inseriti in una coda. Gli handler rappresentano la computazione da svolgere per i vari eventi. In questo modello è presente un singolo thread che ciclicamente preleva dalla coda un evento, seleziona l'handler relativo ad esso e lo esegue. Avendo un solo thread si evita quindi il costo legato al context switch e con un solo thread è possibile eseguire tantissime richieste al secondo senza problemi di concorrenza. Molto importante è prevedere handler non bloccanti. Se lo fossero, gli handler successivi non verrebbero eseguiti in tempi brevi (o non verrebbero eseguiti affatto, in caso di handler con dei loop).

infiniti). Se è necessario compiere operazioni di lunga durata che bloccherebbero l'event loop, conviene utilizzare un thread. Ora è opportuno andare a introdurre una differenza tra Vert.x e Node.js. In Node viene utilizzato appunto il reactor pattern. Tuttavia l'utilizzo di un singolo thread fa in modo che solo un processore venga sfruttato. Vert.x sfrutta come node.js lo stesso pattern, ma al posto di mantenere un solo event loop ne mantiene di più, di default pari al numero di processori disponibili. Questo permette di utilizzare tutti i processori in un solo processo, non dovendo quindi creare processi diversi per sfruttare tutti i processori, come avverrebbe su Node. Gli sviluppatori di Vert.x chiamano questo pattern Multi Reactor Pattern. Tuttavia, nonostante ci siano più di un event loop, un dato handler non verrà eseguito mai concorrentemente da diversi event loop, ma verrà eseguito sempre dallo stesso event loop. Tutto questo avviene poiché non appena viene lanciato un Verte, ad esso viene assegnato uno degli event loop disponibili. È quindi possibile suddividere la logica applicativa in diversi vertice in modo da sfruttare facilmente tutti i processori disponibili. In caso in cui i vertice condividono dati, è possibile utilizzare delle strutture dati apposite fornite dalle API di Vert.x per condividere dei dati senza dovere gestire manualmente aspetti di concorrenza.

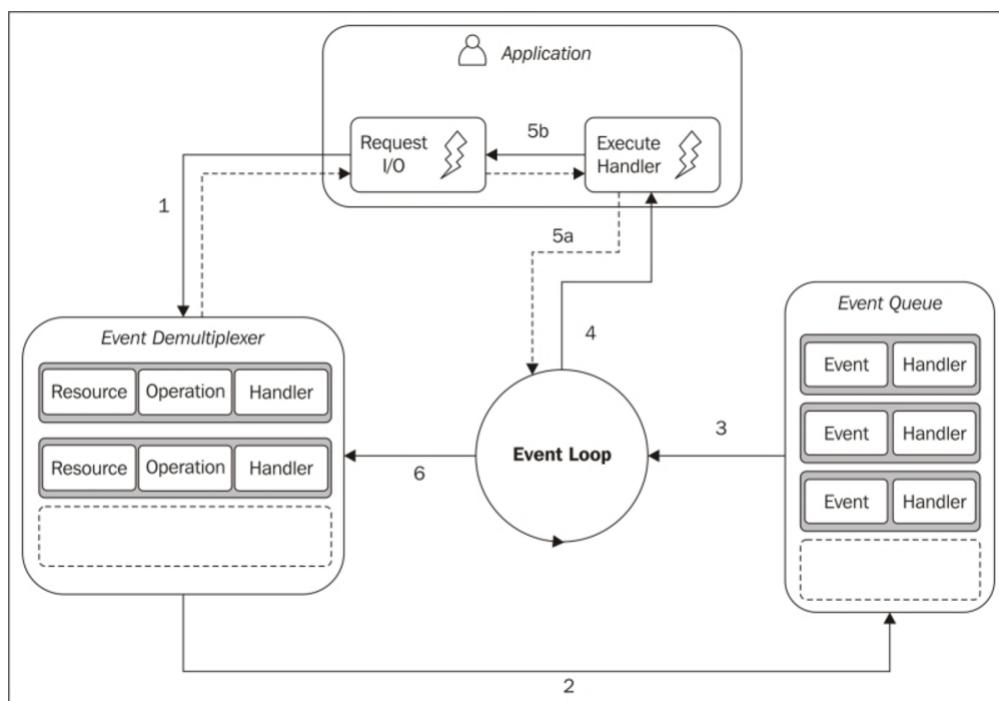


Figura A.5: Modello del reactor pattern [31]

### A.5.2 Architettura

Ora verrà introdotta l'architettura generale di Vert.x illustrando brevemente i suoi componenti:

- *Verticle Instance*: istanza di verticle. Come detto precedentemente, una istanza di Vert.x può contenere al suo interno una o più istanze di verticle
- *HazelCast*: è una In Memory Data Grid, ovvero una struttura dati che risiede totalmente in RAM, e che può essere anche distribuita tra più server. Vert.x utilizza internamente HazelCast per condividere i dati tra diverse istanze di Verticle o diverse istanze di Vert.x
- *Event Bus*: è l'event bus utilizzato per la comunicazione tra diverse istanze di Verticle e di Vert.x. Utilizza Hazelcast per il suo funzionamento interno
- *Shared Data*: come accennato precedentemente, tramite questo componente è possibile condividere dati tra diverse istanze di Verticle o anche diverse istanze di Vert.x
- *Shared Server*: oltre ai dati, le istanze di Verticle condividono anche lo stesso server
- *HTTP Server*: componente utilizzato per registrare gli handler relativi a eventi HTTP (come GET, POST, PUT, DELETE..). Supporta sia richieste HTTP sia il protocollo Websocket
- *Net Server*: componente utilizzato in particolare per gestire gli eventi dell'event bus e per la costruzione di server TCP
- *Event Loops*: gruppo di thread che eseguono l'event loop. Ne viene creato uno per ogni processore presente sulla macchina
- *Background Thread Pool*: gruppo di thread a disposizione in caso in cui in un handler da eseguire sia necessario appunto utilizzare un thread aggiuntivo, in caso l'operazione sia bloccante e di lunga durata. È possibile specificare il numero manualmente oppure utilizzare quello di default che è 20. È possibile prevedere che le operazioni eseguite tramite questi thread siano eseguite in modo sequenziale oppure in modo parallelo
- *Acceptor Thread Pool*: insieme di thread per accettare le socket. Ne viene creato uno per ogni porta. Necessario se si vuole creare un server TCP

- *NioWorker*: handler non bloccanti, che quindi possono essere eseguiti senza utilizzare thread dal Background Pool
- *Blocking job*: handler bloccanti, che quindi dovrebbero essere eseguiti utilizzando thread dal Background Pool
- *Worker mode*: sono istanze di Vertece che eseguono solamente task bloccanti. In questo modo verranno usati i thread del Background Pool. E' possibile creare Vertece che utilizzino un solo thread del Background Pool oppure creare Vertece che ne utilizzino più di uno per sfruttare maggiormente il multithreading (ma in questo modo è necessario gestire manualmente tutte le problematiche relative al multithreading, corse critiche e così via). E' quindi possibile progettare un vertice che esegua sempre task bloccanti utilizzando i thread del Background Pool, oppure utilizzare una soluzione ibrida servendosi dell'Event Loop e utilizzare i thread del Background Tool solo quando è necessario

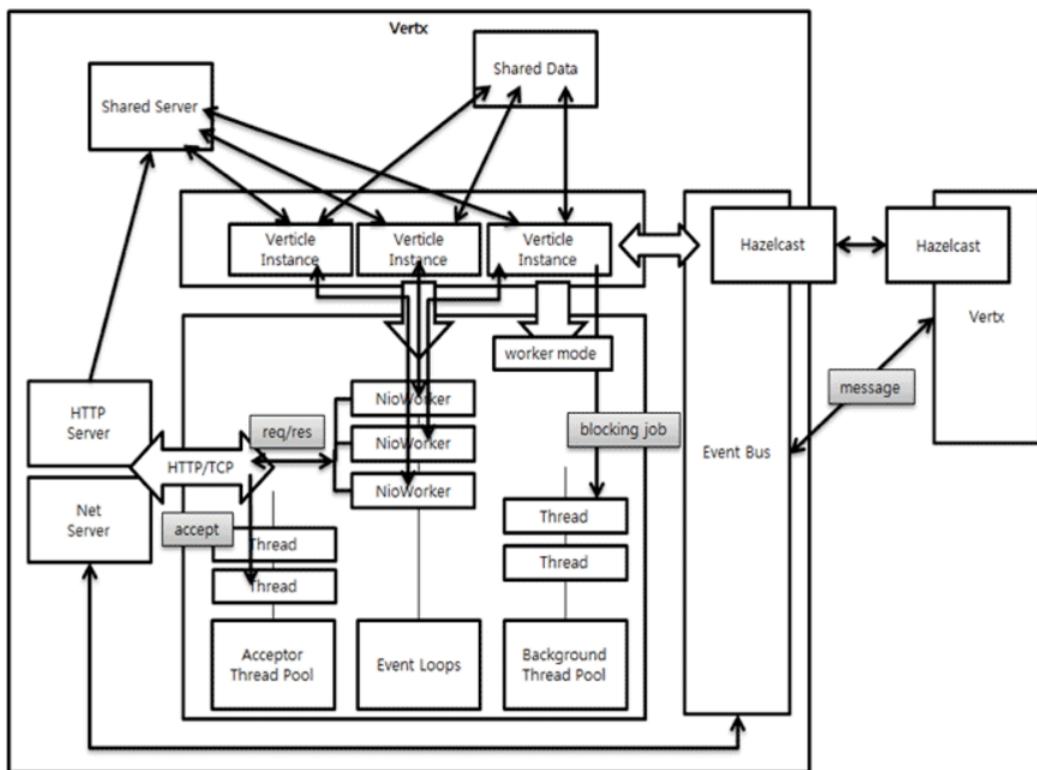


Figura A.6: Architettura componenti di Vert.x [30]

### A.5.3 REST con Vert.x

Grazie alla piattaforma web di Vert.x è possibile utilizzare l'architettura REST. Due concetti principali vanno illustrati:

- *Routes*: entità che permettono di specificare come le richieste sono suddivise tra i vari handler. Queste entità sono gestite dall'entità *Router*. E' quindi possibile definire dei path in modo tale da rispettare l'approccio REST nella definizione delle risorse. Le richieste, sulla base del path a cui sono rivolte, verranno passate all'handler specificato
- *Handlers*: entità che rappresentano le azioni da compiere per processare le richieste

E' quindi possibile definire delle API. Ad esempio tramite questo codice:

```
router.get("/hospital/patients").handler(this::getAll);
```

Si sta istruendo il router di gestire le richieste di tipo GET al path ”/hospital/-patients” creando un handler che esegue il metodo ”getAll”. Vert.x ovviamente mette a disposizione meccanismi per:

- Leggere header e body delle richieste in entrata
- Creare le risposte da inviare specificando i parametri dell'header ed il body
- Specificare lo status code HTTP della risposta da inviare

## A.6 NoSQL

I database di tipo relazionale memorizzano i dati in strutture fisse chiamate tabelle. Le tabelle sono legate tra di loro tramite relazioni. E' possibile eseguire delle operazioni di lettura e scrittura sulle tabelle tramite le transazioni. Le transazioni possiedono le proprietà ACID:

- *Atomicità*: la transazione è atomica, perciò la sua esecuzione non può essere separata in più parti
- *Consistenza*: l'esecuzione della transazione deve lasciare il database in uno stato consistente sia prima sia dopo la transazione. Nessun vincolo sui dati deve essere violato
- *Isolamento*: ogni transazione deve essere indipendente dalle altre, perciò se una di esse fallisce questo non deve interferire con il corretto funzionamento delle altre

- *Durabilità*: dopo l'esecuzione della transazione i dati devono essere memorizzati su un dispositivo di memoria

A questo approccio si contrappone l'approccio NoSQL, che promuove una maggiore flessibilità del modello dei dati rispetto alla rigidità del modello relazionale. I database di tipo NoSQL infatti sono strutturati in modo diverso. Di seguito vengono illustrate alcune caratteristiche:

- I dati memorizzati non hanno una struttura fissa, difatti possono essere composti da campi che variano da entità a entità
- Un'entità contiene tutte le informazioni relative all'oggetto che essa descrive
- Non valgono le proprietà ACID, bensì quelle BASE. Prima di introdurlle è necessario introdurre anche il teorema CAP di Eric Brewer, che sostiene che in un sistema distribuito solo due delle seguenti proprietà possono essere soddisfatte nello stesso momento:
  - *Consistency*: tutti i nodi hanno gli stessi dati nello stesso momento
  - *Availability*: ad ogni richiesta deve susseguirsi necessariamente una risposta
  - *Partition tolerance*: il sistema continua a funzionare nonostante fallimenti e perdite di messaggi

Per soddisfare le proprietà ACID è necessario puntare su consistenza e partition tolerance, a discapito della disponibilità. Di seguito invece le proprietà BASE, che puntano su disponibilità e partition tolerance, a discapito della consistenza:

- *Basic Availability*: la disponibilità di base è sempre garantita grazie a una forte replicazione dei dati tra i vari nodi
- *Soft state*: lo stato potrebbe non essere consistente
- *Eventual consistency*: dopo un'aggiornamento dei dati, è presente una finestra temporale in cui non è detto che i client che accedono ai suddetti dati vedano già il valore aggiornato. Questa forma di consistenza garantisce che se non sono stati fatti nuovi aggiornamenti ad un oggetto, gli accessi successivi ritorneranno l'ultimo valore aggiornato

Come si nota, si punta di più a prestazioni, scalabilità e disponibilità dei dati, a scapito della consistenza. Si ritiene infatti che in contesti come le applicazioni web sia più importante ottenere i dati il più velocemente possibile, anche se i dati non sono totalmente aggiornati

Entrambi gli approcci hanno vantaggi e svantaggi che vanno considerati prima di compiere una scelta. Di seguito vengono elencati alcuni vantaggi dell'approccio NoSQL:

- *Migliori prestazioni*: dato che le entry contengono tutte le informazioni relative all'oggetto descritto, non è necessario effettuare delle join per ottenerle (come invece accade per l'approccio relazionale). Più la mole di dati è grande più il vantaggio in prestazioni cresce
- *Dati non strutturati*: è possibile modellare con più facilità dati non strutturati, dato che è possibile inserire nel database entry strutturate in modi diversi. Per la stessa ragione è più facile gestire requisiti incerti e mutevoli
- *Scalabilità orizzontale*: se è necessario bilanciare il carico, in un sistema NoSQL è possibile scalare orizzontalmente (quindi suddividere i dati su più server). Con il modello relazionale invece è necessario scalare verticalmente, non potendo suddividere i dati su più server (è quindi necessario aumentare la potenza dell'unico server)

Di seguito vengono invece elencati alcuni svantaggi dell'approccio NoSQL:

- *Duplicazione dei dati*: proprio perchè le entry contengono tutte le informazioni dell'oggetto descritto, verrà effettuata una grande duplicazione di dati che porta ad un utilizzo di memoria maggiore. Se ad esempio dobbiamo modellare dei prodotti che fanno parte di una certa categoria, dovremmo indicare la categoria in ogni entry. Con l'approccio relazionale invece andrebbe definita la categoria una volta sola in una tabella a parte. Va però considerato che il costo della memoria cala sempre di più con il passare del tempo, quindi in molti casi questo svantaggio non incide più di tanto
- *Maggior numero di scritture*: per lo stesso motivo di prima, cresce anche il numero di scritture. Se per esempio si vuole aggiornare i dati relativi alla categoria, questo andrà fatto per ogni entry che contiene le informazioni della categoria, poichè le informazioni sono appunto ripetute. Con l'approccio relazionale sarebbe sufficiente modificare una volta le informazioni della categoria. Questo incide anche sulle prestazioni relative alle operazioni di update che risultano quindi più lente
- *Consistenza*: è più facile che si verifichino errori e problemi di consistenza proprio perchè è possibile inserire nelle collezioni documenti con strutture diverse. Nell'approccio relazionale invece si può definire strettamente il modello dei dati e questo porta a un maggiore controllo e rigidità

- *Query complesse*: se è necessario effettuare query molto complesse (in particolare innestate) il tutto risulta più complesso rispetto ad un linguaggio come SQL

### A.6.1 MongoDB

MongoDB è un DBMS open source di tipo NoSQL, e risulta attualmente il più utilizzato di questa categoria. Dopo la sua installazione, si può utilizzare tramite interfaccia a linea di comando, ma sono disponibili anche dei software di terze parti con interfaccia grafica che ne semplificano tantissimo l'utilizzo (come ad esempio RoboMongo). Di seguito vengono illustrate le caratteristiche principali di MongoDB:

- È orientato ai documenti. Il database è composto da collezioni che contengono dei documenti (insiemi di coppie chiave-valore). I documenti vengono memorizzati in linguaggio BSON (Binary JSON), ovvero un'estensione del linguaggio JSON. Rispetto a JSON, in cui vengono memorizzate coppie chiave-valore, in BSON viene memorizzato anche il tipo di dato
- La chiave primaria dei documenti è il campo `_id`. Questo campo viene generato automaticamente da MongoDB. Questo campo è di tipo `ObjectID` ed è costituito da 12 byte:
  - 4 byte per timestamp
  - 3 byte per l'ID della macchina
  - 2 byte per l'ID del processo di creazione del documento
  - 3 byte per il contatore incrementale
- Essendo NoSQL, i documenti possono avere struttura diversa tra loro, non devono seguire uno schema prefissato
- Sono supportate ovviamente le operazioni CRUD:
  - Create (insert)
  - Read (find)
  - Update (update)
  - Delete (remove)
- Supporto per operazioni di aggregazione (in modo simile alla GROUP BY di SQL) grazie all'Aggregation Framework. L'aggregazione viene modellata come una sequenza di operazioni sui dati. Le operazioni principali possono essere di tipo:

- *Filtro*: i dati vengono filtrati
- *Trasformazioni*: i dati vengono modificati

Oltre a questo si aggiunge la possibilità di ordinare i dati, raggrupparli e anche utilizzare degli operatori per effettuare calcoli (come ad esempio la media)

- Supporto ad operazioni di Map Reduce

## A.7 Angular

Angular è uno dei framework più utilizzati per lo sviluppo di applicazioni web client. È un framework single page, perciò il browser caricherà una pagina sola contenente tutta quanta l'applicazione. È totalmente open source ed attualmente è mantenuto da Google. I linguaggi di riferimento per lo sviluppo sono JavaScript o linguaggi come TypeScript che vengono compilati in codice JavaScript. Angular si basa sul MVC (Model View Controller). Un'applicazione Angular di base è composta da dei moduli, ovvero dei componenti che servono come contenitori di altri tipi di componenti. I componenti principali di un'app Angular sono i seguenti:

- *View*: rappresentano la vista dell'applicazione. Sono dei file scritti in linguaggio HTML. In questo caso però HTML viene esteso con dei nuovi attributi tramite delle *direttive*
  - Direttiva: Marcatori del DOM, che vengono usati dal compilatore Angular per iniettare delle specifiche funzionalità. Sono presenti numerose direttive di default ma se ne possono anche creare di personalizzate. Ad esempio è possibile utilizzare la direttiva ng-model, che consente di specificare una variabile a cui un certo componente HTML è collegato. Se tramite i controller il valore della variabile viene modificato, la view viene aggiornata rispecchiando il nuovo valore. La variabile creata farà parte del modello (chiamato precisamente *scope*), e può essere acceduta sia dalla vista sia dal controller. Esistono direttive anche per gestire gli eventi relativi ai vari componenti HTML (click, mouse..)
- *Controller*: rappresentano la logica applicativa. Servono per effettuare richieste a server, modificare i dati collegati alle view e così via. Sono dei file scritti in Javascript

- *Servizi*: oggetti che contengono funzioni che possono venire utilizzate dai controller. Esistono diversi servizi built-in (per interagire con server HTTP ad esempio) ma è possibile crearne di personalizzati

Per concludere, in figura A.7 è mostrato uno schema dell’architettura di un’applicazione angular, composta da:

- Root module: modulo principale dell’applicazione (ognuna deve averne almeno uno), che eventualmente può contenere altri moduli
- Config: componenti che servono per specificare i possibili stati dell’applicazione
- Routes: componenti che servono per collegare gli URL alle viste e ai controller
- View, con associate le direttive
- Controller, con eventualmente dei Services o Factories associate
- Scope: condiviso tra le viste e i controller

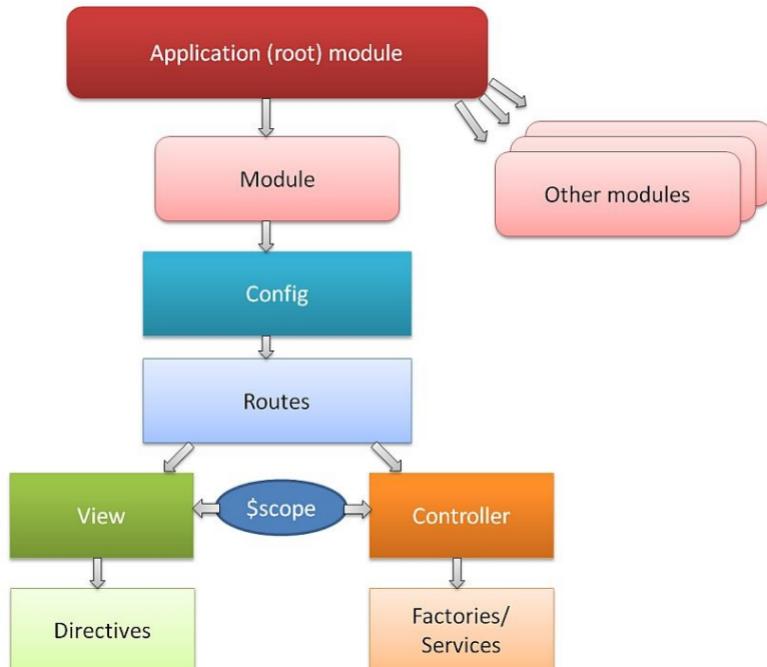


Figura A.7: Architettura di Angular JS [34]

## A.8 HAPI - HL7 Api

HAPI (HL7 Api) è una libreria Java open source. Il progetto non è affiliato con HL7, è semplicemente un software scritto rispettando le specifiche. Il progetto è stato iniziato dalla University Health Network (Toronto, Canada). Il sito di riferimento è il seguente: <http://hl7api.sourceforge.net/>. La libreria supporta tutte le versioni di HL7, e supporta sia il tipo di messaggi di HL7 2.x (ovvero il messaggio di tipo pipe) sia il tipo di messaggi di HL7 3 (formato XML).

### A.8.1 Funzionalità

Prima di utilizzarla nel progetto, è stato fatto qualche esperimento con la libreria seguendo il tutorial e la documentazione. Le funzionalità principali della libreria sono le seguenti:

- *Parsing di messaggi*: dato un messaggio, è possibile effettuarne il parsing per modellarlo ad oggetti ed accedere ai suoi valori
- *Creazione di messaggi*: E' possibile creare un messaggio direttamente sotto forma di stringa, ma anche crearlo mediante la programmazione a oggetti, e in seguito convertirlo a stringa
- *Operazioni di rete*: E' possibile creare server e client conformi da HL7. L'uso del multithreading per gestire le connessioni viene gestito internamente (anche se può venire modificato), e viene anche gestito in automatico l'invio del messaggio di ACK da parte del server alla ricezione di messaggi da parte del client
- *Handlers*: E' possibile definire alcuni tipi di handler per il server:
  - ConnectionListener: tipo di handler che si occupa di specificare il comportamento da tenere quando il server riceve nuove connessioni da parte dei client
  - ReceivingApplicationExceptionHandler: tipo di handler che si occupa di specificare il comportamento da tenere quando ci sono delle eccezioni nell'elaborazione dei messaggi
  - ReceivingApplication: tipo di handler che si occupa di specificare il comportamento da tenere quando si ricevono determinati tipi di messaggi. E' possibile quindi definire il tipo di risposta da inviare sulla base del tipo di messaggio ricevuto da parte del server

- *Crittografia*: E' possibile utilizzare la crittografia SSL/TLS quando si inviano dei messaggi
- *Character set*: E' possibile indicare quale character set dovrà essere utilizzato

### A.8.2 Alcuni esempi

Di seguito alcuni esempi di utilizzo della libreria:

- Parsing di un messaggio e stampa

```
//messaggio
String msg =
    "MSH|^~\&|HIS|RIH|EKG|EKG|199904140038||ADT^A01||P|2.2\r"+"PID|...";
//parsing del messaggio
HapiContext context = new DefaultHapiContext();
Parser p = context.getGenericParser();
Message hapiMsg=p.parse(msg);
ADT_A01 adtMsg = (ADT_A01)hapiMsg;
//recupero valori dal messaggio e stampa
MSH msh = adtMsg.getMSH();
String msgType =
    msh.getMessageType().getMessageType().getValue();
System.out.println(msgType);
PN patientName = adtMsg.getPID().getPatientName();
String familyName = patientName.getFamilyName().getValue();
System.out.println(familyName);
```

- Creazione di un messaggio e stampa

```
//creazione messaggio
ADT_A01 adt = new ADT_A01();
adt.initQuickstart("ADT", "A01", "P");
//popolamento dell'header
MSH mshSegment = adt.getMSH();
mshSegment.getSendingApplication().getNamespaceID().setValue("TestSendingSystem")
mshSegment.getSequenceNumber().setValue("123");
//popolare il PID
PID pid = adt.getPID();
pid.getPatientName(0).getFamilyName().getSurname().setValue("Doe");
pid.getPatientName(0).getGivenName().setValue("John");
pid.getPatientIdentifierList(0).getID().setValue("123456");
//popolare gli altri segmenti
```

```

.....
//encode del messaggio e stampa
HapiContext context = new DefaultHapiContext();
Parser parser = context.getPipeParser();
String encodedMessage = parser.encode(adt);
System.out.println(encodedMessage)

```

- Creazione e avvio server

```

int port = 1011;
boolean useTls = false;
HapiContext context = new DefaultHapiContext();
//creazione server
HL7Service server = context.newServer(port, useTls);
//per la ricezione di messaggi ADT
ReceivingApplication handler = new ExampleReceiverApplication();
server.registerApplication("ADT", "A01", handler);
server.registerApplication("ADT", "A02", handler);
//avvio server
server.startAndWait();

```

- Creazione client, invio e ricezione di messaggi

```

HapiContext context = new DefaultHapiContext();
String msg="..."; //messaggio ADT
Parser p = context.getPipeParser();
Message adt = p.parse(msg);
int port = 1011;
boolean useTls = false;
String ip="localhost";
//creazione client
Connection connection = context.newClient(ip, port, useTls);
//invio di messaggi unsolicited
Initiator initiator = connection.getInitiator();
Message response = initiator.sendAndReceive(adt);
//ricezione di risposte
String responseString = p.encode(response);
System.out.println("Received response:\n" + responseString);

```

- Semplice ReceivingApplication handler che si occupa di inviare un messaggio di tipo ACK quando viene ricevuto un messaggio

```

public class MyADTHandler implements ReceivingApplication {
    private DefaultHapiContext defaultHapiContext;

    @Override
    public boolean canProcess(Message arg0) {
        return true;
    }

    @Override
    public Message processMessage(Message theMessage,
        Map<String, Object> theMetadata) throws HL7Exception {
        defaultHapiContext = new DefaultHapiContext();
        String encodedMessage =
            defaultHapiContext.getPipeParser().encode(theMessage);
        //stampa del messaggio ricevuto
        System.out.println(encodedMessage);
        //generazione e invio di un messaggio di ACK
        try {
            return theMessage.generateACK();
        } catch (IOException e) {
            throw new HL7Exception(e);
        }
    }
}

```

## A.9 JMeter

Apache JMeter è un applicazione open source scritta in Java che è rivolta al testing delle applicazioni, in particolare per quanto riguarda la misurazione delle performance e i test di carico. E' possibile mettere sotto stress le applicazioni con carichi molto pesanti e verificare quindi se esse sono in grado di sopportarli. Supporta diversi tipi di protocolli, tipi di server e tipi di applicazioni, tra cui:

- Server web di tipo REST e SOAP
- TCP
- Protocollo FTP
- Database

- Mail (SMTP, POP3 e IMAP)

- Tanto altro

Dispone sia di un’interfaccia a linea di comando sia di un’interfaccia grafica. Tutti i test svolti possono essere salvati e quindi riconsultati più avanti nel tempo. Questo software è stato utilizzato per testare i tempi di latenza relativi al servizio sviluppato (quindi un server web di tipo REST). I parametri più rilevanti che possono essere impostati: sono:

- *Number of Threads (users)*: sono il numero di utenti che si vuole simulare. Ogni utente avrà il suo thread con cui effettuerà le richieste al web server
- *Ramp-Up period*: tempo che è necessario per inizializzare tutti gli utenti. Se per esempio si sceglie di avere 100 utenti e un tempo di ramp up di 10 secondi, allora ogni utente sarà inizializzato in  $10/100=0.1$  secondi
- *Loop Count*: numero di volte che si vuole che la richiesta al server venga ripetuta (per ogni utente). Si può impostare anche che le richieste vengano effettuate all’infinito

E’ possibile quindi aggiungere al test delle richieste HTTP in cui è possibile specificare URL, numero di porta, tipo di richiesta (GET, POST...), eventuale body e tanto altro. I risultati del test possono essere visualizzati in diverse modalità, tra cui una delle più complete è quella ad albero. Oltre ad albero, si possono visualizzare i dati in tabella e anche in dei grafici. E’ possibile anche specificare un determinato numero di richieste da effettuare in un secondo tramite il *Constant Throughput Timer*. Come si può vedere in figura A.8, durante l’esecuzione del test tutte le richieste man mano che vengono completate vengono inserite nella lista (in questo caso si usa la visualizzazione ad albero) ed è quindi possibile visualizzare ad esempio il tempo in cui la richiesta è stata processata, l’eventuale body della risposta ricevuta e tanti altri parametri.

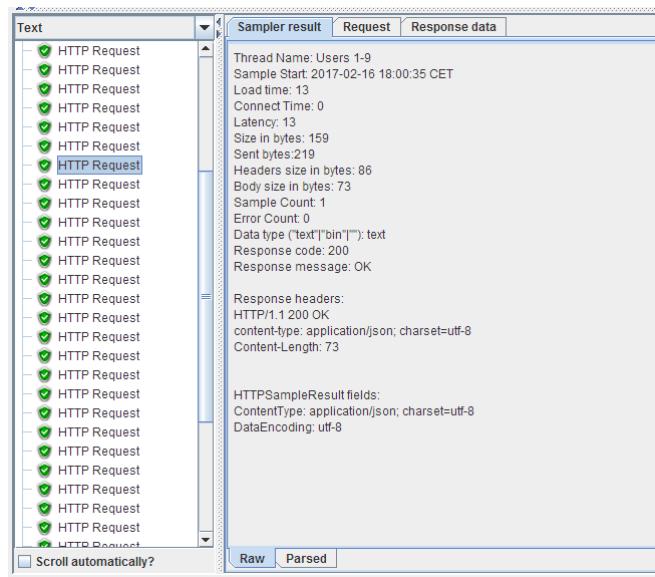


Figura A.8: Interfaccia grafica di JMeter

# **Ringraziamenti**

Al termine di questo percorso, come circa due anni fa, è doveroso ringraziare chi mi ha sempre supportato e sostenuto. In particolare la mia famiglia, che oltre ad avermi fornito i mezzi economici per arrivare fino a questo punto, ha sempre creduto in me e mi ha sempre incoraggiato in ogni momento, sopportandomi nei momenti più duri. La mia ragazza Giorgia, che ormai è tanti anni che mi sopporta e supporta condividendo la sua vita con la mia. I miei amici, che anche se non vedo tanto come una volta non si sono ancora dimenticati di me. I miei amici di nuoto, anche se ridono di me quando uso la tavoletta. I miei amici e compagni di università, vecchi e nuovi, che hanno rese più corte le ore di lezione e i viaggi in treno.



# Bibliografia

- [1] Hovenga, E.J.S., *Health Informatics: An Overview*, IOS Press, 2010.
- [2] Eysenbach G, *What is e-health?*, J Med Internet Res, 2001.
- [3] HL7 <http://www.hl7.org/>
- [4] IHE <https://www.ihe.net/>
- [5] CENSIS – ImpresaLavoro, *Le condizioni per lo sviluppo della Sanità Digitale: scenari Italia-UE a confronto*, 2016.
- [6] Introduction to HL7 Flash Tour [http://www.hl7.org/documentcenter/public\\_temp\\_4622F577-1C23-BA17-0C0F9B33E57FCC93/training/IntroToHL7/player.html](http://www.hl7.org/documentcenter/public_temp_4622F577-1C23-BA17-0C0F9B33E57FCC93/training/IntroToHL7/player.html)
- [7] Alessandro Ricci, *SERVICES AND SERVICE-ORIENTED ARCHITECTURES*, a.a. 2016/2017
- [8] Wided Guedria, Elyes Lamine, Herve Pingaud. HEALTH SYSTEMS INTEROPERABILITY: ANALYSIS AND COMPARISON . MOSIM 2014, 10eme Conference Francophone de Modelisation, Optimisation et Simulation, Nov 2014, Nancy, France.
- [9] D. Gubiani, *Introduzione ai Sistemi Informativi e alle Basi di Dati in Ambiente Medico*, 2011.
- [10] M. Mustra, K. Delac, M. Grgic, *Overview of the DICOM standard*, The 50th International Symposium, Zadar, pp. 39-44, 2008.
- [11] OpenEHR [http://www.openehr.org/what\\_is\\_openehr](http://www.openehr.org/what_is_openehr)
- [12] Jason H. Christensen, *Using RESTful web-services and cloud computing to create next generation mobile applications*, ACM, 2009.
- [13] Ian McNicoll, *openEHR in real world settings*, 2008.

- [14] Grant M. Wood, *HL7 Basic Overview*, HIMSS Las Vegas, 2012 .
- [15] SearchHealthIT <http://searchhealthit.techtarget.com/>
- [16] Health Information and Quality Authority, *Overview of Healthcare Interoperability Standards*, 2013.
- [17] HL7 INTRODUCTION, MESSAGING & THEORY [https://www.otechimg.com/\\_getDoc.cfm?id=74&started=1](https://www.otechimg.com/_getDoc.cfm?id=74&started=1)
- [18] Mare Cannataro, *Clinical Document Architecture (CDA)*, 2013/2014.
- [19] Ann Wrightson, *HL7 Clinical Document Architecture*.
- [20] Alessandro Ricci, Angelo Croatti, Sara Montagna, Vanni Agnoletti *Tracking and Assisting Activities in Trauma Management: The TraumaTracker case*, Submitted to 11th EAI International Conference on Pervasive Computing Technologies for Healthcare May 2017, Barcelona, Spain.
- [21] Bardram, Jakob Eyvind and others, *Pervasive healthcare as a scientific discipline*, Methods of information in medicine, Stuttgartm, vol 47, num 3, 178-185, 2008.
- [22] George W. Beeler Jr, *Introduction to: HL7 Reference Information Model (RIM)*, 2011.
- [23] Corepoint Health, *The Continuity of Care Document*, 2009.
- [24] Arbortext, *An Introduction to Structured Product Labeling*.
- [25] CCOW Technical Committee, *HL7 Context Management CCOW Standard: Best Practices and Common Mistakes*, 2006.
- [26] Manuale Infinity Gateway
- [27] Corepoint Health, *The HL7 Evolution*, 2009.
- [28] HAPI - The Open Source HL7 API for Java <http://hl7api.sourceforge.net/>
- [29] Woo Seongmin, *Inside Vert.x. Comparison with Node.js*, 2013.
- [30] Jaehong Kim, *Understanding Vert.x Architecture - Part II*, 2013.
- [31] The reactor pattern <https://www.packtpub.com/mapt/book/web-development/9781783287314/1/ch01lvl1sec09/the-reactor-pattern>

- [32] Jesper Nordström, Tomas Betzholtz, *Architecting for speed: How agile innovators accelerate growth through microservices*, 2016.
- [33] Vert.x <http://vertx.io/>
- [34] Angularjs architecture overview, <http://tutorialspointexamples.com/angularjs-architecture-overview-core-concepts-advantages-disadvantages/>
- [35] Swagger IO <http://swagger.io/>
- [36] MongoDB <https://www.mongodb.com/it>
- [37] JSON <http://www.json.org/>
- [38] NoemaLife, <http://www.noemalife.com/index.php>.
- [39] Onit, <https://www.onit.it/sanita>.
- [40] Log80, <http://www.log80.it/divisione-healthcare/#section-Oncologia>.