ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria e Architettura

Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

MOBILE CSCW E SINCRONIZZAZIONE DATI PER IL SOCCORSO IN EMERGENZA: UN CASO DI STUDIO

Tesi in

INGEGNERIA DEI SISTEMI SOFTWARE ADATTATIVI COMPLESSI

Relatore
Prof. MIRKO VIROLI

Correlatore
Ing. ANGELO CROATTI

Presentata da
SIMONE GROTTI

Sessione III
Anno Accademico 2014-2015

Parole Chiave

CSCW
Healthcare
Mobile Pervasive Computing
Sincronizzazione dati
REST
CouchDB
Couchbase Lite

Alla mia famiglia
e a Licia

Indice

Introduzione

L'avanzare delle nuove tecnologie ha portato grandi cambiamenti nella vita delle persone, in cui il supporto di quelle afferenti alla sfera dell'informatica sta diventando sempre più fondamentale, soprattutto nei riguardi dell'ambito lavorativo, dove esse sono state spesso applicate con successo.

Un sistema informatico che supporti la collaborazione e la cooperazione tra i componenti di un team può risultare fondamentale nel migliorare l'efficienza delle operazioni e quindi aiutare nel raggiungimento dello scopo comune: a maggior ragione questo supporto risulta necessario in situazioni di soccorso ai feriti in emergenza, ad esempio a seguito di catastrofi naturali o scenari di guerra, in cui la coordinazione delle azioni del gruppo tramite la conoscenza dello stato dei colleghi e della missione può, in generale, dare un grande contributo nel raggiungimento di un risultato migliore.

In tali situazioni la condivisione agevole di informazioni riguardanti il campo d'azione e i feriti ricopre infatti un ruolo fondamentale e può essere di vitale importanza per salvare il maggior numero di vite umane, ma supportarla risulta essere complesso, soprattutto in scenari in cui gli operatori sono geograficamente distribuiti e i dispositivi in loro possesso possono da un momento all'altro disconnettersi dalla rete. Dato che la disponibilità del sistema deve essere garantita, seppur ridotta, anche a fronte di eventuale mancanza di connettività, le informazioni devono persistere sui dispositivi degli operatori, facendo così emergere la problematica della sincronizzazione dati.

É in questo contesto che si colloca il lavoro svolto per questa tesi, il cui obiettivo principale è quello di approfondire il tema della comunicazione e sincronizzazione dati calati in un caso di studio per sistemi informatici di supporto al lavoro cooperativo (CSCW) per il soccorso in situazioni di emergenza, tratto da un più ampio progetto di ricerca svolto in precedenza dal *DISI*, soprattutto al fine di studiarne la fattibilità.

Partendo dall'approfondimento delle tematiche generali che toccano da vicino la natura del sistema descritto dal caso di studio, si passa a una sintetica ricognizione dello stato dell'arte per individuare le principali caratteristiche

comuni ai sistemi che si prefiggono di supportare la cooperazione in situazioni di emergenza, in modo tale da ottenere una maggiore comprensione dell'argomento. In seguito il caso di studio viene introdotto e analizzato, in modo da poterne poi estrarre, nell'ambito di interesse considerato costituto dalla sincronizzazione dati, i requisiti e le caratteristiche che dovrebbe possedere un *middleware* che ne supporti lo sviluppo fornendo una piattaforma per la condivisione agevole delle informazioni.

É a partire dalle considerazioni derivanti dall'analisi che viene poi condotta un'esplorazione del panorama tecnologico, centrale per questa tesi, volta all'individuazione di uno strumento le cui funzionalità siano abilitanti in maniera significativa per la problematica considerata, innalzando il livello di astrazione cui lo sviluppatore può far riferimento: in particolare le aree tematiche di interesse sono state rappresentate da framework per la realizzazione di servizi web, message oriented middleware e database distribuiti.

A conclusione del lavoro vengono così presentate le valutazioni emerse dall'esplorazione, dalle quali *CouchDB* è risultato come strumento che più degli altri si addice alle caratteristiche del problema, e per cui quindi è stata effettuata una breve sperimentazione.

Sommario

La tesi si articola in sei differenti capitoli ognuno dei quali ha un ben preciso ruolo all'interno del percorso di approfondimento svolto e il cui contenuto viene qui di seguito riassunto in maniera sintetica.

Capitolo 1 Nel primo capitolo vengono approfondite in maniera sintetica le principali aree di riferimento per un sistema di supporto al soccorso cooperativo in scenari di emergenza quali, ad esempio, catastrofi naturali o guerre: più di preciso sono state trattate le aree che fanno riferimento ai sistemi informatici a supporto del lavoro cooperativo (**CSCW**), al ruolo e all'impatto dell'informatica in ambito **healthcare** ed infine al **Pervasive Computing**, contestualizzato in particolare per l'utilizzo di dispositivi mobile. Per ognuna di queste aree viene fornita una essenziale descrizione volta ad identificarne le principali caratteristiche in maniera poi da sfruttare tale conoscenza per una maggiore comprensione del caso di studio di riferimento.

Capitolo 2 Il secondo capitolo si occupa di descrivere una ricognizione dello **stato dell'arte** volta a comprendere i principali risultati raggiunti nella realizzazione di sistemi dedicati alla gestione delle emergenze, in particolare in ambito sanitario: a partire dagli esempi analizzati vengono tratte le idee e gli approcci che sono stati utilizzati in modo da avere una buona base di conoscenza sulle soluzioni già adottate.

Capitolo 3 Nel terzo capitolo il **caso di studio** di riferimento per il lavoro svolto viene presentato in dettaglio: dopo una prima descrizione generale, i **requisiti** vengono formalizzati e ne viene svolta un'approfondita analisi, volta ad evidenziare il dominio e le funzionalità desiderate. In particolare viene poi approfondita l'esigenza di poter permettere agli operatori di sincronizzare i dati con un centrale operativa e poterne usufruire anche in assenza di rete: è proprio questa tematica che costituisce il nucleo fondamentale su cui il lavoro svolto nella restante parte si concentra.

Capitolo 4 Il quarto capitolo analizza infatti in maniera più profonda il problema della sincronizzazione dati nel contesto del caso di studio, evidenziando la necessità di un livello infrastrutturale che permetta di non gestirlo a livello applicativo: è a partire da queste considerazioni che vengono poi individuate le aree tecnologiche di potenziale utilità studiate nel seguito.

Capitolo 5 Il capitolo cinque si concentra così sull'analisi e l'approfondimento di alcune delle **tecnologie abilitanti** per la realizzazione del livello infrastrutturale di gestione della sincronizzazione a partire dalle considerazioni espresse nelle precedenti fasi di analisi. Partendo da una descrizione più dettagliata della problematica generale della sincronizzazione dati, vengono così studiate le aree afferenti a database distribuiti, MOM (Message Oriented Middleware) e servizi web RESTful, delle quali per ognuna viene data una breve *overview* delle caratteristiche principali e dei maggiori vantaggi che può portare, poi calata in un particolare strumento che la rappresenti nell'attuale panorama tecnologico, in modo da comprenderne la possibile utilità in relazione alle particolari caratteristiche del caso specifico di collaborazione *remote mobile* considerato.

Capitolo 6 Nel capitolo sei vengono stilate le **valutazioni** e le considerazioni emerse dall'esplorazione, descrivendo le possibilità offerte dai diversi approcci analizzati: dopo un loro confronto ed aver individuato in *CouchDB*, un database distribuito, lo strumento più idoneo, viene così presentata la breve fase di sperimentazione condotta per saggiarne in maniera più concreta le potenzialità.

Capitolo 1

Background

Un sistema informatico per il supporto alla gestione di situazioni di emergenza, in particolare riguardante il soccorso ai feriti, presenta molteplici caratteristiche e peculiarità che fanno riferimento a diverse aree dell'informatica: questo capitolo ha lo scopo di delineare ed approfondire questi temi dandone un panoramica e, allo stesso tempo, evidenziarne i tratti e le proprietà principali, al fine di una migliore comprensione degli ambiti a cui il particolare sistema, analizzato come caso di studio nel seguito della trattazione, fa riferimento.

Così, partendo dalla definizione generale di sistemi informatici a supporto del lavoro cooperativo e dall'interazione tra essere umano e questi strumenti, l'approfondimento mostra come nel tempo anche essi si siano evoluti per sfruttare il supporto dato dai dispositivi mobile; il quadro di riferimento viene poi completato dalla descrizione del ruolo delle tecnologie informatiche in rapporto ai sistemi per la salvaguardia della salute delle persone (*healthcare*) nel particolare contesto dei sistemi pervasivi e mobile, sui quali infine ci si concentra.

1.1 Computer Supported Cooperative Work

L'utilizzo delle tecnologie informatiche ha portato grandi cambiamenti nella vita delle persone, tanto nelle azioni quotidiane quanto in ambito lavorativo: con il termine **Computer Supported Cooperative Work** si intende infatti il campo di ricerca multidisciplinare il cui scopo è studiare il ruolo della tecnologia, in particolare informatica, nel supporto al lavoro cooperativo fra le persone. Utilizzato per la prima volta a metà degli anni '80 da Irene Greif e Paul Cashman per organizzare un workshop in questo campo di ricerca, che iniziava a riscuotere maggior successo anche a causa di una

maggiore diffusione dei personal computer negli uffici risalente a quel periodo, questo termine è stato coniato in ambito accademico e va distinto dal più pragmatico termine **Groupware**, ad esso poi successivamente associato. Se da una parte infatti con CSCW si fa riferimento all'intero ambito di ricerca, che coinvolge differenti discipline quali sociologia, economia, psicologia, scienze gestionali e organizzative e antropologia, cercando di adottare una visione più concettuale e completa di come più persone possano lavorare e collaborare tramite l'utilizzo di strumenti informatici, col termine Groupware si fa prettamente riferimento alle tecnologie (ad esempio software) utilizzate per supportare tale cooperazione ed in questo modo facilitarla e renderla più efficace. In particolare con Groupware si concentra solitamente l'attenzione su quei sistemi basati sull'utilizzo di tecnologie informatiche che offrono supporto al lavoro di gruppi di persone impegnate in un compito o verso un fine comune.[1]

1.1.1 Definizione

Anche se tuttora, dopo tanti anni di intensa ricerca in questa campo, non esiste ancora una definizione generalmente accettata come valida, ai fini di fare ulteriore chiarezza è utile riportare la definizione fornita in [2] per CSCW:

> *CSCW è un termine generico che combina la comprensione del modo in cui le persone lavorano in gruppo attraverso le reti di computer e le tecnologie ad esse associate, comprendenti hardware, software, servizi e tecniche.*

Risulta quindi chiaro come con questo termine si vada oltre al solo utilizzo della tecnologia per mediare questa cooperazione, spostando di fatto il focus su come le persone lavorano in gruppo, sulla loro consapevolezza che il fattore sociale risulta determinante e sull'impatto che le tecnologie informatiche possono avere nel migliorare la cooperazione e la coordinazione: infatti è importante sottolineare che non solo questi sistemi possono abilitare e semplificare la gestione del lavoro cooperativo, ma possono anche potenzialmente avere un impatto sui processi stessi, modificandoli e migliorandoli per ottenere un maggiore grado di cooperazione. É inoltre la natura multidisciplinare di questo ambito, che fin dall'inizio ha riunito ricercatori provenienti da aree anche molto differenti tra loro, ad evidenziare il fatto che esso vuole includere sia lo studio del lavoro cooperativo e dei processi che ne fanno parte sia quello di strumenti informatici e tecnologie a suo supporto: fin dall'inizio infatti sono stati apportati diversi contributi dalle differenti prospettive assunte a seconda della particolare disciplina considerata in modo tale da riuscire a

sfruttare queste conoscenze per migliorare la cooperazione e la coordinazione all'interno del lavoro.

1.1.2 Elementi caratterizzanti

Entrambi i principali aspetti a cui la definizione data fa riferimento sono analizzati in [3], in cui si sottolinea l'importanza di considerarli approfonditamente quando si tratta l'argomento, in modo da avere una visione più completa della problematica in cui si inseriscono ed effettuare così scelte progettuali più mirate ed efficienti.

Dal punto di vista facente riferimento alla *computer science*, CSCW viene concepito come un tentativo di comprendere la natura, le caratteristiche e i requisiti fondamentali dietro il lavoro cooperativo con lo scopo di progettare sistemi e tecnologie informatiche a suo supporto; proprio il fatto che più individui, situati in differenti ambienti di lavoro e con diverse responsabilità e prospettive, debbano interagire e cooperare per portare a termine il proprio lavoro, risultando così essere interdipendenti gli uni dagli altri, ha un grande impatto nel *design* di questo tipo sistemi. Risulta quindi evidente che lo scopo di questo ambito di ricerca sia quello di indagare il tema della cooperazione nel lavoro umano, adottando però una prospettiva volta al design: cercando di rispondere a quesiti riguardanti le caratteristiche o i requisiti generali per il supporto a tale particolare tipo di lavoro che lo differenziano da quello individuale, oppure di come questo processo di coordinazione possa essere reso più snello, efficiente e flessibile, si prova a comprendere come le tecnologie informatiche possano fornire supporto in modo da individuare le architetture che scaturiscono da queste considerazioni.

L'altra prospettiva rilevante da cui considerare questo ambito è quella relativa al lavoro cooperativo (*cooperative work*): non è facile trovare una definizione univoca per questo tipo di lavoro, in quanto nel corso degli anni ne sono state presentate differenti anche in ambiti molto diversi da quello relativo alle scienze informatiche (ad esempio sociologia e filosofia), ma anche se è stato in passato affermato che *"ogni tipo di lavoro è organizzato in maniera sociale"*, e quindi intrinsecamente comprende un parte di cooperazione, è utile caratterizzare più precisamente le diverse forme di attività lavorativa, in modo da comprendere gli aspetti principali che differenziano la cooperazione dalle altre. Centrale per questa categoria di lavoro risulta essere *l'interdipendenza*: si considerano infatti coinvolte in attività di questo tipo le persone che, per svolgere il proprio lavoro, sono mutualmente dipendenti le une dalle attività delle altre, non riferendosi solamente alla condivisione di risorse, nel qual caso le persone devono certamente coordinare le loro attività, ma non in modo completamente integrato, percependo la presenza degli altri quasi co-

me un fastidio e ritenendo che meno il proprio lavoro sia dipendente da quello degli altri, meglio sia. Essere mutualmente dipendenti significa invece affidarsi alla qualità (ad esempio la puntualità) del lavoro effettuato dagli altri nello svolgimento del proprio: è a causa di questa forma di interdipendenza che la cooperazione richiede necessariamente sforzi e attività aggiuntive volte alla mediazione e al controllo di queste attività coordinative, come ad esempio l'allocazione dei compiti tramite alcuni criteri secondo cui devono essere svolti. Fra queste attività risulta essere di fondamentale importanza l'*articolazione* delle attività distribuite individuali all'interno del gruppo nel senso di suddividerle, allocarle, schedularle, comprenderne le inter-relazioni e quindi coordinarle: è in questo senso che i lavoratori vengono coinvolti in attività non direttamente riconducibili alla realizzazione del prodotto o del servizio oggetto del proprio lavoro; si può quindi notare che, rispetto al lavoro individuale, quello cooperativo comporti un ovvio *overhead* in termini di costi (ad esempio dal punto di vista del tempo), il cui prezzo da pagare trova giustificazione nel fatto che, individualmente, i singoli lavoratori non sarebbero riusciti a raggiungere lo scopo che invece si prepone il lavoro di gruppo, motivazione per la quale è emersa la necessità del lavoro cooperativo.

Si può quindi notare che l'attività lavorativa non è intrinsecamente cooperativa, nel senso che non necessariamente i lavoratori sono interdipendenti gli uni dagli altri nello svolgere il proprio lavoro: è in questa prospettiva che si riesce a comprendere la distinzione tra lavoro cooperativo e individuale, che si differenziano proprio per la presenza, nel primo caso, di attività volte all'articolazione delle mansioni lavorative individuali e che non contribuiscono direttamente al fine ultimo del processo, quale ad esempio la realizzazione di un prodotto. Seguendo questa concezione, si desume come le attività volte all'articolazione del lavoro siano di rilevante importanza per il lavoro cooperativo, in quanto la sua intrinseca distribuzione fra i vari componenti del gruppo deve essere in qualche maniera gestita per raggiungere l'obiettivo attraverso la collaborazione. In effetti un'organizzazione cooperativa del lavoro emerge in situazioni che presentano particolari requisiti quali, ad esempio:

- aumentare le capacità informative dei componenti del gruppo in modo da abilitarne la collaborazione per raggiungere uno scopo altrimenti irraggiungibile individualmente;

- combinare le attività specializzate di differenti tipi di lavoratori che si occupano di particolari strumenti, tecniche o routine di lavoro;

- facilitare l'attività di problem solving combinando le differenti capacità ed inclinazioni dei componenti del gruppo, in modo tale da sfruttare an-

che molteplici punti di vista e prospettive sul problema per affrontarne le eventuali diverse nature;

- regolare lo spazio di interazione fra i componenti del gruppo in modo da agevolarne i processi e le routine di collaborazione, ad esempio aumentando il loro livello di comunicazione.

Molte delle attività affluenti a quest'area di ricerca fanno quindi riferimento alla *mediazione* dell'interazione umana in ambito lavorativo tramite l'utilizzo delle tecnologie informatiche come concetto di primaria importanza e in questo senso si può notare la relazione tra CSCW e la più ampia tematica di **HCI** (*Human Computer Interaction*), che studia l'interazione fra uomo e macchina (in particolare dispositivi informatici) al fine di costruire sistemi sempre più affidabili e di facile utilizzo per le persone in modo da supportarne agevolmente le attività. É proprio questo il punto di contatto fra i due ambiti, in quanto una delle prerogative dei sistemi CSCW è proprio quella di agevolare la cooperazione in ambito lavorativo tramite l'utilizzo di tecnologie afferenti all'informatica: per questo motivo essi devono presentare interfacce di utilizzo semplici ed intuitive per l'utilizzatore in modo da renderne facile l'utilizzo e la diffusione all'interno del gruppo di lavoro. Se infatti una particolare tecnologia non soddisfa questi requisiti, molto probabilmente non verrà adottata correttamente da parte del gruppo di lavoro, col risultato di non ottenere i benefici sperati nell'ambito della cooperazione: per essere realmente utili infatti questi sistemi devono essere utilizzati da un'alta percentuale dei componenti del gruppo, chiamata anche *massa critica*.

Altri elementi fondamentali da citare per questo ambito sono certamente la consapevolezza (*awareness*) rispetto agli altri componenti del gruppo e le loro attività che i lavoratori devono poter sfruttare grazie all'utilizzo di questi sistemi, e l'appropriazione di una certa tecnologia da parte di un gruppo che può, successivamente, utilizzarla in modo differente da quanto inizialmente progettato.

1.1.3 Classificazione

I sistemi CSCW, ed in particolare quelli riferiti come Groupware, presentano svariate e particolari caratteristiche sulla base delle quali nel tempo sono state proposte diverse forme di classificazione, che aiutano ad inquadrare e comprendere maggiormente la tematica, in modo da riflettere poi tali conoscenze sull'analisi e la progettazione di questi sistemi.

Una prima importante classificazione per il modello di questo tipo di sistemi riguarda le macro-aree di comportamenti a cui essi fanno riferimento

in modo da supportare l'attività di gruppo, così come viene descritto in [1]. Solitamente ci si riferisce a tale modello con il nome di *"modello delle 3 C"*, in quanto le aree individuate sono:

- **Comunicazione**, in quanto permettere agli utenti di comunicare fra di loro in svariate forme risulta essere fondamentale per raggiungere un buon livello di cooperazione;
- **Collaborazione**, in cui risulta essere di fondamentale importanza la *condivisione* delle informazioni, ad esempio permettendo ai componenti del gruppo di avere un ambiente condiviso di lavoro che offre in modo trasparente la visione del contesto del gruppo e delle attività eseguite dagli altri componenti;
- **Coordinazione**, a cui si fa riferimento con tutte quelle attività non direttamente correlate allo scopo finale ma necessarie quando si svolge un'attività distribuita in gruppo, come ad esempio l'articolazione del lavoro.

Un altro tipo di classificazione che si trova in molta della letteratura scientifica riguardante l'argomento è quella che fa riferimento a due delle dimensioni principali del tipo di collaborazione supportata: quella *spaziale* e quella *temporale*. Questa schematizzazione si presta bene ad essere rappresentata da una matrice, come quella in figura 1.1, che mostra i tipi di sistemi che ricadono in ciascuna delle aree di intersezione delle dimensioni. Secondo questa classificazione infatti dal punto di vista temporale le interazioni possono essere di tipo *sincrono* o *asincrono*, avvenendo fra i vari componenti del gruppo rispettivamente nello stesso momento o in momenti differenti e richiedendo (o meno) la loro contemporanea presenza, mentre dal punto di vista spaziale esse si distinguono in *co-ubicate* o *distribuite* a seconda che i vari collaboratori si trovino nello stesso posto o in luoghi differenti. A seconda dell'intersezione fra queste caratteristiche, i sistemi risultano essere divisi in:

- **sincroni co-ubicati**, che richiedono la simultanea presenza dei componenti del gruppo nello stesso posto, supportando interazioni dirette fra di loro, come ad esempio nel supporto alla generazione di idee (sessioni di brainstorming) oppure di aiuto ad una maggiore comprensione; essi promuovono l'interazione *faccia a faccia*, come ad esempio nell'utilizzo di un singolo display condiviso;
- **sincroni distribuiti**, che permettono ai collaboratori di lavorare insieme nello stesso momento anche se da luoghi differenti, proponendo

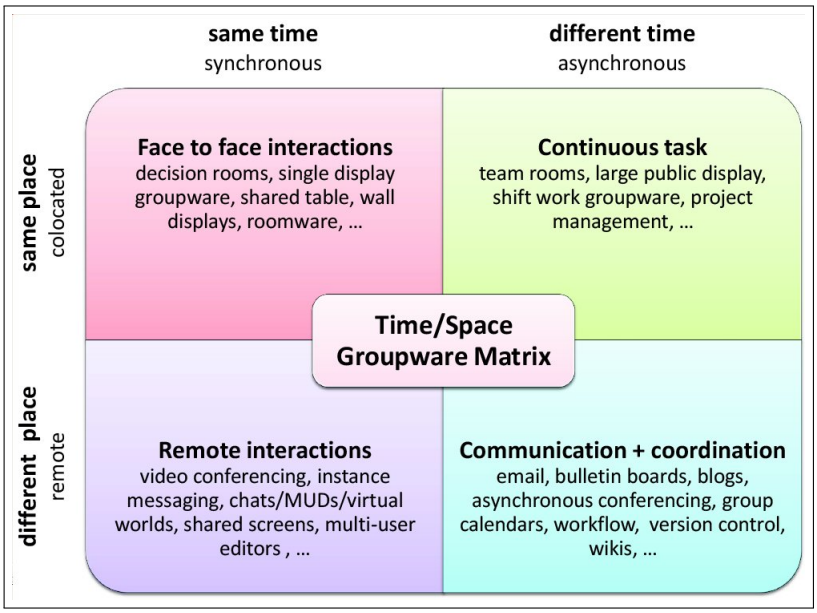

Figura 1.1: Classificazione per sistemi CSCW basata sulle dimensioni di spazio e tempo.

ad esempio spazi condivisi di lavoro in cui gli utenti possono creare e modificare artefatti condivisi; esempi di questi sistemi, che spesso includono anche funzionalità per aumentare la consapevolezza delle rispettive attività svolte all'interno del gruppo, sono quelli che permettono le comunicazioni real-time, sia in forma testuale che video o audio, come anche gli editor di testi condivisi;

- **asincroni co-ubicati**, che consentono la collaborazione in uno stesso luogo senza richiedere la simultanea presenza dei componenti del gruppo; di fatto in questo tipo di sistemi è di solito fornito il supporto alla coordinazione attraverso una postazione centrale in cui le informazioni possono essere mantenute in maniera persistente in attesa che la comunicazione avvenga quando previsto dalle attività dei vari utenti, come ad esempio in quelli che abilitano la gestione condivisa asincrona per il deposito di informazioni quali documenti;

- **asincroni distribuiti**, che permettono agli utenti di collaborare senza porre predeterminati vincoli temporali o spaziali, visto che gli utenti non devono essere presenti simultaneamente nello stesso posto; in questo approccio le informazioni utili alla collaborazione devono necessariamente essere rese persistenti in quanto gli utenti possono accedervi in maniera asincrona quando previsto dalle loro attività.

1.1.4 Mobile CSCW

Seguendo questi principi, un ulteriore livello di classificazione può essere fornito considerando i recenti sviluppi che hanno portato ad una grande diffusione dei dispositivi mobile, come si fa notare in [4], abilitando di fatto nuove forme di collaborazione prima ritenute impraticabili: infatti, considerando la dimensione spaziale di questi sistemi, e quindi la loro distribuzione geografica, si nota che essi possono essere ulteriormente suddivisi secondo la variabilità della loro posizione. I sistemi che mantengono le componenti del sistema sostanzialmente in un'unica ubicazione senza mai variarla, chiamati *statici*, possono essere classificati secondo l'approccio che considera le due dimensioni citate in precedenza, mentre quelli cosiddetti *dinamici*, in cui la mobilità dei componenti rappresenta una caratteristica fondamentale, aggiunge questa dimensione alla classificazione. Tali sistemi dinamici sono inerentemente distribuiti geograficamente e l'informazione riguardante la posizione dei componenti assume un ruolo importante all'interno del sistema: la possibilità di spostarsi fisicamente introduce ulteriori complessità soprattutto dal punto di vista della disponibilità del servizio, che potrebbe subire delle variazioni proprio a seconda della posizione assunta (e.g. in una zona in cui non è possibile avere connettività). Questi sistemi, riferiti anche come "mobile", possono essere ulteriormente suddivisi in:

- **micro-mobile**, in cui è permesso un livello minimo di mobilità nell'accesso ad un artefatto condiviso, che può quindi essere manipolato in maniera circoscritta, prevalentemente in situazioni di compresenza;
- **local-mobile**, in cui viene considerato un livello di mobilità relativo ad un singolo ambiente di lavoro, per esempio un edificio, in cui agli utenti è quindi permesso spostarsi fra i vari piani e le differenti stanze;
- **remote-mobile**, atti a descrivere quei sistemi in cui gli utenti si spostano in differenti luoghi o posti di lavoro con spostamenti anche su larga scala; questo tipo di sistemi si differenzia particolarmente dagli altri le variazioni in termini di spazio e di tempo per la collaborazione possono variare dipendentemente dall'attività lavorativa coinvolta.

I sistemi mobile, e in particolar modo quelli di tipo remoto, presentano peculiari caratteristiche che ne aumentano la complessità di realizzazione: essendo infatti i componenti del gruppo di lavoro fortemente disaccoppiati sia dal punto di vista temporale che spaziale, si riscontrano le tipiche difficoltà causate dall'utilizzo di tecnologie wireless, in generale meno affidabili di quelle wired, come perdita di dati, ritardi e soprattutto frequenti disconnessioni. Per mitigare queste difficoltà nel corso degli anni sono state sviluppate

diverse tecniche che permettono di aumentare il livello di asincronia nell'interazione tra i collaboratori, permettendo loro anche ad esempio di continuare il loro lavoro anche se *offline*, ad esempio tramite strategie di caching o replica dei dati, che però introducono problematiche di consistenza, da gestire poi dipendentemente dai particolari requisiti applicativi, secondo specifiche strategie di versioning o risoluzione dei conflitti. Al momento, però, non si è ancora identificata una strategia generale che permetta di affrontare efficacemente questo problema ai vari livelli (applicativo o infrastrutturale) in cui si pone.

Inoltre è importante ricordare che un altro aspetto di rilievo per questi sistemi è la caratterizzazione del *controllo* che gli utenti hanno in relazione all'interazione con gli altri componenti del gruppo: ci si riferisce infatti ad una forma ci controllo **esplicita** quando gli utenti possono percepire direttamente la dimensione dell'interazione e della cooperazione all'interno del gruppo, come ad esempio nei sistemi basati sul dialogo diretto, mentre si fa riferimento ad una forma **implicita** quando non sono supportate tecniche per rappresentare direttamente questa forma di coordinazione di gruppo, la cui cooperazione viene quindi dettata implicitamente dagli stili di interazione proposte e permesse.

1.2 Sistemi per healthcare e ICT

Nel corso degli ultimi anni il veloce avanzare delle nuove tecnologie ha avuto un forte impatto sui sistemi in ambito **healthcare**: sebbene il supporto dell'ICT abbia ormai un ruolo consolidato ed essenziale nelle strutture adibite normalmente alla cura e all'assistenza sanitaria dei pazienti, come per esempio con l'informatizzazione delle cartelle cliniche dei pazienti negli ospedali, la grande diffusione di dispositivi mobili ha aperto nuovi scenari di utilizzo dell'informatica in questo ambito.

Le tecnologie mobile abilitano infatti il supporto informatico anche al di fuori degli ambienti standard della sanità dove è ormai chiaro che strumenti tecnologici all'avanguardia siano di fondamentale importanza: è proprio in questi scenari particolari che si può immaginare il prolifico utilizzo di questi dispositivi, in modo da supportare il soccorso in situazioni di emergenza causate, ad esempio, da disastri ambientali o guerre.

1.2.1 eHealth

L'importanza dell'ICT in ambito sanitario viene sottolineata dall'introduzione, alla fine degli anni '90, del termine **eHealth** (anche *Electronic Health*

oppure *e-health*), a cui nel corso degli anni sono state date svariate definizioni, come descritto in [5]: con esso ci si riferisce al supporto dato dalle tecnologie informatiche al processo di assistenza sanitaria, sottolineando come queste non si sostituiscano all'attività del personale medico, ma lo assistano e ne aumentino, grazie al loro aiuto, le capacità e le possibilità al servizio del paziente. É in questo senso che la tecnologia si è infatti ritagliata un ruolo fondamentale nella medicina, sin dai suoi primi utilizzi nell'ambito della *telemedicina*, con cui ci si riferisce all'insieme di tecniche che permettono di assistere il paziente a distanza: essa rappresenta solo uno dei molteplici aspetti di applicazione di sistemi eHealth.

Con *eHealth* si fa infatti riferimento ad una vasta gamma di sistemi a supporto delle operazioni sanitarie, che comprende ad esempio la gestione a distanza dei pazienti, l'informatizzazione delle loro cartelle cliniche per un più rapido ed efficace accesso alle informazioni ed anche i sistemi di diffusione di informazioni mediche, che permettono di aumentare l'effetto di campagne di sensibilizzazione e prevenzione.

L'utilizzo di ICT in ambito *healthcare* ha avuto un forte impatto in molti dei suoi aspetti, tra cui i più importanti sono:

- **accessibilità**, in quanto le tecnologie informatiche hanno abilitato un più rapido ed efficace accesso ai trattamenti sanitari sia in generale, che, nello specifico, per persone che vivono in zone remote e difficilmente raggiungibili (ad esempio grazie a servizi di telemedicina) oppure in situazioni particolari come a bordo di un aereo o su una nave, casi in cui non sarebbe altrimenti possibile ricevere supporto; si può inoltre fare riferimento anche ad una più capillare diffusione di informazioni mediche (sia per professionisti che pazienti), che ne permette un accesso più fruibile ed immediato tramite la rete, aumentando anche il grado di *educazione* sanitaria delle persone;

- **economia**, visto che l'utilizzo della tecnologia può ridurre i costi di gestione della sanità, il cui alto livello costituisce un grande problema in molti paesi, in quanto essa può aiutare ad automatizzare alcuni processi rendendoli più efficienti e, ad esempio, decentralizzare l'offerta dei servizi evitando costi relativi ai trasporti;

- **qualità**, in quanto il supporto informatico risulta essere molto importante per aumentare la qualità e l'efficacia del servizio del servizio sanitario, fornendo ausilio al lavoro degli operatori medici sia in fase di diagnosi, ad esempio tramite l'utilizzo di strumentazione avanzata che permette analisi molto precise oppure l'accesso ad informazioni com-

plete sul quadro clinico del paziente, che durante lo svolgimento delle loro mansioni, agevolandole e riducendone le possibilità di errore;

- **educazione**, visto che l'informatica aumenta la diffusione di informazioni, ad esempio attraverso l'uso di Internet che ha reso il loro accesso agevole per le persone, aiutando in tal modo il processo di prevenzione che risulta fondamentale in medicina.

In generale è quindi possibile notare come l'utilizzo di ICT abbia portato notevoli benefici e cambiamenti in ambito sanitario, migliorando l'efficienza e la qualità del servizio offerto tramite l'utilizzo di sistemi informatici e reti di telecomunicazione che hanno aumentato le possibilità di collaborazione nelle attività del personale medico. Trattando però un ambito delicato e di grande rilevanza come quello riguardante la salute delle persone, l'utilizzo delle tecnologie deve essere anche attento a particolari problematiche quali quelle riguardanti la *privacy* dei dati trattati, di grande sensibilità in quanto strettamente personali.

1.2.2 mHealth

L'utilizzo pervasivo della rete Internet e dei supporti informatici ha segnato una svolta in ambito sanitario contrassegnata dall'introduzione di sistemi eHealth, ma la recente grande diffusione di dispositivi mobili ha portato un nuovo e se possibile maggiore cambiamento in questo settore, tanto da giustificare la comparsa di un nuovo termine, **mHealth** (*Mobile Health*), con cui si vuole indicare l'utilizzo di dispositivi e applicazioni mobile in congiunzione con le tecnologie wireless per il supporto alla salute delle persone. Anche se tuttora non esiste una definizione generalmente accettata, quella proposta in [6], in cui ci si riferisce ad mHealth come *"l'uso delle comunicazioni e tecnologie di rete mobile per l'assistenza sanitaria"*, sintetizza bene come la possibilità di usufruire di servizi relativi alla propria salute da dispositivi mobile segni una grande innovazione in questo campo.

Da questo punto di vista, i sistemi mHealth possono essere visti come un sottoinsieme di quelli eHealth in cui si fa riferimento nello specifico all'offerta di servizi sanitari tramite tecnologie mobile, tra cui anche dispositivi indossabili (*wearables*): essi costituiscono perciò l'evoluzione dell'utilizzo delle tecnologie informatiche per la salute, in cui la possibilità di usufruire di connessioni wireless abilita l'accesso in *mobilità* ai servizi e alle informazioni. L'utilizzo di queste nuove tecnologie aggiunge quindi ai tradizionali sistemi la potenzialità di accesso *ovunque* e in *qualunque momento* [7], andando di fatto in direzione dei sistemi pervasivi.

É evidente che la diffusione dei dispositivi mobili è in gran parte dovuta all'evoluzione delle tecnologie di comunicazione senza fili, che costituiscono quindi anche la base fondamentale dei sistemi per mobile health: è grazie ad esse infatti che si può ottenere quell'*ubiquità* nell'accesso che caratterizza questo tipo di sistemi. Tra i maggiori metodi di comunicazione utilizzati che, grazie a stazioni sia satellitari che di terra, coprono gran parte del pianeta, comprendendo anche quelle con scarsa disponibilità di infrastrutture, si possono citare [8] quelle espressamente mobile come GSM, GPRS, EDGE, 3G, UMTS, HDSPA, quelle facenti riferimento alle WLAN (Wireless Local Area Network), tecnologie Bluetooth per l'abilitazione a comunicazioni ad-hoc opportunistiche. Nonostante le recenti grandi innovazioni in questo campo ottenere un livello di connettività efficace per le comunicazioni risulta essere una delle maggiori sfide per questo tipo di sistemi, data la mancanza di infrastrutture che ad esempio gestiscano in maniera trasparente ed efficace repentini cambi del metodo di comunicazione utilizzato a seconda delle contingenze.

Tra le principali potenzialità introdotte da mHealth, oltre ad una migliorata immediatezza e disponibilità nell'accesso ai servizi, si possono riscontrare maggiori possibilità riguardanti il *remote tracking* dei pazienti e del loro stato di salute, sfruttando ad esempio anche la funzionalità di geolocalizzazione che oggigiorno è presente su molti dispositivi mobile. Risulta quindi evidente che l'utilizzo di sistemi mobile in ambito healthcare introduca nuove possibilità dal punto di vista del monitoraggio remoto di pazienti e delle loro patologie (ad esempio l'acquisizione dei loro parametri vitali), così come da quello della raccolta dei dati a fini di analisi come quelle riguardanti la diffusione di una malattia.

Inoltre, per mostrare come mHealth possa essere di utilità anche in scenari non considerati dal tradizionale utilizzo delle tecnologie informatiche in ambito sanitario, si può considerare lo scenario del supporto al soccorso in situazioni di emergenza: in queste situazioni, che presentano caratteristiche totalmente differenti da quelle relative agli ambienti in cui normalmente vengono effettuati le operazioni sanitarie standard, l'accesso immediato ai servizi tramite dispositivi mobili assume una fondamentale importanza nell'aiutare gli operatori sanitari a svolgere il proprio lavoro e, dunque, a salvare delle vite. Il supporto a questo tipo di eventualità appare dunque essere uno dei principali campi di sviluppo per mHealth, tanto che in [8] viene introdotto il termine **eEmergency** per riferirsi a tali sistemi che si prefiggono di affrontare le emergenze in ambito healthcare grazie all'impiego di innovativi strumenti tecnologici e, in particolare, informatici.

1.2.3 Healthcare e Pervasive Computing

Grazie alle recenti innovazioni introdotte in ambito ICT è ora possibile pensare a scenari di *pervasive healthcare*, come descritto in [9]: le nuove tecnologie per le comunicazioni wireless e i progressi dei sistemi embedded e della sensoristica permettono infatti di raccogliere efficacemente dati riguardanti la salute dei pazienti, in modo da renderli disponibili ai sistemi che ne devono fare uso. In questo senso i sistemi per il supporto all'assistenza sanitaria rientrano nella categoria dei sistemi pervasivi, ed in particolare sono progettati per fornire differenti tipi di servizi medici, supportando individualmente gli utenti, adattandosi alle loro condizioni di salute e più in generale al loro stile di vita. Allo stato dell'arte gli approcci utilizzati per raggiungere questi scopi possono essere riassunti in tre categorie [9]:

- *Smart Artefacts e dispositivi mobili*, che rappresenta l'approccio maggiormente utilizzato in cui si sfrutta il dispositivo mobile dell'utente (o un equivalente "smart object") per fornire un servizio che si adatti al contesto in cui si trova indipendentemente però dalla posizione del paziente; esempi di smart artefacts possono essere oggetti di uso quotidiano le cui potenzialità sono state aumentate con capacità computazionali e di comunicazione, oltre all'utilizzo di sensori, che sono in grado di raccogliere informazioni relative al contesto in cui sono immersi e comunicarle;

- *Wearables*, che aggiungono alle caratteristiche dei precedenti artefatti anche quella di essere indossabili, rispondendo allo scopo di fornire all'utente un supporto continuo senza richiedergli particolari interazioni;

- *Smart Environments*, la cui idea fa riferimento allo scenario per eccellenza di pervasive computing, in cui una moltitudine di oggetti con capacità computazionali è sparsa per l'ambiente in modo da supportare il paziente nelle sue attività quotidiane così come fornire un efficace metodo per controllarlo.

1.3 Pervasive Computing

Il termine **Pervasive Computing** ha ultimamente ottenuto grande risonanza grazie all'avanzare delle innovazioni riguardanti tecnologie wireless e dispositivi mobile ed embedded, ma risale in realtà all'idea di *Ubiquitous Computing* degli anni '70, anche se un fondamentale punto di svolta è rappresentato dal 1991, anno in cui Mark Weiser, che faceva allora parte dello

Xerox Palo Alto Research Center, pubblicò il famoso articolo *"The Computer for the 21st Century"* [10], in cui concretizzava la sua visione di evoluzione del mondo dell'informatica verso una sempre maggiore integrazione nell'ambiente circostante di dispositivi che, grazie alle loro capacità computazionali e di comunicazione, facessero in modo che le persone li potessero agevolmente sfruttare e utilizzare per una maggiore interazione con l'ambiente stesso.

Caratteristica principale di questa visione era infatti la *trasparenza* dell'integrazione delle tecnologie informatiche nell'ambiente rispetto alle persone, sintetizzata molto bene da una citazione tratta da [10]:

> *The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.*

Pur descrivendo uno scenario chiaro e concreto, i tentativi effettuati da Weiser in direzione della sua visione risentivano delle limitazioni tecnologiche all'epoca presenti, in cui i dispositivi mobile o embedded per l'interazione con l'ambiente non erano ancora diffusi o le tecniche per la comunicazione wireless non erano così avanzate, ma è proprio a quella che si sono ispirate le ricerche e le innovazioni degli ultimi tempi, in direzione di una sempre maggiore integrazione dell'informatica nell'ambiente e nella realtà, tanto da giustificare l'utilizzo del termine *Pervasive Computing* a sottolineare il carattere pervasivo con cui la capacità computazionale si diffonde e manifesta nel mondo che ci circonda e negli oggetti di uso quotidiano.

1.3.1 Evoluzione verso il Pervasive Computing

I sistemi informatici hanno subito una rapida e notevole evoluzione dalla loro introduzione fino ad ora: dai grandi e costosi mainframe, i computer degli anni '60 per i quali si parlava di paradigma *many people per computer*, in quanto molti utenti condividevano una sola macchina, si è giunti al paradigma *one person per computer*, in cui ogni persona può disporre di un proprio calcolatore, grazie al progresso tecnologico che ha consentito la realizzazione dei personal computer, avvicinando sempre più persone all'utilizzo del computer. Nell'ultimo periodo la diffusione di laptop, Personal Digital Assistant(PDA), smart devices, dispositivi mobile dotati di microprocessori e di capacità di immagazzinare dati ha mutato ulteriormente il rapporto uomo-computer, aprendo le porte all'era dei *many computers per person*: tanti elaboratori per una singola persona.

Questa evoluzione ha portato ovviamente molti cambiamenti anche nel modo di concepire e progettate i sistemi informatici, di cui il Pervasive Com-

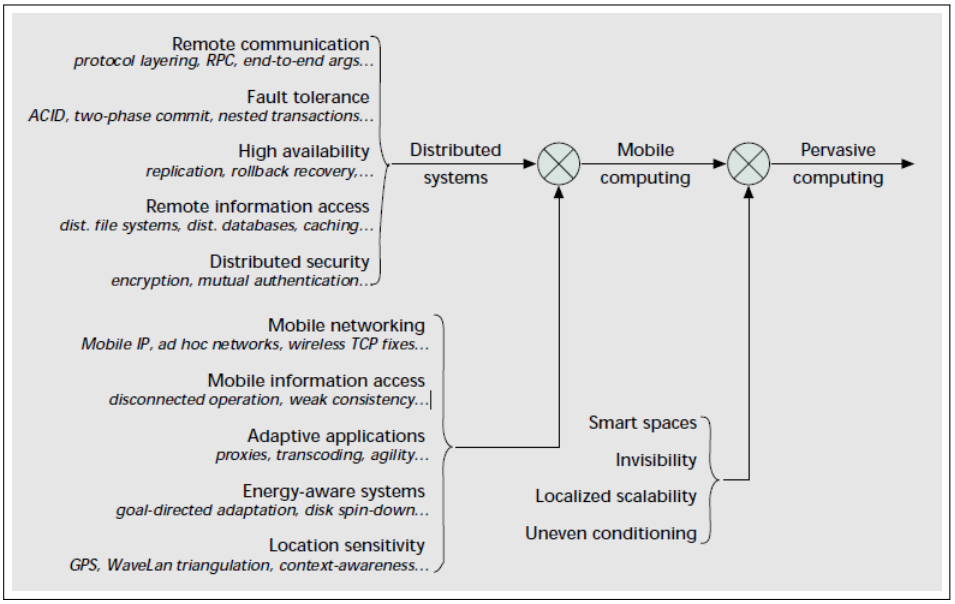

Figura 1.2: Evoluzione verso i sistemi pervasivi (fonte [11])

puting rappresenta uno degli ultimi passi evolutivi, come è possibile notare dalla figura 1.2 e come ben descritto sia in [11] che in [12].

Il Pervasive Computing può essere visto come una nuova forma di computazione altamente dinamica e disaggregata: gli utenti sono mobili, i servizi sono forniti da una serie di componenti distribuiti che collaborano tra loro e le applicazioni necessarie per supportare queste nuove esigenze sono costituite, da un punto di vista architetturale, da moduli allocati in numerosi nodi della rete. In tal senso, il Pervasive Computing costituisce una nuova tappa nel percorso evolutivo a seguito del *Distributed Computing*, più concentrato a ripartire dati e capacità computazionali su componenti indipendenti, potenzialmente residenti su macchine diverse, che comunicano tra loro attraverso l'utilizzo di reti di comunicazione.

L'introduzione di vincoli e problematiche legate al concetto di mobilità ha però reso necessarie nuove soluzioni tecnologiche che hanno portato ad un passo intermedio nell'evoluzione, a cui ci si riferisce con il termine *Mobile Computing*: in questi nuovi scenari di calcolo distribuito, non si hanno più nodi di rete fissi, con connessioni stabili e veloci, ma nodi costituiti da dispositivi mobili che possono spostarsi geograficamente, accedono alla rete e la abbandonano continuamente ed in maniera del tutto imprevedibile, dotati di connessioni precarie, e in cui le limitate capacità di calcolo e di memoria, unite alle esigenze di risparmio energetico, rappresentano altre caratteristiche di cui tenere conto.

Come si nota dalla figura 1.2, ad ogni step sono aggiunti requisiti e caratteristiche in più da considerare nella progettazione e nello sviluppo della nuova categoria di sistemi, tenendo conto del fatto che bisogna prestare attenzione anche a quelli introdotti dalle categorie precedenti: è per questo che la realizzazione di sistemi pervasivi presenta una grande complessità, in quanto le problematiche da affrontare sono molte e di varia natura.

Bisogna inoltre ricordare che, teoricamente, si potrebbe parlare di sistemi pervasivi anche senza considerare una componente strettamente mobile, con il rischio però di tralasciare una grande fetta di caratteristiche e funzionalità che invece gli utenti potrebbero aspettarsi, data la grande diffusione dei dispositivi mobili: risulta così più idoneo integrare le caratteristiche peculiari del mobile computing al loro interno, aggiungendole a quelle più specifiche di pervasività, per cui si arriva a parlare di *Pervasive Mobile Computing*.

1.3.2 Principali caratteristiche

In [12] viene proposto un modello di riferimento per la realizzazione di applicazioni per pervasive computing che consta sostanzialmente di quattro elementi principali (figura 1.3):

- *devices*, dispositivi che possono essere di svariate tipologie ma comunque dotati di capacità computazionali, di una propria interfaccia con il quale è possibile interagire, di supporti per la comunicazione tramite rete ed infine di mezzi, come sensori, per interagire con l'ambiente reale circostante;

- *pervasive networking*, con cui ci si riferisce a tutti quei componenti atti a garantire la connessione e le comunicazioni tra le parti del sistema, affidandosi ad esempio ad un'infrastruttura di rete che dia affidabilità e robustezza;

- *pervasive middleware*, un livello intermedio che, in analogia ai sistemi distribuiti, si frappone come livello di astrazione fra le applicazioni pervasive ed i livelli sottostanti come quello delle comunicazioni in rete; in sostanza i componenti facenti parte di questo livello si occupano di garantire trasparenza nell'utilizzo dei sistemi pervasivi;

- *pervasive applications*, che rappresentano le applicazioni da eseguire grazie all'utilizzo dei precedenti componenti.

Inoltre, componenti fondamentali dal punto di vista tecnologico sono certamente i dispositivi mobile ed embedded, che ben si prestano ad essere

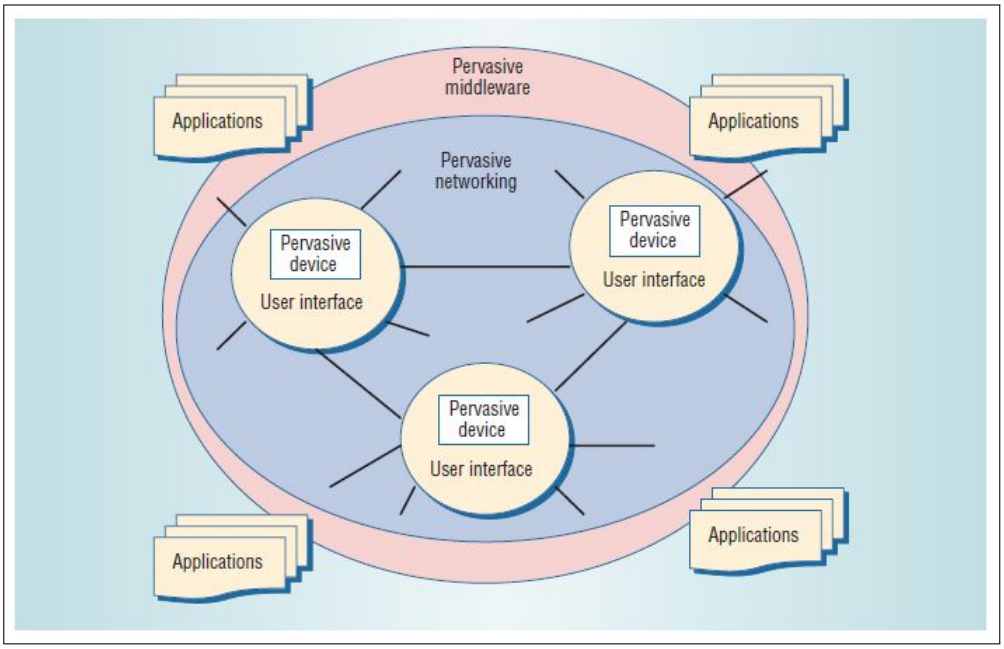

Figura 1.3: Pervasive Computing framework (fonte [12]).

integrati pervasivamente nell'ambiente e un'infrastruttura che abiliti le comunicazioni wireless in maniera robusta ed affidabile, così da permettere le interconnessioni necessarie: oltre a questi anche il *Cloud Computing* può rivestire un ruolo di utilità qualora la connettività ad Internet sia disponibile, abilitando il trasferimento di funzionalità onerose dal punto di vista computazionale dai limitati device a risorse accessibili in rete on-demand.

A partire dal modello descritto, dai tratti molto generali, emergono le principali caratteristiche e problematiche da affrontare nella progettazione di sistemi pervasivi, che inducono scenari in cui servizi ed informazioni sono accessibili virtualmente ovunque ed in qualunque istante, in quanto integrati con l'ambiente circostante.

Rapporto con Mobile Computing

I sistemi per mobile computing presentano delle particolarità che possono tornare utili nella realizzazione di sistemi pervasivi, i quali, costituendone un'evoluzione, ne devono considerare anche i vincoli indotti per sfruttarne efficacemente le feature. Una loro sintesi è, come descritto in [11]:

- *mobile networking*, in quanto i dispositivi mobili comunicano in pratica esclusivamente tramite l'utilizzo di protocolli wireless, di cui perciò vanno tenute in conto anche le eventuali limitazioni;

- *accesso alle informazioni in mobilità*, che include anche la possibilità di demandare servizi troppo onerosi per le limitate capacità computazionali del dispositivo a servizi in rete, oltre che alla possibilità di effettuare operazioni mentre si è disconnessi e valutare la qualità e la consistenza delle informazioni;

- *supporto all'adattatività*, in quanto i sistemi sono prevalentemente di natura eterogenea e le applicazioni devono potersi adattare ai differenti tipi di dispositivi;

- *energy awareness*, visto che la limitazione dal punto di vista energetico degli attuali dispositivi mobile è una caratteristica da tenere ben presente;

- *location awareness*, che include il rilevamento della posizione corrente, che può variare data la mobilità, e l'adattamento ai suoi cambiamenti.

Context awareness, proattività ed adattatività

Una prerogativa di un sistema pervasivo è la capacità di ottenere informazioni sugli utenti e sullo stato dell'ambiente che lo circonda: informazioni come la posizione e l'identità dei singoli utenti, la disponibilità di risorse, la presenza di un certo numero di dispositivi in una certa area oppure i dati provenienti dai sensori o più in generale quelle considerate rilevanti per il **contesto** all'interno del quale il sistema è immerso, sono in grado di modificare i comportamenti intrapresi dai componenti del sistema, che ad esse quindi si adattano. Questa forte dipendenza tra i sistemi pervasivi ed il loro contesto di esecuzione induce quindi a ritenere la **context awareness** come una delle loro precipue caratteristiche, rendendoli di fatto uno dei maggiori esempi di sistemi *context-aware*: due sono le principali modalità con cui questa caratteristica viene esibita, cioè in maniera *attiva* se il sistema monitora cambiamenti nel contesto e agisce autonomamente in base ad essi oppure *passiva* se, a fronte del monitoraggio, si limita ad offrire all'utente opportune scelte di azione tra cui sarà poi lui a decidere quale corso d'azione intraprendere.

Ottenere informazioni di qualità tramite una fase di *acquisizione*, ad esempio collezionando dati da una moltitudine di sensori sparsi nell'ambiente, risulta essere fondamentale al fine di adattare il comportamento delle applicazioni a seconda delle circostanze: un'accurata percezione dell'ambiente risulta quindi necessaria, insieme a meccanismi per il rilevamento e la correzione di informazioni di contesto inattendibili o contrastanti.

Altre fasi di uguale importanza nella gestione delle informazioni contestuali, che costituisce un processo continuo costantemente presente per tutta l'attività del sistema, risultano essere la *manipolazione*, in cui ci si occupa di trasformare le informazioni rilevate nel formato più adatto per essere gestite all'interno del sistema, la *decisione* che comprende anche eventuale reasoning sulle informazioni ottenute per la modifica dello stato del sistema e il supporto alle decisioni riguardante le successive azioni da intraprendere, ed infine la loro *distribuzione* con la quale esse vengono propagate agli altri componenti del sistema.

Così, visto che diversamente dai paradigmi in cui il comportamento del computer è principalmente *reattivo*, composto di risposte all'interazione con l'utente, il pervasive computing mira alla realizzazione di un modello nel quale, a partire dalle informazioni ricavate sul contesto in cui è inserito, il sistema dovrebbe essere in grado di riconoscere le azioni dell'utente e guidarlo nell'attività, possibilmente capendone anche le intenzioni. É in questo senso che assume importanza la *proattività* del sistema, in cui non si considera più solo il livello dato dai singoli dispositivi, ma più in generale il concetto di ambiente, inteso spesso come *smart space* in cui sono immersi dispositivi in grado di comunicare e dotati di sensori e/o attuatori di vario genere.

Trasparenza e invisibilità

Come ricordato da Weiser in [10], l'*invisibilità* delle tecnologie immerse nell'ambiente, intesa come trasparenza rispetto all'utente, risulta fondamentale per un sistema pervasivo, che deve supportare diversi tipi di interazione, anche indiretta o semi-diretta, al fine di provocare minor distrazione possibile: a questo proposito possono essere utili anche meccanismi che consentano di prevedere le azioni dell'utente, realizzati ad esempio tramite una profilazione delle sue abitudini. Questo tipo di previsione, che risulta di grande importanza per aumentare la facilità di utilizzo del sistema e con essa la trasparenza rispetto all'utilizzatore, costituisce comunque una grande sfida da affrontare nella progettazione di questi sistemi, data la loro dinamicità e la grande variabilità di condizioni da affrontare, a fronte delle quali le azioni scelte dall'utente possono potenzialmente cambiare di volta in volta.

Diviene necessario quindi stabilire anche un compromesso tra le caratteristiche di trasparenza e proattività nel sistema: anche se il sistema può prendere autonomamente delle iniziative, per garantire una buona usabilità all'utente deve essergli possibile eseguire le azioni che ritiene più idonee nei momenti che ritiene opportuni.

Un altro tipo di trasparenza che riveste importanza è quella relativa all'accesso alle *risorse*, che dovrebbero essere rappresentate in maniera tale da

poter essere scoperte e interrogate in base alle loro caratteristiche, generali o specifiche che siano; in più la riconfigurazione dinamica del sistema a fronte dell'aggiunta o della perdita di alcune componenti non deve, entro certo limiti, comportare un costo oneroso in termini di usabilità del sistema.

É relativamente all'utilizzo delle risorse che bisogna inoltre prevedere ulteriori meccanismi di adattatività nel caso in cui esse siano temporaneamente non disponibili, come la connettività o l'energia nei mobile device: in questo senso differenti strategie possono essere messe in campo, prevedendo ad esempio di mascherare parzialmente questa mancanza, proponendo azioni correttive all'utente oppure richiedendo una certa qualità di servizio all'ambiente demandando a quel livello la problematica.

Bisogna inoltre considerare anche l'esistenza di ambienti pervasivi caratterizzati da diversi livelli di integrazione con i dispositivi, le cui differenze potrebbero causare delle mancanze in termini di usabilità e trasparenza: affrontare questo aspetto significa permettere al sistema di "compensare" le mancanze degli ambienti meno integrati, ad esempio permettendo un utilizzo offline delle risorse qualora venissero a mancare le condizioni per le comunicazioni in rete.

Eterogeneità

Un ambiente pervasivo è caratterizzato da una moltitudine di dispositivi eterogenei, la cui riduzione delle dimensioni comporta una crescita del loro numero ed una intensificazione delle interazioni uomo macchina: questa grande diversità costituisce una fonte di complessità ad esempio nello sviluppo delle applicazioni, che devono supportare differenti tipi di dispositivi su cui essere eseguite. In particolare anche le interfacce proposte all'utente devono adattarsi alle caratteristiche dei diversi device utilizzati in modo da fornire all'utente in ogni caso un'adeguata qualità di servizio e funzionalità, dimostrando ancora una volta la necessità di esibire trasparenza e adattatività a diversi livelli per immergersi nell'ambiente in maniera tale da esserne indistinguibile dalla prospettiva umana.

Security e Privacy

Un sistema pervasivo generalmente gestisce grosse quantità di informazioni, molte delle quali acquisite proattivamente tramite sensori e dispositivi in possesso degli utenti, riguardanti le loro abitudini e preferenza, dal cui punto di vista è desiderabile che vengano rispettati i principi di privacy e sicurezza già noti in ambito dei sistemi distribuiti e mobile, prevedendo ad esempio strategie di controllo degli accessi.

In molte situazioni però può essere tollerato un compromesso che preveda una certa perdita di *privacy*, come ad esempio in situazioni di pericolo: progettare meccanismi che tenendo conto del contesto decidano quali informazioni divulgare e soprattutto a quali entità renderle disponibili è un altro fattore che aumenta la complessità di questi sistemi.

Fault Tolerance e Scalability

Gli ambienti pervasivi costituiscono sistemi sempre attivi, pertanto un guasto ad un componente non dovrebbe compromettere il funzionamento generale dell'intero sistema, ma al contrario essere gestito tramite una sua riconfigurazione dinamica: i componenti caduti in errore dovrebbero automaticamente ripartire, laddove possibile, magari adoperando, ad esempio, memorie di stato persistenti che consentano di effettuare *resume* rapidi ed efficaci, esibendo *fault tolerance* da questo punto di vista.

Gli ambienti pervasivi sono inoltre caratterizzati da una forte dinamicità, grazie a cui dispositivi possono aggiungersi all'ambiente ed abbandonarlo in qualsiasi momento, alcuni servizi possono risultare non disponibili e altrettanti nuovi possono diventarlo, gli stessi utenti possono entrare ed uscire dall'ambiente secondo la propria volontà. Il sistema deve quindi esibire *scalabilità*, ossia essere in grado di gestire e assicurare il suo funzionamento anche in seguito all'aggiunta di componenti, garantendo allo stesso tempo che la loro introduzione non interferisca con quelli già esistenti: per i sistemi pervasivi parlare di scalabilità in termini generali è abbastanza complicato in quanto costituisce un problema critico. Per questo si parla quindi di *scalabilità localizzata* nel senso che non è necessario che sia uniforme per tutto il sistema, ma deve essere garantita in funzione delle interazioni con gli utenti, concentrandosi sui componenti con le quali avvengono il maggior numero di interazioni: ad esempio si può considerare che il numero di interazioni cali all'aumentare della distanza tra l'ambiente pervasivo e l'utente, riducendo così il loro numero.

Capitolo 2

Stato dell'arte

Tra i vari sistemi che abilitano la cooperazione sono di particolare interesse nel contesto di questa tesi quelli volti alla gestione gestione delle emergenze, che rientrano nella più ampia categoria degli *Emergency Management and Response System* (in breve **EMR**): essi si prefiggono in particolare di supportare la gestione di tali situazioni (come crisi o disastri naturali) anche tramite l'utilizzo di dispositivi informatici, in modo da contenere il più possibile eventuali danni alle persone o alle proprietà materiali. É al loro interno quindi che possono essere considerati in particolare quelli che, in ambito healthcare, riguardano il soccorso alle persone in questi scenari, rientrando nella categoria di sistemi *eEmergency* descritta in 1.2.2: la loro realizzazione pone differenti sfide in quanto essi rientrano nella categoria di sistemi distribuiti, eterogenei e complessi la cui progettazione deve tenere conto anche di specifiche non funzionali per raggiungere un risultato che sia robusto e veramente utilizzabile in queste situazioni, talvolta drammatiche.

Questo capitolo ha così l'obiettivo di descrivere gli esempi principali emersi da una ricognizione dello stato dell'arte, dando per ognuno di essi una breve descrizione dei suoi scopi e della sua realizzazione, in modo da poterne ricavare una sintesi delle proprietà, limitazioni e degli aspetti comuni riscontrati.

2.1 Sistemi a supporto della collaborazione in situazioni di emergenza

Il caso di studio considerato in questa tesi, descritto approfonditamente nel capitolo 3, si inserisce in questa particolare categoria, presentando così alcune delle principali caratteristiche trattate in generale nel capitolo 1, in partico-

lare calate nel contesto specifico della gestione del soccorso cooperativo in risposta all'emergenza.

Più in generale le difficoltà da affrontare nella progettazione di tali sistemi si possono evincere da alcune delle loro caratteristiche peculiari, tra le quali figurano:

- necessità di condivisione delle informazioni tra i differenti attori presenti nella scena e preposti alla gestione di situazioni di emergenza, per esempio operatori sul campo e analisti in una centrale di controllo, al fine di snellire e rendere più efficace il processo decisionale per definire il miglior corso di azioni da intraprendere;

- comunicazione in scenari di mobilità remota, in quanto spesso i membri del personale sono sparsi e distribuiti all'interno di un'area geografica di dimensioni variabili e collaborano grazie al supporto fornito dall'utilizzo di *smart devices*, che gli permettono di comunicare sfruttando connessioni wireless e di interagire con l'ambiente tramite i sensori a disposizione, primo fra i quali quello che abilita la *geo-localizzazione*;

- interfaccia *user-friendly* per l'utente in modo tale da supportare e velocizzare le sue azioni, visto che minimizzare il tempo richiesto per la loro esecuzione può risultare di vitale importanza, permettendo di salvare più persone possibili.

Gestire dunque tutti questi particolari aspetti, primo fra i quali la garanzia di un adeguato livello di comunicazione per il supporto ad una buona condivisione delle informazioni, risulta un'attività complessa che deve tenere conto di molteplici aspetti: *interoperabilità* per fare in modo che dispositivi di diversa natura possano interagire, *context awareness* per abilitare forme di collaborazione dipendenti dal contesto, in particolare in riferimento alla posizione geografica, *fault tolerance* e resistenza ai guasti per dare al sistema un elevato grado di *disponibilità* sono solo alcuni fra quelli più importanti.

Più in generale nel ciclo di gestione delle emergenze i sistemi considerati all'interno di questo lavoro si concentrano nella fase di *risposta*, che avviene subito dopo l'evento a causa della situazione di crisi, in cui si intraprendono le azioni per limitare i danni e salvare il maggior numero di vite coinvolte, mettendo in atto i piani definiti in ambito di *preparazione*.

Attraverso un'esplorazione dell'attuale stato dell'arte si possono ritrovare differenti e molteplici tentativi di affrontare le problematiche presentate in ambito di gestione dell'emergenza e healthcare: realizzati in forma più o meno prototipale, questi sistemi presentano in parte le caratteristiche descritte, e da una loro analisi possono emergere interessanti considerazioni anche in relazione al caso di studio considerato in questo lavoro.

Figura 2.1: Fasi di gestione in sistemi EMR (fonte [13]).

2.2 SAFE

SAFE [14] (Smart Augmented Field for Emergency) rappresenta un sistema collaborativo distribuito per la gestione dei trattamenti sanitari in situazioni di emergenza realizzato in forma prototipale in un progetto di ricerca del *DISI* dell'Università di Bologna e si basa in sintesi sull'integrazione delle nuove tecnologie in ambito di realtà aumentata e *wearable computing* con sistemi multi-agente ed agenti intelligenti.

Tramite questa integrazione esso si prefigge l'obiettivo di promuovere un'efficiente coordinazione delle azioni dei soccorritori coinvolti in una missione, promuovendo così la loro collaborazione, aiutati anche dall'utilizzo di innovativi sistemi *hands-free*: la maggiore particolarità di questo sistema è che la sua realizzazione si basa sull'utilizzo di un middleware chiamato *augmented field* (AF), che fornisce supporto in termini di funzionalità per permettere all'insieme degli agenti intelligenti presenti nel sistema di sfruttare le potenzialità introdotte dalla realtà aumentata. Questo middleware costituisce quindi il componente che permette agli agenti di manipolare, condividere e osservare *entità aumentate* nel senso di risorse computazionali caratterizzate da una specifica posizione nel mondo reale: una delle sue principali caratteristiche è inoltre il fatto che al suo interno le risorse vengano organizzate secondo i principi architetturali indotti dal modello *REST*, esteso per supportare anche forme di comunicazione publish/subscribe, che induce nel sistema proprietà di distribuzione e scalabilità.

Come si può notare dalla figura 2.2, si può pensare ad un Augmented Field come un livello virtuale al di sopra di un ambiente reale, che lo *"aumenta"*

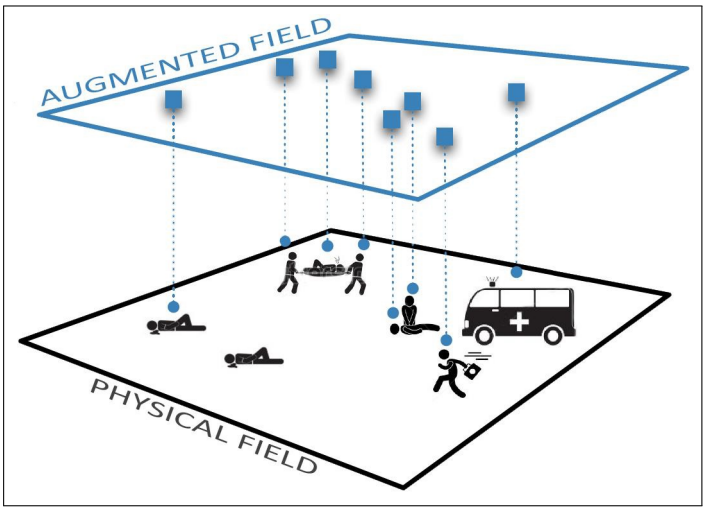

Figura 2.2: AF: una rappresentazione nel caso di SAFE (fonte [14]).

con un insieme dinamico di informazioni situate (ossia agganciate a specifiche posizioni o particolari oggetti del mondo reale) e risorse computazionali, in modo che operatori dotati della necessaria strumentazione tecnologica (ad esempio *smart glasses*) possano percepirle ed interagirci.

Attraverso l'utilizzo di questo middleware SAFE definisce quindi un approccio di realizzazione per un sistema che gestisca il coordinamento e la cooperazione all'interno di missioni di salvataggio, in cui le principali entità in gioco sono, come visibile dalla figura 2.3:

- *operatori*, membri del team di salvataggio che sfruttano le funzionalità offerte dal sistema per il supporto al proprio lavoro;
- *tag*, entità associate ai pazienti che possono comunicare informazioni riguardanti il loro stato di salute;
- *control room*, che agisce da entità di controllo e gestione dell'intera missione e ne ha una visione completa;

Esse costituiscono quindi i concetti fondamentali per definire l'architettura del sistema da realizzare, in quanto possono interagire attraverso l'utilizzo del middleware AF.

Nel prototipo realizzato per validare i concetti rappresentati da SAFE sono state utilizzate tecnologie mainstream ed innovative: gli operatori sono forniti di smart glasses Android-based che comunicano con lo smart device in loro possesso (tramite Bluetooth) in modo da fornirgli un vista caratterizzata dalla sovrapposizione delle informazioni utili alla scena reale, mentre per

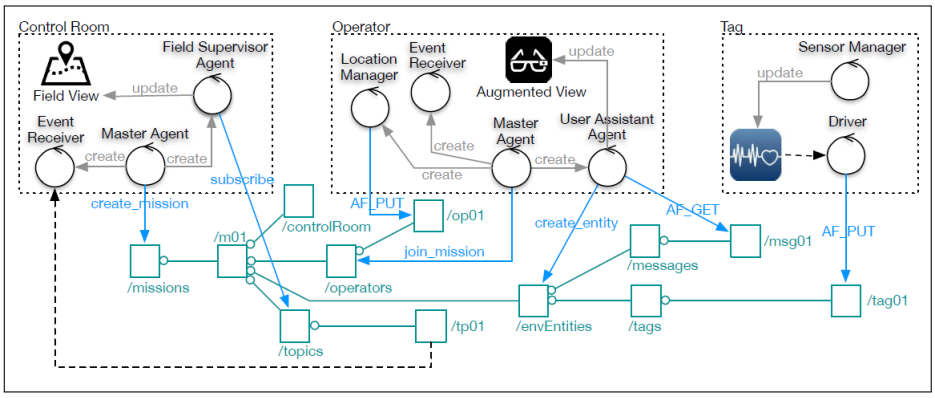

Figura 2.3: Architettura di SAFE.

la control room sono state sfruttate tecnologie web standard come HTML5 e Javascript, con il quale è stato possibile implementare anche funzionalità per la gestione della missione tramite l'interazione con una mappa che mostra la posizione degli operatori e dei pazienti. L'interazione con le risorse nel sistema è resa possibile grazie all'utilizzo di un servizio RESTful, in corrispondenza con il middleware AF descritto; inoltre, per aggiungere funzionalità di routing e consegna degli *eventi* generati all'interno del sistema, anche in ottica dell'estensione verso il paradigma publish-subscribe, è stato utilizzato il particolare MOM RabbitMQ.

Questo sistema rappresenta un esempio di come si possano coniugare le innovazioni tecnologiche in ambito healthcare, fornendo una buona analisi delle caratteristiche che dovrebbe possedere una piattaforma in grado di gestire il soccorso in situazioni di emergenza e proponendo anche una sua architettura in forma di middleware che abiliti la complessa cooperazione all'interno delle missioni di soccorso; inoltre considerando i recenti sviluppi in ambito di realtà aumentata, esso si propone tramite il concetto di "mondi aumentati" di rendere ancora più agevole e fruibile l'accesso alle informazioni da parte degli operatori.

2.3 UbiMedic

UbiMedic [15] costituisce un esempio di design per un framework basato sul concetto di agente software per l'implementazione di servizi di monitoraggio delle condizioni dei pazienti, anche in scenari di soccorso: l'astrazione di agente, grazie alle sue caratteristiche di autonomia, risulta essere di grande utilità nell'affrontare la realizzazione di sistemi distribuiti a larga scala ed eterogenei, quali quelli necessari ad affrontare le situazioni di emergen-

za, in cui fault tolerance, affidabilità e distribuzione sono solo alcune delle caratteristiche chiave.

UbiMedic non rappresenta quindi un'applicazione, ma una piattaforma per la realizzazione di applicazioni in ambito healthcare, dal momento che fornisce un middleware di supporto a diversi tipi di servizi fra i quali alcuni di tipo *user application*, costituenti i componenti da realizzare per ciascuna applicazione particolare, e altri *built-in* di utilità più generale, come quelli per accedere a dispositivi medici, ad esempio per la misurazione dei parametri vitali. Il middleware si basa, per la sua realizzazione, sull'utilizzo di layer sottostanti quali la JVM e JADE[1], una piattaforma ad agenti che può essere utilizzata anche in ambito mobile: dall'analisi dei servizi da esso offerti può essere ricavata una buona overview sui principali componenti necessari alla realizzazione di applicazioni in ambito medico, che comprendono, tra gli altri, anche un gestore di eventi intesi come cambiamenti del contesto.

Proprio la *context awareness* è una delle proprietà che risultano fondamentali per raggiungere gli scopi che si prefiggono i sistemi in ambito medico: questa importanza è sottolineata, in questo sistema, dalla presenza di un componente dedicato alla gestione del contesto e ai suoi cambiamenti per ogni entità presente, che ne mantiene le informazioni e le rende disponibili al resto del sistema. Proprio la gestione degli eventi è una delle difficoltà maggiori da affrontare in quanto bisogna trovare un trade-off fra l'efficienza della soluzione e la sua complessità: se da una parte infatti un'implementazione centralizzata potrebbe essere inefficace presentando dei colli di bottiglia, una totalmente distribuita potrebbe essere troppo complessa da gestire, così una combinazione dei due approcci, basata sul principio di località, è stata adottata, in cui la gestione degli eventi è divisa in aree all'interno delle quali è centralizzata.

Lo scopo di questo lavoro è comunque quello di presentare un diverso tipo di approccio alla costruzione di sistemi per la gestione delle emergenze utilizzando come elemento innovativo il paradigma degli agenti, noto a livello di ricerca: da questo punto di vista, esso fornisce una buona analisi delle caratteristiche che un'architettura per questo tipo di sistemi deve possedere, scendendo anche nei dettagli di alcuni dei suoi componenti.

2.4 WORKPAD

Il sistema rappresentato da WORKPAD [16], inserito nel contesto dell'omonimo progetto di ricerca europeo, propone lo sviluppo di un'infrastruttura peer-to-peer per promuovere la collaborazione del personale impiegato nelle

[1] `http://jade.tilab.com/`

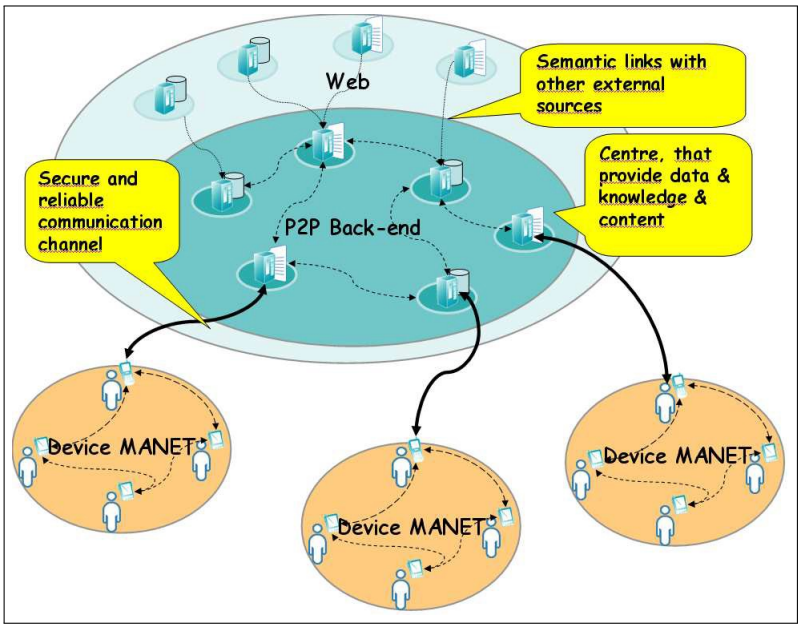

Figura 2.4: Framework di riferimento per le operazioni in WORKPAD (fonte [16]).

operazioni in scenari di emergenza o crisi dovute a disastri, in cui diversi gruppi, anche facenti riferimento a differenti organizzazioni, devono portare avanti determinate mansioni verso uno scopo comune. Inquadrando il sistema all'interno della più ampia categoria di quelli dedicati ad EMR, si può notare come esso si concentri sulla fase di risposta alle emergenze, in particolare nel breve periodo, fornendo le tecnologie informatiche e per le comunicazioni necessarie alla collaborazione degli operatori sul campo: sebbene si riconosca che ogni team ha il suo corso di azioni da effettuare, l'obiettivo è quello di raggiungere un livello di coordinazione che, attraverso l'*interleaving* di queste attività permetta di rispondere il più efficacemente possibile all'emergenza.

In particolare, nel framework di riferimento considerato per le operazioni in questi scenari (rappresentato in figura 2.4) si possono riconoscere due differenti livelli:

- un livello *peer-to-peer back-end*, in cui ogni peer fa riferimento ad una particolare organizzazione ed è costituito in prevalenza da istanze tradizionali di server, eventualmente disposte in cluster o grid, che interagiscono tra loro con un modello P2P, fornendo ad esempio servizi di integrazione;

- un livello *peer-to-peer front-end*, in cui i dispositivi mobili degli operatori sono connessi tramite reti ad hoc (ad esempio MANET) e nel quale

l'adattatività gioca un ruolo fondamentale, soprattutto nei confronti dell'attività da svolgere e della disponibilità di connessione.

Nella sua architettura, l'approccio P2P diviene di fondamentale importanza in quanto fornisce supporto a caratteristiche come adattatività, auto-organizzazione, fault tolerance e distribuzione: una delle principali componenti innovative proposte da WORKPAD è proprio quella di affrontare, fin dal suo design, il fatto che in uno scenario di emergenza un'infrastruttura adeguata di rete possa non essere garantita, in modo tale che la sua presenza non sia assunta per le comunicazioni necessarie alla collaborazione, e andando perciò in direzione di un approccio orientato alle MANET.

In questo senso, esso costituisce un esempio di un diverso approccio per questo tipo di sistemi, in quanto la sua particolare architettura fa riferimento al paradigma P2P come elemento fondante per garantire affidabilità e robustezza, non assumendo la presenza di una componente centralizzata né quella di una robusta infrastruttura di rete per lo svolgimento delle operazioni.

2.5 BigBoard

BigBoard [13] costituisce un esempio di sistema realizzato in forma prototipale per supportare la collaborazione distribuita sincrona fra le persone che devono gestire situazioni di emergenza, quali quelle riguardanti disastri naturali: la sua caratteristica principale è quella di permettere un agevole scambio di informazioni attraverso l'utilizzo di mappe in uno scenario *gather-and-share*, in cui non vi è un leader a capo della conferenza, ma ognuno dei partecipanti collabora al suo contenuto.

In questo modo il sistema si propone di aumentare il livello di consapevolezza condivisa riguardo alle circostanze tramite uno scambio real-time di informazioni geo-referenziate raccolte dalla moltitudine di dispositivi mobile in possesso degli operatori sul territorio: un altro componente molto importante è un'unità di controllo tramite cui si può interagire, all'interno della *conference room*, attraverso l'utilizzo di una mappa in cui le informazioni sono mostrate in livelli sovrapposti.

Un'altra particolarità risiede nel fatto che, alla configurazione della *conference room*, ad ogni utente può essere associato un differente ruolo che influenza attivamente i contenuti e le funzionalità a cui può accedere, guidando in maniera *context aware* la cooperazione.

Realizzato con l'utilizzo di tecnologie web standard come HTML 5, Javascript e Ajax, JSON come formato dei dati anche salvati in maniera persistente con l'utilizzo di MongoDB, un database NoSQL orientato ai documenti, e l'utilizzo del framework Django lato server, esso risulta compatibile con

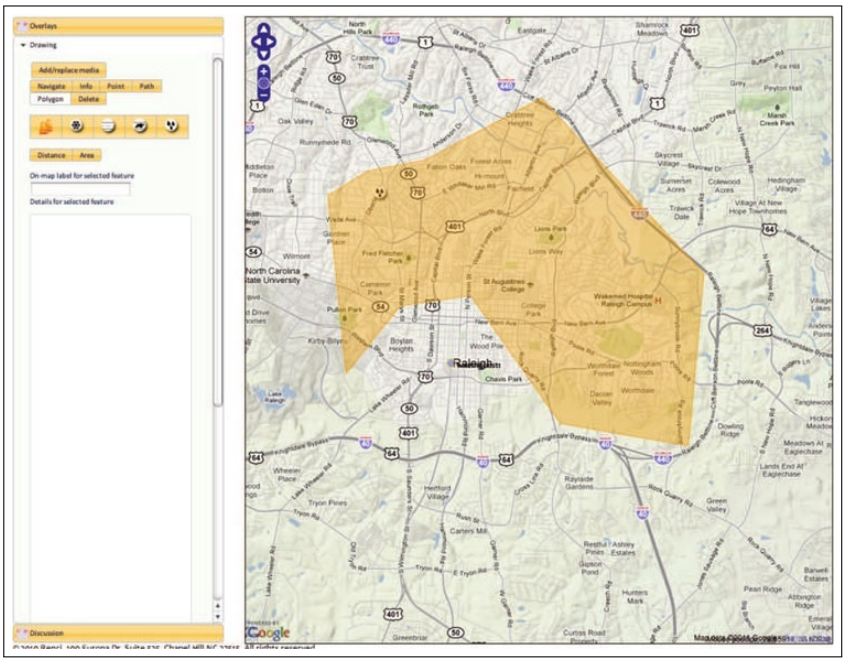

Figura 2.5: Esempio di mappa interattiva in BigBoard (fonte [13]).

la maggior parte dei browser disponibili attualmente, così come su molti dispositivi mobile: è in questo modo possibile, per le persone coinvolte nella gestione dell'emergenza, sia sul campo che nel centro per il controllo delle operazioni scambiarsi i dati riguardanti le caratteristiche del territorio e della zona in esame seguendo la metafora della teleconferenza, in cui però ricopre un ruolo centrale una mappa sulla quale queste informazioni vengono contestualizzate, di cui un esempio è raffigurato in figura 2.5. Fra le maggiori funzionalità che offre rientrano: viste condivise delle informazioni, possibilità di associare annotazioni e disegni alla mappa, integrazione in essa dei dati provenienti dai dispositivi e dai loro sensori e una chat built-in per la comunicazione real-time con gli altri collaboratori.

Sviluppi futuri di questo progetto, realizzato nel 2013, dovrebbero espandere questo insieme di funzionalità permettendo, ad esempio, in caso di mancanza di connettività, di salvare in maniera persistente le informazioni sul dispositivo mobile dell'operatore per inviarle al server in un secondo momento.

Tra i vari test eseguiti per verificare l'efficacia dell'approccio considerato ve n'è uno particolarmente interessante che riguarda la gestione di una situazione di emergenza correlata a condizioni climatiche estreme, nel caso particolare invernali: in questo caso di studio, il sistema è stato utilizzato allo scopo di condividere e collezionare in maniera agevole la maggior parte

delle informazioni utili alla decisione riguardante l'eventuale chiusura delle scuole, in cui, attraverso il feedback fornito dagli utenti, si è rivelato uno strumento di utilità.

2.6 GeoHealth

GeoHealth [17] è un prototipo di sistema a supporto della collaborazione distribuita e in mobilità pensato per assistenti sanitari che devono effettuare visite e prestazioni a domicilio ai propri pazienti: tramite l'utilizzo di un sistema informatico che gestisce dati di tipo geografico esso fornisce informazioni contestuali riguardo i pazienti, gli operatori e i loro compiti, in modo da aumentare il più possibile il livello di coordinazione sostituendo gli strumenti utilizzati in precedenza, costituiti prevalentemente da moduli cartacei o comunicazioni tramite telefoni cellulari. In particolare nello scenario di riferimento per questo prototipo devono collaborare due diversi tipi di operatori: infermieri e assistenti sanitari, che rispettivamente somministrano i trattamenti e si occupano del monitoraggio dei pazienti in una vasta area geografica.

Per supportare le loro attività il sistema fornisce quindi un servizio *location-based* attraverso l'utilizzo di una rappresentazione grafica e intuitiva delle operazioni, individuali o condivise, che devono essere effettuate, sotto forma di una mappa interattiva, tramite la quale è possibile ottenere informazioni sullo stato corrente dei pazienti, dei collaboratori, delle attività in corso e di quelle portate a termine, così come anche comunicare (ad esempio tramite supporto testuale o vocale) oppure scambiarsi attività da eseguire (tramite *drag-n-drop*).

Anche se non pensato specificatamente per gestire situazioni di particolare emergenza, a seconda delle condizioni dei pazienti da visitare possono essere generati allarmi che vengono poi indirizzati all'operatore più vicino, che attraverso la mappa può comprendere quali siano i collaboratori nei suoi pressi in modo tale da richiederne l'assistenza, se necessario.

L'implementazione del prototipo è stata realizzata tramite l'utilizzo di tecnologie Web come uno stack *LAMP* (Linux, Apache, MySQL e PHP) lato server e l'utilizzo di Ajax in Javascript per effettuare comunicazioni HTTP lato client; dal punto di vista della localizzazione è invece stato fatto uso del servizio Google Maps e di un altro particolare servizio di localizzazione nazionale per gli indirizzi civici, tramite i quali è stato possibile calcolare le coordinate delle posizioni delle entità del sistema. Da notare che, in quanto l'utilizzo di mappe può essere dispendioso in termini di banda di rete consumata, esse vengono mantenute in *cache* nel client attraverso un apposito

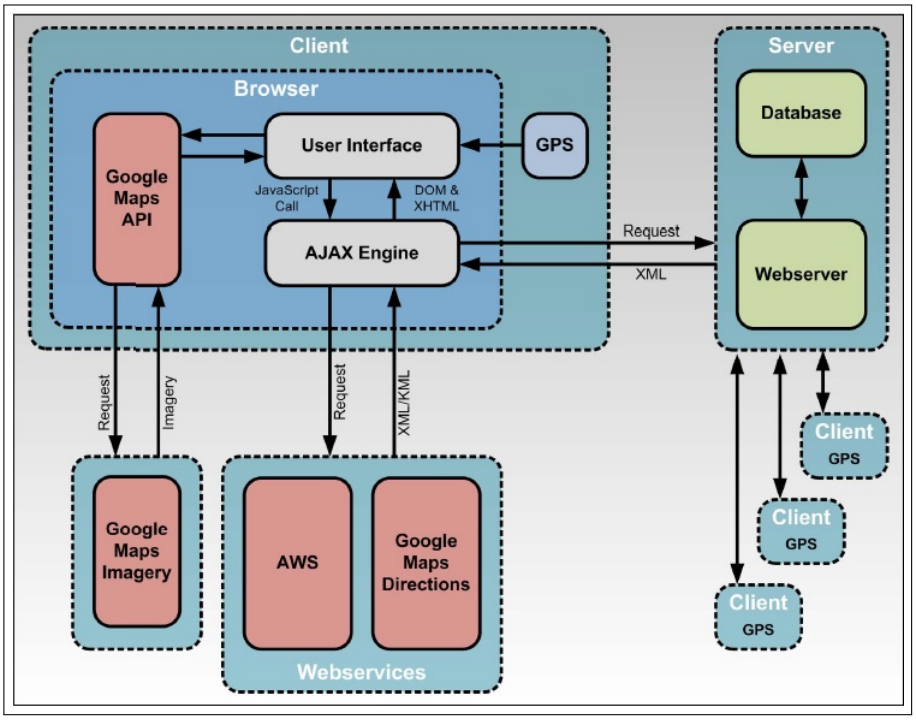

Figura 2.6: Diagramma architetturale di GeoHealth (fonte [17]).

meccanismo almeno fino a quando le informazioni non vengono modificate nel server. Dal punto di vista architetturale quindi i client si interfacciano tramite Ajax con i *web-services* relativi alla geolocalizzazione e con il web server principale del sistema, con il quale avvengono le comunicazioni riguardanti le principali informazioni di interesse nel sistema, memorizzate in maniera persistente tramite l'utilizzo di un database (figura 2.6).

Questo sistema costituisce quindi, anche se sviluppato per un caso particolare, un buon esempio di come l'utilizzo di un servizio location-aware in concomitanza con un'interfaccia user-friendly possa di fatto abilitare un maggior livello di collaborazione in ambito healthcare, e alcuni dei principi introdotti possono essere utilizzati anche per sistemi di valenza più generale nell'ambito della gestione delle emergenze.

2.7 MAETT

La classificazione delle condizioni dei feriti in seguito ad una situazione di emergenza, soprattutto su larga scala, è di fondamentale importanza per la coordinazione delle risorse e degli sforzi profusi nelle operazioni di soccorso, che puntano a salvare il maggior numero di vite: MAETT [18] (Mobile Agent

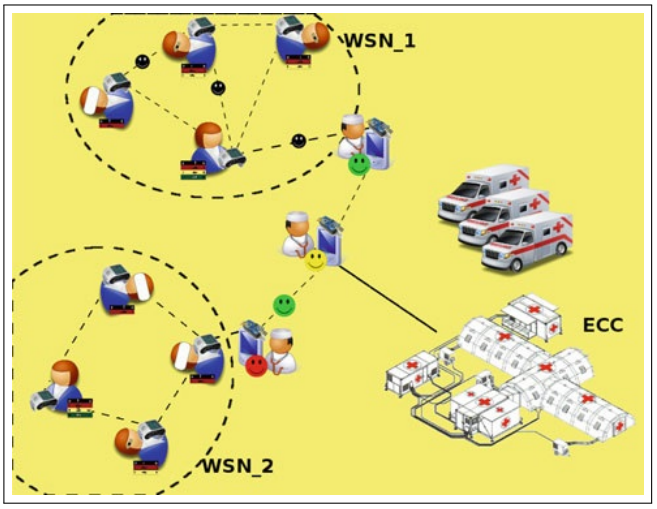

Figura 2.7: MAETT overview (fonte [18]).

Electronic Triage Tag) è un sistema che si pone l'obiettivo di automatizzare questo processo, a cui ci si riferisce con il nome di *Triage*, in questi scenari, spesso *outdoor*, in cui un cospicuo numero di feriti è sparso in un territorio di dimensioni variabili in cui non è detto che le comunicazioni siano garantite.

Attraverso l'utilizzo di un dispositivo mobile dotato di sensore GPS l'operatore è quindi supportato nelle operazioni per il triage, al termine delle quali viene comunque al paziente il tradizionale modulo cartaceo fornito però di un tag identificativo, estendendo il normale protocollo tramite il rilascio aggiuntivo di un dispositivo biomedicale in grado di monitorare i principali parametri vitali del paziente e di comunicarli in maniera wireless: la particolarità di questo sistema è infatti proprio quella di affiancare alla rete dei dispositivi in possesso degli operatori anche una *Wireless Sensor Network* formata da questi sensori associati ai pazienti.

Dal punto di vista tecnico, la dinamicità di questo sistema, che permette il controllo delle condizioni dei pazienti è gestita tramite l'utilizzo di due piattaforme ad agenti:

- *Agilla*[2], per la gestione dei sensori biomedicali all'interno della WSN per il monitoraggio dei pazienti, che comunicano le proprie informazioni ai dispositivi degli operatori con anche la possibilità di aggregazione di quelle disponibili sfruttando il concetto di vicinanza;

- *JADE*, per la gestione delle comunicazioni fra dispositivi degli operatori e il centro di coordinamento delle operazioni, responsabili anche di

[2] `http://mobilab.wustl.edu/projects/agilla/`

trasportare le informazioni giunte dai sensori biomedicali sul campo.

Anche se realizzato e testato in forma solo prototipale, questo esempio dimostra come l'impiego di strumenti all'avanguardia e delle innovazioni in ambito delle connessioni senza fili possano apportare miglioramenti ai protocolli attualmente adottati in ambito di soccorso, aumentando le potenzialità degli operatori che, in questo caso, possono coordinare meglio le loro azioni grazie ad un capillare monitoraggio dei pazienti trattati, in modo da focalizzarsi su quelli maggiormente prioritari.

2.8 Considerazioni generali

A conclusione della panoramica sullo stato dell'arte svolta in questo capitolo è possibile trarre delle considerazioni sugli approcci analizzati per i sistemi di gestione del soccorso in situazione di emergenza: i prototipi o le architetture esposte forniscono infatti un quadro delle caratteristiche principali che sistemi di questo tipo devono possedere oltre che dei loro elementi principali, anche se è possibile riconoscere che la maggior parte di essi affronti il problema da una particolare prospettiva, spesso volta alla realizzazione per uno specifico caso.

Dall'esplorazione condotta emerge inoltre che, nel campo di gestione delle emergenze, spesso il supporto delle tecnologie informatiche sia ancora lontano dal fornire un cospicuo apporto al livello di collaborazione interno alla squadra incaricata della missione, ad esempio promuovendo la consapevolezza delle azioni eseguite dal resto del team tramite meccanismi di condivisione delle informazioni in real-time, anche se forniscono comunque un aiuto: allo stato attuale infatti ad essere automatizzata è, nella maggior parte dei casi, la fase di raccolta ed invio delle informazioni verso un centro di coordinazione, come ad esempio avviene nei sistemi per il supporto elettronico ai triage, di cui un altro caso oltre a quello presentato in 2.7 è descritto in [19]. In questo senso ad essere guidate sono le azioni riguardanti il lavoro individuale dell'operatore, che è aiutato nel processo dal dispositivo che gli permette di seguire uno specifico workflow, ma non vi è un livello di cooperazione e coordinazione tra le parti tale da poter aumentare notevolmente l'efficienza della missione nella sua globalità, scopo per il quale devono essere messi in campo ulteriori meccanismi dedicati.

Più in particolare, nessun sistema tratta nello specifico il tema della *sincronizzazione* dei dati generati durante i soccorsi, in quanto molto spesso si fa l'assunzione che questi siano reperibili tramite la rete al bisogno: in contesti particolari come quello di un'emergenza dovuta, ad esempio, ad un disastro

naturale, non è detto che si possa fare affidamento su una robusta infrastruttura di rete per cui potrebbe essere utile mantenere in maniera persistente i dati in locale sui dispositivi degli operatori aggiornandoli nei momenti in cui la connettività torna disponibile, in modo da averli poi a disposizione anche qualora si dovesse ritornare in uno stato *offline*. Da questo punto di vista infatti i sistemi analizzati si appoggiano fortemente sulle infrastrutture di comunicazione utilizzate, cercando di mantenere la connessione anche sfruttando innovativi metodi, come quelli relativi alle MANET e agli approcci P2P ([16]) oppure cercando di minimizzare la banda utilizzata, ma nel caso il dispositivo fosse offline l'accesso alle informazioni non sarebbe possibile, andando di fatto a minare la *disponibilità* del sistema.

2.8.1 Caratteristiche comuni

Anche se gli approcci analizzati affrontano la problematica del soccorso in emergenza da differenti prospettive, è possibile individuare degli elementi e delle peculiarità ricorrenti: esse costituiscono il filo conduttore dell'esplorazione, denotando i principali aspetti di questi sistemi. Infatti in ambito sanitario l'efficienza del processo operativo risulta essere fondamentale, in quanto ogni momento è prezioso per salvare il numero maggiore di vite umane, però l'ambiente di svolgimento della missione risulta spesso sconosciuto a priori, caratterizzato da una forte **dinamicità** e variabilità a cui il sistema deve far fronte: per agevolare la coordinazione delle operazioni deve quindi essere permesso un rapido accesso a risorse condivise.

Tra le caratteristiche emerse per la maggior parte di questi sistemi, una sintesi delle principali comprende:

- **context awareness**, grazie a cui il sistema può configurarsi ed adattarsi alla variabilità delle situazioni da affrontare;
- caratteristiche **real-time**, nel senso che le informazioni devono essere disponibili nel minor tempo possibile per fornire il massimo livello di supporto e controllo delle contingenze;
- **interoperabilità** e portabilità, in quanto il sistema è intrinsecamente eterogeneo e distribuito, e l'interazione tra dispositivi di diverso tipo dovrebbe essere supportata, oltre al fatto che per ogni situazione dovrebbe poter essere utilizzato il device più adatto;
- **affidabilità** e **fault-tolerance**, soprattutto nei confronti degli errori umani a causa dell'estrema delicatezza dell'ambito sanitario.

Un'altra caratteristica riguarda l'*integrazione* di varie forme di contenuti, ad esempio per supportare la forma di comunicazione (testuale, audio, video) necessaria per la specifica situazione, senza dimenticare che tale scelta va effettuata anche in funzione della connettività disponibile.

Inoltre è possibile notare che la maggior parte di questo tipo di sistemi tratta e sfrutta informazioni geo-referenziate, risultando così in una sorta di *geo-collaborazione*, spesso ottenuta tramite l'utilizzo di mappe (come in [17] o in [13]): a questo proposito è infatti importante citare l'esistenza di particolari sistemi informativi dedicati alla gestione di dati di tipo geografico (**GIS**) che, come espresso in [20], sono molto spesso utilizzati in ambito di sistemi EMR.

Notevole importanza è data inoltre all'ambito delle **comunicazioni**, la cui qualità in scenari di collaborazione remota come quelli di soccorso è connotata da grande variabilità: poter rispondere in maniera efficace a repentini cambiamenti costituisce quindi da una parte uno degli scopi principali, ma dall'altra una grande sfida, in quanto risulta necessaria un'infrastruttura che scelga, a seconda della situazione, il canale più idoneo.

Infine non bisogna dimenticare che particolare cura deve essere riservata all'interfaccia proposta dal sistema all'utente, molto spesso non esperto di tecnologie informatiche: visto che lo scopo ultimo è quello di agevolarlo, rendendo più agile lo svolgimento delle sue mansioni, essa deve essere particolarmente intuitiva e permettere una rapida fruibilità, senza dover richiedere un particolare apprendimento; da questo punto di vista l'utilizzo di mappe tramite cui l'utente può interagire in vari modi (click o gestures tramite touch screen) rappresenta un esempio di successo.

Capitolo 3

Caso di studio: introduzione ed analisi dei requisiti

Lo scenario di riferimento su cui si basa questa trattazione è preso dal più ampio contesto di un progetto di ricerca sviluppato dal *Dipartimento di Informatica - Scienza e Ingegneria* dell'Università di Bologna nel 2015 nell'ambito del supporto di tecnologie innovative al soccorso delle persone in scenari di crisi ed emergenza in campo sia civile che, eventualmente, militare, a cui fanno riferimento anche i lavori descritti in [21] e [22].

Questo capitolo ha lo scopo di introdurre il caso di studio preso in analisi, descrivendolo dapprima in maniera generale ed introducendo le sue principali funzioni, per poi scendere più dettagliatamente nell'analisi dei requisiti che esso pone in modo tale da ottenerne una descrizione formale, sulla quale poi basare le successive fasi di analisi.

3.1 Descrizione generale

Attualmente gli operatori addetti ai soccorsi in situazioni di emergenza svolgono perlopiù operazioni indipendenti e dettate prevalentemente dalla loro esperienza o dalla pianificazione strategica iniziale, utilizzando di fatto uno scarso livello di cooperazione sia riguardo al team di cui eventualmente fanno parte ma anche allo stato globale della missione in cui sono inseriti: la causa principale è la mancanza di strumenti a supporto di tali operazioni, che facilitino e abilitino la condivisione delle informazioni non solo fra i componenti della squadra, ma anche a livello di gestione della missione nella sua completezza, offrendo nuove possibilità di coordinazione.

Il sistema preso come riferimento si colloca proprio nell'area degli strumenti informatici a supporto del lavoro cooperativo per le operazioni di soc-

corso ed assistenza sanitaria effettuate in situazione critiche da gruppi di professionisti dell'emergenza quali, ad esempio, operatori che scendono direttamente in campo come soldati o personale medico, e che coordinano, spesso da una centrale operativa di supporto, le operazioni. Gli interventi di questi operatori sono regolati e guidati da specifici protocolli poi calati in varie forme a seconda del particolare contesto in cui vengono utilizzati: attualmente, però, essi sono molto spesso forniti in forma cartacea, costringendo gli operatori a memorizzarli per poi applicarli sul campo. Inoltre, contando che devono essere utilizzati in scenari di forte criticità e quindi connotati da grande stress e pressione per gli operatori, si può notare come l'applicazione di questi protocolli nella loro attuale forma cartacea induca frequenti errori, dovuti non solo al loro periodico aggiornamento, ma anche al fatto che sempre più spesso personale non strettamente e specificatamente preparato dal punto di vista medico viene addetto a prestare i primi soccorsi sul campo.

Scopo primario di questo sistema è quindi supportare le operazioni di gestione degli interventi di soccorso dei pazienti, non solo fornendo un supporto informatico dotato sempre dei protocolli nella loro versione più aggiornata, ma guidando anche gli operatori durante il processo, rendendogli facile effettuare le giuste operazioni e difficile commettere errori. Al fine di aumentare il livello di cooperazione tra i componenti del team addetto alle operazioni esso deve inoltre garantire un'accurata gestione delle informazioni, in modo tale che possano essere memorizzate e possibilmente condivise: è infatti in questo modo che si può riuscire ad ottenere una buona collaborazione tra l'insieme degli operatori sparsi per il campo d'azione e ad aumentare la qualità dei soccorsi.

In queste situazioni di emergenza, le operazioni da effettuare devono garantire sia l'immediatezza del primo soccorso che la continuità dell'assistenza del paziente fino alla sua evacuazione in una struttura sanitaria: il sistema deve quindi mettere il soccorritore nelle condizioni di prendere decisioni ed agire con immediatezza e accuratezza al fine di salvare il maggior numero di vite umane, anche qualora egli non sia specificatamente preparato dal punto di vista medico, come ad esempio un soccorritore laico che effettua le misure di primo soccorso in attesa dei trattamenti di personale specializzato. Questo primo intervento risulta essere di grande importanza anche per la documentazione medica che produce in quanto, se accurata, può agevolare in maniera importante i successivi trattamenti medici che il paziente deve ricevere ed

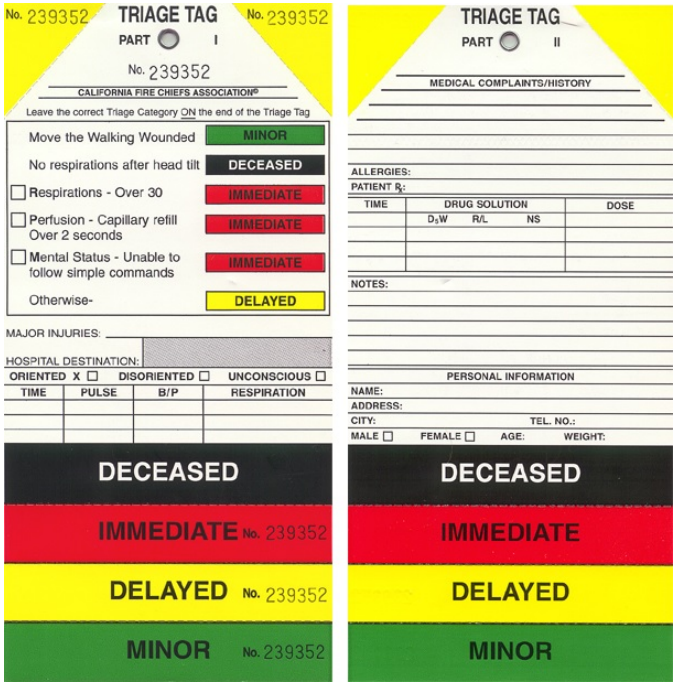

Figura 3.1: Esempio di tag cartaceo per il triage.

inoltre ne riduce il *rischio clinico*[1].

Uno fra i principali esempi di questo tipo di operazioni di primo soccorso è rappresentato dal **triage**: esso consiste sostanzialmente nella classificazione dei feriti secondo determinate classi di gravità delle loro condizioni e del loro quadro clinico.

Questa operazione, che corrisponde solitamente nella realtà all'assegnazione di un *codice colore* al paziente che ne rappresenti la gravità della situazione, ha molteplici scopi: non solo è volta a migliorare l'organizzazione del trattamento dei feriti, garantendo di somministrare prima le cure ai pazienti che più ne necessitano e di fatto snellendo i tempi di risposta dei trattamenti, ma anche quello di aumentare la qualità del servizio offerto (ad esempio valutando periodicamente le condizioni dei pazienti oppure riducendo il loro stato d'ansia). Questa prima classificazione, che può quindi essere svolta anche da personale non strettamente medico, assume notevole importanza nel processo di trattamento dei feriti sul campo, per cui una sua gestione automatizzata, che minimizzi il numero di errori commessi ed aumenti la condivisione delle informazioni può certamente rappresentare un notevole miglioramento nella

[1]Il rischio clinico viene definito come la possibilità che il paziente sia vittima di evento avverso, che gli arrechi cioè danno o disagio, imputabile, anche in maniera involontaria, alle cure mediche prestate.

qualità del servizio. Infatti una volta effettuata la prima operazione di triage è possibile continuare a monitorare il paziente eseguendo successivamente operazioni dello stesso tipo, ma anche approfondire la sua situazione clinica con altre valutazioni sanitarie che seguano protocolli differenti e più approfonditi, ad esempio tenendo conto anche dei valori di alcuni parametri vitali rilevati attraverso appositi strumenti.

Così, quando gli operatori in esplorazione sul campo individuano un soggetto bisognoso di cure sanitarie, essi possono avvalersi del supporto dei dispositivi informatici forniti loro (nella fattispecie, ad esempio, dispositivi mobile quali *tablet*) per eseguire con maggiore tempestività ed accuratezza le prime operazioni di soccorso: sono messi così in grado di classificare rapidamente lo stato di salute del paziente che può essere inoltre memorizzato in appositi *tag*, quali braccialetti NFC, in modo da poter essere poi successivamente a disposizione degli altri operatori. Inoltre queste operazioni devono essere comunicate, il più tempestivamente possibile a seconda del grado di connettività a disposizione, alla *centrale operativa* a supporto delle operazioni: tipicamente essa si trova in zona limitrofa al campo d'azione, che a causa della natura variabile delle sue caratteristiche non permette di fare assunzioni sul livello di connettività di cui possono disporre gli operatori durante la missione. In tal senso risulta necessario porre la giusta attenzione al fatto che il sistema, per essere utilizzato efficacemente in scenari del genere, deve potersi adattare alle condizioni ambientali supportando diversi metodi di comunicazione.

3.2 Glossario

In questa sezione viene proposta una breve descrizione sotto forma di *glossario* dei principali concetti a cui il caso di studio fa riferimento e che emergono, poi, nella seguente analisi delle funzionalità e degli scenari di utilizzo del sistema. Esso può essere sinteticamente descritto dalla tabella 3.1.

Considerazioni aggiuntive

In aggiunta al glossario riportato in tabella 3.1 si possono inoltre aggiungere le seguenti considerazioni, che specificano particolari aspetti dei termini in esso descritti:

- il **campo d'azione** è caratterizzato da una posizione fisica (geografica) e la variabilità delle sue caratteristiche dipende dal fatto che si possono presentare sia scenari di tipo *out-door* che *in-door*, che prevedono

Termine	Descrizione
Campo d'azione	Ambiente fisico (quindi solitamente caratterizzato da una posizione geografica) in cui avviene l'azione di soccorso e che presenta caratteristiche di natura variabile, ad esempio riguardo al livello di connettività disponibile.
Paziente (o ferito)	Soggetto che necessita dell'azione di soccorso: nel contesto di questo sistema tali azioni sono principalmente volte alla valutazione del suo quadro clinico (ad esempio triage) in modo da monitorarlo ed agevolarne le cure.
Operatore	Personale che opera sul campo d'azione ed esegue le operazioni di soccorso usufruendo del supporto fornito dai dispositivi informatici; esso è caratterizzato da un proprio ruolo (ad esempio soccorritore laico o professionista del soccorso).
Missione	Insieme di operazioni di soccorso svolte da un'organizzazione in relazione ad uno specifico campo d'azione; a queste operazioni partecipa il gruppo di operatori coadiuvato dalla presenza della centrale operativa.
Centrale Operativa	Entità di supporto alle operazioni dei soccorritori sul campo, volta alla creazione, configurazione e gestione delle missioni in maniera tale da aumentare il livello di collaborazione; posizionata generalmente in zona del campo d'azione, essa rappresenta il punto centrale di gestione e memorizzazione delle informazioni riguardante la missione, fornendo inoltre possibilità di analizzarla e visualizzarne lo stato globale.
Organizzazione	Organizzazione a cui fanno riferimento gli operatori gli operatori e le missioni supportate dal sistema; alcuni esempi potrebbero essere organizzazioni militari come eserciti o anche di protezione civile.
Tag	Entità rappresentativa delle condizioni di salute del paziente assegnato dopo una valutazione sanitaria da un operatore; ad esempio può essere rappresentato da un braccialetto NFC che memorizza l'ultima valutazione effettuata.
Triage	Metodo di valutazione delle priorità assistenziali di un paziente che ne prevede come risultato l'assegnamento di un codice colore quale codifica delle sue attuali condizioni in classi predefinite di urgenza ed emergenza in base alla gravità del suo quadro clinico.

Tabella 3.1: Tabella per il glossario.

livelli di connettività molto differenti fra loro; questi livelli possono essere classificati in *piena connettività* se si può disporre di un elevato livello di connettività che permetta di usare un qualsiasi protocollo di comunicazione, il cui esempio principale è costituito dalla rete Internet, *connettività locale* se la connessione tra operatori e centrale operativa risulta essere garantita, senza però avere accesso ad Internet, mentre *assenza di connettività* nel caso nessuna infrastruttura di comunicazione sia disponibile;

- la **centrale operativa** rappresenta il punto centrale di snodo per la gestione collaborativa e la memorizzazione delle informazioni, in quanto "media" le comunicazioni tra i vari operatori che ad essa fanno riferimento per ottenere le informazioni di interesse circa la missione che stanno svolgendo; oltre a questa primaria funzione essa abilita anche funzionalità di analisi della missione in corso permettendo una visualizzazione del suo stato, ad esempio tramite una mappa che mostra la posizione di tutti gli operatori attivi, e delle operazioni di soccorso avvenute sul campo; ad essa è affidata la creazione e la gestione delle missioni, compresa la configurazione iniziale in cui vengono associati gli operatori ed i dispositivi che ne faranno parte;

- l'**operatore** è uno dei soggetti centrali nel contesto del sistema ed è caratterizzato da un ruolo all'interno dell'organizzazione che può riflettere le mansioni a cui è addetto e abilitato all'interno della missione (per esempio soccorritore o medico); tramite il supporto fornito dal dispositivo informatico in uso è in grado di comunicare, dipendentemente dal livello di connettività presente sul campo, il suo stato (ad esempio posizione geografica) alla centrale operativa, da cui può ricavare le informazioni di suo interesse per la missione di cui fa parte, per esempio riguardanti i pazienti trattati;

- la **missione** è caratterizzata da diverse informazioni, e tra le più importanti si annoverano obiettivo della missione, posizione e dimensioni del campo d'azione cui risulta associata e informazioni temporali circa il suo inizio e la sua terminazione; ad ogni missione, creata e configurata dalla centrale operativa, viene associato un insieme di operatori facenti parte dell'organizzazione, che poi possono unirsi ad essa ad esempio effettuando il login al sistema tramite i loro dispositivi;

- il **tag** assegnato al paziente in seguito ad una valutazione sanitaria può essere un dispositivo informatico, ma nel contesto di questo particolare caso di studio è rappresentato da dispositivi passivi che sono in grado

di memorizzare informazioni e renderle disponibili agli operatori ma solamente previa esplicita richiesta (ad esempio tramite NFC), ma non hanno capacità comunicative autonome.

3.3 Analisi dei requisiti

L'analisi dei requisiti esposti nelle precedenti sezioni sotto forma di descrizione generale del caso di studio viene approfondita in questa sezione mediante due principali fasi volte a formalizzare il più possibile da una parte il dominio a cui esso fa riferimento e dall'altra le sue funzionalità, in modo da fornire delle informazioni dettagliate ed accurate circa le problematiche da affrontare nelle seguenti fasi del ciclo di realizzazione di un sistema di questo tipo ed individuarne le principali criticità che pone.

3.3.1 Modello del dominio

A partire dai concetti principali individuati per il caso di studio (e descritti brevemente nel glossario in tabella 3.1) è utile formalizzare il modello del dominio a cui esso fa riferimento: in generale, esso costituisce una delle parti *generiche* dell'applicazione ed in quanto tale altamente riusabile. Infatti può essere preso come punto di partenza per lo sviluppo del software, una robusta base sulla quale procedere con il processo di sviluppo (tipicamente incrementale, in un'ottica di *piecemeal growth*): da un'accurata e lungimirante modellazione del dominio possono infatti emergere caratteristiche non funzionali del sistema che ne migliorano la qualità, quali ad esempio l'estendibilità, in quanto promuove una maggiore facilità di aggiunta a posteriori di nuove funzionalità nel sistema, sempre che esse si riferiscano a concetti presenti nel modello.

La descrizione del dominio è formalizzata tramite il modello **REST**, che fa riferimento al concetto di *risorsa* come concetto principale: questa scelta è dettata in parte dalla natura inerentemente distribuita del sistema che ben si sposa con questo modello, i cui principi architetturali inducono proprietà tipiche dei sistemi distribuiti come la scalabilità, e inoltre anche dalla sua generalità, che permette di associare le risorse tramite *hyperlink* dinamici, senza dover sottostare a forti vincoli quali ad esempio quelli che comparirebbero in una modellazione E-R. Questa generalità nell'approccio alla modellazione del dominio permette inoltre di non fare assunzioni in questa fase di analisi circa il modello dei dati all'interno del sistema, lasciando aperte varie possibilità, tra cui quella offerta dai modelli NoSQL, oltre alla tradizionale modellazione relazionale. Questa flessibilità deve poi essere comunque gestita per garan-

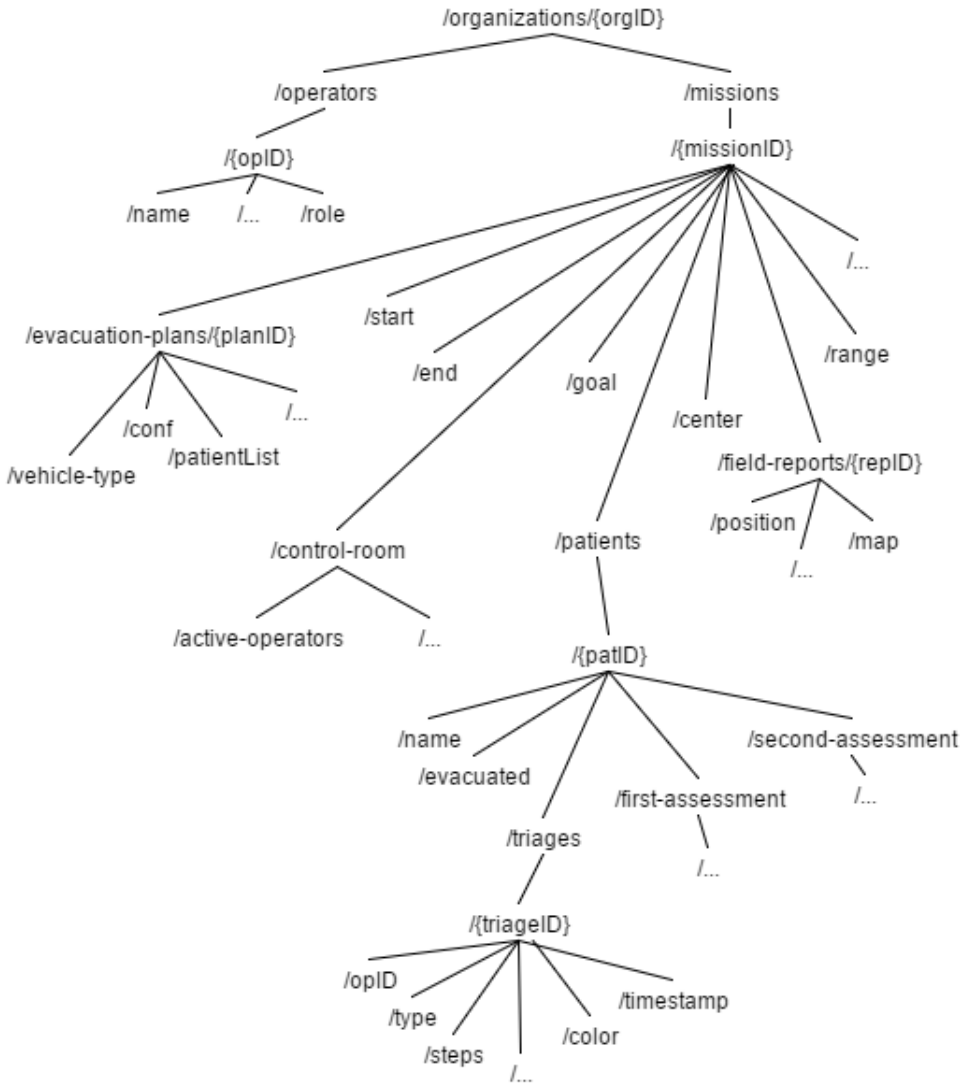

Figura 3.2: Diagramma per il modello del dominio.

tire i vincoli che i dati presentano, ad esempio facendo rispettare, a livello applicativo, determinate regole.

Il modello del dominio viene perciò presentato sotto forma di diagramma ad albero in figura 3.2: esso mostra un'organizzazione gerarchica delle risorse che, come si può notare dalla notazione degli *URI*, fanno riferimento ai concetti esposti nelle precedenti sezioni.

Ulteriori considerazioni

Dal modello raffigurato in figura 3.2 si possono notare tutte le considerazioni esposte nelle sezioni precedenti, in particolare:

- la centralità del concetto di **paziente**, che rappresenta la "risorsa primaria" trattata all'interno del sistema; è infatti attorno questo concetto che ruota il fulcro centrale del sistema, costituito dalle operazioni sanitarie effettuate dagli addetti sul campo;
- ogni paziente deve essere identificato tramite un opportuno meccanismo che ne garantisca l'univocità all'interno del sistema;
- gli operatori fanno parte di un'organizzazione, all'interno della quale possono essere associati alle missioni (ovviamente non risulta possibile associare uno stesso operatore a più missioni contemporaneamente);
- ad un paziente possono essere effettuate diverse operazioni di triage ma è esclusa la possibilità che due operazioni di questo genere vengano effettuate contemporaneamente, in quanto a livello pratico due operatori non agiscono mai contemporaneamente sullo stesso paziente;
- gli *assessment* sanitari risultano essere invece a livello concettuale uno per ogni paziente;
- ad una missione possono essere associati uno o più *report di campo*, che rappresentano concettualmente informazioni riguardanti il campo d'azione (quali posizione e mappa), inviate dall'operatore alla centrale operativa;
- ad una missione possono essere associati inoltre anche diversi piani di evacuazione effettuati dagli operatori, in cui si specifica il veicolo addetto e la lista dei pazienti evacuati;
- le risorse sono strutturate gerarchicamente secondo l'astrazione delle *collezioni* e presentano dei collegamenti impliciti (non esplicitamente sottolineati nel modello) tramite hyperlink (ad esempio l'ID dell'operatore che ha eseguito un determinato triage su un particolare paziente).

In generale, è evidente come vi siano informazioni correlate alla missione in generale, come le informazioni riguardanti il campo d'azione, e quello invece più legate ai pazienti, come i trattamenti: è attorno a queste ultime che ruota il nucleo e le funzionalità principali del sistema.

Risulta inoltre necessario notare che, per motivi di leggibilità, il modello del dominio rappresentato in figura 3.2 risulta essere semplificato in quanto non comprende tutte le caratteristiche delle risorse, ma solamente quelle salienti e rilevanti ai fini della trattazione: a tal fine è stata usata la notazione `/...` per indicare le eventuali risorse o proprietà non modellate.

Devono inoltre essere riportate alcune **assunzioni** ritenute valide a causa di vincoli o particolari caratteristiche introdotte dal determinato dominio in cui il caso di studio si colloca; infatti, poiché questo sistema è pensato per essere usato in scenari reali di emergenza per il supporto delle operazioni di soccorso, si possono trarre le seguenti considerazioni:

- due operatori non effettuano simultaneamente un'operazione (ad esempio una valutazione sanitaria) su un paziente, in quanto se un operatore nota che un paziente sta già venendo trattato, in queste situazioni si dedica agli altri feriti che necessitano della sua attenzione;

- non risulta possibile effettuare più volte l'evacuazione di un paziente in quanto una volta evacuato non è più fisicamente presente sul campo;

- si assume che, dopo un'attenta configurazione iniziale dei dispositivi prima che la missione parta, l'orario da essi segnato sia *sincronizzato*, per cui si possono ritenere significativi i timestamp associati alle operazioni effettuate come criterio di ordinamento delle operazioni (ad esempio per scegliere la più recente).

3.3.2 Scenari e casi d'uso

Le principali funzionalità offerte dal sistema al livello percepito dagli utenti sono riassunte e formalizzate attraverso il diagramma UML dei *casi d'uso* in figura 3.3, da cui si può già notare come i due principali attori nel sistema siano la centrale operativa, che modella l'entità addetta alla gestione delle missioni ed interessata a monitorarne lo stato globale, e gli operatori, ossia i componenti del gruppo fisicamente presenti sul campo per fornire il loro aiuto ai feriti.

Per comprendere meglio le funzionalità riportate nel diagramma in figura 3.3 risulta utile descrivere i principali scenari di utilizzo del sistema, che forniscono maggiori informazioni dando anche una più chiara idea delle operazioni che gli utenti possono effettuare col sistema.

Creazione di una missione Questa funzionalità, di competenza dell'entità *Centrale Operativa*, rappresenta la fondamentale fase di creazione di una

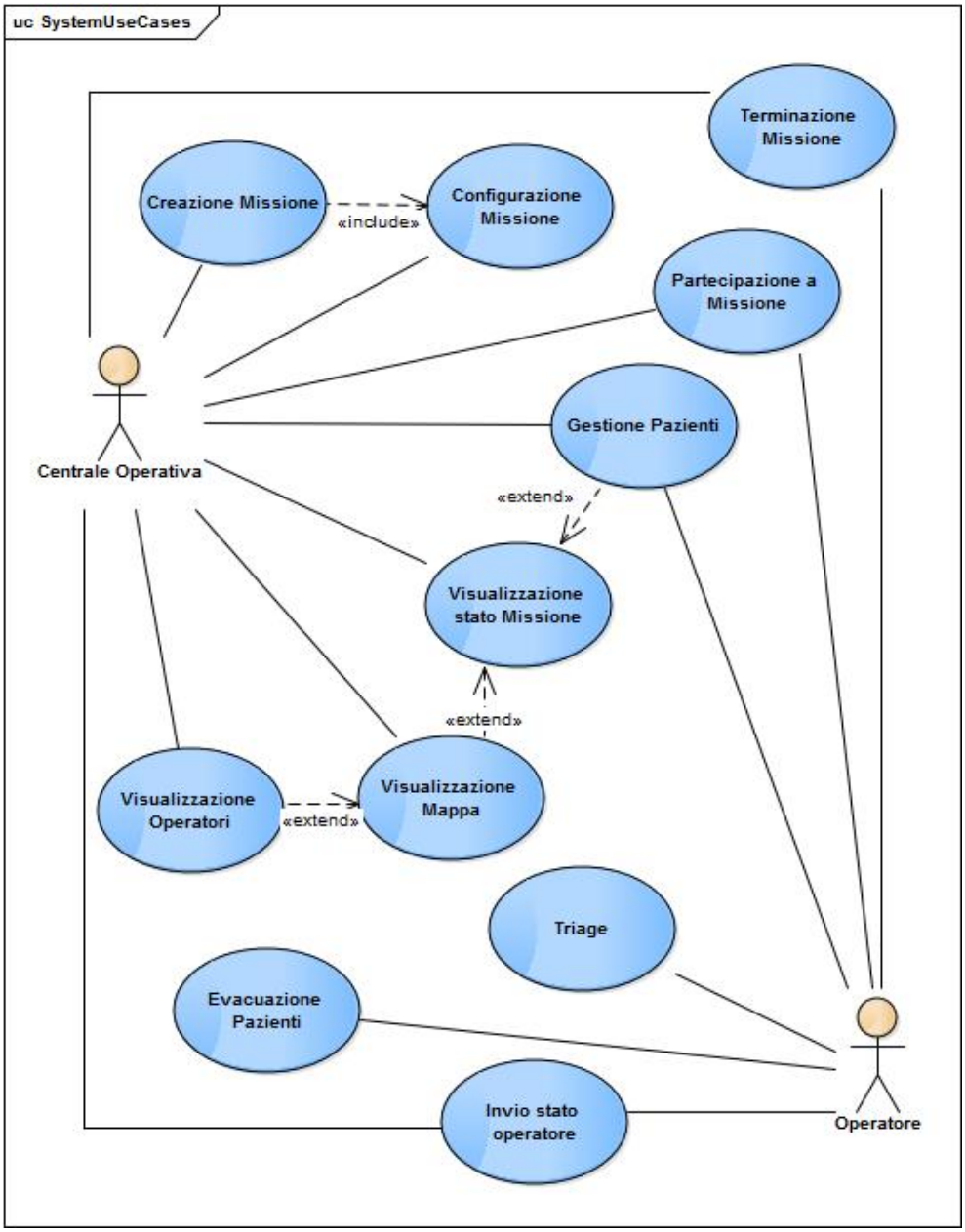

Figura 3.3: Diagramma dei casi d'uso per il sistema.

missione per un determinato campo d'azione e ne comprende anche una iniziale configurazione: all'atto di creazione di una nuova missione infatti vengono specificate le sue informazioni principali tra cui posizione e dimensione dell'area geografica a cui fa riferimento, il suo obiettivo principale e l'insieme iniziale di operatori associati ad essa, e che quindi possono poi parteciparvi.

Essendo la *missione* una delle unità di informazione principali per il sistema (come risulta chiaro dal modello del dominio in figura 3.2), questa fase risulta di importanza cruciale e strategica per il lavoro cooperativo sul campo degli operatori che ne fanno parte.

Partecipazione a una missione Un operatore, se precedentemente associato dalla centrale operativa alla missione, può parteciparvi effettuando il login di sistema tramite il dispositivo informatico a lui associato: una volta confermata la propria partecipazione può utilizzare le ulteriori funzionalità esposte dal sistema tra cui l'invio del proprio stato e delle informazioni riguardanti il proprio lavoro alla centrale operativa che in questo modo può tenerne traccia.

Invio delle informazioni riguardanti l'operatore Una volta entrato a far parte attivamente di una missione, tramite l'utilizzo del proprio dispositivo l'operatore comunica periodicamente le informazioni riguardanti il proprio stato, cioè essenzialmente la propria posizione all'interno del campo d'azione. Date le condizioni variabili di connettività all'interno del campo però questa comunicazione potrebbe non essere possibile: in questo caso l'operatore deve essere conscio del fatto che in quel momento non può interagire con la centrale operativa e che la comunicazione dovrà riprendere non appena possibile.

Visualizzazione stato della missione La centrale operativa è l'unica entità nel sistema interessata a mantenere e ad osservare lo stato globale della missione in corso: è infatti suo compito mantenere una visione completa delle operazioni svolte all'interno della missione, soprattutto al fine di aumentare la collaborazione nelle operazioni da svolgere al suo interno. É tramite questa funzionalità che un utente della centrale operativa può controllare le informazioni *real-time* che gli giungono dagli operatori sul campo: per questo motivo si può considerare che essa "comprenda" le due funzionalità estese **visualizzazione degli operatori** e **visualizzazione mappa del campo**, con le quali in pratica dalla centrale operativa si può ottenere, rispettivamente, una vista degli operatori (ad esempio quelli considerati attivi) e una mappa del campo d'azione sulla quale potrebbero comparire sia gli operatori che i pazienti di cui si conosce la posizione.

Il maggior contributo allo stato della missione viene però fornito dall'insieme delle operazioni effettuate dagli operatori nel contesto della missione, riassunte dalla funzionalità **gestione pazienti**.

Gestione dei pazienti É certamente questa la funzionalità centrale di tutto il sistema, il *core business* del sistema, in quanto esso ruota intorno ai trattamenti sanitari effettuati dagli operatori a beneficio dei pazienti individuati sul campo d'azione: essa risulta però anche una fra le più complesse dell'intero sistema, in quanto ai fini di un buon livello di cooperazione tra il personale sparso sul campo intento a prestare i primi soccorsi è necessario che le informazioni riguardanti i trattamenti effettuati siano disponibili in linea di principio a tutti gli operatori presenti sul campo, anche se poi a livello applicativo si potrebbero voler definire regole e policy tali per cui, ad esempio a seconda del proprio ruolo si può accedere solamente ad un sottoinsieme di queste informazioni.

In questo contesto risulta quindi di chiave importanza che, una volta effettuata un'operazione sanitaria questa venga comunicata in maniera agevole (ad esempio senza esplicito atto dell'operatore) e il più tempestivamente possibile, sempre considerando le condizioni di connettività, alla centrale operativa, tramite cui ne verranno a conoscenza anche gli altri operatori: questo tipo di *condivisione delle informazioni* risulta assumere un ruolo chiave per l'ottenimento di un buon livello di collaborazione e cooperazione all'interno del sistema. É quindi questa funzionalità a racchiudere le maggiori criticità inerenti al sistema, in quanto tratta il sottoinsieme dei dati a cui tutte le entità (operatori e centrale operativa) sono interessati. Promuovendo questa visione condivisa, un operatore potrebbe, ad esempio, visualizzare sul proprio dispositivo il precedente triage effettuato da un diverso operatore e prendere in base a quello le successive decisioni, effettuando un nuovo triage per continuare a monitorare un paziente oppure approfondendo la sua situazione tramite un *assessment*.

Triage Rappresenta l'operazione di associazione di un codice di gravità ad un paziente a seconda delle sue condizioni di salute: uno dei principali scopi del sistema oggetto del caso di studio è proprio quello di automatizzare queste operazioni, proponendo tramite il supporto dei dispositivi informatici gli specifici *workflow* secondo i quai classificare le condizioni dei pazienti. Come risultato, oltre alla valutazione sanitaria da rendere disponibile all'interno del sistema, l'operatore può anche associare al paziente un tag che ne memorizzi la valutazione, in modo tale che successivamente un operatore possa ricavarla tramite l'interazione con il proprio dispositivo.

In questo modo le informazioni sul triage eseguito vengono rese note agli operatori attraverso due modi: tramite la centrale operativa, che agisce perciò da "mediatore" oppure tramite la lettura dell'eventuale tag associato al paziente.

Evacuazione dei pazienti Tramite questa funzionalità un operatore può disporre l'evacuazione di alcuni pazienti, ad esempio collocando i feriti all'interno di uno specifico mezzo dipendentemente dalle loro condizioni: è importante sottolineare che, una volta terminata questa procedura i pazienti sono considerati evacuati in quanto non dovrebbero più essere fisicamente presenti all'interno del campo d'azione.

3.4 Requisiti

Le considerazioni svolte nelle precedenti sezioni di analisi dei requisiti evidenziano che il nucleo principale del sistema preso come caso di studio di riferimento è la gestione delle informazioni circa i trattamenti sanitari effettuati sui pazienti presenti nel campo d'azione: in particolare riveste un ruolo centrale la **condivisione agevole** di queste informazioni fra gli operatori attivi nella missione, mediata dalla presenza di una centrale operativa.

A titolo di esempio per questa funzionalità si può considerare il seguente scenario:

- un operatore svolge un trattamento sanitario a beneficio di un paziente sul campo e condivide le relative informazioni alla centrale operativa in maniera agevole, ossia ad esempio senza dover esplicitamente richiedere l'invio tramite l'interazione col proprio dispositivo;
- una volta arrivate in centrale operativa, le nuove informazioni vengono da essa rese note, seguendo regole e policy dipendenti dalla particolare applicazione, agli altri operatori coinvolti nella missione senza un loro esplicito atto di richiesta.

In questa maniera gli operatori sono consapevoli del lavoro effettuato dagli altri componenti del gruppo che sta affrontando la missione e possono quindi *cooperare* per raggiungere in maniera più efficace l'obiettivo.

Riguardo alla natura e ai componenti del sistema possono essere tratte le seguenti conclusioni di sintesi:

- il sistema è composto da due differenti tipi di componenti principali: la **centrale operativa**, rappresentata da un'entità centrale di controllo posizionata nei pressi del campo d'azione, a cui gli **operatori** fanno riferimento per coordinare le loro operazioni nel contesto della missione;
- il componente del sistema rappresentato dalla centrale operativa rappresenta un elemento di centralizzazione nel sistema in quanto si pone come entità che media le comunicazioni fra gli operatori presenti nel campo e mantiene uno stato globale sulla missione;

- gli operatori devono essere liberi di spostarsi all'interno del campo d'azione per fornire le cure ai feriti, quindi utilizzano le funzionalità esposte dal sistema tramite l'utilizzo di dispositivi *mobile*;

- il sistema risulta così essere inerentemente di natura **pervasiva** e **distribuita**.

Dal punto di vista *funzionale*, oltre alle considerazioni esposte nelle precedenti sezioni che individuano le funzionalità principali del sistema nella gestione delle missioni, analisi del loro stato e gestione delle informazioni riguardanti i trattamenti sanitari svolti nei confronti dei pazienti, vanno tenuti in considerazione i seguenti aspetti:

- il sistema deve presentare caratteristiche di *fault tolerance* rispetto alle eventuali disconnessioni temporanee degli operatori sul campo, e quindi fornire, in maniera trasparente rispetto ad essi, adeguati meccanismi di backup delle informazioni (e.g. salvataggio in locale delle informazioni in temporanea assenza di connettività in modo da procedere successivamente alla loro condivisione e sincronizzazione);

- la fase di connessione necessaria per le comunicazioni con la centrale operativa non deve risultare particolarmente onerosa dal punto di vista temporale, poiché altrimenti, data la frequente possibilità di disconnessione, potrebbe compromettere una rapida ed efficace fruibilità del sistema;

- deve essere prevista la possibilità di condividere anche eventuali immagini, ma la priorità per la loro sincronizzazione è minore rispetto alle informazioni, anche dipendentemente dal livello di connettività;

- poiché il livello di connettività all'interno del campo d'azione può essere variabile, particolare attenzione dovrebbe essere posta verso il *supporto trasparente* di differenti tipi di comunicazione;

- data la sensibilità dei dati trattati dal sistema, che riguardano lo stato di salute di pazienti in contesti di emergenza (ad esempio in scenari militari), il sistema deve assicurare la **sicurezza** delle informazioni, ad esempio utilizzando particolari strategie e strumenti di crittografia.

Per supportare la collaborazione e cooperazione fra gli operatori all'interno della missione il sistema deve inoltre sottostare a vincoli *real-time* nel senso che le informazioni devono essere rese disponibili appena possibile, dipendentemente dal livello di connettività: è in questo modo che si può raggiungere

un alto grado di cooperazione fra gli operatori del campo che risultano così essere consapevoli delle operazioni svolte dagli altri componenti.

Esaminando lo scenario tipico per il particolare caso di studio in analisi, si può considerare che il componente centrale operativa sia rappresentato da un sistema composto da una singola macchina (ad esempio un pc portatile dotato delle necessarie capacità computazionali e caratteristiche di robustezza). Esso deve gestire un carico di lavoro che, per quanto riguarda le missioni per le quali il sistema è stato pensato, è stato stimato in:

- circa un massimo di 15 operatori presenti contemporaneamente nella stessa missione sparsi per il campo d'azione;
- ogni operatore riesce ad eseguire al massimo una operazione sanitaria in 1 minuto, tempo che si dilata poi ad esempio a seconda del tipo di triage svolto.

Fatte presenti queste considerazioni non vanno però dimenticati aspetti **non funzionali** del sistema quali inerente *robustezza* e *affidabilità*, dato il suo contesto di utilizzo, nonché possibilità di ulteriore scalabilità qualora esso dovesse essere utilizzato nel contesto di missioni più ampie.

3.5 Porzione di interesse: condivisione delle informazioni

Il caso di studio presentato ed analizzato nelle precedenti sezioni risulta essere un progetto di ampio respiro, dal quale emergono diversi sottosistemi ognuno dei quali necessita di specifiche considerazioni riguardo alle proprie caratteristiche e alle criticità che deve affrontare. Nel contesto del lavoro di questa tesi l'attenzione viene posta alla funzionalità di **condivisione delle informazioni** all'interno del sistema, che riguarda in particolare l'area dei trattamenti sanitari effettuati sui pazienti: bisogna infatti sottolineare che, nello specifico caso considerato, le informazioni rilevanti da questo punto di vista riguardano solamente la gestione dei pazienti sul campo. Effettivamente gli operatori sono interessati a conoscere il lavoro eseguito dai propri colleghi solo indirettamente tramite la conoscenza delle valutazioni sanitarie svolte sui pazienti: è in questo modo che un soccorritore può, tramite il supporto fornito da un dispositivo informatico, fruire rapidamente dello storico delle valutazioni su un determinato paziente per decidere efficacemente le successive azioni da eseguire, come ad esempio continuare a monitorare il paziente eseguendo ulteriori accertamenti o disporne l'evacuazione.

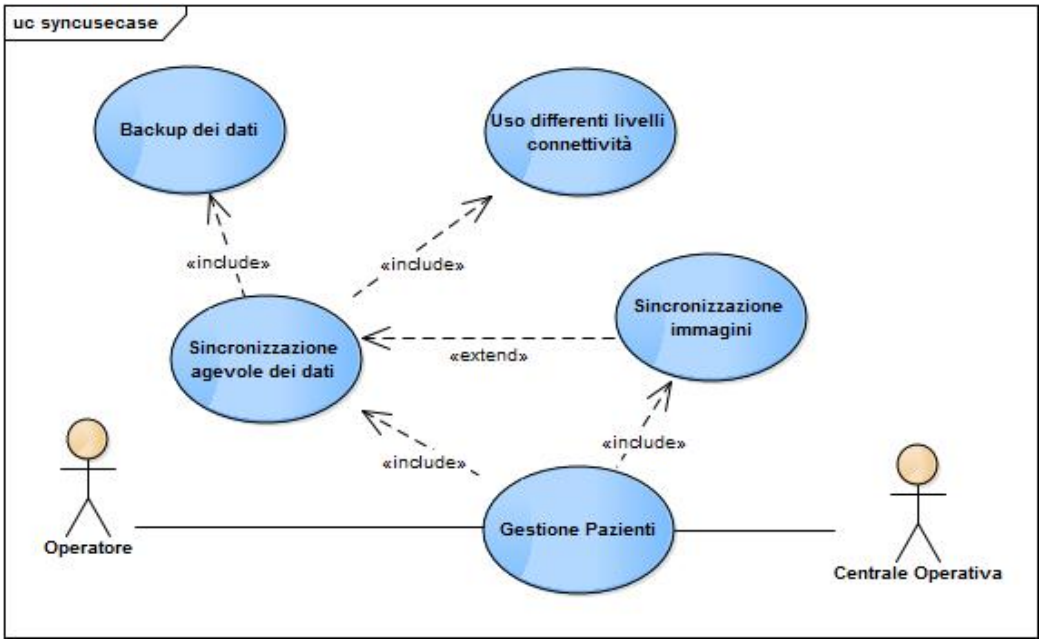

Figura 3.4: Gestione Pazienti: funzionalità interne al sistema correlate.

Dal diagramma in figura 3.4 si può notare come, ad un livello di astrazione interno al sistema considerato, la funzionalità di *gestione dei pazienti* includa quelle di sincronizzazione di tali informazioni tramite strategie che ne garantiscano la disponibilità all'operatore anche qualora sia impossibilitato a connettersi alla centrale operativa. Considerando che la condivisione delle informazioni tra gli operatori è mediata dalla presenza di un'entità centralizzata come la centrale operativa, viene quindi introdotto il concetto di **bidirezionalità** nelle comunicazioni: prendendo come esempio un'operazione come il triage, una volta conclusa dall'operatore esse deve essere inviata alla centrale operativa che in qualche modo deve renderla tempestivamente disponibile a tutti gli altri operatori presenti sul campo.

A livello di dominio le informazioni che devono essere condivise sono rappresentate dal *sotto-albero* delle risorse facente riferimento come radice alla collezione dei pazienti trattati all'interno di una determinata missione, come viene evidenziato nel diagramma ad albero riportato in figura 3.5. É relativamente a questa sola sotto-parte che le informazioni devono essere sincronizzate in maniera bidirezionale rispetto al rapporto fra operatore e centrale operativa.

Per rendere meglio l'idea si può fare riferimento alla metafora di un *archivio pazienti*: per collaborare, gli operatori devono poter accesso a questo archivio *condiviso* in cui sono raccolte le informazioni circa le operazioni

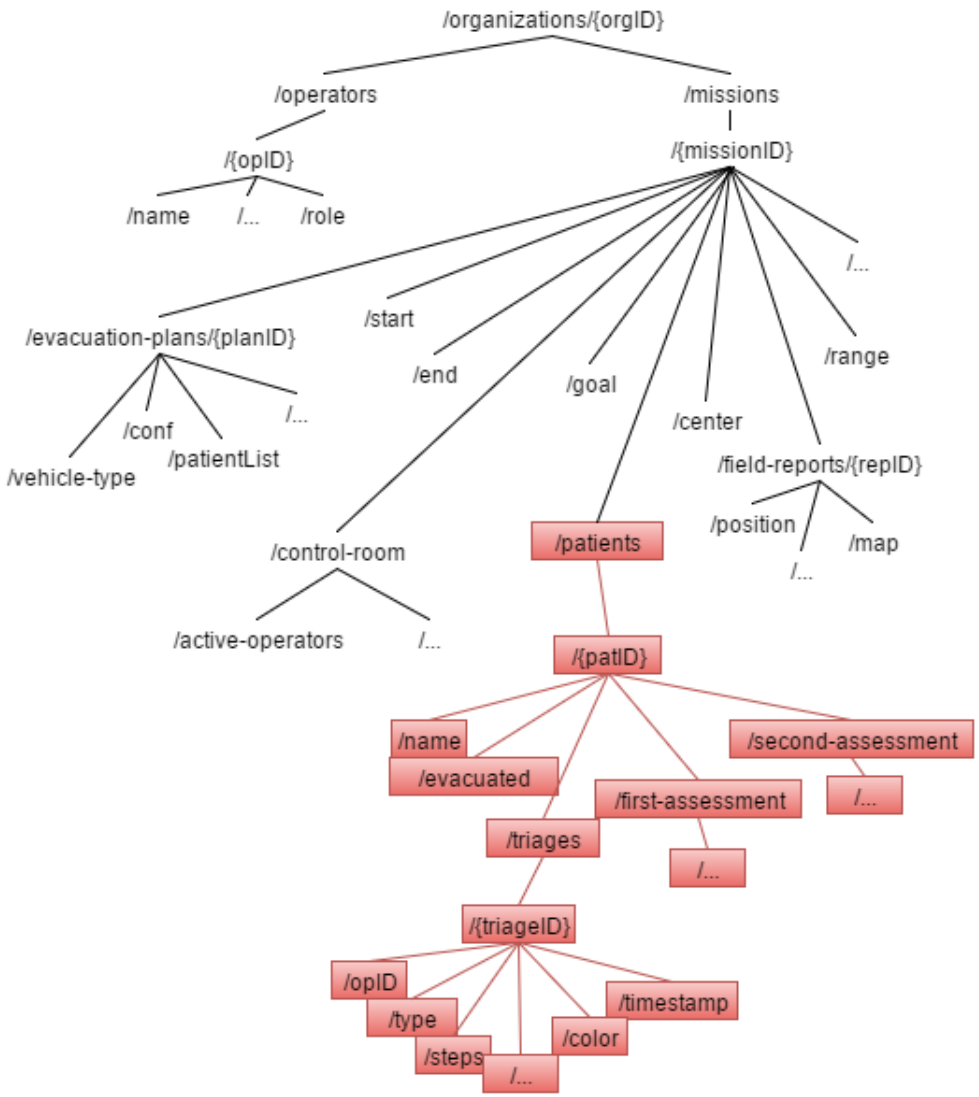

Figura 3.5: Porzione del dominio di interesse per la condivisione.

svolte riguardo ciascun paziente all'interno di una missione, in modo tale che siano facilmente e rapidamente fruibili; in questo modo un operatore aumenta in maniera sostanziale la consapevolezza del lavoro cooperativo svolto all'interno del campo d'azione, potendo inoltre migliorarne la qualità.

3.5.1 Sincronizzazione e atomicità delle operazioni

Dal diagramma in figura 3.5 si può notare che le informazioni da sincronizzare riguardano essenzialmente i pazienti e i trattamenti ad essi forniti, risulta

perciò essere di fondamentale importanza analizzarne i potenziali conflitti che possono emergere dall'interazione tra gli operatori sul campo: è proprio partendo dalle considerazioni sul particolare dominio trattato dal sistema che si possono evidenziare le caratteristiche di *atomicità* delle operazioni svolte e delle informazioni prodotte, in modo da comprendere eventuali ripercussioni sulla loro **consistenza** all'interno del sistema.

La prima considerazione da riportare è che, in linea di principio, ogni tipo di operatore potrebbe essere interessato all'intero sottoinsieme delle risorse riguardanti, mentre è a livello applicativo che si potrebbe poi decidere quali regole applicare nell'accesso a tali informazioni, ad esempio basando i permessi sul ruolo.

Inoltre va ricordato che negli scenari di utilizzo di questo sistema si esclude la possibilità che due operatori effettuino contemporaneamente un'operazione relativa ad un certo paziente poiché si intralcerebbero in vero e proprio senso fisico, oltre al fatto che in tali situazioni se si nota che un operatore si sta già prendendo cura di un paziente, ci si dedica agli altri.

Le principali operazioni sanitarie trattate nel dominio sono quindi riassunte in **triage**, **first assessment**, **secondary assessment** ed **evacuazione**; per ognuna di esse vengono riportate le maggiori considerazioni che possono essere tratte dall'analisi del dominio.

Triage L'operazione di triage risulta essere concettualmente atomica, nel senso che ad un paziente possono essere associati differenti triage, svolti in momenti differenti da differenti operatori: in questa maniera una loro sincronizzazione non causa problemi di consistenza in quanto ogni operatore, nell'effettuarlo, insiste su una differente *risorsa*. Associando un *timestamp* all'operazione, nell'assunzione che gli orologi dei dispositivi siano sincronizzati, si può inoltre ricostruire la storia delle valutazioni sanitarie effettuare su di un paziente, di cui solitamente si è interessati alla più recente. Quindi questa operazione non presenta particolari criticità dal punto di vista della consistenza, in quanto ogni operatore può eseguire questa procedura senza però modificare quelle effettuate dagli altri; inoltre un operatore può fisicamente riconoscere che su un paziente è stato effettuato un triage grazie al tag che gli viene associato, tramite cui può anche essere identificato.

First Assessment Questa operazione, effettuata sul campo e volta a valutare più approfonditamente lo stato di salute del paziente tramite l'utilizzo di determinati protocolli, è invece concettualmente unica all'interno del sistema se considerata in relazione ad un paziente, in quanto ad ogni paziente ne viene associata una soltanto. Anche se dal punto di vista concettuale viene con-

siderata come un'unica risorsa, tecnicamente essa viene modellata come un insieme di valutazioni, anche parziali ed effettuate da diversi operatori, che risultano essere atomiche: più nel dettaglio, se un operatore esegue solamente la prima metà di questa valutazione e successivamente un altro ne esegue la restante parte, il sistema memorizza due diversi documenti, lasciando al livello applicativo di visualizzazione dei contenuti il compito di "ricostruire" l'informazione prendendo per ogni campo la versione più aggiornata. Dal punto di vista tecnico questa modellazione ha inoltre il vantaggio di promuovere la storicizzazione delle informazioni e la loro *tracciabilità*, potendole ricondurre all'operatore che le ha generate.

Secondary Assessment Anche in questo caso al paziente viene concettualmente associata un'unica operazione di questo tipo, costituita da una sorta di cartella clinica a cui gli operatori possono contribuire parzialmente ad esempio effettuando degli esami strumentali sul paziente. Come per il caso precedente, anche in questo conviene tecnicamente considerare ogni singolo contributo parziale come operazione atomica, per poi ricostruire a livello applicativo l'informazione a partire dal valore più aggiornato che si dispone per ciascuna parte.

Evacuazione Come si può notare dal modello del dominio le operazioni di evacuazione non sono strettamente presenti nel sotto-albero delle informazioni da sincronizzare, ma risultano importanti in quanto possono indurre dei *cambiamenti di stato* dei pazienti: anche in questo caso però non sussistono problemi di consistenza in quanto, anche se un paziente può essere associato ad un'unica operazione di evacuazione, questo vincolo viene rispettato in quanto una volta evacuato un paziente non risulta più essere fisicamente presente sul campo, per cui non è più possibile disporne l'evacuazione. Quindi, anche nel caso in cui un operatore che non dispone di connettività e ha a disposizione informazioni potenzialmente non aggiornate e sincronizzate, non può far evacuare una seconda volta un paziente, causando un'inconsistenza nelle informazioni, in quanto esso non è più presente sul campo.

Considerazioni generali Come si può notare dalle precedenti considerazione lo scenario che più pone a rischio la consistenza delle informazioni è quello in cui un operatore offline potrebbe modificare delle informazioni non aggiornate e poi richiederne, una volta riottenuto uno stato adeguato di connettività, la sincronizzazione con le informazioni a disposizione della centrale operativa, introducendo di fatto potenziali conflitti. Nel particolare caso di studio considerato si può assumere che, data una strategia di identificazione

univoca dei pazienti (per esempio l'assegnazione di un ID univoco all'interno di una missione), le operazioni svolte non pongano criticità dal punto di vista della consistenza in quanto possono essere tutte considerate *atomiche*: considerati anche i vincoli fisici imposti dal dominio, infatti, ogni operazione di valutazione sanitaria svolta e conclusa da un operatore concorre a formare lo stato di salute del paziente, ma non può poi essere più successivamente modificata, mentre può essere sostituita nella visualizzazione del suo stato da un'informazione proveniente da una valutazione più recente nel tempo.

Sono infatti le condizioni di salute del paziente il fattore *sincronizzante* dell'intero sistema: gli operatori sono interessati alle informazioni prodotte dagli altri componenti della missione per avere una visione più accurata ed aggiornata possibile del quadro clinico dei feriti, visione che sarebbe altrimenti impossibile da ottenere senza uno strumento informatico a supporto di tale cooperazione.

In conclusione è importante ricordare che, qualora fosse disponibile un adeguato livello di connettività che permette di mettersi in comunicazione con la centrale operativa, è nell'interesse dell'operatore ottenere in maniera tempestiva le informazioni riguardo i trattamenti sui pazienti effettuati durante la missione, in modo tale da averne una visione il più aggiornata possibile nel caso in cui, a causa di condizioni sfavorevoli del campo d'azione, debba eseguire delle operazioni in assenza di connettività.

Capitolo 4

Caso di studio: analisi del problema

Il precedente capitolo introduce il caso di studio considerato per questo lavoro e ne fornisce un'approfondita analisi dei requisiti in modo tale da poter contare su di una solida base per analizzare il problema da affrontare: nel contesto di questa tesi la porzione di interesse riguarda la parte di gestione della condivisione delle informazioni tra gli operatori e la centrale operativa, componente del sistema che risulta però di fondamentale importanza in quanto costituisce il *nucleo* di funzionalità in grado di aumentare il livello di cooperazione e collaborazione tra i componenti del gruppo di soccorso, realizzando così la *vision* espressa dai sistemi per *Computer Supported Cooperative Working*, così come evidenziato nel capitolo di descrizione del background concettuale.

In questo capitolo viene quindi analizzato il problema da affrontare così da farne emergere le *problematiche* generali che pone: in questo modo è possibile, prima di passare alla fase di progettazione/realizzazione, individuare le migliori strategie da mettere in campo per ottenere un risultato che non solo soddisfi i requisiti funzionali, ma che sia inoltre di buona qualità. É infatti solo individuando i vincoli che inerentemente il problema pone che si può pervenire ad un'*architettura* che possibilmente sfrutti e riutilizzi i risultati ottenuti in passato affrontando problemi simili, ad esempio tramite l'utilizzo di *pattern*.

Più nello specifico, partendo dai requisiti applicativi presentati per il caso di studio, e in particolare da quelli riguardanti la porzione di interesse descritta nella sezione 3.5, viene individuato il *gap di astrazione* che la realizzazione di questa funzionalità presenta nei confronti del modello di programmazione preso come riferimento per la realizzazione di sistemi distribuiti aperti ed interoperabili, il paradigma *client-server* nelle applicazioni Web. Infine

vengono individuati i requisiti che un'eventuale infrastruttura deve possedere per colmare questo divario, fornendo in questo senso le informazioni fondamentali per un'esplorazione del panorama tecnologico che offre lo stato dell'arte, obiettivo principale di questa tesi e necessario per una valutazione di fattibilità.

4.1 Architettura logica

Considerando la porzione di interesse del caso di studio, che riguarda la condivisione delle informazioni riguardanti i pazienti trattati sul campo, dai requisiti applicativi (descritti nella sezione 3.5) emerge lo scenario di un sistema di natura fortemente **distribuita**, in cui sul campo sono presenti operatori *loosely coupled* che, per effettuare le operazioni di soccorso, si avvalgono del supporto di dispositivi informatici **mobile**. Si nota inoltre la presenza di un punto di centralizzazione nel sistema rappresentato dalla centrale operativa, che funge da snodo per la cooperazione fra gli operatori e mantiene le informazioni circa la missione, rendendone disponibile uno stato globale fruibile dal personale in centrale.

Già a questo livello di analisi il sistema risulta inoltre essere *data-centrico*: sono i dati infatti a guidare la cooperazione e la collaborazione fra gli operatori, che sulla loro base prendono decisioni critiche per la salute dei pazienti, quindi una loro efficace condivisione rappresenta uno degli obiettivi primari da perseguire.

Una prima overview a livello di analisi del problema della funzionalità per la condivisione delle informazioni è data dal diagramma di sequenza UML in figura 4.1, utilizzato per descriverne la dimensione di **interazione**: esso formalizza il fatto che, a fronte di un avvenuto *triage*, così come per ogni altro trattamento sanitario effettuato, questo debba essere condiviso dall'operatore il prima possibile alla centrale operativa, che a sua volta lo deve rendere disponibile a tutti gli altri operatori disponibili sul campo, eventualmente seguendo prestabilite policy.

É considerando questo scambio di informazioni che ci si riferisce parlando di **bidirezionalità** nelle comunicazioni: è comunque doveroso sottolineare che, a livello concettuale e di analisi, il verso del flusso informativo è proprio quello evidenziato dal diagramma, che, una volta arrivata una nuova operazione, parte dalla centrale operativa (e nello specifico dal suo componente addetto alla gestione della condivisione delle informazioni) per arrivare ai device degli altri operatori sul campo, ma non è detto che questa direzione rimanga tale anche dopo la fase di progetto, in cui, a seconda dei meccani-

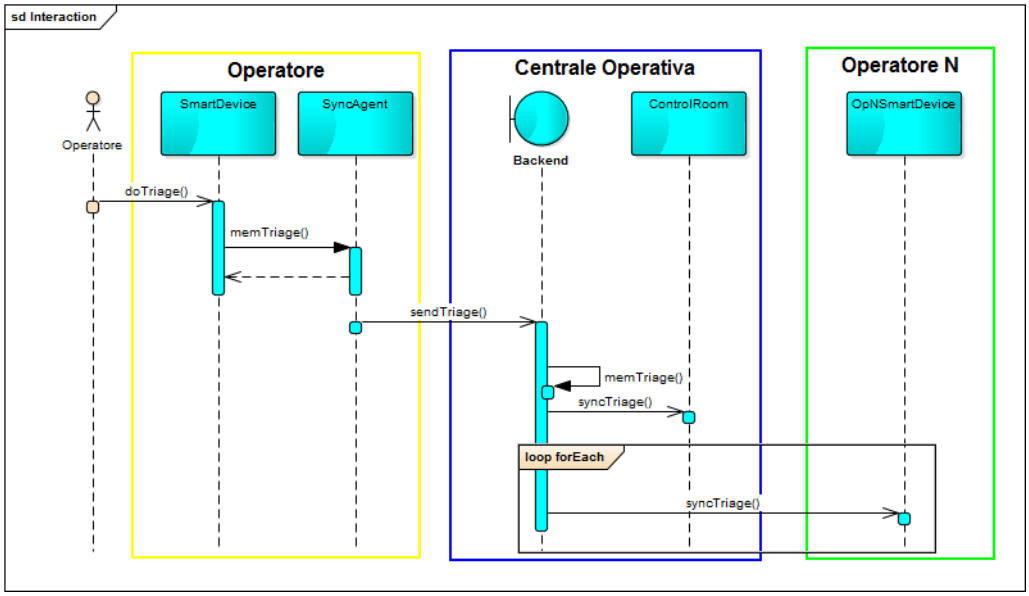

Figura 4.1: Diagramma di interazione per la condivisione delle informazioni: alto livello di astrazione in analisi.

smi e delle modalità di comunicazione usati, si decide chi veramente inizia la comunicazione.

Il diagramma in figura 4.1 costituisce quindi una prima **architettura logica** del sistema dal punto di vista della condivisione delle informazioni, da cui si nota il ruolo assunto dal *sottosistema* centrale operativa: essa rappresenta una sorta di *broker* delle informazioni dal punta di vista degli operatori sparsi in senso fisico sul campo; è infatti passando attraverso essa che le informazioni riguardanti le operazioni effettuate durante la missione pervengono i componenti del team.

Come facilmente intuibile, l'altro sottosistema principale è quello costituito dal dispositivo affidato all'operatore sul campo: già a questo livello di analisi si può notare la necessità di introdurre, al suo interno, un componente che gestisca la memorizzazione persistente e la **sincronizzazione**[1] delle informazioni, a causa del fatto che esso potrebbe trovarsi spesso *offline*, senza quindi la possibilità di comunicare con la centrale operativa. É proprio questo il requisito che maggiormente impatta il problema: dovendo tenere conto che l'operatore deve poter usufruire del supporto del suo dispositivo anche in assenza di connessione, va identificata una strategia per permet-

[1] In questo contesto, per *sincronizzazione* si intende quella riferita ai dati che ha lo scopo di mantenere una visione coerente circa un'informazione nei vari componenti del sistema distribuito, ad esempio mantenendo aggiornate le eventuali copie sparse per il sistema.

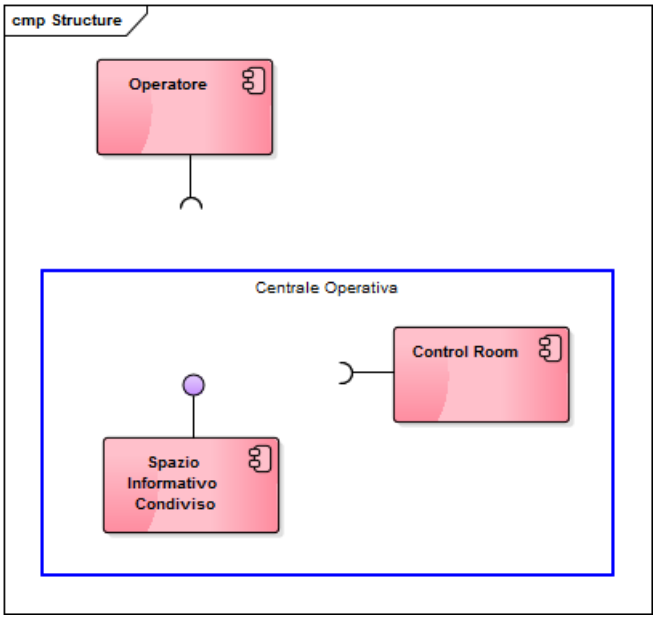

Figura 4.2: Diagramma di Struttura

tergli di usufruire delle informazioni riguardanti il lavoro cooperativo nella loro versione più aggiornata possibile, garantendone quindi un certo livello di **persistenza**.

Dal diagramma in figura 4.1 possono essere già ricavate le fondamentali informazioni anche dal punto di vista **strutturale**, che viene però approfondito da quello dei componenti in figura 4.2, da cui si può notare come nel sistema i due tipi di sottosistemi principali siano quelli già evidenziati, ossia centrale operativa ed operatori.

Dalla figura 4.2 si può però inoltre notare che la centrale operativa risulta essere un'entità composta da un *backend*, che assume il ruolo di server per il sistema, utilizzato per memorizzare le informazioni e gestire la coordinazione fra gli operatori, e una **Control Room**, che può essere ben rappresentata da un'applicazione Web che interagisce con esso per permettere la visualizzazione dei dati e dello stato della missione da parte del personale in centrale. Anche se nello scenario tipico essi risultano essere parte dello stesso sottosistema (per esempio vengono eseguite sulla stessa macchina), questa modellazione presenta grandi vantaggi, svincolando la visualizzazione dal modello dei dati e permettendo un futuro disaccoppiamento delle due entità.

In questo scenario distribuito risulta infatti immediato pensare come **ipotesi tecnologica** ad un architettura basata su servizi Web che sfrutti il protocollo **HTTP** per le comunicazioni, utilizzando perciò un modello di comunicazione basato sui messaggi: questa scelta garantisce innanzitutto grande

interoperabilità, in quanto la maggior parte dei dispositivi la supporta, oltre a promuovere il disaccoppiamento dei componenti. Inoltre il paradigma *client-server* indotto dall'utilizzo dei servizi Web ben si addice allo scenario descritto, in cui è presente un punto di centralizzazione rappresentato dalla centrale operativa.

4.2 Abstraction gap

A questo punto dell'analisi è quindi possibile individuare le maggiori *problematiche* emergenti dal caso di studio che non trovano immediato riscontro nell'ipotesi tecnologica considerata, che possono essere riassunte in:

- **bidirezionalità** nelle comunicazioni tra centrale operativa e operatori per supportare un'efficace e tempestiva condivisione delle informazioni riguardanti i trattamenti effettuati da un operatore sul campo, che devono essere resi noti anche agli altri;

- necessità di meccanismi di **sincronizzazione** delle informazioni, in quanto gli operatori svolgono il loro compito in mobilità in aree particolari e spesso caratterizzate da connettività saltuaria, scenari in cui è molto probabile che essi si trovino spesso *offline*, ma il supporto alle operazioni, e con esso la disponibilità del sistema, deve essere comunque garantito in modo da permettergli di continuare a svolgere il proprio lavoro.

Anche considerando le semplificazioni introdotte dal particolare dominio preso come riferimento (in sezione 3.3.1), queste problematiche risultano essere decisive nel porre un **abstraction gap** non banale per la realizzazione del sistema anche utilizzando un paradigma di programmazione già volto ai sistemi distribuiti come quello affluente all'area dei sistemi per il Web.

4.2.1 Bidirezionalità

La bidirezionalità delle comunicazioni risulta evidente in fase di analisi dalla direzione dei flussi di informazione all'interno del sistema, volti a migliorare la condivisione delle informazioni e quindi la cooperazione fra gli attori in gioco: considerando però l'approccio client-server classico come ipotesi tecnologica, che presenta solamente comunicazioni iniziate dal client, si nota come esso non supporti questo tipo di astrazione, con la risultante che una comunicazione *server-initiated* risulta non essere così banale da realizzare.

A questo proposito è possibile individuare, già a livello di analisi del problema, che potrebbero essere utili per colmare questo gap di astrazione strumenti e tecnologie che supportino efficacemente il pattern *publish-subscribe*, il cui approccio potrebbe semplificare la gestione dello scambio di informazioni in un contesto distribuito: in ambito web, ad esempio, le tecniche raggruppate nella categoria *push* (note anche come stile di programmazione *Comet*), possono abilitare comunicazioni iniziate dal server senza esplicita richiesta del client, dando la possibilità di realizzare una prima forma di interazione push.

Bisogna inoltre tenere presente che questa bidirezionalità risulta essere necessaria a livello *concettuale*: dal punto di vista progettuale questa interazione potrebbe essere realizzata attraverso un meccanismo di *polling* da parte del client, che adotterebbe in tal modo uno stile di programmazione che ricade nella categoria *pull*. Questa considerazione è doverosa anche considerando i particolari scenari che il caso di studio specifico si propone di affrontare, in cui tra un trattamento e l'altro eseguito da un operatore possono intercorrere alcuni minuti e le condizioni del campo influiscono sul livello di connettività: le scelte specifiche ai meccanismi relativi alle comunicazioni in rete devono essere ponderate e prese anche tenendo conto che è preferibile una soluzione leggera dal punto di vista del carico sull'infrastruttura di rete.

4.2.2 Sincronizzazione

Dal punto di vista della sincronizzazione delle informazioni fra le entità distribuite nel sistema è invece opportuno ricordare che la sua necessità si manifesta a partire dallo scenario di operatività offline richiesto agli operatori: essi devono poter infatti essere messi nelle condizioni di svolgere al meglio i trattamenti sanitari riservati ai pazienti e ai feriti sul campo anche qualora non possano usufruire della connettività.

A differenza del caso in cui la connettività è garantita, in cui quindi l'operatore potrebbe usufruire delle informazioni al bisogno con l'utilizzo della rete e viceversa inviarle agevolmente alla centrale operativa, in un contesto applicativo dove *fault* riguardanti la rete sono frequenti per mantenere un alto grado di *disponibilità* (**availability**) il sistema deve sicuramente esporre caratteristiche di fault-tolerance, ad esempio introducendo delle repliche locali dei dati sui device degli operatori sui quali essi possono operare anche a fronte di una situazione di disconnessione dal sistema. Tale considerazione porta all'introduzione di meccanismi di gestione e sincronizzazione di queste repliche per garantire un livello di consistenza delle informazioni adeguato allo scenario delineato dai requisiti: trattando informazioni di primaria importanza per la salute dei pazienti, avere delle informazioni di qualità che riflettano in

ogni momento il reale stato del sistema, ed in particolare delle cure sanitarie prestate durante la missione, risulta fondamentale. Anche se nel particolare caso, come descritto in 3.5, grazie a delle specifiche proprietà indotte dal dominio non dovrebbero essere presenti particolari criticità riguardanti la consistenza e la coerenza delle informazioni anche nello scenario in cui due operatori offline stiano eseguendo le proprie operazioni, scongiurando così a priori la presenza di conflitti visto che essi non modificano mai direttamente la stessa risorsa, è necessario introdurre una strategia che sincronizzi i dati presenti localmente sul dispositivo dell'utente con quelli resi disponibili dalla centrale operativa, eventualmente secondo policy e regole definite a livello applicativo (ad esempio basate sul ruolo o sulla posizione del personale sul campo). Va inoltre tenuto conto che il meccanismo adottato dovrebbe esibire anche un certa *flessibilità* rispetto alle risorse da sincronizzare, in quanto per esempio si potrebbero volere rendere disponibili in maniera prioritaria le informazioni più sensibili ed importanti, come i dati relativi ai triage effettuati, mentre solo in un secondo momento quelle ritenute di secondaria importanza, quali ad esempio eventuali immagini, che risulterebbero pesanti anche dal punto di vista dell'infrastruttura di rete.

In questa maniera è possibile per un operatore rendere noto alla centrale operativa il proprio lavoro e ottenere informazioni circa quello degli altri indirettamente attraverso lo stato dei pazienti, aumentandone in questo senso la loro *awareness*: è evidente che da questo livello di condivisione non può che beneficiare la cooperazione globale tra i componenti del team all'interno della missione, adempiendo ad uno degli scopi principali dei sistemi informatici a supporto del lavoro cooperativo. Inoltre il personale fisicamente sul campo può in questo modo lavorare indifferentemente sia in scenari offline che online, con la differenza che, in caso di disconnessione, stato di cui comunque deve essere conscio, potrebbe non avere a disposizione i dati più aggiornati disponibili all'interno della centrale operativa.

4.3 Verso un'infrastruttura

A fronte del gap di astrazione evidenziato, risulta evidente e necessaria l'introduzione di nuovi concetti e metafore che semplifichino il progetto di un sistema di questo tipo: si possono infatti individuare delle funzionalità che sarebbe auspicabile demandare un livello *infrastrutturale*, una sorta di **middleware** che, innalzando il livello di astrazione sul quale andare a costruire l'applicazione, ne semplifica di fatto il design e la realizzazione.

Per questo motivo risulta utile separare, a partire dai requisiti applicativi presentati, quelli che si potrebbero definire *infrastrutturali*: essi evidenziano

dunque dei *vincoli architetturali* sulla base dei quali è poi possibile individuare le astrazioni che più si addicono al sistema.

Tenute presente le considerazioni finora descritte, i requisiti di un livello infrastrutturale da utilizzare nel sistema per supportare la condivisione delle informazioni possono essere riassunti in:

- **trasparenza**, in quanto deve gestire il processo di scambio, gestione e sincronizzazione delle informazioni generate dalle entità del sistema in maniera trasparente rispetto all'utente, in modo che la sua interazione col dispositivo non sia appesantita e il suo lavoro non ne risenta; condizione essenziale è infatti quella che questo meccanismo risulti **agevole** per l'utente, e ad esempio non richieda un suo atto esplicito per essere attivato;

- **elevata disponibilità**, in quanto gli operatori devono poter accedere alle informazioni anche quando le comunicazioni con la centrale operativa non risultano disponibili, inducendo a mantenere delle copie locali (**repliche**) di tali dati;

- **network awareness**, in quanto l'infrastruttura deve fornire un livello di astrazione che permetta all'utente di lavorare anche senza connettività, ma deve fare in modo che egli sia conscio di questa condizione;

- **flessibilità**, poiché eventualmente si potrebbero prevedere regole a livello applicativo che definiscono dei livelli di accesso alle informazioni, ad esempio basati sulle informazioni del particolare operatore, la cui introduzione deve risultare agevole;

- supporto per **dispositivi mobile**, requisito di fondamentale importanza in quanto si tratta di un sistema distribuito pervasivo, in cui gli operatori fanno affidamento su mobile device per il supporto al proprio lavoro.

In più, data la particolare natura del sistema, che si colloca tra i sistemi per la gestione della salute dei pazienti, va posta attenzione anche ad altri requisiti che non riguardano strettamente funzionalità del sistema, ma che risultano comunque di importanza quali la **sicurezza** delle informazioni, ad esempio prevedendo opportune strategie di backup o crittografia.

Astrazione fondamentale di questo middleware risulta quindi essere quella di uno *spazio informativo condiviso*, in particolare calato nel contesto di un sistema distribuito pervasivo mobile: questo concetto può essere ben reso dall'idea di uno *stato globale* del sistema accessibile, secondo diverse policy, dalle varie entità che lo compongono: nel caso specifico questa astrazione

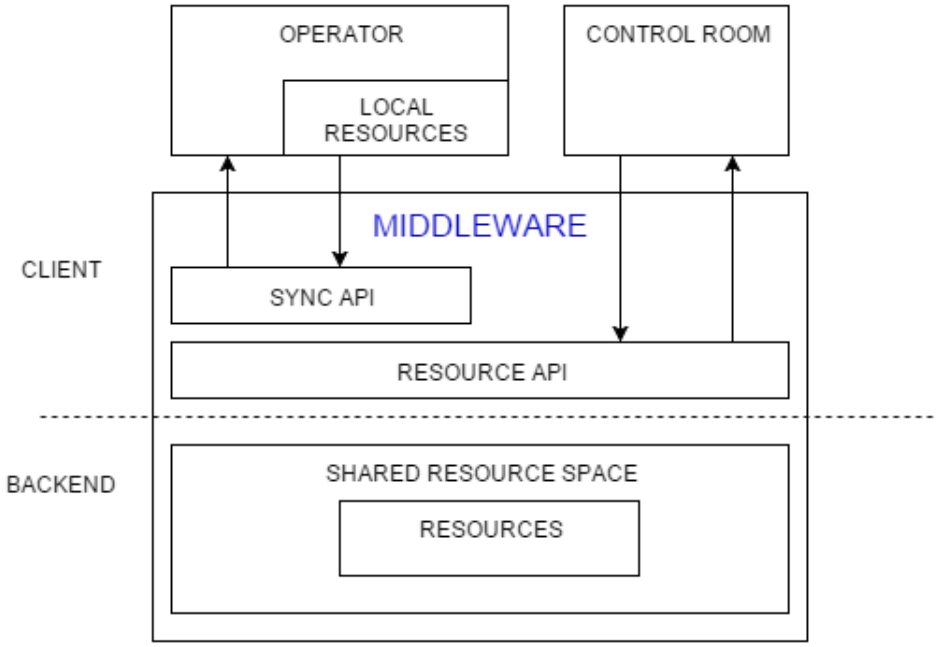

Figura 4.3: Middleware: diagramma a layer

verrebbe tradotta in un *archivio pazienti condiviso*, dal quale ogni operatore potrebbe teoricamente usufruire delle informazioni che necessità per svolgere il proprio lavoro.

Questa metafora vuole abilitare i concetti di condivisione delle informazioni e di un loro utilizzo trasparente, nel senso che all'utente deve sembrare di avere a disposizione localmente (in modo da poterne usufruire anche offline) i dati che gli servono per le proprie mansioni, anche se esse sono state in realtà generate in un diverso punto del sistema distribuito: considerate le particolari caratteristiche del sistema che presenta un evidente punto di centralizzazione nella centrale operativa, parti di questo middleware dovrebbero dunque essere disponibili per il backend in centrale operativa, mentre una serie di API dovrebbe poter essere utilizzata nelle applicazioni "client" quali quelle degli operatori, o anche la stessa control room.

La situazione viene descritta in maniera più intuitiva tramite la figura 4.3, che mostra tramite un diagramma a livelli dove si pone il middleware in mezzo rispetto alle applicazioni per gli operatori e alla centrale operativa, rappresentata da un backend.

4.4 Panorama tecnologico

Grazie alle informazioni e alle considerazioni svolte in questa fase di analisi è dunque risultato possibile individuare le **feature** che eventuali strumenti o tecnologie dovrebbero fornire per innalzare il livello di astrazione e permettere un più agile sviluppo di un sistema che mostri i requisiti presentati. A partire da queste caratteristiche è così possibile inquadrare le aree del panorama tecnologico cui fare riferimento nell'esplorazione che rappresenta l'obiettivo principale di questa tesi: tramite questa ricognizione è infatti possibile effettuare uno di **studio di fattibilità** che evidenzi le maggiori criticità e i potenziali rischi di un'attività progettuale volta alla realizzazione di questo sistema.

4.4.1 Aree tematiche di interesse

A fronte degli obiettivi e dei requisiti esposti nelle sezioni precedenti, le maggiori aree all'interno delle quali esplorare per la ricerca di tecnologie o tools utili per affrontare il gap di astrazione rilevato e realizzare l'infrastruttura che permetta di mettere in campo un componente che agevoli la condivisione dei dati in ambiente mobile distribuito sono:

- l'area afferente ai **database distribuiti** in quanto può fornire degli strumenti che promuovono *trasparenza* nell'utilizzo e nella sincronizzazione dei dati in maniera distribuita, a maggior ragione in un sistema data-centrico come quello considerato;

- l'area riguardante i **MOM** (Message Oriented Middleware), in quanto a causa della sua natura pervasiva e distribuita, è auspicabile che il sistema adotti un paradigma a **scambio di messaggi** per l'interazione fra i suoi componenti, ed essi potrebbero risultare, anche solo in parte, di aiuto nel promuovere questo approccio, fornendo anche un'alternativa all'utilizzo del solo protocollo HTTP che fornisca bidirezionalità nelle comunicazioni;

- l'area dei **Servizi Web**, in particolare quelli che fanno riferimento ad un'architettura basata su **REST**, visto che il sistema è di natura eterogenea e distribuita e il loro utilizzo, in un contesto client server, ha ormai raggiunto un buon livello di maturità anche riguardo al supporto tecnologico; inoltre questo modello, enormemente utilizzato, offre soluzioni di comprovata efficacia e scalabilità, essendo pensato specificatamente per sistemi distribuiti; le tecnologie afferenti a quest'area

possono perciò essere utili per colmare l'abstraction gap nella realizzazione di questo spazio condiviso, fornendo anche, tramite l'estensione *Comet* (chiamata anche stile push), dei meccanismi per abilitare comunicazioni server-initiated.

Partendo dalla generale tematica riguardante i meccanismi per la sincronizzazione dei dati, in particolare in ambiente mobile, questa esplorazione non si prefigge solamente lo scopo di trovare uno strumento open source potenzialmente utilizzabile, ma anche quello di vagliare gli approcci e i metodi usati per affrontare questa problematica e ricavarne così principi, o anche architetture, da poter poi mettere in campo nei riguardi del caso applicativo analizzato.

Capitolo 5

Esplorazione del panorama tecnologico

A partire dalle considerazioni emerse nelle fasi di analisi del caso di studio, in particolare rivolte alla condivisione delle informazioni in ambiente mobile remoto, sono state individuate varie aree in cui possono essere ricercati strumenti le cui funzionalità possono risultare utili nello sviluppo: affrontare il problema descritto richiede infatti meccanismi di **comunicazione** dei dati fra dispositivi mobile per permetterne una **sincronizzazione**, in modo che essi possano essere condivisi all'interno del team incaricato delle operazioni di salvataggio, raggiungendo un maggiore livello di collaborazione.

In questo capitolo viene perciò descritta l'esplorazione condotta in termini di tecnologie disponibili per abilitare le funzionalità necessarie, sottolineando per ognuna i concetti principali e le utilità che fornisce, in maniera tale da avere non solo indicazioni riguardo agli strumenti più idonei da usare, ma anche un chiaro quadro di riferimento in termini di principi architetturali di cui tenere conto.

A partire dalla descrizione della problematica della sincronizzazione dati in generale, calata poi in modo specifico in un contesto in cui l'interazione avviene attraverso l'utilizzo di dispositivi mobile, si passa alla descrizione dei *database distribuiti*, concentrandosi in modo particolare sull'esempio fornito da CouchDB, seguita poi dalla descrizione di tecnologie che abilitano la comunicazione in sistemi distribuiti ed eterogenei, come i *MOM*, di cui vengono forniti i tratti principali, ed infine da una sintetica overview dei servizi Web facenti riferimento al paradigma REST, ormai divenuto di utilizzo standard in ambito di sistemi in cui la distribuzione gioca un ruolo chiave, prestando particolare attenzione a come essi potrebbero essere utili in contesti di sincronizzazione.

5.1 Sincronizzazione dati

In informatica, con il termine **sincronizzazione dati** si intende il mantenimento di coerenza, consistenza ed integrità delle varie copie dei dati sparse all'interno di un sistema: questa problematica è nota soprattutto nell'ambito dei sistemi distribuiti in cui si rivela necessaria affinché ogni dispositivo possa usufruire della versione più aggiornata delle informazioni, anche a fronte di eventuali modifiche avvenute in altri nodi della rete.

In sostanza l'obiettivo principale della sincronizzazione dati è quello di assicurare che la *stessa versione* dei dati venga utilizzata dalle entità che compongono il sistema, propagando in tal modo a tutti i nodi che ne fanno parte le modifiche avvenute: il problema principale sussiste infatti solo in quanto diverse entità possono *accedere* alla stessa informazione contemporaneamente. Da questo punto di vista si può notare la stretta relazione di questo ambito con quello più generale della *sincronizzazione* in informatica, intesa nel senso dei *processi*, in cui ad assumere particolare importanza è l'**ordinamento** delle operazioni, permettendo ad esempio ad un solo processo alla volta di effettuare una particolare azione tramite il concetto di *sezione critica*.

5.1.1 Overview

Oggi la sincronizzazione dati è una funzionalità sempre più spesso utilizzata per fornire servizi innovativi che rispettino le richieste degli utenti: moltissimi sono gli esempi in cui questi meccanismi giocano un ruolo fondamentale, dai sistemi per il backup di file (ad esempio con tecniche RAID) ai file system distribuiti, in particolar modo quelli dedicati ai cluster, passando per la replica di basi di dati e la più comune sincronizzazione di file con servizi Cloud, di cui *Dropbox*, *ICloud* o *Google Drive* sono solo alcuni tra gli esempi più notabili.

In generale la problematica della sincronizzazione è da tempo nota e affrontata come un processo "continuo" nel senso che deve essere attentamente pianificato anche lo *scheduling* della sua esecuzione per il mantenimento nel tempo della consistenza delle informazioni: molte sono a questo proposito le sfide da affrontare nella sua gestione, tra cui la necessità in alcuni contesti di propagare i cambiamenti in *real-time* a seconda della necessità degli utenti, gestire la complessità proveniente da eventuali cambi e modifiche nel formato dei dati da sincronizzare, assicurare la qualità dei dati e la loro sicurezza, le cui policy possono variare a seconda della particolare applicazione considerata, oltre ovviamente alle performance.

Nel processo di sincronizzazione si identificano solitamente due ruoli: una sorgente (*source*) di dati, da cui estrarli e trasferirli poi ad un obiettivo (*target*) una volta trasformati nel formato idoneo; bisogna però notare che i cambiamenti avvenuti in una sorgente potrebbero anche dover essere resi noti a più target, in una relazione *one-to-many*, facendo sempre attenzione a definire quale entità ha la responsabilità di iniziare e controllare questo processo.

Principali tecniche

Dato che quella relativa alla sincronizzazione dati è una problematica dai confini piuttosto ampi, diversi sono i modi con cui classificare le principali tecniche utilizzate nel tempo, ognuno dei quali prende come riferimento un diverso fattore.

Considerando ad esempio il quantitativo di dati scambiati nel processo di sincronizzazione, due sono i principali approcci adottati [23]:

- **full synchronization**, detta anche completa, in cui ogni volta che è richiesta sincronizzazione dei dati una copia completa di essi viene trasferita dal sistema sorgente all'obiettivo; questo approccio è sicuramente semplice dal punto di vista concettuale ma ha il grande svantaggio di comportare uno scambio di un grande quantitativo di dati, che nel caso di sistemi distribuiti si traduce molto spesso in un overhead non accettabile per l'infrastruttura di rete;

- **incremental synchronization**, in cui solamente i dati che hanno subito un aggiornamento (ad esempio sono sono stati creati o modificati) rispetto allo step di sincronizzazione precedente vengono trasferiti verso il target in quello attuale; in questo approccio il trasferimento di dati risulta perciò efficiente, ma la complessità si sposta sui meccanismi per riconoscere i cambiamenti avvenuti, per i quali è spesso necessario introdurre un sistema di **versioning**.

Di questi, nel contesto di un sistema distribuito è sicuramente preferibile usare un meccanismo incrementale, servendosi di un controllo di versione che può essere realizzato tramite l'utilizzo di diversi approcci, di cui i principali sono quelli basati su:

- *timestamp*, che indicano l'istante dell'ultima modifica di un dato, e permettono quindi durante il processo di sincronizzazione di capire quali dati debbano essere effettivamente trasferiti; questa tecnica presenta il

vantaggio di non dover tenere traccia delle precedenti interazioni in termini di sincronizzazione con le altre entità del sistema, ma si basa sulla forte assunzione che, all'interno del sistema, il tempo sia sincronizzato;

- *digest* dei dati, che permettono di verificare se sono avvenuti cambiamenti grazie al confronto dei valori forniti dall'applicazione di funzioni di hash sui dati, essi rientrano

- *log* nei quali ogni cambiamento avvenuto rispetto ai dati è mantenuto, e all'atto della sincronizzazione essi vengono trasferiti in modo da poter effettuare localmente le modifiche ancora non eseguite.

Un altro modo per classificare i tipi di sincronizzazione riguarda le *direzioni* in cui essa viene effettuata: infatti a seconda del tipo di flusso dei dati essa può essere **monodirezionale** (one-way) quando si vogliono sincronizzare gli aggiornamenti avvenuti localmente a un'entità verso un'altra, o **bidirezionale** (two-way) nel caso in cui le informazioni debbano fluire in entrambe le direzioni.

Inoltre, nell'area dei sistemi distribuiti, in particolare in quelli basati sul paradigma client-server, è frequente trovare l'utilizzo dei termini *push* e *pull* per indicare, rispettivamente, la sincronizzazione dei cambiamenti avvenuti localmente al client verso il server e viceversa.

5.1.2 Sincronizzazione e Mobile Computing

L'utilizzo di dispositivi mobile all'interno dei sistemi distribuiti è diventato al giorno d'oggi pratica comune, soprattutto nel contesto di quelle applicazioni in cui si vuole fornire un facile accesso ai dati, potendone in tal modo usufruire secondo la formula *"anywhen, anywhere"*[24]. D'altro canto bisogna però tenere conto delle limitazioni che essi introducono soprattutto dal punto di vista della connessione di rete, visto che sono frequenti infatti le situazioni in cui un mobile device si trova offline, senza poter comunicare: poter accedere ai dati anche in queste situazioni rappresenta quindi una funzionalità molto importante per applicazioni che puntano ad essere sempre disponibili, senza far percepire all'utente eccessivi cali nella *user experience* a causa dei problemi di connettività.

Emerge così la necessità di mantenere una copia, anche parziale, dei dati di interesse sulla memoria locale dei dispositivi mobili in modo che essi possano accedervi senza utilizzare la rete[25]: utilizzando la propria copia locale è possibile fornire un certo grado di *trasparenza* nell'accesso dei dati, con i quali si può interagire senza realmente conoscere se fisicamente provengono dalla rete oppure no. Questo scenario introduce però ulteriori complicazioni

al già difficile compito della sincronizzazione dati, in quanto è possibile che due entità del sistema modifichino contemporaneamente lo stesso dato, ad esempio mentre entrambi erano disconnessi, mettendo così a repentaglio la consistenza delle informazioni.

Garantire la possibilità a multipli utenti di effettuare concorrentemente operazioni sullo stesso dato è complicato, a maggior ragione se gli è permesso farlo anche quando non possono comunicare in rete: assicurare le proprietà transazionali diventa problematico e risulta necessario introdurre meccanismi per il controllo della concorrenza. Le principali strategie da questo punto di vista risultano essere due:

- **locking ottimistico**, in cui le operazioni non vengono bloccate ed il controllo riguardante le eventuali violazioni di integrità dei dati viene effettuato alla loro terminazione; nel caso in cui una tale violazione venga riscontrata l'operazione non viene ritenuta valida, è necessario effettuare un *rollback* che comporta un overhead in termini della sua ri-esecuzione e della gestione dei potenziali conflitti sollevati;

- **locking pessimistico**, in cui al momento di effettuare un'operazione su un dato è necessario ottenere un *lock* che verrà rilasciato al termine dell'operazione, con la possibilità che essa risulti bloccata qualora un altro utente stia già insistendo sullo stesso dato.

Generalmente, utilizzare una strategia ottimistica risulta conveniente se si ritiene poco probabile che due utenti modifichino contemporaneamente lo stesso dato, altrimenti la gestione dei conflitti che ne risulta implicherebbe un overhead tale da far preferire una strategia pessimistica, da valutare con attenzione in quanto presenta il rischio di incorrere in problemi di *deadlock*.

In scenari di mobile computing dove è permesso l'accesso e la modifica dei dati anche in modalità offline è così auspicabile adottare una strategia ottimistica che contempli però la gestione di eventuali **conflitti**: risulta necessario decidere in tal senso una policy che indichi come risolvere le situazioni in cui una modifica sulla copia locale del dato non è consistente con quella remota con la quale la sincronizzazione deve essere effettuata. Usualmente i metodi per l'individuazione e la risoluzione di questi conflitti sono basati sul controllo di versione del dato, che permette di capire se l'aggiornamento effettuato può essere considerato valido o no.

Un altro aspetto da considerare riguarda la porzione di dati sulla quale interessa effettuare la sincronizzazione: la scelta tra una sincronizzazione *totale* e una *parziale* può essere determinata dalle considerazioni dei vincoli imposti dal particolare caso applicativo, quali ad esempio limitazioni specifiche

(spazio di memoria disponibile) dei dispositivi mobili, limitata disponibilità di banda oppure particolari policy che dipendono dal contesto di esecuzione.

Infine, va considerato anche il livello di trasparenza rispetto all'utente che il processo di sincronizzazione deve mostrare, a seconda del quale devono essere prese determinate decisioni progettuali che riguardano la *user interface*: si può infatti optare per un processo sincrono che blocchi l'interfaccia fino alla propria terminazione, rendendo partecipe e cosciente l'utente, per uno di tipo asincrono che mantenga quindi disponibili le funzionalità gestendo la sincronizzazione in *background*, senza necessariamente renderlo noto all'utente, oppure per una combinazione dei due.

5.1.3 Sincronizzazione e collaborazione: Google Docs

Un esempio interessante di sincronizzazione dati applicata in sistemi a supporto del lavoro cooperativo è costituito dalla piattaforma Google Docs, una soluzione cloud SaaS (Software as a Service) accessibile dai suoi utenti tramite web per la condivisione di documenti che offre anche funzionalità di editing collaborativo real-time. Tramite l'editor messo a disposizione degli utenti è possibile lavorare sullo stesso documento in maniera distribuita, sia simultaneamente che non, mostrando così caratteristiche di una collaborazione riconducibile sia al caso sincrono che asincrono: alla base di questa applicazione vi è un algoritmo di sincronizzazione che permette alle copie del documento memorizzate lato client e lato server di tenersi aggiornate riguardo ai reciproci cambiamenti, fornendo così all'utente la possibilità di avere visione degli aggiornamenti effettuati anche dai suoi collaboratori.

L'algoritmo utilizzato, chiamato **Differential Synchronization** [26], è un algoritmo di sincronizzazione state-based che utilizza una strategia ottimistica: esso è basato su un ciclo eseguito in background di operazioni chiamate *diff* e *patch*, che consistono rispettivamente nell'individuazione delle differenze fra due file e nell'applicazione di queste differenze ad uno dei due per farli risultare uguali.

Come si può notare in figura 5.1, l'algoritmo si serve di particolari *shadow file* rispetto ai quali i cambiamenti effettuati dall'utente vengono computati grazie ad un algoritmo diff: questi cambiamenti vengono poi applicati (tramite patch) al file shadow, e trasferiti in rete affinché siano propagati alla versione del documento memorizzata lato server. Una volta applicati anche al file shadow lato server, viene calcolato il diff della nuova versione con quella che il server aveva memorizzato per trasferire al client i cambiamenti da effettuare nella seconda metà dello step di sincronizzazione.

Da questo esempio di successo possono essere ricavate idee e principi da applicare qualora si necessitasse di uno strumento di collaborazione per

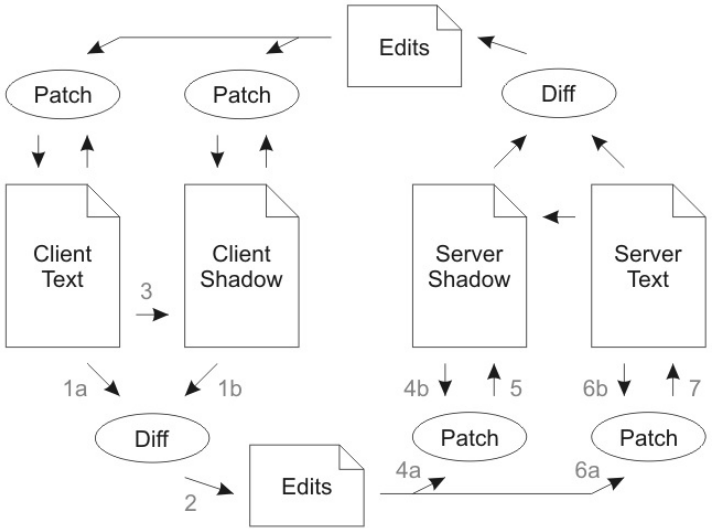

Figura 5.1: Differential synchronization: utilizzo di shadow files.

la modifica real-time, che comunque esula dagli obiettivi specifici di questa tesi che invece si vuole concentrare sulla condivisione di dati generati dagli operatori durante le missioni di soccorso in situazioni di emergenza, in cui la priorità è quella di condividere le informazioni riguardanti le operazioni svolte sui pazienti che però i collaboratori non possono modificare, ma di cui devono essere a conoscenza per coordinarsi in maniera più efficiente.

5.2 Database Distribuiti

Nel contesto di un sistema distribuito ed eterogeneo di mobile computing come quello considerato in questa tesi, un tipo di tecnologia che fornisce astrazioni utili allo sviluppo di un spazio informativo condiviso è quella che fa riferimento ai **database distribuiti**.

Tramite il loro utilizzo è infatti possibile servirsi dei dati necessari in maniera *trasparente* rispetto alla loro posizione, intesa come la macchina sulla quale sono memorizzati: l'idea di una piattaforma che, inoltre, permetta di mantenere in locale un copia dei dati di interesse che sia disponibile anche a fronte di disconnessioni e che possa essere sincronizzata una volta tornata la disponibilità della rete ricalca bene i requisiti espressi in fase di analisi del problema per un eventuale strumento di aiuto nella realizzazione di un livello infrastrutturale. Servendosi di un database distribuito l'applicazione potrebbe infatti demandare ad esso la gestione distribuita dei dati, sincronizzazione compresa: per questo motivo è necessario comprendere le loro caratteristi-

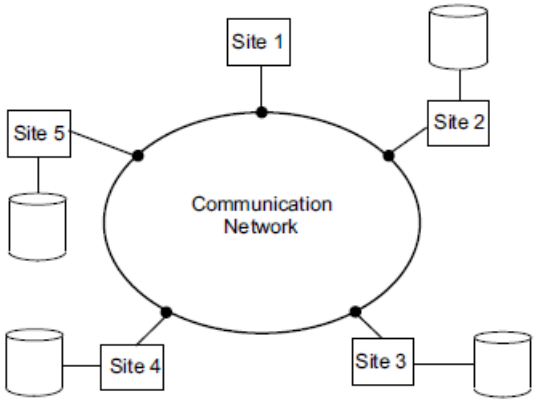

Figura 5.2: Rappresentazione visuale per un database distribuito ([27]).

che principali e ricercare se, allo stato attuale delle tecnologie, ve ne sia uno con le giuste potenzialità e funzionalità, tra le quali *in primis* va ricordato il supporto per dispositivi mobile.

5.2.1 Elementi essenziali

Un database distribuito può essere infatti definito come *"a collection of multiple, logically interrelated databases distributed over a computer network"* [27], e il software che ne permette la gestione e la rende trasparente rispetto agli utenti viene chiamato *distributed database management system* (distributed DBMS): in questa definizione si pone l'attenzione sul fatto che esso si componga di un insieme di database fisicamente distribuiti su diversi dispositivi, i cui dati sono inter-relazionati dal punto di vista logico ed hanno una struttura. É inoltre utile notare come vi sia differenza tra un database distribuito e uno *decentralizzato*, sempre costituito da una collezione di database collocati su diversi dispositivi che però non hanno accesso alla rete: anche se all'apparenza può apparire che i dati provengano da un unico database, non vi è in realtà condivisione di informazioni; per questo da un punto di vista logico un database distribuito fa più riferimento al concetto di un unico database geograficamente distribuito piuttosto che ad un insieme di database indipendenti.

A seconda delle caratteristiche dei DBMS che gestiscono le varie istanze che compongono il database distribuito, eseguite autonomamente, esso può essere considerato:

- **omogeneo**, se in tutti i nodi viene eseguito lo stesso DBMS; in questa situazione solitamente si utilizza lo stesso schema per tutte le istanze,

che in genere hanno conoscenza le une delle altre per poter collaborare nel servire le richieste degli utenti

- **eterogeneo**, se viene utilizzato un DBMS dagli altri in uno o più nodi; in questo caso è probabile anche che vengano utilizzati differenti schemi, complicando ancora di più le possibilità di eseguire query e transazioni dal momento che è possibile che le varie istanza non siano consapevoli della reciproca presenza.

Anche se un ambiente omogeneo è molto più facile da gestire, spesso le situazioni reali presentano casi eterogenei, in cui va quindi realizzata un'integrazione che consenta l'utilizzo trasparente delle risorse messe a disposizione.

Inoltre si può fare una distinzione anche fra database distribuiti *sincroni* e *asincroni*: nei primi tutti i dati sparsi per i vari nodi della rete vengono tenuti costantemente aggiornati, mentre negli altri c'è solitamente un ritardo fra modifica di un'informazione e la relativa diffusione a tutti gli altri nodi. Nei database di tipo sincrono l'utente dovrebbe ottenere la stessa risposta interrogando istanze differenti, in quanto se in rete viene modificata una copia di un dato, in linea di principio essa si dovrebbe riflettere immediatamente a tutte le altre, oppure fallire: per questo motivo essi possono risultare lenti e insoddisfacenti dal punto di vista del tempo impiegato per rispondere alle richieste degli utenti, anche se assicurano integrità e minimizzano la complessità di dover conoscere dove sono localizzate le istanze più aggiornate di un dato. Per quanto riguarda quelli asincroni invece si tollera una temporanea probabile *inconsistenza*, per esempio nei momenti che intercorrono tra una modifica e la sua propagazione, ma essi presentano di solito un miglior tempo di risposta, dato che le operazioni sono eseguite localmente e solo in un secondo momento sincronizzate, anche se si deve affrontare il complesso compito di garantire il giusto livello di consistenza e integrità fra i vari nodi.

Distribuzione

I principali metodi con i quali i dati vengono distribuiti all'interno di un database distribuito possono essere sintetizzati in:

- *replica*, che concerne la distribuzione di copie, parziali o complete, del database in vari nodi del sistema;

- *frammentazione*, a cui ci si riferisce anche col termine *partitioning*, che concerne la divisione di dei dati in frammenti che vengono distribuiti; i due tipi di partitioning solitamente considerati sono orizzontale e verticale.

Mantenere differenti repliche, o copie, di un database può avere diversi scopi fra cui l'aumento dell'affidabilità o delle performance in termini di tempo di risposta, ma la caratteristica di principale interesse nel contesto di questa tesi è l'aumento della *disponibilità*: tramite l'utilizzo di repliche infatti alcune operazioni possono essere effettuate localmente ad un nodo senza la necessità che la connettività di rete sia garantita, con l'assunzione che quando sarà nuovamente disponibile queste verranno sincronizzate nel sistema. Oltre a ridurre il traffico di rete, in quanto gli aggiornamenti vengono di solito resi noti in maniera asincroni tramite schedulazione di appositi task, questa peculiarità risulta importante nel contesto di applicazioni mobile, in cui la copertura di rete non è sempre garantita.

Comportando alcuni svantaggi tra cui requisiti in termini di spazio di memoria sui dispositivi e complessità nella gestione e nel mantenimento della consistenza e dell'integrità dei dati, i meccanismi di replica devono essere usati dopo un'attenta analisi delle caratteristiche del caso considerato: scenari in cui la maggior parte delle richieste sono *read-only* e i dati sono relativamente statici, in cui non vi sia necessità di aggiornare in real time le modifiche avvenute su un altro nodo sono quindi indicati per il loro utilizzo.

Per quanto riguarda il partizionamento di un database, si può fare differenza tra *orizzontale* e *verticale* a seconda di come avviene la frammentazione del dato da distribuire: se ad essere distribuite sono intere righe di una relazione si parla di partizionamento orizzontale, se invece ad esserlo sono varie colonne si parla di partizionamento verticale, nel quale poi la relazione va in qualche modo ricostruita, ad esempio con l'uso di indici creati ad hoc.

Distributed Concurrency Control

Il controllo della concorrenza in un database distribuito concerne la sincronizzazione degli accessi ad esso in modo da garantirne proprietà come l'integrità: dal momento che rappresenta uno dei problemi più complessi da affrontare in questo contesto, è anche uno dei più studiati.

La maggiore differenza rispetto al caso di un database centralizzato è che il focus è spostato sulla consistenza del sistema nella sua globalità inteso come l'insieme di copie o frammenti sparsi per i nodi della rete: i due metodi generali per affrontare il problema si basano su strategie ottimistiche, che permettono di eseguire le richieste per verificarne poi la validità alla fine, oppure pessimistiche, che le bloccano prima che possano essere eseguite, in linea con la descrizione fornita in 5.1.

In genere entrambi questi meccanismi possono fare riferimento a primitive di locking, per garantire mutua esclusione nell'accesso ai dati o di timestamping, per avere un modo di ordinare le operazioni.

Pro e contro

Tra i vantaggi portati dall'utilizzo di un database distribuito uno dei più importanti riguarda sicuramente la *trasparenza* offerta grazie all'utilizzo del livello di astrazione che fornisce; essa può essere intesa in riferimento a svariati aspetti tra cui:

- rete, in quanto l'utente potrebbe non essere interessato alla localizzazione dei dati e alla loro effettiva provenienza dalla rete, dal momento che vi accede utilizzando le modalità;
- replica, visto che demandando la gestione della replica al database distribuito l'utente potrebbe non essere consapevole dell'esistenza di repliche, avendo di fatto la sensazione che i dati siano unici.

Un altro importante vantaggio proviene dal fatto che, essendo distribuito fa vari nodi della rete, anche la *reliability* dell'intero sistema ne risulta aumentata, andando ad eliminare potenziali *single point of failure*: nel caso in cui un nodo dovesse essere non raggiungibile per un qualsiasi motivo, il sistema continuerebbe a fornire il proprio servizio, al più abbassando il proprio livello di performance.

Inoltre, anche performance come il tempo di risposta possono essere migliorate grazie al concetto di località dei dati, che vengono posizionati più vicini ai punti in cui vengono maggiormente usati e al supporto verso una scalabilità orizzontale in cui altri nodi possono essere aggiunti al sistema per far fronte ad un numero maggiore di richieste; infine essi supportano anche più agevolmente l'espansione dei sistemi, permettendo l'aggiunta di componenti.

A fronte di queste peculiarità vanno considerati anche i possibili svantaggi che si possono riscontrare utilizzando tali database, che comprendono spesso un maggiore costo e complessità del software da utilizzare, in quanto deve fare fronte ad un ambiente distribuito, overhead in termini di computazione e rete utilizzata dovuto alle interazioni per la gestione della distribuzione dei dati e soprattutto grande complessità dei meccanismi per garantire integrità e consistenza all'interno del sistema.

5.2.2 Teorema CAP

Secondo il **Teorema CAP** (conosciuto anche come teorema di Brewer), un sistema informatico distribuito è in grado di soddisfare, allo stesso tempo, al massimo due delle seguenti garanzie, ma non tutte e tre:

- **Consistency** (consistenza): tutti i nodi vedono gli stessi dati nello stesso momento;

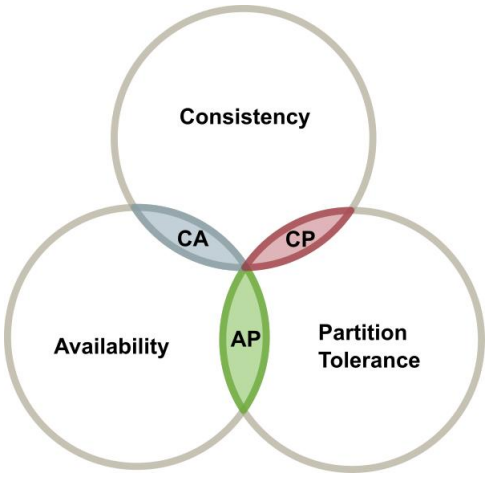

Figura 5.3: Rappresentazione visuale del teorema CAP.

- **Availability** (disponibilità): il servizio risulta disponibile e risponde alle richieste, indicando ciò che è riuscito e ciò che è fallito;
- **Partition tolerance** (partizionamento): il sistema continua a funzionare anche a fronte di partizionamenti dovuti ad errori di rete.

Si può quindi dedurre che tre sono le combinazioni possibili di queste caratteristiche, ossia:

- *CA*, in cui si rappresenta il caso in cui tutti i nodi del sistema sono sempre connessi, e quindi un eventuale partizionamento della rete causa un blocco del servizio;
- *CP*, in cui si considera che il sistema possa essere non disponibile in una qualche sua parte a fronte di partizionamenti, ma il resto dei dati è in uno stato consistente;
- *AP*, in cui il sistema è disponibile anche a seguito di un partizionamento, ma è possibile che i dati reperiti non siano in stato consistente.

Scegliere due delle proprietà descritte non significa però abbandonare del tutto la terza: infatti il significato del teorema sta nel fatto che, ottenendo due proprietà, la terza non può essere garantita completamente; per esempio, se si vuole un sistema partizionato che garantisca sempre consistenza bisogna sacrificare qualcosa in termini di disponibilità, probabilmente bloccando le richieste fino a quando il sistema non ritorna in uno stato consistente ed esse possono essere servite.

Questo teorema viene spesso citato per giustificare l'utilizzo, in campo distribuito, di modelli di consistenza più deboli rispetto alle proprietà **ACID** perseguite nelle transazioni nei tradizionali database centralizzati: uno dei principali in tal senso è rappresentato dall'acronimo **BASE**, *Basically Available Soft-state (services with) Eventual consistency.*

Rispetto alle proprietà ACID, con le quali si induce un modello in cui si vuole ottenere una *consistenza forte*, caratterizzato dall'utilizzo di strategie pessimistiche che bloccano le richieste finché non possono essere servite in uno stato consistente, il modello BASE promuove una forma di consistenza più debole, in cui a seguito di un aggiornamento in un nodo del sistema, seguenti richieste possono accedere al vecchio valore del dato per un certo periodo, chiamato *finestra di inconsistenza*. La metodologia BASE predilige quindi un'elevata disponibilità per i servizi, lasciando poi ad un meccanismo di pulizia di fondo il compito di risolvere i problemi causati da azioni ottimistiche che risultano aver violato la consistenza.

Eventual Consistency

Uno dei modelli di consistenza più deboli rispetto a quello usato nei database tradizionali è costituito dalla cosiddetta **Eventual Consistency**: esso in sostanza garantisce che, se non ci sono nuovi aggiornamenti riguardanti un certo dato, prima o poi (*eventually*) tutti gli accessi ad esso ne restituiranno il valore più aggiornato.

Questo particolare modello di *consistenza debole* viene largamente utilizzato nei sistemi distribuiti ed è promosso dall'utilizzo della metodologia BASE: ottenere questa forma di consistenza significa spesso raggiungere una *convergenza* delle repliche dei dati sparse per il sistema, coinvolgendo meccanismi di riconciliazione e recupero dai conflitti.

Fra le principali e più note varianti di questo modello possono essere incluse:

- *causal consistency*, in cui si garantisce che se un processo A ha comunicato a B l'aggiornamento di un datto, i seguenti accessi di B restituiscono il valore aggiornato;

- *read-your-writes consistency*, in cui se un processo A ha aggiornato un dato, i suoi seguenti accessi ad esso ne restituiscono il valore aggiornato;

- *monotonic read consistency*, in cui se un processo ha ricevuto un certo valore a seguito della lettura di un dato, susseguenti accessi non restituiscono un valore meno aggiornato;

- *monotonic write consistency*, in cui ad essere garantita è la sequenzialità degli accessi in scrittura di un singolo processo.

5.2.3 Single vs multi master replica

Distribuire un database significa spesso replicare, almeno in parte, i dati in esso contenuti in diversi nodi della rete, come descritto nelle precedenti sezioni: scegliere il metodo con cui questo processo viene eseguito nel sistema è un'attività complessa per la quale vanno presi in considerazione molti fattori, tra cui le funzionalità a cui gli utenti devono poter accedere, eventuali vincoli da rispettare e stime di previsione del carico di lavoro da sopportare.

Differenti modi di provvedere alla replica dei dati e alla sua gestione sono stati proposti nel corso degli anni, ma i due principali si differenziano relativamente al numero di repliche responsabili di gestire eventuali aggiornamenti ai dati:

- nei casi in cui i dati debbano poter essere aggiornati attraverso qualsiasi replica si parla di replica **multi-master**, a cui viene demandata quindi anche la gestione della sincronizzazione di potenziali aggiornamenti concorrenti e degli eventuali conflitti che possono sopraggiungere;

- nel caso in cui invece invece sia un singolo nodo della rete a essere responsabile della gestione degli aggiornamenti si parla di replica **single-master**, con la quale viene eletto un "master" attraverso il quale è possibile modificare i dati, i cui cambiamenti vengono poi propagati a tutte le altre repliche dette anche "slaves".

Permettere ad un solo nodo di gestire eventuali cambiamenti nei dati rende sicuramente più facile il mantenimento della consistenza all'interno del database, sacrificando però la flessibilità e la disponibilità introdotta dall'utilizzo del metodo multi-master, tramite cui si possono ottenere miglioramenti in termini di performance ed eliminazione di eventuali single point of failure.

Adottare un approccio multi-master verso la replicazione dati significa infatti aumentare vistosamente la disponibilità del sistema, introducendo però degli svantaggi da tenere in considerazione quali forme più deboli di consistenza o la complessità derivante dalla gestione dei conflitti, che potrebbe diventare intrattabile al crescere del numero dei nodi coinvolti.

Inoltre gli aggiornamenti alle repliche possono essere di tipo sincrono o asincrono, rispettivamente se essi devono avvenire all'interno di un'unica transazione atomica, che quindi coinvolge l'utilizzo di meccanismi di locking, oppure in maniera *lazy*, permettendo una sincronizzazione ritardata delle repliche che necessita così di meccanismi di riconciliazione.

In ambiente mobile, dove la connessione in rete non è garantita, è quindi più frequente utilizzare meccanismi asincroni di sincronizzazione delle repliche, e a seconda delle necessità di permettere o meno all'utente di modificare i dati in situazioni modalità offline bisogna mettere in campo una strategia multi-master, al costo di risolvere in seguito conflitti generati da tale utilizzo.

5.2.4 Tecnologie

Molte e differenti soluzioni sono state sviluppate nel campo dei database distribuiti, ognuna delle quali adotta diverse strategie per gestire la complessità delle problematiche da affrontare: in generale nessuna di esse offre tutte le funzionalità, fornendo specifici tipi e livelli di trasparenza, replica e partizionamento e privilegiando, a seconda dei casi, alcune proprietà CAP piuttosto che altre. L'adozione di una particolare tecnologia deve perciò avvenire a seguito di un dettagliato processo di analisi da cui devono essere emerse sia le funzionalità che il sistema deve esibire sia i suoi eventuali vincoli: nel contesto di questa tesi il caso particolare fa riferimento a uno scenario di collaborazione per il soccorso in emergenza, risultando così in una necessità di supporto per dispositivi mobile e operazioni in modalità offline, vincoli che di fatto riducono abbastanza il panorama tecnologico da esplorare.

Se infatti al giorno d'oggi sono presenti molte tecnologie in ambito dei distributed DBMS, come si può notare dalla figura 5.4[1], la maggior parte di essi è stata progettata per essere utilizzata in contesti di clustering di server, o comunque di situazioni in cui la connessione di rete è considerata garantita o quasi, consentendo quindi di considerare *failure* da questo punto di vista un evento non così frequente da gestire: i requisiti per l'esecuzione di questi strumenti sono anche abbastanza alti in termini di risorse computazionali, in contrasto con le spesso limitate caratteristiche dei dispositivi mobile.

Anche se molti dei maggior vendor di database relazionali hanno sviluppato dei supporti in ambito distribuito, il maggior fermento in questo ambito proviene dall'area dei cosiddetti database **NoSQL**, che non utilizzano il modello relazione per la memorizzazione dei dati: tra i maggiori esempi di database distribuiti NoSQL possono essere considerati *Apache Cassandra*, una soluzione open source progettata per gestire grandi moli di dati con elevata disponibilità in ambienti organizzati in cluster di server; *BigTable*, il database proprietario che Google utilizza per la gestione di dati strutturati su larga scala; *Amazon Dynamo*, che propone un modello key-value per la gestione ad elevata disponibilità di data store distribuiti e viene utilizzata

[1]Fonte: `https://blogs.the451group.com/information_management/2011/04/15/nosql-newsql-and-beyond/` consultata il 28/02/16.

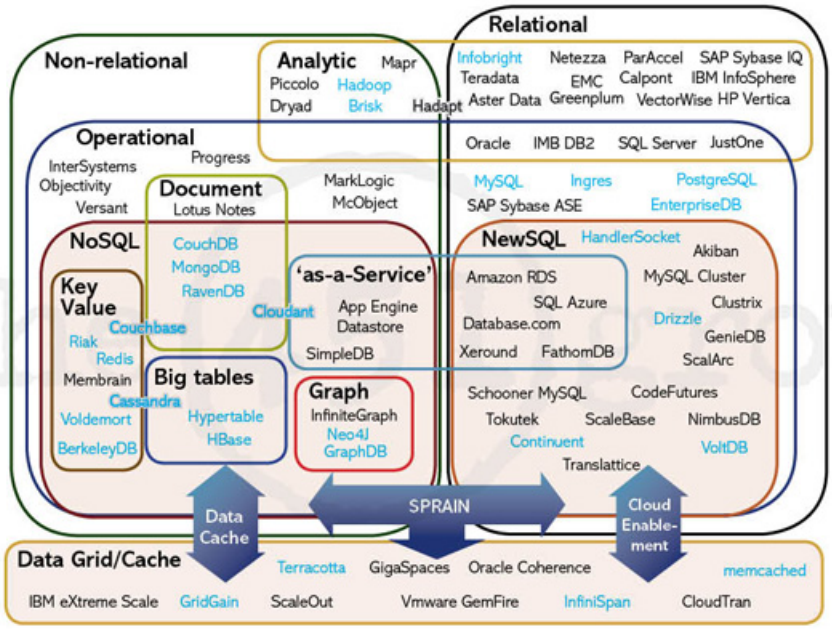

Figura 5.4: Overview del panorama dei database.

nella piattaforma cloud di Amazon; *Apache HBase*, tecnologia che fornisce in maniera fault-tolerant la possibilità di memorizzare grandi quantità di dati sparse e fa parte dell'ecosistema *Hadoop* di Apache; o anche *MongoDB*, uno fra i DB NoSQL più utilizzati che fornisce funzionalità di replica master-slave per aumentare le performance e la scalabilità nella gestione delle richieste.

Supporto per Mobile Computing

Esistono però anche soluzioni per database distribuiti che presentano un supporto diretto per il Mobile Computing, facilitando in questo modo lo sviluppo di applicazioni che permettano l'utilizzo di operazioni anche a fronte di disconnessioni dal sistema: essi infatti permettono di memorizzare in maniera persistente le informazioni direttamente sul device mobile in modo da poterle poi accedere anche localmente. In tale scenario assumono un ruolo fondamentale le funzionalità di replica e relativa gestione messe a disposizione dallo strumento; a questo proposito si possono citare tra le soluzioni proposte dai grandi vendor:

- **Oracle Database**, che grazie alla sua versione **BerkeleyDB** può essere eseguita su dispositivi mobile per lo storage persistente di dati, i quali possono poi essere sincronizzati con un'istanza di database collo-

cata su un server grazie all'utilizzo di un componente particolare chiamato **Mobile Server**, che agisce da *middle-tier* fra il client mobile e il server per abilitare il processo di sincronizzazione dei dati. A questo processo collabora inoltre un ulteriore componente, installato sul device mobile insieme e chiamato *Mobile Sync*, che è responsabile di collezionare i cambiamenti avvenuti localmente al dispositivo per poi comunicarli (ad esempio utilizzando HTTP) al Mobile Server tramite cui è perciò possibile effettuare la sincronizzazione con l'istanza server di Oracle Database. Oltre alla possibilità di aggiungere sicurezza alle comunicazioni tramite l'utilizzo di HTTPS, è possibile definire l'autenticazione degli utenti e anche il sottoinsieme di dati con i quali la sincronizzazione deve essere effettuata. [2]

- **Microsoft Azure Mobile Services** che, grazie all'utilizzo del particolare SDK da installare sul device mobile, permette una sincronizzazione dei dati salvati sul dispositivo tramite il servizio cloud Azure di Microsoft [3].

Il panorama tecnologico offre anche tool prettamente dedicati alla sincronizzazione di database, come **SymmetricDS** [4], un software open source che offre supporto per replica multi-master, sincronizzazione filtrata anche in ambienti eterogenei. Utilizzando i *database trigger* esso cattura i cambiamenti avvenuti in locale per poi poterli sincronizzare: scritto in Java, offre supporto per molti dei maggiori database in circolazione (tra cui MySQL, Oracle, SQL Server, PostreSQL), comprendendo anche SQLite tramite il quale è fornito anche un supporto per l'esecuzione di dispositivi Android, oltre a prevedere differenti opzioni di deployment (ad esempio standalone service oppure embedded in una applicazione).

Un'alternativa molto interessante agli strumenti citati che offre funzionalità di sincronizzazione e di replica in maniera agevole, senza particolari e pesanti fasi di configurazione o vincolare l'utilizzo di strumenti provenienti da un certo vendor, è costituita dal database NoSQL **Apache CouchDB**: grazie al suo supporto nativo per le applicazioni web, esso si presta bene anche all'interazione con dispositivi mobile, che possono utilizzare le API REST offerte per usufruire dei servizi messi a disposizione. Date le sue potenzialità, l'esplorazione condotta all'interno di questa tesi si è focalizzata soprattutto

[2] Fonte: documentazione Oracle, `http://docs.oracle.com/cd/E48200_01/doc.1130/e39324/nvovw.htm`, consultata il 25/02/16.

[3] Fonte: `https://azure.microsoft.com/en-us/documentation/articles/mobile-services-android-get-started-offline-data/`, consultato il 23/02/16.

[4] `http://www.symmetricds.org/`

Figura 5.5: Logo di CouchDB™, Copyright 2016 The Apache Software Foundation.

su di esso, in modo tale da comprenderne le funzionalità e capire se possa fare al caso di uno scenario particolare come quello del soccorso in emergenza: la prossima sezione si dedica quindi alla sua descrizione per darne una sintetica overview delle principali caratteristiche.

5.2.5 CouchDB

Apache **CouchDB**[5] *(Cluster of Unreliable Commodity Hardware database)* è un database document-oriented open source NoSQL che fa della facilità di utilizzo e della sua integrazione con il web i suoi punti di forza: implementato in Erlang e pensato per poter essere distribuito, in esso i dati vengono memorizzati in forma di documenti JSON e su di essi è possibile eseguire delle query grazie all'utilizzo di funzioni di MapReduce scritte in Javascript.

Uno dei suoi principali utilizzi può essere riscontrato nella piattaforma cloud *Cloudant*, acquisita nel 2014 da IBM che perciò è diventato uno dei maggiori contributori attivi al progetto open source di CouchDB.

A differenza di un database tradizionale, CouchDB non memorizza i dati e le relazioni attraverso il concetto di tabelle: un *database* in esso non è altro che una collezione di *documenti* indipendenti, ognuno dei quali mantiene il proprio schema che può essere diverso dagli altri, donando grande flessibilità al modello dei dati. Ogni documento è quindi identificato univocamente all'interno di un database: un'applicazione può accedere a differenti database,

[5] `https://couchdb.apache.org/`

servendosi delle API per le operazioni CRUD [6] esposte attraverso un servizio RESTful over HTTP.

Seguendo la classificazione indotta dal teorema CAP, CouchDB rientra nella categoria **AP**, esibendo un modello di consistenza di tipo *eventual*: utilizzando una strategia ottimistica per l'aggiornamento dei dati, prevede l'implementazione di un meccanismo del controllo di versione per evitare il locking dei file, lasciando all'applicazione il compito di risolvere i conflitti.

Infine una delle funzionalità principali che lo distingue dalle altre tecnologie disponibili è la replicazione di tipo *multi-master* accessibile tramite API HTTP, che ne aumenta le potenzialità in termini di scalabilità e disponibilità: essa è anche la funzione di principale interesse nel contesto di questa tesi, in quanto permette la sincronizzazione fra diverse istanze, anche parziali, del database distribuito in diversi nodi della rete, potenzialmente rappresentati anche da dispositivi mobile, a patto che possano utilizzare le API messe a disposizione.

Caratteristiche principali

Molteplici sono quindi le peculiarità di questo DBMS che può essere considerato distribuito: l'obiettivo di questa sezione è fornirne una sintetica overview focalizzata a far comprendere la sua utilità nel contesto di applicazioni mobile che necessitano di effettuare operazioni *disconnected*[7].

Modello di storage Il modello di storage è basato essenzialmente sul concetto di documento, che costituisce l'unità primaria dei dati in CouchDB: modellato in termini tecnici come documento in formato JSON, esso consiste di un qualsiasi numero di campi il cui valore è rappresentato da oggetti JSON o *attachments*, con cui possono essere modellate informazioni allegate come immagini e di alcuni metadati che vengono mantenuti dal DBMS, molto spesso contrassegnati dal carattere "_" all'inizio del loro nome.

Il modello di aggiornamento è, riguardo al singolo documento, lockless e ottimistico: prendendo come esempio il caso in cui due client vogliano modificare contemporaneamente lo stesso documento, quello che tenta di memorizzare il proprio cambiamento per ultimo riceve in risposta un errore di conflitto, tramite cui può caricare la versione aggiornata del documento e riapplicare la propria modifica; inoltre gli aggiornamenti non sono mai parzia-

[6] Acronimo che rappresenta le principali operazioni che possono essere svolte sui dati: Create, Read, Update, Delete.

[7] I contenuti di questa sezione sono rielaborati quasi integralmente a partire da [28] e dalla documentazione online di CouchDB [29]

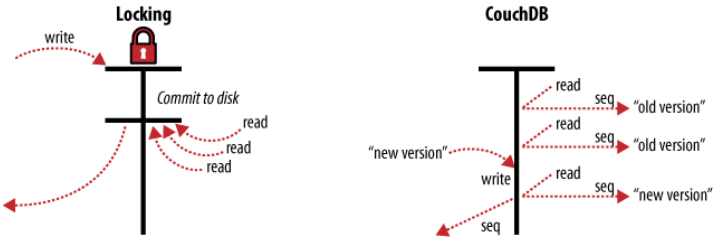

Figura 5.6: Modello di accesso ottimistico di CouchDB (fonte [29]).

li nel senso che hanno interamente successo oppure falliscono, non lasciando mai un documento parzialmente aggiornato.

Gli aggiornamenti al singolo documento sono serializzati e vengono salvati alla fine del file che lo rappresenta, utilizzando un meccanismo *append-only*, mentre gli accessi in lettura non vengono mai bloccati o interrotti da concorrenti aggiornamenti grazie all'utilizzo di un sistema *Multi-Version Concurrency Control* con il quale ogni client ha sempre accesso ad uno snapshot consistente.

In sintesi, CouchDB provvede all'utilizzo di un meccanismo che consente di lasciare un'istanza del database in uno stato consistente anche a fronte di crash del sistema operativo sottostante, perdendo però in tal caso gli aggiornamenti parzialmente effettuati.

Due fra i campi di maggiore importanza per un documento sono:

- `_id`, che rappresenta l'identificatore univoco per il documento all'interno del database, la cui scelta può essere a carico dello sviluppatore dell'applicazione, che deve assicurarne l'univocità, oppure demandare la gestione a CouchDB richiedendogli un UUID da utilizzare poi per la creazione del documento, che avviene tramite la API `PUT`;

- `_rev`, che rappresenta l'identificatore di revisione del particolare documento e serve per monitorarne i cambiamenti di versione e implementare un meccanismo di controllo tramite cui esso viene automaticamente aggiornato quando, ad esempio, i dati vengono aggiornati; da notare che questo campo viene utilizzao anche come *ETag* per le richieste e le risposte HTTP ai fini di caching.

Fra le API più particolari vi è quella tramite cui viene attivato il processo di sincronizzazione, che avviene tramite una richiesta `POST` al database di sistema `/_replicate`.

MapReduce e view I dati semi-strutturati sparsi fra i vari documenti di un database possono essere interrogati attraverso l'utilizzo di **view**: esse

permettono infatti di aggregare i dati contenuti in un database ed organizzarli secondo la struttura che si necessita a partire dai campi presenti nei documenti.

Tramite il concetto di vista, definita tramite una funzione Javascript salvata come stringa nel campo `views` di documenti particolari del database chiamati *design documents*, si possono ottenere rappresentazioni dinamiche del contenuto corrente del database: generare una vista su un numero molto grande di documenti è però un'attività che richiede l'utilizzo di molte risorse computazionali, così CouchDB utilizza un meccanismo per non doverle ricostruire ad ogni avvio, ma le materializza in maniera persistente e le mantiene aggiornate rispetto agli aggiornamenti del database grazie all'utilizzo di particolari indici e allo stesso meccanismo di append utilizzato per i normali documenti.

Una view viene in sintesi costruita attraverso l'esecuzione della funzione di `map` definita: questa funzione, che non deve provocare side-effects, accetta come input un documento ed *emette* coppie chiave-valore a partire dagli attributi in esso contenuti tramite il costrutto `emit`. Eseguendo questa funzione per ogni documento presente nel database si ottiene l'insieme di queste coppie *key-value* che rappresentano i documenti e vengono organizzate in una struttura dati ad albero (*B+ tree*) ordinata, grazie alla quale poi la view può essere costruita come una tabella ordinata in cui ogni riga è rappresentata da una di queste coppie. Insieme ad esse CouchDB emette automaticamente anche il campo `_id` che identifica univocamente il documento rappresentato dalla coppia considerata: sulle view possono essere quindi eseguite delle query che possono filtrare arbitrariamente i risultati a seconda dei valori assunti dalle determinate chiavi oppure ottenere i risultati ordinati nella maniera necessaria, analogamente a quanto succede utilizzando il linguaggio SQL nei database tradizionali. Una delle particolarità è che le chiavi non sono vincolate ad essere stringhe, ma possono assumere anche altri valori: uno di quelli più utilizzati è un array di stringhe, che permette di ottenere ordinamenti a grana molto fine che rispondono a particolari esigenze applicative.

Sulle coppie emesse dalla funzione `map` può operare una funzione di `reduce`, con la quale possono essere ottenuti risultati aggregati tramite la sua esecuzione su ogni nodo della struttura dati ad albero per computare il risultato finale: aggregando i valori a partire dalle foglie si possono ottenere risultati cumulativi che vengono aggiornati automaticamente insieme alle viste e su cui poi è possibile fare delle interrogazioni.

RESTful API over HTTP Un'altra particolarità di CouchDB è quella di offrire nativamente un'interfaccia HTTP per l'interazione con i dati: più

di preciso quello che ne risulta è un servizio web RESTful tramite cui l'utente può effettuare le normali operazioni di creazione, cancellazione, lettura o modifica (CRUD) sui dati utilizzando i metodi HTTP GET, POST, PUT e DELETE sui documenti visto che ognuno di essi è considerato come una risorsa identificata univocamente da un *URI*. Da questo punto di vista l'utilizzo di JSON come formato per i documenti memorizzati aiuta molto in quanto esso è normalmente adottato nel contesto di applicazioni web.

Inoltre anche altre operazioni possono essere effettuate tramite HTTP: ad esempio, visto che anche le view sono salvate come speciali documenti, anche le query possono effettuate tramite una richiesta GET al loro particolare URI; l'operazione più importante che viene però offerta da questo punto di vista è quella della sincronizzazione fra istanze di database, che viene approfondita nelle sezioni seguenti.

La possibilità di comunicare con CouchDB attraverso HTTP non deve essere sottovalutata: anche se di default risponde solamente a richieste provenienti dall'interfaccia di rete di loopback (localhost), esso può essere configurato per rispondere, ad esempio, alle richieste provenienti da una sottorete locale. In linea di principio esso può quindi agire da *web server*, fornendo supporto per la creazione di applicazioni web, ma può essere previsto un suo utilizzo anche in ambienti di deploy più complessi, dove può essere affiancato a diverse componenti in modo da essere utilizzato specificatamente per le particolari funzionalità che offre.

Sicurezza e gestione degli utenti Dal punto di vista della sicurezza, CouchDB offre solamente gli strumenti basilari per implementare *custom policy* dipendenti dal caso applicativo: esso infatti contempla solamente i ruoli di `admin` e `member`, con la differenza che i primi hanno praticamente libero accesso a tutti i documenti, compresi quelli di design, mentre gli altri possono accedere in lettura a tutti i documenti ma non possono modificare quelli di design. Il mapping fra utenti del sistema e questi due tipi di ruoli è fornito tramite il particolare design document `/{db}/_security`: tramite oggetti JSON può essere quindi stabilita anche un'associazione fra username e ruoli definiti appositamente per l'applicazione, in modo poi da implementare la propria policy grazie all'utilizzo di *validation functions* definite dallo sviluppatore che validano gli accessi ai documenti.

Una volta configurati correttamente gli utenti registrati per sistema tramite appositi documenti nello speciale database di sistema per l'autenticazione (`/_users` di default) CouchDB ne permette l'autenticazione in due differenti modi: uno basilare aggiungendo l'username e la password nell'URL della richiesta HTTP, che ovviamente non può essere considerato utilizzabile se non

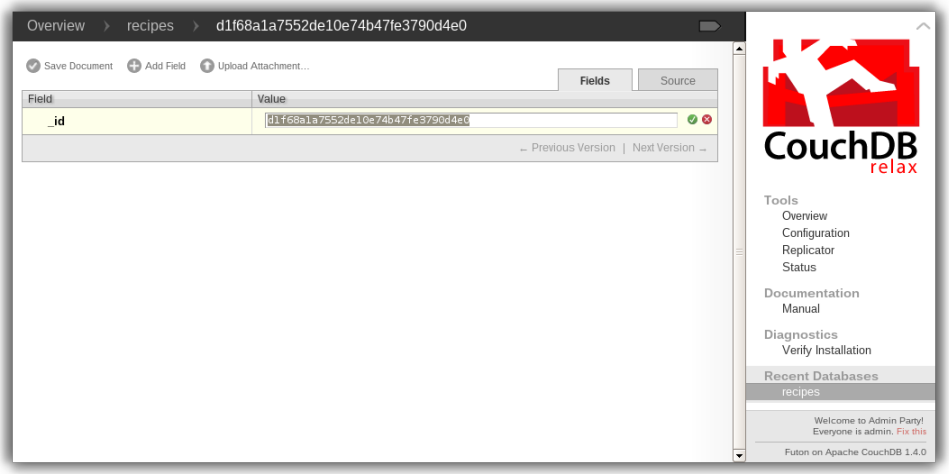

Figura 5.7: Futon: editing di un documento (fonte [29]).

in ambiente di testing in quanto non sicuro, oppure uno basato su *cookie*, nella quale viene attivata una sessione per l'utente a cui viene associato un token che deve essere poi utilizzato dall'utente per autenticarsi.

Infine, bisogna citare che, dalla versione 1.1.0 esso supporta nativamente SSL, ed è quindi possibile stabilire con esso connessioni sicure senza l'utilizzo di un proxy.

Web Console Una funzionalità comoda che CouchDB mette a disposizione è una GUI nativa web-based, chiamata **Futon** e visibile in figura 5.7, che permette di interagire con lo strumento, ad esempio per:

- configurare i suoi parametri di installazione;
- gestire i database e i documenti, creandoli o modificandoli e comprendendo anche quelli di *design* attraverso il quale vengono create le viste o gestiti gli utenti;
- visualizzare lo stato dei task in background sul server, tra cui quelli riguardanti la costruzione delle view e le repliche;
- una prima e basilare interfaccia per configurare una replica tra due istanze (anche remote) di database.

Replica e sincronizzazione

Una delle funzionalità principali di CouchDB è il suo meccanismo di gestione delle repliche distribuite, che permette l'applicazione anche in contesti in cui la rete non è garantita: esso infatti supporta il concetto di *shared nothing* tra le varie istanze di database, sulle quali è perciò possibile agire anche *offline*.

In estrema sintesi, CouchDB utilizza un meccanismo di replica incrementale multi-master basato su HTTP che utilizza come componente fondamentale il suo sistema di versioning (MVCC): ogni documento contiene un campo speciale (`_rev`)che ne indica l'ID di *revisione*, e ad ogni aggiornamento esso viene cambiato in modo da tenerne traccia, memorizzando così anche le revisioni precedenti.

All'atto della sincronizzazione di una replica rispetto ad un'altra è possibile così trasferire solamente i relativi cambiamenti dall'ultima sincronizzazione proprio grazie agli ID di revisione: è inoltre possibile filtrare le repliche tramite funzioni javascript salvate in documenti di design per decidere quali dati replicare ed ottenere repliche *parziali* che riguardano solamente i dati a cui all'utente interessa avere accesso anche offline.

Dal momento che viene utilizzata una strategia ottimistica nella gestione dei dati, CouchDB assume che possano esserci dei conflitti e pertanto offre un meccanismo per loro individuazione e gestione: se due utenti distribuiti cercano di modificare lo stesso documento viene scelto in maniera deterministica il vincitore, in modo tale che ogni istanza di replica scelga lo stesso e dando così modo di gestire il conflitto manualmente o automaticamente a seconda dei casi.

Dettagli per la replica In CouchDB ogni processo di replica coinvolge un database sorgente ed uno destinazione, che possono essere localizzati sulla stessa macchina o distribuiti, ed inoltre è unidirezionale, quindi per ottenerne uno veramente *master-master* bidirezionale bisogna attivarne due, in cui i ruoli di *source* e *target* vengono scambiati tra le relative istanze: il fine ultimo di questo processo è *sincronizzare* i cambiamenti avvenuti nel database sorgente in quello destinazione.

La replica è controllata attraverso l'uso di documenti nello speciale database di sistema `_replicator`, in cui ogni documento descrive uno di questi processi: è quindi attraverso la creazione di uno di questi documenti che il processo di replica viene fatto partire, ed è agendo su di essi che viene controllato, ad esempio fermandolo tramite la sua cancellazione.

Durante la replica, le due istanze di database vengono comparate per determinarne i relativi cambiamenti: attraverso l'utilizzo di un *change feed* della sorgente (nel documento `/{db}/_changes`), che ne memorizza tutti

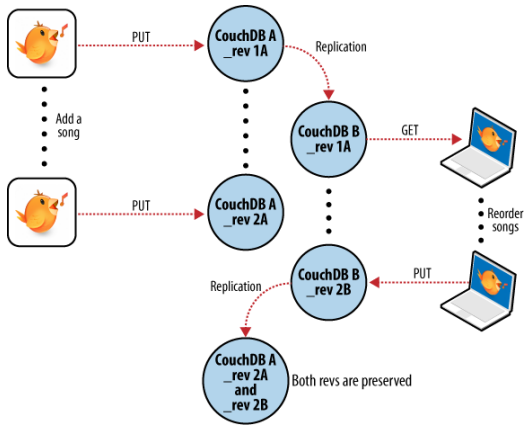

Figura 5.8: Esempio di conflitto (fonte [29]).

gli aggiornamenti, vengono rilevati i cambiamenti non ancora sincronizzati nella destinazione in modo tale da trasferire solo il necessario quantitativo di dati in rete; visto che anche la cancellazione di un documento è gestita come una sua nuova *revisione*, anche i cambiamenti in tal senso vengono propagati. Una volta raggiunto il termine di questo *feed* di cambiamenti il processo di sincronizzazione è terminato: in un processo continuo (la cui proprietà `continuous` è quindi settata a `true`), esso si mette in attesa di nuovi cambiamenti fino a che il task non viene manualmente interrotto.

I metodi di filtraggio permessi per decidere quali dati replicare e sincronizzare sono essenzialmente due:

- documenti *locali*, identificati tramite l'URI `/{db}/_local/{docid}`, non vengono replicati in quanto saltati dal processo di sincronizzazione, e quindi possono essere utilizzati per memorizzare documenti che non si vuole siano replicati;

- l'utilizzo di funzioni di filtro Javascript che servono a valutare ciascun documento permettendone la sincronizzazione solamente se restituiscono in output il valore `true`.

Attraverso il proprio meccanismo di replica e sincronizzazione CouchDB esibisce quindi un modello di *eventual consistency*, in cui a livello distribuito viene sacrificata l'immediata consistenza delle informazioni per permettere una maggiore disponibilità del servizio, reattivo così anche nel caso le repliche non potessero essere sincronizzate: quando poi esse lo saranno, le azioni ottimistiche intraprese dall'utente sono valutate al fine di individuare potenziali *conflitti*.

Essi possono avvenire se, a partire da una stessa revisione di un documento, due utenti hanno effettuato due modifiche differenti: in questo caso entrambe le versioni vengono memorizzate e il documento viene considerato in stato di conflitto, ma internamente a CouchDB ne viene scelta una *vincitrice* con la quale vengono poi aggiornate anche eventuali view.

A livello applicativo è quindi possibile individuare quali documenti siano in conflitto in modo tale da agire per risolverli, ad esempio provvedendo al *merge* (application dependent e quindi a carico dello sviluppatore) per poi cancellare le precedenti revisioni conflittuali in modo tale da eliminare il conflitto. É ovvio che il merge di versioni conflittuali dello stesso documento è un problema che può diventare complesso a seconda dell'applicazione trattata, ma è un costo da pagare nel caso si debba permettere un utilizzo offline dei dati.

5.2.6 Couchbase Lite e supporto mobile

Dal punto di vista dei dispositivi mobile è necessario quindi introdurre introdurre uno strumento che permetta di interagire con il database CouchDB installato su di un server per sfruttarne le possibilità offerte in termini di sincronizzazione di dati: diversi sono i tentativi reperibili tramite un'esplorazione in rete di implementazione di moduli client, come *PouchDB* [8], database in-browser open source in Javascript, ma quello che offre maggior supporto è **Couchbase Lite**, sviluppato dall'azienda *Couchbase Inc.* Essa offre i suoi prodotti in due differenti versioni, una di tipo *community* open-source (licenza Apache2) e un'altra di tipo *enterprise* che presenta maggiori restrizioni e un costo di sottoscrizione: tra di essi compare anche un database NoSQL chiamato *Couchbase Server* che eredita alcune caratteristiche da CouchDB [9], da cui è in qualche modo ispirato (una delle due aziende la cui unione ha prodotto Couchbase è CouchOne, la quale contribuiva attivamente al progetto open source CouchDB).

Couchbase Lite rappresenta un database NoSQL embedded nel dispositivo mobile tramite cui è possibile memorizzare in maniera persistente i dati, ma la sua funzione principale è il supporto alla sincronizzazione con un'istanza server: esso infatti implementa lo stesso protocollo di replica di CouchDB tramite HTTP. Anche se pensato nello specifico per essere sincronizzato con un'istanza server di Couchbase tramite l'utilizzo di un componente di middletier chiamato *Sync Gateway*, che di fatto ha il ruolo di abilitare per esso il protocollo di sincronizzazione tramite API HTTP, il suo utilizzo risulta essere

[8] `http://pouchdb.com/`

[9] Comparazione Couchbase Server - CouchDB `http://www.couchbase.com/couchbase-vs-couchdb`, consultato il 29/02/2016.

compatibile con CouchDB: anche il modello dei dati è di fatto lo stesso, in quanto vengono salvati tramite documenti JSON.

Questo strumento [30] offre così un supporto nativo nello specifico non solo per le maggiori piattaforme come iOS, Android e .NET di Windows, ma anche a strumenti come *Xamarin* per lo sviluppo di applicazioni mobile cross platform, fornendo un livello di astrazione che permette allo sviluppatore di interagire in maniera agevole con i dati nel linguaggio nativo di sviluppo per il dispositivo.

Tra le sue principali feature, molte delle quali in comune con CouchDB, vi sono:

- modello dei dati orientato ai documenti che ne permette un utilizzo flessibile dal punto di vista della struttura, non vincolato da forti relazioni; i dati vengono così serializzati in una rappresentazione interna JSON, identificati da un ID univoco e memorizzati in maniera persistente;

- possibilità di interrogare i dati tramite query eseguite su delle apposite viste definite in linguaggio nativo che utilizzano il concetto di MapReduce analogamente a CouchDB, le quali vengono materializzate persistentemente e aggiornate quando i dati subiscono dei cambiamenti;

- supporto built-in alla *change notification* in quanto sono presenti dei metodi per permettere all'applicazione di restare in ascolto dei cambiamenti che avvengono nei dati locali, seguendo l'idea del pattern Observer; in questo modo lo sviluppatore viene sollevato dal dover implementare questa logica, e gli viene richiesto solamente di implementare la funzione che li gestisca;

- funzionalità di *sincronizzazione* built-in per il supporto alla replica master-master seguendo il protocollo di CouchDB; analogamente ad esso è infatti compreso l'utilizzo di un sistema di versioning che permette di individuare e risolvere eventuali conflitti avvenuti durante l'utilizzo offline.

API Overview Molte sono le API [30] e le funzionalità offerte da questo framework, distribuito sotto forma di libreria da utilizzare nello sviluppo dell'applicazione; tra di esse le principali riguardano:

- un oggetto `Manager`, entità utilizzata per la gestione dell'insieme di database salvati sul dispositivo; la sua creazione risulta perciò necessaria per l'interazione con i dati e inoltre è tramite la configurazione

di sue particolari opzioni che l'utilizzo dei dati dati può essere configurato, ad esempio restringendo le possibilità di accesso o cambiando le funzionalità di logging;

- `Database`, l'astrazione con cui viene modellata una collezione di documenti e accessibile tramite l'utilizzo di un `Manager`; in ogni applicazione possono essere creati ed utilizzati diversi database, ognuno dei quali comunque contiene i propri documenti a cui sono correlati viste, replicazioni ed eventuali funzioni per il loro filtraggio e sui quali tramite esso si può agire;

- attraverso il concetto di `Document` viene rappresentata l'unità fondamentale in cui i dati sono suddivisi all'interno di Couchbase Lite, caratterizzato da campi il cui valore è un oggetto JSON e identificato da un *documentID*; tra le sue proprietà principali spicca *revisionID* (campo `_rev`), tramite cui il sistema tiene automaticamente traccia della versione corrente del documento in modo da memorizzarne lo storico dei cambiamenti; è attraverso le API fornite per questa entità che è possibile il supporto alle operazioni CRUD sui dati e la gestione dei conflitti, visto che è possibile ottenere una lista delle eventuali revisioni in conflitto per ogni documento in modo da tentarne una risoluzione;

- un oggetto `View` con il quale è possibile definire gli indici persistenti con cui poi interrogare i dati presenti nei documenti del database tramite delle query; analogamente a CouchDB anche in questo caso si fa utilizzo di una funzione di `map` da documenti a coppie key-value e di una di `reduce` definite in linguaggio nativo, in modo che possano essere incrementalmente aggiornate con le modifiche avvenute nel database;

- tramite un oggetto `Query` è così possibile definire delle interrogazioni sul database, in particolare sulle viste definite in precedenza, in modo da ottenere risultati strutturati o aggregati a partire dai documenti presenti; è inoltre prevista la possibilità di sfruttare una particolare query (chiamata **all docs query**) per ricavare tutti i documenti di un certo tipo (ad esempio entro un certo range di valore di un campo oppure in stato conflittuale); in più anche il concetto di `LiveQuery`, con cui si definisce una query che rimane attiva per monitorare cambiamenti nei dati e notificarli all'utente, risulta essere di grande utilità, soprattutto nel contesto di aggiornamenti dei componenti della UI.

Va dato infine rilievo all'oggetto `Replication`, tramite cui viene rappresentato il concetto di processo di sincronizzazione tra due repliche: esso può

essere configurato decidendone le varie caratteristiche, viene eseguito in maniera asincrona in un thread in background ed è possibile monitorarne lo stato. Il processo di sincronizzazione è caratterizzato da diverse caratteristiche, tra cui:

- *push vs pull*, rispettivamente i casi in cui i cambiamenti avvenuti nel database locale considerato vengono sincronizzati in una diversa istanza e viceversa;
- *one-shot vs continuous*, in quanto è possibile definire un processo di sincronizzazione che si fermi una volta ottenuti tutti i relativi cambiamenti o che resti continuamente in attesa di cambiamenti futuri per trasferirli non appena essi vengono individuati;
- *filtering*, in quanto è possibile definire funzioni di filtro per decidere il sottoinsieme dei documenti del database da sincronizzare, tipicamente a seconda del valore assunto da determinati campi; esse possono essere definite sia per la sorgente locale in caso di push attraverso funzioni scritte in linguaggio nativo, sia per la sorgente remota in caso di pull, nel caso di CouchDB rappresentate da funzioni Javascript salvate in documenti lato server.

Esiste inoltre la possibilità di autenticare la sincronizzazione in differenti maniere, tra le quali è compreso anche il supporto per l'implementazione di metodologie custom per garantire la sicurezza degli accessi.

In conclusione si può considerare che l'utilizzo in coppia di CouchDB e CouchBase Lite permetta di ottenere un ambiente per uno sviluppo agevole di applicazioni in cui l'utente modifica localmente i dati sul proprio device e poi li sincronizza con l'istanza localizzata su un server remoto: in tal modo il sistema nel suo complesso garantisce una maggiore disponibilità, essendo accessibile anche in situazioni di disconnessioni.

Il loro utilizzo viene inoltre approfondito nel capitolo seguente, in cui alle valutazioni emerse da questa esplorazione si aggiunge anche una fase di sperimentazione di queste tecnologie per saggiarne le potenzialità in maniera più concreta.

5.3 Message Oriented Middleware

In un sistema distribuito molte delle difficoltà da affrontare provengono dall'utilizzo della rete per le comunicazioni: in casi particolari come quello considerato nel contesto di questa tesi, in cui in sintesi si necessita di una forma di comunicazione coordinata persistente per permettere agli operatori di essere

in possesso dei dati anche qualora non avessero possibilità di connettersi in rete, esse si acuiscono ancora di più, rendendo necessario **disaccoppiare** fortemente le entità presenti nel sistema, che devono perciò interagire in maniera asincrona. Ciò risulta ancora più importante se, come nel caso considerato, pur essendo in una situazione in cui è presente un elemento di centralizzazione nel sistema, il tipo di comunicazione necessaria è concettualmente bidirezionale: è infatti tramite l'utilizzo di questo punto di centralizzazione che gli operatori vorrebbero condividere le informazioni riguardanti la missione e i pazienti trattati.

Un paradigma *a scambio di messaggi* è perciò necessario per abilitare questo tipo di disaccoppiamento: una delle principali famiglie di strumenti utilizzati nel panorama tecnologico a questo proposito è quella che fa riferimento ai **Message Oriented Middleware (MOM)**, termine con il quale si definisce una qualsiasi infrastruttura (o middleware) che fornisca funzionalità in termini di scambio di messaggi [31], molto diffusi nella costruzione di sistemi distribuiti a livello *enterprise*. L'utilizzo di un MOM abilita infatti un modello in cui si possono utilizzare relazioni peer-to-peer tra i suoi client, che hanno così la possibilità di comunicare tra di loro attraverso l'utilizzo di uno (o più) server che agiscono da intermediari per l'invio e la ricezione di messaggi: essi abilitano così la costruzione di sistemi flessibili e coesi nel senso che un eventuale cambiamento di una parte del sistema non deve comportare la necessità di cambiamenti nelle altre, supportando in tal modo anche l'interoperabilità.

Fornendo un livello di astrazione intermedio sul quale le applicazioni possono affidarsi per scambiare esplicitamente messaggi, essi permettono inoltre di non preoccuparsi dei dettagli dei livelli sottostanti, nascondendone in parte le complessità di gestione: attraverso le differenti topologie supportate è infatti possibile ottenere diversi modelli di comunicazione che vanno oltre al tradizionale *client-server* utilizzato nei sistemi web.

5.3.1 Principali vantaggi

I maggiori benefici provenienti dall'utilizzo di un MOM sono quelli che derivano dal **disaccoppiamento** che induce fra le entità che fanno parte del sistema: promuovendo uno stile di comunicazione asincrono basato sui messaggi, esso supporta anche l'interazione fra entità che non necessariamente sono in esecuzione nello stesso momento, offrendo un approccio *persistente* tramite cui i messaggi sono temporaneamente salvate in strutture quali *Message Queue*. É quindi grazie alla mediazione del MOM che l'interazione fra due entità nel sistema può essere considerata asincrona anche se sussisto-

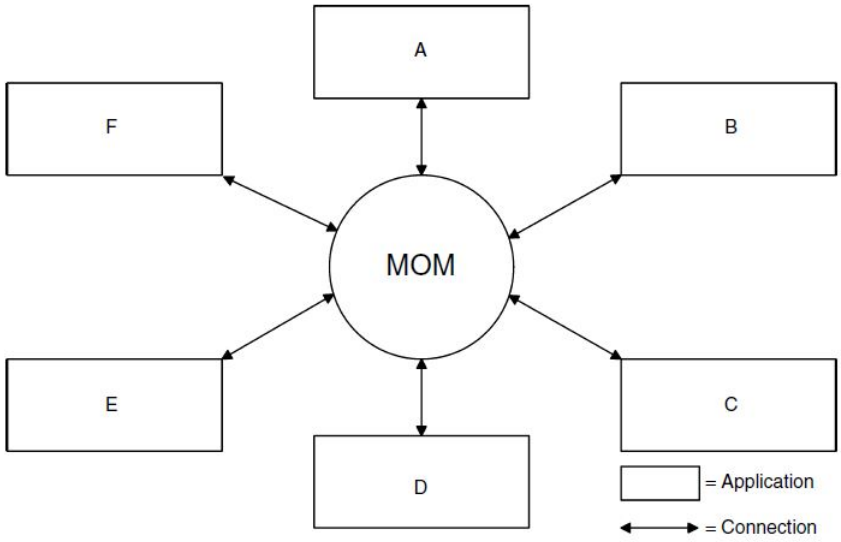

Figura 5.9: Disaccoppiamento tramite MOM (fonte [31]).

no ovviamente comunque relazioni di sincronia nelle interazioni fra entità e middleware stesso.

Il forte disaccoppiamento introdotto induce anche altri benefici, tra cui:

- **affidabilità**, grazie al livello di persistenza introdotto nelle comunicazioni;
- **scalabilità**, dato che il disaccoppiamento può essere utilizzato anche fra i diversi sottosistemi in modo da scalarli in maniera differente a seconda delle esigenze per far fronte, ad esempio, ad improvvisi picchi di attività in uno particolare di essi, prevedendo così meccanismi di load-balancing;
- **disponibilità**, visto che un fallimento in una parte del sistema non implica il fatto che tutto il sistema non sia più disponibile grazie al grado di disaccoppiamento introdotto fra le entità.

A fronte di questi vantaggi è però necessario gestire la maggiore complessità indotta dall'utilizzo di uno stile asincrono e assicurare che la coordinazione basata sui messaggi avvenga in modo consono, responsabilità demandata allo sviluppatore.

A partire da queste considerazioni l'utilizzo di un MOM può essere una soluzione efficace in sistemi i cui componenti sono geograficamente distribuiti e devono soddisfare determinati requisiti in termini di affidabilità o flessibilità pur non disponendo di un'elevata qualità per le comunicazioni in rete: in

questi scenari la mediazione offerta dal MOM, soprattutto in termini di persistenza delle comunicazioni a fronte di errori di rete, può risultare di grande aiuto.

5.3.2 Elementi essenziali

Tra i concetti principali necessari per comprendere l'utilità dei MOM vanno sicuramente citati quello di **Message Queue** e i modelli di comunicazione supportati.

Il concetto di Message Queue risulta di fondamentale importanza per i MOM in quanto è principalmente grazie ad esso che viene realizzata la persistenza in grado di disaccoppiare le entità che devono comunicare: essa fornisce infatti la possibilità di memorizzare i messaggi nella piattaforma, tramite la quale possono essere poi inviati o ricevuti. I messaggi da inviare o ricevere vengono perciò inseriti nelle relative code, che di default sono di tipo FIFO, ma che risultano essere configurabili sotto diversi punti di vista come dimensione o metodi di ordinamento dei messaggi; è inoltre possibile che lo stesso componente utilizzi diverse *queue*, o che diverse applicazioni condividano la medesima. Esistono diversi tipi di code di messaggi tipicamente utilizzate, tra cui pubbliche o private a seconda del tipo di accesso permesso, temporanee, ossia che esistono solo per un certo lasso di tempo o altre destinate a raccogliere i messaggi che non è stato possibile consegnare (*dead-message*).

Inoltre l'utilizzo di particolari code abilita anche meccanismi di *routing* dei messaggi sia a livello di piattaforma che di applicazione, nel quale è compito dello sviluppatore poi fornire le informazioni application-dependent per implementarli, rendendo possibile l'implementazione di paradigmi broadcast o multicast.

Il ruolo del MOM può essere quindi considerato importante anche per la realizzazione del pattern architetturale *Message Broker*; in più oltre al routing vi sono anche molti altri servizi comunemente offerti dalle implementazioni di MOM. Tra questi possono essere considerati:

- *message filtering*, che permette alle entità di essere selettive nei riguardi dei messaggi da ricevere da un determinato canale, e può essere realizzato a diversi livelli (e.g. a seconda del canale, del soggetto o del contenuto);

- supporto per le *transazioni*, che permettono di raggruppare più task in una singola unità logica di lavoro, che ha fallisce o ha completamente successo in ognuna delle sue parti (proprietà ACID);

- *configurabilità QoS* tramite cui possono essere specificati diversi parametri di qualità del servizio offerto tra cui opzioni di invio (*"at-most once, at-least-once, or once-and-once"*) o proprietà di *guaranteed message delivery*,

- *trasformazione* dei messaggi, che non è detto debbano arrivare nello stesso formato con cui sono stati inizialmente inviati a destinazione; ad esempio un caso in cui questo servizio risulta necessario è per disaccoppiare due sistemi di diversa natura cambiando il formato in quello nativo del ricevente.

Due sono sostanzialmente i modelli di comunicazioni supportati tramite l'utilizzo di MOM: **Point-to-Point** e **Publish-Subscribe**; nel primo tipicamente un messaggio viene inviato ad un singolo ricevente, memorizzandolo fino a che esso non è grado di riceverlo, è questo il caso anche di scenari *producer-consumer* dove multiple entità inseriscono messaggi in una coda da cui poi sono recuperati da entità in competizione tra loro. Nel secondo modello invece una copia del messaggio viene recapitata a ciascuna entità che ne aveva in precedenza manifestato interesse, indifferentemente secondo modalità uno-a-molti o molti-a-molti, in cui non vi sono ruoli predefiniti di produttori o consumatori, ma solamente quelli di *publisher* e *subscriber*, rispettivamente le entità che pubblicano e che sono interessate a determinate informazioni. In quest'ultimo modello, con cui può essere ben rappresentata la disseminazione di informazioni tra entità che non hanno necessità di conoscersi a priori, la selezione dei messaggi da ricevere risulta fondamentale, data la potenziale grande mole di informazioni ricevute, ed è basata principalmente sul concetto di *topic* o canale di comunicazione, che può essere organizzato anche in maniera gerarchica permettendo così grande flessibilità e fine granularità nella sottoscrizione al giusto livello delle informazioni.

Una piattaforma MOM offre inoltre due diversi metodi di ricevere i messaggi ai propri client: uno di tipo *push*, in cui è la piattaforma a notificarlo non appena un nuovo messaggio è disponibile, e un altro di tipo *pull*, in cui è il client a dover proattivamente controllare la disponibilità di nuovi messaggi, ad esempio tramite polling sulla *queue*.

5.3.3 Standard e protocolli

A causa della grande diffusione di questo tipo di strumenti e tecnologie, soprattutto in ambito enterprise dove sono molto utilizzate per realizzare integrazioni di servizi eterogenei, molti sono gli standard e i protocolli esistenti, rappresentati specifiche poi implementate nelle varie piattaforme.

AMQP

Advanced Message Queuing Protocol è uno standard open di OASIS di livello applicazione pensato per garantire interoperabilità alle applicazioni che lo implementano: il protocollo definisce in maniera precisa il comportamento dei client e del provider e il formato dei messaggi attraverso una specifica binaria, in modo tale che anche implementazione provenienti da diversi vendor possano interagire. A differenza di altri standard di messaggistica, AMQP separa il modello di trasporto dei messaggi da quello con cui vengono memorizzati, e la sua specifica è definita su diversi livelli: un *type system*, un protocollo simmetrico e asincrono per il trasferimento dei messaggi, uno standard per il formato dei messaggi e un insieme di funzionalità di messaging, tra cui una particolare metodologia di routing che si basa su componenti chiamati *Exchange*.

Esso infatti supporta la maggior parte degli scenari di messaggistica estendendo i due principali metodi di messaggistica presentati per i MOM: point-to-point con il metodo *Direct Exchange*, *Fanout*, *Topic Exchange* con cui il routing è effettuato sulla base del suo identificatore e *Header Exchange*, in cui l'instradamento è effettuato sulla base dei valori del campo header.

Esso prevede inoltre diverse possibilità riguardanti il tipo di message queue utilizzabile, tra cui una forma round-robin e una store-and-forward per un invio ritardato rispetto alla memorizzazione. Infine AMQP fornisce diverse soluzioni anche per quanto riguarda la sicurezza e l'affidabilità, permettendo un controllo abbastanza fine sulla qualità del servizio, decidendo ad esempio se memorizzare i messaggi, garantire una singola ricezione da parte del destinatario o prevedere dei retry a fronte di fallimenti.

AMQP può quindi essere considerato come uno degli standard più completi che supporta un bel numero di funzionalità utili in differenti scenari di integrazione e interoperabilità.

MQTT

Message Queue Telemetry Transport è uno standard ISO basato sul paradigma publish-subscribe per un protocollo leggero di messaggistica, pensato principalmente per scenari di connessione remota in cui la disponibilità di banda è limitata così come le risorse dei dispositivi coinvolti, per i quali è richiesto un basso impatto dell'infrastruttura utilizzata per lo scambio di messaggi, come ad esempio in sistemi in cui sono presenti dispositivi embedded o mobile. Basato su TCP/IP e sviluppato inizialmente da IBM, esso risulta essere ottimizzato per scenari in cui i dispositivi risultano simultaneamente connessi, non presentando un concetto di *message queue* e quindi

supporta un sottoinsieme limitato di funzionalità rispetto ad altri standard più completi.

Il meccanismo di publish-subscribe supportato si basa sulla presenza di un *broker*, responsabile di instradare i messaggi ai client interessati secondo il concetto di topic; inoltre ai client che si connettono per la prima volta è possibile recuperare l'ultimo messaggio inviato, secondo il paradigma *last-value-queue*, ma non sono molte le funzionalità offerte a livello di affidabilità, consentendo solamente l'utilizzo di meccanismi basilari di invio e ricezione.

Sebbene abbastanza limitato come numero di funzionalità, MQTT ha guadagnato nell'ultimo periodo molta popolarità soprattutto a causa della sua leggerezza, che lo rende molto adatto a scenari di mobile e pervasive computing dove è frequente l'utilizzo di dispositivi embedded: per semplici scenari in cui si necessiti solamente di meccanismi di push, come ad esempio l'invio di un aggiornamento da parte di un sensore, rappresenta una soluzione da tenere in considerazione.

XMPP

Extensible Messaging and Presence Protocol è un protocollo di comunicazione che abilita lo scambio *near-real-time* di informazioni di dati strutturati in forma di XML fra due o più entità presenti in rete. Inizialmente pensato come piattaforma di messaggistica istantanea in forma testuale basata su TCP/IP, esso si basa sul supporto a comunicazioni asincrone point-to-point di flussi di dati XML grazie ad un'architettura decentralizzata in cui i client e i server sono univocamente identificati (attraverso JID).

Caratteristica peculiare di questo protocollo è il supporto *presence-aware* tramite una funzionalità built-in che permette ad un'entità di informare le altre presenti nel sistema riguardo il proprio stato di connettività, in modo da aumentare la facilità delle comunicazioni real-time rendendo le sorgenti più consapevoli dello stato dei riceventi prima di iniziare la comunicazione. Tra le feature offerte dalle implementazioni di XMPP se ne possono trovare alcune riguardanti aspetti di sicurezza come autenticazione e cifratura dei canali di comunicazione, ma le funzionalità principali ruotano attorno allo streaming di dati XML di cui si possono individuare tre differenti tipologie di primitive di comunicazione, basate sul concetto di *stanza* che rappresenta in sintesi l'unità di comunicazione: *Message* per gestire il push di generici messaggi, *Presence* per la gestione dei messaggi riguardanti lo stato di connettività e *Info/Query* per comunicazioni più strutturate come ad esempio quelle request/response.

STOMP

Simple (or Streaming) Text Oriented Message Protocol è un semplice protocollo text-based che punta a fornire una semplice specifica interoperabile in modo che un client STOMP possa interagire con un qualsiasi broker che implementi il protocollo: è facile da implementare e agnostico rispetto al linguaggio utilizzato per l'implementazione, così da consentirne un'ampia diffusione in differenti piattaforme.

Pensato principalmente per abilitare l'interazione di componenti enterprise con linguaggi di scripting comuni nello sviluppo web, esso fornisce un sottoinsieme abbastanza limitato delle comuni operazioni di messaging, e non si basa sui concetti di queue o topic: il meccanismo di publish-subscribe viene infatti realizzato tramite una stringa di destinazione del messaggio, interpretata dal broker come una sorta di canale. In effetti, un server per STOMP è modellato come un insieme di destinazioni verso le quali un messaggio può essere inviato, trattate come stringhe opache la cui sintassi è specifica per il particolare server, mentre un client può agire (anche simultaneamente) secondo due ruoli: *producer*, inviando messaggi verso una destinazione oppure *consumer*, ricevendo messaggi e sottoscrivendosi a determinate destinazioni.

Anche se si distingue per supportare solo un limitato set di operazioni sui messaggi, la sua facilità di implementazione e flessibilità, unita all'integrazione in ambito Web, lo rendono un'alternativa appetibile.

5.3.4 Tecnologie

Dal punto di vista tecnologico numerosi sono oggigiorno gli strumenti disponibili nell'area dei MOM, ognuno dei quali implementa una o più specifiche di quelle citate nella precedente sezione: la scelta del protocollo da utilizzare dipende soprattutto dallo scenario da affrontare e dalle sue caratteristiche, in quanto ognuno di essi presenta delle particolarità che lo rende idoneo per determinate situazioni e meno per altre. Ad esempio in scenari in cui a contare molto è la leggerezza dell'infrastruttura e delle comunicazioni MQTT è sicuramente una valida alternativa da considerare, che però risulta essere molto meno idonea in contesti in cui si necessiti di un inseme ricco di funzionalità, per i quali è quindi meglio puntare ad esempio su uno strumento che offra l'implementazione di AMQP, uno dei protocolli più completi da questo punto di vista; inoltre non bisogna sottovalutare neanche l'impatto della grande diffusione odierna delle tecnologie Web a cui STOMP offre supporto, poiché la sua interoperabilità in questo senso può essere considerata come una feature molto importante in più di un contesto.

Tra le implementazioni più famose ed utilizzate del concetto di *message broker*, che forniscono una piattaforma per il supporto agevole allo scambio di messaggi possono essere citate:

- **ActiveMQ**, un message broker open source realizzato da Apache in Java che implementa diversi protocolli fra quelli citati come AMQP, STOMP e MQTT per fornire un elevato livello di interoperabilità, oltre ad essere pienamente compatibile con la specifica JMS [10]; oltre alla flessibilità di essere compatibile con molti protocolli, altre delle maggiori riguardano sicurezza e persistenza, che possono essere configurate, supporto ad avanzate funzionalità di routing ed instradamento dei messaggi e alle transazioni ed infine un vasto insieme di API client disponibili per diversi linguaggi;

- **Apache Kafka**, un'implementazione open source di message broker distribuito general purpose che da supporto a forme avanzate di comunicazione a scambio di messaggi tramite il pattern publish/subscribe; la sua maggiore particolarità è che non si appoggia a nessuno standard di quelli descritti e si basa su un servizio di memorizzazione di log distribuito che gli permette di essere veloce, scalabile e affidabile, in quanto riesce a gestire grandi moli di dati assicurando la persistenza dei messaggi su disco;

- **RabbitMQ**, che è una delle implementazioni open source di riferimento per quanto riguarda il protocollo AMQP, e la cui descrizione viene approfondita brevemente nel paragrafo seguente.

Per molte di queste soluzioni esistono rispettive controparti che abilitano questo tipo di comunicazione su dispositivi mobile, tra cui, per citarne una delle principali, si può ricordare **Paho**, un'implementazione open source di client per il protocollo MQTT e quindi pensata nello specifico per applicazioni M2M e IoT disponibile in svariati linguaggi e per diverse piattaforme, tra cui Android.

Il panorama delle implementazioni di piattaforme per lo scambio di messaggi è molto ampio ed esplorarlo completamente esula dagli scopi di questa tesi, la cui ricognizione è essenzialmente focalizzata ad estrarre le caratteristiche essenziali che questo tipo di strumenti possono fornire, in particolare nel contesto del caso di studio descritto, nel quale l'utilizzo di dispositivi mobile assume rilevanza. Come esempio di riferimento è stato scelto RabbitMQ in

[10] JMS è la specifica di API Java per permettere a componenti basati su Java EE di comunicare attraverso lo scambio di messaggi asincrono, in modo da permettere il loro disaccoppiamento in sistemi distribuiti.

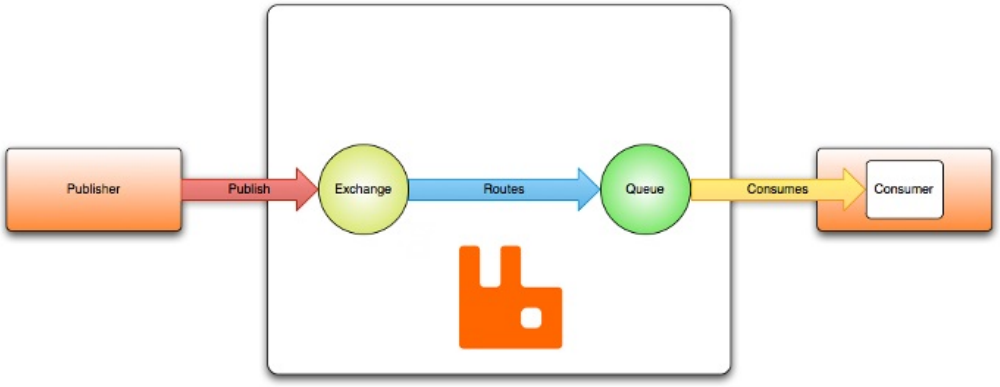

Figura 5.10: Semplice esempio di publish/subscribe in RabbitMQ (fonte [32]).

quanto prodotto ormai noto e maturo che offre non solo un insieme piuttosto ampio di funzionalità ma anche un attivo ecosistema di componenti che poi possono essere sfruttate per aumentarne le possibilità.

RabbitMQ

RabbitMQ[11] è un broker per lo scambio di messaggi general-purpose che costituisce una delle più mature implementazione del protocollo AMQP, oltre a fornire supporto per altri standard come MQTT e STOMP; inoltre librerie client per interagire con esso sono state sviluppate per moltissimi linguaggi e piattaforme, compreso il supporto per dispositivi mobile (ad esempio Android).

In più è possibile espanderne le funzionalità grazie ad un particolare sistema di plugin che è possibile agganciare, aumentandone così la flessibilità di utilizzo: alcuni dei principali riguardano il supporto a protocolli come STOMP e MQTT, oppure per spostare messaggi dalla coda in un determinato broker verso un exchange in un altro (Shovel). Implementato in Erlang, esso fa riferimento ai concetti principali introdotti descrivendo AMQP: il routing dei messaggi avviene tramite l'utilizzo di *exchange* e code di messaggi, ma attraverso il loro utilizzo ed estensione può supportare scenari anche piuttosto avanzati come load balancing o persistenza dei messaggi.

I messaggi sono pubblicati da un publisher attraverso *exchanges*, spesso paragonati a delle *mailbox*, che distribuiscono poi copie di essi alle relative *message queue* secondo regole chiamate *binding*: nel pubblicare il messaggio possono essere specificati i relativi attributi (metadati) che finiscono nell'hea-

[11] Sviluppato da Pivotal, `https://www.rabbitmq.com/`. La maggior parte dei contenuti di questa sezione possono essere reperiti dalla documentazione online di RabbitMQ [32].

der e possono essere utilizzate dal broker per instradarlo, mentre il contenuto del messaggio è ritenuto opaco.

Per garantire una maggiore affidabilità sono definiti i concetti di *acknowledgment* per l'invio e la ricezione di messaggi, rendendo il broker consapevole dei messaggi effettivamente consumati: quando un messaggio viene effettivamente consegnato ad un client, esso notifica il broker, in modo che possa effettivamente decidere quando rimuovere il messaggio dalla coda. Più nello specifico i messaggi possono essere inoltre rifiutati (*message rejecting*) e RabbitMQ prevede anche un modo per rifiutare multipli messaggi tramite *negative acknowledgement*.

Supportando AMQP, RabbitMQ fornisce un'implementazione per tutti i tipi di exchange previsti, i quali forniscono differenti metodi per eseguire l'instradamento dei messaggi, in generale basato sulla proprietà *message routing key*, con la quale si stabilisce in quale coda il messaggio (o una sua copia) debba arrivare, oppure sui valori contenuti nell'header nel caso specifico di un *header exchange*: in questa maniera anche scenari di instradamento anche piuttosto complessi possono essere supportati, oltre ai normali metodi di comunicazione come point-to-point e publish/subscribe.

Esso fornisce inoltre meccanismi di autenticazione per garantire sicurezza delle connessioni ed è possibile configurare alcuni parametri legati allo scambio dei messaggi, come la lunghezza delle code o il loro TTL; non vi è però garanzia riguardo una consegna ordinata dei messaggi per le interazioni in generale, ma solo per il caso in cui si utilizzi un solo exchange a cui è associata una sola coda.

Un certo livello di scalabilità può essere fornito dalla possibilità di organizzare singoli broker in un cluster in maniera trasparente rispetto all'utente, aumentando la disponibilità, le performance e la fault-tolerance del sistema.

5.4 Servizi Web

Nell'ambito di sistemi distribuiti in cui si necessita di forte disaccoppiamento tra le entità in gioco che devono interagire, come il caso di studio considerato per questa tesi in cui esse sono mobili e dislocate geograficamente, una delle architetture che maggiormente si è dimostrata di successo è quella orientata ai **servizi**: attraverso il concetto di servizio, con cui ci si riferisce ad un componente software accessibile dalla rete tramite una ben determinata interfaccia, si riesce infatti ad ottenere un buon livello di interoperabilità astraendo dai dettagli interni dei componenti.

Adottare quindi tale approccio significa affrontare la complessità della realizzazione derivanti dalla distribuzione dividendo i sistemi in unità logiche

semplici e componibili: da questo punto di vista l'esempio di maggior successo è costituito dai **servizi Web**, che negli ultimi anni si sono affermati come standard per la realizzazione di questo tipo di sistemi, anche a causa del grande successo ottenuto dal Web e i suoi principi, caratterizzati da interoperabilità, semplicità e ubiquità di accesso.

In questo contesto, la metodologia per lo sviluppo che ultimamente ha riscosso maggior successo è quella rappresenta dall'applicazione dei principi facenti riferimento allo stile **REST**: rifacendosi al concetto di *risorsa* come elemento fondamentale per la costruzione dei sistemi, esso permette uno sviluppo agile che tenga conto fin dalle prime fasi di progettazione della natura distribuita del problema da affrontare. Diversamente dall'approccio tradizionale basato sull'utilizzo del protocollo SOAP in cui vengono scambiati dati in formato XML e che richiede un ben preciso stack di tecnologie per la realizzazione, il web è alla base di una **ROA** (*Resource Oriented Architecture*), e viene quindi utilizzato da essa come piattaforma sulla quale offrire i servizi che compongono il sistema: per questo motivo molto spesso le è associato l'uso del protocollo HTTP, standard per il web, per le comunicazioni, anche se la validità dei principi espressi da REST è più generale. Ultimamente l'approccio ROA è stato utilizzato con successo anche in ambito IoT, tanto da portare all'introduzione del termine *Web of Things* con cui ci si riferisce alla visione di una molteplicità di oggetti sparsi che hanno la possibilità di comunicare tramite Internet, garantendone così facilità di accesso e relativa semplicità nella realizzazione.

Nella realizzazione di un sistema a supporto della collaborazione per il soccorso in situazioni di emergenza quindi considerare un approccio basato sui servizi Web può essere utile, in quanto permette di garantire fin dalla progettazione l'interoperabilità con un'eterogenea gamma di dispositivi distribuiti proprio come quelli in possesso degli operatori nella missione: questa sezione focalizza quindi l'approfondimento sui principi architetturali indotti da un approccio REST, calandoli in particolar modo nelle esigenze particolari descritte nella fase di analisi, per esplorare possibilità non solo in merito alle comunicazioni di cui si necessita nel sistema, ma anche alla sincronizzazione dei dati.

5.4.1 REST

REpresentetational State Transfer è uno stile architetturale introdotto da Roy Fielding ([33], [34]) per la costruzione di sistemi di ipermedia distribuiti su larga scala, come il WWW: i suoi principi sono stati infatti utilizzati per spiegare e descrivere l'eccellente scalabilità dimostrata nel Web tramite

l'utilizzo del protocollo HTTP, a cui spesso vengono associati seppure trovino validità più generale.

REST rappresenta quindi un'architettura software technology independent che permette a componenti *loosely coupled* di comunicare tramite interfacce su protocolli web standard e fa della *semplicità* uno dei punti cardine: la sua integrazione con principi afferenti alle architetture di rete (portabilità, gestione efficiente della banda, latenze) la rende ideale per sistemi distribuiti in cui la scalabilità ricopre un ruolo importante. Essendo definito in termini architetturali, REST astrae dai dettagli implementativi dei componenti, focalizzandosi quindi sul loro ruolo e la loro interazione: il concetto centrale è quello di **risorsa**, con cui si può identificare qualsiasi informazione a cui possa essere dato un nome, la cui rappresentazione, che ne cattura lo stato corrente, viene scambiata in rete dai componenti del sistema nell'ambito dell'interazione necessaria. Da notare che le risorse possono essere organizzate in collezioni e gerarchicamente, oltre a fare riferimento le une con le altre tramite l'utilizzo di hyperlink.

L'approccio architetturale definito da REST può essere perciò definito in termini di vincoli da applicare ai componenti del sistema, che possono essere riassunti in:

- **interazione client-server**, che promuove il disaccoppiamento dei componenti in termini di client e server, facilitando la divisione delle funzionalità *(separation of concerns)*, e, con essa, la possibilità di evolvere indipendentemente i due tipi di componenti;

- **comunicazioni stateless**, in quanto ogni richiesta che il client effettua deve contenere tutte le informazioni necessarie per essere servita, in modo tale che il server non faccia affidamento su informazioni provenienti da richieste precedenti; in questo modo è il client ad essere responsabile di mantenere lo stato applicativo, i cui cambiamenti avvengono attraverso l'interazione con le risorse;

- **cacheability**, in quanto è possibile definire delle interazioni come cacheable in modo tale che le risposte possano essere prelevate da una *cache*, riducendo il traffico sulla rete e il carico sul server;

- **interfaccia uniforme** per le interazioni, che comprende l'identificazione delle risorse, la manipolazione delle loro rappresentazioni, la presenza di messaggi autodescrittivi e si basa su *hypermedia* come motore dello stato applicativo (HATEOAS); in questo senso un'interfaccia uniforme dovrebbe identificare la semantica e il comportamento di un'interazione senza guardare al contenuto del messaggio che la rappresenta, ma

solo in termini della risorsa sulla quale si vuole agire e dell'operazione richiesta; anche se quest'interfaccia promuove la semplicità dell'architettura e il disaccoppiamento dei componenti, essendo molto generale può risultare non ottimizzata per le esigenze di particolari applicazioni;

- organizzazione **a layer** del sistema, tramite cui si può migliorare la scalabilità e garantire indipendenza nell'evoluzione dei componenti;
- **code on demand**, unico vincolo opzionale, tramite cui è permesso al client di estendere le proprie funzionalità scaricando dal server del codice che poi può eseguire localmente.

Un servizio Web che in generale soddisfa questi principi può essere perciò definito **RESTful**, le cui linee guida per l'implementazione sono definite da una *Resource Oriented Architecture*. Seguendo tale approccio infatti possono essere forniti dei servizi utilizzando il concetto di risorsa, fondamentale per REST: i client accedono così alle risorse utilizzando l'interfaccia comune fornita dai metodi HTTP, i cui messaggi devono essere autocontenuti in accordo con la proprietà di statelessness; il mantenimento dello stato sul client può essere così raggiunto tramite l'utilizzo di un suo esplicito trasferimento, ad esempio indicando lo stato futuro in cui transitare oppure gli **hyperlink** di altre risorse cui è possibile accedere. Le risorse sono inoltre disaccoppiate dalla rappresentazione che viene trasferita nell'ambito di un'interazione, in modo tale che esse possano essere accedute in vari formati (molto spesso in formato XML o JSON), di cui ovviamente il client deve essere a conoscenza, e in più è possibile fare uso di metadati ad esse relativi (come quelli per il controllo della cache).

Utilizzando HTTP come protocollo stateless per la comunicazione client-server, i metodi per l'interfaccia sono `PUT` per la creazione di una nuova risorsa o la modifica del suo stato, `DELETE` per la sua cancellazione, `GET` per ottenerne una rappresentazione dello stato corrente e `POST` per la creazione di una nuova sotto risorsa in una collezione o il trasferimento di un nuovo stato su di essa: risulta evidente quindi immediato il riferimento alla possibilità di effettuare tipiche operazioni CRUD sulle risorse. Come si può notare, non è ancora stata del tutto definita nettamente la differenza nell'utilizzo dei verbi `POST` e `PUT`, ma in [35] viene riportata una convenzione generalmente utilizzata per cui `POST` viene utilizzato specificatamente per la creazione di una risorsa il cui URI è generato dal servizio Web, soprattutto nel caso essa debba essere aggiunta ad una collezione, mentre `PUT` è più indicato per creare o modificare risorse il cui URI è computato dal client.

In genere è quindi consigliabile l'utilizzo di di un'architettura basata su REST per la realizzazione di servizi Web quando la flessibilità e la velocità

di implementazione sono caratteristiche chiave: questo approccio rappresenta una soluzione abbastanza leggera e agile se comparata a quella offerta dall'approccio SOA, e la semplicità gioca un ruolo chiave, abbassando i requisiti per i componenti che fanno parte del sistema e prevedendo quindi anche l'integrazione con device mobile.

Nel contesto particolare indotto dal caso di studio quella di risorsa può rappresentare un'utile astrazione con il quale organizzare il sistema: trattando infatti in maniera principali dei dati, essi possono essere rappresentati da risorse a cui associare delle rappresentazioni del proprio stato, ad esempio un paziente e il suo stato di salute. Questo mapping concettuale tra elementi del dominio e del modello concettuale di riferimento semplificano lo sviluppo del sistema, dando allo sviluppatore le giuste chiavi di astrazione: nulla però in REST è specificato per quanto riguarda la sincronizzazione dei dati, che risulta essere una feature da dover implementare sopra questo livello, ad esempio sfruttando caratteristiche come la cacheability, per capire quali risorse siano state modificate.

5.4.2 Cacheability

Mantenere una cache dei dati ricevuti tramite l'interazione con un servizio Web è uno dei modi per aumentare le sue performance, dal momento che riduce l'ammontare di traffico sulla rete, ma può essere utilizzato anche per altri scopi: nel contesto specifico di questo lavoro, in cui la sincronizzazione dei dati fra client e server gioca un ruolo centrale, essa può essere utile per non trasferire dal server i dati che non sono stati modificati, offrendo un primo meccanismo per un "basilare" controllo di versione.

Il protocollo HTTP, su cui si basa l'implementazione della maggioranza dei servizi Web basati su REST sviluppati oggigiorno, definisce particolari campi che fanno parte dell'**header**[12] del messaggio scambiato per controllare il meccanismo di caching, alcuni dei quali risultano piuttosto utili per controllare se, lato server, una risorsa è stata modificata dall'ultimo accesso effettuato da un determinato client.

Un primo modo per gestire la cacheability di una risposta è associato al concetto di tempo in cui è stata effettuata l'ultima relativa richiesta, utilizzando l'header `cache-control`, con il quale si può controllare, tra le molte cose, se ri-validare la risposta ogni volta (`no-cache`), e il tempo di validità della cache, indicato tramite il valore `max-age`, che dovrebbe avere preceden-

[12]La definizione degli header del protocollo HTTP può essere reperita sul sito del W3C `https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html` (consultato il 20/02/16).

za sul valore definito da `expires`, con il quale si può impostare la data e l'ora dopo la quale la risposta in cache non è più considerata valida.

Un meccanismo che fornisce invece un controllo più fine sul valore delle risorse richieste è invece rappresentato dall'utilizzo del campo chiamato `ETag`, valore secondo il quale il client può effettuare delle richieste condizionali: in questo modo, se la risorsa non è stata aggiornata, il server non necessita di inviare il contenuto all'interno della risposta, migliorando il consumo della banda. Tramite il suo utilizzo può essere inoltre implementato un basilare meccanismo di *optimistic concurrency control*, in quanto un client che desidera modificare una risorsa ha la possibilità di capire se la versione salvata sul server è quella di cui aveva conoscenza o meno. `ETag` è quindi un identificatore opaco, tipicamente in forma di hash del contenuto o di timestamp anche se il suo formato è dipendente dall'applicazione e non fa parte della specifica HTTP, assegnato dal server ad una risorsa per identificarne la versione, e ad ogni modifica deve essere aggiornato per adempiere correttamente alle sue funzioni: in questo senso, è ovviamente necessario che sia il client che il server siano predisposti per il suo utilizzo.

In un tipico scenario di utilizzo, quando un client richiede una risorsa (e.g. tramite una `GET`) il server risponde con la sua rappresentazione attuale corredata, nell'header, del valore `ETag` corrispondente, in modo da memorizzarla nella cache; se in un secondo momento il client invia nuovamente una richiesta per la stessa risorsa, può includere il valore `ETag` precedentemente ricevuto nell'header `If-None-Match`, in modo che il server invii nella risposta la rappresentazione della risorsa solo se il valore corrente dell'entity-tag della risorsa è diverso da quello contenuto nella richiesta, altrimenti inviando una risposta con status HTTP `304 Not Modified`.

Nel caso in cui volesse modificare nel server una risorsa di cui ha ricevuto la rappresentazione in precedenza, ma di cui è a conoscenza che potenzialmente potrebbero essere avvenuti concorrenti cambiamenti provenienti da altri client, può utilizzare il valore di `ETag` nel campo `If-Match` per eseguire la richiesta di modifica solo se il valore della risorsa corrisponde a quello di cui il client era a conoscenza, ricevendo altrimenti una risposta con status `412 Precondition Failed` che indica che non è stato possibile servire correttamente la richiesta.

5.4.3 Tecnologie

Data la grande diffusione dell'architettura REST per la costruzione di servizi Web, attualmente esiste una grande varietà di strumenti e piattaforme che abilita e facilita il loro sviluppo, offrendo le giuste metafore al programmatore e sollevandolo da tipici dettagli implementativi. Fra le tante alternative, per

l'approfondimento svolto all'interno di questo lavoro è stato considerato uno dei framework maggiormente utilizzati in riferimento al linguaggio Java, cioè **Restlet**, in modo da dare una panoramica dei concetti che offre e di come può aiutare nello sviluppo di un sistema come quello considerato per il caso di studio, fornendo le giuste metafore tramite API di alto livello. Un altro strumento molto utilizzato risulta essere **Jersey**, un framework open source che costituisce l'implementazione di riferimento delle API Java *JAX-RS*, nate per semplificare lo sviluppo di questo tipo di servizi web tramite l'utilizzo di *annotazioni*: esso offre così API per l'implementazione agevole di client e server, semplificando la gestione delle comunicazioni e ampliando il supporto anche con funzionalità non presenti nella specifica e riguardanti integrazioni con vari ambienti deploy o strumenti web comunemente utilizzati. Per quanto riguarda altri linguaggi, quelli più popolari che supportano in qualche forma lo sviluppo di servizi REST(ful) risultano essere **Django** in Python e **Ruby on Rails** per Ruby.

Restlet

Restlet[13] è un framework open source object-oriented per lo sviluppo di servizi web RESTful in Java: tramite le API e le astrazioni che mette a disposizione è possibile così realizzare applicazioni web che sfruttino i principi descritti dallo stile REST per ottenere scalabilità e interoperabilità. Esso fornisce un esteso insieme di classi e funzionalità che possono essere utilizzate ed eventualmente estese per permettere allo sviluppatore di concentrarsi principalmente sulla business logic dell'applicazione: fattorizzando infatti i componenti più comuni nella costruzione di questi sistemi risparmia lo sviluppo di molto codice, e quindi tempo.

Tramite il suo utilizzo è possibile sfruttare le potenzialità del web, come ad esempio le caratteristiche del protocollo HTTP per quanto riguarda caching, autenticazione e sicurezza e le tecnologie maggiormente utilizzate nel suo ambito come XML, JSON, SMTP, Atom.

Inoltre esistono differenti versioni del framework specializzate per piattaforma, con il supporto per Android, Google App Engine, Google Web Toolkit, Java SE e EE e OSGi ed inoltre è possibile eseguirlo in maniera standalone all'interno di una JVM oppure integrarlo in ambienti di deploy maggiormente complessi come servlet container.

Concetti essenziali Il framework si compone essenzialmente di due parti: *Restlet API*, rappresentanti le API per il supporto a REST e HTTP che

[13] `https://restlet.com/`

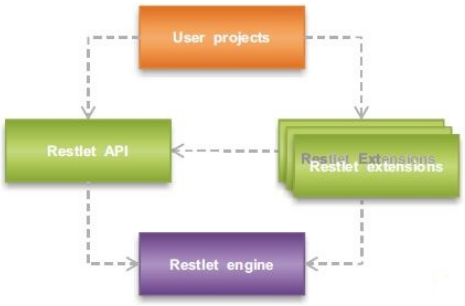

Figura 5.11: Architettura a livelli di Restlet (fonte [36]).

facilitano l'implementazione di applicazioni sia server che client side, e *Restlet Engine* che ne provvede all'implementazione: questa separazione fra API e implementazione rende il framework flessibile e permette di supportare differenti tecnologie di deploy.

Entrambe queste parti fanno parte del *core* del framework distribuito tramite il jar `org.restlet.jar`: ad esso però può essere aggiunto l'uso di diverse estensioni (ad esempio per supporti o integrazioni particolari a causa di specifici requisiti dell'applicazione) che contribuiscono così ad offrire una soluzione completa e ricca pur mantenendo un nucleo minimale di funzionalità che rappresentano il punto focale del sistema. Le principali feature offerte dall'utilizzo di Restlet derivano dal mapping effettuato con i principali elementi individuati per il modello REST, in quanto provvede a fornire API per Java per concetti come:

- *Risorse*, il concetto fondamentale secondo cui vengono organizzate le applicazioni, in modo da fornirne agevole accesso;
- *Componenti*, contenitori logici per le applicazioni Restlet;
- *Connettori*, che abilitano la comunicazione tra entità REST;
- *Rappresentazioni*, per esporre lo stato delle risorse.

Inoltre il framework provvede a dare un'*interfaccia uniforme* per accedere in maniera standard alle risorse; l'utilizzo delle API permette anche di astrarre da molti dettagli tecnici del protocollo HTTP, potendone comunque usufruire delle potenzialità in termini di *negoziazione del contenuto*, con cui si può scegliere la variante di rappresentazione della risorsa, compressione automatica, richieste condizionali (ad esempio tramite ETag) e *caching*. In più è supportata anche una forma di routing dinamica, in contrapposizione

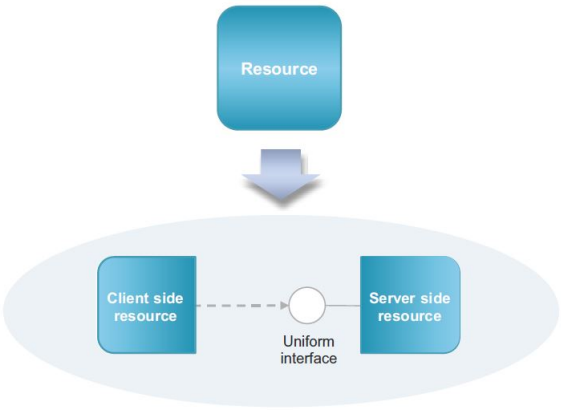

Figura 5.12: Decomposizione di una risorsa in concetti di Restlet (fonte [36]).

a quella statica offerta dalle Servlet tramite configurazione di file XML o annotazioni.

Per quanto riguarda il concetto di risorsa, è interessante notare come, per fornirne un accesso agevole, essa venga "spezzata" in una parte server e in una parte client tramite cui si vuole accedere ad essa: lato client quindi l'entità può essere considerata come un proxy locale per la risorsa remota fisicamente collocata nel server; `Resource`, `ClientResource` e `ServerResource` sono classi che lo sviluppatore può estendere e usare a suo piacimento per rappresentare le risorse che l'applicazione deve esporre, e rispettivamente definiscono la classe astratta per le risorse, una classe per la gestione di risorse lato server, che quindi si occupa di gestire in modo corretto le azioni eseguite sulla risorsa, ad esempio agendo sulla rappresentazione a seconda del metodo HTTP usato e la classe di utilizzo del client per accedervi, generando richieste HTTP.

Inoltre, in Restlet è possibile usare alcune delle annotazioni espresse nella JAX-RS API per semplificare lo sviluppo, ossia `@Get`, `@Put`, `@Delete`, `@Post` e `@Options`, rispettivamente per identificare metodi dalla semantica simile agli omonimi in HTTP.

Una delle particolarità di Restlet è proprio quella di non fare una netta distinzione fra i concetti di client e server, in quanto un server potrebbe assumere il ruolo di client nel consumare dei dati provenienti da un differente servizio web, scenario oggi usuale nei casi di sviluppo di applicazioni *mash-up*.

Anche se Restlet può essere utilizzato come una libreria da cui prendere le astrazioni di cui si necessita, uno dei suoi maggiori utilizzi riguarda la gestione e l'organizzazione dell'applicazione web da sviluppare nella sua completezza, e a questo proposito offre il concetto di *application*, che fornisce un modo per

raggruppare e organizzare l'insieme coerente di risorse che fanno parte del servizio. Un oggetto *application* in Restlet ha un design a livelli, e come detto può gestire sia *inbound calls* lato server che *outbound calls* lato client: una richiesta in arrivo da un client verso una risorsa lato server passa attraverso un livello i *service filtering*, comune a tutte le risorse dell'applicazione, seguito da un livello di *user routing* dove e possibile fare autenticazione e routing della richiesta verso una determinata risorsa, ad esempio a seconda del suo URI, per poi arrivare allo strato di *resource handling*, dove la risorsa target gestisce la richiesta e fornisce la rappresentazione al client. Analogamente attraverso questi livelli possono essere effettuate richieste tramite risorse di tipo client dall'applicazione, anche all'interno della gestione di risorse server, tipicamente richiedendo l'utilizzo delle funzionalità un servizio remoto.

Nel framework è presente inoltre il concetto di *Component*, utile per adattare il deploy di applicazioni Restlet a seconda dell'ambiente target, le quali possono comunicare tra loro attraverso l'uso di *Connectors*, di cui ne esistono differenti tipi, anche provvisti tramite estensioni: gli esempi principali sono rappresentati dal connettore client, che inizia la comunicazione con un altro di tipo server, che rimane in attesa di richieste.

Anche il concetto di rappresentazione è trattato come entità di prima classe del framework, con il quale si possono gestire le diverse varianti in termini di rappresentazione delle risorse, riuscendo così a sfruttare anche le potenzialità di *contet-negotiation* di HTTP: sono infatti contemplati molti casi, tra cui quelli di più comune utilizzo come XML e JSON per la serializzazione/deserializzazione di oggetti.

Bisogna inoltre riportare il fatto che, pur promuovendo la filosofia REST, il framework permette l'utilizzo di elementi che non fanno parte del modello, come ad esempio i *cookies*, meccanismo utilizzato pervasivamente nel web per mantenere sessioni.

In conclusione, Restlet rappresenta un framework che facilita lo sviluppo di servizi web RESTful fornendo in Java i concetti chiave del modello come astrazioni che lo sviluppatore può utilizzare: tramite la composizione delle necessarie risorse e rappresentazioni in applicazioni, è infatti possibile offrire servizi ricchi non dovendosi soffermare su numerosi dettagli implementativi, visto che Restlet presenta supporto per numerosi ambienti di deploy, risultando di fatto agnostico anche rispetto al livello tecnologico utilizzato per la persistenza delle informazioni. Esso fornisce un buon livello di supporto per le tipiche operazioni da effettuare sulle risorse, permettendo inoltre di sfruttare le potenzialità insite nel protocollo HTTP con il controllo degli header, mappati all'interno del framework: è in questo modo che è possibile sfruttare, ad esempio, un `ETag` per usufruire di un minimo livello di controllo di versione sulle risorse.

Capitolo 6

Valutazione

L'esplorazione del panorama tecnologico descritta sinteticamente nel capitolo precedente ha evidenziato le principali caratteristiche della problematica di sincronizzazione dati calata nel particolare contesto applicativo considerato, mettendo in rilievo per ogni area tecnologica individuata le funzionalità offerte per innalzare il livello di astrazione nello sviluppo.

In questo capitolo vengono così sintetizzate le considerazioni più rilevanti emerse, focalizzandosi in particolar modo su quale degli approcci sembri più adatto: è in questo senso che, dopo aver individuato l'area dei database distribuiti come quella di maggior interesse, viene descritta la breve fase di sperimentazione sulla tecnologia scelta (**CouchDB**) in modo da poterne concretamente valutare l'apporto che può fornire.

6.1 Considerazioni sull'esplorazione

Come evidenziato nella fase di analisi del problema per il caso di studio, le maggiori problematiche da affrontare per permettere una buona condivisione di dati in un contesto di sistema di mobile computing sono *bidirezionalità* e *sincronizzazione*. Più in particolare, un ideale middleware dovrebbe abilitare una forma di comunicazione coordinata persistente, in modo da permettere agli operatori mobile di accedere ai dati anche se non dispongono di connessione: nel contesto di questo lavoro vengono quindi considerate le caratteristiche abilitanti per il caso particolare considerato, non tralasciando però la possibilità di supportare eventuali modifiche ai requisiti.

6.1.1 MOM

É proprio dal punto di vista della comunicazione che i **MOM** possono dare il loro apporto: tramite l'utilizzo del paradigma a scambio di messaggi essi abilitano infatti forme più avanzate di comunicazione rispetto alla tipica interazione *client-server*, tra le quali le più utilizzate sono point-to-point e publish/subscribe. Il concetto di coda di messaggi, in particolare, abilita anche una forma di persistenza delle informazioni e delle comunicazioni che torna molto utile in contesti dove le entità sono fortemente disaccoppiate e distribuite tra loro come quello considerato: nel caso un dispositivo non risulti connesso, il messaggio viene mantenuto nella coda per essergli consegnato quando nuovamente sarà disponibile. Essi quindi possono essere molto utili in contesti di integrazione di sistemi di diversa natura, promuovendone l'indipendenza e l'interoperabilità grazie al medium di comunicazione che mettono a disposizione, semplificando di fatto la loro interazione, ma non mettono a disposizione funzionalità dedicate dal punto di vista della sincronizzazione dei dati, meccanismo che quindi andrebbe implementato in maniera custom sopra il livello di astrazione che offrono.

Un possibile approccio di realizzazione in questo senso sarebbe adottare il pattern publish/subscribe attraverso il quale ogni dispositivo pubblica le informazioni da lui generate riguardo a un determinato paziente in modo che chiunque ne sia interessato le riceva: questa soluzione dovrebbe però prevedere comunque l'integrazione con un livello di persistenza delle informazioni (e.g. database) ulteriore a quello, molto spesso temporaneo, che i MOM offrono. Oltretutto, molto spesso questi strumenti non trovano una controparte di supporto per dispositivi mobile ufficialmente supportata, anche se da questo punto di vista gli strumenti open source hanno un'attiva comunità alle spalle che ne produce qualche esempio.

Essi mostrano quindi la loro utilità soprattutto nel promuovere la *reattività* della comunicazione, ad esempio per lo scambio di informazioni real-time: in questo senso la loro introduzione potrebbe essere utile nel caso, non considerato in questa tesi, della necessità di un'architettura a eventi, dove anche lato backend si potrebbe voler distribuire i sottosistemi per potenziarne le performance, mentre per il particolare caso di studio sembrano indicati solo per il ruolo di eventuale componente per la gestione delle comunicazioni, a cui vanno affiancati un meccanismo di sincronizzazione e l'integrazione con un necessario livello di persistenza delle informazioni, a carico dello sviluppatore.

6.1.2 Servizi Web

L'utilizzo di un **servizio web** RESTful presenta diverse caratteristiche interessanti per la realizzazione uno spazio informativo condiviso: considerando come fondamentale il concetto di risorse ed organizzando il sistema attorno alla loro composizione, la filosofia proposta si sposa bene con la natura *data-centrica* del problema da affrontare, facilitando il mapping fra informazioni e risorse.

Inoltre l'uso di tecnologie web standard (come HTTP) permette un grande livello di interoperabilità grazie al fatto che sono praticamente supportate dalla maggior parte dei dispositivi attualmente presenti, mobile device compresi, rendendo possibile l'accesso ai dati da un insieme molto eterogeneo di essi.

La complessità di realizzazione di tali servizi può essere notevolmente abbassata servendosi di appositi framework, quali Restlet (5.4.3), che offrono un livello di astrazione basato sulla programmazione ad oggetti per uno sviluppo agile e flessibile ma al contempo ricco di possibilità, date le numerose estensioni che ognuno di essi presenta per integrazioni con vari e numerosi strumenti, tra cui l'utilizzo di database per memorizzare in maniera persistente le informazioni.

In più il fatto che il sistema presenti nella centrale operativa un punto di centralizzazione ben si confà alla natura client-server dei sistemi web, che possono però porre delle limitazioni per quanto riguarda la bidirezionalità nelle comunicazioni, visto che in questo paradigma ad iniziarle è solitamente il client: a parte l'esistenza di vari metodi per estendere il paradigma ed abilitare questo tipo di comunicazione dal server al client (detta *push*), si può anche notare che, seppure a livello logico la bidirezionalità nelle comunicazioni è richiesta, a livello implementativo essa potrebbe essere realizzata tramite *polling*, rientrando così nei canoni tradizionali.

Le astrazioni di risorsa e sua rappresentazione, unite alla caratteristica di *statelessness* delle interazioni, che secondo il modello REST devono essere autocontenute, conferiscono scalabilità a questo approccio e lo rendono perciò adatto a una grande varietà di scenari, come si può notare dalla sua enorme diffusione nella costruzione di sistemi distribuiti, ultimamente anche in ambito di sistemi embedded ed IoT.

Per quanto riguarda il caso specifico considerato però, gli strumenti analizzati non offrono nativamente delle soluzioni *off-the-shelf*, preconfezionate, per realizzare dei meccanismi di sincronizzazione dati: questo approccio fornisce però i **building block** necessari per realizzare la propria implementazione, ad esempio utilizzando il *caching* delle risorse tramite header

dei messaggi HTTP, con cui si può capire, senza effettivamente ricevere il contenuto della risorsa, se essa lato server è stata modificata.

Ad esempio, visto che la coordinazione delle informazioni a disposizione degli operatori avviene tramite la comunicazione con un server centrale, esse potrebbero essere esposte come risorse di un servizio web, permettendo così ad un **agente** presente sul loro dispositivo mobile di recuperarle a seconda delle condizioni di rete, effettuando *polling* su un certo sottoinsieme delle risorse, organizzate in maniera gerarchica e mantenute persistentemente lato server, comprendendo così l'integrazione con un modulo di memorizzazione dei dati. Questo componente dovrebbe essere perciò realizzato in termini di *client web* che, attraverso l'interfaccia fornita dal servizio, recuperi in qualche modo solamente le informazioni in più o modificate dall'ultimo atto di sincronizzazione, in maniera da non appesantire il carico di rete, date le precarie condizioni di connessione e preveda anch'esso un livello di persistenza dei dati in modo da fornire un certo livello di trasparenza rispetto alla connessione in rete.

Sebbene quindi l'utilizzo di servizi web costituisca una buona base per l'implementazione del sistema, visto che incorpora nativamente alcune delle caratteristiche necessarie quali interoperabilità, distribuzione e flessibilità, esso non permette di colmare completamente l'abstraction gap evidenziato, lasciando il compito allo sviluppatore di implementare, sopra il livello di astrazione fornito dall'utilizzo del particolare framework e degli strumenti utilizzati per il livello di persistenza delle informazioni, le politiche di sincronizzazione necessarie.

Bisogna inoltre contare che, da questo punto di vista, particolarmente difficoltosa risulterebbe la gestione dei conflitti, in quanto data la natura del problema, che prevede che gli operatori possano eseguire le loro mansioni anche offline generando informazioni, la strategia per il controllo concorrente degli accessi deve essere *ottimistica*: date anche le particolari caratteristiche del dominio, il caso di studio non dovrebbe presentare particolari problemi in merito, come evidenziato in 3.5, ma essere a conoscenza di questa evenienza e considerarla può essere decisivo per future estensioni.

6.1.3 DBMS Distribuiti

Quella riguardante i database distribuiti è di certo l'area tecnologica che per caratteristiche più si avvicina all'astrazione necessaria per il caso di studio: infatti la metafora è quella di un insieme di dati logicamente correlati distribuiti in vari nodi della rete, nel caso particolare costituiti da una centrale operativa e dispositivi mobili che non sempre dispongono di connettività.

La necessità di un database che mantenga memorizzati i dati anche sui dispositivi mobili emerge dalla **trasparenza** che si vuole fare percepire all'utente nel loro accesso rispetto alla mancanza di connettività, visto che gli deve essere permesso di conoscere le informazioni relative ai pazienti anche qualora non potesse direttamente temporaneamente connettersi alla centrale operativa: è ovvio che in questo caso l'utente deve essere consapevole che i dati potrebbero non essere aggiornati, e quindi essere conscio del suo stato offline.

Adottando un database distribuito lo sviluppatore potrebbe infatti demandare a quel livello la gestione di problematiche riguardanti la consistenza e la sincronizzazione dei dati, concentrandosi così esclusivamente sulla logica applicativa: l'idea è quella di avere un unico *workspace condiviso* tramite il quale gli operatori salvano le informazioni da loro generate nel processo di soccorso ai feriti e contemporaneamente possono usufruire di quelle inviate alla centrale operativa dai loro colleghi, tecnicamente costituito dalle varie copie distribuite dei dati che risiedono sui dispositivi.

A seconda della particolare tecnologia adottata svariate possono essere le funzionalità offerte: in questo lavoro quella analizzata è rappresentata da **CouchDB**, che presenta molte delle caratteristiche necessarie per l'ambito analizzato. Infatti le principali funzionalità di interesse che espone sono:

- modello dei dati NoSQL orientato a documenti di tipo JSON, che costituiscono l'unità fondamentale con cui salvare i dati;
- utilizzo di una **API HTTP** che si rifa ai concetti di REST per l'accesso ai dati, permettendo di fatto le operazioni CRUD su di essi attraverso l'esposizione di un servizio web, che garantisce interoperabilità e supporto per l'eterogeneità dei dispositivi;
- sistema di **versioning** (Multi Versioning Concurrency Control) integrato che mantiene traccia dei cambiamenti effettuati sui dati in modo da identificare eventuali conflitti nell'accesso e permetterne la gestione;
- meccanismo di replica **multi-master** tramite HTTP per tenere sincronizzate due istanze distribuite del database, utilizzando una politica di **optimistic locking** per quanto riguarda l'accesso offline ai dati;
- flessibilità nella replica dei dati grazie all'applicazione di funzioni di filtro con la quale decidere quali dati sincronizzare;
- presenza di una libreria di supporto (sviluppata da Couchbase) per lo sviluppo su device mobile (Android o iOS) per il salvataggio dei dati e la sincronizzazione in linguaggio nativo, che offre quindi tutte le

funzionalità necessarie per la gestione dei dati locale ai dispositivi degli operatori, permettendone l'accesso tramite query.

É evidente che questa tecnologia è quella che offre il livello di astrazione più alto, dato che la maggior parte delle funzionalità necessarie per un livello infrastrutturale che si occupi della sincronizzazione dati sono coperte dalle sue caratteristiche, prima fra tutte l'esposizione di un servizio web tramite cui interagire con i dati, non solo per accedervi ma anche per gestire il processo di sincronizzazione; da questo punto di vista può essere così individuato un punto di contatto con la categoria tecnologica dei servizi web descritta nella sottosezione precedente (6.1.2), mostrando come di fatto essi siano uno standard per la realizzazione di sistemi distribuiti e interoperabili.

É inoltre da notare che l'utilizzo lato server (centrale operativa) di CouchDB ben si sposa anche con l'ambiente operativo prefigurato, costituito in linea di principio da una sola macchina: in questo contesto di deploy non si può quindi prevedere un'infrastruttura molto pesante, e quindi l'assenza di un componente di middle-tier per la sincronizzazione, strategia molto adottata per i database distribuiti (di cui esempi possono essere Couchbase Server e Sync Gateway oppure Oracle Mobile Server) può essere considerata un vantaggio, anche considerando il carico di lavoro stimato che non risulta essere problematico.

Infine va ricordato anche il fatto che CouchDB permette in linea di principio anche una *sincronizzazione continua*, in cui i dispositivi rimangono in continua attesa di cambiamenti avvenuti lato server: nel caso particolare considerato però sembra presentare vantaggi la realizzazione di un componente *network aware* che, a seconda delle condizioni di rete, gestisca il processo di sincronizzazione tramite le API messe a disposizione dalla libreria per dispositivi mobile con una certa frequenza, in modo anche da non sovraccaricare la rete.

In conclusione quindi CouchDB appare come un'alternativa molto interessante per il caso di studio considerato, in quanto fornisce praticamente tutte le metafore e le astrazioni di cui si necessita per sollevare lo sviluppatore dalla gestione dei dettagli della sincronizzazione: inoltre, utilizzando HTTP come metodo di accesso ai dati e un formato JSON per la loro rappresentazione esso abbraccia il paradigma web garantendo così le proprietà che lo caratterizzano e affrontando anche il problema delle comunicazioni distribuite. Da ricordare inoltre la possibilità, nel caso si debba sottostare a necessità molto particolari non supportate dalla libreria per mobile device, che è in linea di principio possibile sviluppare un client custom che, sfruttando le API messe a disposizione tramite servizio REST su HTTP da CouchDB, usufruisca delle possibilità da esso offerte.

6.1.4 Confronto

A partire dalle considerazioni svolte nella sezione precedente sugli approcci per la realizzazione di un middleware per sincronizzazione dati in un contesto di mobile computing si può delineare un confronto tra le due soluzioni che sembrano maggiormente percorribili:

- sviluppo di un servizio web RESTful custom con cui gestire l'interazione con le informazioni, modellate come risorse, e la loro sincronizzazione;
- utilizzare una tecnologia esistente che già prevede delle funzionalità di sincronizzazione di dati memorizzati in maniera persistente, come l'accoppiata CouchDB lato server e Couchbase Lite lato client.

Non considerando infatti i MOM che, come già descritto, dovrebbero comunque essere accoppiati ad un servizio web che gestisca le informazioni, dato che la console per la *control room* è stata definita come applicazione web e visto che presentano la loro maggiore utilità nell'abilitare diversi tipi di comunicazione, i due diversi approcci rimasti sono stati valutati effettuando un mapping delle loro caratteristiche e funzionalità con i *layer* richiesti per il middleware ipotizzato nella sezione 4.3, espressi sinteticamente dalle figure 6.1 e 6.2.

Servendosi di un framework come *Restlet* è infatti possibile implementare agilmente e flessibilmente un servizio web RESTful secondo le proprie esigenze: il maggior vantaggio di questa soluzione è infatti quello di poter soddisfare in maniera puntuale le esigenze che il problema pone visto che in pratica viene fornita una piattaforma per lo sviluppo di applicazioni web distribuite, ma di contro è sulle spalle dello sviluppatore il peso dell'implementazione di tutta la logica di sincronizzazione dei dati, compresa l'individuazione dei cambiamenti (risorse aggiunte o modificate) e l'eventuale gestione di conflitti nel loro accesso; inoltre per completarlo è necessario anche lo sviluppo di un componente che, sul dispositivo mobile, si occupi di memorizzare le informazioni ed aggiornarle sulla base dei bisogni dell'utente.

Questa problematica, che risulta abbastanza difficile da trattare, potrebbe essere affrontata in questo approccio in diversi modi:

- lato client il componente addetto esegue polling sulle risorse esposte tramite servizio web dalla centrale operativa, utilizzando una politica di *caching* per minimizzare le richieste in rete e ridurre così il traffico; questa soluzione risulta essere la più semplice dal punto di vista concettuale e nel caso specifico, dato il carico di lavoro non troppo ampio, potrebbe anche rivelarsi efficace, anche se rimane da gestire la sincronizzazione lato client delle informazioni;

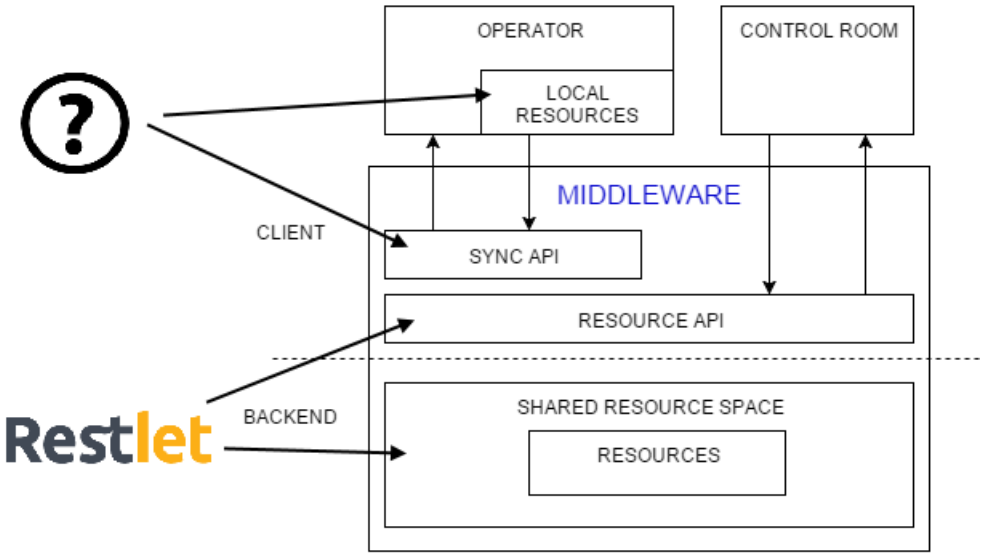

Figura 6.1: Mapping dell'utilizzo di Restlet nei livelli per il middleware individuati nel capitolo 4.

- adottare una strategia a *log* con cui il client e il server memorizzano i cambiamenti effettuati sulla base di dati ed esporli tramite un apposito servizio web, tramite cui i client possono recuperare le modifiche occorse dall'ultima sincronizzazione.

É comunque evidente che lo sviluppo di una tale infrastruttura, per quanto un framework abilitante per servizi RESTful possa essere utile ed innalzare il livello di astrazione, comporti una mole di lavoro non banale da affrontare, soprattutto per quanto riguarda la gestione della sincronizzazione: come si può infatti notare dalla figura 6.1 tramite le API offerte da Restlet si può ottenere una facile organizzazione del sistema in termini di risorse e anche facilitarne l'accesso dai client, ma resta comunque scoperto il livello che permette di gestire i loro aggiornamenti.

Per questo motivo la soluzione che più sembra essere adatta in tale contesto è rappresentata dall'utilizzo di un database come **CouchDB** che memorizzi lato server le informazioni sui pazienti e contemporaneamente offra un servizio web che permetta a database dal formato compatibile con il suo e residenti in altri nodi della rete di sincronizzarsi: da questo punto di vista è come se questo approccio fosse un'evoluzione e un'estensione di quello precedente, visto che tramite un servizio web si prefigge l'obiettivo di affrontare la distribuzione e l'interoperabilità fra i vari sottosistemi, ma ci aggiunge un li-

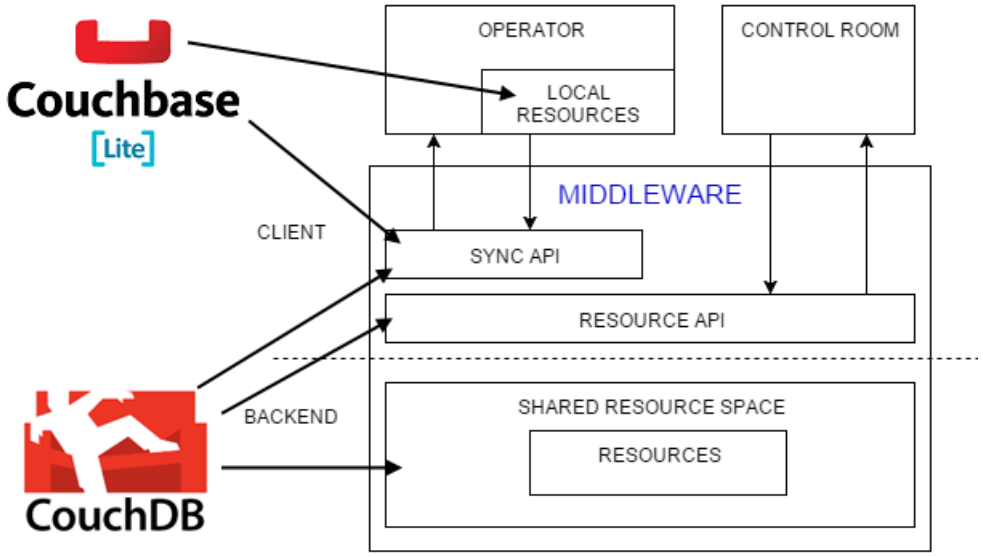

Figura 6.2: Mapping dell'utilizzo di CouchDB e Couchbase Lite nei livelli per il middleware individuati nel capitolo 4.

vello di persistenza delle informazioni che già gestisce il versioning e l'accesso concorrente alle risorse.

Se inoltre si considera che esiste anche una libreria che gestisce la persistenza dei dati sul dispositivo mobile ed implementa nel suo linguaggio nativo le API per raggiungere il servizio web, allora è evidente che questo approccio è quello che offre il maggior livello di astrazione allo sviluppatore, fornendogli una piattaforma di gestione dati completa e anche con qualche grado di flessibilità: come si può notare dalla figura 6.2 infatti tutti i livelli del middleware sono coperti dall'utilizzo di questi due strumenti, con il risultato che lo sviluppatore può concentrarsi sulla logica applicativa.

La possibilità che l'utente acceda o modifichi offline le informazioni (ad esempio aggiungendone di nuove) viene gestita tramite la memorizzazione sul suo dispositivo, la *replica* dei dati come servizio su HTTP e il meccanismo di versioning implementato, che permettono a due istanze di database di conoscere i cambiamenti avvenuti dall'ultima sincronizzazione e trasferire in rete solo quelli.

In conclusione, è evidente che l'approccio dato dall'utilizzo di un database distribuito come CouchDB per la realizzazione di un livello infrastrutturale che permetta una sincronizzazione agevole e trasparente rispetto alla presenza di connessione dei dati sia quello che più facilita lo sviluppatore, fornendogli gli strumenti di cui necessita: compito dello sviluppatore è perciò solo quello

di coniugare il modello dei dati del dominio con quello NoSQL offerto dalla tecnologia usata, in cui in particolare ogni database è costituito da una collezione di documenti JSON e lo sviluppo di un componente che effettui lo *scheduling* dei task di sincronizzazione a seconda del livello di connettività. In particolare, il suo utilizzo si concilia bene con i requisiti esposti per questo livello in 4.3:

- la **trasparenza** della sincronizzazione rispetto alle azioni dell'utente viene gestita grazie al fatto che basta salvare localmente al suo dispositivo le informazioni, poi trasmesse alla centrale operativa dal componente che si occupa di avviare il processo di sincronizzazione;
- la **disponibilità** del sistema anche in caso di disconnessione è garantita dal fatto che le informazioni vengono memorizzate localmente sui dispositivi degli operatori, anche se comunque la *network awareness* permette loro di essere consci del fatto che le informazioni potrebbero non essere aggiornate;
- **flessibilità** dato che è possibile decidere, tramite apposite funzioni di filtro, quali dati scambiare;
- supporto per lo sviluppo su dispositivi mobile grazie alla libreria Couchbase Lite, che permette uno sviluppo agevole anche su tali dispositivi.

Inoltre, anche se per il caso in questione non dovrebbe servire, viene garantita la possibilità di gestire eventuali conflitti nella modifica di informazioni grazie al fatto che le versioni *in conflitto* vengono mantenute, in modo tale da permetterne la gestione e la risoluzione: in questo modo viene garantita anche l'**estendibilità** del sistema, in quanto anche a fronte di nuovi requisiti per la sincronizzazione che introducessero dei potenziali conflitti, l'utilizzo della tecnologia non verrebbe invalidato.

Di contro, questa soluzione vincola l'utente ad adattare il proprio modello di dati a quello NoSQL offerto dallo strumento, in cui viene persa una forte strutturazione dei dati che deve essere ricomposta tramite collegamenti fra i vari documenti ed inoltre essendo di carattere generale potrebbe non essere ottimizzata per la particolare evenienza.

A fronte di questo confronto è evidente che l'utilizzo di uno strumento come CouchDB rappresenti la soluzione più idonea dal punto di vista del supporto allo sviluppo di un livello infrastrutturale di sincronizzazione dati, soprattutto in un contesto mobile e distribuito come quello indotto dal caso di studio: esso infatti gestisce praticamente tutte le componenti essenziali per distribuire i dati tra dispositivi sparsi per la rete, permetterne l'utilizzo anche offline e poi sincronizzare i cambiamenti.

6.2 Sperimentazione CouchDB e Couchbase Lite

Dato che le considerazioni emerse dall'esplorazione del panorama tecnologico hanno evidenziato in CouchDB uno strumento idoneo per il caso di studio analizzato in questo lavoro, su di esso è stata effettuata una breve fase di sperimentazione al fine di testarne concretamente le principali potenzialità e quanto realmente possa agevolare lo sviluppo di un sistema per la condivisione delle informazioni a supporto del lavoro cooperativo per il soccorso, in linea con l'obiettivo volto ad uno studio di fattibilità.

A questo proposito è stata realizzata una semplice applicazione per dispositivi Android in grado di sincronizzare documenti (che in linea di principio vorrebbero emulare dei *triage*) tramite un server su cui è installato CouchDB, con l'obiettivo di ottenere una minimale *proof of concept* che dimostri l'effettivo funzionamento e utilità di questo strumento, oltre a verificarne l'eventuale presenza di limiti o difetti che ne pregiudichino l'uso.

L'idea è quella di verificare le principali caratteristiche a cui si è fatto riferimento nel corso della trattazione per il processo di sincronizzazione, in particolare:

- bidirezionalità nello scambio di dati;
- disponibilità del sistema a fronte di mancanza di connessione;
- semplicità di utilizzo delle API per dispositivi mobile;
- possibilità di decidere il sottoinsieme di dati da sincronizzare.

6.2.1 Overview e ambiente operativo

Dal punto di vista strutturale il sistema per la proof of concept è costituito essenzialmente da due diversi tipi di sottosistemi, che ricalcano le considerazioni esposte nel capitolo 4 riguardante l'analisi del problema:

- un server con un'istanza di CouchDB installata per il ruolo di centrale operativa, che agisce da mediatore per la coordinazione e la comunicazione fra gli operatori;
- dispositivi mobile Android nel ruolo di client, da cui generare nuove informazioni (documenti) e accedere a quelle messe a disposizione sul server, in modo da poter verificare che la sincronizzazione avvenga in maniera *bidirezionale*, cioè inviando i propri aggiornamenti e ricevendo quelli effettuati dagli altri di cui il server è a conoscenza.

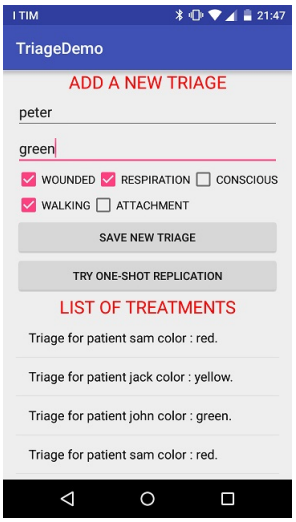

Figura 6.3: Applicazione Android per la sperimentazione.

A differenza dal diagramma per l'interazione descritto in 4.1, utilizzando HTTP come protocollo per le comunicazioni è sempre il client a iniziare le comunicazioni: tale scelta può essere giustificata in questo scenario anche dal fatto che una sua disconnessione è molto probabile, e quindi gli si lascia la responsabilità di decidere quando è più opportuno effettuare la sincronizzazione.

L'ambiente operativo minimale per la proof of concept è quindi costituito da un server e due dispositivi mobile (nello specifico Android, ma potrebbero essere anche di diversa natura, purché supportati da Couchbase Lite oppure adottino un formato dei dati e il protocollo di replica di CouchDB): in questo modo è possibile testare che le modifiche effettuate da un dispositivo siano visibili all'altro dopo essere state rese note al server. Infatti per ora non è prevista una sincronizzazione P2P, ma solo mediata tramite la comunicazione con il server centrale situato nella centrale operativa: per verificare l'effettiva disponibilità delle informazioni lato server è stata molto utile anche la GUI web messa a disposizione da CouchDB (di nome **Futon**) accessibile anche da browser connettendosi all'indirizzo `{server_url}:5984/_utils`, da cui è possibile visualizzare e gestire il contenuto dei database contenuti nell'istanza in esecuzione di CouchDB.

Server Più nel dettaglio, lato server il sottosistema è costituito dalla sola istanza di CouchDB, che offre (di default) il servizio web per accedere alle sue funzionalità tramite la porta `5984`: in questo modo i dispositivi possono comunicare tramite HTTP con il server cononscendone l'URL. Ai fini della

```
manager = new Manager(
    new AndroidContext(getApplicationContext()), Manager.
   DEFAULT_OPTIONS);
database = manager.getDatabase(DATABASE_NAME);
```

Figura 6.4: Accesso al database.

sperimentazione è stato creato e configurato un database (che in CouchDB rappresenta un contenitore per una collezione di documenti) apposito a cui i dispositivi fanno riferimento per la sincronizzazione.

Client La maggior parte delle operazioni viene infatti svolta lato client, per cui è stata predisposta una semplicissima applicazione che permette di effettuare le seguenti operazioni:

- creare un nuovo documento da aggiungere all'istanza locale del database, simulando di fatto un'operazione di triage, il quale deve poi essere inviato alla centrale operativa;
- avere visione dei documenti presenti nell'istanza locale del database, permettendo così di verificare se, dopo la sincronizzazione, sono presenti quelli generati dagli altri operatori;
- attivare il processo di sincronizzazione, in modo da controllare, durante i test, quando questo viene effettuato.

Tutte queste semplici funzionalità sono fornite per mezzo dell'implementazione di una semplice `Activity` che costituisce quindi l'unico entry point dell'applicazione, la UI da cui l'utente può effettuare le operazioni di test (figura 6.3).

6.2.2 Funzionalità testate

Di seguito vengono descritte le maggiori funzionalità testate, riportando gli stralci di codice salienti dell'applicazione Android utilizzata per dare un'idea della semplicità con cui possono essere svolte le operazioni grazie all'utilizzo della libreria Couchbase: per una descrizione dei concetti principali esposti si rimanda alle sezioni 5.2.5 e 5.2.6, in cui ne viene data una sintetica ma essenziale overview.

Ovviamente, per usufruire delle funzionalità è necessario avere accesso al database sul dispositivo, che viene permesso grazie all'entità `Manager` (figura 6.4).

```
//create the JSON object as a map of properties
Map<String, Object> properties =
    new HashMap<String, Object>();
properties.put("_id", id);
properties.put("type","triage");
properties.put("patient","john");
properties.put("color","red");
...
//get a new document
Document document = database.createDocument();
//save to the db
document.putProperties(properties);
```

Figura 6.5: Creazione e salvataggio di un documento.

Creazione e salvataggio dati

La memorizzazione delle informazioni su un database locale al dispositivo è necessaria per offrirne un accesso trasparente alle condizioni della rete e disaccoppiare l'accesso ai dati dalla presenza della connessione: il database locale può essere concettualmente visto come una *replica* parziale di quello presente sul server, con il quale viene sincronizzato nei momenti in cui la connessione è disponibile.

Per rendere nota quindi al server la presenza di nuovi documenti basta memorizzarli in locale, in quanto in seguito è compito del processo di sincronizzazione trasferirli in modo che i cambiamenti siano ripercossi anche nella centrale operativa.

L'operazione di creazione e salvataggio di un nuovo documento locale risulta essere molto semplice grazie al suo formato JSON, compatibile con il modello dei dati utilizzato lato server da CouchDB: grazie alla libreria Couchbase Lite, basta associare al documento un insieme di proprietà con le quali si rappresentano i campi del documento JSON, ognuno identificato da un nome, come si può notare dallo stralcio in figura 6.5

Da notare il fatto che la serializzazione dell'oggetto in un documento JSON viene gestita dalla libreria, sollevando il programmatore anche da questo aspetto: inoltre particolare rilevanza assume il campo `_id`, che identifica univocamente il documento, mentre l'assenza del campo `_rev` fa capire che questo viene gestito internamente dal framework, in accordo con il meccanismo di versioning.

```
//pull & push to have bidirectional sync
Replication pullReplication =
    database.createPullReplication(syncUrl);
Replication pushReplication =
    database.createPushReplication(syncUrl);
//start the sync processes
pullReplication.start();
pushReplication.start();
//attach listener to monitor the processes
pullReplication.addChangeListener(this);
pushReplication.addChangeListener(this);
```

Figura 6.6: Creazione e attivazione del processo di sincronizzazione.

Sincronizzazione e filtering

Il processo di sincronizzazione con l'istanza del database mantenuta nel server, identificata dall'URL a cui può essere acceduta, deve essere bidirezionale e per questo è necessario attivarne due, uno per ogni direzione: in Couchbase entrambi sono visti come oggetti di classe `Replication`, il cui verso viene deciso all'atto della creazione, che avviene chiamando il relativo metodo sull'oggetto database. Per replica *pull* si intende il processo atto a trasferire i cambiamenti avvenuti sul server nella copia locale, mentre per il processo inverso si utilizza il termine *push*, come si può notare dallo stralcio in figura 6.6: il processo di replica può essere monitorato tramite l'utilizzo di un *listener* in cui specificare tramite una callback il comportamento da tenere rispetto ai suoi cambiamenti di stato. Si può anche notare come la libreria *nasconda* la complessità di questo processo, che coinvolge richieste HTTP al server e la gestione delle relative risposte: in particolare viene implementato il protocollo definito da CouchDB che, grazie al sistema di versioning e all'utilizzo di un *change feed*, riesce a discriminare quali siano i giusti cambiamenti da apportare.

Inoltre va ricordato che i processi avviati in questo modo sono di tipo *one-shot* nel senso che terminano una volta sincronizzati tutti i cambiamenti disponibili fino a quel momento, ma è possibile renderli continui, lasciandoli così in attesa di nuovi cambiamenti, invocando su di un oggetto di tipo `Replication` il metodo `setContinuous(true)`.

Infine, entrambi i processi possono essere filtrati per decidere quali dati effettivamente considerare e quali no tramite funzioni di filtro che vengono eseguite su ciascun documento del database per determinare se soddisfano o meno la condizione per essere replicati. Per una replica di tipo push è possibile definire una funzione di filtro in linguaggio nativo (che ad esempio operi

```
//create a local filter for push replication
database.setFilter(pushFilter, new ReplicationFilter() {
  @Override
  public boolean filter(SavedRevision revision,
      Map<String, Object> params) {
    String nameParam = (String) params.get("name");
    //sync only docs with 'name' different from param
    return nameParam != null &&
        !nameParam.equals(revision.getProperty("patient"));
  }
});
//attach the filter to a push replication process
pushReplication.setFilter(pushFilter);
//create and pass the params
Map<String, Object> pushParams =
    new HashMap<String, Object>();
pushParams.put("name", "sam");
pushReplication.setFilterParams(pushParams);
```

Figura 6.7: Esempio di filtraggio parametrico del processo di sincronizzazione a seconda del parametro *name* passato.

sui valori dei campi del documento), registrarla nel database ed in seguito associarla al processo di replica tramite il metodo `setFilter(pushFilter)` a cui passare la stringa identificativa della funzione definita nella registrazione al database (figura 6.7); per una replica di tipo pull invece bisogna fare riferimento a una funzione di filtro definita lato server che, nel caso di CouchDB, è salvata in uno specifico documento di design, il cui nome va passato, insieme al nome della funzione in esso contenuta da utilizzare, alla funzione `setFilter` nella forma `docName/filterFunName`.

Ai filtri può inoltre essere aggiunta dinamicità rendendoli parametrici: le funzioni di filtro infatti possono fare riferimento a dei parametri che poi vengono passati a *runtime* all'attivazione del processo del processo di replica: in questo modo utilizzando uno stesso filtro si possono ottenere dei comportamenti differenti ad ogni sincronizzazione, ad esempio scartando documenti relativi ogni volta ad un paziente diverso.

View e Query

Sui dati possono essere effettuate delle interrogazioni sia lato client che lato server attraverso i concetti di **query** e **view** e nello specifico sono state testate quelle lato client per poter mostrare sul dispositivo la lista di documenti presenti nel database locale.

```
//obtain a view from a db
View triageView = database.getView("triages");
//set the map function
triageView.setMap(new Mapper() {
  @Override
  public void map(Map<String, Object> document,
      Emitter emitter) {
    if (document.get("type").equals("triage")) {
      Object timeAtt = document.get("created_at");
      if (timeAtt != null) {
        String time = timeAtt.toString();
        if (time != null) emitter.emit(time, null);
      }
    }
  }
}, "1.1");
```

Figura 6.8: Esempio di view che contiene i documenti di tipo *triage* indicizzati rispetto al momento della loro creazione.

A tal proposito è necessario definire un oggetto di tipo `View` che rappresenta un mapping chiave-valore eseguito per ciascun documento presente e materializzato in maniera persistente, ad esempio per indicizzare i documenti presenti secondo la data della loro creazione o un qualsiasi altro loro attributo: esso va associato al relativo database tramite il metodo `getView(viewName)` il quale restituisce una nuova vista della quale bisogna poi definire la funzione di map tramite il metodo `setMap()`, a cui passare un oggetto `Mapper` che incapsula questa funzione, al cui interno si può fare uso dei campi contenuti nel documento per emettere le coppie key-value da indicizzare (figura 6.8).

Bisogna inoltre ricordare che ad una view può essere associata anche una funzione di reduce, che nei test svolti non è stata utilizzata, ma che permette di calcolare risultati aggregati a partire dai valori contenuti in ogni documento.

Sulle viste definite è poi possibile eseguire delle `Query`, interrogazioni con cui raccogliere i risultati in un certo ordine o in un certo range di valori: solitamente oggetti di questo tipo vengono recuperati a partire dalla relativa `View` tramite il metodo `createQuery()` e vengono eseguiti tramite il metodo `run()`, ma è presente anche uno speciale tipo di interrogazione che restituisce tutti i documenti presenti nel database e che si può recuperare invocando `createAllDocumentsQuery()` sul relativo database. Di grande utilità è la possibilità di trasformare la query in una `LiveQuery` (figura 6.9), una sorta di interrogazione attiva che monitora i cambiamenti sull'insieme dei dati definito dalla vista e li notifica tramite la callback del listener associato, permettendo

```
//create an all-doc query
Query query = database.createAllDocumentsQuery();
query.setAllDocsMode(Query.AllDocsMode.ALL_DOCS);
//transform it to a live query
LiveQuery liveQuery = query.toLiveQuery();
//attach a listener to manage the change
liveQuery.addChangeListener(new LiveQuery.ChangeListener(){
  public void changed(final LiveQuery.ChangeEvent event) {
    runOnUiThread(new Runnable() {
      public void run() {
        triageArrayAdapter.clear();
        Iterator<QueryRow> it=event.getRows();
        for (; it.hasNext();){
          triageArrayAdapter.add(it.next());
        }
        triageArrayAdapter.notifyDataSetChanged();
      }
    });
  }
});
liveQuery.start();
```

Figura 6.9: Esempio di live query per tutti i documenti

così di applicare il pattern *Observer*: quest'oggetto è stato utilizzato nei test effettuati per aggiornare la lista dei documenti visibile all'utente non appena questi venissero aggiunti nel database, risparmiando quindi la scrittura di molto codice che sarebbe stato necessario altrimenti.

6.2.3 Considerazioni sulla sperimentazione

La breve sperimentazione effettuata ha dato esiti positivi nei riguardi della **fattibilità**, nel senso che sia CouchDB che la libreria Couchbase per il supporto a dispositivi mobile sono risultati strumenti il cui utilizzo semplifica notevolmente lo sviluppo di un'applicazione che necessita di sincronizzazione dati: essi forniscono un alto livello di astrazione e delle API che permettono di risparmiare il tempo di implementazione di molte funzionalità correlate a tale e complessa problematica.

Inoltre risalta subito all'occhio la **trasparenza** con cui è possibile accedere ai dati, infatti anche provando a disconnettere i device all'utente rimane possibile usufruire del servizio offerto: nel caso del soccorso in situazioni di emergenza questo risulta di fondamentale importanza in quanto il supporto fornito non deve mai mancare all'operatore. Qualora non vi sia connessione di rete infatti le informazioni generate vengono semplicemente salvate trami-

te database nella memoria del dispositivo in attesa poi di essere inviate alla centrale operativa, e all'operatore è possibile disporre di alcuni dati da essa provenienti grazie alla loro memorizzazione come copia locale.

Inoltre anche la possibilità di scelta dei dati da sincronizzare è risultata essere abbastanza flessibile e di utilità: poter passare dei parametri alla funzione di filtro responsabile di decidere se un certo documento deve essere sincronizzato costituisce una buona possibilità per ottenere un fine controllo su di essi.

Anche la presenza di una modalità *continua* di sincronizzazione costituisce una potenziale interessante alternativa, soprattutto nel caso si necessitasse di ricevere nuove informazioni nel più breve tempo possibile: per il caso considerato sembra però più opportuno realizzare un componente che scheduli periodicamente i task di sincronizzazione, anche a seconda delle possibilità di connettività, fornendo comunque trasparenza all'utente che non deve così esplicitamente richiederla.

Lato server l'effettiva presenza dei dati è stata verificata grazie alla GUI di gestione offerta nativamente da CouchDB, ma in linea di principio è possibile effettuare query sul database tramite un qualsiasi client web che faccia le giuste richieste HTTP, in tal senso per una rapida verifica è bastato interrogare il database tramite il tool a linea di comando **cURL**[1].

La sperimentazione effettuata non aveva l'obiettivo di testare tutte le varie funzionalità degli strumenti ed infatti sono stati tralasciati in particolare gli aspetti di *sicurezza* relativi alla *user authentication e authorization*: in questo senso sia CouchDB che Couchbase Lite offrono qualche funzionalità, che però a prima vista sembrano solo basilari; soprattutto dal punto di vista dell'autorizzazione degli utenti CouchDB non sembra dotato di un proprio robusto meccanismo, ma solamente dei concetti di *admin* e *member*, con la definizione dei ruoli e la gestione delle autorizzazioni che deve quindi essere demandata al livello applicativo.

Il modello dei dati utilizzato dagli strumenti, in cui il concetto di *documento* rappresenta l'unità atomica sul quale le operazioni hanno garanzia di consistenza, vincola a modellare le relazioni insite nel modello dei dati spezzandoli in differenti documenti e collegandoli tramite il loro identificatore: questa "perdita" di strutturazione dei dati, le cui relazioni sono "spezzate" per i vari documenti salvati nel database causa una maggiore complessità nell'effettuare query complesse.

Inoltre, tra le principali limitazioni riscontrate bisogna citare la difficoltà di manutenibilità e debugging dei filtri lato server, in quanto essi sono rappresentati da funzioni Javascript memorizzate in particolari documenti di

[1] `https://curl.haxx.se/`

design, e per ritrovare la causa di un malfunzionamento, anche se banale, è necessario visionare i file di log di *CouchDB*.

Infine è necessario citare anche che è stata riscontrata un'incompatibilità nell'utilizzo dei due strumenti riguardo la gestione di *attachment* (e.g. immagini) come proprietà del documento: infatti la sincronizzazione di documenti che hanno al loro interno tale proprietà non avviene a causa, da quello che si è potuto evincere dai log sia lato client che lato server, di una piccola incompatibilità tra Couchbase Lite e CouchDB nella loro serializzazione JSON. Bisogna comunque ricordare che, probabilmente, la soluzione migliore sarebbe inviare file multimediali tramite un apposito canale, in quanto per essere inviati insieme agli altri dati dei documenti devono essere codificati in *base64*, con un overhead di circa il 33% in termini di spazio, ed inoltre ciò permetterebbe di dare diverse priorità all'invio dei dati rispetto a quello di elementi come immagini.

Conclusioni

L'obiettivo di questa tesi era quello di studiare e approfondire il tema della **sincronizzazione dati** in relazione a un caso di studio, in particolare inserito nel contesto di sistemi informatici a supporto del lavoro cooperativo per lo svolgimento di operazioni di soccorso a seguito di situazioni di emergenza: questa ampia e complessa tematica è stata quindi calata nei determinati requisiti e vincoli indotti dal caso applicativo, in maniera tale da focalizzare l'esplorazione verso uno studio di fattibilità per una futura realizzazione.

Preso dal più ampio ambito di un progetto svolto durante gli anni scorsi dal dipartimento di informatica della facoltà, il caso applicativo ha giocato un ruolo centrale in questo lavoro: al suo interno è stata isolata infatti la necessità di una funzionalità di sincronizzazione dati tra gli operatori attivamente coinvolti in una missione, la cui analisi, volta a comprendere la presenza nel **panorama tecnologico** di eventuali strumenti o particolari approcci per la risoluzione della problematica, ha portato a uno studio e un'esplorazione dai confini piuttosto ampi.

L'approfondimento delle tematiche di riferimento e dei sistemi che, in vari modi hanno cercato di affrontare il problema del supporto informatico in situazioni di emergenza ha permesso di estrapolare gli elementi principali, i concetti comuni e gli approcci di successo utili a una maggiore comprensione del tema, anche se la quasi totalità di essi sono stati realizzati in forma perlopiù solo prototipale e senza trattare direttamente la sincronizzazione dati.

Esso infatti è risultato importante per l'analisi del caso di studio, da cui è emersa la necessità di un **livello infrastrutturale** che supporti la sincronizzazione dati in uno scenario **remote mobile**, caratterizzato da frequenti disconnessioni: proprio la necessità di poter comunque accedere ai dati anche in assenza di connessione risulta un requisito molto importante, in quanto aumenta la complessità del problema da affrontare, già di suo non banale. É in queste considerazioni che l'esplorazione del panorama tecnologico condotta come fase centrale di questo lavoro trova la sua base di appoggio: a causa delle caratteristiche del problema, soprattutto influenzate dalla possibilità

che l'utente non abbia accesso alla rete proprio quando necessita delle informazioni, fra quelle descritte l'area dei database distribuiti è stata reputata come la più idonea, nella quale in particolare ci si è concentrati su CouchDB per le motivazioni espresse in 6.1.4.

Poter usufruire dei suoi meccanismi di replica, versioning e sincronizzazione dà allo sviluppatore la possibilità di investire molto più tempo nella realizzazione della logica applicativa, rendendo lo sviluppo molto più agile: gestito tramite chiamate HTTP implementate lato device mobile da API di una libreria compatibile con CouchDB, il processo di sincronizzazione permette di agire sui dati localmente, e quindi anche offline, per poi rendere noti i cambiamenti quando un livello di connettività minimo viene riottenuto. Queste caratteristiche sono state confermate da una breve fase di sperimentazione non solo atta alla conferma della possibilità di un suo utilizzo in un'eventuale realizzazione, ma anche a testarne i limiti e la compatibilità con la libreria Couchbase per lo sviluppo mobile.

In questo senso, l'approfondimento condotto ha permesso di verificare la fattibilità della realizzazione di un componente che gestisca la sincronizzazione dati per un'applicazione a supporto del soccorso ai feriti in uno scenario *remote mobile*, una sorta di middleware che sgravi il livello applicativo da questo tipo di operazioni: utilizzando infatti CouchDB e Couchbase Lite, come descritto in 6.1.4, si hanno gli strumenti sia per lo sviluppo lato server che lato dispositivo mobile, fornendo così un ambiente completo.

Inoltre l'aver analizzato anche altri tipi di approcci ha permesso di ottenere una più ampia consapevolezza della tematica, soprattutto in tema di una futura estensione dei requisiti: conoscere infatti le caratteristiche principali e le funzionalità offerte anche da altri strumenti permette di avere una visione più completa del quadro in cui l'applicazione particolare si colloca e il possibile ruolo che ognuno di essi potrebbe ricoprire al suo interno.

Data l'ampiezza delle problematiche introdotte dal caso di studio, alcune di esse non sono state considerate in maniera approfondita in questo lavoro in quanto esulavano dall'obiettivo di analizzare la funzionalità di sincronizzazione e condivisione delle informazioni: si è però consci di quanto esse possano essere rilevanti nel contesto di applicazioni pervasive, soprattutto in scenari di comunicazione remota e mobile; tra di esse una delle più rilevanti è il possibile utilizzo di diversi canali di comunicazione a seconda delle caratteristiche del campo d'azione e del suo livello di connettività (e.g. Internet, locale, comunicazioni ad hoc), che possono variare anche a seconda del dominio applicativo, come in scenari militari.

Sebbene ci si sia soffermati su una specifica funzionalità del sistema l'approfondimento svolto ha avuto dei confini piuttosto ampi, anche a causa della complessità del tema trattato: la sincronizzazione dati è infatti una proble-

matica nota e affrontata in vari contesti, ma che ancora non ha trovato una soluzione completa, anche se esistono, come descritto nei precedenti capitoli, differenti approcci per fronteggiarla a seconda delle particolari caratteristiche del problema in cui è inserita.

In conclusione è possibile notare che una sincronizzazione che abiliti una maggiore condivisione di informazioni in ambito lavorativo ha una rilevanza molto più generale di quella mostrata nel particolare caso applicativo analizzato relativo al soccorso in emergenza: essa infatti risulta essere utile in praticamente ogni contesto di supporto al lavoro dei professionisti che devono eseguire azioni all'interno di un gruppo, necessitando quindi di meccanismi di coordinazione che ne aumentino la sinergia e la collaborazione. Da questo punto di vista è indubbio infatti che conoscere le informazioni generate dai propri collaboratori aumenti la consapevolezza dell'individuo aiutandolo a prendere decisioni migliori e più efficaci nei riguardi del proprio corso d'azione: come emerso dalla trattazione, sebbene sia possibile implementare questi meccanismi in differenti modi, strumenti quali **CouchDB** possono essere di grande aiuto nel loro sviluppo, candidandosi quindi a ritagliarsi un ruolo importante nello sviluppo di applicazione soprattutto nel campo del Mobile Computing, dove permettere all'utente di usufruire dei servizi delle *app* anche offline sta diventando una delle funzionalità più richieste e che più fanno la differenza per il loro successo e la loro diffusione.

6.2.4 Sviluppi futuri

A partire dall'analisi e dall'approfondimento svolto in questo lavoro diversi possono essere gli **sviluppi futuri** che è possibile portare avanti, soprattutto da un punto di vista progettuale e sperimentale nei riguardi non solo del caso di studio considerato, ma anche dei tanti scenari in cui una buona condivisione dei dati può essere di utilità.

Dall'analisi svolta è infatti emersa la fattibilità della realizzazione di un componente per la sincronizzazione agevole dei dati tramite l'utilizzo di uno strumento come CouchDB: è quindi possibile pensare a una conseguente fase di progettazione e sviluppo che ne sfrutti le caratteristiche per integrare questa funzionalità in un prototipo per il caso di studio considerato, dimostrando così sia l'effettiva utilità dal punto di vista dello sviluppo che i benefici tratti dagli utenti dell'applicazione a seguito dell'introduzione della nuova *feature*.

Inoltre, sempre a livello progettuale, si potrebbe prendere spunto dall'approccio utilizzato da CouchDB per la sincronizzazione ed eventualmente realizzarne ed implementarne una versione specifica per le esigenze del caso considerato, qualora ce ne fosse la necessità, ad esempio servendosi di un framework per lo sviluppo di servizi RESTful come *Restlet* con l'integrazione di

un differente livello di persistenza: grazie alla sua flessibilità allo sviluppatore sarebbe infatti possibile sviluppare un componente secondo le sue esatte necessità utilizzando un approccio che si è dimostrato di successo nel contesto di sistemi distribuiti.

Infine, da un punto di vista più sperimentale potrebbe rivelarsi interessante testare più a fondo *CouchDB* per verificare quanto effettivamente sia pronto per essere utilizzato in un contesto reale di produzione, ma anche il database *Couchbase Server*, che tramite l'utilizzo di un componente chiamato *Sync Gateway* offre delle funzionalità di replica e sincronizzazione molto simili a quelle di CouchDB, da cui è ispirato, in modo da comprendere se effettivamente può costituire una valida alternativa.

Ringraziamenti

Arrivato alla fine del mio percorso universitario, che mi ha sì richiesto impegno e sacrificio ma che è stato davvero stimolante e denso di soddisfazioni, mi sento di dover ringraziare coloro i quali durante questo cammino mi hanno supportato e sostenuto.

I primi a dover essere ringraziati sono di certo i membri della mia famiglia, che non mi hanno mai fatto mancare il loro sostegno: i miei genitori, Ombretta e Luciano, senza la cui fiducia, approvazione e il cui supporto, comprensivo dei sacrifici anche economici che hanno fatto, non avrei potuto intraprendere gli studi e raggiungere questo obiettivo, e mio fratello Matteo, mai mancato nel momento del bisogno e che ha sempre costituito per me un esempio. Ci tengo a ringraziare inoltre anche i miei nonni, Giovanni e Floriana, per essere stati sempre presenti per me ed avermi sempre sostenuto.

Vorrei ringraziare sinceramente anche la mia ragazza, Licia, che mi ha sempre aiutato e sostenuto con la sua insostituibile presenza, soprattutto nei momenti di difficoltà, credendo fermamente in me e sopportando pazientemente la mia assenza nei momenti di maggior impegno.

É anche e soprattutto merito vostro se sono riuscito a raggiungere questo importante traguardo, che segna un punto di svolta per la mia vita.

Grazie inoltre anche agli amici con cui ho condiviso il tempo al di fuori dell'università, in particolare a Filippo, che ormai posso quasi considerare un vero e proprio fratello, e ai ragazzi della Soglianese, con cui ho condiviso gioie e dolori sui campi da calcio.

Dei compagni universitari desidero ringraziare in particolare Simone, con cui ho condiviso praticamente tutto il percorso svolto, e Michele, compagno di diversi progetti, che hanno reso ancora più interessante la mia esperienza in Università.

Infine un sentito ringraziamento va al Prof. Mirko Viroli che, con grande pazienza e disponibilità, mi ha seguito e supportato durante l'attività di tesi e ad Angelo Croatti, che mi ha dimostrato altrettanta disponibilità e cortesia nella collaborazione che la redazione di questo elaborato ha richiesto.

Bibliografia

[1] Clarence A Ellis, Simon J Gibbs, and Gail Rein. *Groupware: some issues and experiences.* Communications of the ACM, 1991.

[2] Paul Wilson. *Computer supported cooperative work: An introduction.* Springer Science & Business Media, 1991.

[3] Kjeld Schmidt and Liam Bannon. *Taking CSCW seriously.* Computer Supported Cooperative Work (CSCW), 1992.

[4] David Pinelle. *Improving groupware design for loosely coupled groups.* PhD thesis, Citeseer, 2004.

[5] Hans Oh, Carlos Rizo, Murray Enkin, and Alejandro Jadad. *What is eHealth?: a systematic review of published definitions.* World Hosp Health Serv, 2005.

[6] E Kyriacou, MS Pattichis, CS Pattichis, A Panayides, and Andreas Pitsillides. *m-Health e-Emergency systems: current status and future directions.* Antennas and Propagation Magazine, IEEE, 2007.

[7] Simone Liu and Thomas Jepsen. *Mobile Health.* IEEE Computer Society, 2012.

[8] EC Kyriacou, CS Pattichis, and MS Pattichis. *An overview of recent health care support systems for eEmergency and mHealth applications.* In *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE.* IEEE, 2009.

[9] Martina Ziefle and Carsten Röcker. *Acceptance of pervasive healthcare systems: A comparison of different implementation concepts.* PervasiveHealth, 2010.

[10] Mark Weiser. *The computer for the 21st century.* Scientific american, 1991.

[11] Mahadev Satyanarayanan. *Pervasive computing: Vision and challenges.* Personal Communications, IEEE, 2001.

[12] Debashis Saha and Amitava Mukherjee. *Pervasive computing: a paradigm for the 21st century.* Computer, 2003.

[13] Jefferson Heard, Sidharth Thakur, Jessica Losego, and Ken Galluppi. *Big board: Teleconferencing over maps for shared situational awareness.* Computer Supported Cooperative Work (CSCW), 2014.

[14] Pietro Brunetti, Angelo Croatti, Alessandro Ricci, and Mirko Viroli. *Smart Augmented Fields for Emergency Operations.* Procedia Computer Science, 2015.

[15] Francesco De Mola, Giacomo Cabri, Nicola Muratori, Raffaele Quitadamo, and Franco Zambonelli. *The UbiMedic Framework to Support Medical Emergencies by Ubiquitous Computing.* ITSSA, 2006.

[16] Massimo Mecella, Michele Angelaccio, Alenka Krek, Tiziana Catarci, Berta Buttarazzi, and Schahram Dustdar. *Workpad: an adaptive peer-to-peer software infrastructure for supporting collaborative work of human operators in emergency/disaster scenarios.* In *Collaborative Technologies and Systems, 2006. CTS 2006. International Symposium on.* IEEE, 2006.

[17] Claus M Christensen, Jesper Kjeldskov, and Klaus K Rasmussen. *Geo-Health: a location-based service for nomadic home healthcare workers.* In *Proceedings of the 19th Australasian conference on Computer-Human Interaction: Entertaining User Interfaces.* ACM, 2007.

[18] Estanislao Mercadal, Sergi Robles, Ramon Martí, Cormac J Sreenan, and Joan Borrell. *Double multiagent architecture for dynamic triage of victims in emergency scenarios.* Progress in Artificial Intelligence, 2012.

[19] Tia Gao, Dan Greenspan, Matt Welsh, Radford R Juang, and Alex Alm. *Vital signs monitoring and patient tracking over a wireless network.* In *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the.* IEEE, 2006.

[20] Thomas J Cova. *GIS in emergency management.* Geographical information systems, 1999.

[21] Pietro Brunetti. *Eco-sistemi informatici distribuiti real-time a supporto edl lavoro cooperativo in scenari di emergenza: studio e realizzazione di un caso applicativo.* Master's thesis, Università di Bologna, 2014.

[22] Angelo Croatti. *Sistemi informatici mobili e realtà aumentata a supporto del lavoro cooperativo in scenari d'emergenza: studio e realizzazione di un caso applicativo*. Master's thesis, Università di Bologna, 2014.

[23] Alexander Stage. *Synchronization and replication in the context of mobile applications*. May, 2005.

[24] Jan Sedivy, Tomas Barina, Ion Morozan, and Andreea Sandu. *MCSync-Distributed, Decentralized Database for Mobile Devices*. In *Cloud Computing in Emerging Markets (CCEM), 2012 IEEE International Conference on*. IEEE, 2012.

[25] JiuLing Feng, Xiuquan Qiao, and Yong Li. *The research of synchronization and consistency of data in mobile environment*. In *Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on*. IEEE, 2012.

[26] Neil Fraser. *Differential Synchronization*. In *Proceedings of the 2009 ACM Symposium on Document Engineering*, 2009.

[27] M Tamer Özsu and Patrick Valduriez. *Principles of distributed database systems*. Springer Science & Business Media, 2011.

[28] J Chris Anderson, Jan Lehnardt, and Noah Slater. *CouchDB: the definitive guide*. O'Reilly Media, Inc., 2010.

[29] *Documentazione online CouchDB*. `http://docs.couchdb.org/en/1.6.1/contents.html`. consultato il 28/02/2016.

[30] *Documentazione online Couchbase Lite*. `http://developer.couchbase.com/documentation/mobile/1.2/get-started/couchbase-mobile-overview/index.html`. consultato il 29/02/2016.

[31] Edward Curry. *Message-Oriented Middleware*. In *Middleware for communications*. John Wiley & Sons, Ltd, 2004.

[32] *Documentazione online RabbitMQ*. `http://www.rabbitmq.com/getstarted.html`. consultato il 19/02/2016.

[33] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[34] Roy T Fielding and Richard N Taylor. *Principled design of the modern Web architecture.* ACM Transactions on Internet Technology (TOIT), 2002.

[35] Jim Webber, Savas Parastatidis, and Ian Robinson. *REST in practice: Hypermedia and systems architecture.* O'Reilly Media, Inc., 2010.

[36] Jerome Louvel, Thierry Templier, and Thierry Boileau. *Restlet in Action: Developing RESTful Web APIs in Java.* Manning Publications Co., 2012.