

ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

Seconda Facoltà di Ingegneria con sede a Cesena

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Tesi di Laurea in Sistemi di Elaborazione dell'Informazione

Trasformazione di un ambiente fisico in Cyber-Physical Ecosystem

Candidato:

Patrizia De Luca

Relatore:

Prof. Tullio Salmon Cinotti

Correlatori:

Ing. Fabio Vergari

Ing. Alfredo D'Elia

Ing. Francesco Morandi

Indice

Introduzione	4
1 Il Web Semantico	9
1.0.1 Web Semantico come scommessa del futuro della gestione dell'informazione	11
1.0.2 Il goal del Web Semantico	13
1.0.3 Architettura del Web Semantico	15
2 Elementi di architettura del Web Semantico come discussi nel progetto RECOCAPE	23
2.1 Il progetto RECOCAPE nella Commissione Europea Ricerca e Innovazione	24
2.2 RECOCAPE Work Package 3 - Il Corso	29
2.3 Il Corso - Semantic Web Introduction	35
2.4 Il Corso - Smart Spaces e Smart M3	35
2.5 Il Corso - Sensor Data Ontology	35
3 Cyber Physical Systems e Smart Environments	37
3.1 Il Web Semantico per la Telemedicina	37
3.1.1 Telemedicina come Tele-assistenza: Health Smart Homes	38
3.2 I Sistemi Cyber-Fisici	40
3.2.1 Integrazione del Web Semantico nei Sistemi Cyber-Fisici	43
3.3 Smart Spaces	44
3.3.1 Smart M3	46

4	Arces Smart Space - CHIRON	55
4.1	Open CHIRON Smart Space Ontology	59
4.2	Open CHIRON Smart Space - ambiente fisico	62
4.2.1	Architettura Logica	63
4.2.2	Architettura Fisica	66
4.3	Progetto del cyber-physical ecosystem	68
5	Librerie Java KPI con supporto a SPARQL	71
5.1	Java KPICore	72
5.2	SPARQL	77
5.2.1	SPARQL Path Expressions	79
5.2.2	Output di una query SPARQL	83
5.3	Implementazione del supporto SPARQL per le librerie Java KPI	87
5.3.1	Struttura dati per la memorizzazione dei risultati della query SPARQL	87
5.3.2	Metodi di supporto alla manipolazione dei risultati . .	93
5.3.3	Metodi di abilitazione alle query SPARQL	95
5.3.4	Aggiornamento GUI	96
5.4	Risultati e sviluppi futuri	96
6	Il sistema Android	99
6.1	La piattaforma Android	100
6.2	Origini di Android	101
6.3	Architettura Software	104
6.3.1	Il Kernel Linux	106
6.3.2	Librerie Native	107
6.3.3	Sviluppo ed esecuzione delle applicazioni	109
6.3.4	Application Framework	112
6.4	Le Applicazioni	114
6.4.1	Ciclo di vita dei componenti	117
6.5	Ambiente di Sviluppo	123
6.5.1	Eclipse ADT Plugin	124

6.5.2	Il file Manifest	126
6.5.3	Debug e logging di un'applicazione Android	126
7	Heart Life Doctor Android Application	129
7.1	Specifiche Tecniche	130
7.2	Funzionalità	131
7.2.1	Il tab <i>Remote</i>	132
7.2.2	Il tab <i>Patient</i>	134
7.2.3	Il tab <i>Environment</i>	136
7.3	Architettura	137
7.3.1	Gestione delle Sottoscrizioni e Update dell'interfaccia .	139
7.4	Robustezza	146
7.4.1	Connessione all'avvio	147
7.4.2	Controllo attivo di connessione	148
7.5	Prossime Release	153
	Conclusioni	154
	Bibliografia	157

Introduzione

Obiettivo di questo lavoro di tesi è il perfezionamento di un sistema di *Health Smart Home*, ovvero un ambiente fisico (ad esempio un'abitazione) che incorpora una rete di comunicazione in grado di connettere apparecchi elettronici e servizi controllabili da remoto, con l'obiettivo di facilitare la vita ad anziani, malati o disabili nelle loro case.

Questo lavoro di tesi mostrerà come è stato possibile realizzare tale sistema partendo dalle teorie e dalle tecnologie sviluppate per il Web Semantico, al fine di trasformare l'ambiente fisico in un Cyber Physical (Eco)System perfettamente funzionante.

I Cyber-physical systems (CPS) o Sistemi cyber-fisici, sono sistemi fisici e artificiali la cui attività è monitorata, coordinata, controllata e integrata da un core di calcolo e comunicazione. Proprio come Internet ha cambiato il modo in cui gli esseri umani interagiscono tra loro, i sistemi cyber-fisici trasformano il modo in cui interagiamo con il mondo fisico intorno a noi.

Come vedremo in questo lavoro di tesi, lo sviluppo di un sistema cyber fisico richiede competenze informatiche (ma anche elettroniche e di telecomunicazione) eterogenee quali la conoscenza di algoritmi, reti, sviluppo di sistemi distribuiti, sviluppo di architetture software e hardware, programmazione embedded, elaborazione ed analisi dei dati, intelligenza artificiale, robotica, e di altrettanta importanza è possedere un approccio multidisciplinare ai problemi.

Mostreremo anche come i sistemi CPS porteranno, già nel breve termine, una potenziale rivoluzione in ogni settore, dal settore automobilistico al controllo

di processo, al risparmio energetico, ai sistemi di difesa e alla domotica, e in questo lavoro di tesi ci concentreremo sull'aspetto di telemedicina.

In particolare vedremo come un CPS possa ridurre il tempo che il personale medico deve spendere per il paziente garantendo al contempo un alto livello di qualità.

Il Capitolo 1 tratta il Web Semantico nella sua interezza e le motivazioni per cui esso è la scommessa del futuro della gestione dell'informazione. Vengono mostrati i principali elementi architetturali e tecnologici del Web Semantico e il sistema di rappresentazione della conoscenza. Tramite tale sistema di rappresentazione si pongono le basi per costruire il Cyber Physical System obiettivo del lavoro di tesi.

Il Capitolo 2 approfondisce alcuni aspetti del lavoro di studio operato sul Web Semantico. La prima parte di questo lavoro di tesi è infatti lo sviluppo di un corpo di lezioni interamente in lingua inglese per uno dei corsi previsti dal progetto RECOCAPE. Il progetto RECOCAPE è la risposta alla Call FP7 dell'Unione Europea, settimo programma quadro per la ricerca e lo sviluppo tecnologico promosso dalla Commissione Europea Ricerca e Innovazione. Obiettivo del corso è fornire un training di base sulle tecnologie di sviluppo per il Web Semantico, della durata complessiva di un mese e un impegno previsto di 25 giorni lavorativi. Il corso in oggetto si terrà ad Ottobre 2012 presso Smart Village Egypt a Il Cairo, Egitto.

Il Capitolo 3 tratta in maniera estensiva i Cyber Physical Systems, Smart Environments e Smart Spaces, e presenta la piattaforma Smart M3 sviluppata all'interno del progetto europeo SOFIA (Smart Object for Intelligent Applications)¹.

Nel Capitolo 4 viene mostrato come lo scenario proposto è realmente realizzabile nel laboratorio Arces MARS in cui ho svolto la mia attività di tesi. La piattaforma hardware e software di Health Smart Home messa a punto all'interno del laboratorio è inserita all'interno di CHIRON, progetto di ricerca

¹www.sofia-project.eu

Europeo finanziato dall’iniziativa Europea ARTEMIS e dagli Stati aderenti nell’ambito del bando ARTEMIS 2009 – Sottoprogramma ASP2: “Gestione della salute centrata sulla persona”².

Vedremo come *Open Chiron Smart Space* realizza un sistema di person-centric health management in grado di seguire il paziente in tutte le sue cure e di agevolare il personale medico nella presa di decisioni.

Verrà quindi mostrata l’ontologia di Open Chiron Smart Space e lo sviluppo di una serie di Knowledge Processors per ottenere il comportamento desiderato.

Il Capitolo 5 mostra il lavoro di supporto ed estensione al sistema Open Chiron Smart Space. La libreria Java KPICore è interamente sviluppata in linguaggio Java e offre tutte le primitive e i metodi di accesso, manipolazione, visualizzazione, interrogazione e inserimento di dati all’interno del sistema Open Chiron Smart Space.

Il mio lavoro di supporto è l’estensione delle librerie KPICore con l’inserimento del supporto alle query di tipo SPARQL. La possibilità di effettuare query di tipo SPARQL, come vedremo, migliora le prestazioni di risposta del sistema e snellisce il lavoro dei programmatori nello sviluppo di nuovi KP e componenti del sistema.

Arces MARS ha deciso di ampliare l’offerta di Open Chiron Smart Space ed estenderne le funzionalità tramite un’applicativo Android sviluppato ad hoc.

Il Capitolo 6 motiva la scelta dello sviluppo di un’applicazione Android a supporto del medico, ne presenta il sistema operativo e gli strumenti necessari allo sviluppo di applicazioni.

Il Capitolo 7 tratta infine dell’applicativo Heart Life Doctor (HLD) da me sviluppato a conclusione del percorso di trasformazione del sistema fisico in Cyber Physical Ecosystem. Grazie alla programmazione Java è possibile sfruttare appieno le potenzialità delle librerie Java KPI con estensione SPARQL che ho contribuito ad implementare, trattate nel Capitolo 5.

²<http://www.chiron-project.eu>

Lo scopo di HLD è consentire il monitoring del paziente tramite dispositivo mobile, quindi senza la necessità di essere insieme al paziente stesso. HLD mostra in tempo reale i parametri vitali del paziente: peso, battito cardiaco, saturazione del sangue, pressione, temperatura corporea, etc. Inoltre, informa il medico anche dei parametri d'ambiente quali temperatura ambientale, umidità, concentrazione di polvere e inquinamento ³.

Inoltre, obiettivo di HLD è fornire al medico la possibilità di inviare la cartella clinica completa e/o solo alcuni dei parametri vitali a KP esterni detti *HL7Dispatcher* in ascolto, e ad un servizio web progettato ad hoc da UNIGE, Università di Genova partner del progetto CHIRON.

Heart Life Doctor è stato presentato durante la conferenza ICOST 2012 ⁴: la Conferenza Internazionale sulle Smart Homes ed Health Telematics la cui decima edizione si è tenuta presso Villa Medici Artimino Resort di Firenze nei giorni 12-15 Giugno 2012.

³I rilevatori di polvere e inquinamento non sono attualmente disponibili nel laboratorio Arces MARS. HLD è sensor-ready, cioè già configurato per ricevere informazioni appena i sensori saranno disponibili.

⁴[urlwww.conference-icost.org/](http://www.conference-icost.org/)

Capitolo 1

Il Web Semantico

Il problema della descrizione dei dati ha origini antiche, nel solco di tradizioni e teorie che si sono occupate di rappresentare la conoscenza. Lo strumento informatico è a tutti gli effetti un mezzo di rappresentazione della conoscenza (la conoscenza di *cosa* sia contenuto nello strumento stesso) e dalla sua nascita tenta di definire tipologie di relazioni fra i dati e regole che le definiscano. Tali regole sono necessariamente sovrapposte ai dati che si desidera trattare attraverso lo strumento stesso.

Senza un'adeguata consapevolezza delle regole proprie di ogni strumento di questo genere, e senza l'indispensabile senso critico, si corre il rischio di assumere regole e modelli di descrizione e archiviazione non adeguati ai dati da rappresentare, alterandone i rapporti interni e, di conseguenza, il significato.

L'evoluzione dello strumento informatico e lo sviluppo tecnologico hanno portato sempre più a modificare i criteri e le modalità di organizzazione dei contenuti. Lo strumento informatico a cui si fa riferimento in questo lavoro di tesi è il Web: costituito da una rete di risorse di informazioni, basata sulla infrastruttura di Internet, che rende tali risorse disponibili al più vasto insieme possibile di utenti.

Il Web è uno spazio elettronico e digitale di Internet destinato alla pubblicazione di contenuti multimediali (testi, immagini, audio, video, ipertesti, ipermedia, ecc.) nonché uno strumento per implementare particolari servi-

zi come ad esempio il download di software (programmi, dati, applicazioni, videogiochi, ecc.). I suoi servizi possono essere resi disponibili dagli stessi utenti di Internet, e per quanto riguarda i contenuti, quindi, il Web possiede la straordinaria peculiarità di offrire a chiunque la possibilità di diventare *editore* e, con una spesa estremamente esigua, di raggiungere un pubblico potenzialmente vastissimo distribuito in tutto il mondo [16].

Il web rende disponibili le risorse tramite tre meccanismi:

- un meccanismo per identificare ogni risorsa;
- un protocollo per dereferenziare tali identificativi o ottenere una rappresentazione della risorsa e per negoziare una specifica rappresentazione della risorsa tra le rappresentazioni alternative disponibili;
- un linguaggio ipermediale per rappresentare le risorse che consenta, inoltre, di collegare le risorse tra loro utilizzando degli *hyperlink*

Il World Wide Web Consortium (W3C)¹, è un consorzio di imprese che regola l'evoluzione del web, sviluppando tecnologie e definendo protocolli comuni che ne favoriscano l'evoluzione e assicurino l'interoperabilità.

La pubblicazione della prima pagina web risale ai primissimi anni Novanta (Figura 1.1). Si trattava di un semplice documento di testo, espresso e strutturato tramite l'Hyper Text Markup Language (HTML), dotato di link. Si tratta di un documento che ben rispecchia la natura che ha caratterizzato tutta la prima fase della storia del web. In esso è, infatti, evidente una impostazione unidirezionale nella trasmissione di contenuti dall'autore e/o dal publisher all'utente/visitatore della pagina. Il rapido sviluppo del web e la sua caratterizzazione in termini di ambiente di condivisione, partecipazione, collaborazione in cui gli utenti assumono un ruolo attivo diventano essi stessi produttori e divulgatori di contenuti hanno determinato l'aumento

¹World Wide Web Consortium, <http://www.w3.org>

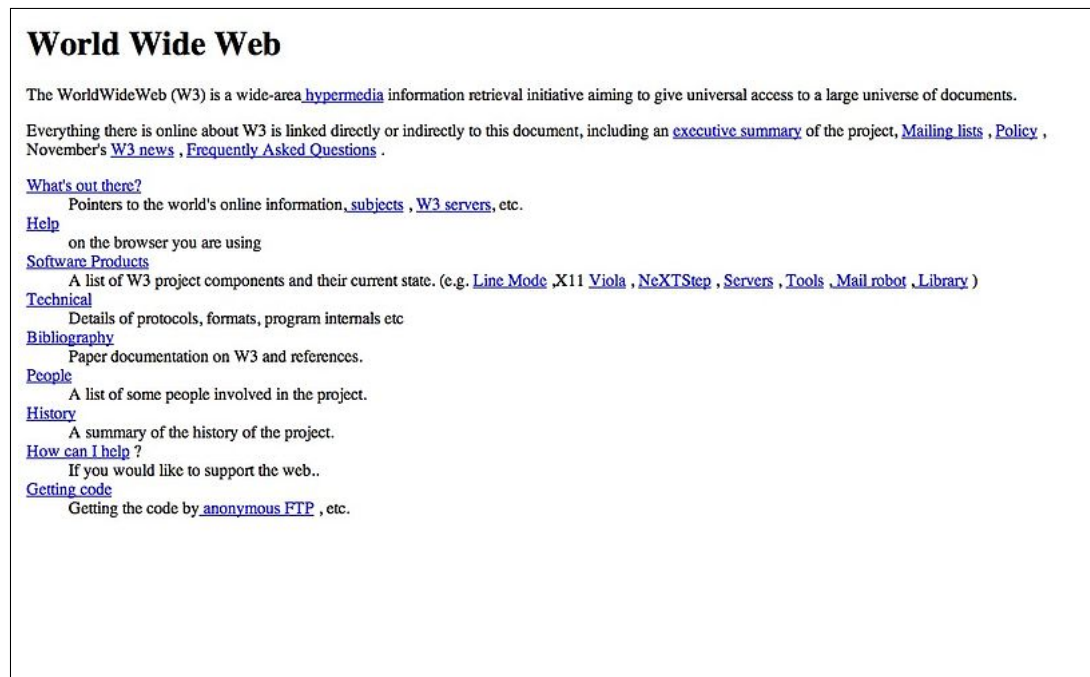


Figura 1.1: Home page del primo sito web, sviluppato da Berners-Lee e messo online il 6 agosto 1991

esponenziale del numero di risorse e il cambiamento netto dell'impostazione suddetta.

La gestione dell'informazione è il cuore del web, e i temi di grande dibattito sono la rappresentazione della conoscenza, la sua acquisizione e la sua elaborazione. È difficile immaginare di poter gestire l'esplosione informativa del web senza un approccio che affronti il problema in modo radicale: ormai molto del tempo trascorso sul web è dedicato alla ricerca di informazioni.

1.0.1 Web Semantico come scommessa del futuro della gestione dell'informazione

Motori di ricerca come Google e indici strutturati come Yahoo sono diventati il punto di riferimento per chiunque, nel mondo della scuola come in quello del lavoro e dello svago. Le risorse informative percepite come effettivamente utili a risolvere le esigenze di chi le ricerca sono però sepolte da una mol-

titudine di documenti scarsamente pertinenti. Questo comporta la perdita di molto tempo per rintracciarle, oltre che la “sindrome da sovraccarico di informazione”.

L’idea sviluppatasi con forza negli ultimi tempi del *Semantic Web* abbraccia l’ipotesi cioè di arrivare a una sorta di formalizzazione – la più generale possibile – di aspetti semantici, di contenuto, di molteplicità di documenti e soprattutto di tipologie di documento reperibili in rete. Obiettivo del Semantic Web è riuscire a fornire risposte con una interrogazione fatta in linguaggio naturale o con un linguaggio formale facile da utilizzare.

Non solo: uno dei maggiori ostacoli è che la maggior parte dell’informazione sul Web è disegnata per essere fruita dall’uomo. La macchina o il mezzo informatico è in grado di fornire una *rappresentazione* di tale informazione, ma in genere non ha una *conoscenza* del suo *significato*. Scopo del Web Semantico diventa quindi anche sviluppare linguaggi e tecnologie per esprimere le informazioni in una forma accessibile e processabile da una macchina.

Le pagine web sono collegate sintatticamente mediante indici che localizzano la URL della pagina e tali collegamenti consentono di identificare le pagine in modo univoco. Uno dei principali limiti di tale impostazione risiede nell’assenza di significato dei collegamenti, in altre parole questo sistema manca di una qualche capacità semantica: i collegamenti dovrebbero non solo condurci in un determinato luogo (la pagina web) ma anche esplicitare la natura della relazione che connette due differenti risorse, due differenti concetti.

Ciò che differenzia questa struttura da quella prevista nel web semantico è proprio l’attribuzione della capacità semantica alle informazioni e ai collegamenti. Lo schema di collegamenti che abbiamo oggi a disposizione e ciò che lo differenzia da quello caratteristico dell’impostazione del web semantico può essere rappresentato con la Figura 1.2.

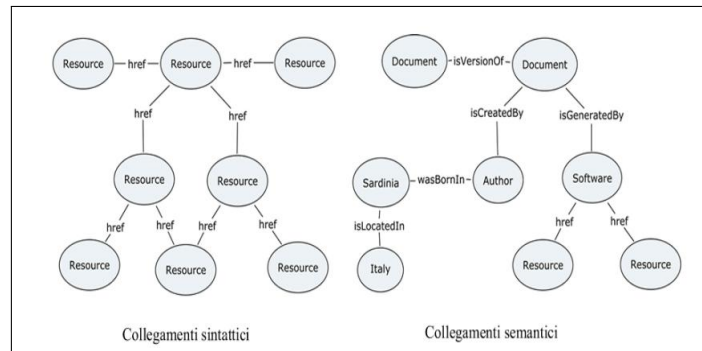


Figura 1.2: Collegamenti sintattici e semantici a confronto

Berners Lee, in un celebre articolo del 2001, ha definito il web semantico come:

Un'estensione del web corrente in cui le informazioni hanno un ben preciso significato e in cui computer e utenti lavorano in cooperazione.

Si tratta di un ambiente in cui sarà possibile pubblicare e rintracciare documenti e informazioni in un formato adatto all'interrogazione, interpretazione ed elaborazione automatica; un web caratterizzato dalla presenza di strutture di collegamento più espressive di quelle attuali. Il web semantico infine, non si configura come un'alternativa al web attuale ma bensì come una sua evoluzione, contraddistinta dall'attribuzione di significato alle informazioni.

1.0.2 Il goal del Web Semantico

Internet consente ai computer di comunicare tra loro, ma non è progettato per insegnare ad essi cosa le informazioni realmente *significhino*.

L'informazione su Internet si basa su documenti scritti in Hypertext Markup Language (HTML): HTML definisce una sintassi che i computer in grado di capire, dice al computer come visualizzare il documento richiesto (ad esempio digitando l'indirizzo URL di un sito web su un browser), ma non è in grado di separare i dati di *presentazione* dalle informazioni contenute nelle

pagine HTML, nè consente ad agenti software di identificare, isolare, estrarre o trattare informazioni pertinenti.

Lo scopo esplicito di Semantic Web è modificare il web dall'essere una piattaforma di *presentazione distribuita* ad una per la *conoscenza distribuita*.

I tentativi di eliminare le carenze di HTML sono realizzati includendo descrizioni “machine-interpretable” dei dati sparpagliati in tutto il web, e quindi abilitare gli agenti software a stabilire collegamenti tra i dati ed a integrare quei dati provenienti da fonti disparate.

In altre parole, le tecnologie del Semantic Web sono per i dati ciò che HTML è per i documenti: proprio come HTML permette di scrivere documenti online per generare l'illusione di un *libro* voluminoso, le tecnologie del web semantico contribuiscono a creare l'illusione di un enorme *database* composto da tutte le fonti di dati disparate nel mondo.

Un esempio rilevante in cui le tecnologie del Semantic Web sarebbero estremamente utile è in aree in cui i concetti vengono espressi con nomi diversi in paesi diversi. In medicina, i farmaci e le malattie sono conosciute con nomi differenti a seconda della posizione geografica in cui vengono scoperte o si trovano. Le tecnologie del web semantico vedono oltre le nomenclature diverse e aggregano le informazioni pertinenti in modo appropriato, senza ambiguità dovuta alle differenze di denominazione.

Ci sono sempre più dati sul Web: dati governativi, dati relativi alla salute, alla cultura generale, informazioni aziendali, informazioni su voli, ristoranti, etc, e le applicazioni dipendono sempre più dalla disponibilità di tali dati. Questo scenario necessita di una infrastruttura adeguata: i dati devono essere disponibili sul Web, accessibili tramite le tecnologie del web standard, e devono essere interconnessi e integrati sul Web stesso.

Per raggiungere un alto livello di interoperabilità l'integrazione dei dati ha bisogno di una infrastruttura adeguata, che renda disponibili i dati indipendentemente dalla loro rappresentazione interna, che unisca le rappresentazioni che ne derivano, che mappi dati diversi su una rappresentazione

astratta, in modo che sia possibile infine *interrogare* il sistema stesso. La descrizione di questa struttura è ancora grezza: si necessita infatti di un vocabolario per rappresentare la conoscenza, di una ontologia e infine di una concettualizzazione della conoscenza.

1.0.3 Architettura del Web Semantico

Per chiarezza di terminologia, va ricordato che la filosofia di base del Web è quella di uno spazio informativo universale, navigabile, con un mapping da URI (Uniform Resource Identifier) alle risorse. Nel contesto del Semantic Web, il termine semantico assume la valenza di “elaborabile dalla macchina” e non intende fare riferimento alla semantica del linguaggio naturale e alle tecniche di intelligenza artificiale.

Come chiaramente descritto in [17], il Semantic Web può funzionare solo se le macchine possono accedere ad un insieme strutturato di informazioni e a un insieme di regole di inferenza da utilizzare per il ragionamento automatico. La sfida del Semantic Web, quindi, è fornire un linguaggio per esprimere dati e regole per ragionare sui dati, che consenta l’esportazione sul web delle regole da qualunque sistema di rappresentazione della conoscenza.

Il termine Semantic Web è spesso usato più specificamente per indicare le tecnologie che lo abilitano. Secondo questa visione, gli ingredienti del Semantic Web sono:

- una struttura (le informazioni devono essere strutturate)
- markup (una forma che sia machine-understandable)
- una tassonomia (l’informazione deve essere classificata)
- degli agenti (che ragionino e svolgano operazioni sulle informazioni)

Pertanto, i componenti fondamentali del Semantic Web sono:

- Un modello di dati (che vedremo sarà rappresentato dal Resource Description Framework (RDF))

- Formati di interscambio di dati (quali saranno RDF/XML, N-Triples, N3)
- Notazioni per la descrizione formale di ontologie o vocabolari (tra cui RDF Schema e Web Ontology Language)

Le principali tecnologie del Web Semantico si inseriscono in una serie di specifiche organizzate a più livelli. Il consorzio W3C ha prodotto una serie di linguaggi standard per la rappresentazione della conoscenza che supportano elaborazioni più o meno sofisticate (e computazionalmente complesse): questi linguaggi rappresentano gli strati della cosiddetta *Semantic Web Tower* o *Semantic Web Stack*, in cui ogni strato è definito come un'estensione semantica² dello strato sottostante (Figura 1.3).

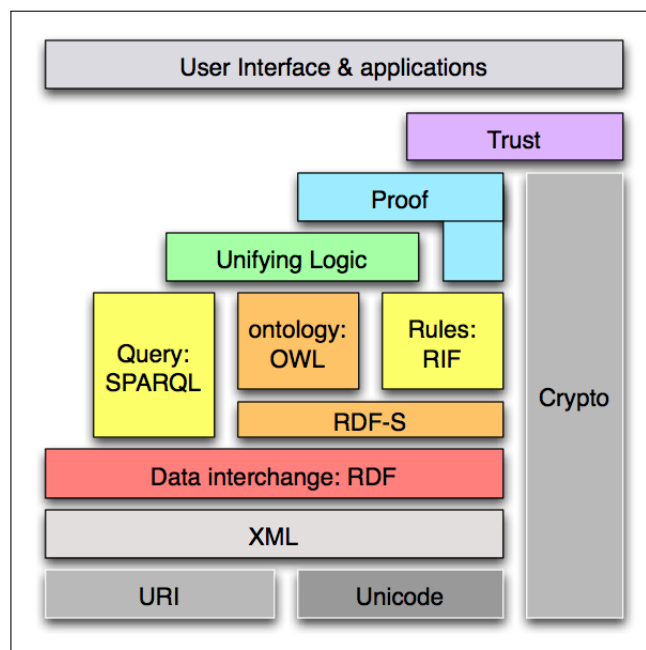


Figura 1.3: Semantic Web Stack

²<http://www.w3.org/TR/rdf-mt/>

The Semantic Web Stack is an illustration of the hierarchy of languages, where each layer exploits and uses capabilities of the layers below. It shows how technologies that are standardized for Semantic Web are organized to make the Semantic Web possible. The stack is still evolving as the layers are concretized.

- **Unicode e URI** - URI è l'acronimo di Uniform Resource Identifier, ossia una stringa che identifica univocamente una risorsa generica. Tale risorsa può essere un indirizzo web, un documento, un'immagine, un file, un servizio, un indirizzo di posta elettronica. Gli URL, Uniform Resource Locator, sono un sottoinsieme degli URI. Mediante URL si identificano risorse specifiche della rete, come ad esempio pagine web, immagini, etc., facendo riferimento alla loro localizzazione e mettendo in evidenza la modalità di accesso (il protocollo). L'Unicode è invece un sistema di codifica che assegna un numero (o meglio, una combinazione di bit) a ogni carattere in maniera indipendente dal programma, dalla piattaforma e dalla lingua (e dal suo sistema di scrittura).
- **XML** - XML (eXtensible Markup Language) è un metalinguaggio che permette di definire sintatticamente linguaggi di mark-up. Nato come linguaggio utile allo scambio dei dati, permette di esplicitare la struttura, quindi la sintassi, di un documento in modo formale mediante marcatori (mark-up) che vanno inclusi all'interno del testo. Per definire la struttura di un documento XML, o più precisamente la grammatica che tale documento deve rispettare, esistono linguaggi quali DTD e XML Schema (XSD); entrambi sono strumenti per eseguire la validazione del documento XML. Ad esempio XSD (XML Schema Definition) permette di specificare vincoli sia strutturali che di contenuto ed è a tutti gli effetti un linguaggio XML. XML si limita a descrivere dati senza entrare nel merito della semantica contenuta. Un documento XML rispecchia la classica struttura ad albero (gerarchica), all'interno del quale le informazioni sono correlate secondo una relazione di subordinazione (classificazione) (Figura

1.4). Nei documenti XML, i dati riportati acquistano un loro significato dettato esclusivamente da un modello gerarchico / classificatorio la cui interpretazione è a discrezione umana e non è comprensibile dalla macchina, non essendo presente un esplicito formalismo di definizione della classificazione.

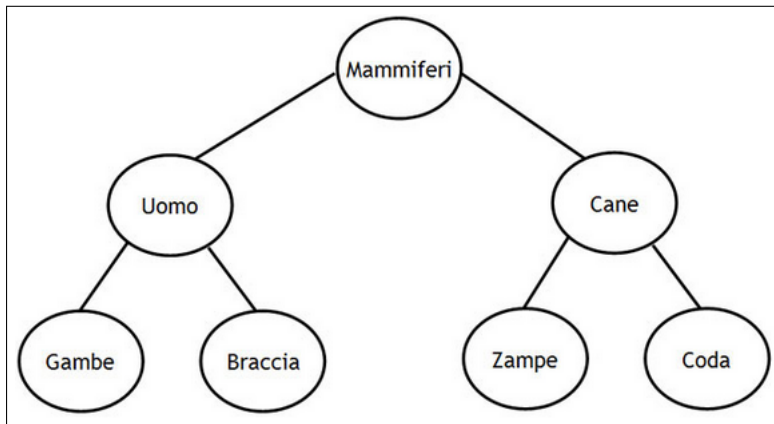


Figura 1.4: La struttura ad albero di XML

- **Resource Description Framework (RDF)** - È il primo livello che abilita il Semantic Web. Per colmare le lacune che impediscono alla macchina di interpretare l'informazione possono essere utilizzati metadati, cioè descrizioni aggiuntive ai dati. Questa è appunto la funzione di RDF, il quale introduce un formalismo per la rappresentazione di metadati basato sul concetto di *statement* (asserzioni) codificati in triple: soggetto - predicato - oggetto.

In questo modo RDF (che è un'implementazione di XML) introduce maggiore capacità espressiva permettendo di definire diverse tipologie di relazione secondo un modello relazionale / predittivo. L'idea è quella di poter utilizzare una struttura dati organizzata secondo un grafo orientato: sui nodi di questo grafo sono poste le risorse (soggetto e oggetto dello statement), mentre gli archi del grafo rappresentano le

relazioni (predicato dello statement). È possibile così aggiungere connessioni (relazioni) tra molteplici risorse, permettendone l'estensione della conoscenza.

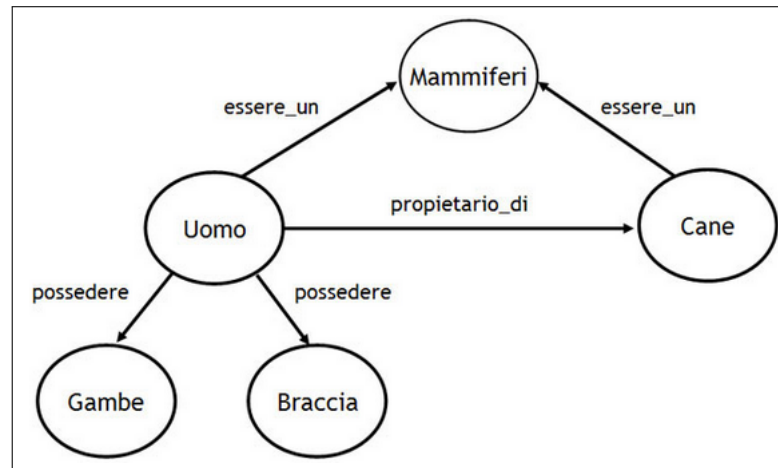


Figura 1.5: Struttura a grafo di RDF

- **RDF Schema** - Il puro RDF serve unicamente per descrivere modelli di dati e può esprimere semplici affermazioni come “il nome del mio cane è Fido”. RDF Schema permette di definire il significato e le caratteristiche delle proprietà e delle relazioni che esistono tra queste e le risorse descritte nel data model RDF. RDF Schema fornisce un insieme di risorse e proprietà predefinite. L'insieme delle risorse e delle relative proprietà di base è detto vocabolario dell'RDF Schema. Attraverso tale vocabolario base è possibile definire specifici vocabolari per i metadati e creare relazioni tra oggetti.

I concetti messi a disposizione da RDF Schema sono quelli di Classe e SottoClasse, SottoProprietà, Dominio e Codominio di una Proprietà, Commenti, Etichette e Informazioni Aggiuntive. Attraverso uno Schema RDF è possibile assegnare un significato ai vari termini utilizzati nelle asserzioni RDF; una risorsa può, per esempio, essere definita come

istanza di una classe (o di più classi) e le classi possono essere organizzate in modo gerarchico, permettendo di derivare, per ereditarietà, nuova conoscenza. Inoltre fornisce un meccanismo di specializzazione delle proprietà, definendone i vincoli d'applicabilità e organizzandole gerarchicamente. Questo modo è possibile aggiungere connessioni (relazioni) tra molteplici risorse permettendone di fatto l'estensione del significato (semantica).

- **Web Ontology Language (OWL)** - OWL estende RDFS con l'aggiunta di costrutti più avanzati per descrivere la semantica delle dichiarazioni RDF. Esso permette di indicare vincoli aggiuntivi quali ad esempio cardinalità, restrizioni di valori, o caratteristiche delle Proprietà come la transitività. Si basa su logiche descrittive e porta così il potere del ragionamento al web semantico.
- **SPARQL** - SPARQL è una specifica sviluppata dal W3C RDF Data Access Working Group a supporto dell'interrogazione dei documenti RDF. Può essere usato per effettuare query su qualsiasi dato descritto tramite RDF, RDFS e OWL. Un linguaggio di query è necessario per il recupero delle informazioni dalle applicazioni del semantic web.
- **RIF** È un sistema a regole basato sulle clausole di Horn, nato per essere utilizzato in motori d'inferenza con lo scopo di estrarre informazioni dalle ontologie e derivarne di nuove.
- **Unifying Logic, Proof, Trust, Crypto** - Questi rimanenti blocchi non sono ancora stati oggetto di lavoro da parte del W3C, e la loro definizione è demandata al futuro. Unifying Logic è il layer che permette l'unificazione di tutti i risultati ottenuti ai livelli precedenti; Proof è il sistema di prova che deve controllare la validità delle conclusioni; Cryptography è il livello di crittografia estesa a tutti i livelli già menzionati. Infine, il più problematico da definire sembra essere il livello Trust, ovvero la fiducia. La potenza del Semantic Web si basa

infatti (come già visto nei precedenti paragrafi) sul fatto che chiunque possa asserire qualsiasi cosa su qualsiasi risorsa, anche se non ne è il proprietario. In questo modo gli agenti possono sfruttare l'intelligenza collettiva della rete. È chiaro però che tale meccanismo presta il fianco ad usi scorretti se non addirittura fraudolenti.

Capitolo 2

Elementi di architettura del Web Semantico come discussi nel progetto RECOCAPE

La prima parte del mio lavoro di tesi è lo sviluppo di un corpo di lezioni interamente in lingua inglese e della durata complessiva di un mese, per uno dei corsi previsti dal progetto RECOCAPE. Obiettivo del corso è fornire un training di base sulle tecnologie di sviluppo per il Web Semantico, ed avrà luogo ad Ottobre 2012 presso Smart Village Egypt (Il Cairo, Egitto).

Lo sviluppo di un corpo di lezioni è un'attività didattica molto formativa. Richiede che sia chiaro qual è il pubblico a cui ci si rivolge, e una approfondita conoscenza degli argomenti oggetto del corso.

L'attività di studio preparatoria alla stesura del corso ha avuto una durata di circa due mesi, durante i quali ho avuto la possibilità di condurre prove pratiche e testare le tecnologie messe a disposizione del laboratorio Arces MARS in cui ho svolto la mia attività di tesi.

2.1 Il progetto RECOCAPE nella Commissione Europea Ricerca e Innovazione



Figura 2.1: Logo del progetto RECOCAPE

Il consorzio formato dall'egiziana SECC, la spagnola Tecnalia, la finlandese VTT e l'Università di Bologna, da cui nasce il progetto RECOCAPE, è la risposta alla *Call FP7* dell'Unione Europea per il rafforzamento della cooperazione con gli stati vicini all'Europa.

Il *Seventh Framework Programme* (FP7) è il settimo programma quadro per la ricerca e lo sviluppo tecnologico promosso dalla Commissione Europea Ricerca e Innovazione. Ha una durata di sette anni, dal 2007 al 2013. Il programma usufruisce di uno stanziamento di bilancio che supera i 50 miliardi di euro, beneficiando di un aumento del 41% rispetto al quadro precedente (2004). FP7 è lo strumento principale per soddisfare le necessità dell'Europa in termini di posti di lavoro e competitività e per permettere all'Europa di continuare ad avere un ruolo di guida nell'economia globale della conoscenza.

Per essere complementari con i programmi di ricerca nazionali, le attività di ricerca finanziate in base al FP7 devono avere un *valore aggiunto europeo*. Un aspetto fondamentale del valore aggiunto europeo è il carattere transazionale di molte azioni: i progetti di ricerca devono essere condotti da consorzi costituiti da partecipanti provenienti da diversi paesi europei e non, mentre le borse di ricerca del FP7 prevedono la mobilità oltre i confini nazionali. Ciò dipende dal fatto che molte sfide nel campo della ricerca (ad esempio la

ricerca sulla fusione nucleare) sono così complesse da poter essere affrontate soltanto a livello europeo.

In generale, FP7 ed i programmi quadro per la ricerca hanno due principali obiettivi:

- rafforzare la base scientifica e tecnologica dell'industria europea e incoraggiare la sua competitività internazionale;
- promuovere la ricerca che appoggia le politiche dell'UE.

RECOCAPE è quindi la risposta di SECC, Tecnalta, VTT e Unibo.



Figura 2.2: I partecipanti al progetto RECOCAPE



Figura 2.3: Software Engineering Competence Center (SECC)

Software Engineering Competence Center (SECC) è una dei principali attori del settore ICT (Information and Communication Technology) in Egitto. Fu inaugurata nel 2001 dal Ministero delle Comunicazioni e della Tecnologia dell'Informazione (MCIT); la sua missione è sostenere lo sviluppo dell'industria del software in Egitto attraverso il miglioramento delle pratiche di ingegneria del software ai più elevati livelli di maturità, raggiungendo una forte presenza sul mercato globale. Nel 2005 è stata incorporata nella struttura organizzativa della Information Agency Development Technology Industry (ITIDA).



Figura 2.4: Fundación Tecnalia Research and Innovation

La *Fundación Tecnalia Research and Innovation* è un'organizzazione di ricerca privata senza fini di lucro con sede nei Paesi Baschi, nata dalla fusione di otto centri di ricerca preesistenti con un lungo e ampio curriculum di ricerca sviluppo e innovazione tecnologica. Con circa 1500 ricercatori e quasi 3.800 clienti, Tecnalia è il maggiore centro di ricerca privato in Spagna e il quinto in Europa; le sue attività di R&D e di Trasferimento Tecnologico sono rivolte ai settori dell'Industria e dei Trasporti, delle ICT, dello Sviluppo Sostenibile, dei Sistemi di Innovazione, della Salute e Qualità di Vita. Tecnalia è specializzata nello sviluppo di soluzioni e tecnologie innovative in settori quali robotica e manipolazione, sistemi manifatturieri intelligenti e meccatronica, con particolare riferimento a processi non convenzionali e macchine utensili avanzate per materiali speciali, energy-efficient/green manufacturing, dematerializzazione delle macchine utensili e diagnostica predittiva.



Figura 2.5: VTT Technical Research Centre

Fondata nel 1942, il *VTT Technical Research Centre* della Finlandia è la più grande organizzazione multi-tecnologica di ricerca applicata nel Nord Europa. VTT offre soluzioni tecnologiche di fascia alta e innovazione nei servizi. VTT crea innovazione e una gamma considerevole di tecnologie di livello mondiale e dei servizi di ricerca applicata migliorando in tal modo competitività e competenza dei propri clienti. VTT è una parte del sistema d'innovazione finlandese sotto il dominio del Ministero del Lavoro e dell'Economia ed è un'organizzazione senza fini di lucro.



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Figura 2.6: Alma Mater Studiorum Università di Bologna

L'*Università di Bologna* ha origini molto antiche che la indicano come la prima Università del mondo occidentale. La sua storia si intreccia con quella di grandi personaggi che operarono nel campo della scienza e delle lettere ed è riferimento imprescindibile nel panorama della cultura europea. L'origine dell'Università di Bologna è attribuita all'anno 1088, data convenzionale

fissata da un comitato di storici guidato da Giosuè Carducci. L'Università mantiene una posizione di centralità sulla scena della cultura mondiale fino al periodo tra le due guerre, quando altre realtà iniziano a prendere il sopravvento nel campo della ricerca e della formazione. Essa è dunque chiamata a rapportarsi con le Istituzioni dei Paesi più avanzati intraprendendo un percorso di aggiornamento e crescita. Tra le sfide raccolte con successo, l'Università si impegna in quel confronto con la nuova dimensione europea che condurrà all'innovazione del sistema universitario.

Gli obiettivi del progetto RECOCAPE sono:

- Lo sviluppo del piano strategico della tecnologia messa a punto da SECC;
- lo sviluppo e la fornitura di moduli di formazione per costruire competenze in Service-Oriented Architecture, Enterprise Service Bus, Web Semantico, Model-Driven Development, ed Embedded Ubiquitous Computing;
- l'esecuzione di rilevanti e congiunti esperimenti negli ambiti tecnologici di cui sopra;
- Scambio di personale SECC con personale di TECNALIA per la preparazione delle proposte e dei relativi progetti per FP7;
- Diffondere la conoscenza attraverso l'organizzazione di eventi chiave, come SPI EuroMed, per coinvolgere i soggetti locali e generare un interesse regionale.
- Valorizzazione dei risultati attraverso l'implementazione di un progetto cluster che coinvolge imprese locali in materia di ICT.

Ulteriori informazioni su RECOCAPE sono disponibili all'indirizzo <http://www.secc.org.eg/RECOCAPE/>.

2.2 RECOCAPE Work Package 3 - Il Corso

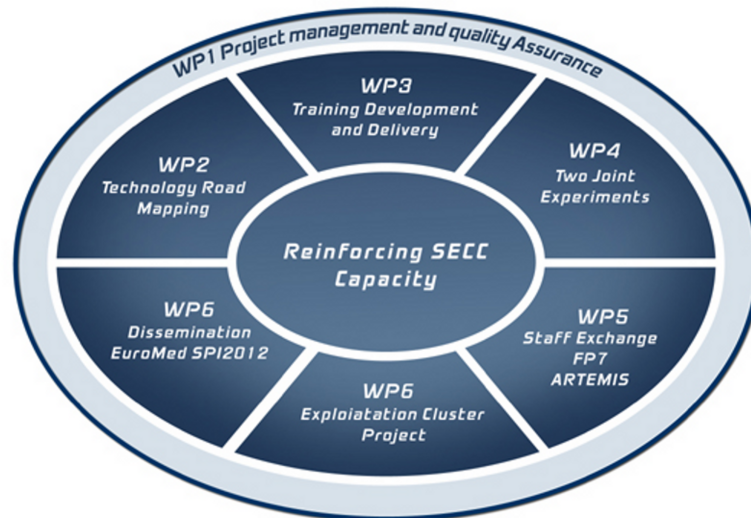


Figura 2.7: I work packages del progetto RECOCAPE

Il progetto RECOCAPE si articola in sette *Work Packages* (WP). La mia attività di lavoro si inserisce all'interno del Work Package 3 (WP3) “*Training Modules Development and Delivery*”. Il WP3 si articola in quattro *tracks*:

- Track 1: Service Oriented Architecture/Enterprise Service Bus (SOA/ESB)
- Track 2: Semantic Web
- Track 3: Model Driven Development (MMD)
- Track 4: Ubiquitous Computing

La Track 2 si occupa nello specifico del Web Semantico e delle tecnologie che lo abilitano. Si tratta di un corpo di lezioni sviluppato in team da me - Patrizia De Luca-, due dei tre correlatori di questo lavoro di tesi - Alfredo D'Elia e Francesco Morandi- e Luca Faggianelli. Obiettivo della Track 2 è fornire un training di base sulle tecnologie di sviluppo per il Web

Semantico della durata complessiva di un mese e un impegno previsto di 25 giorni lavorativi.

Il corso si terrà ad Ottobre 2012 presso *Smart Village Egypt*, il Business Park egiziano situato presso Cairo-Alessandria Desert Road nella zona di Abu Rawash, a ovest della capitale. Smart Village Egypt è un complesso moderno di edifici amministrativi e spazi verdi, ed è il primo Communication and Information Technology Cluster (CIT) egiziano. Ospita aziende multinazionali e locali, organizzazioni governative e finanziarie, istituti scolastici e centri di ricerca e sviluppo, fornendo le strutture per meeting, summit, lezioni, conferenze e in generale attività di business.

Il corso, interamente in lingua inglese, tratta nello specifico le tecnologie che abilitano il web semantico ed è suddiviso nei seguenti moduli:

- **Overview of semantic web technologies** - Il primo modulo fornisce una panoramica di largo respiro su cosa si intende per Web Semantico e si propone di motivarne l'importanza teorica e tecnologica.
- **Semantic web architecture** - Il secondo modulo è l'introduzione a un insieme di lezioni molto specifiche, ciascuna delle quali analizza un layer del Semantic Web Stack.
- **XML** - Un corso esaustivo su XML a partire dalle basi: regole sintattiche, struttura ad albero, regole di naming, attributi, validazione e un accenno a DTD, XSD, DOM, e namespaces. L'intero capitolo è corredato da esempi di codice e best practices.
- **Resource Description Framework (RDF)** - Alla stregua del capitolo precedente, questo modulo offre nozioni di base ed avanzate su RDF ed il suo utilizzo pratico, e su RDF Schema (RDFS).
- **Ontologies and OWL** - Capitolo esaustivo sulle ontologie in generale ed in particolare su OWL e le versioni Lite, DL e Full.

- **Building blocks for semantic technology applications** - Questo modulo esplora da vicino uno degli esempi più convincenti di applicazione delle tecnologie del web semantico: la piattaforma Smart M3, il progetto SOFIA e l'attività svolta dal centro Arces MARS. In chiusura viene presentato un tutorial per la piattaforma Protegè, pensato anche e soprattutto per essere utilizzato nelle lezioni del corso di Elaborazione Delle Informazioni M, in programma nel CDL Ingegneria Informatica Laurea Magistrale con sede didattica a Cesena.

L'outline globale del corso è riportato nella sua ultima revisione (Maggio 2012). Sono possibili ulteriori modifiche prima dell'inizio delle lezioni ad Ottobre 2012.

- **1, 2 - Semantic Web Introduction (38 slides)**
 - Introduction
 - State of The Art
 - The Goal of the Semantic Web
 - Interoperability
 - The Semantic Web Architecture
 - Semantic Web Stack - Overview
 - Next Chapter
- **3 - XML (Complete Course - 90 slides)**
 - What is XML
 - Differences with HTML
 - XML Simplifies...
 - XML Tree Structure
 - Examples
 - XML Syntax Rules

- Entity References
 - XML Elements
 - Naming Rules
 - XML Attributes
 - XML Validation
 - DTD
 - XSD
 - DOM
 - Namespaces
 - Related Technologies
 - XML Semantics
 - Summary
 - Next Chapter
- **4 - RDF (Complete Course - 53 slides)**
 - Introduction
 - Examples
 - Serialization Formats
 - RDF Principles
 - RDF URIs
 - RDF Graphs
 - The Global Graph
 - RDF Statements
 - RDF Element
 - RDF Description
 - Properties vs Attributes
 - RDF Container Elements

- RDF Collections
 - RDF SCHEMA
 - RDFS Classes
 - RDFS Properties
 - RDFS Containers
 - RDFS Collections
 - Summary
 - Next Chapter
- **5 - Ontologies and OWL**
 - History
 - Overview
 - Sublanguages: OWL Lite, DL, Full
 - Syntax and Construct
 - OWL2
 - Tools - Protege
 - Tools - RDF stores
 - Tools - Integrated IDE
 - Description logic, TBox / Abox
 - Examples - REGALS
 - Examples - DOLCE
 - Examples - FOAF
 - Rules - Non/Monotonic rules
 - Rules - Formats
 - Rules - RuleML
 - Rules - SWRL
 - Rules - RIF

- SPARQL
- Summary
- Next Chapter

- **6a - Smart M3 Introduction (63 slides)**
 - Physical Space vs Smart Space
 - What is a Smart Environment
 - Vision
 - Distribution of Computation
 - Smart Environment Axioms
 - Benefits
 - Smart M3 Platform - Introduction
 - Smart M3 Architecture
 - Smart M3 Domain Model
 - Smart M3 Logical Architecture
 - Smart M3 Implementation Architecture
 - Smart M3 Principles
 - Notion of Application
 - SSAP
 - SIB
 - KP
 - A short Recap
 - SOFIA - Introduction
 - SOFIA - Guiding Principles

- **6b - Sensor Data Ontology (45 slides)**
 - What is Protegè
 - Arces MARS Ontology

- Build the ontology in Protegè

Il corso è interamente sviluppato dalla sottoscritta, fatta eccezione per il capitolo *Ontologies and OWL*, con la supervisione di Tullio Salmon Cinotti - relatore del presente lavoro di tesi - , Alfredo D'Elia e Francesco Morandi.

Informazioni su RECOCAPE sono disponibili all'indirizzo <http://www.secc.org.eg/RECOCAPE/>.

Informazioni su Smart Village Egypt sono disponibili all'indirizzo <http://www.smart-villages.com/>.

2.3 Il Corso - Semantic Web Introduction

Il blocco completo di lezioni di introduzione al Web Semantico e alla sua architettura è riportato in maniera integrale nel Capitolo 1.

2.4 Il Corso - Smart Spaces e Smart M3

Il blocco completo di lezioni relative a Smart Environment, Smart Spaces e all'architettura Smart M3 è riportato in maniera integrale nel Capitolo 3, a partire dalla sezione 3.2.

2.5 Il Corso - Sensor Data Ontology

Il blocco completo di lezioni relative alla Sensor Data Ontology per Open Chiron Smart Space, con riferimento all'architettura logica e fisica e riportato in parte nel Capitolo 4.

Capitolo 3

Cyber Physical Systems e Smart Environments

In questo capitolo partendo dalle definizioni teoriche di Smart Environment e Cyber Physical System mostreremo che è possibile ottenere gli scopi prefissati di interoperabilità integrando le tecnologie del web semantico, al fine di ottenere una piattaforma tecnologica per una gestione della salute del paziente che sia efficace, copra il ciclo completo di cura e metta il cittadino come “persona” al suo centro. Come vedremo, con questo progetto è condiviso lo scenario e l’obiettivo di realizzare un sistema per il controllo della salute personalizzato.

3.1 Il Web Semantico per la Telemedicina

In letteratura i primi riferimenti alla telemedicina risalgono agli anni settanta [18] ma in realtà la telemedicina ha preso piede velocemente solo dagli anni novanta cioè solo quando i servizi di comunicazione dati sono stati resi disponibili in larga scala sulle linee telefoniche [19].

L’OMS (Organizzazione Mondiale della Sanità) adotta nel 1997 la seguente definizione:

La telemedicina è l'erogazione di servizi sanitari, quando la distanza è un fattore critico, per cui è necessario usare, da parte degli operatori, le tecnologie dell'informazione e delle telecomunicazioni al fine di scambiare informazioni utili alla diagnosi, al trattamento ed alla prevenzione delle malattie e per garantire un'informazione continua agli erogatori di prestazioni sanitarie e supportare la ricerca e la valutazione della cura.

È implicito in tale definizione il concetto per cui, quando si parla di telemedicina essa non deve essere confinata al singolo aspetto disciplinare introdotto, bensì a quell'insieme di applicazioni pratiche che oggi sono già in fase di implementazione.

Da allora un ampio numero di sistemi è stato realizzato con lo scopo principale di ridurre il tempo che il personale medico deve spendere per il paziente garantendo al contempo un alto livello di qualità. Inizialmente i sistemi di telemedicina erano dedicati a specifiche applicazioni quali tele-cardiologia, teleradiologia, tele-dialisi,.. Con l'avvento di nuove piattaforme, rese possibili dalla presenza di Internet, incluse piattaforme mobili e con lo sviluppo di Location Based Service, la collezione di dati acquisiti da differenti sensori (indossabili, stazionari, posizionati in casa piuttosto che in altri ambienti) è diventata una realtà. Lo sviluppo di nuovi sensori e attuatori hanno aperto alla strada a nuovi scenari in campo sanitario e nuove opportunità per i servizi di healthcare attualmente investigate in progetti di ricerca.

3.1.1 Telemedicina come Tele-assistenza: Health Smart Homes

Tutti questi progetti traggono vantaggio dai progressi dell'ICT; questi sistemi come molti altri presenti sul mercato consentono di fornire assistenza a distanza e collezionare dati in modo automatico ma non forniscono un flusso di decisioni automatiche.

Questa funzionalità è invece fornita in quelle che sono le più moderne soluzioni di telemedicina ovvero le *Health Smart Homes*; questi sistemi nascono con l'obiettivo di facilitare la vita degli anziani o disabili nelle loro case. Per Smart Home si intende un'abitazione che incorpora una rete di comunicazione che è in grado di connettere apparecchi elettronici e servizi e permette di controllarli e monitorarli da remoto [20]. In particolare le Health Smart Homes dovrebbero garantire una vita autonoma a quelle persone che altrimenti sarebbero collocate in istituti [21].

Queste soluzioni sono nate dalle ricerche nel campo degli ambienti intelligenti e cercano di adattare la tecnologia ai bisogni dell'utente basandosi su tre concetti principali: ubiquitous computing, ubiquitous communication e intelligent user interface. I risultati e le principali ricerche in tale ambito possono essere trovate in [22, 23, 24, 25, 26, 27, 28].

Una caratteristica comune e limitante di tutte le soluzioni sopra menzionate è che ogni implementazione è basata su propri protocolli proprietari, piattaforme, store di informazioni e modelli di rappresentazione dei dati. L'informazione prodotta da un sistema rimane all'interno del sistema stesso; la conseguenza ovvia è una frammentazione lungo il ciclo di assistenza sanitaria. Questa frammentazione è non solo causa di un aumento generale dei costi ma anche una barriera per quanto riguarda l'innovazione di servizi sanitari.

Se i dati sullo stato di salute e dati ambientali raccolti da sistemi di monitoraggio ambientale fossero interoperabili, sarebbe possibile combinarli e generare una nuova conoscenza in modo tale da poter realizzare applicazioni innovative a beneficio di differenti istituzioni e utenti.

Tale sistema, in cui si richiede che gli oggetti fisici possano aver integrati elementi con capacità di calcolo, memorizzazione e comunicazione, e che sono collegati in rete tra loro si definisce *Sistema Cyber Fisico* o *Cyber Physical System* (CBS).

Nelle prossime sezioni e capitoli verrà esposto in che modo le tecnologie del web semantico aiutano e migliorano la vita delle persone e le tecnologie per

le Health Smart Homes proprio tramite i sistemi cyber fisici.

3.2 I Sistemi Cyber-Fisici

Un sistema di tipo Cyber-Physical (CPS) è un sistema in cui si implementano una combinazione e collaborazione strettamente integrata tra componente elaborativa ed elementi fisici; derivano direttamente dai sistemi embedded, con cui condividono buona parte dell'architettura hardware e software: mentre questi ultimi sono fortemente votati ad una componente elaborativa, i CPS possiedono un'accezione maggiorata alla parte di interfaccia microprocessore / componenti esterni.

Lo sviluppo di un sistema cyber fisico richiede competenze informatiche eterogenee (algoritmi, reti, sviluppo di sistemi distribuiti, sviluppo di architetture SW, programmazione embedded, elaborazione ed analisi dei dati, intelligenza artificiale, robotica) ed un approccio multidisciplinare ai problemi.

I Cyber-Physical Systems devono possedere le seguenti caratteristiche:

- **Availability:** la probabilità che un sistema funzioni in un determinato tempo t' deve essere costante e pari a quella al tempo $t = 0$. Questo deve comportare un'alta affidabilità sia hardware che software;
- **Maintainability:** il corretto funzionamento anche dopo l'occorrenza di una situazione di errore. Ciò può essere riferito sia alla parte hardware (il default di uno o più componenti) che alla parte software del sistema (ad esempio il crash del sistema);
- **Safety:** il sistema non deve causare errori o malfunzionamenti nel sistema controllato;
- **Security:** le comunicazioni ed i dati utilizzati per poter rimanere confidenziali ed autentiche.

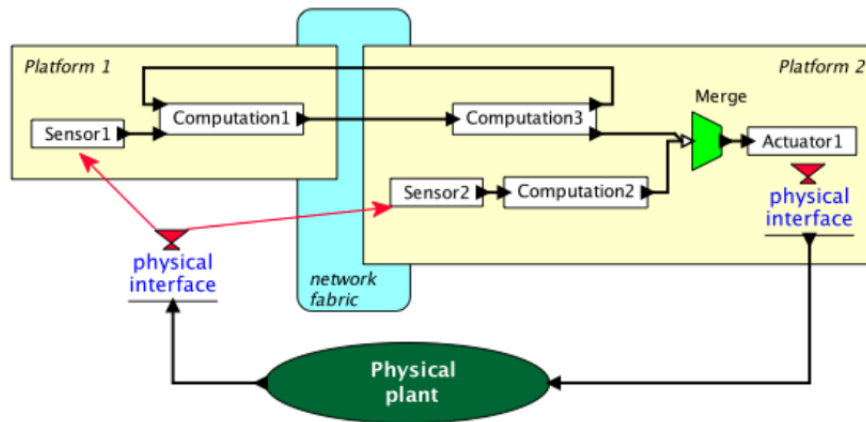


Figura 3.1: Schema concettuale di un Cyber-Physical System (CPS)

Dal punto di vista prettamente ingegneristico il sistema deve avere i seguenti requisiti funzionali:

- Sistemi Real time o sistemi Reattivi
- Dimensioni contenute in termini di spazio e peso
- Basso costo
- Capacità di limitare la potenza assorbita dal sistema

Edward A. Lee [29] conduce un'ottima analisi e riflessione sui sistemi Cyber-Physical, focalizzando il fulcro del suo discorso intorno al modello concettuale dei linguaggi di programmazione attualmente utilizzati che si interfacciano con il livello hardware sottostante. I sistemi fisici introducono requisiti di affidabilità e sicurezza qualitativamente molto differenti da quelli generalmente trattati da una macchina general purpose. Inoltre, i componenti fisici sono - anche intuitivamente - molto diversi da componenti software object-oriented. Vengono così a crollare le astrazioni standard di base, come le chiamate a metodo oppure i thread, che ora non possono più funzionare nell'integrazione col mondo esterno. I sistemi CPS porteranno, già nel breve termine, una potenziale rivoluzione in ogni settore, dalla medicina al settore automobilistico,

al controllo di processo, risparmio energetico, sistemi di difesa, edifici intelligenti. . . Tuttavia, la rivoluzione starà nel fatto che tutti i sistemi saranno collegati alla rete, in un futuro scenario di pervasive computing estremo.

Analoghi problemi li possiamo trovare per quanto riguarda la concorrenza, fondamentale quando si parla di CPS essendo loro stessi sistemi concorrenti. Già oggi i sistemi devono reagire, in tempo reale, a stimoli provenienti da sensori multipli attraverso un set di attuatori in modo concorrente l'uno all'altro.

Quello che si vuole ottenere dalla prossima generazione di sistemi cyber-physical è un modello elaborativo concorrente, deterministico, predicibile. Per realizzare completamente il potenziale dei CPS dunque bisogna ripensare alle astrazioni centrali dei sistemi. Miglioramenti incrementali porteranno alla semplificazione nella realizzazione dei sistemi altamente integrati nel mondo reale, senza dimenticare che la vera sfida consisterà nell'affrontare la progettazione non più come l'unione di due sistemi separati - quello elaborativo e quello fisico - ma come un unico sistema interagente.

In questo nuovo scenario, il software gioca un ruolo di fondamentale importanza, imponendosi come gestore e coordinatore di dispositivi hardware e del sistema nel suo insieme, talvolta operando in situazioni critiche, senza margine di errore. Per tale motivo lo sviluppatore è chiamato alla programmazione di piattaforme il cui compito esula dal perseguimento di un singolo obiettivo, ma di molteplici finalità ove l'interazione e lo scambio di informazioni con altri dispositivi è un requisito fondamentale.

Compito dell'ingegneria del software è quello di mettere a disposizione validi strumenti che permettano un utilizzo semplice, ma allo stesso tempo potente, di tali piattaforme, sollevando lo sviluppatore dai compiti di più basso livello innalzando le astrazioni di prim'ordine. Ne risulterà così un tempo di sviluppo inferiore, maggiore stabilità del codice ottenuto e migliori strumenti per la rilevazioni di eventuali errori concettuali. In questo quadro, vengono considerati quindi nuovi paradigmi di programmazione, tra i quali quello ad agenti da me sperimentato in questa tesi, in quanto perfettamente calzante

alla metafora che ne viene rappresentata: ogni CPS infatti può essere interpretato come l'ambiente in cui è immerso l'agente (la nostra entità attiva), il quale dispone di opportune strumenti di percezione ed attuazione, il cui compito è il controllo globale o locale del sistema.

3.2.1 Integrazione del Web Semantico nei Sistemi Cyber-Fisici

Per realizzare appieno il potenziale dei Cyber Physical Systems le astrazioni di base dell'informatica in generale devono essere ripensate, chiaramente tramite miglioramenti incrementali. Perché vi sia una orchestrazione efficace dei processi software e fisici è necessario ricorrere a modelli semantici che riflettano le proprietà di interesse per entrambi.

Allo stato dell'arte, i dispositivi elettronici ed informatici a disposizione dell'uomo sono limitati nelle funzioni proprie di un sistema Cyber Fisico. In realtà, la complessità nella progettazione di sistemi di questo tipo, cioè *context-aware*, riguarda settori di ricerca che coinvolgono differenti domini. Infatti questi sistemi richiedono una conoscenza di elementi eterogenei che vanno dai sensori alle architetture hardware, dai sistemi di comunicazione ai modelli di rappresentazione dei dati.

Per arrivare ad ottenere sistemi del genere dobbiamo prima introdurre un altro concetto che è quello di Smart Space. Ed è altrettanto importante sottolineare che il concetto chiave che abilita l'esistenza di sistemi così complessi è il concetto di *interoperabilità*. La definizione proposta da Brownsword [43] si avvicina bene all'uso del concetto di interoperabilità trattata in questa tesi. Brownsword definisce l'interoperabilità come l'abilità di un insieme di entità in grado di comunicare di condividere specifiche informazioni e operare su queste informazioni in accordo con una determinata semantica. Questa definizione combacia con il nostro scopo dove le entità in grado di comunicare sono dispositivi elettronici e persone e non è specificato un determinato tipo di comunicazione. Un altro punto chiave di questa definizione che sarà

fondamentale in questa tesi è l'utilizzo di una semantica, cioè il significato dell'informazione.

In questo lavoro sarà considerata una netta distinzione tra i seguenti livelli concettuali di interoperabilità:

- *Information level* - per information (o semantic) interoperability si intende la condivisa comprensione del significato dell'informazione;
- *Service level* - la service interoperability è l'abilità di più sistemi di condividere, scoprire e comporre servizi;
- *Communication level* - la communication level è intesa come l'interoperabilità a livello OSI 1,2,3 e 4.

Questa tesi si focalizzerà principalmente sull'interoperabilità semantica. Un ramo dell'intelligenza artificiale chiamato rappresentazione dell'informazione studia questa problematica da anni. Una parte dell'informazione chiamata *ontologia* è l'elemento più all'avanguardia dell'ultimo decennio in questo campo. Nei prossimi capitoli si mostrerà come le ontologie e le altre tecnologie del web semantico (quali RDF) abilitano i sistemi cyber-fisici e gli smart spaces.

3.3 Smart Spaces

Prima di definire uno Smart Space (SS) è necessario definire uno Smart Environment. Cook [44] definisce uno Smart Environment

Un sistema in grado di acquisire e applicare conoscenza su un ambiente e capace di adattarsi ai propri abitanti in modo tale da migliorare la loro esperienza in quell'ambiente.

Lo Smart Space è una porzione fondamentale dello Smart Environment. Per SS intendiamo una rappresentazione digitale dell'informazione rappresentante il modo fisico e tale da offrire un'estensione per la ricerca dell'informazione

attraverso un *Semantic Information Broker* (SIB). La SIB è un'entità (posta sull'information level) per la memorizzazione, la condivisione e la gestione dell'informazione dello SS.

L'informazione è rappresentata nella SIB mediante un grafo che può essere scritto e/o letto; l'informazione del grafo è rappresentata mediante un'ontologia per fornire univocamente significato all'informazione.

Gli Smart Spaces sono utilizzati per raccogliere dinamicamente dati locali che portano informazione sul mondo reale (come la temperatura di una stanza, le condizioni fisiologiche di un individuo, il traffico di una strada) e mettono a disposizione tali dati per applicazioni che possono beneficiarne. Uno SS è una parte di uno Smart Environment poiché è il componente centrale usato per condividere e memorizzare l'informazione che necessita di essere distribuita in un ambiente. Diversi progetti di ricerca su Smart Environment sono presenti in letteratura. Lo schema di Cook è mostrato in Figura 3.2.

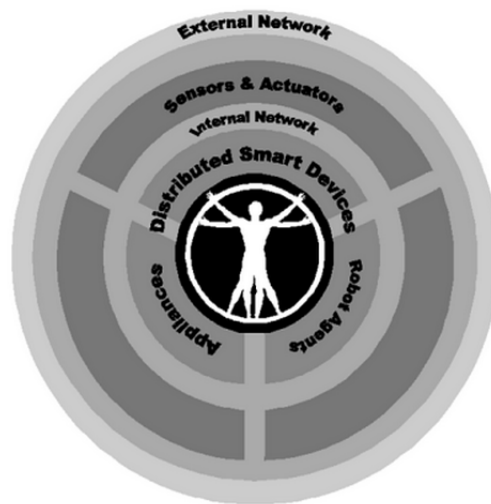


Figura 3.2: Schema di Smart Environment secondo Cook [44]

Alla base di uno SE è presente un ambiente “connesso” che permette ai dispositivi di comunicare facilmente. Sopra il livello di comunicazione, sensori

e attuatori permettono al sistema di comprendere ciò che sta accadendo nell'ambiente per agire di conseguenza. Al centro di tutto si trovano dispositivi intelligenti, apparecchiature e, elemento più importante di tutti, la persona che interagisce e vive nell'ambiente.

Come già citato, diverse soluzioni sono già presenti e consentono di costruire ambienti che rilevano l'utente mediante diverse tecnologie (video, audio,..) e forniscono vari servizi; il problema principale è la mancanza di interoperabilità avendo ogni realizzazione un proprio protocollo e modello di database per la rappresentazione dei dati.

Il potenziale offerto da questi device ha accresciuto l'interesse nel rendere questi dispositivi *interoperabili*. Allo stato dell'arte tuttavia, ogni device dovrebbe implementare molteplici protocolli per comunicare con l'ambiente e con dispositivi diversi in domini diversi.

Un tentativo per definire e implementare un framework per l'interoperabilità è proprio il Semantic Web. La piattaforma Smart M3 che verrà ora introdotta ne è una possibile implementazione, e come vedremo consente a dispositivi diversi di condividere ed accedere ad informazioni semantiche locali e remote.

3.3.1 Smart M3

In questa tesi, la piattaforma di riferimento utilizzata come Smart Space è Smart-M3 [45, 46, 47], sviluppata all'interno del progetto europeo SOFIA (Smart Object for Intelligent Applications)¹[36, 37].

Sofia è un progetto Artemis della durata di tre anni e coinvolge diciannove partner di quattro nazioni europee diverse. Obiettivo di SOFIA è realizzare una piattaforma innovativa di *interoperabilità semantica*.

¹www.sofia-project.eu

Architettura

La *interoperability platform* Smart M3 consiste di due macro-componenti: un *Semantic Information Broker* (SIB) e molteplici *Knowledge Processors* (KP). In questo modello architetturale l'informazione è contenuta all'interno di una o più SIB.

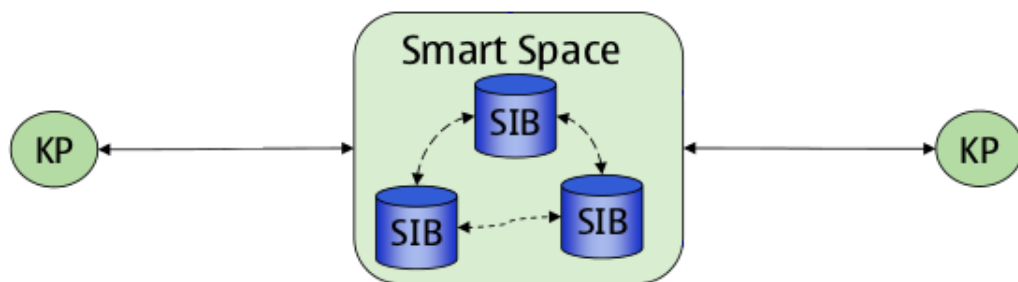


Figura 3.3: Architettura di Smart M3

Il modello di dominio che descrive i principali componenti di uno Smart Space sono rappresentati in figura 3.4. Nel caso più semplice, tutta l'informazione di uno smart space è immagazzinata da una sola SIB, ma vi è la possibilità concettuale di connettere più SIB al sistema. Ogni KP vede tutto il contenuto informativo dello smart space indipendentemente dalla SIB a cui effettivamente si connette.

L'informazione in uno smart space viene memorizzata come grafo RDF, in accordo ad una ontologia definita. Il ruolo dei KP è di interrogare il sistema richiedendo informazioni o effettuando modifiche su di esse utilizzando le primitive previste dal protocollo di comunicazione con la SIB.

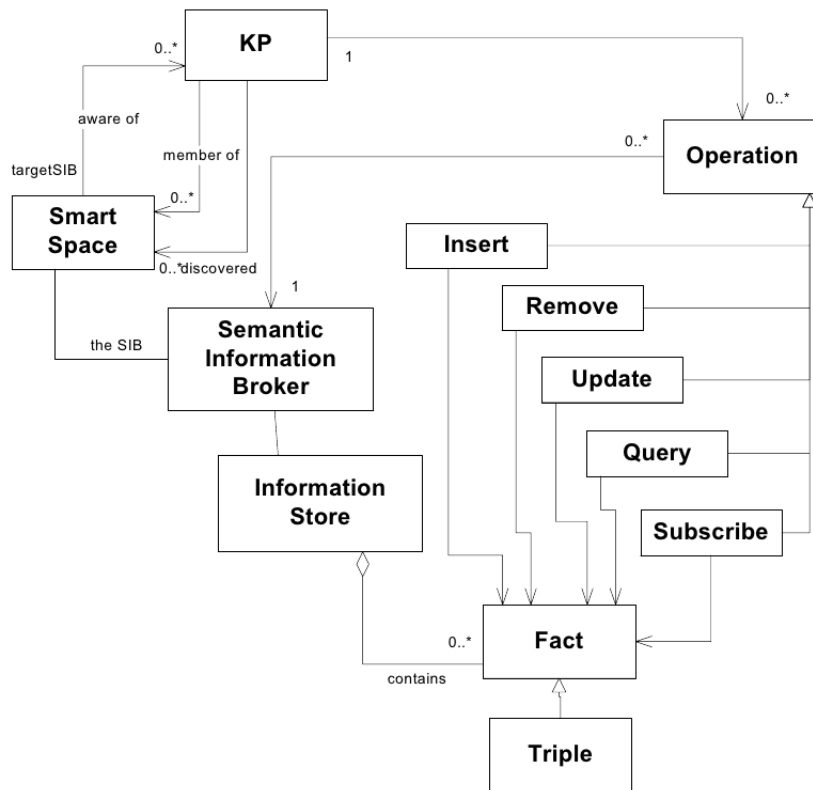


Figura 3.4: Modello di dominio

Protocollo di comunicazione

La comunicazione con la SIB richiede un protocollo ben definito: lo Smart Space Access Protocol (SSAP), basato su messaggi in linguaggio XML. L'utilizzo di questo protocollo permette un accesso universale alla SIB poiché il linguaggio XML è compreso da tutte le piattaforme e dispositivi. Qualsiasi messaggio inviato alla SIB è seguito da un messaggio di risposta dalla SIB, mentre un messaggio inviato dalla SIB non prevede risposta. L'SSAP al momento prevede i seguenti tipi di messaggi:

- **Join** – È il primo messaggio da inviare alla SIB per iniziare la comunicazione e contiene informazioni di chi effettua la richiesta di connessione. Il messaggio di risposta è il rifiuto della connessione o la conferma insieme alle informazioni sulla SIB.

- **Leave** – Chiude la connessione con la SIB: per riaprirla si deve inviare nuovamente il messaggio di Join. La risposta dalla SIB conferma o meno la chiusura della connessione.
- **Insert** – È il messaggio che permette di inserire una tripla nella SIB. La tripla è in formato RDF ed è formata da Soggetto Predicato e Oggetto. La SIB conferma o meno l’inserimento, ma non effettua alcun controllo sulla coerenza dei dati, quindi chi inserisce si deve preoccupare di effettuare un controllo sul SIB prima di inviare un messaggio di Insert.
- **Remove** - È l’operazione opposta alla Insert: viene rimossa una tripla. Anche in questo caso il SIB conferma o meno la rimozione, ma non effettua alcun controllo.
- **Update** – Ha come parametri due triple: la prima viene rimossa e la seconda viene aggiunta. La SIB conferma o meno. Da notare che quello che si ottiene con l’Update si può ottenere anche con una Remove e una Insert, ma l’Update è un’operazione atomica.
- **Query** – È il messaggio per leggere informazioni dal SIB. Sono supportate le query RDF, che servono per cercare triple nel grafo RDIF, alle quali il SIB risponde con tutte le triple trovate, e le Wilbur query (WQL) con le quali si può cercare un intero percorso nel grafo specificandone i nodi.
- **Subscribe** – Serve per ricevere una notifica dal SIB quando viene manipolata la tripla inviata come parametro della Subscribe. Il SIB conferma o meno la Subscribe e invia una notifica appena viene effettuata una Insert, Remove o Update che riguarda quella tripla.
- **Unsubscribe** – Cancella la sottoscrizione, interrompendo le notifiche.

Il formato di questi messaggi di comunicazione tra KP e SIB è definito dal modello Resource Description Framework (RDF). Come abbiamo visto,

l'unità base per rappresentare un'informazione è lo statement, ovvero una tripla del tipo Soggetto – Predicato – Oggetto, dove il soggetto è una risorsa, il predicato è una proprietà e l'oggetto è un valore. Per i campi “Soggetto” e “Oggetto” devono essere indicati anche i corrispondenti tipi, che possono essere due:

- “uri”: specifica che quella risorsa è valore è unico e la identifica attraverso un numero di identificazione.
- “literal”: specifica che l'oggetto è un valore letterale e non univoco.

Gli elementi che costituiscono una tripla sono da ricercarsi nell'ontologia definita per lo specifico smart space. La corrispondenza tra gli elementi dell'ontologia e la loro codifica con il formato RDF è quella indicata in figura 3.5.



Figura 3.5: Mapping SSAP / RDF - Ontologia

Knowledge Processor KP

Il KP Può essere paragonato ad un interprete che traduce dal linguaggio specifico di un dispositivo al linguaggio parlato dal SIB. Il software di ogni dispositivo comprende una parte di gestione del dispositivo stesso e un proprio KP. In realtà visto che non si può realmente distinguere la parte di gestione del dispositivo, dalla parte di comunicazione con il SIB, per KP si

intende tutto il programma presente sul dispositivo. Vista la complessità delle informazioni da gestire, i KP sono generalmente scritti in linguaggio ad alto livello come il Python o il C#.

Un KP viene realizzato utilizzando delle apposite librerie, le KPI (KP Interface), che semplificano notevolmente la comunicazione con il SIB, in particolare si occupano di creare messaggi secondo il complesso protocollo di Smart-M3 (SSAP). Nella release di Smart-M3 sono disponibili KPI nei linguaggi GLib/C, Qt/C++, Python, C# e Java. Oltre alle KPI, un KP può utilizzare le Ontology API, delle librerie contenenti l'ontologia che servono allo sviluppatore per accedere più facilmente al grafo RDF.

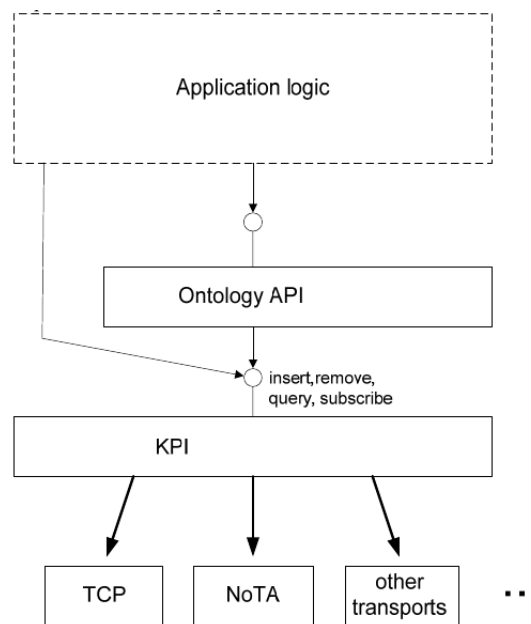


Figura 3.6: Struttura di un KP

Un KP può essere un KP cosiddetto Consumer se viene utilizzato per leggere informazioni dalla SIB ma non inviarne; viceversa può essere un KP

Producer (ad esempio nel caso del software di un sensore ambientale); o infine un Aggregator se svolge entrambe le funzioni.

La registrazione del KP avviene tramite l'invio in successione di messaggi contenenti triple di elementi (Figura 3.5) che ne descrivono le caratteristiche e la loro registrazione all'interno del SIB. Anche l'inserimento o la richiesta di informazioni da parte di qualsiasi KP avviene sempre con la trasmissione di triple, è il tipo di messaggio che discrimina l'operazione che il SIB deve eseguire una volta ricevuto il comando. Il KP implementato nell'applicazione deve così poter trasmettere e ricevere informazioni tramite una rete Wi-Fi e/o cablata e/o disposta ad-hoc.

Semantic Information Broker SIB

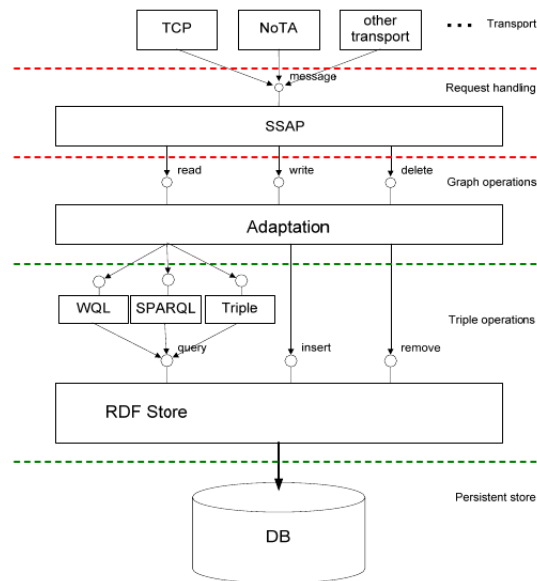


Figura 3.7: Semantic Information Broker

La figura 3.7 rappresenta la struttura di un generico Semantic Information Broker (SIB). L'architettura interna consiste di cinque livelli: Transport

layer, Operation handling layer, Graph operations layer, Triple operations layer e Persistent storage layer.

Il livello di trasporto consiste in uno o più processi che rendono disponibile il servizio SIB ad architetture e reti di tipo diverso. Il generico KP invia un messaggio con protocollo SSAP al SIB. Ogni messaggio viene gestito in modo autonomo, tramite un thread separato per richiesta. La richiesta viene quindi passata al livello successivo, il Graph Operations che si occupa di eseguire le operazioni richieste sul grafo RDF, nell'RDF Store. In questo livello viene utilizzato un solo thread per eseguire le operazioni sul grafo, quindi le numerose richieste dal Request Handler vengono accettate in modo asincrono e una per volta secondo un certo scheduling.

Il livello Triple Operations è l'ultimo livello del SIB e comunica direttamente con il DataBase SQLite che contiene le informazioni. Questo livello è implementato usando il Piglet RDF Store, uno Store progettato ad hoc per Smart M3 che contiene strutture per il supporto di query di tipo Wilbur, SPARQL e RDF-Triple.

Applicazioni per Smart Spaces

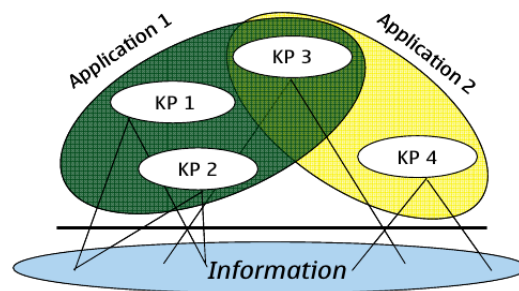


Figura 3.8: Applicazioni per Smart Space

La nozione di applicazione in uno smart space differisce radicalmente dal concetto di applicazione tradizionale. Invece di avere in esecuzione una applicazione monolitica su una singola schermata, le applicazioni per smart

spaces sono meglio riconducibili a scenari che possono essere eseguiti per raggiungere gli obiettivi degli utenti.

Tale scenario emerge dalle azioni svolte dai KP basate sulle informazioni residenti nello smart space. Uno scenario può dunque essere transitorio: cambia quando un KP accede o lascia lo smart space o quando certi servizi diventano disponibili o non disponibili. Obiettivo di Smart M3 è combinare in modo semplice molteplici scenari. Ciò si ottiene attraverso un *loose coupling* tra i KP, in quanto essi comunicano soltanto richiedendo informazioni o modifiche nello smart space. In questo modo, ogni effetto che l'apparire o lo scomparire di un KP può avere sul resto del sistema è limitato all'informazione visibile nello smart space.

Un KP ha conoscenza di un set non esclusivo di informazioni, come illustrato in figura 3.8. La sovrapposizione (o meglio l'intersezione) di questo set di informazioni note al KP è una precodizione fondamentale per raggiungere l'interoperabilità tra KP e fare in modo che i diversi KP vedano reciprocamente le azioni svolte sul set informativo.

Capitolo 4

Arces Smart Space - CHIRON

Lo scenario proposto è realmente disponibile nel laboratorio Arces MARS e fa riferimento al suddetto progetto SOFIA ed al progetto CHIRON [42].

CHIRON è un progetto di ricerca Europeo parzialmente finanziato dall’iniziativa Europea ARTEMIS e dagli Stati aderenti nell’ambito del bando ARTEMIS 2009 – Sottoprogramma ASP2: “Gestione della salute centrata sulla persona”.

Il progetto CHIRON sviluppa e realizza una piattaforma tecnologica per una gestione della salute che sia efficace, copra il ciclo completo di cura e metta il cittadino come “persona” al suo centro. Con questo progetto è condiviso lo scenario e l’obiettivo di realizzare un sistema per il controllo della salute personalizzato; la realizzazione di questo obiettivo riguarda chiaramente più domini coinvolgendo più attori tra cui il paziente, i medici e la comunità scientifica. L’obiettivo di CHIRON è di progettare una architettura per realizzare effettivamente un sistema per il person-centric health management in grado di seguire il paziente in tutte le sue cure. La principale sfida è quindi integrare le informazioni più recenti del paziente con gli esami e i dati storici e trasformare questa informazione in un utile supporto per facilitare la presa di decisioni.

La architettura è basata su Smart-M3 ed è descritta in Figura 4.1.

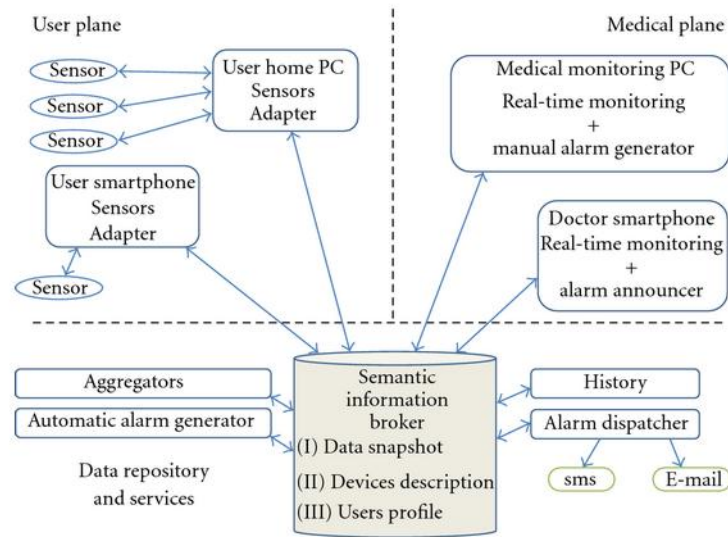


Figura 4.1: Architettura del sistema Smart M3

Nella architettura di CHIRON vengono definiti i tre seguenti layer:

- Lo user plane riguarda le interazioni da e con il paziente (monitoraggio e feedback locale);
- Il medical plane riguarda le interazioni da e con i medici (valutazione dei dati clinici, diagnosi, pianificazione e esecuzione di un trattamento, feedback al paziente);
- Lo statistical plane riguarda le interazioni con i ricercatori medici (controllo e raccolta dati).

L'intero ciclo di cura è basato sulla continua interazione e scambio di informazioni tra questi tre piani.

Nello User plane, dati provenienti da sensori eterogenei sono raccolti da un PC o uno smartphone e inviati alla SIB. I Sensors Adapters mostrati in figura 4.1 sono legacy adapters; possono essere visti come traduttori guidati da un'ontologia consentendo lo scambio di informazione tra lo SS il mondo

legacy; per mondo legacy intendiamo dispositivi e sistemi commerciali già esistenti e presenti sul mercato.

La SIB è il nucleo del sistema; memorizza e condivide non solo i dati prodotti dai sensori ma tutte le informazioni create durante il processo di inizializzazione. Questa implementazione è coerente con il concetto di Alter Ego proposto in CHIRON [42]. Alter Ego è un'entità virtuale dinamica che modella tutte le informazioni rilevanti legate allo stato di salute di un paziente includendo la sua storia, le abitudini, i suoi parametri fisiologici. . . Questa entità deve avere la capacità di evolversi adattandosi a vari domini e condizioni di utilizzo. Una volta generato, questo profilo virtuale è costantemente aggiornato attraverso le informazioni del sistema di monitoraggio; in ogni caso il medico può aggiungere informazioni in qualsiasi momento. Il profilo include informazioni statiche, semi-statiche e dinamiche e su di esso potranno lavorare motori a regole per determinare la presenza di situazioni anomale.

Utilizzando i dati sensoriali collezionati e sfruttando la conoscenza delle relazioni tra le entità interessate, è possibile creare diversi tipi di servizi.

Gli aggregatori sono servizi generici che consumano l'informazione già presente nella SIB arricchendo questa informazione tramite regole di inferenza. La nuova informazione è quindi memorizzata nella SIB e condivisa in modo tale da poter essere riutilizzata da altri servizi e include per esempio indici e parametri.

L'Automatic Alarm Generator è un servizio per la generazione automatica di uno stato di allarme nel profilo dell'utente; questo è generato quando un parametro del paziente esce dal range di sicurezza descritto nel suo profilo.

Il servizio di History colleziona i dati rilevanti in modo da consentire una analisi successiva per scoprire o validare relazioni macroscopiche tra il profilo del paziente, la diagnosi e l'efficacia del trattamento medico.

L'Alarm Dispatcher è un servizio che notifica la presenza di uno stato di allarme al medico attraverso dei mezzi di comunicazione standard cioè tramite SMS o e-mail.

In riferimento al medical plane viene utilizzata la seguente politica. Lo

staff medico può visualizzare tutti i dati dei suoi pazienti attraverso diverse piattaforme (smartphone e PC) in tempo reale. I medici possono visualizzare questi dati e gli specialisti sono notificati in caso di allarme. I medici possono visualizzare e modificare il profilo del paziente e possono creare degli stati di allarme manualmente.

Il sistema necessita di condividere informazione eterogenee da diversi sistemi / dispositivi commerciali. Lo scopo è creare un sistema che permetta allo staff medico di monitorare lo stato di salute di un paziente insieme ai parametri ambientali. Il sistema è finalizzato al monitoraggio fuori dalla struttura ospedaliera per persone soggette a malattie cardiovascolari. Il sistema monitora parametri fisiologici; tra questi la frequenza cardiaca, la frequenza respiratoria, la temperatura della pelle, la pressione arteriosa, il peso, la saturazione dell'ossigeno e l'attività del paziente.

Questi dati prodotti da diversi dispositivi sono raccolti da uno smartphone associato all'utente e da un PC 7.

Il sistema di monitoraggio è il meno invasivo possibile e soddisfa i criteri di usabilità; inoltre l'interazione con i dispositivi e essere il più semplice possibile. Il sistema fornisce inoltre informazioni sui dati ambientali; in particolare sono monitorati temperatura, umidità, concentrazione di polvere nell'ambiente e inquinamento.

I sensori ambientali sono sparsi e fissati negli ambienti in cui l'utente si può muovere. L'informazione di localizzazione è fondamentale in questo sistema e la granularità è limitata al livello di stanza. La localizzazione può essere fornita da più sistemi di localizzazione in modo tale da aumentare la robustezza del sistema. Allo stesso modo diversi sistemi possono monitorare la situazione ambientale per aumentare la ridondanza di dati. Obiettivo di raccolta dei dati ambientali è astrarre un indice di disagio bioclimatico da associare ad ogni ambiente, così da tarare degli allarmi basati su soglie per avvertire il personale medico della presenza di situazioni anomale.

4.1 Open CHIRON Smart Space Ontology

L'implementazione di un'applicazione Smart Space basata sull'approccio proposto richiede che venga prima sviluppata un'ontologia per descrivere il dominio di interesse; richiede poi lo sviluppo di una serie di KP per ottenere il comportamento desiderato.

L'ontologia di riferimento è quella già sviluppata per Smart M3 e presente nel sistema disponibile in Arces MARS. Il linguaggio scelto per l'ontologia è RDF++ serializzato in standard XML/RDF. Il tool utilizzato per lo sviluppo e l'editing dell'ontologia è Protégè, tool conveniente per effettuare modifiche all'ontologia e per i plug-in che mette a disposizione che permettono di avere una visualizzazione grafica dell'ontologia¹.



Figura 4.2: Albero delle classi dell'ontologia

¹The Protégé Ontology Editor and Knowledge Acquisition System, <http://protege.stanford.edu/>

Come mostrato in Figura 4.2, la classe *Data* è utilizzata per descrivere un generico dato prodotto da un dispositivo o sensore. Un'istanza di *Data* può essere associata ad un utente o ad un ambiente e può essere anche prodotto in seguito ad un'elaborazione; un esempio può essere la temperatura media di un ambiente che è monitorato da più sensori di temperatura.

SensorData è una sottoclasse di *Data* utilizzata per rappresentare il dato grezzo ottenuto da un sensore. Per esempio una lettura del sensore di temperatura è un *SensorData* ed è descritto da un valore, da un timestamp ed è collegato a istanze di *MeasurandType* e *UnitOfMeasurement*. *UnitOfMeasurement* contiene le istanze delle varie unità di misura usate nel sistema.

La classe *Measurand* descrive i diversi tipi di misurandi. Sviluppando in questo modo il concetto di *Data* si ottiene una soluzione molto flessibile ed estendibile: quando è aggiunto un nuovo sensore che misura una grandezza non ancora descritta basta aggiungere nuove istanze per caratterizzarla.

La classe *IdentificationData* è usata per descrivere l'identificatore necessario per identificare una persona, un dispositivo o un generico oggetto; una istanza di *IdentificationData* può avere un valore e un tipo (per esempio RFID, MACAddress o SemaCode).

La classe *Device* è usata per descrivere un dispositivo generico come un PC o uno smartphone; la sua sottoclasse *SensorPlatform* è stata invece introdotta per descrivere una piattaforma di sensori. La classe *ProtocolType* descrive i protocolli utilizzati da una *SensorPlatform*. *Environment* descrive un ambiente. La classe *Event* ha due sottoclassi *ArtificialEvent* e *NaturalEvent*; è usata per descrivere un evento come può essere per esempio un allarme creato quando un parametro eccede la soglia di sicurezza; *EventType* specifica il tipo di evento.

Infine, la classe *Person* descrive gli utenti (pazienti e medici).

Le proprietà sono mostrate in figura 4.3:

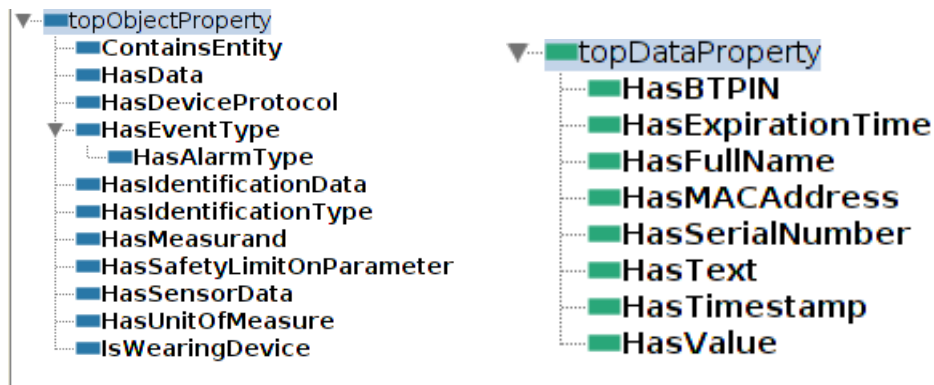


Figura 4.3: Seconda activity visualizzata - il Tab *Patient*

- ContainsEntity è usata per collegare un Environment con una Person o un Device. Questo link viene creato per definire la posizione di una persona o di un dispositivo.
- HasData è usato per collegare una qualsiasi istanza che può avere un Data con il rispettivo Data.
- HasMeasurand collega un Data ad un Measurand; è utile per esempio per effettuare query quando si è interessati solo ai dati con un determinato significato fisico.
- HasSensorData è usato in modo simile a HasData ma collega tra loro SensorPlatform a SensorData.
- HasUnitOfMeasure connette Data a UnitOfMeasure.
- HasEventType è usato per collegare una istanza di Event al rispettivo tipo.
- HasIdentificationData collega una generica istanza ad un IdentificationData.
- HasIdentificationType connette un IdentificationData con IdentificationType.

- HasSafetyLimitOnParameter collega o un Environment o una Person a una istanza di SafetyLimitOnParameter.
- IsWearingDevice è usato per collegare una persona con un dispositivo o un sensore.
- HasDeviceProtocol collega un Device ad un ProtocolType.

Le seguenti sono invece tutte le datatype property:

- HasBTPin associa il pin bluetooth al dispositivo.
- HasExpirationTime è usato per dare un tempo di validità ad un generico dato.
- HasFullName è usata per linkare una Person al suo nome.
- HasText è usato per aggiungere una descrizione testuale a una qualsiasi entità.
- HasTimestamp associa un timestamp ad un Data HasValue associa un valore ad un Data.
- HasValue associa un valore ad un Data.

Guardando lo scopo dell'ontologia e le classi, le proprietà e le istanze che sono state create, l'ontologia permette di rappresentare tutte le informazioni rilevanti nella nostra applicazione. Inoltre l'ontologia permette di rispondere alle domande di competenza e di utilizzare i termini importanti.

4.2 Open CHIRON Smart Space - ambiente fisico

Mostriamo ora l'architettura fisica di Open Chiron Smart Space come resa disponibile dal laboratorio Arces MARS.

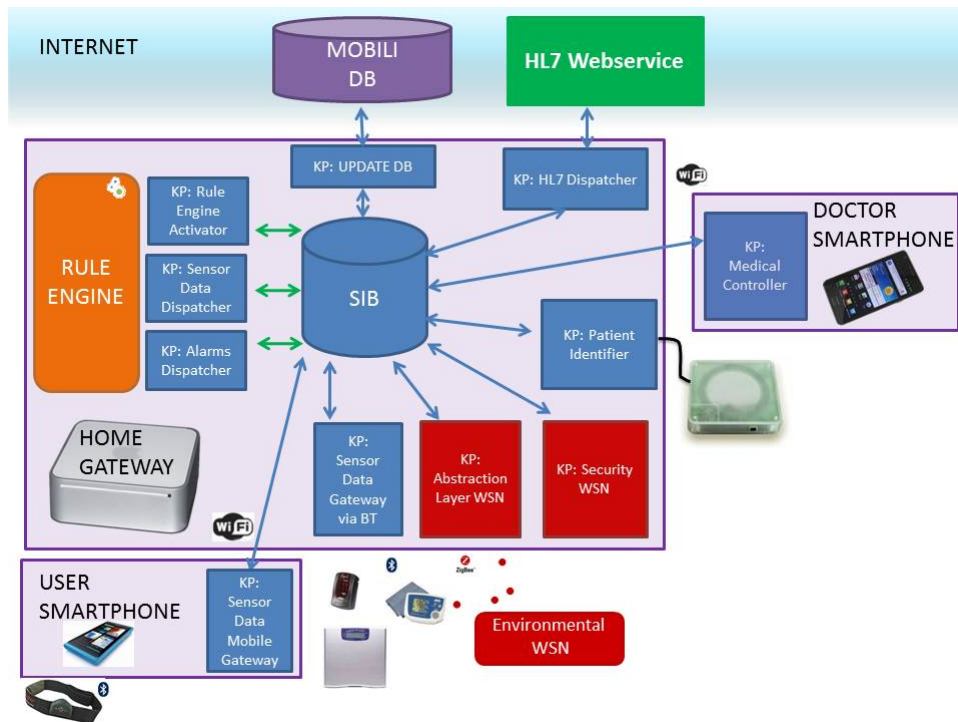


Figura 4.4:

La SIB è ospitata all'interno del Home Gateway domestico, un Apple Mac Mini PC con sistema operativo Ubuntu Linux. Il gateway è connesso con un cavo lan ad un access point collegato ad internet (nel sistema attuale la connessione ad internet è attivata tramite un modem HDSPA). La versione della SIB utilizzata è Smart-M3.B.v0.3.1 disponibile su <http://sourceforge.net/projects/smart-m3/files/>. L'ontologia usata è la Open Chiron Smart Space Ontology.

4.2.1 Architettura Logica

Quando il gateway si accende, il sistema si avvia automaticamente e viene eseguito uno script che avvia la SIB e i KP seguenti:

- Un **Push KP** popola la SIB con le informazioni (attualmente hard coded) riguardo le persone ed i device / sensori;
- il **Patient Identifier KP** viene usato per riconoscere un paziente utilizzando la tecnologia RFID, attraverso un lettore RFID collegato al gateway. Ogni paziente dispone infatti di un tag RFID (il codice RFID è associato al paziente al momento del lancio del KP Push). Quando il paziente pone il suo tag RFID personale sul lettore, il KP riconosce il paziente ed associa allo stesso tutti i dispositivi medici Bluetooth disponibili.
- l'**Abstraction Layer WSN KP** è stato realizzato dal partner WLAB. WLAB offre un ZigBee wireless sensor network in grado di monitorare temperatura ambientale, umidità ambientale, concentrazione di polvere ed inquinamento. L'Abstraction Layer coordina la rete e i dati ricevuti dai sensori mentre un KP Environmental recupera i dati e li condivide sulla SIB.
- La **Security WSN** è stata realizzata per modificare i parametri di sicurezza della WSN; questi parametri possono essere configurati dal Medical Controller KP ed il Security WSN KP è incaricato di modificare queste informazioni. Attualmente tale è ancora in fase di sviluppo e non supporta ancora l'integrazione con WSN.
- Il **Sensor Data Gateway** attraverso il KP Bluetooth gestisce tutti i sensori che comunicano con protocollo bluetooth. Dopo la registrazione con il suo tag RFID, il paziente può semplicemente selezionare un dispositivo e prendere la misurazione senza l'interazione con nessuna interfaccia utente. Questo KP rileva lo strumento, interroga la SIB per ottenere l'instrument profile e l'identificativo del paziente corrente, poi aggiorna i nuovi valori nella SIB ed i dati associati al paziente.
- il **KP Dispatcher HL7** è responsabile di inviare i dati fisiologici e ambientali relativi all'utente in un formato HL7 compliant, all'HL7 service

sviluppato da UNIGE. il KP può spedire automaticamente ogni singola misurazione al web service oppure può spedire un completo CDA (Clinical Document Architecture) alla richiesta del medico. Questa KP può essere configurato dal Medical Controller KP

- il **Update DB KP** è molto simile al HL7 Dispatcher KP: i dati relativi vengono inviati ad un database esterno sviluppato da Mobili.
- il **Rule Engine** (motore a regole) è stato realizzato dalla Orangee all'interno del WP3 e integrato con lo smart space con tre KP. Il rule engine può essere configurato dal medical controller KP (grazie al RuleEngineController KP) ed è sottoscritto a tutte le informazioni pertinenti; in questo modo ogni nuovo dato alimenta automaticamente la rule engine che esegue le regole (SensorDataDispatcher KP). Quando una regola non è stata verificata, un feedback viene condiviso nella SIB (AlarmsDispatcher KP).

Tutti i KP finora definiti sono eseguiti a partire dal gateway e fanno riferimento a dispositivi fisici realmente presenti nel laboratorio Arces Mars.

Per quanto riguarda la parte “mobile” sono stati sviluppati due KP:

- il **Sensor Data Mobile Gateway KP** è usato dal paziente per associare un sensore indossabile e per condividere le informazioni sulla SIB. Questa applicazione connette un dispositivo smartphone allo Zephyr Bioharness (sistema per la rilevazione di segnali e parametri fisiologici sviluppato e prodotto dalla Zephyr Technology [41]) e aggiorna la SIB una volta al secondo con le informazioni del paziente riguardo battito cardiaco, frequenza respiratoria, temperatura della pelle, indice di attività ed angolo di postura.
- il **Medical Controller KP** è una applicazione android usata dal medico per visualizzare e configurare il sistema in tempo reale. L'applicazione si chiama Heart Life Doctor ed è stata sviluppata da me durante il lavoro di tesi. Verrà discussa approfonditamente nel capitolo 7

4.2.2 Architettura Fisica

La configurazione fisica presente nel laboratorio Arces MARS e costituente Open Chiron Smart Space al momento della redazione del presente documento di tesi è la seguente:

- **Smartphone e PC** - L'Home Gateway domestico è un Apple Mac Mini PC con sistema operativo Ubuntu Linux. I dispositivi mobili presenti e su cui girano KP funzionanti sono smartphone Nokia N900, Samsung Galaxy S II, Samsung Nexus S.
- **BioHarness BT** - è un sistema per la rilevazione di segnali e parametri fisiologici sviluppato e prodotto da Zephyr Technology. BioHarness BT è una piattaforma sensoriale indossabile in grado di memorizzare i dati registrati nella sua memoria interna e di trasmetterli via radio tramite il protocollo di comunicazione Bluetooth (standard IEEE 802.15.1) di classe I. Viene utilizzato per misurare i parametri: Heart Rate (Frequenza Cardiaca), Respiration Rate (Frequenza di respiro), Skin Temperature (Temperatura corporea), Posture (Postura), Activity (Attività).



Figura 4.5: BioHarness BT

- *Pulsiossimetro Onyx II 9560* - Sviluppato dall'azienda Nonin Medical, è attualmente il sensore ossimetro più piccolo al mondo. Il pulsiossimetro

(o ossimetro o saturimetro) è un'apparecchiatura medica che permette di misurare la quantità di emoglobina legata nel sangue e la frequenza cardiaca in maniera non invasiva. Non permette di stabilire con quale gas è legata l'emoglobina, ma solo la percentuale di emoglobina legata. Normalmente l'emoglobina lega l'ossigeno, per cui possiamo ottenere una stima della quantità di ossigeno presente nel sangue. Il dispositivo 9560 è in grado di tramettere i dati misurati attraverso il protocollo di comunicazione Bluetooth SSP (Serial Port Profile).



Figura 4.6: Pulsiossimetro Onyx II 9560

- **A&D Weight scale and Sphygmomanometer** - Questi dispositivi consentono ai pazienti di controllare la loro pressione sanguigna e il peso; la trasmissione dei dati avviene sempre tramite protocollo Bluetooth. In particolare lo sfigmomanometro rileva la pressione sistolica, pressione diastolica, la pressione atriosa media e la frequenza cardiaca. Rispettivamente nelle unità di misura mmHg e Bpm. La bilancia rileva il peso in Kg.
- **Eurotech Ambiental Sensors** - riportati in Figura, in grado di monitorare temperatura ambientale, umidità ambientale, concentrazione di polvere ed inquinamento. Anche la comunicazione di questi sensori avviene tramite Bluetooth.



Figura 4.7: A&D Weight scale and Sphygmomanometer

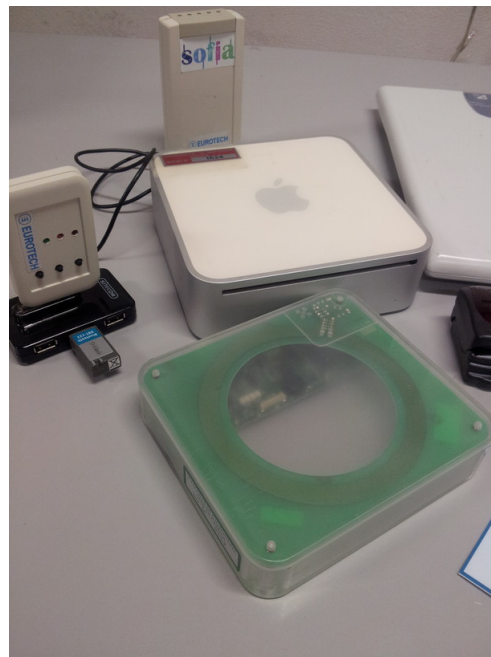


Figura 4.8: Ambiental Sensor, Lettore RFID e Apple Mac Mini

- **Lettore RFID** - mostrato in figura, è gestito dal Patient Identifier KP e collegato all'Apple Mac Mini tramite collegamento diretto.

4.3 Progetto del cyber-physical ecosystem

Come abbiamo detto dunque, un sistema Cyber Fisico è un sistema fisico ed ingegneristico le cui operazioni sono monitorate, coordinate, controllate ed integrate da un core di comunicazione e computing.

Il forte accoppiamento tra la parte *cyber* e la parte *fisica* si manifesta dunque a partire dagli elementi meno “visibili” fino ad avere effetti sulla struttura dell’ambiente e sulla vita della persona umana.

A questo punto del lavoro di tesi abbiamo introdotto gli elementi chiave per la trasformazione di un sistema fisico in un sistema Cyber Fisico.

Il prossimo capitolo tratterà del lavoro di miglioramento delle librerie Java KPI utilizzate dai programmatori del sistema Open Chiron Smart Space, fondamentale per snellire ancora di più il lavoro degli sviluppatori di applicazioni ed elementi del Cyber Physical System.

Infine, nei capitoli seguenti verrà messo a punto l’ultimo elemento di trasformazione del sistema in Cyber Physical System tramite la realizzazione dell’applicazione Heart Life Doctor per il monitoraggio in real time del sistema e dei pazienti.

Capitolo 5

Librerie Java KPI con supporto a SPARQL

Questo capitolo mostra il mio lavoro di supporto ed estensione al sistema fisico e cyber fisico Open Chiron Smart Space.

Il punto di partenza del mio lavoro sono le librerie Java KPICore sviluppate da Daniele Manzaroli in supporto a Open Chiron Smart Space [57]. KPICore è interamente sviluppata in linguaggio Java e offre tutte le primitive e i metodi di accesso, manipolazione, visualizzazione, interrogazione e inserimento di dati all'interno del sistema Open Chiron Smart Space.

Il mio lavoro di supporto è l'estensione delle librerie KPICore con l'inserimento del supporto alle query di tipo SPARQL. La possibilità di effettuare query di tipo SPARQL, come vedremo, migliora le prestazioni di risposta del sistema e snellisce il lavoro dei programmatori nello sviluppo di nuovi KP e componenti del sistema.

Verranno introdotti ora gli elementi principali del linguaggio SPARQL e spiegate infine le modifiche implementative al sistema.

5.1 Java KPICore

Le librerie Java KPICore sono librerie di supporto al sistema Open Chiron Smart Space, sviluppate interamente in linguaggio Java e disponibili per il download all'indirizzo http://sourceforge.net/projects/smart-m3/files/Smart-M3_B_v0.3.1-alpha/.

Attualmente, le librerie che implementano il protocollo SSAP, ovvero che consentono ai KP di comunicare con la SIB, sono al momento disponibili nei seguenti linguaggi: C#, Java, Python, Php, C e Prolog.

Lo scopo della libreria è fornire un set di classi e primitive Java per l'accesso alla SIB offerta dalla piattaforma Smart-M3 attraverso il protocollo SSAP (Smart Space Access Protocol). Queste librerie consentono allo sviluppatore di astrarsi dalle interfacce della SIB per lavorare direttamente sulle informazioni a livello di triple su cui operare. Le librerie semplificano notevolmente la comunicazione con il SIB, in particolare si occupano di creare messaggi secondo il complesso protocollo di Smart-M3 (SSAP) chiamando una sola funzione.

La libreria Java KPICore contiene le seguenti classi e interfacce:

- **Interfacce**

- **iKPIC** - Interfaccia che definisce i metodi del protocollo SSAP.
- **iKPIC_subscribeHandler** - Interfaccia che definisce il metodo di gestione dei messaggi di evento da parte della SIB (è un Handler)

- **Classi**

- **ArcesServiceRegistry** - Classe che offre i metodi di inserimento di un nuovo servizio all'interno della SIB, tramite l'inserimento o la rimozione dell'url associata al servizio.
- **KP_GUI** - Classe che permette di svolgere tutte le operazioni consentite dalle librerie tramite una comoda interfaccia grafica.

- **KPICore** - Implementa le interfacce iKPIC e iKPIC_subscribeHandler, e contiene tutti i metodi previsti dal protocollo SSAP.
- **KPIEventListener** - Sotto-classe di KPICore, offre i metodi per effettuare le operazioni di *subscribe* (sottoscrizioni).
- **SSAP_XMLTools** - Classe cuore delle librerie, insieme alla classe *KPICore* offre metodi di implementazione del protocollo SSAP nonché una serie di metodi di accesso e parsing dei dati per una gestione ottimale delle informazioni provenienti / dirette alla SIB.
- **SSAP_sparql_response** - Classe che offre la rappresentazione del messaggio SSAP in risposta a query di tipo SPARQL operate sulla SIB. Offre metodi per estrarre e manipolare i dati restituiti dalla SIB e per incapsulare il messaggio di query stesso.

Funzioni offerte

Le librerie Java KPI sono di facile utilizzo, molto intuitivo se si dispone delle basi di programmazione Java. Implementano interamente il protocollo SSAP previsto per comunicare con la SIB e offrono una molteplicità di metodi di parsing dei dati e di monitoring davvero utili al programmatore.

La comunicazione tra il Semantic Information Broker (SIB) e i diversi software si basa sul protocollo Smart Space Access Protocol (SSAP) basato a sua volta su messaggi XML, che non prevede fasi di negoziazione complesse tra le uniche due parti coinvolte (la SIB e i singoli KP). SSAP permette di collegarsi o abbandonare il sistema (join e leave) e gestire cinque tipi di transizioni (insert, remove, update, subscription e query).

Fra le funzionalità di maggior rilievo compaiono:

- Inserimento di triple nella SIB, fornendo gli elementi della tripla come stringhe separate o come Vector di String;
- Procedure join, leave, update, remove;

- La possibilità di effettuare query di tipo RDF specificando i parametri oggetto della ricerca;
- Il meccanismo delle sottoscrizioni con i metodi `subscribe` e `unsubscribe`.

Per recuperare informazioni il protocollo SSAP mette a disposizione la transazione query. Il sistema supporta due tipi di approcci: `wilbur query language (WQL)` e `query RDF-M3`. Il `Wilbur query language` mette a disposizione diversi tipi di query. Attraverso di essi si possono ricavare triple effettuando una navigazione all'interno del grafo RDF, specificando un nodo di partenza e un percorso lungo il quale navigare il grafo RDF. La query `RDF-M3` permette di recuperare informazioni sfruttando la struttura degli `statement RDF`. Attraverso l'utilizzo dell'URI "any" (<http://www.nokia.com/NRC/M3/sib#any>) è possibile recuperare tutte le triple RDF che fanno matching con gli `statements` specificati nella query. E' anche possibile specificare più `statement RDF` in una query e in questo caso il risultato ottenuto sarà l'unione dei singoli risultati.

Altra transazione importante è la `subscription`: permette a un KP di chiedere che gli venga notificato il cambiamento del valore di un dato di particolare interesse. Il meccanismo attraverso il quale si può specificare l'informazione interessata è simile a quello utilizzato dalle query `RDF-M3`: specificando per esempio gli URI di due delle tre componenti (soggetto, predicato o oggetto), e utilizzando l'URI "any" per la terza, è possibile ricevere notifiche sugli aggiornamenti del valore della terza componente.

Per un elenco completo delle funzionalità, con relativi esempi e descrizioni d'uso dettagliato rimandiamo alla documentazione Java KPI disponibile all'indirizzo http://sourceforge.net/projects/smart-m3/files/Smart-M3_B_v0.3.1-alpha/.

Esempio d'uso

Vediamo ora un esempio di utilizzo delle librerie per effettuare un'operazione di update di un dato sulla SIB. Per puro fine didattico, il codice di esempio

riportato presenterà alcuni elementi e variabili *hard-coded*.

Si procede innanzitutto con la definizione delle variabili di connessione e la creazione delle istanze di KPICore e SSAP_XMLTools:

```
//...
String HOST = "127.0.0.1";
String PORT = "10010";
String SSNM = "X";

KPICore kp = new KPICore(HOST,Integer.parseInt(PORT),SSNM);
SSAP_XMLTools xmlTools = new SSAP_XMLTools(null,null,null);

kp.enable_debug_message();
kp.enable_error_message();
```

Alla classe KPICore vengono passati come parametri di creazione i dati di connessione, che da questo momento rimangono memorizzati. Il costruttore dell'oggetto SSAP_XMLTools non ha bisogno dei parametri perchè come vedremo svolge il solo scopo di parser XML.

Tramite i metodi *enable_debug_message()* e *enable_error_message()* è possibile scegliere se visualizzare in console eventuali messaggi di debug o errore messi a punto dalla libreria stessa.

La prima operazione da svolgere è la *join* alla SIB.

```
String xml = kp.join();
String ack = xmlTools.isJoinConfirmed(xml);
System.out.println("Join confirmed:"+(ack?"YES":"NO")+"\n");
```

Il metodo *join()* fornito dalla classe KPICore effettua l'operazione di JOIN sulla SIB e restituisce una stringa rappresentazione della risposta XML fornita dalla SIB. Il messaggio di join è creato incapsulando le informazioni richieste dal protocollo SSAP come mostrato nel codice:

```
String join_message = "<SSAP_message>"
+"<message_type>REQUEST</message_type>"
+"<transaction_type>JOIN</transaction_type>"
+"<transaction_id>"+ ++transaction_id +"</transaction_id>"
+"<node_id>"+nodeID+"</node_id>"
+"<space_id>"+ SMART_SPACE_NAME +"</space_id>"
+"</SSAP_message>";
```

Il tipo di messaggio è REQUEST, la transazione richiesta è JOIN, in più vengono fornite alcune informazioni necessarie all'identificazione della

connessione quali l'Id della transazione, l'Id del nodo e lo Smart Space Name. La richiesta di join così incapsulata viene inoltrata tramite il metodo della classe KPICore:

```
public String join()
{
    return sendSSAPMsg( this.xmlTools.join() );
} //String join()
```

Una volta incapsulato il messaggio ed inviato alla SIB, si cerca nel messaggio di ritorno l'informazione CONFIRM che attesta l'avvenuta connessione. Come abbiamo visto, è sufficiente una chiamata al metodo *isJoinConfirmed(String xml)* per ottenere un boolean di conferma. Il metodo *isJoinConfirmed(String xml)* è offerto dalla classe SSAP_XMLTools:

```
public boolean isJoinConfirmed(String xml)
{
    String id[]={"transaction_type","message_type"};
    String ref[]={"JOIN","CONFIRM"};
    return autoCheckSibMessage(xml,id,ref)
    && getParameterElement(xml,"name","status").getValue().equals("m3:Success");
}
```

Effettuare l'operazione di UPDATE è altrettanto semplice e si effettua tramite una sola chiamata al metodo *update(String sn,String pn,String on,String sn_type,String on_type, String so, String po, String oo, String so_type, String oo_type)* offerto dalla classe KPICore. L'operazione di UPDATE prevede l'aggiornamento di una tripla esistente nella SIB con un'altra specificata dall'utente. I parametri della funzione sono soggetto, predicato e oggetto della nuova tripla e di quella già esistente.

Nell'esempio che segue si vuole aggiornare le informazioni dell'ambiente Environment_01 ed Environment_02 cambiando l'attributo *ContainsEntity* che attribuisce la presenza di una persona (Person_01) all'uno o all'altro ambiente.

```
//...
xml=kp.update("Environment_02", "ContainsEntity", "Person_01", "uri", "uri",
              "Environment_01", "ContainsEntity", "Person_01", "uri", "uri");
print("Update confirmed: "+(this.xmlTools.isUpdateConfirmed(xml)?"YES":"NO")+"\n");
//...
```

Allo stesso modo è possibile effettuare l'operazione di *leave* dalla SIB, del tutto analoga alla *join*:

```
//...
xml=kp.leave();
ack=xmlTools.isLeaveConfirmed(xml);
System.out.println("Leave confirmed:"+ (ack?"YES":"NO"));
```

5.2 SPARQL

SPARQL (Simple Protocol And RDF Query Language) è il linguaggio di interrogazione per il recupero dei dati espressi in RDF nonché un linguaggio di query RDF e un protocollo di accesso ai dati per il Web Semantico. Rappresenta l'ultimo tassello per l'edificazione del Semantic Web da W3C e il 15 Gennaio del 2008 è stato standardizzato dallo SPARQL Working Group del W3C.

SPARQL consiste in tre specifiche:

- SPARQL Query Language specification [58] che costituisce il nucleo
- SPARQL Query Results Format specification [59] che descrive un formato XML per serializzare il risultato di una query SPARQL
- SPARQL Protocol for RDF specification [60] che utilizza WSDL 2.0 per definire protocolli semplici HTTP e SOAP per fare le query in remoto su basi di dati RDF

Lo SPARQL Protocol for RDF permette a un generico client di interrogare uno o più endpoint SPARQL inviando una richiesta espressa nel linguaggio di interrogazione e ricevendo come risposta il risultato in formato XML. Il protocollo SPARQL è basato su WSDL 2.0 (Web Services Description Language)¹ e la sua specifica descrive sia l'interfaccia astratta, sia i legami di questa verso gli standard attualmente utilizzati sul Web.

¹[30] WSDL 2.0, <http://www.w3.org/TR/wsd120/>

Lo SPARQL Query Language è un linguaggio di interrogazione pensato per il Web, infatti oltre a prevedere la clausola `SELECT` come in SQL, offre anche altri costrutti. E' possibile infatti prevedere che :

- non si conosca a priori lo schema dei dati. Per risolvere questo problema SPARQL ha la clausola `DESCRIBE`, che permette di ottenere una descrizione della risorsa cercata;
- si presenti la necessità di sapere se un certo enunciato o un certo pattern di dati sia presente nella sorgente dati. Per questo motivo SPARQL propone la clausola `ASK`.

In aggiunta SPARQL consiste in un linguaggio di query (ovvero un formato XML) dove i risultati delle query verranno restituiti, e un protocollo per sottoporre una query ad un servizio di elaborazione di query in remoto.

I vantaggi principali di avere un linguaggio di query come SPARQL sono:

- fare le query a dei grafici RDF per ricevere informazioni specifiche;
- fare le query ad un server remoto RDF e ricevere indietro i risultati in streaming;
- eseguire delle query regolarmente in automatico verso insiemi di dati RDF per generare resoconti;
- consentire lo sviluppo di applicazioni ad un livello più alto; le applicazioni possono lavorare con risultati delle query SPARQL, non direttamente con le dichiarazioni RDF.

Nonostante tale vantaggi, rimane un linguaggio incompleto in confronto ad altri linguaggi di interrogazione, ma ciò è dovuto al fatto che è ancora in fase di sviluppo. Una query SPARQL si basa sul graph matching [61] e si compone di quattro parti:

- RDF dataset che permette di scegliere il grafo su cui eseguire la query;

- il graph pattern che applica il graph matching e crea la soluzione parziale della query;
- i modificatori di soluzione che applicati alla soluzione parziale manipolano il risultato;
- la forma del risultato in cui si può scegliere l'output della query

5.2.1 SPARQL Path Expressions

SPARQL adotta la sintassi Turtle (un'estensione di N-Triples), alternativa al tradizionale RDF/XML. Per esprimere le interrogazioni, SPARQL introduce il concetto di *path expression*.

la path expression è l'insieme delle triple necessarie a rispondere all'interrogazione, in cui sostituiamo uno o più identificativi (risorse o letterali) con una variabile, espressa da una parola arbitraria preceduta dal simbolo "?". Inoltre, la path expression rappresenta il pattern dei dati che vogliamo recuperare e quindi i risultati dell'interrogazione saranno tutte e sole le triple RDF che soddisfano la path expression sostituendo alle variabili le risorse o i letterali corrispondenti.

Le query SPARQL si basano in particolare sul triple pattern² che ricalca la configurazione a triple delle asserzioni RDF, fornendo un modello flessibile per la ricerca di corrispondenze. Infatti, soggetto, predicato e oggetto possono essere delle variabili come nell'esempio sottostante che rappresentano una tripla che costituisce una path expression:

```
?titolo cd:autore ?autore
```

Al posto del soggetto e dell'oggetto questo triple pattern prevede due variabili, contrassegnate con "?". Le variabili fungono in un certo senso da incognite dell'interrogazione, cd:autore funge invece da costante: le triple RDF che trovano riscontro nel modello assoceranno i propri termini alle variabili corrispondenti.

²Triple pattern,
#sparqlTriplePatterns

<http://www.w3.org/TR/rdf-sparql-query/>

Le forme di query di SPARQL sono:

- SELECT query
- CONSTRUCT query
- ASK query
- DESCRIBE query

Tra queste, la query SELECT è la forma di query più utilizzata. Molte forme di query SPARQL contengono un insieme di tripple pattern chiamati graph pattern. I triple pattern sono come le triple RDF ad eccezione che ciascuno degli elementi della tripla (soggetto, predicato e oggetto) può essere una variabile, come mostrato nell'esempio seguente:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
<http://danbri.org/foaf.rdf#danbri> foaf:name ?name.
```

In questo esempio il soggetto di questo triple pattern è l'URI di Dan Brickley, il predicato è foaf:name e il componente oggetto della triple pattern è una variabile, identificata dal carattere ? all'inizio della stringa name.

Un esempio invece di graph pattern (ovvero una collezione di triple pattern) è il seguente:

```
{
?who foaf:name ?name.
?who foaf:interest ?interest.
?who foaf:knows ?others.
}
```

Per capire come un graph pattern viene usato per selezionare le risorse da un dato grafo RDF dobbiamo ricordare un punto chiave sul graph pattern: se una variabile compare in diversi triple pattern all'interno del graph pattern, il suo valore in tutti i triple pattern in cui compare deve essere lo stesso. In altre parole ogni risorsa restituita deve essere in grado di sostituire tutte le occorrenze della variabile. Per chiarire meglio questo concetto, ecco una semplice query di selezione SPARQL:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore ?anno
FROM <http://cd.com/listacd.ttl>
WHERE {?titolo cd:autore ?autore.
?titolo cd:anno ?anno
}
```

Nella prima riga viene dichiarato il namespace utilizzato e a differenza della sintassi N3, la parola chiave PREFIX è senza il simbolo ‘’ ed alla fine della dichiarazione non c’è il punto. Se si volesse dichiarare un namespace di default, si potrebbe usare la parola chiave BASE al posto di PREFIX.

Nelle righe successive ci sono altre parole chiave del linguaggio SPARQL:

- SELECT definisce le variabili di ricerca da prendere in considerazione nel risultato;
- FROM specifica il set di dati su cui opererà la query. E’ inoltre possibile utilizzare le clausole FROM NAMED e la parola chiave GRAPH per specificare più set di dati;
- WHERE definisce il criterio di selezione specificando tra parentesi graffe uno o più “triple patterns” separati da punto.

La query precedente ha catturato esclusivamente le triple dotate di tutti e tre i termini richiesti (titolo, autore, anno). È possibile riformulare la query in modo più flessibile, con la possibilità di inserire triple in cui vi sia l’assenza di alcuni termini come mostra l’esempio seguente:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore ?anno
FROM <http://cd.com/listacd.ttl>
WHERE {
?titolo cd:autore ?autore.
OPTIONAL { ?titolo cd:anno ?anno }
}
```

Nell’esempio precedente, il secondo pattern è dichiarato opzionale: l’informazione è inclusa nel risultato solo se disponibile, altrimenti le variabili appariranno prive di valore. Le risorse prive della proprietà ‘anno’ sono mostrate ugualmente e le celle dei valori mancanti sono lasciate vuote. Un

altro modo che ci assicura una certa flessibilità nel reperimento dei dati è il seguente:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore ?anno
FROM <http://cd.com/listacd.ttl>
WHERE {
  { ?titolo cd:autore ?autore. }
  UNION { ?titolo cd:anno ?anno }
}
```

La parola chiave UNION esprime un OR logico: la query non si limita pertanto alle triple che soddisfano entrambi i triple patterns, ma sia quelle che soddisfano solo il primo, sia quelle che soddisfano solo il secondo. È possibile mettere restrizioni sui valori da associare alle variabili:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?anno
FROM <http://cd.com/listacd.ttl>
WHERE {
  ?titolo cd:autore ?autore.
  FILTER ( ?anno > 2000 )
}
```

La restrizione è effettuata tramite l'operatore di confronto '=' e il filtro esclude i termini che non soddisfano la condizione definita tra le parentesi tonde. Gli operatori utilizzabili all'interno di una clausola FILTER sono costituiti da connettivi logici (AND e OR, rappresentati da '&&' e '||'), operazioni di comparazione (come '>', '<', '=', '!=', ecc.), espressioni regolari ed una serie di operatori unari specifici di SPARQL.

Le query CONSTRUCT consentono di restituire il risultato dell'interrogazione sotto forma di grafo RDF, sulla base di un template³. Questo è costituito prendendo ogni soluzione della query nella sequenza della soluzione, sostituendo le variabili nel template del grafo e combinando le triple in un unico grafo RDF tramite l'unione.

L'esempio seguente mostra come viene utilizzato:

```
PREFIX cd: <http://example.org/cd/>
CONSTRUCT { ?titolo ?autore }
FROM <http://cd.com/listacd.ttl>
WHERE {
```

³Template, <http://www.w3.org/TR/rdf-sparql-query/#construct>

```
?titolo cd:autore ?autore.
}
```

5.2.2 Output di una query SPARQL

Il risultato di una query SPARQL è uno SPARQL Results Document, che altro non è che un documento XML che incapsula i risultati della query SPARQL rispettando la struttura ora riportata.

Lo SPARQL Results Document inizia con uno `sparql document element` nel namespace `http://www.w3.org/2005/sparql-results#`, scritto come segue:

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
...
</sparql>
```

Dentro l'elemento *sparql* vi sono due sott-elementi, *head* e un elemento *results* (che può essere *results* o *boolean*) e che deve apparire in quell'ordine.

L'elemento *head* è il primo elemento figlio dell'elemento *sparql*. Perché il risultato della query sparql vincoli le variabili, *head* deve contenere una sequenza di elementi che descrivono il set di *Query Variable Names* nella *Solution Sequence* (qui definita come "risultati della query").

L'ordine in cui compaiono le variabili nella sequenza è l'ordine in cui i nomi delle variabili sono passati come argomento allo statement `SELECT` nella query SPARQL. Se viene usato `SELECT *`, l'ordine delle variabili non è definito.

All'interno dell'elemento *head*, la sequenza ordinata delle variabili scelte sono usate per creare elementi figli come riportato nell'esempio:

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="x"/>
    <variable name="hpage"/>
    <variable name="name"/>
    <variable name="mbox"/>
```

```

    <variable name="blurb"/>
  </head>
  ...
</sparql>

```

In caso di un risultato *boolean*, non c'è bisogno di alcun elemento all'interno dell'elemento *head* e quindi *variable* non deve essere presente.

In alcuni casi, *head* può contenere degli elementi con attributo *href* contenenti un URI che fornisce un link a qualche metadata aggiuntivo per i risultati della query:

```

<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    ...
    <link href="metadata.rdf"/>
  </head>
  ...
</sparql>

```

Il secondo elemento figlio dell'elemento *sparql* deve apparire dopo *head* e può essere sia l'elemento *results* che *boolean*. Tale elemento viene scritto anche se il risultato della query è vuoto.

Per ogni *Query Solution* all'interno dei query results, viene creato un elemento figlio di *results* che si chiama *result* ed è aggiunto al documento in questo modo:

```

<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  ... head ...

  <results>
    <result>...
  </result>
    <result>...
  </result>
    ...
  </results>
</sparql>

```

Ogni elemento *result* corrisponde ad una *Query Solution* e contiene anch'esso degli elementi figli (in ordine sparso) per ogni *Query Variable* che appare nella soluzione. Esso viene usato per registrare in che modo le variabili della query si legano ai termini RDF.

Ogni *binding* all'interno di una soluzione è figlio di *result* e ha come variabile il valore dell'attributo *name*. Nel caso di due variabili ed una *hpage* il documento risultante sarà:

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="x"/>
    <variable name="hpage"/>
  </head>

  <results>
    <result>
      <binding name="x"> ... </binding>
      <binding name="hpage"> ... </binding>
    </result>

    <result>
      <binding name="x"> ... </binding>
      <binding name="hpage"> ... </binding>
    </result>
    ...
  </results>
</sparql>
```

Il valore di un binding di una variabile, che è un RDF Term, è incluso come contenuto del *binding* come segue:

- RDF URI Reference U: $\langle binding \rangle \langle uri \rangle U \langle /uri \rangle \langle /binding \rangle$
- RDF Literal S: $\langle binding \rangle \langle literal \rangle S \langle /literal \rangle \langle /binding \rangle$
- RDF Literal S with language L: $\langle binding \rangle \langle literalxml : lang = "L" \rangle S \langle /literal \rangle \langle /binding \rangle$
- RDF Typed Literal S with datatype URI D: $\langle binding \rangle \langle literaldatatype = "D" \rangle S \langle /literal \rangle \langle /binding \rangle$
- Blank Node label I: $\langle binding \rangle \langle bnode \rangle I \langle /bnode \rangle \langle /binding \rangle$

In caso una variabile risulti *unbound*, cioè non vi fossero risultati ad essa associabili, nessun elemento *binding* è incluso nell'elemento *result*.

Un esempio di una Query Solution nel formato appena spiegato è la seguente:

```

<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">

  <head>
    <variable name="x"/>
    <variable name="hpage"/>
    <variable name="name"/>
    <variable name="age"/>
    <variable name="mbox"/>
    <variable name="friend"/>
  </head>

  <results>

    <result>
      <binding name="x">
        <bnode>r2</bnode>
      </binding>
      <binding name="hpage">
        <uri>http://work.example.org/bob/</uri>
      </binding>
      <binding name="name">
        <literal xml:lang="en">Bob</literal>
      </binding>
      <binding name="age">
        <literal datatype="http://www.w3.org/2001/XMLSchema#integer">30</literal>
      </binding>
      <binding name="mbox">
        <uri>mailto:bob@work.example.org</uri>
      </binding>
    </result>

    ...
  </results>
</sparql>

```

Un risultato di tipo *boolean* compare come contenuto di un elemento *boolean* figlio dell'elemento *sparql* direttamente dopo l'elemento *head*, e può contenere sia il valore “vero” che “falso” come segue:

```

<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  ... head ...

  <boolean>true</boolean>
</sparql>

```

Ulteriori informazioni sul documento SPARQL Results Document sono disponibili all'indirizzo <http://www.w3.org/TR/rdf-sparql-XMLres/>

5.3 Implementazione del supporto SPARQL per le librerie Java KPI

L'introduzione al supporto di query SPARQL si rende fondamentale per il completamento del sistema Open Chiron Smart Space. Le librerie Java KPI, e anche le altre implementazioni C#, Python, Php, C e Prolog consentono uno sviluppo snello ed efficace per i programmatori. Con l'introduzione del supporto a SPARQL vedremo che sarà possibile manipolare una grande quantità di dati tramite strutture adeguate, ottenendo quindi anche un notevole miglioramento delle prestazioni.

Le nuove funzionalità potranno essere utilizzate per poter navigare il grafo RDF contenuto nella SIB di uno Smart Environment con tutte le funzionalità fornite da SPARQL 1.0. Queste funzionalità sono state testate e sono già disponibili pubblicamente su Sourceforge in un pacchetto completo che include anche i sorgenti per installare la SIB sul proprio sistema, reperibili all'indirizzo http://sourceforge.net/projects/smart-m3/files/Smart-M3_B_v0.3.1-alpha/.

5.3.1 Struttura dati per la memorizzazione dei risultati della query SPARQL

La principale novità all'interno del codice esistente delle Java KPI è l'introduzione di una classe *SSAP_sparql_response* che rappresenta proprio la risposta alla query SPARQL e fornisce i metodi di memorizzazione e parsing dei risultati.

La soluzione implementativa della nuova classe e delle altre features è coerente con l'intero progetto già svolto da Daniele Manzaroli. Per questo motivo mi sono attenuta all'utilizzo di strutture dati come *java.util.Vector* anziché implementare soluzioni che usassero *ArrayList* o strutture similari, note per essere più performanti e specifiche per il trattamento di una molteplicità di dati.

Lo Sparql Results Document viene rappresentato dalla classe `SSAP_sparql_response`. Tutti i risultati del Results Document sono salvati in una struttura dati del tipo `Vector < Vector < String [] >>`. Ogni singolo risultato è un Vector di array di String. Ogni elemento di questo vettore rappresenta il dettaglio dell'elemento *binding* del Results Document ed è composto dai seguenti campi:

- name - il binding name dell'elemento *result*
- category
 - URI - per un binding element della forma `< binding >< uri > U < /uri >< /binding >`
 - LITERAL - per un binding name di uno dei seguenti tipi:
 - `<binding><literal> S </literal></binding>` (il cui *type* è SIMPLE)
 - `<binding><literal datatype = D > S </literal></binding>` (il cui *type* è WDATT, ed è aggiunto il valore dell'attributo *datatype*)
 - `<binding><literal xml:lang=L > S </literal></binding>` (il cui *type* è WLANG, ed è aggiunto il valore dell'attributo *xml:lang*)
 - BNODE - per un binding name della forma `<binding><bnode> I </bnode></binding>`
 - UNBOUND - per le variabili slegate (si ricordi che nel result element non compare alcun binding element per questo tipo di risultato)
- value - il valore dell'elemento *result*
- type - il tipo si distingue in SIMPLE, WDATT, e in caso di viene aggiunto il valore

Per creare la struttura dati il Results Document viene parsato da un XML Parser, che ne restituisce la rappresentazione sotto forma di *org.jdom.Document*.

```
private static Document loadXMLFromString(String xml) throws Exception
{
    SAXBuilder builder = new SAXBuilder();
    Document doc = builder.build(new ByteArrayInputStream(xml.getBytes()));
    return doc;
}
```

Tale documento viene poi scansionato elemento per elemento, valore per valore.

```
//...
Document sparql_response_document=loadXMLFromString(sparql_response);

// GET ROOT ELEMENT + NAMESPACE
Element root = sparql_response_document.getRootElement();
Namespace ns;

// GET ROOT CHILDREN (with name PARAMETER)
List <Element> rootchildrenList = root.getChildren("parameter");
Iterator <Element> rootIter = rootchildrenList.iterator();
```

A questo punto il meccanismo per recuperare l'intera struttura dati è abbastanza semplice e fatto di molteplici iterazioni. Bisogna però prestare molta attenzione a non confondersi nella scansione dell'albero del documento jdom:

```
while(rootIter.hasNext())
{
    Element rootchildren = rootIter.next();

    // GET ROOT CHILDREN (with name PARAMETER and value RESULTS)
    if(rootchildren.getAttributeValue("name").equals("results"))
    {
        List<Element> sparqlList = rootchildren.getChildren();
        Iterator <Element> parameterIter = sparqlList.iterator();

        while(parameterIter.hasNext())
        {
            Element parameterchildren = parameterIter.next();
            ns=parameterchildren.getNamespace();

            //...

            List<Element> head= parameterchildren.getChildren();
            Iterator <Element> sparqlIter = head.iterator();

            while(sparqlIter.hasNext())
            {
                Element sparqlchildren = sparqlIter.next();
                ns=sparqlchildren.getNamespace();

                //...
            }
        }
    }
}
```

Solo a questo punto siamo all'interno dell'elemento *head* del Results Document:

```
// INTO <HEAD>
if(sparqlchildren.getName().equalsIgnoreCase("head"))
{
    List <Element> headList = sparqlchildren.getChildren();
    Iterator <Element> headListIter = headList.iterator();

    while(headListIter.hasNext())
    {
        Element headchildren = headListIter.next();
        List<Attribute> attribute = headchildren.getAttributes();

        for (int i =0;i<attribute.size();i++)
        {
            Attribute att = (Attribute) attribute.get(i);

            // IT'S A VARIABLE!
            if(att.getName().equals("name")){
                this.var_names.add(att.getValue());
            }

            // IT'S A LINK!
            if(att.getName().equals("href")){
                this.has_links = true;
                this.link_hrefs.add(att.getValue());
            }
        }
    }
}
} //headListIter
} //endif head
//...
```

A questo punto è possibile salvare nella struttura dati tutto ciò che corrisponde a *name* e *href*, cioè i nomi delle variabili (per associarli poi ai risultati) e i link *href*.

All'interno dell'elemento *results* avviene il grosso del lavoro di parsing:

```
//...
// INTO <RESULTS>
if(sparqlchildren.getName().equalsIgnoreCase("results"))
{
    List <Element> resultsList = sparqlchildren.getChildren();
    Iterator <Element> resultsListIter = resultsList.iterator();

    // INTO <RESULT>. it iterates the result(s)
    while(resultsListIter.hasNext())
    {
        //FLAG HAS RESULTS
        this.has_results = true;

        Element result = resultsListIter.next();
        List <Element> bindingList = result.getChildren();
        Iterator<Element> bindingListIter = bindingList.iterator();
```

```
// THE SINGLE RESULT STRUCTURE, it is going to be populated
Vector<String[]> single_result = new Vector<String[]>();
//...
```

In questa porzione di codice viene impostata a *true* la variabile *has_results*, ad indicare che il Results Document non è vuoto. Inoltre, viene predisposta la struttura dati per la memorizzazione del singolo risultato.

A questo punto si entra all'interno del campo *result* e si iterano i singoli *bindings*. Viene adoperata una struttura dinamica per la memorizzazione dei risultati, da trasferire poi nella struttura statica *String[]* che non è possibile allocare al momento dato che non si conosce il numero totale dei risultati.

```
//...
// INTO <RESULT> it iterates the bindings
while(bindingListIter.hasNext())
{
    Vector<String> cell_temp = new Vector<String>();
    Element binding = bindingListIter.next();
    List attribute = binding.getAttributes();

    for (int i =0;i<attribute.size();i++)
    {
        Attribute att = (Attribute) attribute.get(i);

        // IT'S A BINDING
        if(att.getName().equals("name")){
            cell_temp.add(att.getValue());
        }
    }
}
//...
```

Ora, all'interno di un singolo *binding* element, si iterano tutti i componenti di una singola “cella” e si cercano gli attributi “literal”, “bnode”, “uri” e “xml:lang” o “datatype”:

```
//..
while (bindindChildrenIter.hasNext())
{
    Element bindingchild = bindindChildrenIter.next();

    if(bindingchild.getName().equals("uri"))
    {
        cell_temp.add(bindingchild.getName());
        cell_temp.add(bindingchild.getValue());
    }
    else if(bindingchild.getName().equals("bnode"))
```

```

{
    cell_temp.add(bindingchild.getName());
    cell_temp.add(bindingchild.getValue());
}
else if(bindingchild.getName().equals("literal"))
{
    cell_temp.add(bindingchild.getName());
    cell_temp.add(bindingchild.getValue());

    List lit_attribute = bindingchild.getAttributes();
    for (int i =0;i<lit_attribute.size();i++)
    {
        Attribute att = (Attribute) lit_attribute.get(i);

        if(att.getQualifiedName().equals("xml:lang")){
            cell_temp.add(att.getQualifiedName()+"=\""+att.getValue()+"\"");
        }
        else if(att.getQualifiedName().equals("datatype")){
            cell_temp.add(att.getQualifiedName()+"=\""+att.getValue()+"\"");
        }

        //...
    }
}
}
//...

```

Ora è possibile riempire la struttura `String[]` con i valori trovati:

```

String[] single_cell = null;
if(cell_temp.size()==3)
{
    single_cell= new String[3];
    single_cell[VARNAME]=cell_temp.elementAt(VARNAME);
    single_cell[CATEGORY]=cell_temp.elementAt(CATEGORY);
    single_cell[VALUE]=cell_temp.elementAt(VALUE);
}
else if(cell_temp.size()==4)
{
    single_cell= new String[4];
    single_cell[VARNAME]=cell_temp.elementAt(VARNAME);
    single_cell[CATEGORY]=cell_temp.elementAt(CATEGORY);
    single_cell[VALUE]=cell_temp.elementAt(VALUE);
    single_cell[TYPE]=cell_temp.elementAt(TYPE);
}
}
//...

```

Ed infine si popolano le strutture `Vector` ; `Vector` ; `String []` *ll* appropriate:

```

//...
//ADDING THE CELL TO SINGLE_RESULT

```

```

        single_result.add(single_cell);
    }//bindingListIter

    //ADDING THE SINGLE_RESULT TO RESULTS
    this.sparql_response_results.add(single_result);
} //resultsListIter
} //endif results
//...

```

Similarmente per la parte relativa ai risultati di tipo boolean:

```

//...
//INTO <BOOLEAN>
else if(sparqlchildren.getName().equalsIgnoreCase("boolean"))
{
    this.has_booleans = true;
    this.booleans.add(sparqlchildren.getValue());
} //endif boolean
//...

```

5.3.2 Metodi di supporto alla manipolazione dei risultati

Dal momento in cui la struttura dati che rappresenta i risultati è memorizzata e disponibile, è possibile procedere all'implementazione di tutta la serie di metodi di accesso e parsing dei dati.

Metodi come *getResults()* per ottenere l'intera struttura dati contenente i risultati sono di immediata implementazione:

```

public Vector<Vector<String[]>> getResults(){
    return this.sparql_response_results;
}

```

Altri metodi simili sono:

- *hasResults()* - indica se la risposta alla query contiene almeno un risultato
- *hasBooleans()* - indica se la risposta alla query contiene almeno un risultato di tipo booleano

- *hasLinks()* - indica se la risposta alla query contiene almeno un risultato di tipo *href*
- *size()* - restituisce l'esatto numero di risultati della query sparql

Altri metodi di interrogazione dei dati sono:

- *getVariablesNames()*, che restituisce la sequenza ordinata dei nomi delle variabili richieste alla query sparql
- *getBooleans()*, che restituisce un `Vector<String>` contenente i risultati di tipo booleano, se presenti
- *getGraph()* che, in caso di query di tipo CONSTRUCT, restituisce il grafo corrispondente
- *getLinksHrefs()* restituisce i link di tipo *href* eventualmente presenti nei risultati
- *getResultsForVar(String varname)*, che restituisce solo i risultati associati alla variabile specificata come parametro
- *isQueryTypeConstruct()* e *isQueryTypeSelect()* rispondono con un booleano che indica il tipo di query sparql
- *printGraph(Element graph)*, in caso di query di tipo CONSTRUCT stampa a video il grafo richiesto

Infine, è anche possibile accedere ai dati tramite i metodi più specifici:

- *getRow(int index)*, che restituisce un `Vector<String[]>` corrispondente alla riga di risultati richiesta tramite l'indice passato come parametro
- *getNextRow()*, che restituisce la riga successiva all'indice *index* usato all'ultima interrogazione
- *getCellCategory(String[] cell)*, che restituisce la CATEGORY di una determinata cella, ovvero di un determinato singolo risultato, qualora si passi l'array di String da cui si vuole estrarlo

- *getCellLiteralTypeName(String[] cell)*, restituisce il TYPE dell'elemento literal (xml:lang o datatype)
- *getCellLiteralTypeValue(String[] cell)*, restituisce il valore dopo la dicitura xml:lang o datatype
- *getCellName(String[] cell)*, restituisce la variabile associata al singolo risultato
- *getCellType(String[] cell)*, restituisce il tipo associato al singolo risultato
- *getCellValue(String[] cell)*, restituisce il valore associato al singolo risultato

5.3.3 Metodi di abilitazione alle query SPARQL

Ora che abbiamo mostrato come memorizzare i risultati di una query SPARQL e quali sono le strutture dati ed i metodi per manipolarli, l'ultimo passaggio a supporto dell'introduzione di SPARQL nelle Java KPI è proprio la possibilità di effettuare una query da indirizzare alla SIB.

Per fare questo è sufficiente aggiungere un metodo alla classe SSAP_XMLTools ed uno alla classe KPICore, del tutto analoghi a quelli già visti per RDF o Wilbur.

```
public String querySPARQL(String string) {
    return sendSSAPMsg( this.xmlTools.querySPARQL(string) );
}
```

```
public String querySPARQL(String query_string)
{
    return
    "<SSAP_message><transaction_type>QUERY</transaction_type>"
    + "<message_type>REQUEST</message_type>"
    + "<transaction_id>"+ ++transaction_id + "</transaction_id>"
    + "<node_id>"+ nodeID + "</node_id>"
    + "<space_id>"+ SMART_SPACE_NAME + "</space_id>"
    + "<parameter name = \"type\">sparql</parameter>"
    + "<parameter name = \"query\">"+ correctEntityReferences(query_string) + ""
    + "</parameter></SSAP_message>";
}
```



```
}
```

Il supporto alle query SPARQL introduce la possibilità di manipolare una grande quantità di dati in modo semplice, veloce e snello. Consente inoltre, proprio per come è stato progettato, di effettuare richieste complesse ed articolate per ottenere la migliore selezione dei dati di cui si ha bisogno.

L'utilizzo delle primitive per query SPARQL è davvero molto semplice ed intuitivo. Le potenzialità offerte verranno evidenziate soprattutto nel Capitolo 7, sezione 7.3.1.

5.3.4 Aggiornamento GUI

```
///
```

5.4 Risultati e sviluppi futuri

Come detto in precedenza, l'implementazione del supporto alle query SPARQL è coerente con l'intero progetto Java KPI e fa quindi riferimento a strutture di memorizzazione quali *java.util.Vector*. Lo sviluppo di una nuova versione della libreria potrebbe prevedere un refactory che parta proprio dalla scelta delle strutture più adeguate allo scopo, siano esse usate per memorizzare triple RDF o risultati di query Wilbur o SPARQL.

In conclusione, i risultati ottenuti sono comunque notevoli in termini di prestazioni e di facilità di utilizzo. Al momento della creazione di un'istanza della classe *SSAP_sparql_response*, vengono caricati in memoria la rappresentazione della risposta SSAP sotto forma di stringa (con un peso computazionale quindi molto basso) e l'intera struttura di risultati (che come abbiamo visto possono essere molti, persino centinaia).

Grazie ai metodi offerti dalla classe *SSAP_sparql_response*, la struttura rappresentativa dei risultati viene restituita per la manipolazione non a run-time: essa viene processata e memorizzata nel momento in cui si ricevono i risultati

della query, per poi rimanere in memoria disponibili all'uso. Questo modello di programmazione è molto attento al tempo di esecuzione e quindi può avere effetti collaterali di grande occupazione di memoria.

Per i test effettuati presso Arces Mars tuttavia, anche l'occupazione di memoria non è risultata un problema e la libreria Java KPI supporta la gestione anche fino a 600 risultati di una sola query SPARQL.

Capitolo 6

Il sistema Android

L'idea di sviluppare un'applicazione per dispositivi mobili per usufruire delle funzionalità del cyberphysical environment di Arces MARS nasce all'interno del laboratorio stesso.

Certamente la soluzione più economica, e la prima ad essere valutata, è stata quella di sviluppare una web app, ossia un sito web ottimizzato per mobile, che abbia lo stesso look-and-feel di una applicazione nativa. Il primo beneficio di un'applicazione mobile nativa, a differenza di un sito web ottimizzato per consultazione mobile, è che l'applicazione nativa può sfruttare maggiormente le funzionalità dell'hardware sottostante.

Secondo le stime di Mashable (<http://mashable.com/>), sviluppare una applicazione di alta qualità per iPhone può costare mediamente 30.000 dollari americani; per piattaforme come BlackBerry e Android, che soffrono di una maggiore frammentazione, i costi possono essere ancora maggiori. Insomma, creare applicazioni native può costare anche 10 volte di più di una web app.

Tuttavia, con le applicazioni native, si possono offrire ai clienti servizi più avanzati, compresi i pagamenti mobile, che possono giustificare la maggiore spesa per la realizzazione di app native per diverse piattaforme. Questa è la soluzione ideale per la grande distribuzione, per aziende di valenza nazionale o transnazionale, le cui grandi dimensioni consentono di recuperare in tempi relativamente brevi i costi di sviluppo e manutenzione delle app. Per aziende

più piccole, l'approccio ideale potrebbe essere invece la web app. L'Osservatorio Mobile Marketing & Service della School of Management del Politecnico di Milano (http://www.osservatori.net/mobile_marketing_service) ha evidenziato come L'adozione di un approccio strategico al canale Mobile presuppone di considerarlo come un possibile strumento per seguire il cliente lungo tutto il ciclo della relazione con l'azienda, come un media utilizzato in maniera continuativa e come parte integrante di una strategia di marketing e di gestione della relazione con il cliente, complessiva e multicanale". In sostanza, a un elevato livello di diffusione fa riscontro un alto valore generato per le imprese che hanno sviluppato applicazioni specifiche.

Arces MARS ha già diversi progetti aperti su piattaforma Android anche grazie al lavoro di studenti del corso di Elaborazione delle Informazioni M del CDL Ingegneria Informatica Laurea Magistrale a Cesena, ed ha deciso quindi di ampliare il pacchetto-offerta del prodotto cyberphysical environment tramite un'applicativo Android sviluppato ad hoc.

Heart Life Doctor (HLD) è l'applicazione sviluppata su piattaforma Android che consente il monitoring del cyberphysical environment presente in Arces MARS.

In questo capitolo viene presentato il sistema operativo Android e gli strumenti necessari che consentono lo sviluppo delle applicazioni.

6.1 La piattaforma Android

Android è oggi il sistema operativo per dispositivi mobili più diffuso al mondo. Si è imposto sul mercato in tempi brevissimi sbaragliando la concorrenza e divenendo una piattaforma di riferimento quando si parla di *mobile device*. Un'ascesa tanto rapida non è dipesa solamente dalla solidità del sistema e dalla semplicità di utilizzo offerta agli utenti. Il ruolo chiave infatti è stato giocato dalle applicazioni.

Android non è un linguaggio di programmazione, ma un vero e proprio insieme di strumenti e librerie per la realizzazione di applicazioni mobili. Android ha la fondamentale caratteristica di essere open, dove il termine assume diversi significati:

- Android è open in quanto utilizza tecnologie open come il kernel di Linux nella versione 2.6.
- Android è open in quanto le librerie e le API utilizzate per la sua realizzazione sono le stesse che verranno utilizzate per lo sviluppo di nuove applicazioni. Questo permette allo sviluppatore di rimpiazzare la quasi totalità dei componenti di Android con i propri rendendo la personalizzazione del sistema quasi totale.
- Android è open in quanto il suo codice è open source, consultabile da chiunque possa contribuire a migliorarlo, lo voglia documentare o semplicemente voglia scoprirne il funzionamento. La licenza scelta dalla Open Handset Alliance è la Open Source Apache License 2.0, che permette ai diversi vendor di costruire su Android le proprie estensioni anche proprietarie senza legami che ne potrebbero limitare l'utilizzo. Ciò significa che non bisogna pagare alcuna royalty per l'adozione di Android sui propri dispositivi.

6.2 Origini di Android

Ogni tecnologia nasce da una esigenza. Android è nato per fornire una piattaforma aperta e per imporsi come standard per la realizzazione di applicazioni mobili. Google non ha realizzato Android da zero, infatti nel 2005 ha acquistato la Android Inc. con i principali realizzatori che hanno poi fatto parte del team di progettazione di questa piattaforma. Nel 2007 le principali aziende nel mondo della telefonia hanno dato origine alla Open Handset Alliance (OHA), di cui fanno parte, oltre a Google:

- Motorola, Samsung, Sony-Ericsson, HTC, Asus e Toshiba come produttori di dispositivi;
- Sprint-Nextel, Vodafone, T-Mobile e altri operatori telefonici;
- Intel, Texas Instruments e NVIDIA come costruttori di componenti.

Le aziende citate sono solo alcune che compongono questa grande alleanza [62]. L'obiettivo comune è quello di creare una piattaforma open in grado di tenere il passo del mercato senza il peso di royalties che ne possano frenare lo sviluppo.

La prima versione del Software Development Kit (SDK) uscita nel 2007 venne testata dagli sviluppatori sul primo dispositivo reale, ovvero il G1 della T-Mobile, nei primi mesi del 2008.

Un passo fondamentale nella storia di Android è avvenuto nell'ottobre del 2008, quando è stato rilasciato il sorgente in open source con licenza di Apache ed è stata annunciata la release candidate dell'SDK 1.0. Nel febbraio del 2009 la versione 1.0 è stata affinata con la versione 1.1.

Nell'Aprile 2009 venne rilasciata la versione 1.5 dell'SDK detta anche Cupcake. La principale novità è stata l'introduzione della tastiera virtuale, liberando così i produttori di hardware dal vincolo della realizzazione di una tastiera fisica.

Ancora acerbo, Android dovette scontrarsi con altri OS già consolidati e corredati quindi di un nutrito campionario di applicazioni, come Symbian, allora molto apprezzato per le elevate prestazioni e il supporto nativo del Wi-Fi, oppure Windows Mobile. Nonostante questo, il nuovo OS targato Google ebbe un successo straordinario dovuto principalmente alla sua natura open source (in quanto basato sul kernel Linux) e alla elevata adattabilità per dispositivi mobili. Contrariamente a Symbian e ad altri sistemi proprietari, Android è offerto gratuitamente da Google ai produttori, basando il guadagno sui servizi online e sulla pubblicità ed è in continua evoluzione grazie alla presenza di una nutrita community alle spalle.

Dall'aprile 2009 ogni versione di Android viene rilasciata con un nome in

codice basato su un dolce. Esse sono state rilasciate in ordine alfabetico: Cupcake (1.5), Donut (1.6), Eclair (2.0/2.1), Froyo (2.2), Gingerbread (2.3), Honeycomb (3.0), Ice Cream Sandwich (4.0) e Jelly Bean (4.1).

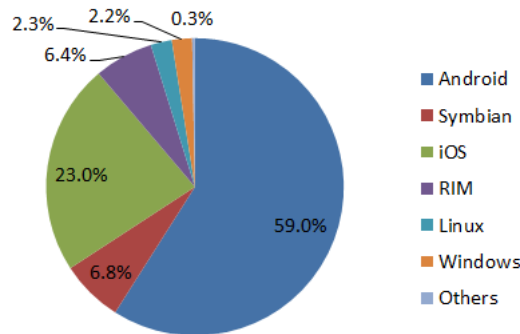


Figura 6.1: Diffusione dei sistemi operativi per smartphone secondo IDC Worldwide Mobile Phone Tracker del 24 Maggio 2012

Ad oggi Android, giunto alla versione 4.1, viene adottato da aziende come HTC, Acer, Samsung, Motorola, LG, Sony Ericsson. Secondo la International Data Corporation (IDC) Worldwide Quarterly Mobile Phone Tracker ¹, i sistemi operativi mobili Android e iOS detenevano quote di mercato del 59,0% e 23,0% rispettivamente dei 152,3 milioni di smartphone venduti nel primo quadrimestre del 2012. Già nel corso del primo trimestre del 2011, i due sistemi operativi hanno tenuto una quota complessiva del 54,4%. Gli incrementi di quota indicano che Android e iOS hanno nettamente preso le distanze dai precedenti leader di mercato Symbian e BlackBerry, ma anche da Linux, Windows Phone 7 e Windows Mobile.

Android si conferma l'attuale ² leader globale tra i sistemi operativi mobili, rappresentando più della metà di tutte le forniture di smartphone. Inoltre, Android vanta la più lunga lista di partner fornitori di smartphone; Samsung

¹International Data Corporation (IDC) è una società di analisi e ricerche di mercato specializzata in information technology e telecomunicazioni. www.idc.com

²Periodo di riferimento: primo semestre 2012.

tra tutti è stato il maggior contributo al successo di Android rappresentando il 45,4% di tutti gli smartphone basati su Android.

Ricordiamo che il sistema Android viene utilizzato anche per dispositivi non inerenti alla telefonia, come tablet pc o ebook reader, ed ha raggiunto importanti quote di mercato anche nel settore tablet.

I sorgenti di Android sono rilasciati interamente sotto licenze open source. Il kernel, derivato da Linux 2.6 e modificato architetturealmente al punto di perdere compatibilità con applicazioni e librerie Linux standard, è rilasciato sotto licenza GPLv2. Il resto del codice, strutturato a stack con vari livelli di astrazione, è invece rilasciato sotto licenza Apache License 2.0. Sebbene l'OS sia open source, per poter rilasciare commercialmente un dispositivo che usi il trademark Android è necessario che questo soddisfi le specifiche, sia dal punto di vista hardware che software, definite nel Compatibility Definition Document (CDD) periodicamente rilasciato da Google in occasione di ogni release principale. Tali specifiche, che includono la compatibilità con le API e gli standard supportati, e la presenza di applicazioni fondamentali e componenti hardware di base, garantiscono la conformità ad un set di caratteristiche standard per ogni cellulare o tablet Android, e permettono di considerare tali dispositivi come una piattaforma hardware e software unica, con software applicativo crosscompatibile e potenzialità simili nonostante le differenze tra i vari modelli, rilasciati da produttori differenti e destinati a diverse fasce di mercato.

6.3 Architettura Software

Android è al tempo stesso un sistema operativo, una piattaforma di sviluppo e una collezione di software di base per l'utilizzo di un dispositivo portatile [65]. Il suo target sono i dispositivi mobili, ma l'architettura che lo realizza ha poco da invidiare a quelle dei comuni sistemi operativi per desktop o laptop.

Essendo sviluppato su kernel Linux (versione 2.6), Android eredita un insieme di valide caratteristiche. Oltre l'affidabilità e la stabilità intrinseche al kernel, la piattaforma risulta essere largamente portabile e ciò ne ha favorito la diffusione. I costruttori di dispositivi non hanno problemi di licenza o di costi nell'implementare Android sui propri prodotti e possono scegliere numerose soluzioni hardware ben supportate dal sistema.

Lo stack software che funge da struttura per il sistema operativo, i cui componenti principali sono dettagliati in Figura 6.2, include il kernel Linux al livello di astrazione più vicino all'hardware, per servizi di sistema fondamentali quali sicurezza, gestione dell'energia, dei processi e della memoria a basso livello, stack di rete e driver delle periferiche di input/output.

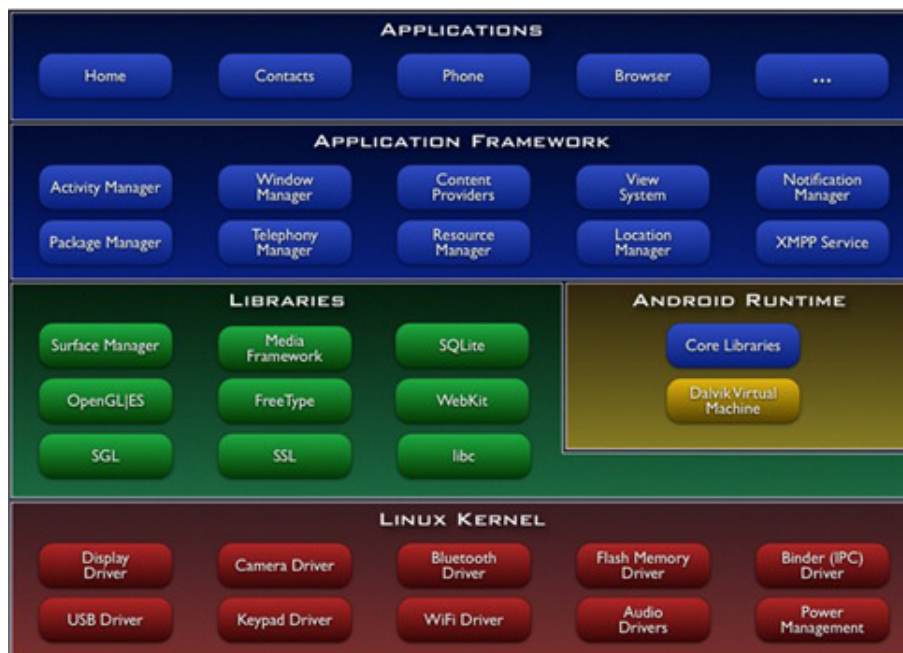


Figura 6.2: Architettura del sistema operativo

Al livello di astrazione immediatamente superiore, si trovano le librerie principali. Le librerie grafiche 2D (custom) sono denominate SGL, mentre quelle

3D sono basate su OpenGL ES, 1.0 nelle prime versioni e 2.0 in quelle successive, con accelerazione hardware opzionale. L'archiviazione dei dati è gestita mediante un database relazionale SQLite, ed il rendering del testo, sia bitmap che vettoriale, mediante FreeType. Le altre librerie di base includono il supporto al protocollo SSL per la sicurezza, un'implementazione delle librerie di sistema C (libc), derivata da BSD e ottimizzata per sistemi embedded basati su Linux, e librerie multimediali per la riproduzione e lo streaming di file multimediali in diversi formati.

6.3.1 Il Kernel Linux

Il layer di più basso livello è rappresentato dal kernel Linux nella versione 2.6. La necessità era infatti quella di disporre di un vero e proprio sistema operativo che fornisse gli strumenti di basso livello per la virtualizzazione dell'hardware sottostante attraverso la definizione di diversi driver. È possibile notare la presenza di driver per la gestione delle periferiche multimediali, del display, della connessione Wi-Fi e dell'alimentazione.

A differenza di un kernel Linux standard, al Kernel Android sono stati aggiunti ulteriori moduli:

- **Binder IPC Driver**, un driver dedicato che permette a processi di fornire servizi ad altri processi attraverso un insieme di API di più alto livello rispetto a quelle presenti su un kernel Linux standard, ciò permette la comunicazione tra processi con un costo computazionale minore.
- **Low memory killer**, sistema che si preoccupa di “uccider” i processi liberando così spazio nella memoria centrale per soddisfare le richieste di un altro processo. Ad ogni processo viene assegnato un punteggio e il processo che detiene il punteggio più alto verrà scelto come candidato per un'eventuale eliminazione. Questi punteggi vengono assegnati in funzione dell'importanza di un processo, per esempio il processo che

controlla la user interface (UI) di un'applicazione che per il momento non è visibile sullo schermo ha un punteggio superiore rispetto al processo che gestisce la UI che in questo momento è in primo piano. Il processo `init` non può essere ucciso.

- **Ashmen**, sistema di memoria condiviso anonimo (Anonymous Shared Memory) che definisce interfacce che consentono ai processi di condividere zone di memoria attraverso un nome. Per esempio il sistema potrebbe utilizzare Ashmen per memorizzare delle icone che possono essere utilizzate da più processi al momento dell'elaborazione della loro interfaccia utente. Il vantaggio di Ashmem rispetto ai sistemi Linux che fanno uso della tradizionale memoria condivisa è che fornisce un mezzo al kernel di recuperare questi blocchi di memoria se non sono attualmente in uso. Se un processo tenta di accedere a un blocco di memoria condivisa che il kernel ha liberato, verrà generato un errore, e sarà quindi necessario riallocare il blocco e ricaricare i dati.
- **RAM console e log devices**, per agevolare il debug, Android fornisce la capacità di memorizzare i messaggi di log generati dal kernel in un buffer RAM. Inoltre, Android fornisce un modulo separato che può essere utilizzato dai processi utente per leggere e scrivere messaggi di log.
- **Android Debug Bridge**, protocollo che permette di gestire in maniera versatile un'istanza dell'emulatore o un dispositivo reale.
- **Power Management**, progettato per permettere alla CPU di non consumare energia se nessuna applicazione o servizio ne fa richiesta.

6.3.2 Librerie Native

Sopra lo strato costituito dal kernel Linux, risiede il livello che contiene un insieme di librerie realizzate in C e C++, e che rappresentano il core di Android. Alcune delle librerie di base sono elencate di seguito.

- **Surface Manager:** è un componente fondamentale che ha la responsabilità di gestire le view ovvero ciò di cui un'interfaccia grafica è composta. Ha quindi accesso alle funzionalità del display e permette la visualizzazione contemporanea di grafica 2D e 3D dalle diverse applicazioni.
- **OpenGL ES:** è la libreria per la grafica 3D, la quale permette di accedere alle funzionalità di un eventuale acceleratore grafico hardware. Si tratta di una versione ridotta di OpenGL specializzata per dispositivi mobili.
- **SGL:** la Scalable Graphics Library è una libreria in C++ che insieme alla OpenGL costituisce il motore grafico di Android. Mentre per la grafica 3D ci si appoggia all'OpenGL, per quella 2D viene utilizzato un motore ottimizzato chiamato appunto SGL.
- **Media Framework:** basato sulla libreria open source OpenCore di PacketVideo fornisce il supporto per la gestione dei più popolari formati audio e video oltre a quelli per la gestione delle immagini.
- **FreeType:** è il motore di rendering per i font FreeType. Attraverso di esso, le applicazioni di Android sono in grado di visualizzare immagini di alta qualità. E' stato utilizzato questo motore perché è di piccole dimensioni, molto efficiente e portabile.
- **SSL:** libreria per la gestione dei Secure Socket Layer. Permette una comunicazione sicura e una integrità dei dati su reti TCP/IP come, ad esempio, internet cifrando la comunicazione dalla sorgente alla destinazione sul livello di trasporto.
- **SQLite:** è un potente e leggero database relazionale a disposizione di tutte le applicazioni.
- **WebKit:** si tratta di un browser engine open source basato sulle tecnologie HTML, CSS, Javascript e DOM.

- **Libc:** si tratta di un'implementazione della libreria standard C `libc` ottimizzata per i dispositivi basati su Linux come Android.

6.3.3 Sviluppo ed esecuzione delle applicazioni

Mentre i livelli più bassi dell'architettura Android sono sviluppati in linguaggio C, le applicazioni sono scritte in codice Java ed eseguite mediante una macchina virtuale open source, la *Dalvik Virtual Machine*, con architettura register-based e compilazione just-in-time.

Nata dall'esigenza di eseguire del software su macchine con risorse limitate la Dalvik virtual machine è un'ottimizzazione della macchina virtuale della Sun Microsystem che permette di sfruttare al massimo le caratteristiche del sistema operativo ospitante. La Dalvik Virtual Machine è in grado di eseguire codice contenuto all'interno di file con estensione `.dex` ottenuti a partire dal bytecode Java. I file con estensione `.dex` hanno un tipo di compressione che permette di dimezzare lo spazio adibito alla loro memorizzazione rispetto ai file `jar` non compressi. La riduzione dello spazio utilizzato deriva dal fatto che stringhe e costanti presenti in più classi vengono incluse solo una volta nel singolo file `.dex`.

I motivi che spingono i programmatori a prediligere la piattaforma Android sono principalmente tre:

- La programmazione è in stile Java.
- Gli strumenti di sviluppo sono gratuiti.
- Le app possono essere distribuite liberamente.

Il byte code eseguito dalla Dalvik VM è differente da quello Java, ma l'uso di *dex* (Dalvik EXecutable) è trasparente al programmatore.

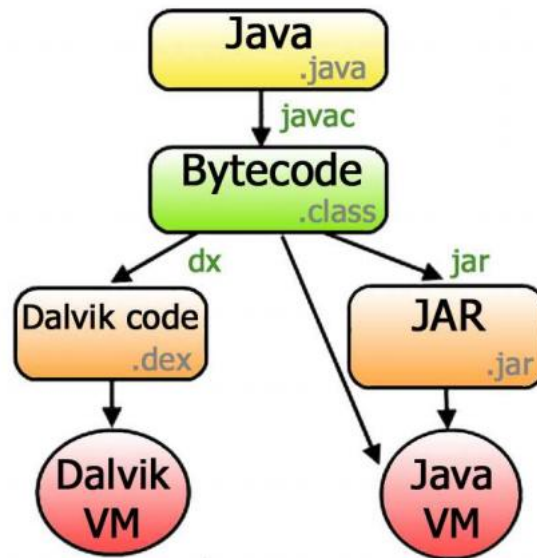


Figura 6.3: Processo di compilazione per Dalvik Virtual Machine

Come si può osservare in Figura 6.3, il codice Java verrà compilato dapprima in modo standard fino a ottenerne il relativo bytecode; successivamente, l'ambiente di sviluppo lo trasformerà in dex, più compatto e ottimizzato per eseguire sulla Dalvik VM. La libreria di base che affianca la macchina virtuale offre buona parte dei package propri di Java Standard Edition, ma il supporto non è completo: si può notare la mancanza di AWT e Swing d'altra parte sostituiti da librerie grafiche dedicate. I pacchetti fondamentali sono presenti e in larga misura rispettosi dei corrispettivi Java. Da questo punto di vista, Android si dimostra generoso se non addirittura ambizioso: non si limita a garantire una compatibilità con Java ME (Micro Edition), ma offre una libreria molto più completa e orientata ai sistemi desktop (SE). In aggiunta esistono molti package con prefisso android, esclusivi di questa speciale piattaforma e capaci di estenderne potenzialità e flessibilità [65].

Android Software Development Kit

La filosofia open che pervade Android si ripercuote anche sugli strumenti di sviluppo: l'Android SDK (Software Development Kit) è disponibile in forma libera e gratuita per i principali sistemi operativi (Windows, Mac OS X, Linux). Per gli utilizzatori di Eclipse, rinomato ambiente integrato di sviluppo multi-linguaggio e multi-piattaforma, è possibile installare un comodo plugin dedicato: Android Development Tools (ADT) Plugin.

Come anticipato in precedenza, oltre che tecnicamente avanzata e flessibile, la piattaforma Android si dimostra essere una delle più semplici e veloci da usare per chi realizza le app. Il processo di produzione, distribuzione e diffusione è aperto, poco costoso e rapido: ne consegue un mercato molto appetibile formato da una numerosa schiera di utenti e sviluppatori.

Android Security Sandbox

Pensate per dispositivi mobili, le applicazioni debbono prevedere un buon grado di interazione con l'utente, mantenersi intuitive ed evitare lo spreco di risorse. Per tali motivi, i progettisti di Android hanno pensato di definire alcune tipologie di componenti e un meccanismo di comunicazione fra gli stessi in grado di ottimizzare l'impiego del sistema ospite, permettere una elevata personalizzazione e garantire una forte estendibilità della piattaforma.

Allo stesso tempo, ogni applicazione è eseguita come processo separato con una propria istanza della macchina virtuale, grazie al ridotto footprint della stessa e dell'eseguibile, in modo da garantire una completa separazione tra processi concorrenti. Tale approccio è denominato *security sandbox*: le applicazioni sono isolate e indipendenti le une dalle altre, e hanno accesso a un sottoinsieme limitato e controllato di dati e risorse, per garantire sicurezza nell'accesso a dati sensibili e tolleranza ai guasti in caso di malfunzionamento. Cio' è possibile grazie al supporto multiutente del kernel Linux: ogni applicazione è gestita come se fosse un utente diverso a cui è associato un proprio user ID, accessibile esclusivamente dal sistema operativo e non conosciuto

dall'applicazione stessa, ed i permessi di ogni file da essa utilizzato vengono settati in modo che solo ed esclusivamente tale user ID possa accedervi. Il processo, inoltre, viene avviato soltanto quando la sua esecuzione è necessaria e termina immediatamente non appena smette di esserlo, o quando il sistema ha bisogno di recuperare memoria per gli altri processi eventualmente in esecuzione. In questo modo Android implementa il *principio del privilegio minimo*, ovvero rende disponibili ad ogni applicazione esclusivamente i componenti immediatamente necessari al proprio funzionamento e impedisce l'accesso alle parti del sistema per cui essa non dispone di privilegi sufficienti. La condivisione di dati e risorse tra più applicazioni differenti è comunque possibile tramite meccanismi ad-hoc che consentono ad esse di condividere lo stesso user ID, e quindi gli stessi privilegi sui file.

6.3.4 Application Framework

L'application framework è costituito da un'insieme di API che sfruttano le librerie sottostanti nello stack Android e permettono allo sviluppatore di realizzare le applicazioni.

- **Activity Manager**, modulo che si occupa della gestione delle activity, ovvero quelle entità che sono associate a una schermata e che quindi permettono all'utente finale di interagire con l'applicazione. Il suo compito è quello di gestire il ciclo di vita delle activity e di organizzare le loro schermate in uno stack secondo l'ordine di visualizzazione sullo schermo.
- **Package Manager**, gestisce il processo di installazione e rimozione di un'applicazione.
- **Window Manager**, astrazione attraverso le API Java dei servizi nativi del Surface Manager, si occupa della gestione delle finestre di applicazioni differenti eseguite da processi differenti.

- **Telephony Manager**, permette l'interazione con le caratteristiche proprie di un telefono, ad esempio iniziare una chiamata o controllare lo stato della stessa.
- **Content Provider**, gestisce la condivisione di informazioni tra i vari processi. Il suo utilizzo è simile a quello di un repository condiviso con cui le diverse applicazioni possono interagire inserendo o leggendo informazioni.
- **Resource Manager**, gestisce le informazioni relative a un'applicazione quali file di configurazione, file di definizione di layout, immagini utilizzate per la personalizzazione dell'interfaccia grafica e così via.
- **View System**, gestisce l'insieme delle viste utilizzate nella costruzione dell'interfaccia grafica di un'applicazione come bottoni, griglie, text boxes, etc.
- **Location Manager**. Le applicazioni che gestiscono le informazioni relative alla localizzazione si chiamano Location Based Application (LBA). Queste informazioni possono essere realizzate utilizzando API messe a disposizione dal

Location Manager. Tramite il Location Manager è possibile accedere a funzioni legate alla localizzazione, tra cui le operazioni di georeferenziazione.

- **Notification Manager**, rappresenta un insieme di strumenti utilizzabili dalle applicazioni per notificare eventi al dispositivo, il quale reagirà in conseguenza della notifica ricevuta ad esempio emettendo una vibrazione, facendo lampeggiare i LED, visualizzando un'icona e altro ancora.

6.4 Le Applicazioni

All'ultimo livello dello stack architetturale di Android troviamo le applicazioni. Scritte in linguaggio Java, si presentano sotto forma di pacchetti con estensione .apk che incapsulano, oltre al codice java compilato, dati e risorse necessarie all'applicazione stessa.

Ogni applicazione viene eseguita all'interno di un processo Linux che viene avviato nel momento in cui il codice dell'applicazione deve essere eseguito. Ogni processo, a sua volta, viene eseguito all'interno di una propria virtual machine, isolando in questo modo il codice di due applicazioni in esecuzione nello stesso momento. Ad ogni applicazione viene assegnato un Linux user ID, questo permette di definire i permessi in modo tale che il codice, i dati e le risorse appartenenti ad un'applicazione siano visibili soltanto all'applicazione stessa a meno che non venga esplicitato il contrario.

Infatti è possibile fare in modo che due o più applicazioni condividano lo stesso user ID e che quindi detengano gli stessi permessi sui loro dati. Applicazioni che condividono lo user ID possono essere eseguite dallo stesso processo e quindi nella stessa VM.

Grazie ai permessi un'applicazione può far uso di una componente di un'altra applicazione. Tutto ciò è molto utile perchè permette allo sviluppatore di riutilizzare codice di altre applicazioni senza doverlo implementare nella propria.

Esistono quattro tipologie di componenti:

- **Activities:** un activity è un'interfaccia grafica attraverso la quale un utente può interagire con l'applicazione. A ciascuna activity viene data una finestra di default nella quale può essere disegnata. I contenuti visivi di ciascuna finestra vengono realizzati attraverso una gerarchia di viste (views), ciascuna delle quali controlla un particolare spazio rettangolare all'interno della finestra. Questa gerarchia può essere vista come un albero dove viste genitori contengono e controllano la disposizione delle viste figlie. Le viste foglie invece rispondono direttamente

alle azioni dell'utente. Una gerarchia di viste è collocata all'interno di una finestra activity attraverso il metodo `Activity setContentView()`. Il content view è l'oggetto vista radice della gerarchia.

- **Services:** sono dei task che vengono eseguiti in background per un indefinito periodo di tempo. Un service può essere ad esempio della musica che viene suonata in background mentre l'utente non sta interagendo con un media player ma stà facendo dell'altro. Android mette a disposizione delle interfacce attraverso le quali un'applicazione può connettersi a un determinato servizio e usufruirne in base alla definizione dell'interfaccia stessa. Nell'esempio della musica l'interfaccia può mettere a disposizione comandi come play, stop e pause.
- **Broadcast Receivers:** è un componente che ha il compito di eseguire delle azioni solo in seguito alla ricezione di un determinato evento. Un evento può essere ad esempio la notifica del basso livello di energia della batteria, il cambio di fuso orario, etc. Un'applicazione può avere più di un broadcast receiver per rispondere agli annunci che si reputano importanti. Ogni ricevitore estende la classe `BroadcastReceiver`.
- **Content Providers:** permette la condivisione di dati tra le applicazioni. Un content provider estende la classe `ContentProvider` al fine di implementare un insieme standard di metodi che permettono ad altre applicazioni di recuperare e memorizzare i dati che esso gestisce.

Diversamente dal Content Provider, che viene attivato in seguito a una richiesta da parte del Content Resolver, le altre tre componenti vengono attivate da messaggi asincroni chiamati *intent*. Gli intent realizzano un meccanismo in grado di riutilizzare le activity (e altri componenti) per eseguire quelle operazioni che possono essere comuni a più applicazioni: un classico esempio è la necessità di accedere alla rubrica telefonica e recuperare un contatto da un elenco. Evitare che ciascuna applicazione definisca un proprio metodo di accesso alla rubrica e di gestione dei contatti permette di risparmiare risorse, codice e ottenere vantaggi dal punto di vista dell'usabilità. Un'attività che

vorrà accedere alla rubrica del telefono definirà un opportuno intent che poi potrà essere utilizzato per l'avvio di un'altra attività capace di gestirlo. A sostegno del meccanismo appena descritto esiste l'*intent filter* che permette ai componenti di un'applicazione di dichiarare l'insieme degli intent che sono capaci di gestire. Ci sono più modi per attivare ciascun componente:

- Una activity può essere attivata passando un oggetto di tipo Intent ai metodi `Context.startActivity()` oppure `Activity.startActivityForResult()`. L'activity può ottenere informazioni sull'intent che la ha attivata invocando il metodo `getIntent()`.
- Un service viene attivato passando un oggetto di tipo intent al metodo `Context.startService()`. Un intent può anche essere passato al metodo `Context.bindService()` per stabilire una connessione tra la componente chiamante e il servizio che in questo caso riceve un oggetto intent passato al metodo `onBind()`.
- un'applicazione può iniziare un broadcast passando un oggetto intent ai metodi `Context.sendBroadcast()`, `Context.sendOrderedBroadcast()`, e `Context.sendStickyBroadcast()`. Il sistema consegnerà gli intent a tutti i ricevitori interessati invocando il loro metodo `onReceive()` ereditato dalla classe `BroadcastReceiver`.

Ciascuna componente deve poter essere disattivata nel momento in cui diventa non necessaria. Per i content provider, che vengono attivati solo in risposta ad una richiesta del content resolver, e per i broadcast receiver, attivati in risposta ad un messaggio broadcast, non è necessario esplicitarne la disattivazione. Un discorso diverso va fatto per le activity che possono essere disattivate invocando il proprio metodo `finish()` oppure possono essere disattivate da un'altra activity che in questo caso chiama il metodo `finishActivity()`. Un service, invece può essere invece bloccato chiamando il suo metodo `stopSelf()` oppure `Context.stopService()`.

Da questa descrizione è possibile intuire che una singola applicazione è composta da molteplici componenti che intervengono in vario grado durante l'esecuzione, con interazioni anche complesse.

6.4.1 Ciclo di vita dei componenti

Un aspetto fondamentale dell'architettura Android è che il ciclo di vita di ciascun componente è di completa responsabilità dell'ambiente e l'unico punto di intervento da parte dello sviluppatore è quello relativo all'implementazione dei metodi di callback invocati a seguito di modifiche nello stato del componente stesso. Per comprendere meglio questi concetti è utile riferirsi alle attività e descriverne la gestione da parte del sistema.

Tipico della maggior parte delle applicazioni è fare uso di diverse attività: ciascuna di esse è eseguita all'interno di un determinato processo Linux e la percezione che ne ha comunemente un utente è quella di "schermata" [66]. Si può dire che un'activity sia un contenitore di componenti grafici che "riempiono" lo schermo. Per ogni componente si dovrà associare un *listener* che si occupa di intercettare gli eventi di I/O corrispondenti (per esempio un'azione di tocco sullo schermo) e reagire di conseguenza. Una activity rappresenta quindi ciò che l'utente vede e può fare durante l'esecuzione dell'applicazione.

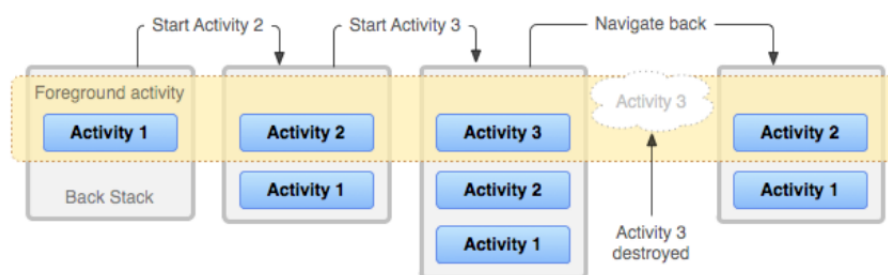


Figura 6.4: Activity Stack

Ogni applicazione può essere composta di molte activity. Una di esse sarà la activity principale, che rappresenta cioè l'entry point dell'applicazione,

ovvero l'activity riconosciuta dal S.O. come la prima view da rappresentare all'avvio.

La piattaforma Android organizza le activity secondo una struttura a stack (come mostrato in Figura 6.4) dove quella posta più in alto è attiva in un certo momento. Visualizzare una nuova schermata, e dunque all'avvio di una nuova activity, porterà quest'ultima in cima allo stack mettendo in uno stato di pausa le altre. Terminato il proprio lavoro, le eventuali informazioni raccolte saranno passate all'attività precedente, la quale diventerà nuovamente attiva.

Ottimizzare le risorse prevede che una activity non visualizzata possa essere eliminata dal sistema per poi essere eventualmente ripristinata successivamente.

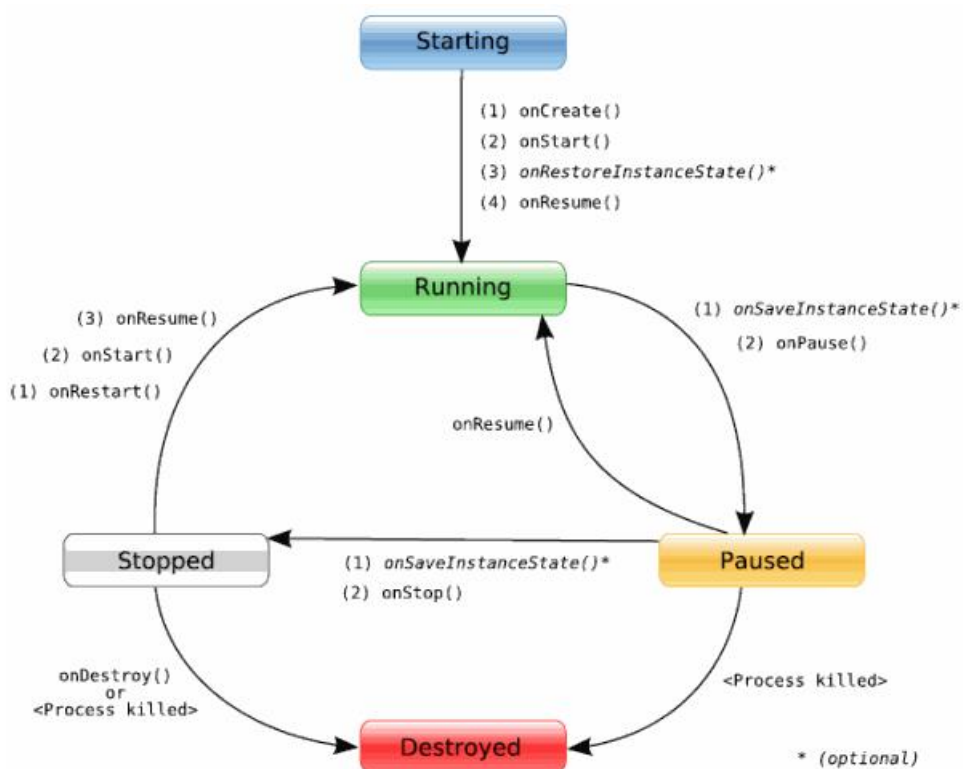


Figura 6.5: Ciclo di vita di una Activity

Gli stati possibili per una activity sono rappresentati in Figura 6.5 e per maggiore chiarezza li descriveremo di seguito:

- **Running:** l'attività è in cima allo stack, è visibile e ha il focus; può ricevere gli eventi da parte dell'utente.
- **Paused:** l'activity non è attiva ma è ancora visibile per la trasparenza di quelle superiori o perché queste non occupano tutto lo spazio a disposizione. Insensibile agli eventi da parte dell'utente viene eliminata dal sistema solo in caso di estrema necessità.
- **Stopped:** è lo stato delle activity non attive né visibili; è insensibile agli eventi dell'utente ed è fra le prime candidate a essere eliminata.
- **Inactive:** una activity si trova in questo stato quando viene eliminata o prima di essere creata.

Quando l'ambiente decide per il cambiamento di stato dell'activity, invocherà i metodi di callback predefiniti, anch'essi indicati in Figura 6.5.

Il ciclo di vita segue attentamente lo schema proposto in Figura 6.6.

All'avvio dell'applicazione vengono invocati tre metodi di inizializzazione ciascuno dedicato a un aspetto specifico dell'activity: esistere, essere visibile, essere attiva.

Il metodo *onCreate()* contiene la maggior parte delle operazioni di inizializzazione e può ricevere un parametro di tipo Bundle che permette di ottenere un riferimento a un eventuale stato che l'activity aveva prima di essere eliminata dal sistema. Se il metodo va a buon fine, l'activity è stata creata e si prepara alla propria visualizzazione.

Il metodo *onStart()* viene eseguito subito dopo *onCreate()* e al suo termine l'activity può aver ottenuto il focus ed essere posta in cima allo stack.

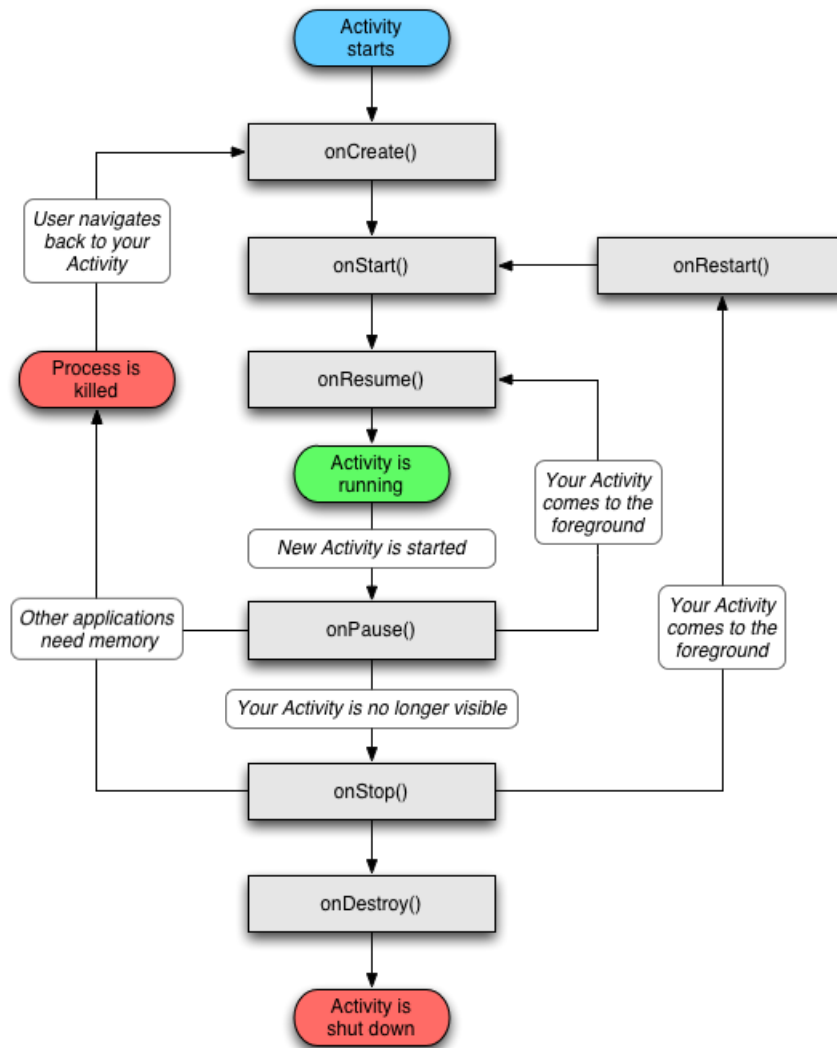


Figura 6.6: Activity Lifecycle

Se `onStart()` riesce nei suoi intenti, il metodo `onResume()` viene eseguito ponendo l'activity nello stato *Running* ove permarrà finché non sarà più quella attiva. Ciò può avvenire per diverse ragioni, ad esempio l'utente può aver premuto il tasto Back sul dispositivo: in questo caso l'attività corrente viene rimossa dalla cima dello stack e messa in stato *Paused*. Affinché ciò sia possibile deve essere invocato il metodo `onPause()` che permette di disabilitare gli input dell'utente.

L'attività che è posta in cima allo stack sarà quella da ripristinare e rendere nuovamente visibile: allo scopo il sistema invoca su di essa il metodo `onRestart()` e subito dopo `onStart()` e `onResume()`, secondo le modalità descritte. Notiamo che il metodo di callback `onRestart()` è molto simile a `onCreate()`, il primo però è usato su una activity esistente e deve preoccuparsi dell'eventuale recupero del suo stato.

Ripristinata l'attività in cima allo stack, dopo che è stata inserita nello stato di *Running* e abilitata a ricevere gli input dall'utente, il sistema interviene su quella lasciata in *Paused*. Quest'ultima vi permane se resta parzialmente visibile all'utente, altrimenti su di essa viene invocato dapprima il metodo `onStop()` (con trasferimento nello stato *Stopped*) e successivamente `onDestroy()` che la rende *Inactive*.

Il ripristino di un'attività spiegato precedentemente descrive la procedura prevista per una activity in stato *Stopped*. Qualora l'attività avesse assunto stato *Paused*, si sarebbe eseguito unicamente il metodo di callback `onResume()`, come indicato in Figura 6.5.

Come già indicato, lo sviluppatore può istruire il sistema Android su cosa fare nel momento in cui si verifichi un cambiamento di stato dell'activity per mezzo dell'invocazione dei metodi di callback relativi, ma non ha controllo su quando ciò accada. È l'ambiente a deciderlo e seppur fortemente influenzato dalla volontà dell'utente che in modo diretto interagisce col display o con i tasti Home e Back del dispositivo, può anche dipendere da eventi esterni. Il caso più ricorrente è quello della telefonata in arrivo: se il telefono squilla mentre si sta usando la calcolatrice, quest'ultima sarà automaticamente messa in stato di *Stopped* e mandata in sottofondo. All'utente verrà quindi sottoposta l'activity di gestione delle chiamate e potrà decidere di rispondere. Conclusa la telefonata sarà in grado di visualizzare l'attività interrotta e riportarla in primo piano (stato *Running*), riprendendo i calcoli esattamente da dove li aveva interrotti.

Al fine di migliorare la comprensione del modello di concorrenza in Android aggiungiamo in questa sezione introduttiva pochi ulteriori dettagli: il con-

petto di task e l'eliminazione delle attività.

Ciascuna applicazione è eseguita all'interno di un proprio processo Linux ed è frammentata in molteplici componenti con lo scopo di migliorare le prestazioni a tempo di esecuzione, ma è anche possibile eseguire i componenti in processi diversi. Ciò accade di norma quando si utilizza un'attività non definita nell'applicazione, ma richiamata tramite intent, per esempio è richiesto l'accesso a un servizio in background. Nonostante ciò, ovvero vi siano in gioco processi distinti, Android li aggrega in un unico task. Esso consiste in una successione di attività legate all'esecuzione di una applicazione e può includere activity appartenenti allo stesso processo o a processi differenti. È possibile eseguire più applicazioni e le relative attività simultaneamente, ma come descritto in precedenza, soltanto un'activity alla volta può occupare il display, in modo esclusivo. L'alternanza delle stesse è tenuta in considerazione e gestita dal task di appartenenza.

In termini di risorse di calcolo, le attività ibernatae non ne consumano e il concetto di chiusura è secondario e tenuto nascosto all'utente. Lo sviluppatore può intervenire direttamente invocando il metodo `finish()`, ma solitamente evita di gestire questo aspetto demandandolo al sistema. Le attività non dispongono di un pulsante "x" o di un tasto equivalente con il quale terminarle, di conseguenza l'utente non può chiuderle, ma solo mandarle in background. I casi in cui un'attività può terminare sono due:

- ha completato la sua esecuzione, è ibernata e il sistema decide di eliminarla;
- il sistema ha poca memoria libera a disposizione e per recuperare spazio inizia a sopprimere bruscamente quelle in sottofondo.

Anche nei casi più critici, le attività da terminare sono scelte oculatamente, in base alla categoria di appartenenza del loro processo. Le tipologie di processo sono [66]:

- foreground process,

- visible process,
- service process,
- empty process.

I processi a priorità maggiore sono quelli foreground e che si occupano dell'esecuzione dei componenti di interazione con l'utente. Si tratta del processo che sta eseguendo l'attività in cima allo stack, le azioni di BroadcastReceiver o i metodi di callback di un particolare servizio. Questi non saranno eliminati se non nei casi estremi in cui scarseggiassero le risorse per la loro stessa esecuzione.

I processi visible sono a priorità inferiore ai precedenti, ma anch'essi importanti poiché eseguono le attività nello stato *Paused*: visibili parzialmente, non interagiscono con l'utente. Anche questi processi saranno eliminati per condizioni critiche del sistema, ma comunque prima di quelli foreground.

I service process includono i processi che non hanno bisogno di un'interfaccia per relazionarsi con l'utente, ma pur sempre di elevata importanza, come ad esempio un servizio che riproduce musica.

I processi in background sono invece quelli che si occupano di un'activity che non è più visibile all'utente e sulla quale è stato invocato il metodo `onStop()`. Il numero di questi processi è solitamente elevato rispetto a quelli foreground o visible e il sistema li ordina in base al tempo trascorso dall'ultimo utilizzo, attraverso la lista LRU (Last Recently Used). L'ultima categoria di processi prevista da Android viene definita empty in quanto non legati ad alcun componente predefinito sulla piattaforma: sono tra i primi candidati all'eliminazione.

6.5 Ambiente di Sviluppo

Per agevolare lo sviluppatore nel compito di realizzare applicazioni per la piattaforma Android, la Open Handset Alliance ha rilasciato, nel novembre

2007, il software development kit (SDK), ovvero un'insieme di utili strumenti utilizzabili per il debugging, il packaging, l'installazione delle applicazioni.

Lo strumento più importante messo a disposizione dal SDK è senza dubbio l'emulatore, un software che consente di emulare un dispositivo reale sul quale è possibile effettuare il debug, testare e progettare la propria applicazione. Scaricabile all'indirizzo <http://developer.android.com/sdk/index.html> è utilizzabile per le tre diverse piattaforme Windows, Apple Mac OS X e Linux.

6.5.1 Eclipse ADT Plugin

Come già anticipato, per gli utilizzatori di Eclipse android mette a disposizione un comodo plugin dedicato: Android Development Tools (ADT) Plugin, che integra strumenti di sviluppo come compilatore, debugger e simulatore.

All'interno del software ogni progetto è gestito come una gerarchia di folder in cui ogni cartella ha il suo scopo preciso. Android supporta l'esternalizzazione delle risorse, una pratica che evita di cablare le risorse a livello di codice. Le risorse esternalizzate in Android possono essere di vario tipo: stringhe, colori, immagini, animazioni, temi e layout. Esternalizzare una risorsa significa aumentare la manutenibilità l'aggiornamento e la gestione in generale di una applicazione. Un esempio su tutti è il concetto d'internazionalizzazione che utilizza un file diverso per ogni lingua.

Come mostrato in Figura 6.7, le risorse applicative sono memorizzate sotto la cartella */res* del progetto, ogni risorsa è contenuta all'interno di una sotto-cartella specifica in base alla sua tipologia. Alla creazione di un nuovo progetto Eclipse, ADT crea automaticamente il folder */res* e tre sotto-cartelle principali: *values*, *drawable* e *layout*:

- **Drawable** contiene i files immagini, possono essere usati oltre i file png anche formati jpeg e gif.

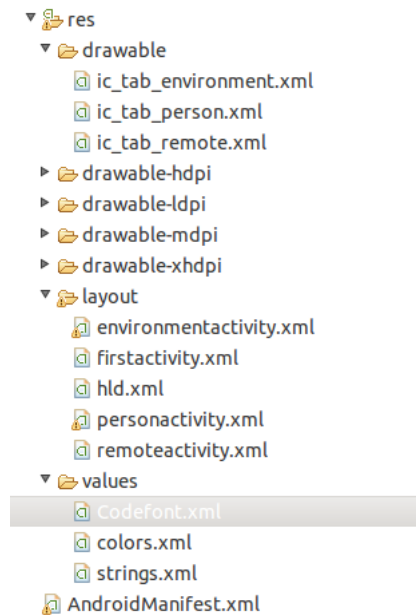


Figura 6.7: Memorizzazione delle risorse in un progetto Android

- **Layout** contiene i file di layout, che hanno lo scopo di disaccoppiare il layer di presentazione da quello di business. Tramite questi file xml è possibile infatti disegnare la user interface per ogni singola activity. Avere un disaccoppiamento della view dal codice offre una serie di benefici, un esempio su tutti è la possibilità di definire viste diverse in base a l'hardware e/o alle dimensioni dello schermo o al suo orientamento (verticale/orizzontale).
- **Values** contiene le definizioni di risorse quali stringhe, colori e dimensioni.

Le risorse possono essere utilizzate direttamente nel codice, oppure referenziate in altre risorse. La classe *R* contiene un riferimento statico per ogni tipo di risorsa (*R.string*, *R.drawable*): ogni volta che viene aggiunta o modificata una risorsa, ADT si occupa di aggiornare la classe *R* aggiungendo i riferimenti alle risorse definite nelle directory contenute in */res*.

6.5.2 Il file Manifest

Ogni applicazione sviluppata per la piattaforma Android deve contenere all'interno della root directory il file *AndroidManifest.xml*. Questo file contiene informazioni riguardanti l'applicazione necessarie alla sua corretta esecuzione. L'Android Manifest ha il compito di:

- definire il nome del pacchetto java dell'applicazione
- descrivere le componenti dell'applicazione. Definire il nome delle classi in cui queste componenti vengono implementate e pubblicare le loro caratteristiche
- determinare quali processi ospiteranno le componenti dell'applicazione
- definire i permessi che un'applicazione deve avere per poter utilizzare alcune API Java protette e per poter interagire con altre applicazioni
- definire un livello minimo di API Java necessarie all'applicazione
- definire una lista di librerie aggiuntive (oltre al core library) necessarie all'applicazione.

6.5.3 Debug e logging di un'applicazione Android

Un aspetto fondamentale del processo di sviluppo di un'applicazione è sicuramente il debug. Anche nel caso di Android sono disponibili strumenti che aiutano i programmatori a sviluppare applicazioni.

Il primo di questi è il debug di Eclipse che permette di eseguire l'applicazione passo dopo passo in modo da trovare facilmente l'eventuale errore.

Importante lo strumento di debug *Logcat* che permette di tracciare cronologicamente le varie operazioni eseguite in modo da controllare la corretta esecuzione dell'applicazione. Il termine *Log* indica la registrazione cronologica delle operazioni man mano che vengono eseguite; il termine *Cat* indica un

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.HLD.DeLuca"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="7" />

    <application android:icon="@drawable/ic_launcher2" android:label="@string/app_name"
        android:debuggable="true" android:theme="@android:style/Theme.Light.NoTitleBar.Fullscreen">
        <activity android:name="app.Activities.FirstActivity"
            android:label="@string/app_name"
            android:screenOrientation = "portrait"
            android:configChanges = "keyboardHidden|orientation">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name="app.Activities.PersonActivity" android:label="@string/app_name"></activity>
        <activity android:name="app.Activities.HLDActivity" android:label="@string/app_name"></activity>
        <activity android:name="app.Activities.EnvironmentActivity" android:label="@string/app_name"></activity>
        <activity android:name="app.Activities.RemoteActivity" android:label="@string/app_name"></activity>
    </application>

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

Figura 6.8: Esempio di file *AndroidManifest.xml* [67]

comando dei sistemi operativi Unix e Unix-like che legge i file che gli sono specificati come parametri (o lo standard input) e produce sullo standard output la concatenazione del loro contenuto. I due termini uniti formano Logcat, che indica quindi il tracciamento cronologico delle operazioni con la relativa visualizzazione sullo standard output del sistema.

E' anche possibile filtrare le informazioni di Logcat (molto verboso) in modo da concentrare l'attenzione solo su alcune di queste ed ottenere quindi uno strumento molto più efficace di individuazione dell'errore.

Capitolo 7

Heart Life Doctor Android Application

Heart Life Doctor (HLD) è il nome scelto per l'applicazione di visualizzazione e controllo dello stato del paziente tramite mobile.



Figura 7.1: Logo di Heart Life Doctor

Il laboratorio Arces MARS dispone già di un'applicativo mobile sviluppato in python per i dispositivi Nokia N9 e N900, ormai fuori dal mercato. Da qui la necessità di evolvere e migrare a una scelta sicura come Android. La vasta disponibilità di dispositivi ha permesso di testare l'applicativo su terminali fisici già presenti in Arces MARS anzichè su simulatore. Grazie alla programmazione Java è possibile sfruttare appieno le potenzialità delle librerie Java KPI con estensione SPARQL che ho contribuito ad implementare.

Lo scopo di HLD è consentire il monitoring del paziente tramite dispositivo mobile, quindi senza la necessità di essere insieme al paziente stesso. HLD mostra in tempo reale i parametri vitali del paziente: peso, battito cardiaco, saturazione del sangue, pressione, temperatura corporea, etc. Inoltre, informa il medico anche dei parametri d'ambiente quali temperatura ambientale, umidità, concentrazione di polvere e inquinamento ¹.

Inoltre, obiettivo di HLD è fornire al medico la possibilità di inviare la cartella clinica completa e/o solo alcuni dei parametri vitali a KP esterni detti *HL7Dispatcher* in ascolto, e ad un servizio web progettato ad hoc da UNIGE, Università di Genova partner del progetto CHIRON.

Come verrà spiegato in seguito, Heart Life Doctor gestisce dunque sia un flusso di dati che un flusso di controllo.

Heart Life Doctor è stato presentato durante la conferenza ICOST 2012 ²: la Conferenza Internazionale sulle Smart Homes ed Health Telematics la cui decima edizione si è tenuta presso Villa Medici Artimino Resort di Firenze nei giorni 12-15 Giugno 2012.

7.1 Specifiche Tecniche

Heart Life Doctor è stato sviluppato per piattaforma Android 2.3 (Gingerbread) e testato su dispositivi Samsung Galaxy S1 SCL mod. I9003 (no brand) di mia proprietà, Samsung Galaxy S2 mod. I9100 (con brand Tim), e Samsung Nexus S mod. I9023 entrambi messi a disposizione da Arces MARS. Al momento dello sviluppo di HLD su tutti i dispositivi era disponibile una versione di Android Gingerbread (2.3 e seguenti) e non Ice Cream Sandwich (4.0) rilasciata al momento della stesura di questo documento di tesi ³.

¹I rilevatori di polvere e inquinamento non sono attualmente disponibili nel laboratorio Arces MARS. HLD è sensor-ready, cioè già configurato per ricevere informazioni appena i sensori saranno disponibili.

²[urlwww.conference-icost.org/](http://www.conference-icost.org/)

³Rilasciata per Samsung Galaxy SII in Luglio 2012, per soli dispositivi con brand Vodafone. <http://www.sammobile.com/>

Lo sviluppo dell'applicazione è stato interamente eseguito tramite Eclipse Platform versione 3.7.2 (Indigo), Android SDK Platform e Android Development Tools (ADT) Plugin per Eclipse.

7.2 Funzionalità

Si riporta ora una breve panoramica dell'uso di Heart Life Doctor, mostrando una ad una le schermate visibili all'utente. In questo modo sarà più facile per il lettore comprendere le scelte progettuali spiegate ampiamente in seguito.

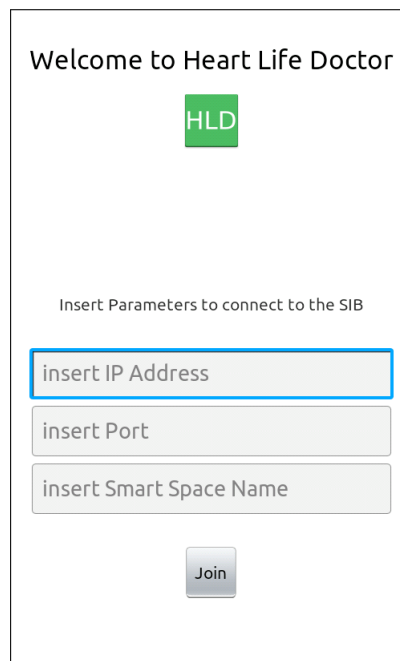


Figura 7.2: Prima activity visualizzata

In Figura 7.2, è mostrata la prima schermata ad essere visualizzata dall'utente. Nelle *EditText* è possibile inserire rispettivamente l'indirizzo IP della SIB a cui connettersi, la porta, ed eventualmente il nome dello Smart Space (il valore di default è *X*).

La pressione del bottone *Join* provoca il passaggio alla schermata successiva (Figura 7.2.1).

7.2.1 Il tab *Remote*

Nella parte alta dello schermo si vedono i 3 tab di cui è composta l'applicazione. Il primo ad essere selezionato è il tab cosiddetto *Remote*. Tramite i *RadioButton* si inviano alla SIB le informazioni relative alla voce in colore verde di fianco al bottone stesso.

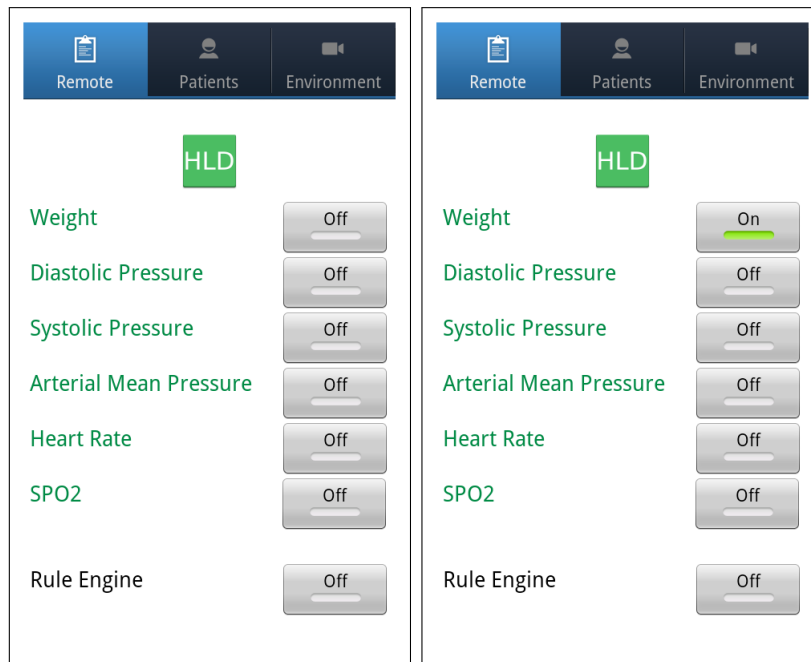


Figura 7.3: Seconda activity visualizzata - il Tab *Remote*

Come abbiamo detto, la SIB è affiancata da una serie di KP in ascolto sulle attività della SIB stessa. Lo scopo del *RadioButton* è provocare l'inserimento di determinate triple all'interno della SIB. Abilitando ad esempio il *RadioButton Weight*, ogni misura di peso rilevata verrà automaticamente

notificata. I KP esterni sottoscritti alle triple del tipo inserito dal RadioButton verranno risvegliati.

```
//...
triples.add(sib.getXmlTools().newTriple( HL7DispatcherWeight, "rdf:type",
    sib.getNs() + "HL7Dispatcher", URI, URI ) );
triples.add( sib.getXmlTools().newTriple( HL7DispatcherWeight,
    sib.getNs() + "Has_Loinc", sib.getNs() + "LOINC-29463-7", URI, URI ) );
//...
```

L'inserimento di tali triple provoca il risveglio del KP *HL7Dispatcher* che, se attivo e collegato alla SIB, ha il compito di prelevare i dati della persona ed è responsabile dell'invio dei dati fisiologici e ambientali al servizio HL7 sviluppato da UNIGE [68]. Il KP *HL7Dispatcher* è in grado di inviare automaticamente ogni singola misura al servizio web (tramite i RadioButton singoli), oppure può inviare una cartella clinica completa se innescato dalla richiesta del medico (si veda il tab *Patient* in seguito).

Il valore di LOINC che appare nel codice fa riferimento alla standardizzazione internazionale LOINC (Logical Observation Identifiers Names and Codes). In breve, l'obiettivo di LOINC è di permettere il movimento elettronico di dati clinici dai laboratori che li producono agli ospedali e studi medici. LOINC offre un set standard di nomi e codici per identificare risultati clinici: ogni singolo LOINC record ha un codice che può essere usato nei messaggi in HL7.

Infine, il RadioButton *Rule Engine* attiva il motore a regole, inserendo in SIB delle triple che lo risvegliano. Il motore a regole è stato realizzato da Orangee⁴ all'interno del WP3 del progetto RECOCAPE⁵ e integrato con lo Smart Space con tre differenti KP.

⁴www.orangee.com/

⁵Si veda il Capitolo 2, Sezione 2.1.

7.2.2 Il tab *Patient*

Per accedere al tab *Patient* (Figura 7.4) è sufficiente cliccare la relativa icona nella barra dei Tab. Tramite lo *Spinner* si seleziona il nome del paziente a cui si è interessati, e verranno subito visualizzate le informazioni disponibili all'interno della SIB.

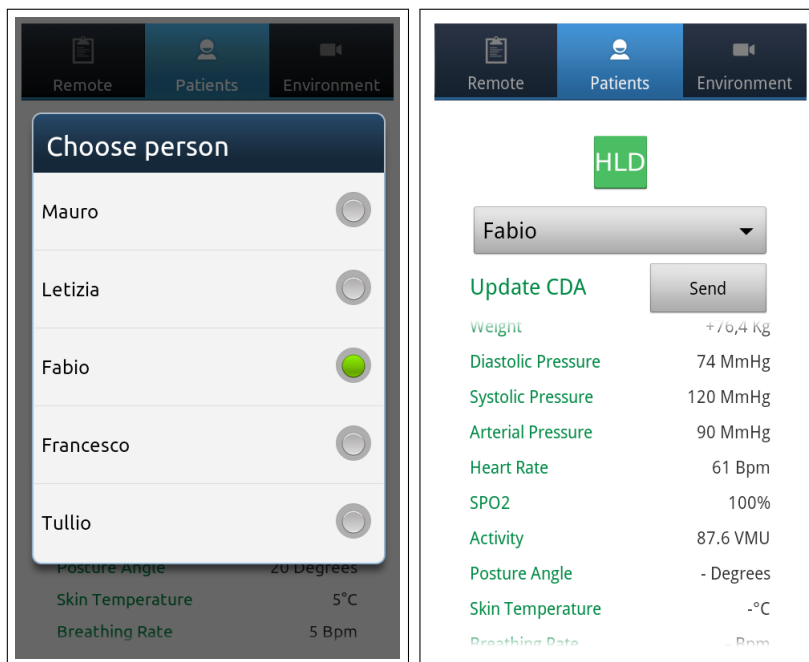


Figura 7.4: Seconda activity visualizzata - il Tab *Patient*

I parametri vitali visualizzati sono:

- Weight - Peso in kilogrammi
- Diastolic Pressure - Pressione diastolica in millimetri di mercurio (mmHg)
- Systolic Pressure - Pressione sistolica in millimetri di mercurio (mmHg)
- Mean Arterial Pressure - Pressione arteriosa media in millimetri di mercurio (mmHg)

- Heart Rate - Battito cardiaco in battiti per minuto (Bpm)
- Spo2 - Concentrazione di ossigeno nel sangue misurata in percentuale
- Activity - Attività fisica in vector magnitude units (VMU)
- Posture Angle - Angolo posturale in gradi geometrici (0° - 180°)
- Skin Temperature - Temperatura esterna della pelle misurata in gradi centigradi (°C)
- Breathing Rate - Tasso di respirazione in battiti per minuto (Bpm)

I dati vengono aggiornati in tempo reale sfruttando il meccanismo delle sottoscrizioni. Alla rilevazione di una nuova misura disponibile in SIB per il paziente correntemente selezionato, HLD avvisa l'utilizzatore tramite un *Toast* a video. Il nuovo valore viene visualizzato in tempo reale, senza bisogno di alcun intervento da parte dell'utilizzatore.

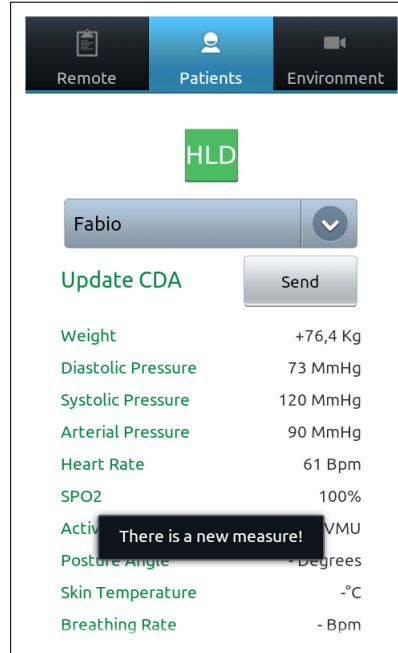


Figura 7.5: Avviso video di una nuova misura rilevata dall'applicazione

Il pulsante *Update CDA* effettua un'operazione di Update sulla SIB inserendo delle triple del tipo:

```
triples.add( sib.getXmlTools().newTriple( UpdateDB, "rdf:type", sib.getNs()
+ "UpdateDB", URI, URI) );
triples.add(sib.getXmlTools().newTriple(UpdateDB, sib.getNs()
+ "HasValue", selected_person , URI , LITERAL));
String xml=sib.getKp().insert(triples);
```

In questo modo il KP HL7Dispatcher creerà ed invierà il *Clinical Document Architecture (CDA)*[68], standard di markup per documenti clinici nonchè la prima specifica XML certificata per la sanità.

7.2.3 Il tab *Environment*

Cliccando sul Tab *Environment* si accede alle informazioni relative all'ambiente.

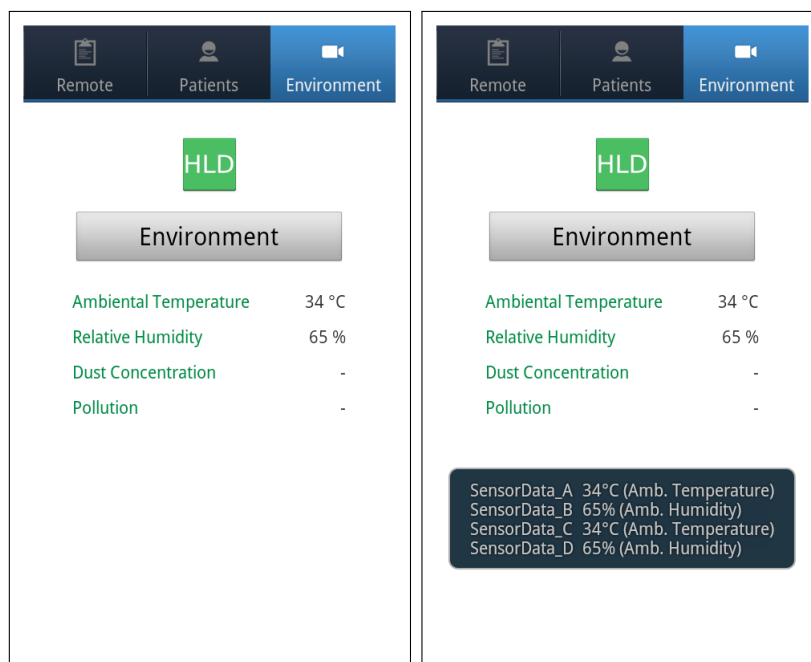


Figura 7.6: Seconda activity visualizzata - il Tab *Environment*

Nel nostro caso, i sensori disponibili per rilevare i parametri ambientali sono di quattro tipi. I parametri rilevati sono:

- Ambiental Temperature - Temperatura ambientale in gradi centigradi (°C)
- Relative Humidity - Umidità ambientale misurata in percentuale
- Dust Concentration - Concentrazione di polvere in milligrammi per metro cubo
- Pollution - Inquinamento in parti per milione (ppm)

Il tab Environment è del tutto analogo al tab Patient, con la differenza che attualmente è disponibile un solo ambiente, e non è quindi possibile la scelta fra vari Environment come per i Pazienti. Cliccando sul Button *Environment* vengono visualizzati a video i dati relativi ad ogni singolo sensore. Nel caso dell'ambiente infatti, è possibile che vi siano più sensori ad effettuare lo stesso tipo di misura. HLD mostra sempre una media pesata dei valori correntemente registrati dai sensori.

Anche nel caso dell'ambiente, HLD usa il meccanismo delle sottoscrizioni per visualizzare in tempo reale le informazioni sull'environment.

7.3 Architettura

Le funzionalità descritte implicano che Heart Life Doctor non solo gestisca un flusso di dati ma anche un flusso di controllo.

Il flusso di dati è inteso come misure, come informazioni relative ai pazienti ed è quindi la parte di pura visualizzazione di HLD, che vedremo sarà gestita interamente sfruttando le primitive offerte dalle librerie Java KPI.

Il flusso di controllo rappresenta la serie di operazioni di comando che il medico è in grado di operare tramite HLD, come inviare la cartella clinica o informare i KP esterni sui nuovi dati disponibili. La parte di controllo stimola quindi agenti esterni a HLD ed è comandata dall'utente, che usa HLD come un vero e proprio "telecomando" (*remote*).

HLD è composta da 5 activities e alcune classi Java sviluppate in supporto alla connessione e all'update.

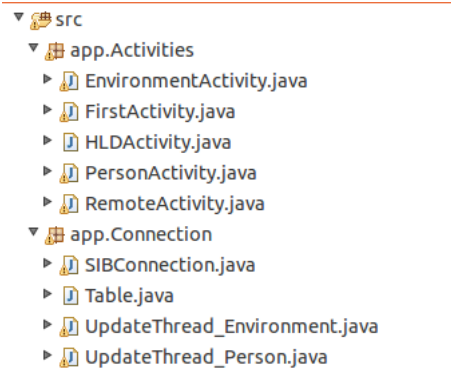


Figura 7.7: File sorgenti del progetto Heart Life Doctor

Il package *app.Activities* contiene i file java specifici di ogni singola Activity del progetto:

- **FirstActivity.java** - Corrisponde alla prima schermata visualizzata dall'utente all'avvio di HLD. Permette l'inserimento dei parametri di connessione alla SIB come visto nella sezione 7.2, poi cede il controllo a *HLDActivity*.
- **HLDActivity.java** - Activity "contenitore" dei tre tab cuore dell'applicazione: *EnvironmentActivity*, *PersonActivity*, *RemoteActivity*.
- **EnvironmentActivity.java** - Visualizza a video le informazioni relative all'ambiente, come spiegato nella sezione 7.2.3.
- **PersonActivity.java** - Visualizza in tempo reale le informazioni dei pazienti, come visto nella sezione 7.2.2
- **RemoteActivity.java** - Tramite l'uso dei RadioButton invia informazioni selezionate alla SIB, come visto nella sezione 7.2.1.

Il package *app.Connection* contiene invece classi java di supporto alla connessione e alle operazioni di aggiornamento dell'interfaccia:

- **SIBConnection.java** - Memorizza le informazioni di connessione della SIB e fornisce tutte funzionalità di connessione.
- **Table.java** - Classe di supporto al parsing dei dati ricevuti dalla SIB.
- **UpdateThread_Person.java** - Effettua la sottoscrizione al cambiamento dei valori dei dati associati alla persona e provvede a comunicarlo in tempo reale alla activity *PersonActivity*.
- **UpdateThread_Environment.java** - Effettua la sottoscrizione al cambiamento dei valori dei dati associati all'ambiente e provvede a comunicarlo in tempo reale alla activity *EnvironmentActivity*.

Le scelte progettuali sono quindi ricadute sull'avere activity separate per ogni funzionalità, piuttosto che una singola activity e più *views*. Come vedremo, questo pone dei problemi di comunicazione tra una activity e l'altra e in seguito verrà spiegato l'uso delle classi java di supporto alla comunicazione e alla connessione.

7.3.1 Gestione delle Sottoscrizioni e Update dell'interfaccia

Come detto in precedenza, il meccanismo delle sottoscrizioni permette di avere le informazioni relative al paziente e all'ambiente in tempo reale. Verrà mostrato ora il caso delle sottoscrizioni per i pazienti; le modalità che verranno spiegate sono identiche a quelle utilizzate per l'activity *Environment*.

Elenco dei pazienti

Quando l'utente clicca sul tab *Patients*, l'activity inizializza l'interfaccia associando i componenti grafici (*Spinner*, *TextView*, *Button*, etc) presenti nel

file XML di layout *personactivity.xml*. Vengono poi eseguite le funzioni per popolare lo Spinner con i nomi dei pazienti e con i dati ad essi relativi.

Qui entrano in gioco le potenzialità offerte dalle Java KPI discusse nel Capitolo 5. È sufficiente una sola query SPARQL per ottenere dalla SIB i nomi di tutti i pazienti registrati nel sistema:

```
xml = kp.querySPARQL("SELECT ?name WHERE {?person_uri <"+this.sib.getNs()+
  +"HasFullName> ?name}");
SSAP_sparql_response myresponse = new SSAP_sparql_response(xml);
Vector<Vector<String[]>> results_vector = myresponse.getResults();
```

Tramite la funzione *getResults()* si hanno a disposizione tutti i nomi da aggiungere allo Spinner.

Dati associati al paziente

Quando viene selezionata una voce dello Spinner (di default viene selezionato il primo elemento), viene eseguita la funzione:

```
void onItemClick(AdapterView<?> parent, View view, int pos, long id)
```

tramite la quale si comanda il comportamento da eseguire. Quando si seleziona un paziente, vengono per prima cosa caricati i dati disponibili in SIB del paziente stesso, poi vengono attivate le sottoscrizioni.

Per inizializzare l'interfaccia con i valori disponibili sulla SIB è sufficiente una sola query SPARQL:

```
xml=kp.querySPARQL("SELECT ?measurand ?value ?uom WHERE {{?person_uri
  <"+this.sib.getNs()+"HasFullName> '"+this.selected_person+"'}}.
  {?person_uri <"+this.sib.getNs()+"HasData> ?data_uri}.
  {?data_uri <"+this.sib.getNs()+"HasMeasurand> ?measurand}.
  {?data_uri <"+this.sib.getNs()+"HasValue> ?value}.
  {?data_uri <"+this.sib.getNs()+"HasUnitOfMeasure> ?uom}}");
```

La query restituisce la struttura dati contenente il misurando (*?measurand*), il valore (*?value*) e l'unità di misura (*?uom*) di tutti i dati associati al paziente. La query cerca, nell'ordine: l'uri associata alla persona che ha come

nome (*HasFullName*) quello del paziente scelto, cerca poi tutti i dati (*HasData*) associati a tale uri, e per ogni dato richiede il misurando (*HasMeasurand*), il valore (*HasValue*) e l'unità di misura (*HasUnitOfMeasure*).

A questo punto è sufficiente un controllo sul nome del misurando per ottenere il valore numerico associato al dato. In caso in cui il dato non sia presente viene visualizzato un trattino (-).

Sottoscrizioni

La seconda operazione effettuata dalla funzione *onItemSelected* è la creazione di un'istanza di *UpdateThread_Person*, thread che si occupa della gestione delle sottoscrizioni e aggiornamento dell'interfaccia.

```
UpdateThread_Person update = new UpdateThread_Person(Newhandler);
update.start();
```

UpdateThread_Person è una classe java contenuta nel package *app.Connection*; estende la classe *Thread* e implementa l'interfaccia *iKPIC_subscribeHandler*.

```
class UpdateThread_Person extends Thread implements iKPIC_subscribeHandler{
    //...
}
```

L'interfaccia *iKPIC_subscribeHandler* è contenuta nelle librerie Java KPI. Il suo scopo è gestire i messaggi ricevuti dalla SIB, e contiene un solo metodo da implementare, ovvero:

```
void kpic_SIBEventHandler(String xml);
```

Il funzionamento può essere brevemente spiegato così: il thread *UpdateThread_Person*, attivato dalla activity *PersonActivity*, effettua una serie di sottoscrizioni alla SIB. Quando viene inserito un nuovo dato in SIB, chi è sottoscritto viene notificato del nuovo dato. L'handler che reagisce agli eventi derivanti dalla sottoscrizione è proprio *kpic_SIBEventHandler*: a questo punto è sufficiente prelevare il nuovo dato dalla SIB e aggiornare l'interfaccia grafica di HLD. Vediamo come opera il meccanismo passaggio per passaggio.

La funzione *onItemSelected* innescata dentro *PersonActivity* quando il medico seleziona un paziente crea un'istanza del thread *UpdateThread_Person* e la avvia tramite il metodo *run()*.

UpdateThread_Person effettua l'operazione di sottoscrizione ai dati del paziente. Anche in questo caso emerge la potenza delle query SPARQL: con una sola query abbiamo tutti i dati (valore, misurando, timestamp) del paziente. I dati vengono salvati in una struttura di supporto fornita dalla classe *Table.java* realizzata ad-hoc e contenuta nel package *app.Connection*. La prima riga del codice mostra come viene impostato l'handler delle sottoscrizioni:

```
sib.getKp().setEventHandler(this);
//...
t=new Table();
String query="select ?data ?measurand where {?person";
    query+=" <"+ns+"HasFullName> '"+name+"'}. ";
    query+="{?person <"+ns+"HasData> ?data}. ";
    query+="{?data <"+ns+"HasMeasurand> ?measurand}}";
xml=sib.getKp().querySPARQL(query);

SSAP_sparql_response myresponse = new SSAP_sparql_response(xml);
Vector<Vector<String[]>> results_vector = myresponse.getResults();

if (results_vector.size()>0){
    saveResults(results_vector, t);
}
//...
```

Una volta salvati i risultati della query nella struttura *Table*, si effettua una sottoscrizione per ogni dato: una per il peso, una per il battito cardiaco, etc.

```
xml = sib.getKp().subscribeRDF(t.getData(i), "HasTimestamp", null, LITERAL);
String subID_current=sib.getXmlTools().getSubscriptionID(xml);
```

La funzione *subscribeRDF* è resa disponibile dalle Java KPI. I parametri da passare alla funzione sono, nell'ordine:

- la rappresentazione in stringa del soggetto
- la rappresentazione in stringa del predicato
- la rappresentazione in stringa dell'oggetto
- la rappresentazione in stringa del tipo dell'oggetto (gli unici valori ammessi sono *uri* o *literal*)

Per tutti i casi, il valore *null* ha il significato di “qualsiasi valore”.

Effettuare una sottoscrizione significa dire alla SIB di voler essere avvisati quando vi è un cambiamento nella tripla che ho specificato con la funzione `subscribeRDF`. In questo caso ci si sta sottoscrivendo a dei valori numerici che sono misure di parametri vitali di un paziente. È possibile sottoscrivere al cambiamento di valore del dato, oppure al cambiamento del suo timestamp. Ogni dato ha infatti associato un timestamp, noto tramite la proprietà *HasTimestamp*. All'atto pratico non sussistono reali differenze fra la sottoscrizione al valore o al timestamp, se non in un caso: quando il paziente - ad esempio - si pesa sulla bilancia e ottiene un peso corporeo identico a quello effettuato durante l'ultima misurazione la SIB non notifica l'accaduto, perchè rileva lo stesso valore numerico.

Vita di una sottoscrizione

Ogni sottoscrizione creata con la SIB ha un id univoco, che è opportuno memorizzare per tenere traccia delle sottoscrizioni attive:

```
String subID = sib.getXmlTools().getSubscriptionID(xml);
```

Ricordiamo che in questa fase stiamo attivando molteplici sottoscrizioni, più precisamente una sottoscrizione per ogni dato associato al paziente. È opportuno che tutte le sottoscrizioni vengano chiuse nel momento in cui il medico sceglie di monitorare un altro paziente, anche per non sovraccaricare la SIB con decine di sottoscrizioni aperte.

Gli id univoci associati alle sottoscrizioni vengono quindi passati tramite il metodo *notifyMessage*⁶ a `PersonActivity`: quando il medico seleziona un nuovo paziente, la activity è in grado di chiudere autonomamente le sottoscrizioni del “vecchio paziente” e scatenare la chiamata di un nuovo `UpdateThread_Person` per il nuovo paziente selezionato.

⁶Per i meccanismi di comunicazione fra `Thread` e `Activity` si veda la sezione 7.3.1

Per annullare una sottoscrizione è sufficiente chiamare il metodo seguente, offerto dalle Java KPI, passando come parametro l'id univoco associato alla sottoscrizione che vogliamo chiudere.

```
public String unsubscribe(String subscription_id){
//...
}
```

Aggiornamento dell'interfaccia

Come abbiamo detto, l'handler che gestisce il comportamento dell'applicazione al momento della ricezione di un nuovo dato è rappresentato dal metodo *kpic_SIBEventHandler*.

```
public void kpic_SIBEventHandler(String xml)
{
    Vector<Vector<String>> triples = sib.getXmlTools().
                                   getResultEventTriple(xml);
    //...
}
```

Questa funzione non fa altro che chiamare il metodo *getResultEventTriple(xml)*, disponibile nelle Java KPI. Il parametro passato dev'essere il messaggio ricevuto dalla SIB come notifica, mentre il parametro di ritorno è una struttura riempita con ogni nuova tripla disponibile in SIB.

```
//...
String m=t.getMeasurandFromData(dat);

int me=0;

if(m.equals("Weight")){
    me=WEIGHT;
}else if(m.equals("DiastolicPression")){
    me=DIASTOLIC_PRESSION;
}else if(m.equals("SystolicPression")){
    me=SYSTOLIC_PRESSION;
}else if(m.equals("MeanArterialPression")){
    me=ARTERIAL_PRESSION;
}else if(m.equals("HeartRate")){
    me=HEART_RATE;
}else if(m.equals("Spo2")){
    me=SPO2;
}else if(m.equals("Activity")){
    me=ACTIVITY;
}else if(m.equals("PostureAngle")){
    me=POSTURE_ANGLE;
}else if(m.equals("SkinTemperature")){
    me=SKIN_TEMPERATURE;
}else if(m.equals("BreathingRate")){
    me=BREATH_RATE;
```

```
}  
if(val!=null){  
    notifyMessage(me, val);  
}  
//...
```

Dalla struttura dati appena ricevuta - dopo alcuni passaggi di parsing della struttura dati omessi per concisione - si estrae il valore corrispondente al misurando del dato (*Weight*, *DiastolicPression*, etc), lo si incapsula in un messaggio e tramite il metodo *notifyMessage* lo si “invia” a *PersonActivity*. Per identificare univocamente ogni misurando vi è stato associato un numero intero anzichè una stringa, come richiesto dal metodo *obtainMessage*.

```
private void notifyMessage(int i, String str)  
{  
    byte[] tbuf = str.getBytes();  
    myHandler.obtainMessage(i,tbuf.length, -1 ,tbuf).sendToTarget();  
}
```

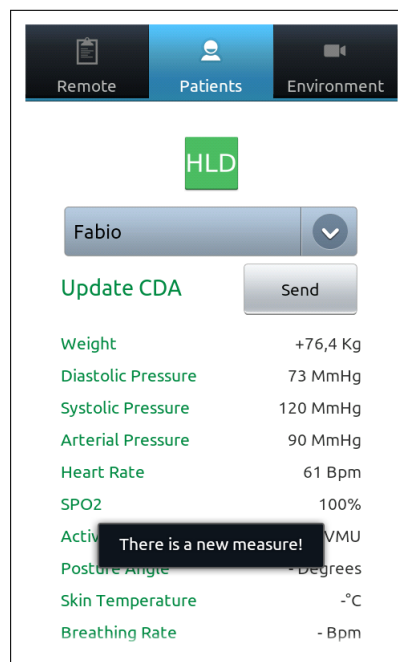


Figura 7.8: Toast che avverte della presenza di una nuova misura

Il messaggio viene recapitato all'handler di PersonActivity: ora è possibile aggiornare l'interfaccia grafica con il nuovo valore. Ad esempio, nel caso del peso:

```
//recupero la textview che avevo associato al peso
TextView tv = (TextView)findViewById(R.id.textViewWeight);

//vi imposto il nuovo valore (la variabile val estratta dal messaggio)
tv.setText(val+" Kg");

//creo un messaggio toast da visualizzare a video
Toast.makeText(this, "There is a new measure!", Toast.LENGTH_SHORT).show();
```

Il tempo di aggiornamento di Heart Life Doctor è praticamente in tempo reale. I test effettuati mostrano una rapidità di aggiornamento istantanea sia nel caso in cui le misure vengano aggiornate “manualmente” (è sufficiente inserire una nuova tripla nella SIB contenente il valore fittizio di misurazione), sia nel caso in cui il paziente effettui da solo e con gli strumenti di laboratorio la misura.

Questi risultati sono dovuti principalmente alla buona programmazione dell'architettura della SIB, alla buona gestione dei protocolli di comunicazione fra SIB e KP, nonché alle ottime prestazioni delle librerie Java KPI e delle strutture dati snelle scelte per raccogliere i dati.

7.4 Robustezza

Heart Life Doctor è a tutti gli effetti un'applicazione che si serve di un servizio remoto, e per usufruire di tale servizio è necessario collegarsi a chi lo eroga. Non sarebbe possibile effettuare alcuna operazione senza il collegamento alla SIB, che in questo caso è proprio il fornitore del servizio.

Si rende quindi necessario mettere a punto un meccanismo che controlli la validità della connessione alla SIB e che abbia un comportamento *safe* in caso di connessione non riuscita. Altrettanto importante è controllare che non ci siano cadute di connessione durante l'utilizzo di Heart Life Doctor e, in caso di mancato collegamento, adottare il comportamento più opportuno.

7.4.1 Connessione all'avvio

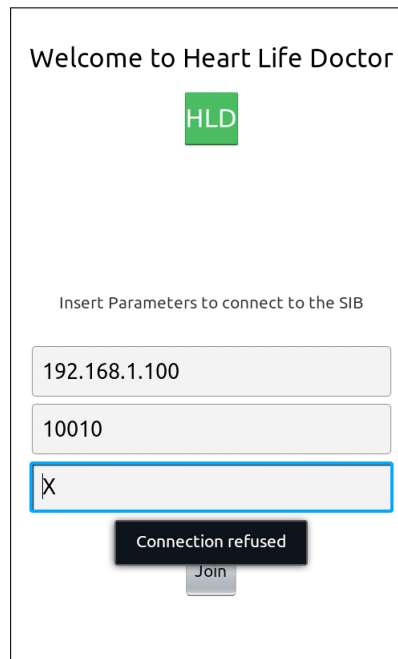


Figura 7.9: In caso di connessione rifiutata, HLD invita a re-inserire i dati

In caso in cui i dati inseriti siano scorretti, o comunque la connessione non vada a buon fine, Heart Life Doctor presenta un messaggio a video in cui si invita l'utente a re-inserire i dati. A questo proposito non è possibile dare informazioni sulla natura dell'errore, perchè le primitive di connessione offerte dalle librerie Java KPI al momento dello sviluppo di HLD sono in grado di fornire solo un dato di tipo booleano (“connessione riuscita” / “connessione non riuscita”).

```
public void onCreate (Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.personactivity);

    associate();
    getSibPrefs();

    if(controller_join()){
        //...
    }
}
```

```
    }else{
        Toast.makeText(this, "Connection refused",
            Toast.LENGTH_SHORT).show();
        finish();
    }
    //...
}
//...
private boolean controller_join()
{
    //...
    KPICore kp = new KPICore(HOST, Integer.parseInt(PORT), SSNM);
    SSAP_XMLTools xmlTools = new SSAP_XMLTools(null,null,null);
    String xml = kp.join();
    return xmlTools.isJoinConfirmed(xml);
}

public boolean isJoinConfirmed(String xml)
```

La funzione *isJoinConfirmed* è contenuta nelle Java KPI ed è fornita da *SSAP_XMLTools.java*, la classe che gestisce tutti i messaggi provenienti dalla SIB.

7.4.2 Controllo attivo di connessione

Una caduta di connessione con la SIB è un evento che può accadere ad esempio in caso di guasto sulla connessione di rete o wireless. Heart Life Doctor dispone di un meccanismo di controllo di connessione attivo durante tutto il ciclo di vita dell'applicazione.

Le informazioni strettamente necessarie alla connessione sono tre:

- l'indirizzo IP della SIB
- il numero di porta
- il nome dello Smart Space (se esistente)

Queste informazioni sono inserite manualmente dall'utente all'avvio di Heart Life Doctor e devono essere note e disponibili durante tutto il ciclo di vita dell'applicazione.

Memorizzazione delle variabili di connessione come Shared Preferences

Le informazioni di connessione vengono memorizzate dall'applicazione come **Shared Preferences**. Android mette a disposizione un framework che consente di gestire contemporaneamente sia la logica per l'accesso e la persistenza della configurazione di una applicazione che la logica per la presentazione dell'interfaccia utilizzata per la modifica dei parametri della stessa. I parametri di configurazione sono memorizzati in uno o più file gestiti automaticamente dal framework. L'interfaccia che consente la lettura e la scrittura delle preferenze di un'applicazione è l'interfaccia *android.content.SharedPreferences*. La classe *android.content.SharedPreferences* permette allo sviluppatore di salvare dei settaggi applicativi in un file e condividerli nella applicazione stessa o tra tutte le applicazioni. La classe *SharedPreferences* consente di memorizzare coppie chiave-valore; in particolare la chiave della coppia è sempre una *String* mentre il valore può essere un tipo primitivo (*boolean*, *int*, *float*, *long*) o un oggetto di tipo *String*. I valori sono memorizzati fisicamente in file *xml* e in genere sono disponibili anche dopo un riavvio dell'applicazione o del telefono.

I valori di *host*, *porta* e *Smart Space name* vengono memorizzati come *Shared Preferences* nella prima *activity* di HLD (*FirstActivity*) e sono così disponibili in modo persistente anche alle altre *activities* e/o classi *Java* che effettuano i controlli di connessione sulla *SIB*.

```
import android.content.SharedPreferences;

//...

public void onCreate(Bundle savedInstanceState)
{
    //...

    joinButton.setOnClickListener(new OnClickListener()
    {
        @Override
        public void onClick(View v)
        {
            setSharedPreferences();
        }
    });

    //...
}
```

```
        }
    });
}
//...
private void setSharedPreferences()
{
    SharedPreferences sibPrefs = getSharedPreferences("SibPreferences", 0);

    SharedPreferences.Editor prefEditor = sibPrefs.edit();

    prefEditor.putString("Host", HOST.getText().toString());
    prefEditor.putString("Port", PORT.getText().toString());
    prefEditor.putString("NameSpace", HOST.getText().toString());

    prefEditor.commit();
}
//...
```

In seguito alla pressione del bottone *Join* visibile sulla prima schermata di HLD, le variabili inserite dall'utente vengono memorizzate prima di effettuare l'operazione di connessione.

Le coppie chiave-valore da memorizzare sono inserite tramite l'interfaccia *android.content.SharedPreferences.Editor* tramite il metodo *putString*. Per ottenere un'istanza della classe che implementa questa interfaccia è necessario utilizzare il metodo *edit()* dell'interfaccia *SharedPreferences*. Successivamente è possibile modificare i parametri utilizzando i metodi dell'interfaccia *Editor*.

Tutte le modifiche effettuate saranno solo temporanee finché non sarà chiamato il metodo *commit()* della stessa interfaccia.

Quando un'altra activity necessita dei parametri di connessione, è sufficiente invocare i metodi:

```
SharedPreferences sibPrefs = getSharedPreferences("SibPreferences", 0);
String host = sibPrefs.getString("Host", "0");
String port = sibPrefs.getString("Port", "10010");
String nameSpace = sibPrefs.getString("NameSpace", "X");
```

Il controllo attivo

Il meccanismo di controllo attivo è effettuato da un thread Java e si basa sui meccanismi della programmazione concorrente. La funzione `Thread.sleep(long time)` fa sì che il thread corrente sospenda l'esecuzione per un periodo specificato dal valore del parametro `time`.

Si tratta di un mezzo efficace per rendere il processore a disposizione degli altri thread di un'applicazione o di altre applicazioni che potrebbero essere in esecuzione.

La funzione `Thread.sleep()` non utilizza cicli del processore ma è un metodo statico che mette in pausa il thread corrente. Tramite `Thread.sleep()` non è possibile mettere in pausa un altro thread.

Si tenga presente che la sospensione del thread per mezzo di una `sleep` può essere pericolosa nel caso in cui si implementi una qualche gestione sincronizzata delle variabili, dato che non rilascia gli eventuali lock acquistati dal thread. In questo caso, il thread di controllo di connessione non richiede il lock di alcuna variabile ed effettua un semplice test di connessione sfruttando le primitive rese disponibili dalle Java KPI.

```
//...
if(sib.getXmlTools()!=null){
    boolean ack = sib.getXmlTools().isJoinConfirmed(xml);
}
//...
try {
    Thread.sleep(millis);
    //...
} catch (InterruptedException e){
    //...
}
```

La variabile `millis` esprime l'intervallo di tempo di `sleep` in millisecondi, prima del prossimo controllo di connessione. Sono stati eseguiti test di connessione con diversi valori della variabile `millis`, dal valore di un secondo al valore di diversi minuti. Anche sotto le condizioni temporali più stringenti, Heart Life Doctor ha mostrato un comportamento fluido e corretto.

Nel caso in cui la connessione alla SIB non fosse più attiva, Heart Life Doctor mantiene inoltre un comportamento *safe*:

- effettua almeno un tentativo di chiusura delle sottoscrizioni attive, se presenti;
- effettua almeno un tentativo di leave dalla SIB;
- avvisa l'utente della perdita di connessione;
- termina tutte le activity e riporta l'utente alla schermata iniziale, invitandolo a ri-effettuare la connessione.

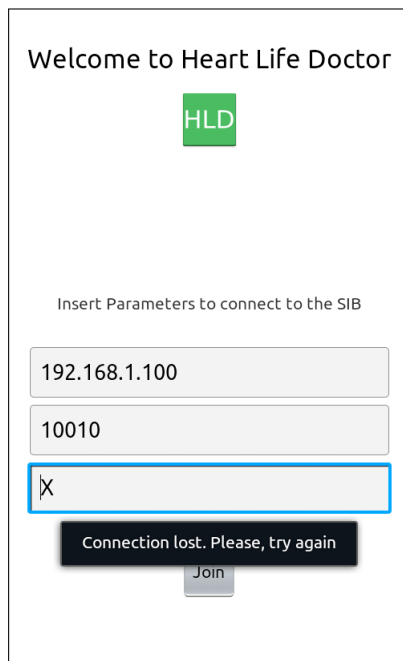


Figura 7.10: Gestione della perdita di connessione

Qualora la connessione fosse davvero impossibile da ottenere, neanche per effettuare le operazioni di leave e unsubscribe, Heart Life Doctor riporta comunque l'utente alla schermata iniziale senza mai andare in crash e senza mai mostrare comportamenti anomali.

7.5 Prossime Release

Heart Life Doctor soddisfa tutti i requisiti richiesti da Arces MARS per lo sviluppo. HLD mostra in tempo reale i parametri vitali del paziente e i parametri ambientali, consente al medico di inoltrare la cartella clinica e informare il servizio web. HLD mostra tempi di reazione molto buoni in fase di connessione e soprattutto in fase di aggiornamento dei dati, che avviene praticamente in tempo reale con la misurazione.

I prossimi sviluppi dell'applicazione procederanno di pari passo con le modifiche effettuate all'ontologia dello Smart Space. Introdurre il concetto di *appartenenza* di un paziente all'ambiente (environment) comporterà sicuramente l'aggiornamento delle activity Environment e Patients per aggiungere le nuove informazioni.

Un'altra modifica può essere pensata per la schermata iniziale in cui avviene la connessione. Un meccanismo di discovery per le SIB che fornisca all'utente un elenco delle SIB disponibili alla connessione potrebbe sostituire le tre form in cui vengono inseriti manualmente i parametri per connettersi. Interessante è immaginare la possibilità di visualizzare informazioni di traffico e/o di carico per la SIB, ad esempio quanti pazienti essa stia gestendo o quanti medici sono collegati in un certo momento. Questa prospettiva ovviamente comporta tutto un altro scenario di progettazione di un sistema inter-SIB o intra-SIB al quale Heart Life Doctor chiederebbe solo informazioni di traffico senza effettuare alcuna operazione decisionale.

Un'ultima possibilità è quella di sviluppare Heart Life Doctor anche per dispositivi tablet, e continuarne lo sviluppo per le nuove versioni di Android quali Jelly Bean.

Conclusioni

La trasformazione di un ambiente fisico in Cyber Physical Ecosystem è un processo che coinvolge molte tecnologie software e dispositivi hardware, e richiede che tutti gli elementi in esso presenti siano interoperabili e perciò adottino dei protocolli sviluppati per riflettere la natura profondamente semantica del sistema.

Le tecnologie messe a punto dall'infrastruttura del Web Semantico sono fondamentali per sviluppare un sistema come quello richiesto: RDF e OWL consentono di strutturare l'informazione, renderla machine-understandable, classificarla e consentono ad agenti software di svolgere operazioni di *reasoning* su di essa.

L'ontologia sviluppata nel capitolo 4 realizza la knowledge base della piattaforma Open Chiron Smart Space. Tramite la piattaforma, i diversi dispositivi fisici per il monitoring del paziente e dell'ambiente possono interfacciarsi al sistema e iniziare la raccolta di dati ambientali e fisici.

I dispositivi fisici operano tramite tecnologia Bluetooth e non presentano alcuna interfaccia al paziente. In questo modo è così davvero facile per il paziente usare i dispositivi fisici ed effettuare misurazioni, senza dover utilizzare complicate user interfaces che possono confondere un paziente in stato di salute cagionevole o anziano. Tramite il protocollo Bluetooth, i dispositivi inviano direttamente i dati raccolti alla SIB senza bisogno di alcun intervento da parte dell'utente.

La SIB si interfaccia con altri dispositivi e permette la raccolta e la manipolazione dei dati tramite Knowledge Processors (KP), tramite librerie KPI

sviluppate in linguaggio Java, C, C#, Python etc, e da oggi anche tramite l'applicazione android Heart Life Doctor.

L'operazione di trasformazione da sistema puramente fisico, in cui una miriade di agenti dovevano interfacciarsi tra di loro e con gli utenti tramite protocolli diversi e non interoperabili è completa. Open Chiron Smart Space è un Cyber Physical Ecosystem che ha raggiunto gli obiettivi di interoperabilità, di facilità d'uso da parte di personale medico e paziente. Inoltre, è ridotto il tempo che il personale medico deve spendere per il paziente, garantendo al contempo un alto livello di qualità usufruendo di informazioni disponibili in tempo reale e sul dispositivo smartphone del medico. Infine, viene realizzata proprio quell'infrastruttura in cui flusso di dati e flusso di controllo coesistono e cooperano all'interno di un sistema intelligente e in grado di prendere decisioni.

La validità del progetto è dimostrata anche dalle conferme ottenute all'esterno dell'ambito strettamente didattico: il corpo di lezioni del progetto Europeo RECOCAPE si terranno ad Ottobre 2012 a Il Cairo, in Egitto; Open Chiron Smart Space è inserito nei due progetti Europei SOFIA e CHIRON. Le librerie Java KPI sono già disponibili su Sourceforge insieme alla release ufficiale dei codici sorgenti di Open Chiron Smart Space.

Infine, Heart Life Doctor è stato presentato durante la Conferenza Internazionale sulle Smart Homes ed Health Telematics di Firenze nei giorni 12-15 Giugno 2012, raccogliendo impressioni molto positive.

In conclusione, il lavoro svolto a supporto di Open Chiron Smart Space è stabile ed è già stato interamente rilasciato per lo sviluppo e l'installazione da parte di strutture mediche e ospedaliere. Il sistema può certamente essere esteso ulteriormente per introdurre nuovi servizi e supportare altre piattaforme hardware e/o software, come accennato anche nei Capitoli 4, 5, 7.

Nuove soluzioni sono già state discusse con il Prof. Salmon Cinotti, relatore di questo lavoro di tesi, e con il gruppo di lavoro del laboratorio Arces Mars ed in parte sono già in fase di implementazione.

Bibliografia

- [1] “Semantic Web Technologies” - Brian Matthews, JISC Technology and Standards Watch, CCLRC Rutherford Appleton Laboratory, World Wide Web Consortium (W3C)
- [2] “Intelligent Agents Meet the Semantic Web in Smart Spaces” - H. Chen, T. Finin, A. Joshi, L. Kagal, IEEE press, Oct Nov 2004
- [3] “Interoperability Standards in the Semantic Web” - Steven R. Ray, Journal of Computing and Information Science in Engineering
- [4] “Foundations of Semantic Web Technologies” - P. Hitzler, M. Krotzsch, S. Rudolph, Chapman & Hall/CRC, 2010
- [5] “Bridging Multi Agent Systems and Web Services: towards interoperability between Software Agents and Semantic Web Services” - M. Omair Shafiq, Ying Ding, Dieter Fensel, Digital Enterprise Research Institute (DERI)
- [6] W3C Talks “The Current Web” - Eric Miller, <http://www.w3.org/2004/Talks/0120-semweb-umich/slide5-0.html>
- [7] W3C Talks “Semantic Web Technologies” - Ivan Herman, <http://www.w3.org/2004/Talks/SpanishTour-IH/Overview.html>
- [8] “Linked Data – Evolving the Web into a Global Data Space” - T. Heath, C. Bizer, Morgan & Claypool Publishes, 2011

-
- [9] “Who is afraid of the GGG?” - P. Miller, Nodalities Blog, available online at http://blogs.talis.com/nodalities/2007/11/who_is_afraid_of_the_ggg.php
- [10] “Tutorial on Semantic Web”, Ivan Herman, W3C, 2011. Available online at <http://www.w3.org/People/Ivan/CorePresentations/SWTutorial/>
- [11] Grigoris Antoniou, Frank van Harmelen (March 31, 2008). A Semantic Web Primer. Mit Press, 31/mar/2008
- [12] John Davies (July 11, 2006). Semantic Web Technologies: Trends and Research in Ontology-based Systems. Wiley. ISBN 0-470-02596-4.
- [13] Thomas B. Passin (March 1, 2004). Explorer’s Guide to the Semantic Web. Manning Publications. ISBN 1-932394-20-6.
- [14] Liyang Yu (June 14, 2007). Introduction to Semantic Web and Semantic Web Services. CRC Press. ISBN 1-58488-933-0.
- [15] Jeffrey T. Pollock (March 23, 2009). Semantic Web For Dummies. For Dummies. ISBN 0-470-39679-2
- [16] “Internet: 2,2 miliardi di utenti nel 2013” http://www.key4biz.it/News/2009/07/21/e\discretionary{-}{-}{-}Society/forrester_internet_Zia_Daniell_Wigder.html
- [17] Berners-Lee T., Hendler J., Lassila O.: The Semantic Web, Scientific American, May 2001, <http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html>
- [18] R. Mark, Telemedicine system: the missing link between homes and hospitals?, Mod Nurs Home, 1974 Feb, 32(2), pp. 39-42
- [19] J. P. Finley, D. G. Human, M. A. Nanton, D. L. Roy, R. G. Macdonald, D. R. Marr, H Chiasson, Echocardiography by telephone - Evaluation of

- pediatric heart disease at a distance, *The American Journal of Cardiology*, Volume 63, Issue 20, 15 June 1989, Pages 1475-1477, ISSN 0002-9149, DOI: 10.1016/0002-9149(89)900118
- [20] L. Jiang, D. Liu, B. Yang, Smart home research, *Proceedings of the 2004 International Conference on Machine Learning and Cybernetics*, Vol. 2, pp. 659- 663, August 2004
- [21] N. Noury, G. Virone, P. Barralon, J. Ye, V. Rialle, J. Demongeot, New trends in health smart homes, *Proceedings of the 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (Healthcom 2003)*, pp. 118 – 127, June 2003
- [22] Mainstreaming on Ambient Intelligence (MonAMI) project, Available: <http://www.monami.info>
- [23] Smart Medical Home Research Laboratory, Center for Future Health, University of Rochester, Available: <http://www.futurehealth.rochester.edu>
- [24] C. Kidd, R. Orr, G. Abowd, C. Atkeson, I. Essa, B. MacIntyre, E. Mynatt, T. Starner, W. Newstetter, The Aware Home: A Living Laboratory for Ubiquitous Computing Research, *Proceedings of the 2nd International Workshop on Cooperative Buildings, Integrating Information, Organization, and Architecture*, pp. 191-198, October 1999
- [25] F. Knoefel, V. Emerson, B. Schulman, TAFETA: An Inclusive Design for Tele-Health, *Technology and Persons with Disabilities Conference*, March 2005
- [26] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, E. Jansen, The Gator Tech Smart House: A Programmable Pervasive Space, *IEEE Computer*, Vol. 38, No. 3, pp. 50-60, March 2005
- [27] Home Depot Smart Home, Pratt School of Engineering, Duke University, Available: <http://www.smarthome.duke.edu>

-
- [28] P. Bartolomeu, J. Fonseca, V. Santos, A. Mota, V. Silva, M. Sizenando, Automating Home Appliances for Elderly and Impaired People: The B-Live Approach, Proceedings of the 2nd International Conference on Software Development for Enhancing Accessibility and Fighting Info-exclusion (DSAI2007), November 2007
- [29] E. Lee. Cyber physical systems: Design challenges. In Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on, pages 363-369. IEEE, 2008.
- [30] World Wide Web Consortium. Extensible Markup Language (XML) 1.0 Second Edition. W3C Recommendation. 6 October 2000. Available on-line at <http://www.w3.org/TR/REC-xml>
- [31] World Wide Web Consortium. Namespaces in XML. W3C Recommendation. 14 January 1999. Available on-line at <http://www.w3.org/TR/REC-xml-names>
- [32] World Wide Web Consortium. XML Schema: Part 1: Structures. W3C Recommendation. 2 May 2001. Available on-line at <http://www.w3.org/TR/>
- [33] World Wide Web Consortium. XML Schema: Part 2: Datatypes. W3C Recommendation. 2 May 2001. Available on-line at <http://www.w3.org/TR/>
- [34] World Wide Web Consortium. Document Object Model (DOM) Level 2 Core Specification Version 1.0. W3C Recommendation. 13 November 2000. Available on-line at <http://www.w3.org/TR/DOM-Level-2-Core/>
- [35] I. Bartolini, P. Bellavista – Corso di “Tecnologie Web”, Alma Mater Studiorum Università di Bologna, Corso di Laurea in Ingegneria Informatica. Available online at <http://www-db.deis.unibo.it/courses/TW/>

-
- [36] “Sofia, M3 Smart Space Infrastructure” - J. P. Soininen, A. Lappeteläinen, Nokia
- [37] SOFIA (Smart Object For Intelligence Applications), Available: <http://www.sofia-project.eu/>
- [38] “The Components of a Smart Space Platform for Smart Service Deployment” - D. Cawley, Telecommunications Software and Systems Group Waterford Institute of Technology, 2003
- [39] “Smart-M3 Information Sharing Platform” - J. Honkola, H. Laine, R. Brown, O. Tyrkko, 2010, Nokia Research Center
- [40] Smart-M3 release, <http://sourceforge.net/projects/smart-m3/> Referenced March 4th 2010
- [41] Fabio Vergari, “Sistemi per il monitoraggio concorrente di parametri biometrici e ambientali finalizzato alla valutazione di situazioni critiche”, Tesi di Dottorato di ricerca in Bioingegneria, Università di Bologna, 2011
- [42] CHIRON (Cyclic and person-centric Health management: Integrated appRoach for hOme, mobile and clinical eNvironments) Project. Available online at <http://www.chiron-project.eu>
- [43] L. L. Brownsword, D. J. Carney, D. Fisher, G. Lewis, C. Meyers, Current perspectives on interoperability, Technical Report, CMU/SEI-2004-TR-009, Carnegie Mellon University, March, 2004
- [44] D. Cook and S. Das, Smart Environments: Technology, Protocols and Applications (Wiley Series on Parallel and Distributed Computing). Wiley-Interscience, 2004
- [45] J. Honkola, H. Laine, R. Brown, O. Tyrkko, Smart-M3 information sharing platform, Computers and Communications (ISCC), 2010 IEEE Symposium on, vol., no., pp.1041-1046, 22-25 June 2010

-
- [46] Smart-M3, Wikipedia, Available: <http://en.wikipedia.org/wiki/Smart-M3>
 - [47] Smart-M3, public source code, Available: <http://sourceforge.net/projects/smart-m3/>
 - [48] Resource Description Framework, <http://www.w3.org/RDF/> Referenced March 4th 2010
 - [49] World Wide Web Consortium. RDF Primer. W3C Recommendation 10 February 2004. Available on-line at <http://www.w3.org/TR/rdf-primer/>
 - [50] World Wide Web Consortium. W3C Semantic Web Frequently Asked Questions. Available on-line at <http://www.w3.org/RDF/FAQ>
 - [51] World Wide Web Consortium. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation 10 February 2004. available online at <http://www.w3.org/TR/rdf-schema/>
 - [52] “Guide to the Resource Description Framework” - Renato Iannella, The New Review of Information Networking, Vol 4, 1998
 - [53] OWL web ontology language, <http://www.w3.org/standards/techs/owl> Referenced March 4th 2010
 - [54] “A Practical Guide To Building OWL Ontologies Using The Protege-OWL Plugin and CO-ODE Tools” - Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, Chris Wroe, The University Of Manchester, Stanford University, 2004
 - [55] “Building OWL Ontologies with Protege” - INFO 4302 – November 1, 2011 Carl Lagoze – Cornell University
 - [56] Dean Allemang, James Hendler (May 9, 2008). Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL. Morgan Kaufmann. ISBN 978-0-12-373556-0.

- [57] D. Manzaroli, “Architectures for Context Aware Services in Smart Environments”, Tesi di Dottorato di ricerca in Information Technology, Università di Bologna, 2009
- [58] SPARQL Query Language, <http://www.w3.org/TR/rdf-sparql-query/>
- [59] SPARQL Query Results Format, <http://www.w3.org/TR/rdf-sparql-XMLres/>
- [60] SPARQL Protocol for RDF, <http://www.w3.org/TR/rdf-sparql-protocol/>
- [61] Graph Matching, <http://www.w3.org/TR/rdf-sparql-query/#sparqlBGPEExtend>
- [62] Open Handset Alliance. Open Handset Alliance. <http://www.openhandsetalliance.com/>
- [63] Google, Documentazione ufficiale Android, <http://developer.android.com>
- [64] IDC - International Data Corporation press releases <http://www.idc.com/getdoc.jsp?containerId=prUK23507512>
- [65] Carlo Pelliccia, *Programmazione android*. www.informatica.uniroma2.it/upload/2009/LIS/, 2009.
- [66] Massimo Carli, *Android: guida per lo sviluppatore*, Apogeo Editore Srl, 2010.
- [67] Android Developers, <http://developer.android.com/>
- [68] Clinical Document Architecture (CDA) e HL7, <http://www.hl7.com.au/FAQ.htm>