ALMA MATER STUDIORUM
UNIVERSITÀ DEGLI STUDI DI BOLOGNA
**CAMPUS DI CESENA**
**SCUOLA DI INGEGNERIA E ARCHITETTURA**

**CORSO DI LAUREA MGISTRALE IN INGEGNERIA INFORMATICA**

# *Implementazione e validazione di una versione trasportabile su piattaforma Android di un broker semantico OSGI*

**TESI DI LAUREA IN SISTEMI DI ELABORAZIONE DELL'INFORMAZIONE LM**

TESI DI LAUREA DI:
Ali Nadjoui

RELATORE:
Prof. Ing. Tullio Salmon Cinotti

CORRELATORE:
Ing. Alfredo D'Elia

Anno Accademico 2012/13

Sessione III

# Indice


**INTRODUZIONE** 4

**CAPITOLO 1** 7

**SCENARIO TECNOLOGICO** 7

**1.1 IL "CONTESTO" ED IL SEMANTIC WEB** 7

**1.2 SOFIA E LA SIB** 11

**CAPITOLO 2** 13

**SIB-O** 13

**2.1 STRUTTURA INTERNA** 13

**2.2 FUNZIONALITÀ** 15

**CAPITOLO 3** 19

**IL MIO CONTRIBUTO** 19

**3.1 OBIETTIVI DEL PASSAGGIO DALL'ARCHITETTURA OSGI ALL' ARCHITETTURA SIB MANAGER:** 19

**3.2 LAVORO SVOLTO** 20

**3.3 ANALISI DEL PASSAGGIO DALL'ARCHITETTURA OSGI ALL' ARCHITETTURA SIB MANAGER** 21

**3.3.1 Servizi vs Processi Java (Thread) :** 21

**3.3.2 Coerenza nell' interazione fra i processi** 22

**3.3.3 Accesso concorrente ai dati condivisi** 22

**3.3.4 Soluzione proposta** 23

**3.3.5 SIB-J** 23

**3.4 ARCHITETTURA DELLA SIB-J** 24

**3.5 SIB SU ANDROID** 25

**3.5.1 Esecuzione del porting su Android** 26

**CAPITOLO 4** 29

**ANALISI DELLE PRESTAZIONI** **29**

**4.1 SCENARI DI TEST** 29

**4.2.1 – SIB-J v1** 30

**4.3 SIB OTTIMIZZATA E SIB ANDROID** 31

**CONCLUSIONI** **34**

**BIBLIOGRAFIA** **35**

# Indice delle figure

FIGURA 1: LA STRUTTURA E LE TECNOLOGIE DEL WEB SEMANTICO .....8

FIGURA 2: RAPPRESENTAZIONE DEI PIÙ IMPORTANTI REPOSITORY RDF NEL 2008 .....10

FIGURA 3: ARCHITETTURA SOFTWARE BASATA SU SIB .....12

FIGURA 4: ARCHITETTURA DELLA SIB OSGI .....15

FIGURA 5: DOMAIN MODEL DELL'ARCHITETTURA SIB MANAGER. .....24

FIGURA 6: DIAGRAMMA DI CLASSI DEL SIB CORE E SIB MANAGER .....25

FIGURA 7: DOMAIN MODEL SIB ANDROID .....28

FIGURA 8: DIAGRAMMA DELLE CLASSI DELLA SIB ANDROID .....28

FIGURA 9: CONFRONTO TEMPO DI INSERIMENTO SENZA SOTTOSCRIZIONI SIB-J V1 E SIB-O .....30

FIGURA 10: CONFRONTO TEMPO DI INSERIMENTO DI 20 TRIPLE CON SOTTOSCRIZIONI SIB-J V1 E SIB-O .....31

FIGURA 11: CONFRONTO TEMPO DI INSERIMENTO DI 200 TRIPLE CON SOTTOSCRIZIONI SIB-J V1 E SIB-O .....31

FIGURA 12: CONFRONTO TEMPO DI INSERIMENTO SENZA SOTTOSCRIZIONI SIB IMPLEMENTATE E SIB-O .....32

FIGURA 13: CONFRONTO TEMPO DI INSERIMENTO DI 20 TRIPLE CON SOTTOSCRIZIONI SIB IMPLEMENTATE E SIB-O .....32

FIGURA 14: CONFRONTO TEMPO DI INSERIMENTO DI 200 TRIPLE CON SOTTOSCRIZIONI SIB IMPLEMENTATE E SIB-O .....33

# Introduzione

In questo lavoro di tesi mi sono occupato della realizzazione di un database semantico puramente in Java a partire da una sua versione in tecnologia OSGI. Le ragioni per cui questo è importante sono molto legate al campo di utilizzo: il mondo del web semantico e dell'intelligenza ambientale fornita da agenti software autonomi.

La visione del "pervasive computing" che Mark Weiser ebbe nel 1991, si sta trasformando in realtà nei nostri giorni grazie ai recenti sviluppi teorici e tecnologici; egli immaginò la realizzabilità di spazi in cui l'elettronica fornisse servizi, pur rimanendo invisibile e fornendo quindi l'illusione che fosse l'ambiente stesso a capire i bisogni dell'utenza e ad agire di conseguenza. Oltre alla miniaturizzazione sono necessari anche altri tipi di tecnologie per consentire ai diversi attori, operanti in un ambiente intelligente, di comprendersi e cooperare. Le tecnologie del Semantic WEB, che aggiungono il concetto di semantica dell'informazione e si appoggiano sui comunissimi e supportati formalismi non semantici, sono fra i maggiori candidati a divenire standard di rappresentazione dell'informazione contestuale quando essa deve essere condivisa e compresa da un ampio raggio di applicazioni.

La SIB ( Semantic Information Broker ) è un middleware per gestire e consentire l'accesso ad una base informativa semantica. Venne concepita e sviluppata a partire dal 2008 nell'ambito del progetto Europeo SOFIA, poi a partire dal 2011 è stata mantenuta e potenziata da una attiva comunità di cui è parte l'università di Bologna, la quale ne ha rilasciato diverse versioni. Una grande varietà di scenari è implementabile sulla base di informazioni in forma semantica, ma essendo le tecnologie nuove ed in forma sperimentale, alcuni scenari sono preclusi da vincoli tecnologici, funzionali, di portabilità o di performance. Ad esempio la versione ufficiale della SIB è compatibile solo con il sistema operativo Linux e questo limita il numero degli scenari implementabili e quindi limita la velocità di diffusione del nuovo approccio di rappresentazione dell'informazione.

Per questa ragione, ed essendo inoltre la SIB un modulo astratto implementabile tramite diverse tecnologie, si è cominciato a lavorare su versioni diverse, più portabili e che presentassero un'architettura interna flessibile e modulare. La SIB OSGI (SIB-O) è stata implementata presso l'università di Bologna con il supporto di Eurotech, è stata testata su Windows Linux e Macintosh  e fornisce un livello di portabilità simile a Java, sul quale gira OSGI. Tuttavia OSGI è molto più legata al mondo Industriale rispetto che a una comunità aperta come quella che per ora ha preso in consegna il compito dello sviluppo della piattaforma semantica. Inoltre OSGI non è esattamente Java e quindi non è portabile allo stesso modo: ad esempio è molto semplice far girare un programma Java su smart-phone Android, ma non è assolutamente banale lanciare sulla medesima piattaforma mobile dei bundle OSGI. Questo perché il framework OSGI richiede delle caratteristiche avanzate di Java che sono solo parzialmente supportate dalla macchina virtuale Dalvik di Android. Solo dopo laboriose e minuziose modifiche è possibile far girare correttamente il framework Equinox ed i relativi bundles, componenti la SIB-O, su Android, queste modifiche rimarrebbero per altro legate alla specifica versione di Android e potrebbero richiedere nuove modifiche in seguito.

Per queste ragioni una l'implementazione di una SIB in JAVA ( SIB-J ) risulta motivata ed importante. Essa consentirebbe anche di supportare nativamente l'integrazione di framework Java legati al mondo del semantic web come il resoner Pellet ed il rule-engine JESS.

E importante notare che OSGI è un framework estremamente diffuso e dalla grande efficienza e solidità, la SIB-O, pur essendo una versione alfa, utilizza questo framework che ottimizza l'esecuzione in vari modi senza che il programmatore debba occuparsi di questioni di bassissimo livello. Sarà obiettivo di questa tesi preservare il più possibile le caratteristiche della SIB originale o addirittura migliorarle ove possibile.

La trattazione è organizzata come segue: nel primo capitolo verranno introdotte nozioni generali sullo scenario di rilevanza, e le tecnologie. Nel

secondo capitolo si parlerà della SIB-O, introducendone l'architettura e le funzionalità. Il terzo capitolo descriverà il lavoro da me svolto ed infine il quarto capitolo riporterà i risultati di test di performance allo scopo di validare il lavoro svolto e caratterizzare l'efficienza dei prodotti software da me realizzati.

# Capitolo 1

# Scenario tecnologico

Fin dall'inizio degli anni '90 era chiaro come la legge di Moore avrebbe portato nel breve termine ad un drastico aumento del numero di dispositivi che ci circondano. L'avvento del fenomeno Internet, sommato al miglioramento delle tecnologie sensoriali ed alla connettività multimodale a basso consumo, hanno favorito il fatto che questi dispositivi fossero interconnessi fra loro e potessero quindi cooperare per scopi comuni.

Da tutti i fattori analizzati sembrerebbe che la visione di Mark Weiser [1] stia divenendo realtà, e ciò è supportato anche dal fatto che i database non SQL iniziano a prendere la loro fetta di mercato, essendo più adatti a memorizzare la grande mole di dati prodotta dal crescente numero di dispositivi.

Tuttavia in realtà ci sono ancora degli ostacoli alla realizzazione di un mondo di dispositivi autonomi che cooperano per offrire i migliori servizi possibili, le architetture software ed i formalismi di rappresentazione dell'informazione correntemente usati non sono adatti a supportare il trend emergente e quindi il mondo della ricerca sta esprimendo un grande sforzo per sperimentare e diffondere nuove pratiche e nuovi modelli di programmazione e rappresentazione dell'informazione.

## 1.1 Il "Contesto" ed il semantic web

Le applicazioni in contesti di intelligenza ambientale appartengono spesso alla macro-famiglia della applicazioni context-aware, consapevoli cioè del contesto e quindi in grado di prendere decisioni più mirate ai bisogni dell'utenza. Il contesto viene definito come l'insieme dei valori delle variabili di stato degli attori che caratterizzano una situazione. Se ci riferiamo ad una applicazione context-aware il contesto è rappresentato da

tutti i dati di tutti gli attori rilevanti per l'applicazione compresi l'utente e l'applicazione stessa [2].

Requisiti per la context-awareness sono un respository di contesto accessibile a tutti gli agenti interessati ed una metodologia per la codifica dell'informazione che garantisca la comprensione automatica priva di ambiguità. Il primo punto verrà descritto nella sezione 1.2 e poi ripreso più avanti. Per quanto riguarda i formalismi di rappresentazione del contesto, i linguaggi appartenenti al semantic WEB sono degli ottimi candidati ad essere il riferimento, fra i linguaggi disambigui.

Il semantic web (SW)[3] è un progetto nato dalla mente di Tim Berners Lee, il fondatore del World Wide Web. Il SW si basa sulle attuali tecnologie di rappresentazione dell'informazione nel comune web, ma aggiunge delle regole strutturali e sintattiche che consentono un'interpretazione automatica e priva di ambiguità da parte degli agenti software ( "machine understandability" ).

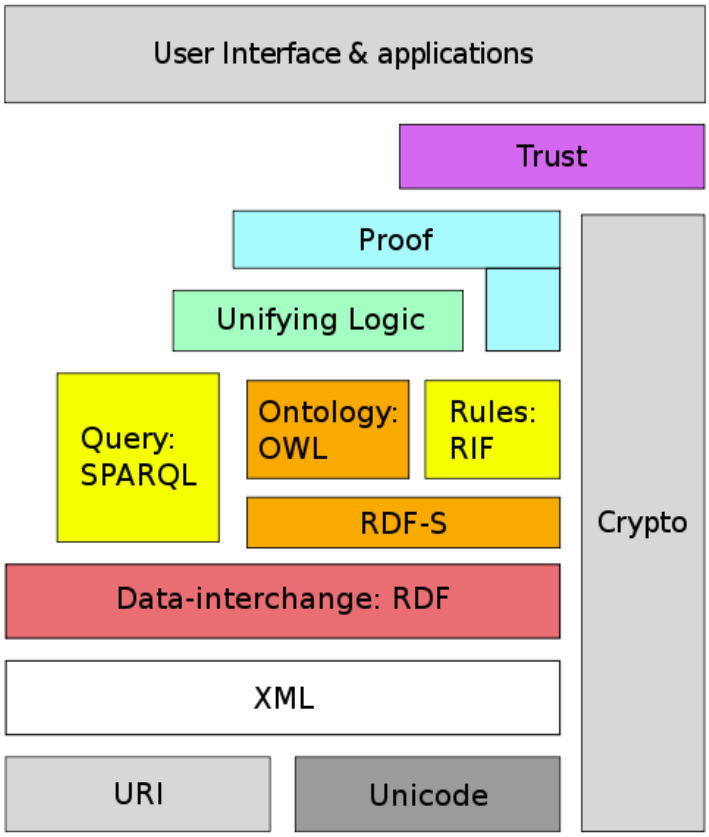


**Figura 1: La struttura e le tecnologie del web semantico**

La figura 1 mostra sinteticamente le tecnologie del SW: alla base della piramide ci sono i comuni formalismi di rappresentazione del testo, poi vi è l'XML che perde un po' di leggibilità per l'essere umano, ma ne acquista per un lettore software, visto che qualsiasi documento XML ben formato può essere ricondotto ad un albero e quindi ad una struttura dati ben nota in tutti i linguaggi di programmazione. XML però non fornisce tutti gli strumenti necessari all'interpretazione del significato, questo perché non esiste alcun significato intrinseco nell'innestamento dei tag e quindi il significato della relazione padre-figlio di due nodi dell'albero è ignoto all'agente software generico, ma noto solo all'architetto del documento XML.

RDF (Resource Description Framework) ha un modello associato molto più solido dal punto di vista semantico. Un documento RDF corrisponde infatti ad un grafo orientato labellato in cui la relazione fra due nodi è esplicitata dal label dell'arco che li congiunge. RDF, oltre a proporre una sintassi basata su XML ( molto usata, ma non l'unica possibile ) impone due regole molto importanti: ciascun nodo del grafo deve essere identificato da un identificatore univoco (Uniform Resource Identifier URI), tranne i singoli valori letterali di descrizione delle risorse. L'informazione è composta da un insieme di triple del tipo soggetto-predicato oggetto, che considerate nel loro insieme corrispondono appunto ad un grafo orientato labellato. Da quanto detto consegue che il soggetto di una tripla deve essere identificato da un URI, ma anche il predicato dovrà esserlo, altrimenti potrebbero insorgere ambiguità nell'interpretazione del loro significato. L'oggetto di una tripla può invece essere identificato da un URI nel caso in cui rappresenti una risorsa o da una semplice successione di caratteri alfanumerici nel caso in cui rappresenti un numero, o una stringa o un altro tipo di dati primitivo. In Fig. 2 è riportato un noto esempio di grafo RDF che mette in relazione le più grandi basi dati semantiche.

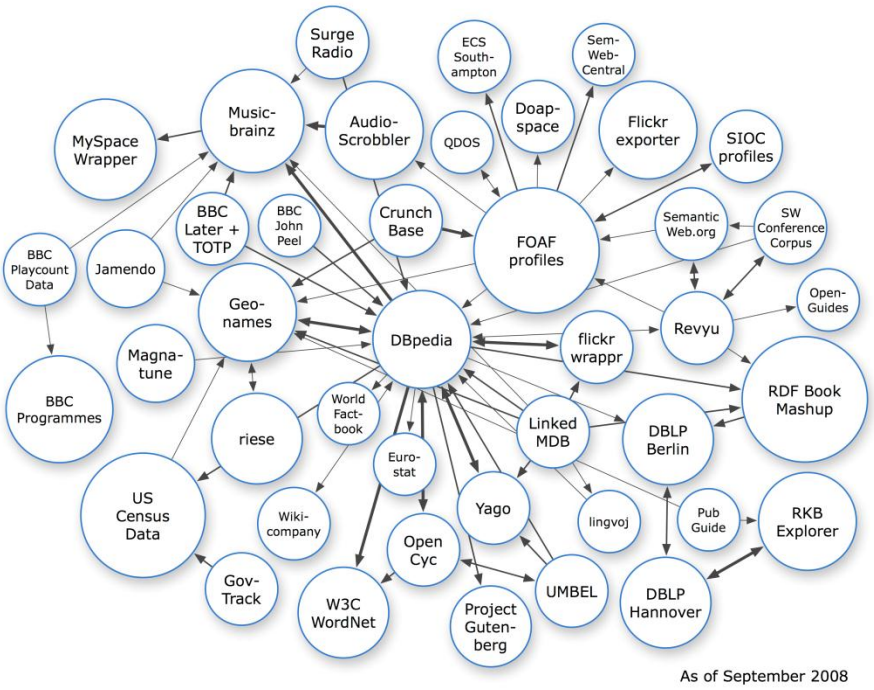


**Figura 2: Rappresentazione dei più importanti repository RDF nel 2008**

Il modello dei dati di RDF è diverso da quello relazionale, che solitamente è usato nel mondo dei servizi web e delle applicazioni commerciali, i comuni database relazionali non possono essere efficientemente utilizzati mentre alcuni fra i database non SQL sono particolarmente adatti a gestire una base informativa basata su un grafo semantico. Modelli informativi come Jena, Sesame, Virtuoso o la stessa SIB si stanno diffondendo anche per questo motivo.

Sopra a RDF il SW ha molti altri livelli di astrazione, ma mentre il livello ontologico ha ormai raggiunto un buon livello di stabilità ed è usato in molti campi applicativi, il livello delle regole, quello logico e gli altri, sono ancora in fase di definizione e quindi bisognerà attendere prima di poter cogliere appieno il potenziale che il progetto del SW ha.

OWL, l'ultimo dei livelli stabili del SW è un linguaggio per la specifica di ontologie le quali a loro volta rappresentano un dominio concettuale in termini di classi proprietà istanza e relazioni. Alle ontologie corrisponde

anche un noto modello logico detto logica descrittiva, che consente per altro di effettuare inferenze e controlli di consistenza.

## 1.2 SOFIA e la SIB

Per ottenere applicazioni autonome e consapevoli della semantica del contesto c'è bisogno di strumenti per la corretta gestione di una base di conoscenza con un modello di dati a grafo orientato labellato. Per questa ragione nel 2008 il progetto europeo SOFIA è nato allo scopo di fornire alla comunità un nuovo strumento software in grado non solo di operare su un database semantico, ma anche di fornire un modello di utilizzo basato sugli eventi ( publish-subscribe ) ed un'implementazione non legata alla singola tecnologia realizzativa. Una SIB può essere scritta in diversi linguaggi e girare su diversi framework, essa deve però rispettare il protocollo di comunicazione, chiamato SSAP ( Smart Space Access Protocol ), ed essere compatibile con le altre implementazioni in modo che gli agenti software non debbano conoscere la particolare versione di SIB con cui stanno interloquendo, ma solo il protocollo di comunicazione.

La prima versione di SIB stabile è stata rilasciata durante il progetto SOFIA, ad essa si sono affiancate nel tempo diverse implementazioni che sperimentano nuove caratteristiche o provano ad offrire funzionalità innovative. La Red-SIB, sviluppata da una comunità in cui è attivamente presente l'università di Bologna, è divenuta la versione più utilizzata ed ha aggiunto diverse caratteristiche funzionali e non alla versione originale come ad esempio le query/sottoscrizioni SPARQL, il supporto al linguaggio SPARQL update ed un algoritmo ottimizzato di processamento delle sottoscrizioni che evita la valutazione continua di query sulla base di conoscenza.

La SIB-O è stata implementata su tecnologia OSGI parallelamente alla RED-SIB dall'università di Bologna in collaborazione con l'azienda Eurotech. Correntemente supporta un insieme di primitive molto simile a quello della Red-SIB, ma essendo basata sul linguaggio a oggetti Java offre

caratteristiche di maggiore versatilità, modularità e portabilità al prezzo di performance meno stressate.

Qualunque sia la tecnologia realizzativa, come anticipato prima, il modello di utilizzo di una SIB in un'applicazione di intelligenza ambientale è il medesimo ed è rappresentato in figura 3.

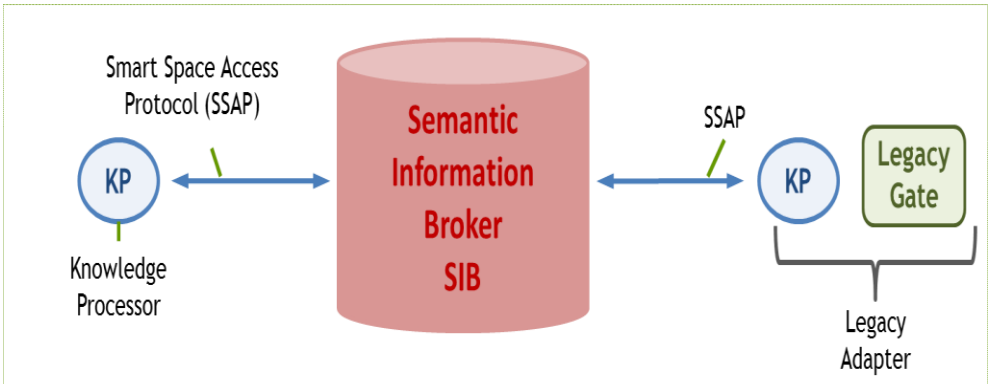


**Figura 3: Architettura software basata su SIB**

La SIB contiene il contesto e viene acceduta in lettura e scrittura da agenti software che prendono il nome di KP ( Knowledge processor ). I KP interagiscono con la SIB tramite le primitive del protocollo SSAP, i dispositivi non nativamente programmati per comunicare tramite SSAP ( denominati legacy ) vengono integrati nell'architettura di alto livello tramite degli opportuni KP detti legacy adapters.

E' facile intuire che l'architettura software di cui la SIB è il centro è molto generica e si adatta ad un'estrema varietà di scenari. In alcuni scenari però saltano all'occhio vincoli di portabilità e flessibilità che fanno propendere per l'una o per l'altra versione. Nel prossimo capitolo analizzeremo brevemente la SIB-O, che risolve i problemi di portabilità su sistemi operativi diversi da Linux e quelli di monoliticità del software della Red-Sib

# Capitolo 2

# SIB-O

## 2.1 Struttura interna

La SIB-O[4] è stata testata sull'implementazione Equinox di OSGI: l'implementazione più nota su cui ad esempio è basato il noto ambiente di sviluppo Eclipse. La sua struttura interna è mostrata in Figura 4: ci sono sette bundle distinti ciascuno dei quali, secondo il paradigma di OSGI offre servizi agli altri oppure è in grado di trovare i servizi già registrati dagli altri bundle. Quando un bundle usa i servizi di un altro si crea una dipendenza: il bundle TCP, ad esempio, dipende da MH (Message Handler) e da SIB. Tutti i bundle tranne Join, utilizzano una libreria di supporto condivisa a livello di framework chiamata tools, che non è stata indicata in figura per non appesantire la rappresentazione grafica.

Ciascun bundle all'inizio registra, a livello di framework, i servizi offerti e poi, in base all'ordine di lancio, cerca i servizi che richiede, se tutti i bundle verificano che le dipendenze sono rispettate la SIB parte correttamente.

Provando ad analizzare come viene processata una generica richiesta da un client (ad esempio una query) il flusso delle informazioni è il seguente: si parte da TCP, che riceve la richiesta in forma di stringa di caratteri, poi il Message handler (MH) si incarica di gestire la richiesta chiedendo al token handler (TH) di fornire un identificatore univoco ed al SSAP di trasformare il messaggio, ancora in forma di stringa, in oggetto strutturato. A questo punto il message handler può chiedere l'intervento del SIB bundle che fornisce il messaggio di risposta in forma di oggetto strutturato. Quindi il message handler dovrà occuparsi di mandare la risposta al client corretto: per prima cosa chiedendo al bundle SSAP di fornire la stringa dei caratteri che compongono il messaggio di risposta e poi al TCP di eseguire la funzione call-back sulla socket corrispondente all'identificatore che precedentemente era stato associato all'intera transazione dal token handler.

Il SIB bundle, prima di fornire la rappresentazione a oggetti del messaggio di risposta esegue diverse operazioni usando bundle di supporto quali JOIN, che si occupa delle questioni inerenti l'autenticazione ed il controllo accessi e Subscribe (SUB) che invece verifica se a causa dell'operazione in corso sia necessario notificare eventuali client che avevano precedentemente eseguito una sottoscrizione.

Si nota da subito che la struttura interna della SIB OSGI non è monolitica, ma modulare. Nella sua realizzazione si è cercato, per quanto possibile, separare le responsabilità fra i moduli in modo da semplificare eventuali variazioni, localizzando le modifiche nei soli moduli interessati. Ad esempio una modifica del protocollo SSAP influisce sul solo bundle omonimo e parimenti una modifica della libreria di gestione della base di conoscenza influisce sul solo SIB bundle.

Di seguito sono brevemente riportati i compiti dei bundles componenti la SIB-O

- **TCP:** si occupa di mettersi in ascolto delle richieste TCP inviate dai KP client e mantiene attive le socket relative alle sottoscrizioni. Più TCP possono essere lanciati su porte diverse lasciando invariato il resto dell'architettura interna.
- **Message Handler:** gestisce la coda delle richieste e la comunicazione fra i bundles. Si appoggia ad altri due bundle per la produzione e la consumazione dei messaggi.
- **SSAP:** serve per il parsing delle richieste e per formattare le risposte secondo le specifiche correnti del protocollo SSAP.
- **Token Handler:** associa ad ogni richiesta un identificatore che verrà utilizzato in fase di risposta per contattare il client corretto.
- **SIB:** è il cuore del sistema. Dove vengono effettivamente eseguite le istruzioni fornite dai KP, come inserimenti, rimozioni, query ecc.
- **Subscribe:** serve per tener traccia ed ottimizzare le operazioni concernenti le sottoscrizioni.

- **Join:** un KP può comunicare con la SIB solo se ha precedentemente comunicato la sua esistenza ed il desiderio di interagire tramite l'apposita primitiva JOIN. Il bundle, che prende il nome dalla primitiva, verifica se un KP ha preventivamente effettuato l'operazione di JOIN con le adeguate credenziali. Se non vengono riscontrati problemi di sicurezza l'operazione viene consentita.

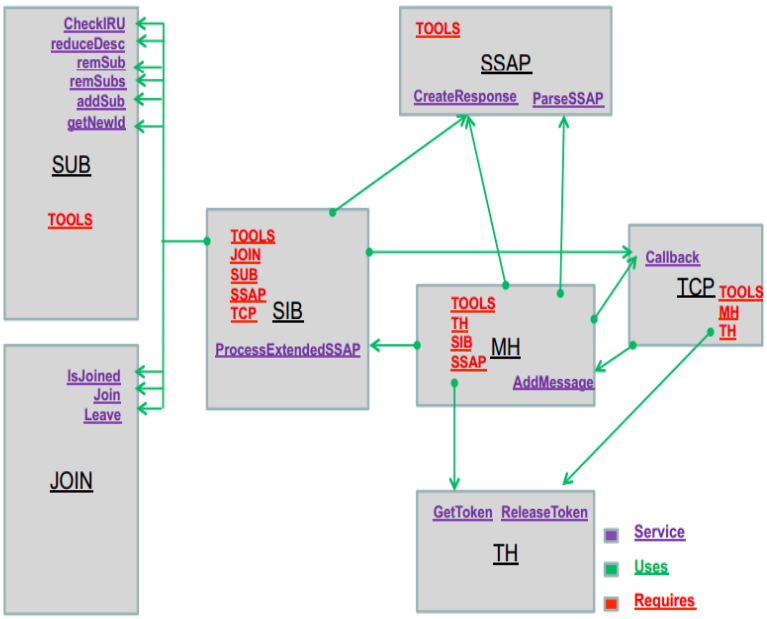


**Figura 4: Architettura della SIB OSGI**

## 2.2 Funzionalità

Essendo la SIB uno strumento per lavorare nel mondo del SW ed in quello della programmazione orientata agli eventi, le operazioni che essa espone sono legate all'elaborazione di triple ed al meccanismo publish-subscribe. Riporto di seguito l'elenco delle primitive supportate con una breve spiegazione per ciascuna di esse

- **Join:** Primitiva di accesso allo smart-space, nella versione corrente non supporta parametri d'ingresso visto che le politiche di autenticazione e controllo accesso sono ancora in fase di definizione. Un KP che ha effettuato l'operazione di Join può interagire in lettura/scrittura con il contenuto della SIB. Ciascun KP è associato

ad un identificatore univoco detto KP-ID per verificare se la join è avvenuta o no

- **Leave:** operazione duale alla join che informa la SIB del fatto che un determinato KP ha terminato di interagire. Un KP che avesse effettuato l'operazione di LEAVE è tenuto ad effettuare nuovamente una JOIN prima di poter interagire con la base di conoscenza.
- **Insert:** inserisce triple di tipo soggetto predicato oggetto all'interno del grafo condiviso. Una o più triple possono essere trasportate contemporaneamente. Ci sono tre opzioni di formattazione. Il formato rdf-M3 consiste nello specificare le triple separando soggetto predicato e oggetto tramite un'opportuna indentazione di specifici elementi xml all'interno del protocollo, essa non corrisponde ad uno standard ed è nata con M3 ai tempi dell'inizio del progetto SOFIA. Il formato RDF-XML è uno standard raccomandato dal W3C per la specifica di grafi RDF ed è supportato nelle operazioni di inserimento, rimozione aggiornamento. SPARQL-update è uno degli ultimi risultati a cui è giunto il mondo del SW, uno standard poco verboso e concepito per usare il potere espressivo di SPARQL non solo in operazioni di lettura, ma anche in operazioni di modifica del contenuto.
- **Remove:** operazione che consente di rimuovere degli archi dal grafo condiviso. Anche per la remove sono supportati i tre formati di rappresentazione delle triple validi per la insert.
- **Update:** è una primitiva con due argomenti, entrambi gli argomenti sono un insieme di 0 o più triple. Il primo argomento andrà rimosso dalla base di conoscenza, mentre il secondo argomento andrà inserito. L'uso di questa primitiva è tipico nelle applicazioni sensoriali dove capita spesso di dover sostituire un valore con uno più aggiornato e di dover usare due volte una primitiva quando sarebbe possibile usarne una sola con due argomenti. L'insieme delle triple rimosse e quello delle triple inserite non sono comunque correlati e quindi è possibile utilizzare le update anche nel caso in

cui vogliamo velocizzare una procedura che presenta inserimenti e rimozioni consecutivi. Anche per la primitiva di update sono supportati i formati di specifica delle triple RDF-M3, RDF-XML e SPARQL-Update.

- **Query:** la query è la primitiva che consente di accedere in lettura al contenuto della SIB. Due formati sono supportati di cui uno è standard è l'altro è specifico della SIB. Il formato RDF-M3 consiste nello specificare in ingresso alla query un insieme di pattern di triple in cui ciascun elemento può essere un valore, un URI o una wildcard, se una tripla della base di conoscenza ha il soggetto il predicato e l'oggetto uguali a quelli di uno dei pattern allora essa fa parte della soluzione. Le wildcard consentono di estendere l'espressivita dell'operazione estendendola a casi di pattern non completamente specificati. L'altro formato di query è lo SPARQL. SPARQL è il linguaggio raccomandato dal W3C per eseguire query su basi di conoscenza RDF, è provvisto di potenti costrutti sintattici per specificare query molto precise. SPARQL rende possibile anche la ricerca per negazione ( che torna risultati quando le triple non ci sono ) il filtraggio dei risultati ed altre interessanti e potenti funzionalità.
- **Subscribe:** è un primitiva che ha una query. SPARQL o basata su pattern, come input. Il messaggio di risposta contiene i risultati della query, tuttavia la sottoscrizione resta attiva anche dopo il ricevimento della SIB e si occupa di tenere il KP aggiornato sullo stato del contesto espresso nella query iniziale. Dopo la prima risposta, infatti, il KP mantiene una connessione aperta con la SIB che invia notifiche ogni qualvolta ci sono cambianti nella SIB che modificano lo stato della query iniziale. Sia le triple aggiunte al grafo che le triple rimosse vengono notificate al KP chiamante a patto che esse siano incluse fra i risultati della query specificata all'atto della sottoscrizione. La sottoscrizione è un'operazione molto importante perché evita continue query agli agenti software

interessati alle variazioni di un contesto, riducendo così il traffico sulla rete ed aumentando allo stesso tempo la reattività del sistema.

- **Unsubscribe:** operazione che rimuove una sottoscrizione. Richiede l'ID della sottoscrizione che viene fornito dalla SIB al KP richiedente quando si sottoscrive.
- **Persistent Update:** è un'operazione introdotta sperimentalmente per supportare il concetto di regola, fin'ora assente in tutte le implementazioni di SIB. La primitiva ha come input una sparql update che, se provvista di clausola "WHERE" può essere letta come una regola. La persistent update viene eseguita la prima volta, le viene poi associato un ID che viene spedito al KP richiedente nel messaggio di conferma. A differenza delle normali SPARQL update, l'update persistente continua la sua azione anche nel futuro della base di conoscenza. Questo vuol dire ad esempio che se noi inseriamo nella SIB una persistent update che per ciascuna persona aggiunge una tripla per indicare che è anche un essere umano, anche le persone inserite nella SIB in futuro saranno provviste immediatamente di questa proprietà, quasi come se il sistema fosse provvisto di intelligenza
- **Cancel persistent update:** rimuove un update persistente se l'id corretto le viene fornito in ingresso.

Sotto il profilo delle performance alcune primitive hanno tempi di risposta che dipendono dalla mole dei dati in ingresso, come ad esempio gli inserimenti, altre sono più difficili da caratterizzare perché dipendono dalla mole dei dati di uscita, da quella dei dati contenuti nella SIB e dalla struttura interna della base di conoscenza, cioè dalle proprietà del grafo che la rappresenta. Infine ci sono delle primitive che influiscono sulla performance delle altre perché, fornendo un servizio continuativo e reattivo, di notifica o di update persistente, svolgono delle operazioni sul percorso di processamento dei dati. Vedremo nell'ultima sezione di questa tesi come si possano stimare le performance del sistema allo scopo di produrre un'analisi comparativa fra diverse implementazioni di SIB.

# Capitolo 3

# Il mio contributo

## 3.1 Obiettivi del passaggio dall'architettura OSGI all' architettura SIB Manager:

L'obiettivo di questa tesi è stato l'implementazione di una SIB in Java (SIB-J) che fosse compatibile con la Java virtual machine ma che non utilizzasse framework esterni come OSGI. Partendo da una versione già scritta in linguaggio Java è stato necessario rimuovere tutte le chiamate al framework e poi sostituirle con procedure equivalenti. Siccome i primi risultati sono stati incoraggianti ho anche mirato a dimostrare la maggiore portabilità della versione java, eseguendo un porting della stessa su piattaforma Android. Vale la pena notare che, introducendo un componente di SIB mobile negli scenari di intelligenza ambientale si moltiplicano enormemente le possibilità di realizzare servizi innovativi. Siccome molte nuove tecnologie, per emergere, hanno bisogno di raggiungere una massa critica di utenza o di interesse, l'implementazione di un software che aumenti il campo di utilizzabilità e che introduca nuovi scenari applicativi è della massima utilità nell'attuale contesto della ricerca in cui ho collaborato.

Scendendo maggiormente in dettaglio ci sono diversi aspetti che rendono rilevante il lavoro svolto

- **Portabilità:** una SIB in Java presenterà caratteristiche di portabilità più marcate rispetto ad una OSGI e questo potrebbe portare a nuove sperimentazioni come ad esempio quella della SIB su Android smart-phone.

- **Leggerezza:** al momento la SIB-O non utilizza appieno le potenzialità offerte dal framework in cui gira, quindi molte risorse allocate dal framework non vengono utilizzate.

- **Curva di apprendimento:** la comunità java è più ampia di quella OSGI, in particolar modo se limitiamo la discussione all'ambito della ricerca, ove

attualmente la SIB-O è utilizzata. La SIB-J potrebbe essere appresa più rapidamente e da un'utenza più ampia e questo gioverebbe al suo sviluppo.

**- Mobilità:** aumento di scenari dove viene usato il sistema SIB.

- **Ottimizzazione:** un ultimo obiettivo della mia tesi è stato quello di utilizzare le mie conoscenze maturate in ambito lavorativo per apportare ottimizzazioni e migliorie al codice in modo da sostituire al meglio il Framework OSGI ed eventualmente guadagnare in performance ove possibile.

## 3.2 Lavoro svolto

Il lavoro svolto è stato suddiviso in fasi al termine di ciascuna delle quali ho rilasciato una versione di SIB, infine, nella fase finale ho eseguito test di performance e funzionali per valutare se durante il porting erano stati introdotti bug non voluti e per quantificare le prestazioni delle nuove versioni.

La prima fase è consistita nel rilasciare una versione di SIB-J con il codice il più possibile uguale a quello della SIB-O, rimuovendo solo i richiami al framework e sostituendoli con il minor numero possibile di righe di codice. L'output di questa prima fase è stato la SIB-J v1 che aveva l'obiettivo di essere un rilascio minimale con la minima probabilità di introduzione di errori nuovi, una versione di passaggio insomma sulla quale fosse possibile lavorare ulteriormente.

Nella seconda fase, dopo una rapida analisi delle prestazioni della SIB-J v1 e dopo aver verificato che le funzionalità della SIB-O erano preservate, sono passato ad una fase di ottimizzazione che mi ha consentito sia di ottenere la SIB-J ottimizzata (o semplicemente SIB-J) sia di contribuire allo sviluppo ulteriore della SIB-O perché alcune delle ottimizzazioni da me applicate potevano essere introdotte anche nel codice di tale SIB.

Nella terza fase mi sono concentrato sul porting della SIB-J su Android, ho quindi prima studiato il sistema Android[5][6], i suoi requisiti ed il modo in cui le applicazioni native Java fossero includibili all'interno di un file

eseguibile per la Dalvik Virtual machine (file .apk). Una volta capito come fare ho prima adattato le librerie ed infine le differenze nella gestione dell'input-output. L'ultimo rilascio è stato la SIB Android che come vedremo avrà delle prestazioni leggermente inferiori rispetto alla SIB-J, viste le risorse inferiori a disposizione, ma che, come precedentemente discusso, amplia i confini degli scenari plausibili introducendo le SIB portabili come nuovi attori nei casi d'uso di intelligenza ambientale.

Nelle prossime sezioni descriverò alcuni dettagli del lavoro svolto soffermandomi solo su ciò che ritengo più importante e tralasciando i dettagli implementativi.

## 3.3 Analisi del passaggio dall'architettura OSGI all' architettura SIB Manager

La SIB OSGI è basata sull'interazione service-oriented fra i bundles che compongono il sistema. Ciascun bundle è in grado di reperire gli altri e di invocare i loro servizi grazie alle primitive di registrazione e le chiamate per riferimento fornite dall'architettura OSGI. OSGI offre anche una potente gestione del multithreading che consente di ottimizzare le operazioni concorrenti e di gestire autonomamente gli accessi alle parti condivise in modo sicuro e affidabile.

Fin dalla prima versione della SIB-J v1 ho quindi dovuto definire dei meccanismi per sostituire il framework tramite del codice Java standard ed in particolare mi sono concentrato sulla gestione dei servizi e su quella degli accessi alle parti condivise.

### *3.3.1 Servizi vs Processi Java (Thread) :*

Ogni servizio OSGI è stato sostituito da un Thread Java. I vari bundle della SIB-O sono quindi sostituiti da Thread in SIB-J che chiameremo processi principali. I processi principali partono nello stesso ordine dei loro corrispondenti bundle ed espongono le medesime primitive. Ciascuna primitiva esposta corrisponde alle chiamate a servizio che avvenivano in

Equinox. Tuttavia risulta ancora difficile assicurare la mutua raggiungibilità dei processi principali e sincronizzarne l'accesso a risorse condivise per garantire il continuo funzionamento della SIB. La sincronizzazione fra i diversi processi deve essere ben gestita soprattutto per quanto riguarda lo stato del processo visto che questo cambia costantemente.

### *3.3.2 Coerenza nell' interazione fra i processi*

I processi in java vengono realizzati con i thread, questi ultimi sono difficili da referenziare una volta lanciati e questa è una grossa differenza rispetto all'architettura OSGI, munita della possibilità di richiamare il servizio ogni volta che serve visto che è stato registrato. Questo problema ci obbliga ad trovare una soluzione o un meccanismo simile nel caso dei processi principali che compongono la SIB-J che semplifichi il loro rintracciamento oltre ad avere uno stato corretto dove i dati sono coerenti.

La coerenza dei dati e degli stati dei processi è un problema in generale difficile da risolvere che potrei spiegare tramite un esempio tratto dalla mia esperienza durante la tesi. Si potrebbe pensare che una prima soluzione per risolvere il problema dell'accesso fra i processi è mappare ogni processo come proprietà all'interno degli altri che lo usano. In questo caso però, nonostante le cose sembrino funzionare quando il sistema non è sotto stress, ci sarà comunque una incoerenza fra i processi visto che ci sarà una copia dell'istanza mappata localmente diversa dall'istanza globale. In questo caso avremo un malfunzionamento sotto stress a causa di dati non aggiornati.

E' necessario quindi trovare una soluzione architetturale che risolva il problema senza intaccare il funzionamento corretto sia quando il sistema usa poche risorse che quando è impegnato a gestire un grosso traffico di dati.

### *3.3.3 Accesso concorrente ai dati condivisi*

Un altro grosso problema dove l'OSGI offre una soluzione gratuita è la gestione della sincronizzazione fra i servizi (bundle), dove ogni accesso fra un bundle e altro è sincronizzato perciò non ci sono problemi di coerenza al

livello globale, invece nella nostra struttura serve un meccanismo che garantisce la sincronizzazione nell'accesso fra i processi.

### *3.3.4 Soluzione proposta*

Prendendo in considerazione i problemi descritti prima, sono alla fine giunto a proporre un'architettura software basata su un nuovo processo che chiamerò Manager. Il Manager per primo istanzia e tiene traccia di tutti i processi principali, poi li gestisce in modo globale avendo visibilità su di essi e sul loro stato. Le comunicazione fra i processi avviene attraverso il manager: il manager rappresenta infatti un by-pass fra i processi. Tramite questo approccio, si risolve il problema del rintracciamento dei thread ed ogni istanza di processo principale sarà opportunamente osservabile dal livello alto cioè dal Manager. L'architettura SIB manager risolve anche il problema dell'accesso concorrente alle aree di memoria condivisa. Questo avviene perché essendoci un modulo con visibilità globale, diventa semplice usare gli strumenti di sincronizzazione messi a disposizione da Java per gestire il problema, come ad esempio i blocchi di codice synchronized.

L'architettura SIB manager ha quindi consentito di rilasciare la SIB-J v1, risolvendo tutti i problemi di reperibilità dei servizi e di coerenza sui dati.

### *3.3.5 SIB-J*

Una volta verificato che le funzionalità esposte dalla SIB-J v1 fossero equivalenti a quelle della SIB-O, anche ponendo il software sotto stress, sono passato alla fase 2 del mio lavoro in cui, avrei dovuto ottimizzare il codice, tenendo presente la mia esperienza nella programmazione, ed il fatto che un porting da OSGI con la minima modifica del codice di partenza avrà sicuramente posto in evidenza delle parti che prima erano utili e gestite dal framework ma che ora possono essere rimosse o gestite diversamente.

La prima ottimizzazione è derivata da un'analisi della vita dei processi che ha messo in evidenza come non fosse necessario istanziare dei processi principali per le funzionalità esposte da

- Token handler
- SSAP

- Subscription
- Join

Tutti questi moduli che nella SIB-O erano dei Bundle e nella SIB-J v1 erano dei processi principali gestiti dal manager, sono divenute delle semplici librerie. I processi principali TCP-IP, Message Handler e SIB oltre al SIB Manager sono rimasti invece processi che si attivano quando la SIB viene inizializzata e si chiudono solo durante la fase di disallocamento delle risorse della SIB.

## 3.4 Architettura della SIB-J

La SIB-J è composta da tre package (figura 5): il primo include il SIB Manager e serve per la gestione della memoria condivisa fra tutti i processi oltre alla sincronizzazione degli accessi. Il secondo package è una libreria che fornisce le funzionalità di base inizialmente mappate in bundle OSGI come il Token Handler e il SSAP. Il terzo package invece è il business-core del sistema SIB e contiene i processi principali per il funzionamento del sistema SIB (TCPIP, Message Handler e lo Store SIB).

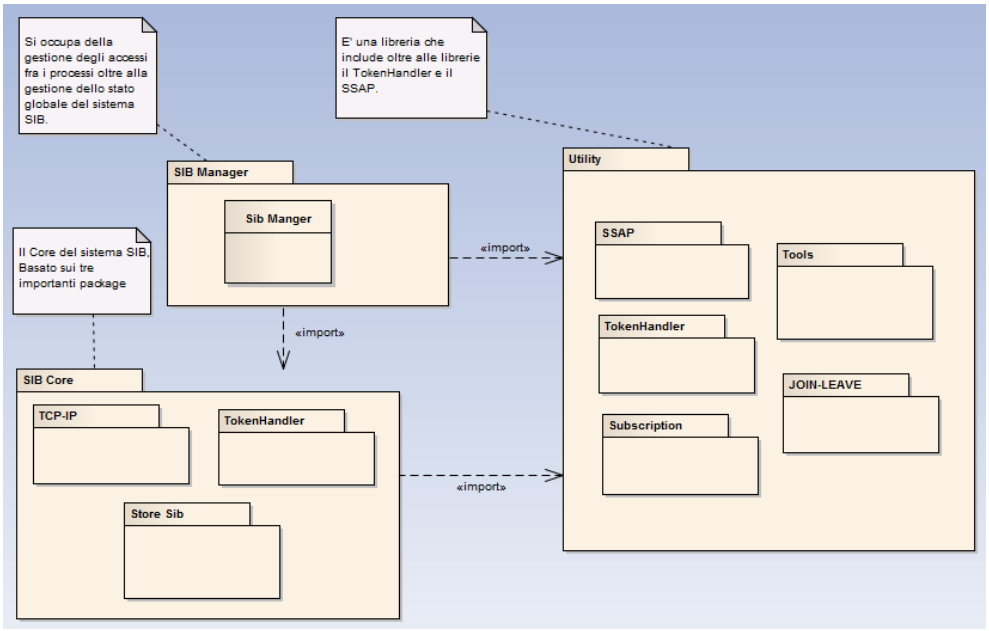


**Figura 5: Domain Model Dell'Architettura SIB Manager.**

Per spiegare meglio il funzionamento interno riporto infine in figura 6 il diagramma delle classi di due fra i moduli più importanti del sistema: il processo SIB e il Sib Manager. Questa rappresentazione mette in evidenza la loro struttura interna di attributi e metodi oltre al legame di ereditarietà che c'è fra le classi.

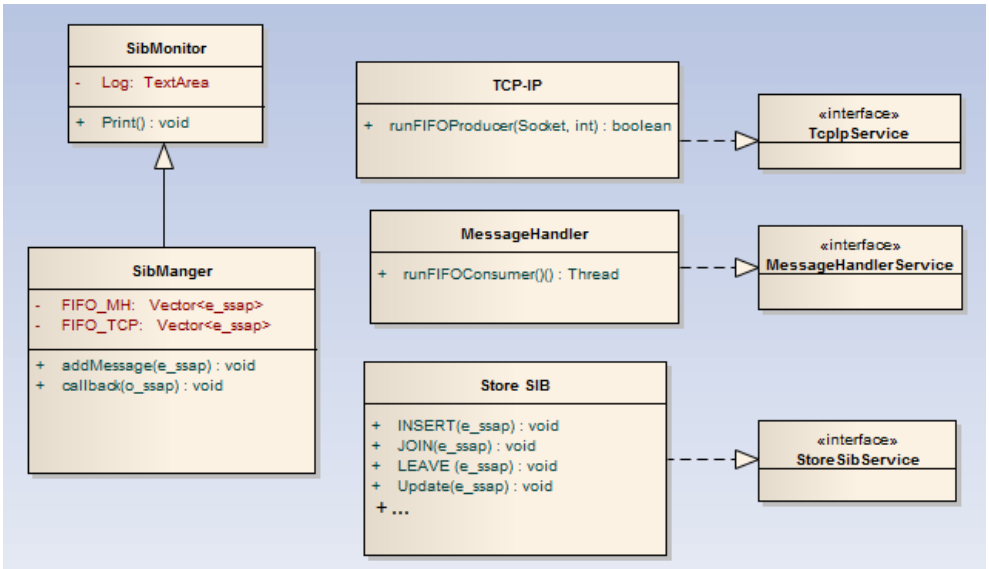


**Figura 6: Diagramma di classi del SIB Core e SIB Manager**

## 3.5 SIB su Android

L'ultima fase della mia tesi prima dell'esecuzione dei test prestazionali è stata il porting della SIB-J su smarphone Android. Questo porting, oltre a dimostrare la validità e l'utilità del lavoro svolto, incrementa come precedentemente discusso, il numero di scenari e casi d'uso che possono essere gestiti con architetture software SIB based.

Una SIB portabile costituirebbe un nuovo attore negli scenari di intelligenza ambientale ampliando la gamma dei servizi che è possibile fornire all'utente finale, ma per realizzarne una occorre rispettare i vincoli imposti dal sistema ospite. Negli anni passati le restrizioni imposte ai programmatori dai dispositivi mobili erano notevoli ed il porting del codice tutt'altro che facile.

Android è un sistema operativo per dispositivi mobili che dispone di una vasta comunità di sviluppatori che realizzano applicazioni e che lavorano a

livello di piattaforma per aumentare le funzionalità dei dispositivi. Le applicazioni possono essere scritte in linguaggio Java standard, ma vengono compilate con una macchina virtuale personalizzata denominata Dalvik virtual machine.

Android è ha costituito una vera e propria svolta per quanto riguarda la portabilità del codice su piattaforma mobile, consentendo di eseguire un vasto numero di librerie scritte in codice java standard senza particolari accorgimenti.

Ovviamente c'è un limite a questa portabilità ad esempio non è affatto semplice lanciare un'istanza del framework OSGI su Android[ref lanciare Equinox su Android][ref lanciare Apache felix su Android]. Diverse versioni di OSGI come Equinox ed Apache Felix sono diversamente compatibili con Android e per entrambe ci sono restrizioni sull'effettiva eseguibilità di codice generico. La compatibilità cresce con le versioni di Android, ma non c'è al momento una completa integrazione.

La SIB-J parte avvantaggiata rispetto alla SIB-O in termini di portabilità perché racchiude in sé solo l'essenziale per svolgere correttamente le sue funzioni e non tutto il potente ma dispendioso e complesso framework OSGI, che in questo caso risulta un ostacolo.

### *3.5.1 Esecuzione del porting su Android*

Android è provvisto di una macchina virtuale diversa da quella Java quindi il porting del codice risulta semplice se tutte le funzionalità che si desidera integrare nella nuova applicazione mobile sono già supportate dal convertitore Java-Dalvik, altrimenti occorre operare delle modifiche al codice o al packaging per far si che il porting vada a buon fine.

Durante il passaggio di versione dalla SIB-J alla SIB Android non ho incontrato problemi relativi alla compatibilità del codice, il codice della SIB J era per tanto interamente convertibile in formato virtuale per Dalvik. Tuttavia ho avuto un po' di problemi nell'integrare tutte le librerie di cui il progetto fa uso.

La SIB-J si basa su un framework Java per la semantica molto conosciuto chiamato Jena. All'interno del framework Jena sono presenti alcune librerie per le quali dovuto usare un tool di conversione per portarle in un formato leggibile per Dalvik. Fra le librerie integrate spicca la libreria ARQ che consta di diversi Megabyte di byte-code e che è stato particolarmente difficile da integrare. ARQ serve per la risoluzione, il pre-processamento e l'esecuzione di query SPARQL e di SPARQL updates, ma per fare ciò richiede molte risorse ed offre una grande varietà di metodi complessivi. Il problema è che Android impone un numero massimo di metodi per libreria e quindi ho dovuto creare delle librerie di appoggio e dei link simbolici allo scopo di separare ARQ in sotto-librerie più piccole e quindi adatte ad essere integrate in Android.

Altri cambiamenti che ho dovuto apportare per l'integrazione su Android sono relativi all'input Output ed all'inizializzazione.

A livello di inizializzazione ho creato una Activity ( classe specifica del mondo Android ) che si collega al SIB manager .A livello di I/O ho dovuto gestire tutte le stampe con delle notifiche Android. La visualizzazione delle notifiche in tempo reale non è stata semplice, ma sembra tutt'ora priva di errori e stabile.

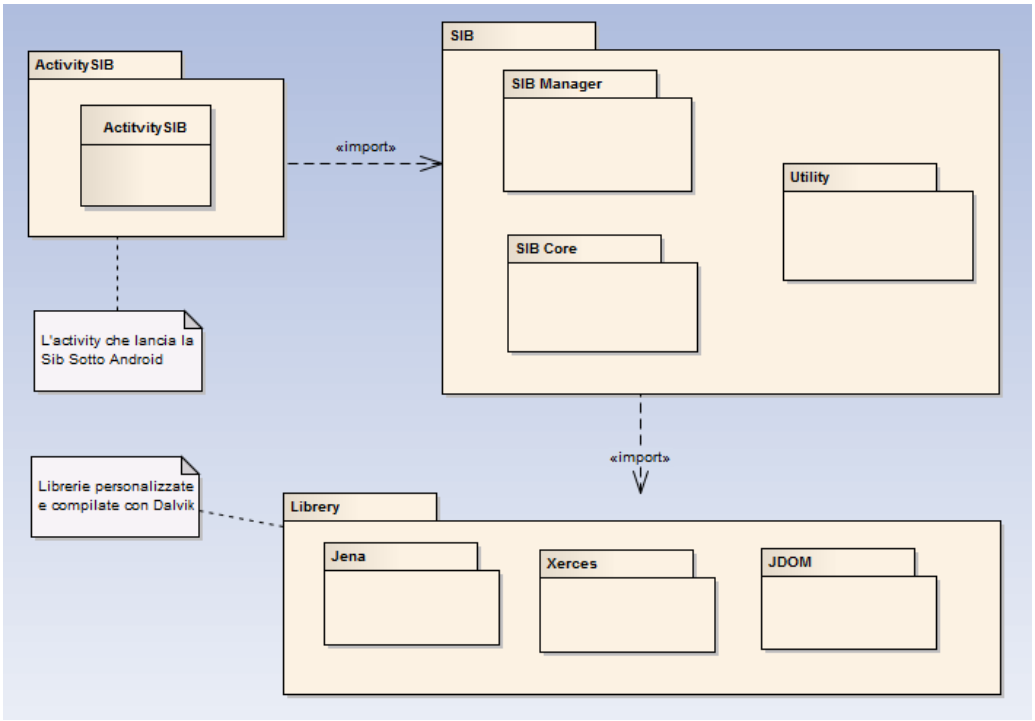


**Figura 7: Domain Model SIB Android**

Ogni messaggio di stampa che viene dalla SIB viene stampato su una area di test che rappresenta uno strumento di monitoraggio per la SIB Android.

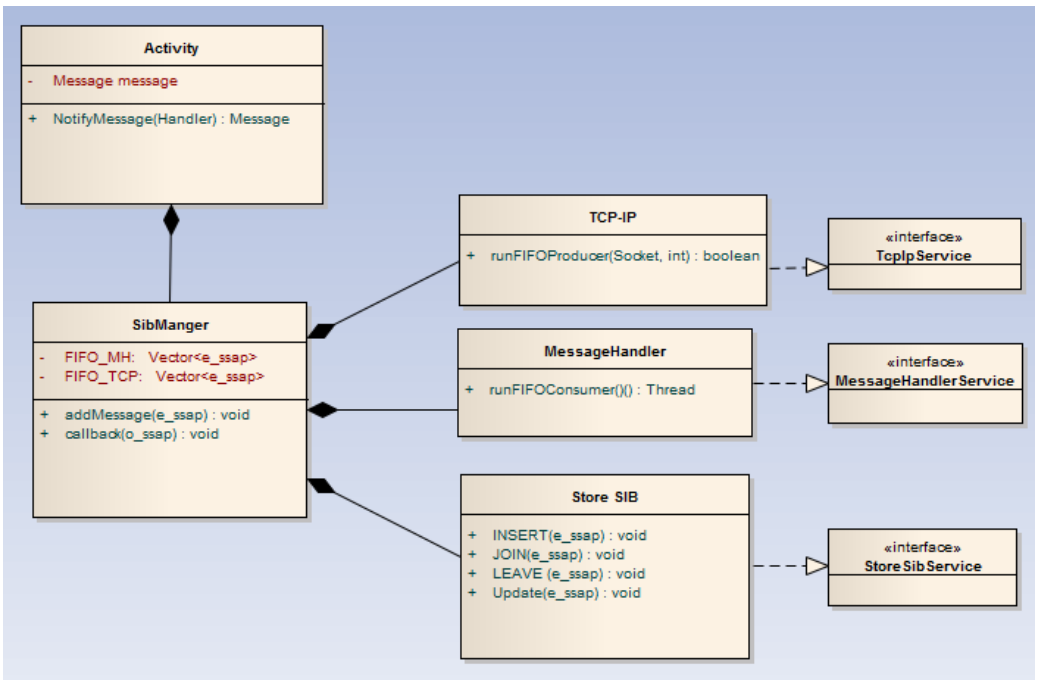


**Figura 8: diagramma delle classi della SIB Android**

# Capitolo 4

# Analisi delle prestazioni

## 4.1 Scenari di test

Alla fine del mio lavoro e dopo il rilascio di tre nuove versioni funzionalmente corrette di un software di partenza, è necessario effettuare dei test di performance per trarre delle considerazioni quantitative e verificare la validità dell'approccio utilizzato.

Si possono scegliere diverse strategie per paragonare fra loro le prestazioni di diverse versioni di SIB. Ad esempio si potrebbe scegliere uno scenario applicativo, porre le diverse versioni di SIB a gestirne la base di conoscenza e poi variare un parametro di carico fino a che lo scenario non viene più gestito correttamente. Un'altra opzione comparativa potrebbe essere la misura del tempo di completamento di un algoritmo fissato. Oppure si potrebbe scegliere di misurare singolarmente le performance relative al tempo necessario per eseguire delle singole primitive parametrizzate.

Tutti i metodi esposti hanno dei punti a favore e delle controindicazioni, alla fine ho optato per considerare i tempi impiegati dalla SIB per rispondere a primitive di basso livello, essendo difficile trovare un benchmark o uno scenario abbastanza generali per le nostre esigenze.

La primitiva scelta per eseguire i test è il tempo di inserimento, sono inoltre stati scelti due parametri: il numero delle triple nell'operazione ed il numero di sottoscrizioni. Le sottoscrizioni costituiscono infatti un carico per la SIB ed al loro aumentare le prestazioni calano, anche se non ci sono notifiche da inviare. Un buon parametro di efficienza per un'implementazione di SIB è una bassa dipendenza del tempo di esecuzione di primitive in scrittura, dal numero di sottoscrizioni attive, mantenendo la condizione che nessuna operazione durante il test scateni notifiche.

## 4.2 Test effettuati

### *4.2.1 – SIB-J v1*

Sono stati effettuati 3 test per confrontare la prima versione di SIB-J con la SIB-O, nel primo test (figura 9 ) è stato confrontato il tempo di inserimento di un numero di triple variabile secondo la seguente successione { 1, 20, 50, 100, 200, 500, 1000 }. Per ciascun test i risultati sono stati calcolati più volte e poi mediati per evitare il più possibile oscillazioni dovute a possibili diversità non controllabili fra l'esecuzione di un test ed il successivo.

Notare come al crescere del numero delle triple inserite le prestazioni della SIB-J incrementino più rapidamente rispetto a quelle della SIB-O fino a divergere completamente quando si passa 200 triple a 500 triple inserite.

Ho poi scelto due valori rappresentativi di triple inserite ed ho eseguito il test 2 (Figura 10) ed il test 3 (Figura 11). Nel secondo e nel terzo test il numero di triple è stato tenuto costante ed il numero di sottoscrizioni aumentato gradualmente da 0 a 900 con passo 100. Si nota come l'algoritmo che gestisce le sottoscrizioni che non danno luogo a notifica funzioni bene in entrambe le SIB, rimanendo il tempo di inserimento pressoché invariato anche con un numero considerevole di sottoscrizioni.

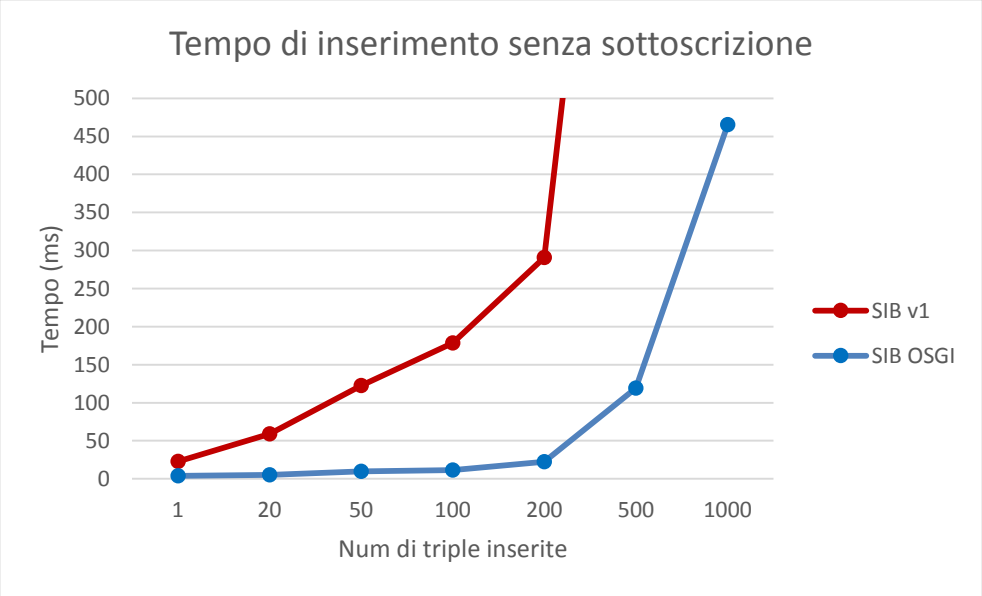


**Figura 9: Confronto tempo di inserimento senza sottoscrizioni SIB-J v1 e SIB-O**

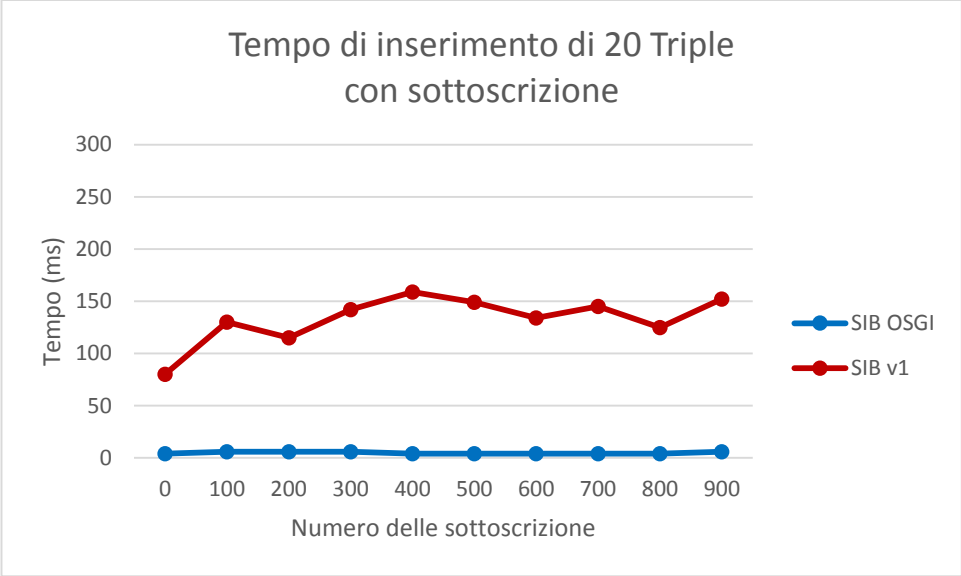


**Figura 10: confronto tempo di inserimento di 20 triple con sottoscrizioni SIB-J v1 e SIB-O**

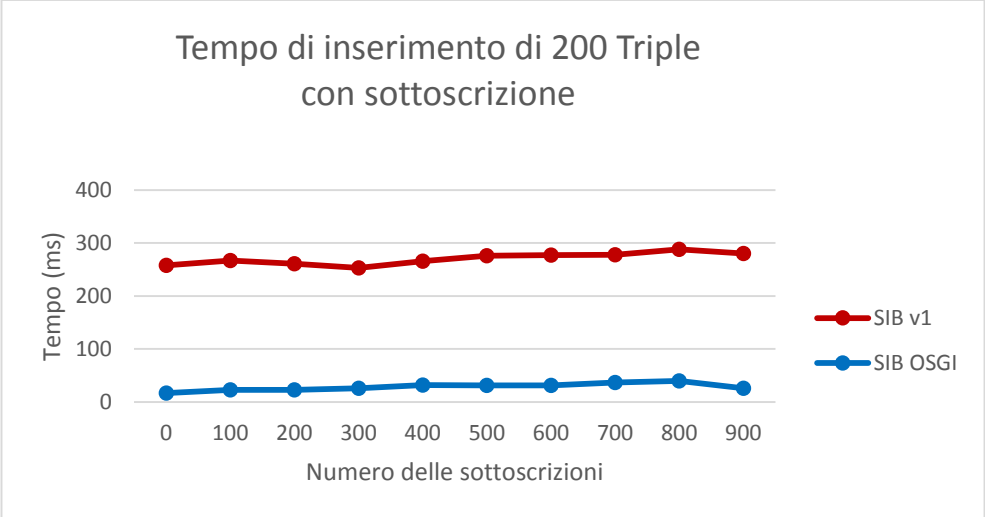


**Figura 11: confronto tempo di inserimento di 200 triple con sottoscrizioni SIB-J v1 e SIB-O**

## 4.3 SIB ottimizzata e SIB Android

Dai primi risultati sembra che la prima versione della SIB - Sib v1 – abbia prestazioni deboli in confronto alla versione OSGI, in particolare nel primo scenario senza sottoscrizioni.

Per verificare l'effettivo miglioramento che ha apportato il refactoring del codice in termini di prestazioni fra la SIB-J v1 e la SIB-J, i 3 test descritti sopra sono stati

eseguiti anche sulla SIB-J e sulla SIB Android e poi graficati insieme per avere una visione complessiva (figure 12,13 e 14).

Negli ultimi tre grafici si può notare come la SIB Android, nonostante abbia a disposizione risorse molto inferiori, si comporti in modo simile alla SIB-J v1 mentre la SIB-J risulta comparabile alla SIB-O di partenza e non fa quindi rimpiangere le prestazioni del framework OSGI.

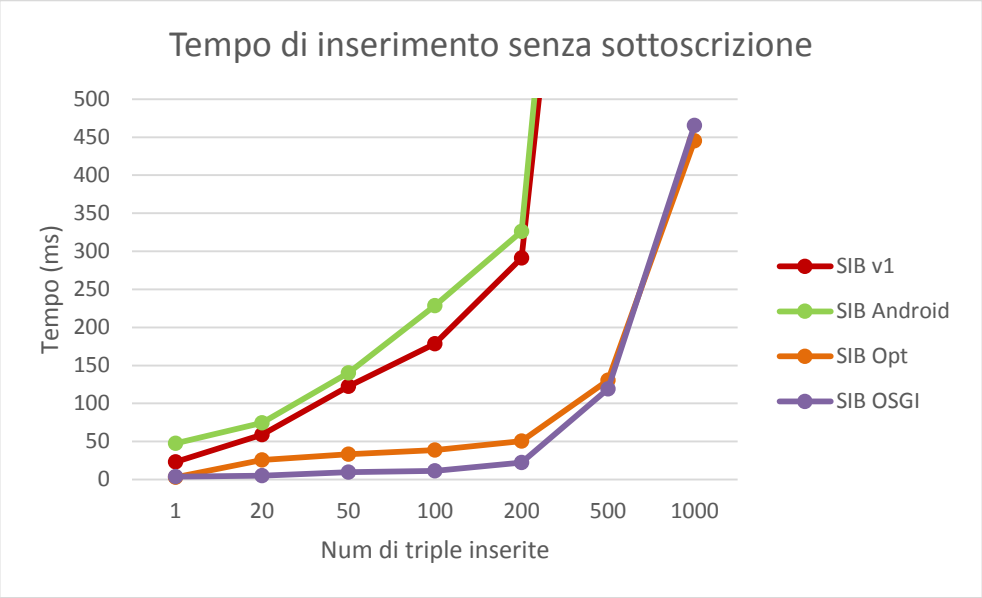


**Figura 12: confronto tempo di inserimento senza sottoscrizioni SIB implementate e SIB-O**

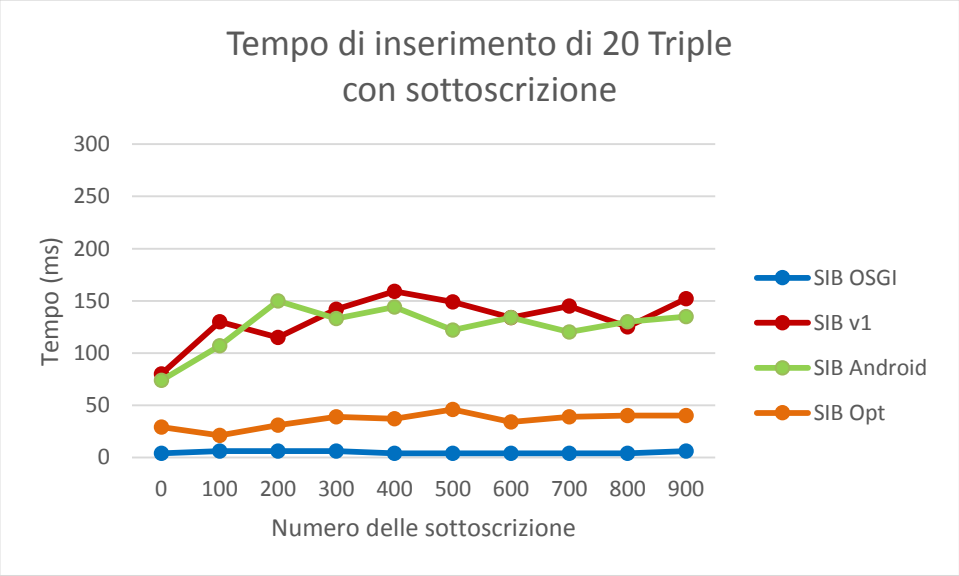


**Figura 13: confronto tempo di inserimento di 20 triple con sottoscrizioni SIB implementate e SIB-O**

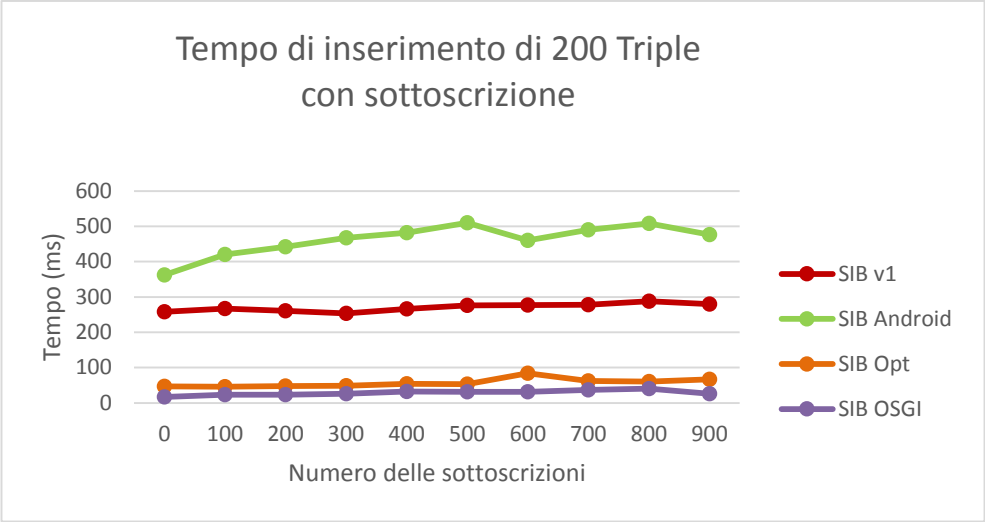


**Figura 14: confronto tempo di inserimento di 200 triple con sottoscrizioni SIB implementate e SIB-O**

# Conclusioni

In conclusione durante la mia tesi ho avuto l'opportunità di studiare i principi alla base delle applicazioni di intelligenza ambientale e di elettronica pervasiva per capire la necessità di avere una base di conoscenza portabile allo scopo di gestire più situazioni possibile. Sono poi passato alla sua effettiva implementazione, partendo da una versione solo parzialmente portabile in tecnologia OSGI. In fine ho anche dimostrato la portabilità del codice integrando il nuovo triple-store da me realizzato su Smartphone Android ed ho caratterizzato dal punto di vista quantitativo i miei prodotti software.

Ho rilasciato in tutto tre versioni: la prima era una versione di sicurezza, molto simile all'originale in cui sostituivo con del codice Java standard le chiamate che nel software originale erano lanciate al framework OSGI. Questa versione si è rivelata poco performante ed ho quindi operato un'ottimizzazione del codice basata sul principio di località e sulla riduzione del numero delle branch condizionate.

La SIB-J ottenuta come risultato dell'ottimizzazione, ha dimostrato in opportuni test, un comportamento comparabile con la versione di partenza pur avendo meno requisiti, essendo essa basata sulla sola Java virtual machine.

Ho scritto e testato la SIB Android a partire dalla SIB-J con un basso numero di modifiche, agendo solo sulle librerie importate, inizialmente non compatibili con Android, sulle fasi di lancio dell'applicativo e sulle funzioni di input-output.

La SIB Android si è rivelata performante in modo simile al mio primo rilascio, ma bisogna tenere in considerazione il fatto che su smartphone le risorse di computazione sono notevolmente inferiori.

# Bibliografia

[1] M. Weiser, The computer for the 21st century, Scientific American (1991) 94–104.

[2] Dey, A.K. Abowd, G.D. Towards a Better Understanding of Context and Context-Awareness. CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness (2000)

[3] Berners-Lee, T., Hendler, J. and Lassila, O. The Semantic Web. Scientific American. 2001.

[4] A portable implementation of Semantic Information Broker in OSGi technology. Reperibile a: https://research.it.abo.fi/confluence/download/attachments/20021391/23%20Delia.ppt.pdf?version=1&modificationDate=1384438616000&api=v2/

[5] Il Sito e il blog ufficiale per lo sviluppo delle apps Android http://developer.android.com/sdk/index.html

[6] Il sito ufficiale del Genymotion Tools per le machine virtuali Android http://www.genymotion.com