Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Sistema adattivo, che integra un motore inferenziale a regole, per la presentazione di contenuti dinamici su dispositivi mobili e sviluppo di un client dedicato.

A. Fiannaca, M. La Rosa, A. Urso

Rapporto Tecnico N.:
RT-ICAR-PA-15-03 **giugno 2015**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sede di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sede di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Sistema adattivo, che integra un motore inferenziale a regole, per la presentazione di contenuti dinamici su dispositivi mobili e sviluppo di un client dedicato.

A. Fiannaca [1], M. La Rosa [1], A. Urso[1]

Rapporto Tecnico N.:
RT-ICAR-PA-15-03

Data:
giugno 2015

---

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Palermo, Viale delle Scienze edificio 11, 90128 Palermo.

Introduzione

Lo scopo del presente documento è la progettazione di un sistema in grado di fornire una fruizione adattiva di servizi web e contenuti multimediali, in modo tale da effettuare un adattamento dinamico della modalità di presentazione dell'informazioni in maniera dipendente dal contesto in cui si verifica l'interazione stessa. L'architettura che verrà delineata in seguito prevede la presenza di un dispositivo mobile che assuma il ruolo di client e di un servizio web che consenta la personalizzazione dei contenuti in modo adattivo. La progettazione del sistema ha richiesto la definizione di un protocollo di comunicazione in grado di trasmettere al server lo stato del dispositivo mobile; per mezzo di questa conoscenza, il servizio web sarà in grado di offrire un livello di presentazione dei contenuti che sarà il più indicato per il client, istante per istante. Al fine di implementare un servizio in grado di adattare tale fruizione dei contenuti, verrà utilizzato un sistema esperto in grado di compiere ragionamenti sfruttando le caratteristiche tipiche dei dispositivi mobili. Uno schema di massima del sistema proposto è riportato in **Figura 1**, dove sono rappresentate le tre componenti di base del lavoro in oggetto, ossia un Modulo di Acquisizione, una Rule Base ed un Modulo di Adattamento.

Nel dettaglio, si avrà un modulo di acquisizione (lato client) che si occuperà di trasformare i segnali rilevati dal dispositivo in un insieme di attributi che verranno inoltrati al server per le successive elaborazioni. Gli attributi ricevuti dal server saranno strutturati nella base di conoscenza (Rule Base) per mezzo di un insieme di template e costituiranno il tipo di dato su cui il sistema eseguirà il ragionamento. Una volta attivate le regole in base agli attributi ricevuti dal server, il terzo modulo, chiamato Modulo di Adattamento, seleziona tramite un processo di inferenza automatica, l'azione correttiva che il layer di presentazione è in grado di attuare.

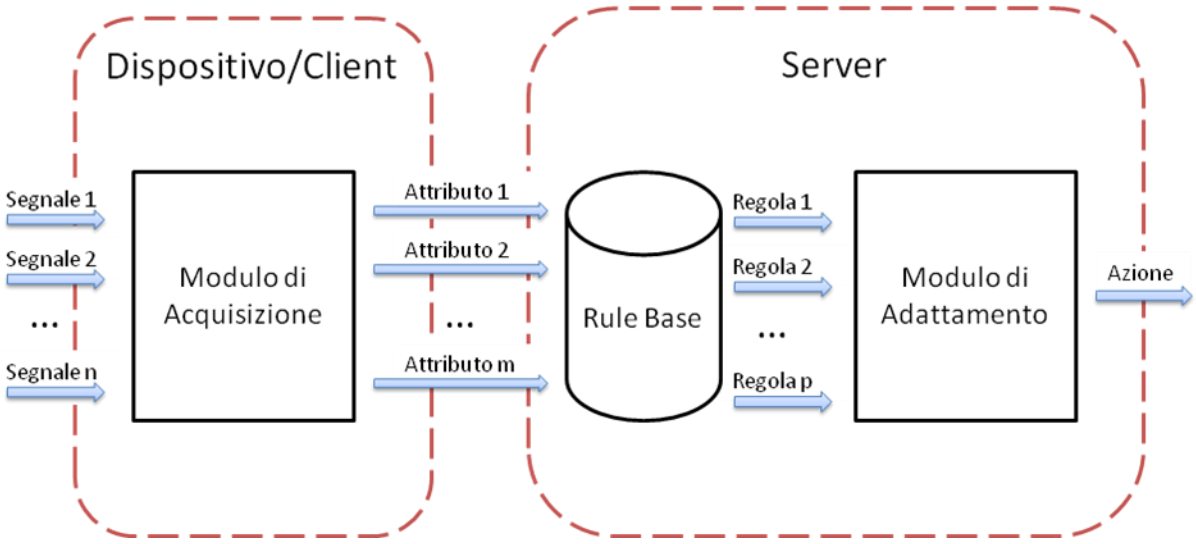

Figura 1 - Schema generale del sistema.

Il presente documento è strutturato nella seguente maniera: nel paragrafo successivo verranno descritte le caratteristiche salienti di un sistema esperto a regole in grado di soddisfare quanto precedentemente descritto, a seguire verrà specificato il protocollo di comunicazione utilizzato per lo scambio di messaggi tra client e server ed infine verrà presentato il progetto del sistema con l'utilizzo della notazione UML.

1. Sistema esperto a regole

Un sistema Rule-Based è un sistema intelligente che è in grado portare a termine delle conclusioni, anche dette inferenze, a partire da un insieme di conoscenza di base. Gli elementi della conoscenza sono chiamati fatti, mentre l'attività di ragionamento, che porta alle inferenze, è realizzato attraverso un insieme di regole che agiscono sui fatti. Le regole sono solitamente scritte nel formato tradizionale *if-then* dei più comuni linguaggi di programmazione. La parte sinistra di una regola (*if*) è chiamata predicato o premessa; la parte destra (*then*) è chiamata azione o conclusione. L'insieme dei fatti e delle regole costituisce la cosiddetta base di conoscenza (*knowledge base*).

I sistemi rule-based non sono sistemi generici: essi sono progettati ed impiegati per uno specifico dominio applicativo. Un dominio rappresenta l'ambito in cui il sistema agisce, ossia tutto l'insieme di informazioni su cui il sistema può svolgere la sua attività di ragionamento automatico per mezzo delle regole.

I sistemi rule-based sono anche conosciuti come sistemi esperti (*expert systems*) poiché essi catturano e codificano la conoscenza di un esperto umano di un particolare dominio. Usando questa definizione, le regole possono essere viste come la *skill* (abilità) tipiche di un esperto umano. I sistemi esperti hanno rappresentato una delle principali tematiche di ricerche nell'intelligenza artificiale negli anni 70 e 80. I primi sistemi esperti sono stati sviluppati in ambito medicale: uno dei primi sistemi esperti di successo è stato infatti MYCIN, un programma per la diagnosi di infezioni batteriche del sangue.

Oggi, i sistemi generici rule-based, sia quelli progettati per sostituire un esperto umano, sia quelli utilizzati per automatizzare o codificare pratiche di business o altre attività. Fanno parte virtualmente di ogni impresa (*enterprise*). Questi sistemi sono usati continuamente per ordinare forniture, per monitorare processi industriali, per elaborare pagine web.

La principale differenza tra i sistemi rule-based e i comuni sistemi di elaborazione risiede nel loro paradigma di programmazione. I programmi tradizionali utilizzano un approccio procedurale, nel senso che il programmatore decide che cosa fare (*"what to do"*), come farlo (*"how to do"*) e in che ordine. La programmazione procedurale è adatta per quei problemi in cui gli input sono ben specificati e per iquali esiste un insieme ben noto di passi da portare avanti per risolvere un problema, cioè esiste un algoritmo deterministico. I calcoli matematici, per esempio, rappresentano il caso migliore in cui adottare un approccio procedurale.

I sistemi rule-based, d'altra parte, usano un approccio dichiarativo: un programma dichiarativo definisce soltanto cosa fare (*"what to do"*), ma non fornisce istruzioni su come farlo (*"how to do"*). Ciò significa che i programmi dichiarativi hanno bisogno di un qualche tipo di sistema runtime che sia in grado di utilizzare quell'informazione dichiarativa al fine di effettuare conclusioni, o inferenze. Poiché i programmi dichiarativi includono soltanto i dettagli importanti di una soluzione, essi possono essere compresi più facilmente rispetto ai programmi procedurali. Inoltre, dal momento che il controllo di flusso è scelto dal sistema runtime, un programma dichiarativo può essere più flessibile nel caso di input frammentario o incompleto. Un approccio dichiarativo è particolarmente indicato soprattutto per la risoluzione di quei problemi senza una chiara e ben definita soluzione algoritmica, come per esempio la classificazione, la predizione, la diagnosi che hanno solitamente delle euristiche o delle linee guida piuttosto che un insieme predefinito di istruzioni.

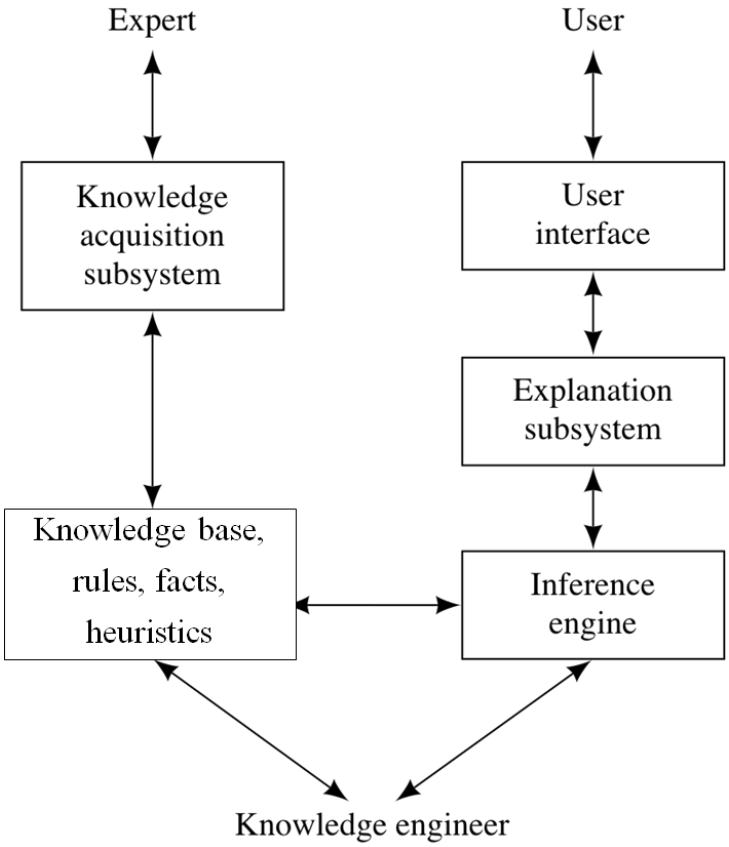

Figura 2 - Elementi dell'interazione utente/esperto del dominio/ingegnere della conoscenza.

a. Architettura di un sistema rule-based

Le componenti principali di un tipico sistema rule-based sono la knowledge base (kb) ed il motore inferenziale (*inference engine*). La kb a sua volta è formata dalla *working memory*, contenente i fatti, e dalla *rule base*, contenente le regole. Il motore inferenziale invece consiste di un *pattern matcher*, un'*agenda* ed un *execution engine*. Queste componenti sono rappresentate schematicamente in **Figura 3**.

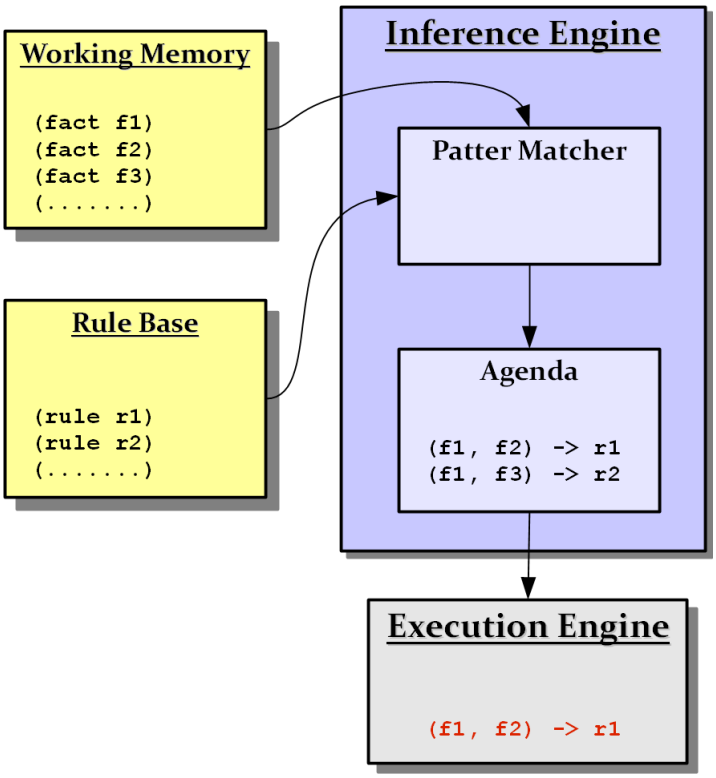

Figura 3 - Architettura funzionale di un sistema rule-based.

i. Il motore inferenziale

Il compito principale del motore inferenziale è quello di applicare le regole ai dati (fatti). Il motore inferenziale controlla l'intero meccanismo di applicazione delle regole della rule base ai fatti della working memory per ottenere gli output del sistema.

Il motore inferenziale solitamente funziona attraverso una serie di cicli discreti definiti di seguito:

2. Tutte le regole sono confrontate con la working memory, usando il pattern-matcher (cfr 2.1.4), per decidere quali regole dovrebbero essere attivate durante questo ciclo. La lista non ordinata delle regole attivate, insieme a tutte le altre regole attivate nei cicli precedenti, viene chiamata *conflict set*.
3. Il conflict set è ordinato per formare l'agenda (cfr. 2.1.5), ovvero la lista di regole la cui parte destra deve essere eseguita (*fired*). Il processo di ordinamento dell'agenda, cioè l'ordine di esecuzione delle regole, è chiamato *conflict resolution*. La strategia di conflict resolution dipende da molti fattori, alcuni controllabili dal programmatore.
4. Per completare il ciclo, la prima regola dell'agenda viene eseguita e l'intero processo ricomincia d'accapo. Questa ripetizione implica una gran quantità di lavoro ridondante, ma molti motori inferenziali adottano tecniche sofisticate per evitare la maggior parte della ridondanza. In particolare, i risultati del pattern matcher e quelli del risolutore di conflitti dell'agenda possono essere conservati durante i cicli, così che ad ogni nuova iterazione devono essere fatte soltanto le elaborazioni inedite.

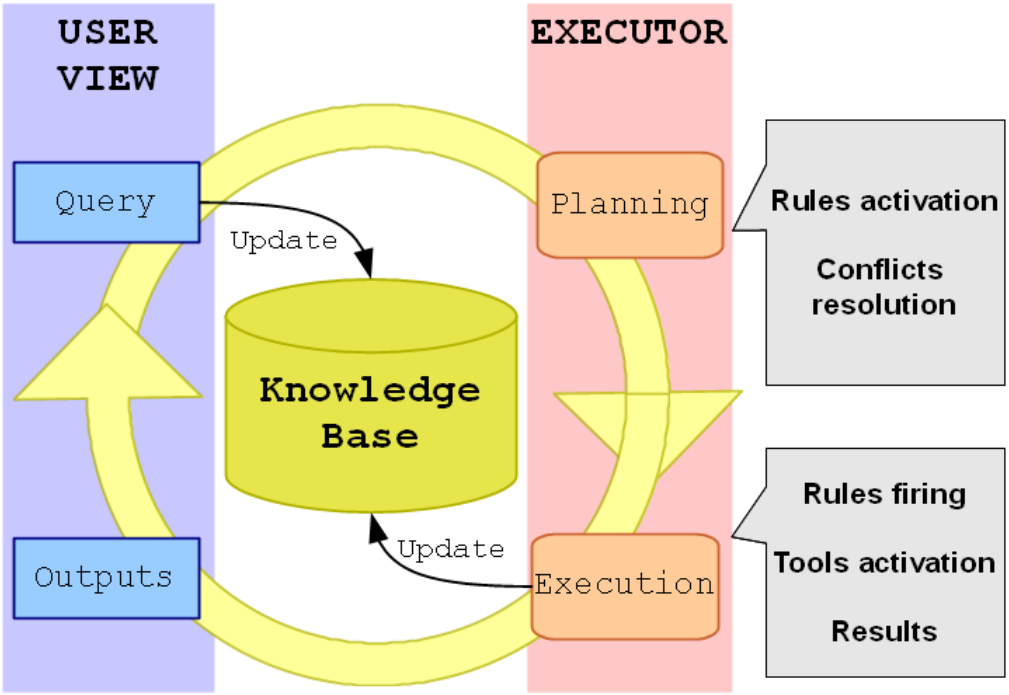

Figura 4 - Modalità di ragionamento di un sistema rule-based.

i. La rule base

La rule base contiene tutte le regole che il sistema conosce. Quest'ultime possono essere semplicemente memorizzate come stringhe di testo, ma molto spesso un rule compiler le trasforma in una forma che il motore inferenziale può elaborare in maniera più efficiente. In aggiunta il rule compiler può aggiungere o riorganizzare le premesse o le conclusioni di una regola, sia per renderle più efficienti, sia per chiarirne il significato per l'esecuzione automatica. A seconda del particolare motore a regole, questi cambiamenti possono risultare invisibili per il programmatore.

Alcuni motori inferenziali permettono, ed in alcuni casi richiedono, di memorizzare la rule base in un database relazionale esterno. Utilizzando un database esterno consente di selezionare le regole da includere in un sistema basandosi su criteri come la data, l'ora e i diritti di accesso dell'utente.

ii. La working memory

In un sistema rule-based, la working memory, chiamata alcune volte anche fact base, contiene tutti i pezzi di informazione, i fatti appunto, che il sistema è in grado di elaborare. Solitamente il motore inferenziale mantiene uno o più indici, simili a quelli utilizzati in un database relazionale, per rendere più veloce la ricerca nella working memory. La working memory può contenere oggetti di diverso tipo, come per esempio oggetti Java.

iii. Il pattern matcher

Il compito del pattern matcher è quello di decidere quali regole applicare, dato il contenuto corrente della working memory. In generale, questo è un problema difficile da risolvere. Se la working memory contiene migliaia di fatti, ed ogni regola ha due o tre premesse, allora il pattern matcher potrebbe avere bisogno di cercare attraverso milioni di possibili combinazioni di fatti per individuare quelle combinazioni che soddisfano le regole. Fortunatamente, esistono molti approcci efficienti per la risoluzione di questo problema. Ancora oggi, comunque, per la maggior parte dei programmi rule-

based, la fase di pattern matching rappresenta la parte più costosa dal punto di vista computazionale di tutto il processo decisionale.

iv. L'agenda

Dopo che, grazie al pattern matcher, il motore inferenziale ha individuato quali sono le regole che andrebbero eseguite, esso deve ancora decidere l'ordine di esecuzione di tali regole. La lista delle regole che potenzialmente possono essere eseguite è conservata nell'agenda. L'agenda è responsabile dell'utilizzo di una strategia di risoluzione dei conflitti per decidere quali regole, tra tutte quelle attivate, hanno la priorità più alta e dovrebbero essere quindi eseguite per prima. Ancora una volta, questo è un problema potenzialmente difficile da gestire, ed ogni motore inferenziale implementa il proprio approccio. Nei casi più comuni, la strategia di risoluzione dei conflitti tiene in considerazione la specificità o la complessità di ogni regola ed il tempo di permanenza della regola nell'agenda. Alcune regole possono anche avere una specifica priorità, così che tali regole sono più importanti delle altre e sono eseguite per prima.

v. L'execution engine

Dopo che il motore inferenziale decide quale regola eseguire, occorre materialmente compiere l'azione indicata nella parte destra della regola selezionata. L'execution engine è il componente preposto all'esecuzione delle regole. In un classico sistema a produzioni come MYCIN, le regole non fanno altro che aggiungere, togliere o modificare fatti nella working memory. Nei sistemi moderni, l'esecuzione di una regola può avere un ampio raggio di effetti. I sistemi moderni, infatti, offrono un linguaggio di programmazione completo in modo tale che è possibile gestire e definire in maniera molto particolareggiata ciò che accade quando una regola viene eseguita.

5. I Web Service

I Web service sono sistemi software progettati per supportare l'interoperabilità tra dispositivi di differente natura che condividono la medesima rete.

La principale caratteristica di un Web Service è quella di offrire un'interfaccia software (descritta in un formato standard quale, per esempio, il Web Services Description Language) attraverso la quale altri sistemi possono interagire con il Web Service stesso , attraverso l'attivazione di operazioni descritte nell'interfaccia tramite appositi "messaggi" inclusi in una "busta" (la più famosa è SOAP): tali messaggi sono, solitamente, trasportati tramite il protocollo HTTP e formattati secondo lo standard XML. Quest'ultimo linguaggio, in supporto ad una architettura chiamata Service oriented Architecture – SOA, consente la comunicazione di applicazioni sviluppate in diversi linguaggi ed implementati su diverse piattaforme hardware, che viene implementata, tramite le interfacce che queste "espongono" pubblicamente e mediante l'utilizzo delle funzioni che sono in grado di effettuare (i "servizi" che mettono a disposizione) per lo scambio di informazioni e l'effettuazione di operazioni complesse (quali, ad esempio, la realizzazione di processi di business che coinvolgono più aree di una medesima azienda) sia su reti aziendali come anche su Internet: la possibilità dell'interoperabilità fra diversi linguaggi di programmazione (ad esempio, tra Java e Python) e diversi sistemi operativi (come Windows e Linux) è resa possibile dall'uso di standard "open". La principale differenza tra questo modello e quello basato sul paradigma client/server, è che i primi incapsulano le interfacce applicative originarie con un nuovo strato software che consente l'utilizzo di XML ed il protocollo http per effettuare le chiamate e ricevere le risposte attraverso la rete internet. La parte di servizio che incapsula il server è chiamata "Producer", mentre la parte relativa al client è chiamata "Consumer". L'architettura che sta alla base dei Web services è ben definita e sempre la stessa, indipendentemente dai Web services specifici dei vari fornitori utilizzati. L'architettura descrive le interazioni tra tre componenti principali:

- Il ***Web Service Provider***, che sviluppa e definisce i Web Services e li pubblica poi all'interno dei Web Service Registries, oppure li rende direttamente disponibili ai Web Service Requester.
- Il ***Web Service Requester***, che effettua un'operazione di ricerca per localizzare i servizi desiderati resi disponibili dai Web Service Provider e poi richiedono questi stessi servizi ai Web Service Registries oppure direttamente al luogo della loro pubblicazione. Una volta localizzato, il Web Service Requester si connette al suddetto servizio.
- I ***Web Service Registries***, che assumono il ruolo di directory centralizzate, e sono depositari dei Web Services definiti e pubblicati dai Web Service Provider.

In **Figura 5** sono mostrate le relazioni tra le tre componenti.

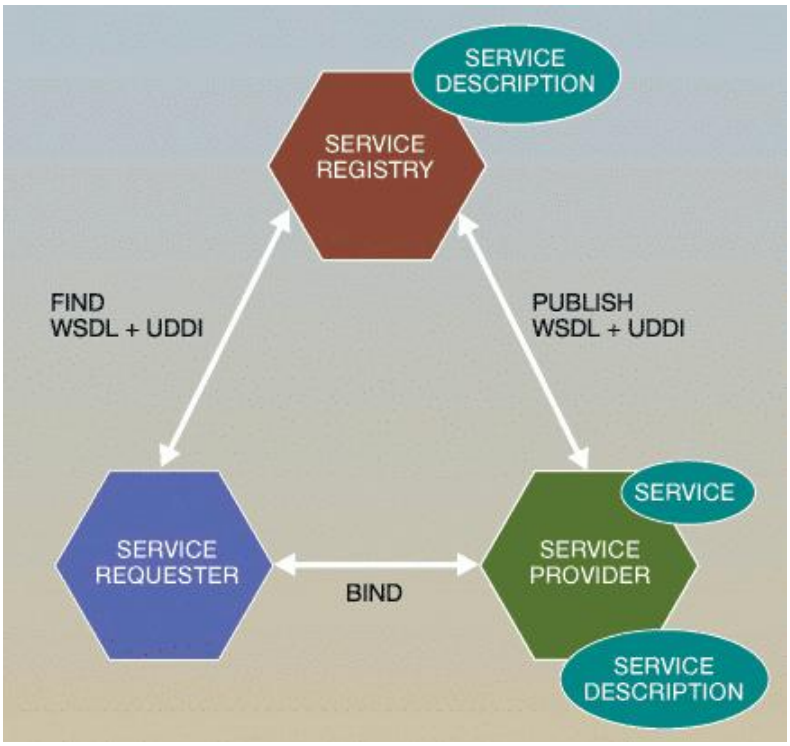

Figura 5 - Attori, oggetti e operazioni dei servizi web.

a. I protocolli usati dai Web Service

I protocolli utilizzati per definire, localizzare, realizzare e far interagire tra di loro i Web Service, possono essere suddivisi in 4 aree:

- Trasporto del servizio: responsabile per il trasporto dei messaggi tra le applicazioni in rete, include protocolli quali HTTP, SMTP, FTP, XMPP ed il recente Blocks Extensible Exchange Protocol (BEEP).
- XML Messaging: tutti i dati scambiati sono formattati mediante "tag" XML in modo che gli stessi possano essere utilizzati ad entrambi i capi delle connessioni; il messaggio può essere codificato conformemente allo standard SOAP, come anche utilizzare JAX-RPC, XML-RPC o REST.
- Descrizione del servizio: l'interfaccia pubblica di un Web Service viene descritta tramite WSDL (Web Services Description Language) un linguaggio basato su XML usato per la creazione di "documenti" descrittivi delle modalità di interfacciamento ed utilizzo del Web Service.

- Elencazione dei servizi: la centralizzazione della descrizione e della localizzazione dei Web Service in un "registro" comune permette la ricerca ed il reperimento in maniera veloce dei Web Service disponibili in rete; a tale scopo viene attualmente utilizzato il protocollo UDDI.

Ulteriori protocolli standard utilizzati sono:

- WS-Security: il protocollo Web Services Security protocol è stato adottato come standard OASIS; tale standard permette l'autenticazione degli utenti e la confidenzialità dei messaggi scambiati con l'interfaccia del Web Service
- WS-Reliability: si tratta di specifiche basate su SOAP ed accettate come standard OASIS che soddisfano la richiesta di messaggi "affidabili" (reliable), richiesta critica per alcune delle applicazioni che utilizzano i Web Service (come, ad esempio, transazioni monetarie o applicazioni di E-commerce).

Naturalmente, quando si crea un Web Service producer, non è strettamente necessario crearne anche la descrizione; questo soprattutto se l'utilizzo del Web Service viene riservato ad un numero ristretto di soggetti, nel qual caso le stesse informazioni possono essere fornite direttamente. La descrizione WSDL è però generata automaticamente dai principali tool di sviluppo per i Web Services e può essere utilizzata per creare automaticamente la struttura di base dei Web Service consumer corrispondenti.

b. Linguaggo di definizione dei servizi WSDL

Per meglio comprendere come funziona il lingaggio WSDl si può ipotizzare lo scenario di una banca che vuole fornire un servizio di approvazione automatica di prestiti che permette ai correntisti della banca di chiedere un prestito direttamente on-line. Supponendo che l'utente si sia precedentemente registrato, questi non dovrà far altro che indicare il proprio identificativo che corrisponderà al proprio numero di conto corrente e la somma di denaro richiesta. Da questa breve descrizione si evince come il servizio metta a disposizione dell'utente un'operazione (approva, per esempio) che richiederà dei dati in ingresso e in uscita. Attraverso WSDL è possibile formalizzare tutte queste caratteristi che del servizio secondo uno schema che non differisce molto dalla specifica delle API (Application Program Interface) di un sistema.

L'identificazione delle componenti fondamentali di un file WSDL parte dal "service" che identifica un insieme logico di servizi, ognuno dei quali è specificato da una "port", che individua l'indirizzo a cui il servizio risponde e il tipo di protocollo supportato dal servizio stesso e non le caratteristiche del servizio. Per questo motivo a ogni port è associata una portType il cui compito è quello di specificare la reale sintassi del servizio. La portType ha, quindi, il ruolo di dire cosa il servizio fa, mentre la port specifica come il servizio possa essere acceduto. Questo tipo di specializzazione viene specificato attraverso il legame che data una portType descrive in che modo questa si inserisce all'interno di un particolare protocollo quale SOAP, HTTP o SMTP. Per accedere a questo servizio è necessario che il client inoltri le proprie richieste secondo uno dei protocolli sopra elencati. Nel dettaglio, una portType è composta da una serie di operation che rispecchiano le funzionalità fornite dal servizio e che interagiscono con l'utente e che possono basarsi su uno dei quattro pattern predefiniti:

- One_way: l'operazione è composta da un solo messaggio in ingresso al fornitore del servizio.
- Request_response: l'operazione prevede una risposta del fornitore del servizio successiva a un messaggio ricevuto dall'utente.
- Solicit_Response: l'operazione prevede l'attesa da parte del fornitore del servizio, di una risposta a fronte di una richiesta effettuata dal fornitore stesso.
- Notification: l'operazione è composta da un solo messaggio in uscita al fornitore del servizio.

Indipendentemente dal pattern, ogni interazione cliente-fornitore del servizio avviene attraverso uno scambio di messaggi opportunamente identificati in WSDL attraverso il tag "message", grazie al quale è possibile specificare il formato del messaggio. In particolare, ogni messaggio è visto come un insieme di una o più port ognuna delle quali aderente a un tipo di dato che può essere uno di quelli primitivi di XML (per esempio, int e string), oppure un tipo di dato definito in types.

Al di là delle specifiche sintattiche di WSDL, soffermandosi ulteriormente sulla relazione che esiste tra port e portType, è possibile dire che una port può essere vista come una specializzazione di una portType operata secondo un particolare protocollo di comunicazione. È quindi possibile che all'interno del medesimo file WSDL la stessa portType, quindi il servizio, possa essere reso accessibile, grazie ai binding, su diversi protocolli di trasporto. Al momento accanto alla specifica di

WSDL sono stati specificati i binding su SOAP, HTTP e SMTP; è quindi possibile specificare solo Web Service che comunicano attraverso questi protocolli. Sulla base di ciò un buon modo di operare prevede che una specifica WSDL di un servizio sia composta da almeno due file: il primo chiamato WSDL Interface document composto solo dalla specifica dei tipi, dei messaggi e delle portType, quindi per ogni tipo di binding verrà definito un documento apposito detto anche WSDL implementation document.

Sebbene risulti molto generico, WSDL è al tempo stesso molto esauriente e in grado di descrivere servizi di diverse tipologie. L'utilizzo dei quattro pattern di comunicazione permette, infatti, di specificare sia servizi asincroni che sincroni. Nel primo caso saranno utilizzati solo operazioni di tipo one-way e notification, mentre nel secondo caso di tipo request-response e solicit-response. Di seguito lo schema utilizzato per la determinazione della porta corretta per il servizio richiesto.

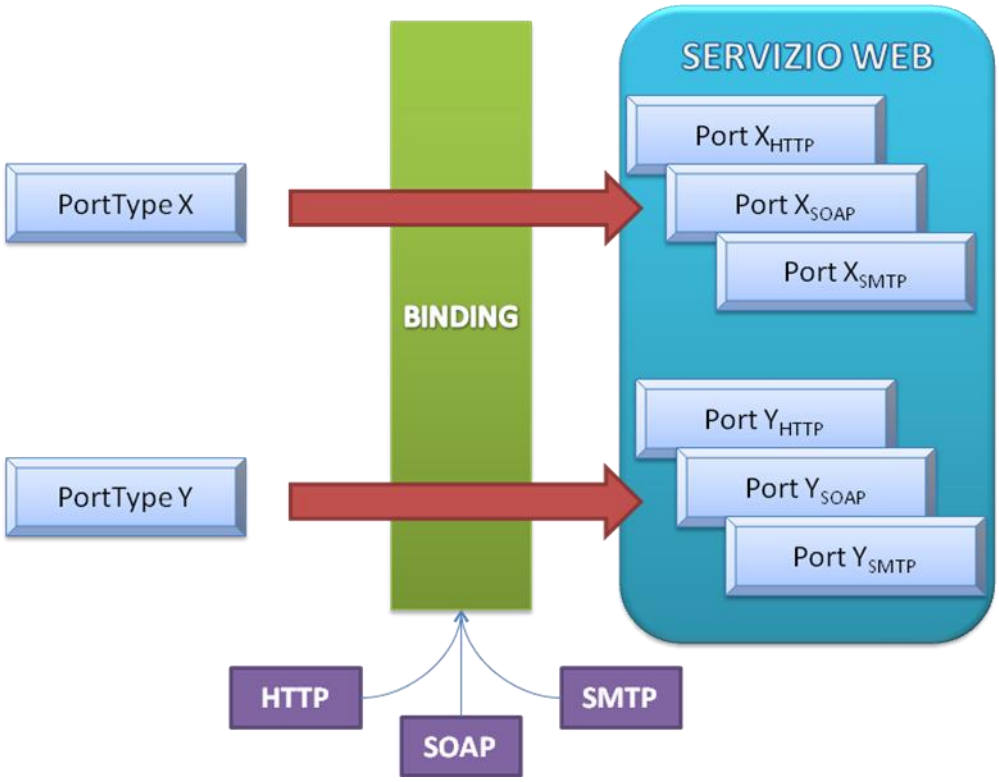

Figura 6 - Processo di binding che definisce le tecnologie utilizzate per implementare le operazioni offerte da un portType.

6. Protocollo SOAP: accesso ai servizi web

Il Protocollo "Simple Object Access Protocol", o SOAP, è un protocollo basato su XML che consente alle applicazioni di scambiare informazioni sul protocollo http: più semplicemente SOAP è un protocollo per accedere ai web service. SOAP offre la possibilità di far comunicare due o più applicazioni che vengono eseguiti su sistemi operativi differenti, con differenti tecnologie e linguaggi di programmazione. Tale comunicazione viene realizzata sfruttando il protocollo HTTP (over HTTP), poiché quest'ultimo è supportato da tutti i browser e server. SOAP non definisce alcuna semantica delle applicazioni, come un modello di programmazione o una specifica semantica, bensì definisce un meccanismo semplice per esprimere la semantica delle applicazioni, fornendo un modello di impacchettamento modulare e la codifica dei meccanismi per la codifica di dati all'interno di moduli.

Ciò consente a SOAP di essere impiegato in una grande varietà di architetture che vanno dai sistemi di messaggistica a RPC.

SOAP si compone di quattro parti:

- La busta SOAP, definisce un quadro generale per esprimere che **cosa** c'è nel messaggio, **chi** dovrebbe affrontare il problema, e **se** esso è facoltativo o obbligatorio.
- La descrizione di come un messaggio SOAP dovrebbe essere trasportato usando http (per interazioni tipo web) o SMTP (per informazioni di tipo e-mail)
- Le regole di codifica SOAP, definiscono un meccanismo di serializzazione che può essere utilizzato per lo scambio di istanze di tipi di dati definiti dall'applicazione.
- La rappresentazione RPC di SOAP, definisce una convenzione che può essere utilizzata per rappresentare le chiamate e risposte a procedure remote.

Sebbene queste parti sono descritte insieme come parte del protocollo, esse sono funzionalmente ortogonali. In particolare, la busta e le regole di codifica sono definite in namespace differenti al fine di promuovere la semplicità attraverso la modularità. Rispetto ai protocolli più pesanti (come ORPC), SOAP possiede due proprietà fondamentali: inviare e ricevere pacchetti sul protocollo di trasporto ed elaborare messaggi XML.

Un esempio di messaggio distribuito attraverso il protocollo SOAP è riportato in **Figura 7**. Tale esempio appare alquanto interessante poiché rispecchia il caso di studio in oggetto. In figura è possibile notare come la comunicazione parte dall'applicazione sul client (1) che imbusta la richiesta in un messaggio SOAP che viene instradato sul protocollo di trasporto HTTP. Va evidenziato il fatto che il protocollo SOAP non viene direttamente sottoposto al controllo del firewall in quanto usa lo stesso approccio di un metodo HTTP POST, per tale ragione un messaggio SOAP non viene direttamente filtrato da un firewall standard. Una volta giunto a destinazione (2), il messaggio viene spacchettato ed interpretato dall'applicazione web-service ed inviato (3) alla infrastruttura vera e propria del web service. Lo stesso percorso (4-5-6) verrà utilizzato per la risposta del provider alla richiesta SOAP.

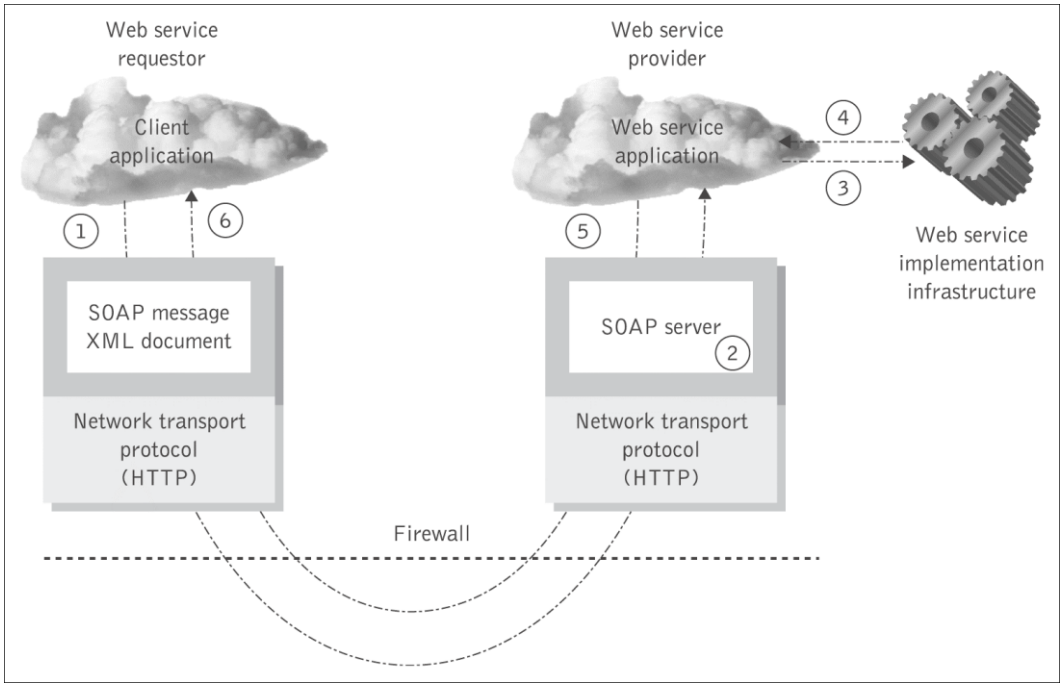

Figura 7 - Messaggio distribuito con SOAP. La figura è estratta da "*Michael P. Papazoglou, Web Services, 1st Edition, © Pearson Education Limited 2008*".

a. Il modello di scambio dei messaggi in SOAP

I messaggi SOAP sono fondamentalmente trasmessi in modo monodirezionale dal mittente al ricevente, ma vengono spesso combinati allo scopo di implementare dei pattern di "request/response";infatti, le implementazioni SOAP possono essere ottimizzate per sfruttare le caratteristiche di uno particolare sistema. Indipendentemente dal protocollo SOAP a cui è legato, i messaggi vengono instradati lungo un cosiddetto "percorso del messaggio", che consente l'elaborazione a uno o più nodi intermedi, prima di arrivare al nodo di destinazione.

Un'applicazione SOAP che riceve un messaggio SOAP deve elaborare il messaggio eseguendo le seguenti operazioni nell'ordine indicato di seguito:

1. Identificare tutte le parti del messaggio SOAP destinati a tale applicazione;
2. Verificare che tutte le parti obbligatorie individuate nella fase 1 siano supportate dall'applicazione di questo messaggio ed di conseguenza effettuare l'elaborazione. In caso contrario, rifiutare il messaggio. Il processore deve ignorare le parti opzionali identificate nella fase 1, senza compromettere l'esito dell'elaborazione.
3. Se l'applicazione SOAP non è la destinazione finale del messaggio, ne segue che tutte le parti del messaggio identificate nella fase 1 devono essere rimosse prima di inoltrare il messaggio.

L' elaborazione di un messaggio, o di una parte di esso, necessita che il processore SOAP sia in grado di comprendere, tra le altre cose, il modello di scambio utilizzato (monodirezionale, request/response, multicast, ecc), il ruolo del destinatario, l'impiego (se presente) di meccanismi RPC e la rappresentazione o la codifica dei dati, così come altre semantiche necessarie per la corretta elaborazione.

Come detto in precedenza, tutti i messaggi SOAP sono codificati usando l'XML, seguendo alcune importanti regole sintattiche. Un'applicazione SOAP dovrebbe includere un adeguato *namespace* in tutti gli elementi e attributi definiti in SOAP, all'interno del messaggio generato. Specularmente, un'applicazione SOAP deve essere in grado di elaborare un *namespace* presente nel messaggio ricevuto ed eliminare i messaggi che contengono un *namespace* non riconosciuto, mentre potrebbe elaborare messaggi senza *namespace* come se, al contrario, fossero presenti. Il protocollo SOAP definisce due namespace:

- La busta SOAP (envelope) ha il namespace identificatore: *"http://schemas.xmlsoap.org/soap/envelope/"*
- La serializzazione SOAP (serialization) ha l'identificatore: *"http://schemas.xmlsoap.org/soap/encoding/"*

Un messaggio SOAP non deve contenere un Document Type Declaration, né delle istruzioni di elaborazione. Per specificare l'identificatore unico di elemento codificato, SOAP utilizza l'attributo "id", mentre l'attributo "href" specifica il riferimento a valore, in modo conforme alle specifiche XML.

b. L'imbustamento dei messaggi SOAP

Un messaggio SOAP è un documento XML che consiste di una busta SOAP obbligatoria, un'intestazione SOAP opzionale, e un corpo SOAP obbligatoria. Questo documento XML è indicato come un messaggio SOAP per il resto di questa specifica. L'identificatore *namespace* per gli elementi e gli attributi definiti in questa sezione è "*http://schemas.xmlsoap.org/soap/envelope/*". Un messaggio SOAP è composto dai seguenti elementi, riportati anche in **Figura 8**:

- Un elemento "***Envelope***" che identifica il documento XML come un messaggio SOAP;
- Un elemento "***Header***" che rappresenta il meccanismo utilizzato per aggiungere caratteristiche al messaggio, senza che sia richiesto un preventivo accordo tra le parti: SOAP definisce alcuni attributi che possono essere usati per indicare chi dovrebbe occuparsi di una funzione e se questa è facoltativa o obbligatoria;

- Un elemento "***Body***" che rappresenta un contenitore di informazioni obbligatorie destinate al destinatario finale del messaggio; nel body è definito anche l'elemento "Fault" utilizzato per la segnalazione di errori e informazioni sullo stato.

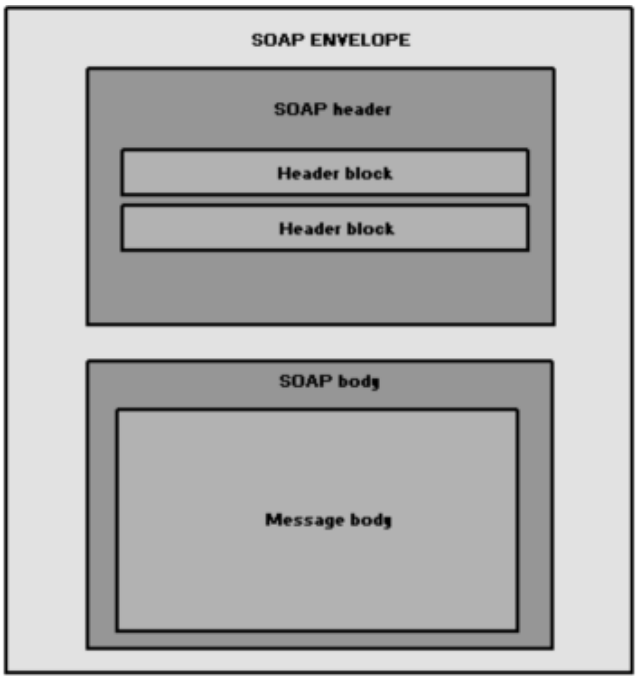

Figura 8 - Struttura del pacchetto SOAP.

Le regole grammaticali, utilizzate per gli elementi sopraindicati, sono le seguenti:

1) Imbustamento:
   - Il nome dell'elemento è "Envelope".
   - L'elemento deve essere presente nel messaggio SOAP.
   - L'elemento potrebbe contenere dichiarazioni di namespace, così come attributi aggiuntivi. Se presenti, tali attributi devono possedere un namespace qualificato. Allo stesso modo, l'elemento potrebbe contenere sotto elementi aggiuntivi: se presenti questi ultimi devono contenere dei namespace qualificati e l'elemento Body SOAP.
2) Intestazione:
   - Il nome dell'elemento è "Header".
   - L'elemento può essere presente in un messaggio SOAP. Se presente, l'elemento deve essere il primo elemento figlio diretto di un elemento SOAP Envelope.
   - L'elemento può contenere un insieme di voci di intestazione, ognuno dei quali è un elemento figlio diretto dell'elemento SOAP Header. Tutti gli elementi figlio immediati del elemento Header SOAP DEVE avere un namespace qualificato.
3) Corpo:
   - Il nome dell'elemento è "Body".
   - L'elemento deve essere presente in un messaggio SOAP e deve essere il figlio diretto di un elemento SOAP envelope. Se è presente un elemento header, il body lo deve seguire; diversamente, il corpo deve essere il primo figlio diretto dell'elemento SOAP envelope.
   - L'elemento può contenere un di voci body, essendo ognuno un figlio diretto dell'elemento SOAP body. Tali figli diretti possono avere un namespace qualificato.

Altra caratteristica dei messaggi SOAP è quella di fornire un meccanismo flessibile per estendere il messaggio in modo decentralizzato e modulare senza una conoscenza a priori tra le parti interessate alla

comunicazione. Degli esempi di estensioni che possono essere implementate come voci “header” sono: l’autenticazione, il controllo di transazione, il pagamento, ecc.

7. Analisi dei requisiti del sistema.

In questo capitolo verranno riportati i diagrammi UML relativi alla fase progettuale, al fine di stabilire che cosa il sistema in questione deve essere in grado di rispondere alle esigenze dell’utente. In seguito alla descrizione tecnologica riportata in questo documento, si è scelto di dividere il sistema in due sottosistemi; uno fungerà da Client e verrà eseguito sul dispositivo mobile, mentre l’altro sarà costituito dal Server, che sarà accessibile da remoto. Quest’ultimo infatti, metterà a disposizione del client tutte le interfacce necessarie al fine di sfruttare le potenzialità dei web service ed ottenere l’adattamento visuale più idoneo al contesto. Pertanto, saranno trattati in parallelo i due sottosistemi, seguendo la logica di base del protocollo SOAP utilizzato nella comunicazione asincrona tra i due sottosistemi.

a. Diagrammi dei Casi d’Uso

A seguire sono riportati i diagrammi dei casi d’uso, che esprimono i comportamenti del sistema in oggetto, offerti o desiderati, sulla base dei suoi risultati osservabili. Sono riportati i due diagrammi dei casi d’uso principali, ossia quello inerente al Dispositivo e quello relativo al Server.

i. Diagramma dei casi d’uso dell’interazione col dispositivo mobile

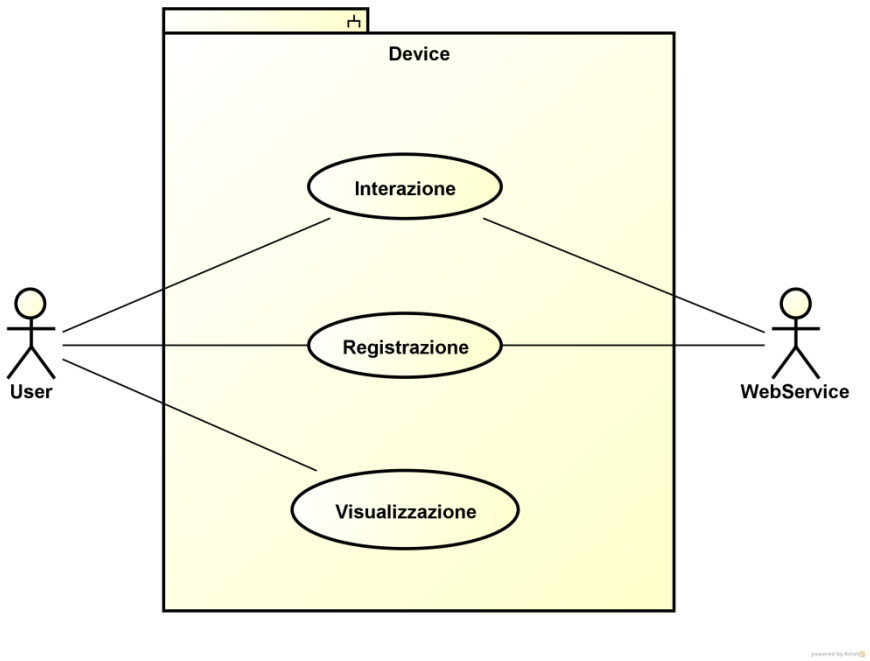

Figura 9 - Diagramma dei casi d'uso n° 1.

Descrizione dei casi d’uso dell’interazione col dispositivo.

Le funzionalità che andranno implementate nel dispositivo sono le seguenti.

- Interazione: il sistema sarà in grado di raccogliere gli input dell'Utente e dell'ambiente e tradurli in segnali. Dopo di ciò, il dispositivo provvederà ad impacchettare questi ultimi ed inviarli al WebService che si occuperà delle fasi successive dell'elaborazione.
- Registrazione: il sistema si occuperà della fase di registrazione dell'Utente nel WebServer, al fine di garantire la sicurezza e l'affidabilità delle successive operazioni. Questa funzionalità entrerà in gioco durante il primo accesso del dispositivo al servizio web.
- Visualizzazione: Il sistema che viene eseguito nel dispositivo si occuperà della visualizzazione delle pagine e dei contenuti web disponibili per l'Utente.

Attori

Gli attori che andranno ad interagire col sistema in questo caso d'uso si dividono in:

- ✓ Attore principale → **Utente**. Colui che accede e sfrutta il sistema.
- ✓ Attore secondario→ **WebServer**: Sistema IT esterno responsabile dell'adattamento dei contenuti web.

ii. Diagramma dei casi d'uso inerenti al servizio web

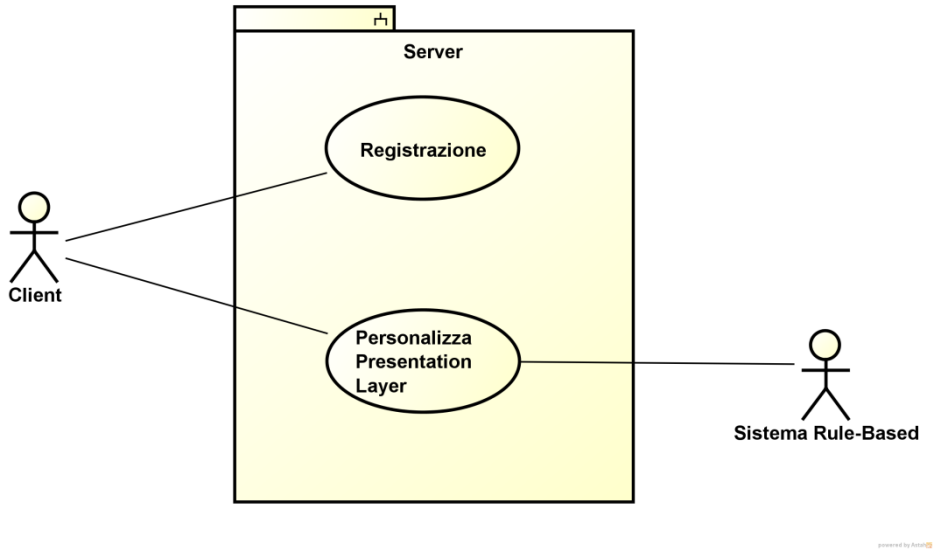

Figura 10 - Diagramma dei casi d'uso n° 2.

Descrizione dei casi d'uso dell'interazione col servizio web.

Le funzionalità che andranno implementate nel dispositivo sono le seguenti.

- Registrazione: il sistema si occuperà della registrazione del Client (dispositivo), allo scopo di stabilire un canale di connessione in grado di garantire la robustezza delle successive comunicazioni.
- Personalizza Presentation Layer: il sistema si occuperà di immagazzinare le informazioni sui sensori ricevuti dal Client, e trasformarli in attributi comprensibili al Sistema Rule-Base. Quest'ultimo dovrà essere in grado di stabilire quale migliore strato di presentazione dei servizi web si adatta allo stato richiesto dal dispositivo mobile.

Attori

Gli attori che andranno ad interagire col sistema in questo caso d'uso si dividono in:

- ✓ Attore principale → **Client**. Il dispositivo che invia informazioni al sistema.
- ✓ Attore secondario→ **Sistema Rule Based**: Sistema in grado di ragionare sui segnali forniti dal Client.

b. Diagrammi di Sequenza

Di seguito sono riportati quattro diagrammi dinamici inerenti ad alcune operazioni fondamentali per il sistema nella sua interezza. Tali diagrammi sono costruiti a partire dai relativi scenari applicativi.

i. Diagramma di sequenza della registrazione dell'utente sul dispositivo

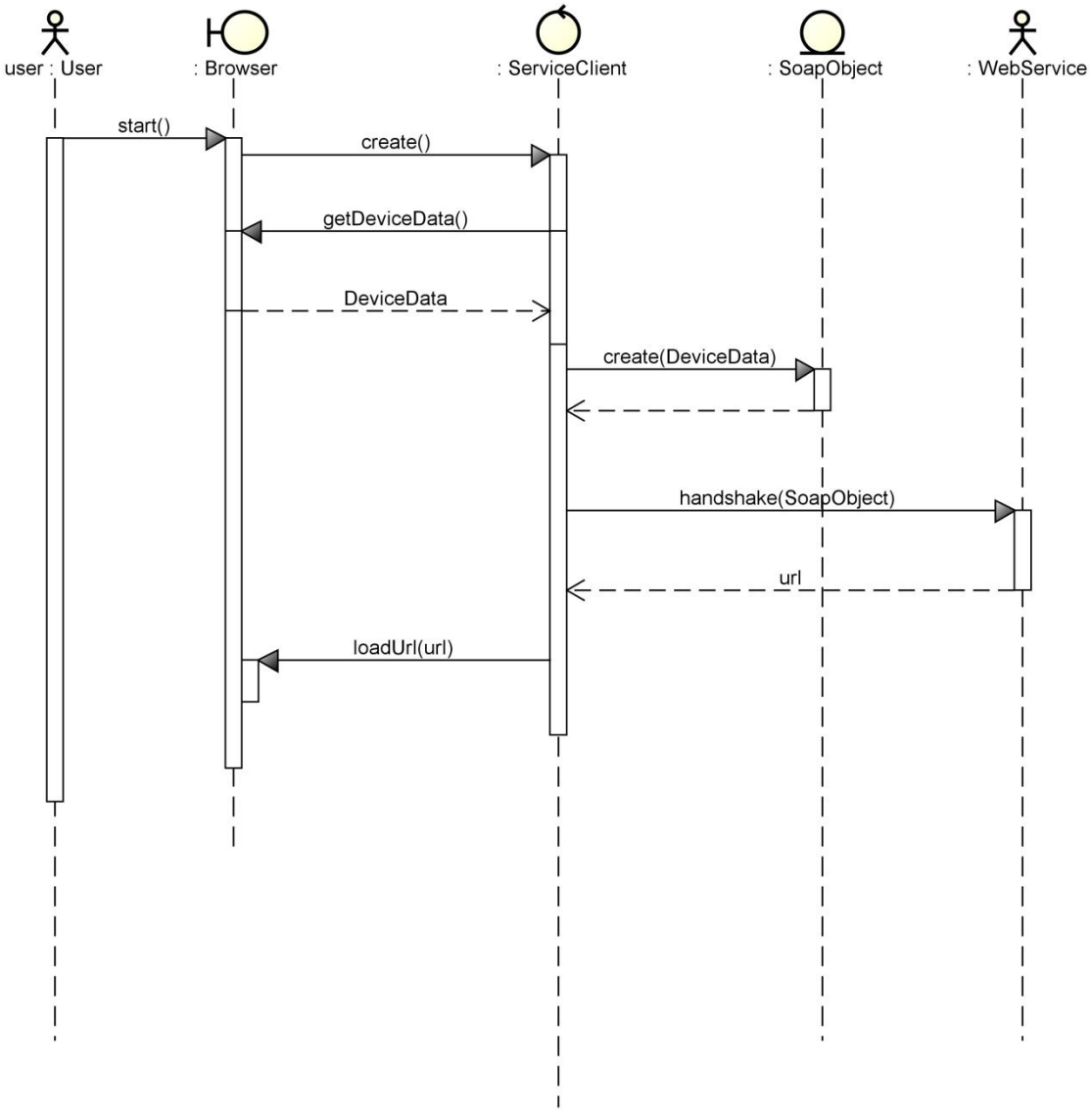

Figura 11 - Diagramma di sequenza n° 1.

Il presente diagramma, riporta la fase di registrazione (handshake) dell'utente sul dispositivo. Questa fase3 interesserà anche il servizio web che verrà contattato tramite oggetto SOAP.

Sono interessate le seguenti classi:

- **Browser: Boundary**, è la classe di interfaccia tra il dispositivo e l'utente. In virtù delle caratteristiche fisiche e tecnologiche degli smartphone di ultima generazione, si suppone che lo stesso browser sia embedded nel dispositivo.
- **ServiceClient:Control**, è la classe di controllo che si occupa di gestire i dati in ingresso ed impacchettare le informazioni da fornire al client tramite protocollo SOAP.
- **SoapObject:Entity**, costituisce l'oggetto che viene trasmesso dal dispositivo (attraverso la classe ServiceClient) all'attore WebService.

ii. Diagramma di sequenza della registrazione del client sul server

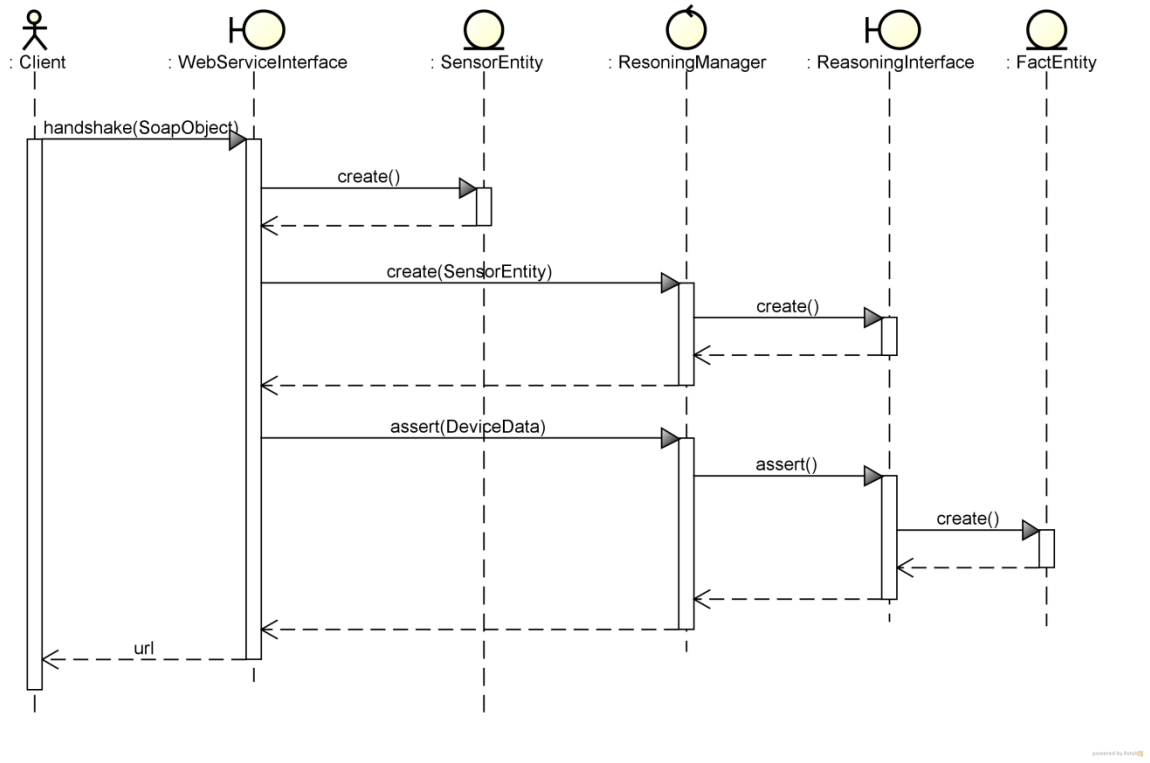

Figura 12 - Diagramma di sequenza n° 2.

Il presente diagramma, riporta la fase di registrazione (handshake) tra il Client ed il server. Durante questa fase il dispositivo viene abilitato alla comunicazione con il server. In tal modo potrà comunicare tramite messaggi SOAP durante le successive interazioni.

Sono interessate le seguenti classi:

- **WebServiceInterface: Boundary**, è la classe di interfaccia tra il client ed il servizio web. Si occupa di ricevere e gestire i messaggi SOAP, nonché di inoltrare alla porta di destinazione i messaggi ricevuti.
- **SensorEntity: Entity**, contiene la struttura dei segnali che arrivano dal dispositivo. In questa fase, saranno memorizzati, come istanza di questa classe, i dati relativi alle caratteristiche del dispositivo client connesso (per esempio, l'ID del dispositivo, la dimensione dello schermo, il browser utilizzato, la posizione geografica, ecc.).

- **ReasoningManager: Control**, è la classe responsabile della gestione del ragionamento sugli attributi inerenti alle caratteristiche del dispositivo. Questa classe si occupa di instanziare l'interfaccia al motore inferenziale.
- **Reasoning Interface: Boundary**, costituisce l'interfaccia del ragionamento. Essa viene instanziata per la prima volta durante la fase di handshake.
- **FactEntity: Entity**, contiene i fatti sul quale ragionerà il motore inferenziale. Tali fatti sono asseriti dal Reasoning Manager e creati dalla classe di controllo ReasoningInterface.

iii. Diagramma di sequenza dell'invio dati sensori sul dispositivo.

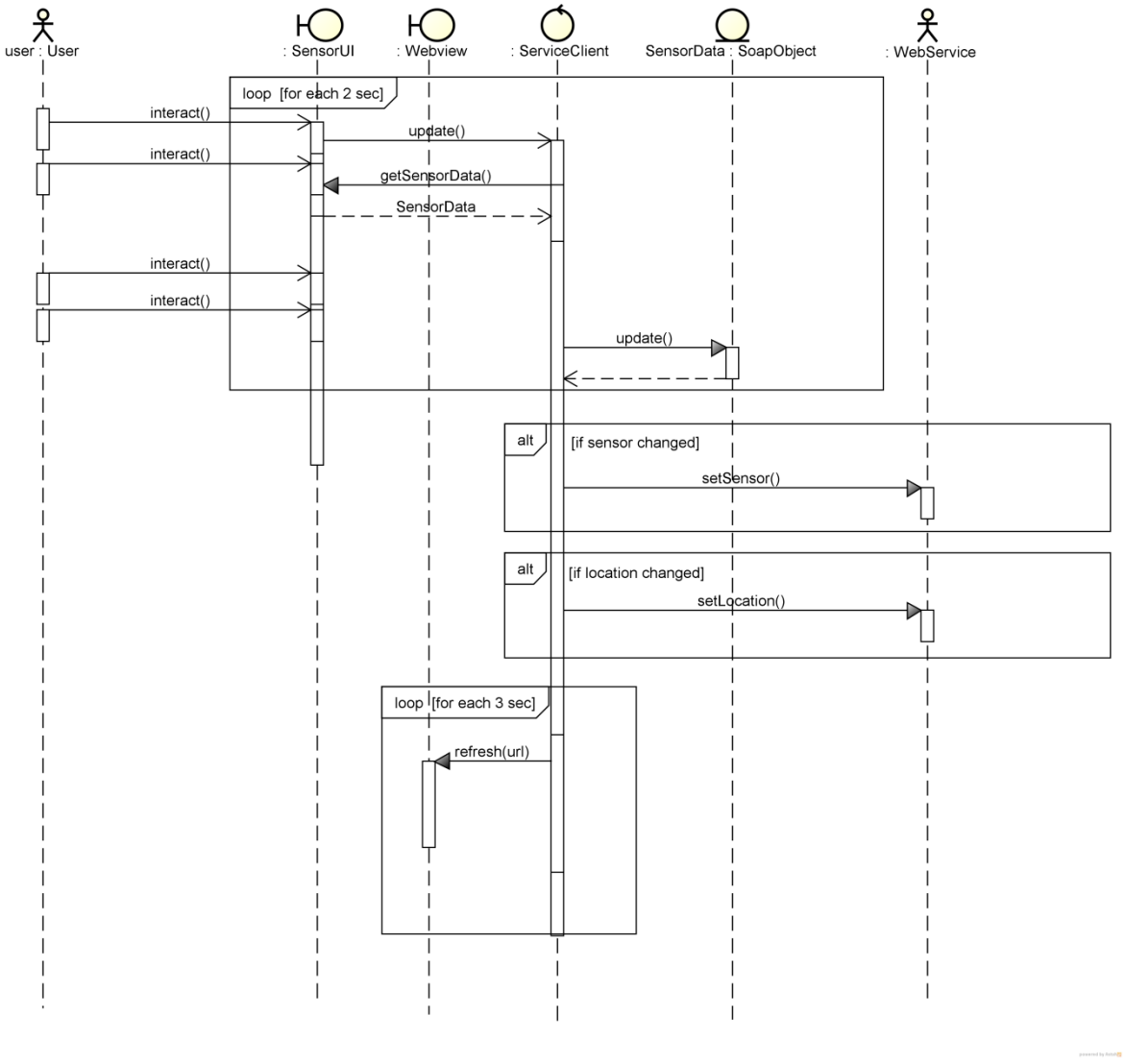

Figura 13 - Diagramma di sequenza n° 3.

Il presente diagramma, riporta la fase di raccolta dei segnali da parte del dispositivo Questi dati vengono dapprima filtrati ed interpretati dal dispositivo stesso e, successivamente, inviati al server come messaggio SOAP. In questo scenario verrà coinvolto anche il servizio web, in qualità di attore, il quale riceverà l'oggetto SOAP. Da notare la presenza nel diagramma dei "Loop" che gestiscono la frequenza di refresh sia della raccolta dati sensoriali, che dell'aggiornamento del link web da visualizzare nel dispositivo.

Sono interessate le seguenti classi:

- **SensorUI: Boundary**, è la classe di interfaccia tra il dispositivo e l'utente. In virtù delle caratteristiche fisiche e tecnologiche degli smartphone di ultima generazione, si suppone che il dispositivo sia in grado di raccogliere diversi segnali dal mondo esterno e dallo stesso Utente. Tali segnali verranno raccolti dal sistema per mezzo di questa classe.
- **WebView:Boundary**, è la classe di interfaccia che si occupa di visualizzare il contenuto della pagina web.
- **ServiceClient:Control**, è la classe di controllo che si occupa di gestire i dati in ingresso, aggiornare lo stato interno dei sensori ed impacchettare le informazioni da fornire al client tramite protocollo SOAP. Inoltre, esso si occuperà di scandire i cicli di aggiornamento sia per quanto riguarda la lettura dei sensori, che il refresh della URL.
- **SoapObject:Entity**, costituisce l'oggetto che immagazzina i dati provenienti dall'utente e dall'ambiente (filtrati attraverso la classe ServiceClient. Esso viene trasmesso dal dispositivo (attraverso la classe ServiceClient) all'attore WebService.

iv. Diagramma di sequenza della ricezione dati dispositivo sul server.

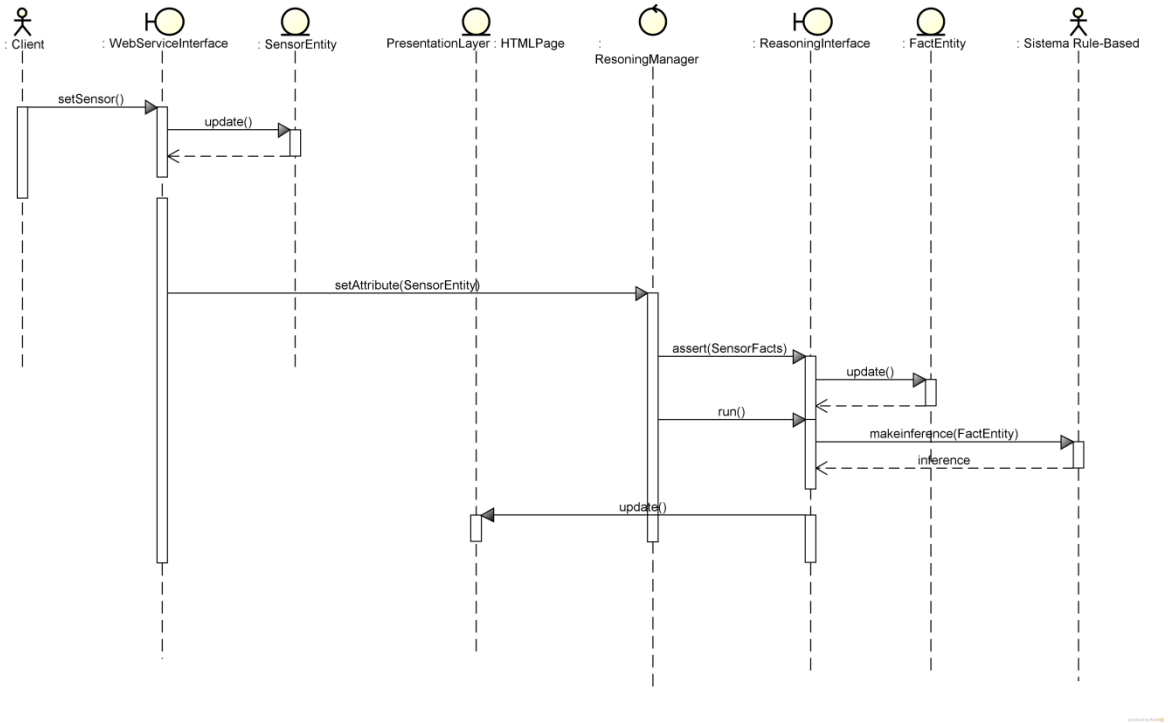

Figura 14 - Diagramma di sequenza n° 4.

Il presente diagramma, riporta la fase di ricezione dei dati del server. Durante questa fase il dispositivo invia tramite richiesta SOAP, i dati dei sensori. Il web service, una volta ricevuti i dati, li trasforma in attributi per il ragionatore, il quale sintetizza i fatti su cui effettuare il ragionamento stesso. A questo punto, il Sistem Rule-based si occuperà di effettuare l'inferenza e produrre una uscita che verrà trasferita al livello di presentazione adattiva, che avrà il compito di trasformarla in pagina web visualizzabile dal dispositivo.

Sono interessate le seguenti classi:

- **WebServiceInterface: Boundary**, è la classe di interfaccia tra il client ed il servizio web. Si occupa di ricevere e gestire i messaggi SOAP, nonché di inoltrare alla porta di destinazione i messaggi ricevuti.
- **SensorEntity: Entity**, contiene la struttura dei segnali che arrivano dal dispositivo. Saranno memorizzati, come istanza di questa classe, i dati relativi alle caratteristiche del dispositivo client connesso (per esempio, l'ID del dispositivo, la dimensione dello schermo, il browser utilizzato, la posizione geografica, ecc.).
- **PresentationLayer: HTMLPage**, è la classe responsabile dell'aggiornamento del presentation layer e della produzione di una URL adattiva.
- **ReasoningManager: Control**, è la classe responsabile della gestione del ragionamento sugli attributi inerenti alle caratteristiche del dispositivo. Questa classe si occuperà di lanciare il motore inferenziale.
- **Reasoning Interface: Boundary**, costituisce l'interfaccia del ragionamento. Essa si occuperà di leggere e tasmetter al ragioantore i fatti contenuti il FactEntity.
- **FactEntity: Entity**, contiene i fatti sul quale ragionerà il motore inferenziale. Tali fatti sono asseriti dal Reasoning Manager e creati dalla classe di controllo ReasoningInterface

c. Diagramma dei Package

Tale diagramma è utilizzato per illustrare l'architettura logica del sistema. In particolare è possibile vedere i due sottosistemi prima descritti, ovvero il "FrasiClient" ed il "FrasiWebService". Tali sistemi saranno in grado di interfacciarsi attraverso il protocollo SOAP. Per ottenere questo risultato, sarà necessario implementare nel client un pacchetto di "protocollo SOAP", rappresentata nel diagramma con una relazione di dipendenza. Quest'ultimo consentirà al client di interfacciarsi al servizio web, inviando messaggi SOAP. Per quanto riguarda il lato server, invece, sarà necessario adoperare un motore per servizi web, che rispetti gli standard WSDL/SOAP illustrati nella prima parte di questo documento. Inoltre sarà necessario un sistema a regole in grado di compiere ragionamento sui dati ricavati dai sensori. Quest'ultimo package dovrà essere in grado di comunicare con il sistema ed interpretare i fatti posti in essere dal "Reasoning Manager".

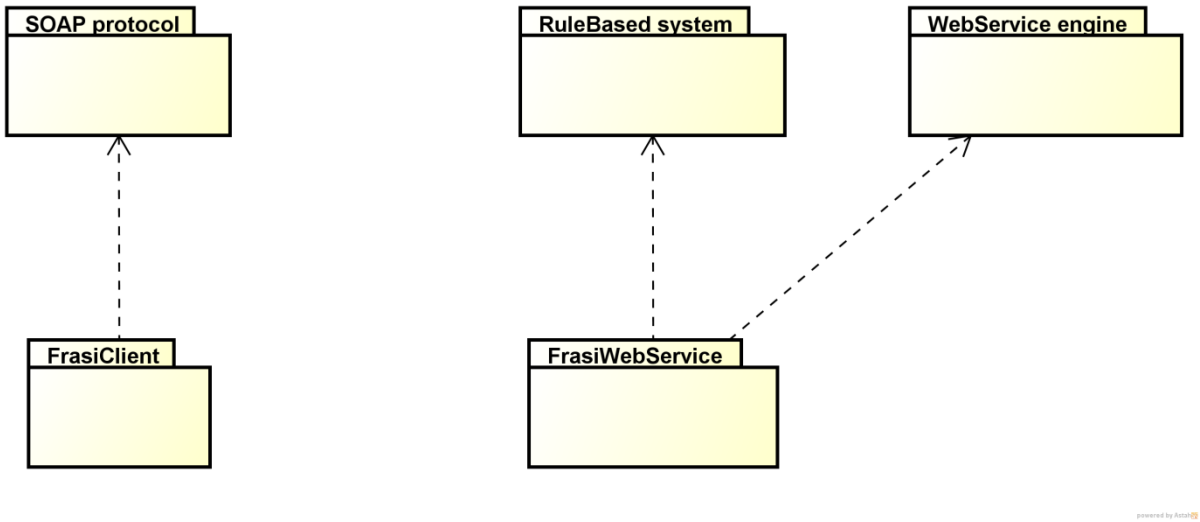

Figura 15 - Diagramma dei Package.

d. Diagramma delle Classi

Nel diagramma riportato di seguito è rappresentata una prima stesura dell'organizzazione delle classi per il presente progetto. Verranno descritti i tipi di entità, con le loro caratteristiche e le relazioni fra questi tipi. Lo strumento concettuale utilizzato per rappresentare tali elementi dichiarativi (statici) è il concetto di classe del paradigma object-oriented utilizzato da Java. Innanzitutto è possibile vedere la suddivisione delle classi in due package principali: quello inerente al dispositivo e quello relativo al server, nominati rispettivamente "FrasiClient" e "FrasiWebServer". Nei package sono riportate le classi ed alcuni metodi ricavati dai diagrammi di sequenza. Sono inoltre riportate le relazioni che legano le precedenti classi.

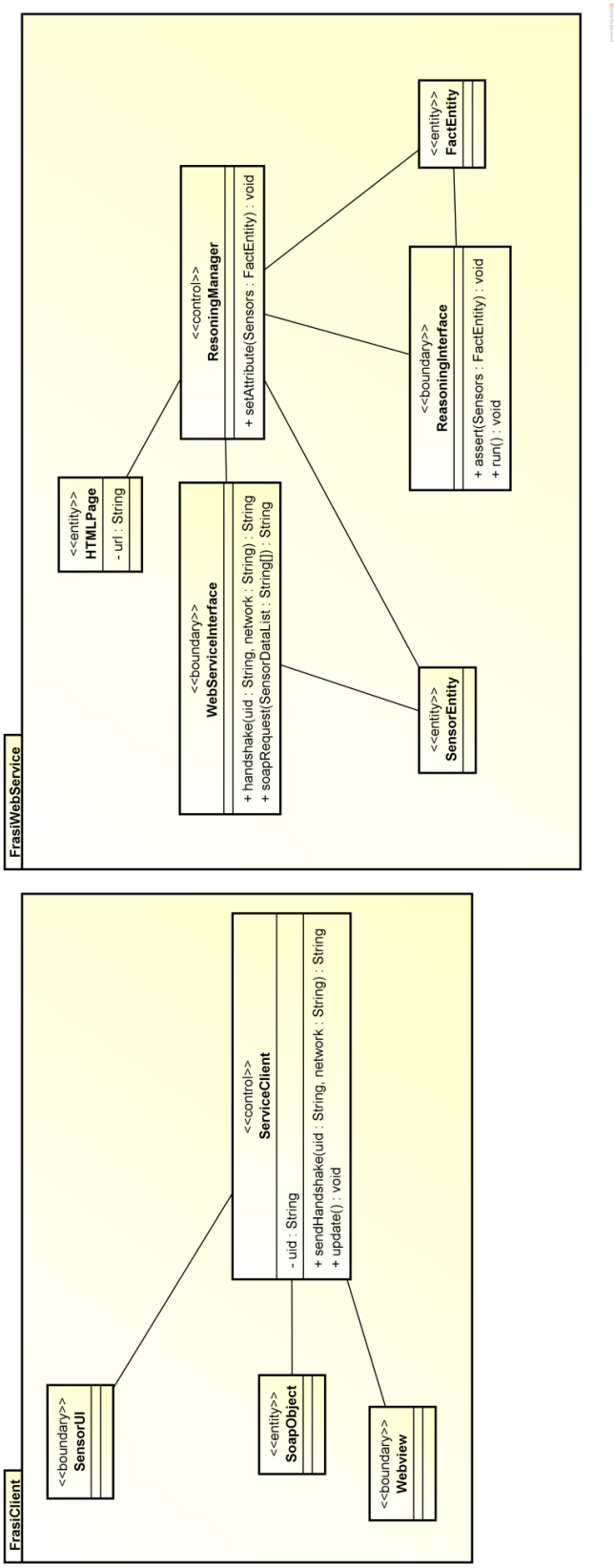

Figura 16 Diagramma delle Classi.

8. Descrizione delle componenti del sistema

In questo capitolo verranno fornite le specifiche delle componenti individuate in fase di descrizione del sistema (documento DES). In particolare verranno presentate le tecnologie, librerie e metodologie scelte per l'implementazione di

- Client del dispositivo (Modulo di acquisizione)
- Sistema Rule-based (Rule base e modulo di adattamento)
- Web Service engine
- Web Server

Client del dispositivo: Android Platform

Il client del sistema proposto è stato implementato su dispositivi mobili utilizzando la piattaforma Android.

Android è un sistema operativo per dispositivi mobili sviluppato da Google, rilasciato nella sua versione 1.0, denominata Apple Pie, il 23 settembre 2008. Android, essendo un sistema operativo di moderna fattura, è abbastanza complesso. Anche se il suo target sono i dispositivi mobili, l'architettura di Android ha poco da invidiare a quelle dei comuni sistemi per desktop o laptop. Tale architettura è presentata schematicamente in **Figura 17**. Come si evince dalla figura, Google ha attinto a piene mani dal mondo Open Source. Il cuore di ogni sistema Android, tanto per cominciare, è un kernel Linux, versione 2.6. Direttamente nel kernel sono inseriti i driver per il controllo dell'hardware del dispositivo: driver per la tastiera, lo schermo, il touchpad, il Wi-Fi, il Bluetooth, il controllo dell'audio e così via. Sopra il kernel poggiano le librerie fondamentali, anche queste tutte mutuate dal mondo Open Source. Da citare sono senz'altro OpenGL, per la grafica, SQLite, per la gestione dei dati, e WebKit, per la visualizzazione delle pagine Web. L'architettura prevede poi una macchina virtuale e una libreria fondamentale che, insieme, costituiscono la piattaforma di sviluppo per le applicazioni Android. Questa macchina virtuale si chiama Dalvik, e sostanzialmente è una Java Virtual Machine. Alcune delle caratteristiche di Dalvik e della sua libreria non permettono di identificare immediatamente la piattaforma Java disponibile in Android con una di quelle di riferimento (Java SE, Java ME). Nel penultimo strato dell'architettura è possibile rintracciare i gestori e le applicazioni di base del sistema. Ci sono gestori per le risorse, per le applicazioni installate, per le telefonate, il file system e altro ancora: tutti componenti di cui difficilmente si può fare a meno. Infine, sullo strato più alto dell'architettura, poggiano gli applicativi destinati all'utente finale. Molti, naturalmente, sono già inclusi con l'installazione di base: il browser ed il player multimediale sono dei facili esempi. A questo livello si inseriranno anche le applicazioni sviluppate da terze parti.

Android fornisce quattro componenti di base:

- Attività (Activity): Le attività sono quei blocchi di un'applicazione che interagiscono con l'utente utilizzando lo schermo e i dispositivi di input messi a disposizione dallo smartphone. Comunemente fanno uso di componenti UI già pronti, come quelli presenti nel pacchetto android.widget, ma questa non è necessariamente la regola. Le attività sono probabilmente il modello più diffuso in Android, e si realizzano estendendo la classe base android.app.Activity.
- Servizio (Service): Un servizio gira in background e non interagisce direttamente con l'utente. Ad esempio può riprodurre un brano MP3, mentre l'utente utilizza delle attività per fare altro. Un servizio si realizza estendendo la classe android.app.Service.
- Broadcast Receiver: Un Broadcast Receiver viene utilizzato quando si intende intercettare un particolare evento, attraverso tutto il sistema. Ad esempio lo si può utilizzare se si desidera compiere un'azione quando si scatta una foto o quando parte la segnalazione di batteria scarica. La classe da estendere è android.content.BroadcastReceiver.
- Content Provider: I Content Provider sono utilizzati per esporre dati e informazioni. Costituiscono un canale di comunicazione tra le differenti applicazioni installate nel sistema. Si può creare un Content Provider estendendo la classe astratta android.content.ContentProvider.

Un'applicazione Android è costituita da uno o più di questi elementi. Molto frequentemente, contiene almeno un'attività, ma non è detto che debba sempre essere così.

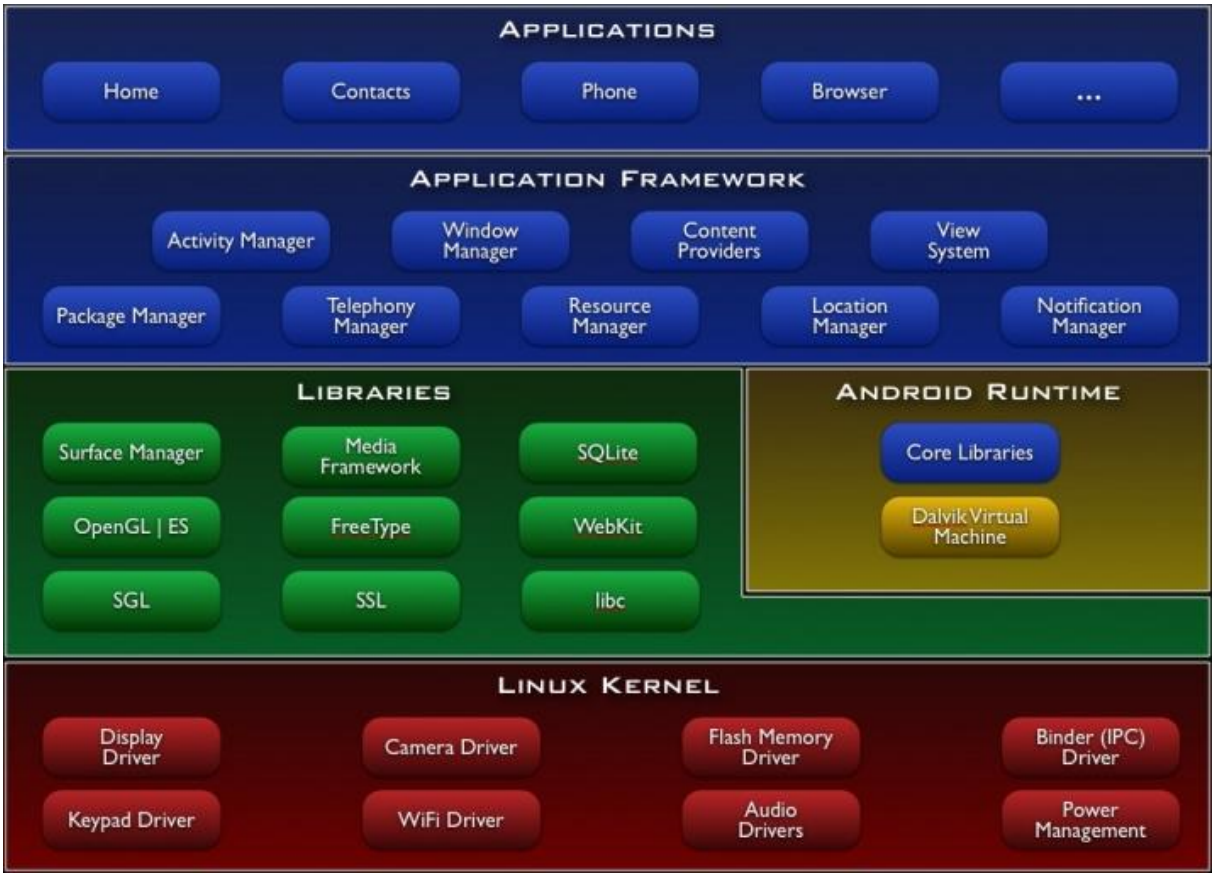

Figura 17: Architettura della piattaforma Android

Le applicazioni Android sono distribuite sotto forma di file APK (Android Package). Al loro interno vengono raccolti gli eseguibili in formato DEX, le eventuali risorse associate e una serie di descrittori che delineano il contenuto del pacchetto. In particolare, nel cosiddetto manifesto, vengono dichiarate le attività, i servizi, i provider e i receiver compresi nel pacchetto, in modo che il sistema possa agganciarli e azionarli correttamente.

Tipi di segnali rilevati dal dispositivo

Come evidenziato nel documento di descrizione del software (DES), il dispositivo è in grado di catturare dei segnali dall'ambiente circostante, grazie ad un set di sensori, ed inviarli al web service. Utilizzando la piattaforma Android, per lo sviluppo del prototipo sono stati presi in considerazione i seguenti tipi di sensori e di segnali (come da documentazione ufficiale Android):

- **Sensor.TYPE_LIGHT:** livello di luminosità ambientale misurata in *lux*
- **Sensor.TYPE_PROXIMITY:** prossimità di un oggetto rispetto al dispositivo misurata in centimetri
- **Sensor.TYPE_ACCELEROMETER:** vettore tridimensionale che indica l'accelerazione lungo ciascun asse del dispositivo.
- **Location:** posizione del dispositivo in termini di latitudine e longitudine. Le informazioni sulla posizione possono essere ottenute da provider differenti (GPS, punto di accesso WIFI, cella telefonica)

Sistema Rule-based: Jess the rule engine for the Java platform

Il sistema rule-based è stato implementato utilizzando Jess, il motore a regole per la piattaforma Java. Jess è scritto interamente in Java e costituisce esso stesso un linguaggio per la descrizione dei fatti e delle regole del sistema esperto. Il motore inferenziale di Jess implementa l'algoritmo Rete come

pattern matcher: l'algoritmo verrà descritto nel prossimo paragrafo. L'agenda del sistema Jess lavora per mezzo di due differenti strategie di risoluzione dei conflitti: in profondità e in ampiezza. Con la strategia in profondità le regole attivate più recentemente sono eseguite per prime; con la strategia in ampiezza, le regole sono eseguite secondo l'ordine di attivazione: in questo modo le regole attivate più recentemente sono eseguite per ultime. In entrambe le strategie di risoluzione, l'ordine di esecuzione delle regole può essere modificato dall'utente cambiando la priorità delle regole.

La working memory di Jess può essere organizzata in moduli: ogni modulo possiede il proprio insieme di fatti e regole. Solamente un modulo alla volta può essere attivo, o in altre parole avere il "focus", e soltanto le regole appartenenti al modulo attivo possono essere eseguite. Di default il module MAIN ha il focus; gli altri moduli, definiti dallo sviluppatore del sistema a regole, ricevono il focus per mezzo di regole speciali, la cui esecuzione sposta il focus da un modulo all'altro. L'intero meccanismo è gestito tramite una pila (stack), con il modulo attivo in cima e gli altri moduli di sotto, a seconda dell'ordine di spostamento del focus. In questo modo, non appena un modulo completa le proprie operazioni, il focus è automaticamente restituito all'ultimo modulo attivo.

Algoritmo Rete

Il compito principale del pattern matcher di un motore inferenziale è quello di controllare la Knowledge Base (KB) per trovare quali regole sono soddisfatte, di modo tale da poter essere attivate e successivamente eseguite secondo la tabella di schedulino dell'agenda. Un approccio *brute force*, che consiste nell'analisi di tutte le parti sinistre (premesse) delle regole con la KB sarebbe inefficiente e molto difficile da scalare nel caso di working memory molto grandi. L'algoritmo Rete rappresenta un modo efficiente per gestire il problema del pattern matching. Durante gli anni è stato migliorato e raffinato in molti sistemi a regole, come per esempio OPS5, ART e CLIPS, e Jess implementa la versione di Rete più recente e performante. L'algoritmo Rete migliora l'approccio semplice del pattern matching considerando soltanto i fatti nuovi o eliminati della working memory per verificare quali regole sono soddisfatte. Esso inoltre conserva i risultati delle operazioni di matching durante ogni ciclo di iterazione del processo di ragionamento. Rete organizza il pattern matcher per mezzo di un network, appunto "rete", di nodi interconnessi cosicché i pochi fatti interessati nel meccanismo di inferenza sono testati con un sottoinsieme di regole con cui potrebbero eventualmente matchare.

Le performance dell'algoritmo Rete, rispetto al semplice algoritmo di pattern matching, quello con approccio brute force, dipende dal numero di cicli di ragionamento. Durante il primo ciclo, infatti, dal momento che l'algoritmo Rete deve analizzare tutti i fatti della working memory perché non ci sono risultati precedenti da sfruttare, le performance tra i due algoritmi sono sostanzialmente le stesse. Rete, d'altro canto supera di gran lunga in prestazioni l'algoritmo base per tutti i cicli di ragionamento successivi al primo.

Web Service engine: Axis2

La prima generazione di Web Services, risalente ai primi anni del 2000, era in grado solamente di eseguire un piccolo insieme di interazioni controllate attraverso il web. A quel tempo, l'obiettivo dei motori SOAP, come per esempio Apache SOAP, era quello di dimostrare che i web services erano qualcosa in più di un concetto teorico. Pertanto i middleware di web services sviluppati in quel periodo erano concentrati maggiormente sulla precisione piuttosto che sulle performance.

La seconda generazione di web services è stata considerata una tecnologia plausibile. Sono stati realizzati molti motori SOAP come Apache Axis, Systinet WASP e gSOAP che erano in grado di supportare diversi stili di interazione. Inoltre sono state messe a punto delle specifiche riguardanti gli aspetti più importanti dei web services e la maggior parte dei motori SOAP hanno aderito a quelle specifiche comuni, permettendo in questo modo le interazioni tra piattaforme differenti (cross platform). Axis è stato uno di questi motori SOAP di seconda generazione e presto divenne molto popolare tra i fornitori di web services. Ma i web services erano ancora carenti dal punto di vista della robustezza e dell'efficienza necessarie per essere una vera piattaforma middleware. Il problema è stato risolto dalla terza generazione di web services, che risultano essere veloci, robusti, efficienti ed affidabili.

Axis2 rappresenta uno dei web services di terza generazione, e deriva dall'esperienza maturata dalla famiglia dei web services Axis 1.x delle passate generazioni. Gli sviluppatori di Axis2 si sono

concentrati soprattutto sulla velocità di risposta e le performance di memoria, con caratteristiche e funzionalità addizionali delle ultime versioni. In particolare, il progetto Axis2 ha introdotto cambiamenti nelle aree seguenti:

- Nuovo modello si SOAP Object
- Miglior supporto per lo scambio di pattern di messaggi
- Comportamento sincrono e asincrono
- Modello di deployment migliorato

Il nuovo modello di SOAP Object

La sfida principale di un middleware basato sui web services è quella di evitare di mantenere in memoria l'intero messaggio SOAP. Mantenere l'intero modello dell'oggetto in memoria rende più facile la manipolazione degli oggetti XML. Di contro la memoria è penalizzata in quanto il modello dell'oggetto richiede una considerevole quantità di memoria che è uguale almeno alla dimensione del documento XML che deve essere elaborato.

Axis2 evita di mantenere l'intero messaggio SOAP in memoria introducendo un nuovo modello di Oggetto per rappresentare i messaggi SOAP, denominato AXIOM. AXIOM, sebbene abbia una "somiglianza" apparente con DOM, differisce da quest'ultimo nel fatto che esso genera gli oggetti solamente quando richiesto. Questa costruzione "on-demand" fornisce ad AXIOM quel margine necessario per superare quel vincolo di memoria che caratterizzava i primi motori SOAP. AXIOM è basato sul concetto di "pull parsing".

Nell'elaborazione di documenti XML, i termini "pull parsing" e "push parsing" sono riferiti all'entità che ha il controllo. Un push parser acquisisce il controllo non appena inizia il parsing del documento e continua a inviare (push) eventi ed oggetti a uno o più gestore di contenuti. Dal momento che il parser ha il controllo durante l'attività di parsing, nessuna entità esterna può interrompere o mettere in pausa il processo di parsing finché il documento non è completamente analizzato. Il SAX parser è un esempio di push parser. Dal momento che il parsing "fuori controllo" di documenti XML di grandi dimensioni hanno reso il parsing stesso difficile ed inefficiente, si è andato sviluppando il concetto di pull parser, che separa il controllo dall'attività di parsing vera e propria.

Il cuore di AXIOM è quindi un Pull parser XML che consente la messa in pausa del processo di parsing. AXIOM utilizza la Streaming API for XML (StAX), rendendo più semplice manipolare ed utilizzare soltanto una frazione di memoria usata dai tradizionali modelli di oggetti. Sfruttando la velocità dello streaming pull parser, AXIOM migliora notevolmente Axis2 in termini di efficienza e velocità.

Modello di deployment migliorato

Nella famiglia Axis 1.x, il deployment di un web service richiede l'aggiornamento di un file di configurazione ed inoltre i file delle classi devono essere disponibili nel class path. La prima operazione può essere fatta per mezzo di un tool chiamato AdminClient, la seconda, invece, risulta più difficoltosa. Il modo migliore di aggiornare il class path è quello di copiare manualmente i file delle classi nella locazione corretta e dopodiché riavviare il server Axis. Ciò rappresenta un vero e proprio collo di bottiglia per quanto riguarda l'usabilità di Axis.

Il modello di deployment di Axis2 è basato su un file system ad archivio. Ogni componente della configurazione di Axis2 è immagazzinata in un archivio o in un file di configurazione, il che lo rende simile al meccanismo di deployment di J2EE. Axis2 introduce pure il cosiddetto "hot deployment": esso individua la presenza dell'archivio del web service nella directory di repository e lo carica automaticamente. Sebbene questa caratteristica non sia molto utile in un ambiente di produzione, è invece di grande utilità per i principianti. Il reale valore di questo modello di deployment sta nel fatto che l'informazione riguardante ogni web service è contenuta in un singolo archivio. Gli archivi dei web services sono portabili su tutti i container Axis2 e il caricamento dei file delle classi del web service risulta semplificato. Gli archivi Axis possono essere messi manualmente nella directory di repository o caricati attraverso l'Axis Web application per un container di servlet J2EE. Per rendere le cose più semplici, Axis2 fornisce anche diversi tool grafici per aiutare il programmatore nella creazione degli archivi dei web service.

Supporto migliorato per lo scambio di pattern di messaggi

Quando Axis 1.x è stato introdotto, lo scopo principale di un web service era quello di emulare RPC (Remote Procedure Call) attraverso SOAP. Per questo motivo, Axis 1.x possiede un modello di interazione RPC-style, che richiede che il sistema risponda ad ogni richiesta. Questo comportamento causa problemi per le interazioni non RPC-style, come per esempio l'invio di messaggi "one-way".

Le interazioni message-style sono definite dai Message Exchange Patterns (MEP) le specifiche di WSDL 2.0 definisco otto MEP. Di questi, i tre modelli essenziali per lo sviluppo di una soluzione di base che utilizza lo scambio di messaggi sono supportati direttamente da Axis2. Essi sono:

1. In-Only
2. Robust-In
3. In-Out

Il modello di interazione In-Only ha solamente una SOAP request ma nessuna SOAP response. Robust-In ha una SOAP request ed una SOAP response è inviata solo in caso di errore. In-Out ha sempre una SOAP request e una SOAP response.

Axis 2 supporta sia le interazioni RPC-style sia quelle message-style basate sul modello di messaggistica one-way. Inoltre le interazioni RPC-style possono essere implementate al di sopra di un modello di interazione message-style. In questo modo Axis2 può coesistere con altri SOAP engine basati su RPC, come per esempio Axis 1.x.

Aspetti sincroni e asincroni

L'elaborazione su sistemi distribuiti si è focalizzata per molto tempo sul paradigma RPC. Come risultato, la nozione di una risposta (response) facente seguito immediatamente ad una richiesta (request) è impressa nelle menti degli sviluppatori di web service. In degli scenari reali, una response può non verificarsi del tutto o avvenire dopo un certo tempo, che può essere un minuto o un anno. A seconda del tempo impiegato a rispondere, la response può essere inviata attraverso la connessione entrante o una connessione differente.

Nel paradigma RPC, l'invocazione di un metodo blocca il sistema nell'attesa dell'arrivo di una risposta. Come detto precedentemente gli scenari reali sono molto più complessi ed il modello di programmazione client-side ha bisogno di essere esteso. I modelli di programmazione sincrona e asincrona sono entrambi supportati da Axis2., sebbene implementare un SOAP engine che supporti questi aspetti della messaggistica non sia semplice.

Web Server: Apache Tomcat

Tomcat è un Servlet Engine (chiamato anche Servlet Container o Servlet/JSP); è cioè un software capace di gestire "Web applications" scritte secondo le specifiche da utilizzare per il linguaggio Java della Sun Microsystem. A ben vedere Tomcat, un programma open source della Apache Software Foundation nato in seno al cosiddetto "Progetto Jakarta", possiede molti tratti in comune con il suo "fratello maggiore": Apache. In effetti questo Servlet Engine è in tutto è per tutto un Web server in quanto interpreta le richieste provenienti dai browser dei client e provvede a generare dinamicamente i relativi output. Rispetto ad Apache, Tomcat ha però delle caratteristiche particolari che descriveremo in questo capitolo. Digitando le URL, visualizziamo delle determinate "zone" controllate da un Web server, ogni "zona" in Tomcat prende il nome di Contesto e ogni Contesto fa capo oggetti definiti validi sia per il loro ambito di azione che per gli utenti che ad esso accedono. I Contesti non vengono generati sulla base di chiamate, ma "esistono" dal momento in cui parte il processo di Tomcat e come quest'ultimo "attendono" di rispondere alle richieste degli utenti senza mai arrestarsi. Ad ogni utente viene affidato dal Web server un Contenitore "personale" detto Sessione caratterizzato dalle risorse a cui quel determinato utente ha la possibilità di accedere. All'interno di questa sessione e per tutta la durata del suo collegamento alle risorse gestite da Tomcat, l'utente potrà effettuare delle richieste sotto forma di input da soddisfare tramite risposte generate dinamicamente in output. Le richieste a loro modo sono dei Contenitori particolari, in quanto esistono solo nel lasso di tempo che separa l'invio della domanda e la fornitura della relativa risposta. Chi conosce bene i meccanismi che regolano i Web server non avrà di certo notato consistenti novità nel criterio di gestione input/output appena descritto, per quanto riguarda i metodi possiamo invece osservare sostanziali particolarità: all'atto della richiesta,

classicamente la digitazione di una URL tramite protocollo di comunicazione HTTP, viene lanciata l'istanza di un oggetto chiamato Request o Req, questo input porta il Servlet Engine ad istanziare un oggetto chiamato Response o Res come reazione "naturale" alla richiesta del client. Req e Res dovranno quindi essere dirottati verso il Contesto contenente le risorse utili a concludere la transizione in atto. Oltre a Req e Res, Tomcat provvederà ad istanziare un ulteriore oggetto, Ses, riferito al Contenitore di Sessione, con il compito di identificare univocamente il richiedente e di controllare che la sua Sessione sia "aperta". Le Servlet dovranno rendere "consistente" Res. In pratica il Contesto a cui appartengono le risorse necessarie per la soddisfazione della richiesta dovrà indirizzare queste ultime alla "Web application" Java necessaria al completamento della transizione. Nel momento in cui verrà generato l'output avremo tre esiti possibili:

1. La Servlet esaudisce la richiesta e lascia a Tomcat il compito di consegnare l'output al client (per esempio, sotto forma di ipertesto HTML).
2. La risorsa necessaria alla soddisfazione della richiesta non è presente nel Contesto utilizzato, dovrà quindi essere ricercata in un ulteriore Contesto esterno, questo compito verrà quindi affidato a Tomcat che riavvierà la procedura di input/output.
3. La richiesta può essere soddisfatta all'interno del Contesto ma non tramite la risorsa richiesta, dovrà quindi essere generato un Forward per il reindirizzamento verso la risorsa adeguata.

Apache Tomcat è costituito da tre componenti principali:

- Catalina: è il contenitore di servlet Java di Tomcat. Catalina implementa le specifiche di Sun Microsystems per le servlets Java e le "JavaServer Pages (JSP, Pagine JavaServer).
- Coyote: è il componente "connettore HTTP" di Tomcat. Supporta il protocollo HTTP 1.1 per il web server o per il contenitore di applicazioni. Coyote ascolta le connessioni in entrata su una specifica porta TCP sul server e inoltra la richiesta al Tomcat Engine per processare la richiesta e mandare indietro una risposta al client richiedente.
- Jasper: è il motore JSP di Tomcat, ovvero è un'implementazione delle specifiche 2.0 delle Pagine JavaServer (JSP) di Sun Microsystems. Jasper parsa i file JSP per compilarli in codice Java come servlets (che verranno poi gestite da Catalina). Al momento di essere lanciato, Jasper trova i cambiamenti avvenuti ai file JSP e, se necessario, li ricompila.

Modello degli oggetti

Nella fase di progettazione delle specifiche del sistema sono state individuate, rispetto all'analisi del problema effettuata nel documento di descrizione (DES), nuove classi lato server, ed inoltre le classi individuate durante l'analisi dei requisiti sono state eventualmente arricchite con nuovi attributi e/o metodi. Sono presenti anche gli oggetti appartenenti alle librerie esterne utilizzate (cf 3.4).

Oggetti Entity

Nome:	Package:	Descrizione
SensorEntity	frasi.entity	Rappresenta i segnali ricevuti dal client del dispositivo

Oggetti Boundary

Nome:	Package:	Descrizione
SensorUI	frasi.client	Rappresenta l'interfaccia tra utente e sensoristica del dispositivo
WebServiceInterface	frasi.service	Interfaccia tra client e web service

Oggetti Control

Nome:	Package:	Descrizione
ServiceClient	frasi.client	Il client del dispositivo che acquisisce i segnali dei sensori e li invia al web service
ReasoningManager	frasi.engine	Gestisce l'attività di ragionamento acquisendo i segnali ricevuti dal web service e inviandoli al sistema esperto a regole per mezzo dell'oggetto Rete
MakeHTML	frasi.engine	Gestisce la creazione del file html adattato al contesto

Librerie esterne (jar)

Per lo sviluppo sia del client che del web service, sono state adottate classi appartenenti a librerie esterne (jar).

Librerie esterne utilizzate dal Client:

- Android.jar: librerie di riferimento della piattaforma Android
- Ksoap2-android-assembly-2.6.5.jar: implementazione per piattaforma Android del protocollo SOAP

Librerie esterne utilizzate dal Web Service:

- Axis2-1.6.2 (insieme di librerie): implementazione del web service container
- Jess.jar: libreria per l'interfacciamento con il sistema esperto a regole

Nel dettagli le classi utilizzate appartenenti alle librerie indicate

Nome:	Libreria:	Descrizione
WebView	Android.jar	Consente la visualizzazione di una pagina web al di fuori di un browser
SoapObject	Ksoap2-android-assembly-2.6.5.jar	Rappresenta il messaggio SOAP che viene inviato al web service
Fact	Jess.jar	Rappresenta un fatto della base di conoscenza, secondo la nomenclatura di Jess
Rete	Jess.jar	Interfaccia tra il web service e il sistema esperto a regole
MessageContext	Axis2_1.6.2	Gestisce il contesto del messaggio ricevuto dal web service
ServiceContext	Axis2_1.6.2	Gestisce il contesto globale del servizio offerto dal web service

Diagramma delle Classi

Di seguito è mostrato il diagramma delle principali classi sviluppate, mettendo in evidenza le relazioni tra le classi stesse e i package di appartenenza:

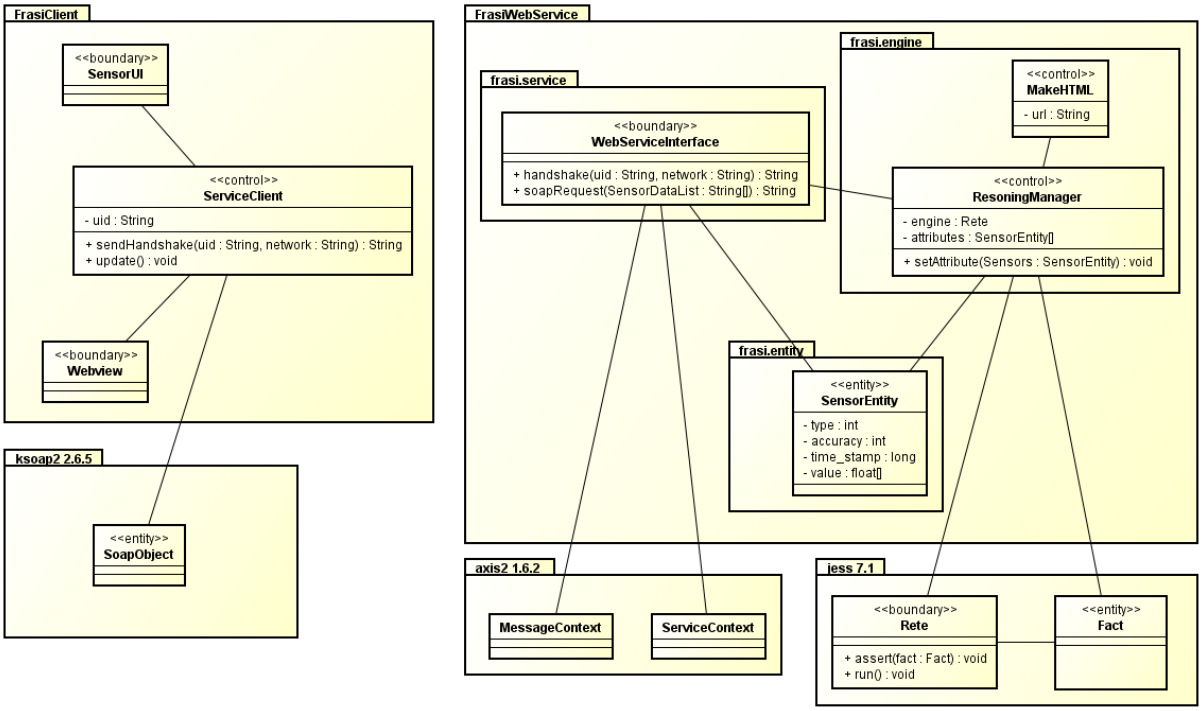

Figura 18: Diagramma delle classi

Diagramma dei Package

Le relazioni tra i package implementati e i package facenti riferimento alle librerie esterne utilizzate

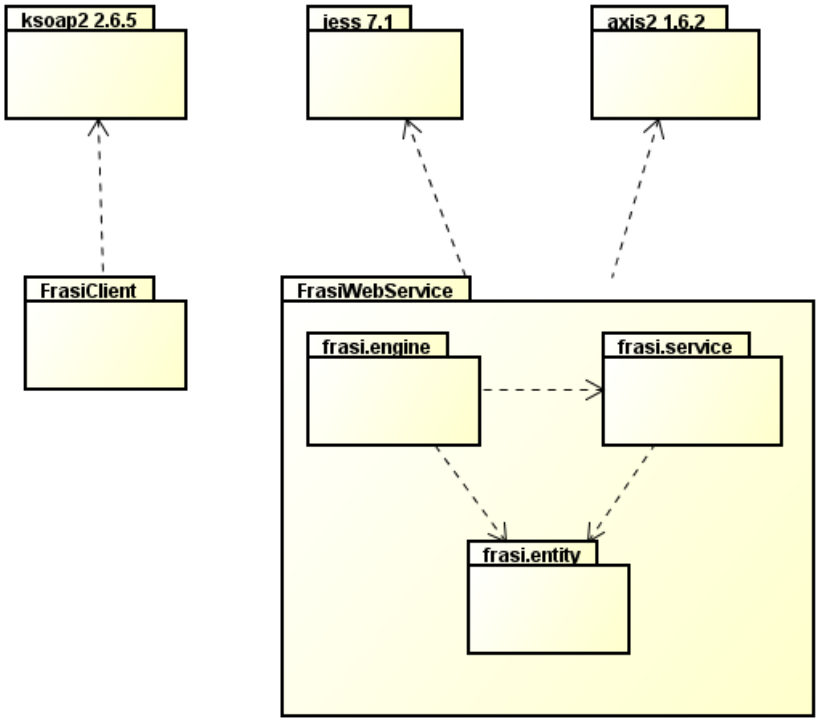

Figura 19: Diagramma dei Package

Diagramma di Deployment

Nel diagramma di Deployment sono mostrate dove vengono allocate (deployed) le risorse sviluppate, mettendo in evidenza i dispositivi fisici utilizzati

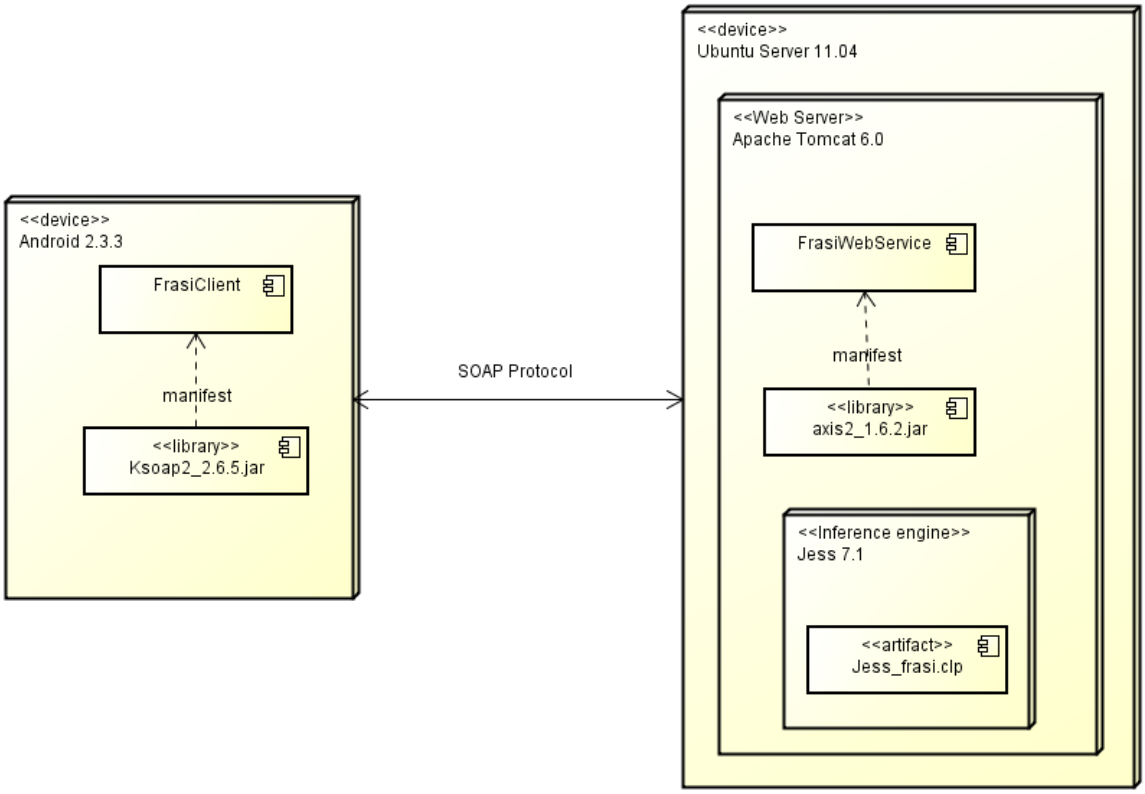

Figura 20: Diagramma di Deployment

Tool di sviluppo

Per lo sviluppo del prototipo, sono stati utilizzati i seguenti tool:

- Astah Community: strumento CASE per la progettazione del modello degli oggetti, con cui sono state definite le classi e i relativi diagrammi delle classi, dei package e di deployment
- Eclipse IDE Indigo version: Ambiente di sviluppo JAVA
- Android SDK: piattaforma di sviluppo Android

9. Appendice

a. Codice Sorgente Java

i. WebServiceInterface.java

```
package frasi.service;

import org.apache.axis2.context.MessageContext;
import org.apache.axis2.context.ServiceContext;
import frasi.engine.ReasoningManager;
import frasi.entity.SensorEntity;

/**
 *
 * @author ICAR-CNR
 *
 */

public class WebServiceInterface {


    /**
     * effettua la registrazione del dispositivo
     * @param uid identificativo univoco del dispositivo
     * @param width larghezza (in pixel) dello schermo
     * @param height altezza (in pixel) dello schermo
     * @return url pagina web personalizzata
     */
    public String handshake(String uid, int width, int height, String network, String speed) {
```

```
		//buffer dei sensori
		SensorEntity[] allAttributes = new SensorEntity[4]; //0=luce, 1=accelerazione, 2=prossimita', 3=location
		for (int i=0; i < allAttributes.length; i++){
			allAttributes[i]= new SensorEntity();
		}

		ReasoningManager rete = new ReasoningManager(allAttributes);
		rete.assertScreenSize(width, height);
		rete.assertConnection(network, Double.parseDouble(speed));
		ServiceContext context = MessageContext.getCurrentMessageContext().getServiceContext();
		context.setProperty(uid, rete);
		rete.start();
		return "http://150.145.115.245:8080/FrasiWebService/result.html"; //indirizzo ip della pagina adattata
	}



	/**
	 * setta i valori dei sensori ricevuti dal dispositivo
	 * @param uid identificativo univoco del dispositivo
	 * @param type tipo del sensore
	 * @param accuracy precisione del sensore
	 * @param ts istante di rivelazione del sensore (millisecondi)
	 * @param values valori rivelati dal sensore (stringa separata da ";")
	 */
		public void setSensor(String uid, int type, int accuracy, long ts, String values) {

			// parser
			String[] result = values.split(";");
			float[] floatvalues = new float[result.length];
			for (int i=0; i<result.length; i++) {
				floatvalues[i] = Float.parseFloat(result[i]);
			}

			SensorEntity sensor = new SensorEntity(type, accuracy, ts, floatvalues);
			sensor.setJesstype();

			String sensorType= sensor.getJesstype();
```

```
		int index=0;
		if (sensorType.equals("light"))
			index=0;
		else if (sensorType.equals("accelerometer"))
			index=1;
		else if (sensorType.equals("proximity"))
			index=2;
		else if (sensorType.equals("location"))
			index=3;
		ServiceContext context = MessageContext.getCurrentMessageContext().getServiceContext();

		ReasoningManager rete = (ReasoningManager) context.getProperty(uid);
		SensorEntity[] attributes = rete.getAttributes();
		attributes[index] = sensor;
		rete.setAttributes(attributes);

	}

	/**
	 * setta i valori della localizzazione ricevuti dal dispositivo
	 * @param uid identificativo univoco del dispositivo
	 * @param provider fornitore della location (GPS, WIFI, Cella)
	 * @param accuracy precisione della loaction
	 * @param ts istante di rivelazione della location (millisecondi)
	 * @param lat latitudine
	 * @param lon longitudine
	 */
	public void setLocation(String uid, String provider, String accuracy, long ts, double lat, double lon) {

		ServiceContext context = MessageContext.getCurrentMessageContext().getServiceContext();

		ReasoningManager rete = (ReasoningManager) context.getProperty(uid);
		rete.assertLocation(provider, accuracy, ts, lat, lon);
	}

}
```

ii. SensorEntity.java

```
package frasi.entity;

import frasi.entity.SensorType;

/**
 *
 * @author ICAR-CNR
 *
 */
public class SensorEntity {

	private int type;
	private int accuracy;
	private long time_stamp;
	private float[] value;
	private String jesstype; //il tipo di template jess

	public SensorEntity(){

	}

	/**
	 *
	 * @param t tipo di sensore
	 * @param accuracy precisione del sensore
	 * @param ts time stamp (millisecondi) di rilevazione del sensore
	 * @param v valori rilevati dal sensore
	 */
	public SensorEntity(int t, int accuracy, long ts, float[] v){
		this.setType(t);
		this.setAccuracy(accuracy);
		this.setTime_stamp(ts);
		this.setValue(v);
	}


	/**
	 *
	 * @param t tipo di sensore
```

```
 * @param ts time stamp (millisecondi) di rilevazione del sensore
 * @param v valori rilevati dal sensore
 */
public SensorEntity(int t, long ts, float[] v){
		this.setType(t);
		this.setTime_stamp(ts);
		this.setValue(v);
}

/**
 *
 * @param t tipo di sensore
 * @param v valori rilevati dal sensore
 */
public SensorEntity(int t, float[] v){
		this.setType(t);
		this.setTime_stamp(0);
		this.setValue(v);
}

/**
 *
 * @param t tipo di sensore
 */
public void setType(int t) {
		this.type = t;
}

/**
 *
 * @return tipo di sensore
 */
public int getType() {
		return type;
}

/**
 * @param time_stamp time stamp (millisecondi) di rilevazione del sensore
 */
public void setTime_stamp(long time_stamp) {
```

```
		this.time_stamp = time_stamp;
}

/**
 * @return time stamp (millisecondi) di rilevazione del sensore
 */
public long getTime_stamp() {
		return time_stamp;
}

/**
 * @param value valori rilevati dal sensore
 */
public void setValue(float[] value) {
		this.value = value;
}

/**
 * @return valori rilevati dal sensore
 */
public float[] getValue() {
		return value;
}

/**
 * @param accuracy precisione del sensore
 */
public void setAccuracy(int accuracy) {
		this.accuracy = accuracy;
}

/**
 * @return precisione del sensore
 */
public int getAccuracy() {
		return accuracy;
}

/**
 * setta il tipo di template jess
```

```
 */
public void setJesstype() {
	switch ( type ) {
	// Sensor.TYPE_ACCELEROMETER = 1
	case SensorType.TYPE_ACCELEROMETER:
		jesstype= "accelerometer";
		break;
	case SensorType.TYPE_GYROSCOPE:
		jesstype="gyroscope";
		break;
	case SensorType.TYPE_LIGHT:
		jesstype="light";
		break;
	case SensorType.TYPE_LINEAR_ACCELERATION:
		jesstype="linear_acceleration";
		break;
	case SensorType.TYPE_ORIENTATION:
		jesstype="orientation";
		break;
	case SensorType.TYPE_PROXIMITY:
		jesstype="proximity";
		break;
	case SensorType.TYPE_TEMPERATURE:
		jesstype="temperature";
		break;
	case SensorType.TYPE_LOCATION:
		jesstype="location";
	default:
		break;
	}
}

/**
 *
 * @return tipo di template jess
 */
public String getJesstype() {
	return jesstype;
}
```

```
	/**
	 *
	 * @param jesstype tipo di template jess
	 */
	public void setJesstype(String jesstype) {
		this.jesstype = jesstype;
	}
}
```

iii. ReasoningManager.java

```
package frasi.service;

import org.apache.axis2.context.MessageContext;
import org.apache.axis2.context.ServiceContext;
import frasi.engine.ReasoningManager;
import frasi.entity.SensorEntity;

/**
 *
 * @author ICAR-CNR
 *
 */
public class WebServiceInterface {

//	SensorEntity[] allAttributes;

//  ReteManager rete;
   /**
    * effettua la registrazione del dispositivo
    * @param uid identificativo univoco del dispositivo
    * @param width larghezza (in pixel) dello schermo
    * @param height altezza (in pixel) dello schermo
    * @return url pagina web personalizzata
    */
   public String handshake(String uid, int width, int height, String network, String speed) {

	//buffer dei sensori
	SensorEntity[] allAttributes = new SensorEntity[4]; //0=luce, 1=accelerazione,
	for (int i=0; i < allAttributes.length; i++){
```

```
                allAttributes[i]= new SensorEntity();
        }

        ReasoningManager rete = new ReasoningManager(allAttributes);
        rete.assertScreenSize(width, height);
        rete.assertConnection(network, Double.parseDouble(speed));
        ServiceContext context = MessageContext.getCurrentMessageContext().getServiceContext();
        context.setProperty(uid, rete);
        rete.start();
//      return "http://194.119.214.102:8080/FrasiWebService/result.html"; //tonino
        return "http://150.145.115.245:8080/FrasiWebService/result.html"; //lavoro
//      return "http://192.168.0.9:8080/FrasiWebService/result.html"; //casa
    }



    /**
     * setta i valori dei sensori ricevuti dal dispositivo
     * @param uid identificativo univoco del dispositivo
     * @param type tipo del sensore
     * @param accuracy precisione del sensore
     * @param ts istante di rivelazione del sensore (millisecondi)
     * @param values valori rivelati dal sensore (stringa separata da ";")
     */
        public void setSensor(String uid, int type, int accuracy, long ts, String values) {

                // parser
                String[] result = values.split(";");
                float[] floatvalues = new float[result.length];
                for (int i=0; i<result.length; i++) {
                        floatvalues[i] = Float.parseFloat(result[i]);
                }

                SensorEntity sensor = new SensorEntity(type, accuracy, ts, floatvalues);
                sensor.setJesstype();

                String sensorType= sensor.getJesstype();
                int index=0;
                if (sensorType.equals("light"))
                        index=0;
```

```
                else if (sensorType.equals("accelerometer"))
                        index=1;
                else if (sensorType.equals("proximity"))
                        index=2;
                else if (sensorType.equals("location"))
                        index=3;
                ServiceContext context =
MessageContext.getCurrentMessageContext().getServiceContext();

                ReasoningManager rete = (ReasoningManager) context.getProperty(uid);
                SensorEntity[] attributes = rete.getAttributes();
                attributes[index] = sensor;
                rete.setAttributes(attributes);

        }

        /**
         * setta i valori della localizzazione ricevuti dal dispositivo
         * @param uid identificativo univoco del dispositivo
         * @param provider fornitore della location (GPS, WIFI, Cella)
         * @param accuracy precisione della loaction
         * @param ts istante di rivelazione della location (millisecondi)
         * @param lat latitudine
         * @param lon longitudine
         */
        public void setLocation(String uid, String provider, String accuracy, long ts, double lat,
double lon) {

                ServiceContext context =
MessageContext.getCurrentMessageContext().getServiceContext();

                ReasoningManager rete = (ReasoningManager) context.getProperty(uid);
                rete.assertLocation(provider, accuracy, ts, lat, lon);
        }

        /**
         * setta i valori della connessione ricevuti dal dispositivo
         * @param uid identificativo unico del dispositivo
         * @param type tipo di connessione (WIFI, ETHERNET, ecc...)
         * @param bandwidth larghezza di banda
         */
```

```
//		public void setConnectivity(String uid, String type, float bandwidth) {
//			return type;
//		}

}
```

iv. MakaHTML.java

```
package frasi.engine;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

import jess.Context;
import jess.JessException;
import jess.RU;
import jess.Userfunction;
import jess.Value;
import jess.ValueVector;

public class MakeHTML implements Userfunction{


		public static String movingResult = "";
		public static String accResult = "";
		public static String lightResult = "";
		public static String proximityResult = "";
		public static String connectionResult = "";
		public static String screenResult = "";
		/** The Jess name of the user function */
	public static final String functionName = "make-html";

		@Override
		public Value call(ValueVector vv, Context context) throws JessException {

			accResult= vv.get(1).stringValue(context);
			movingResult= vv.get(2).stringValue(context);
			lightResult= vv.get(3).stringValue(context);
```

```
		proximityResult= vv.get(4).stringValue(context);
		connectionResult = vv.get(5).stringValue(context);
		screenResult = vv.get(6).stringValue(context);
		File outputHTML = new File("D:/workspace/.metadata/.plugins/org.eclipse.wst.server.core/tmp0/wtpwebapps/FrasiWebService/result.txt"); //percorso della pagina web nel server
		FileWriter fileoutHTML;
		try {
			fileoutHTML = new FileWriter(outputHTML);
			PrintWriter out = new PrintWriter(fileoutHTML);

			// CREA LA PAGINA HTML
			out.println(screenResult);
			out.println(connectionResult);
			out.println(accResult);
			out.println(movingResult);
			out.println(lightResult);
			out.println(proximityResult);
			out.close();




		} catch (IOException e) {
			e.printStackTrace();
		}

		// TODO Auto-generated method stub
		return new Value("ok", RU.STRING);
	}



	@Override
	public String getName() {
		// TODO Auto-generated method stub
		return functionName;
	}
```

```
}
```

v. ServiceClient.java

```
package frasi.client;

import java.util.Timer;
import java.util.TimerTask;

import org.ksoap2.SoapEnvelope;
import org.ksoap2.serialization.SoapObject;
import org.ksoap2.serialization.SoapPrimitive;
import org.ksoap2.serialization.SoapSerializationEnvelope;
import org.ksoap2.transport.HttpTransportSE;
import frasi.entity.MarshalDouble;

import android.app.Activity;
import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.net.ConnectivityManager;
import android.net.wifi.WifiInfo;
import android.net.wifi.WifiManager;
import android.os.Bundle;
import android.provider.Settings.Secure;
import android.telephony.TelephonyManager;
import android.util.DisplayMetrics;
import android.view.View;
import android.view.View.OnClickListener;
import android.webkit.WebView;
import android.widget.Button;
import android.widget.TextView;

/**
 * The main activity of the android app.
```

```
 * This client application manages the variation of all device sensors and
 * it sends attributes configuration to web service,
 * that is able to generate a proper presetation layer.
 *
 *
 * @author ICAR-CNR
 *
 */
public class ServiceClient extends Activity implements OnClickListener, SensorEventListener, LocationListener {
    private Button myButton;
        private TextView lblResult;
//      private TextView output;
        private WebView webview;
        private SensorManager mSensorManager;
        private LocationManager locationManager;
        private WifiManager wifi;
        private TelephonyManager tm;
        private Sensor accel;
        private Sensor light;
        private Sensor prox;

        private static final String NAMESPACE = "http://service.frasi";
        private static String URL="http://192.168.0.9:8080/FrasiWebService/services/FrasiInterface?wsdl"; //indirizzo del webservice
        private static final String METHOD_NAME = "handshake";
        private static final String SOAP_ACTION = "http://service.frasi/handshake";
        private static final String METHOD_NAME2 = "setSensor";
        private static final String SOAP_ACTION2 = "http://service.frasi/setSensor";
        private static final String METHOD_NAME3 = "setLocation";
        private static final String SOAP_ACTION3 = "http://service.frasi/setLocation";

        private String android_id;
        private Timer time;



        /**
         * Empty constructor. It is never used by the user.
         */
```

```
public ServiceClient() {
        // TODO Auto-generated constructor stub
}



        /**
         *
         *  It is called when the activity is first created.
         *  */
 @Override
 public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        lblResult = (TextView) findViewById(R.id.result);
        time = new Timer(true);
        android_id = Secure.getString(getApplicationContext().getContentResolver(),
                        Secure.ANDROID_ID);
        webview = (WebView) findViewById(R.id.webView);
        webview.getSettings().setJavaScriptEnabled(true);
        webview.clearCache(true);
        myButton = (Button) findViewById(R.id.button);
        myButton.setOnClickListener(this);

        mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        accel = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        light = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
        prox = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);

        locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);



 }



 /**
  * It catches touchscreen input,
  * getting connection parameter to set network information.
  *
  */
```

```
    public void onClick(View v) {
        // TODO Auto-generated method stub
        DisplayMetrics displaymetrics = new DisplayMetrics();
        getWindowManager().getDefaultDisplay().getMetrics(displaymetrics);
        int height = displaymetrics.heightPixels;
        int width = displaymetrics.widthPixels;
        double speed = 0.0;
        String networktype ="";
        myButton.setEnabled(false);
        webview.setMinimumHeight(600);
        ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
      String networkinfo = cm.getActiveNetworkInfo().getTypeName();

      if (networkinfo.equals("WIFI")){
        wifi = (WifiManager) getSystemService(Context.WIFI_SERVICE);
        WifiInfo wifiinfo = wifi.getConnectionInfo();
        speed = wifiinfo.getLinkSpeed();
        networktype = "WIFI";
      }
      else if (networkinfo.equals("MOBILE")) {
         tm = (TelephonyManager) getSystemService(TELEPHONY_SERVICE);
         int type = tm.getNetworkType();
         switch (type) {
         case TelephonyManager.NETWORK_TYPE_EDGE:
                networktype = "EDGE";
                speed = 0.237;
                break;
         case TelephonyManager.NETWORK_TYPE_GPRS:
                networktype = "GPRS";
                speed = 0.115;
                break;
         case TelephonyManager.NETWORK_TYPE_HSDPA:
                networktype = "HSDPA";
                speed = 14.4;
                break;
         case TelephonyManager.NETWORK_TYPE_UMTS:
                networktype = "UMTS";
                speed = 16.0;
                break;
```

```
            }

        }
            this.sendhandshake(android_id, width, height, networktype, speed);
            time.scheduleAtFixedRate(new OnListenr(), 2000, 5000);//da rimettere
            time.scheduleAtFixedRate(new OffListener(), 2000, 2000);//da rimettere
        locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,    5*60*1000,    0,
Listener); //mintime = 10000


    }



    /**
     * Start the handshake authentication by means of the SOAP protocol.
     * Some information about the device are embedded into a SOAP object and serialized.
     *
     *
     * @param uid  Unique Identifier of device
     * @param w  screen width
     * @param h  screen height
     * @param network  network parameter
     * @param speed  connection speed
     */
            public void sendhandshake(String uid, int w, int h, String network, double speed) {

        SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME);
        request.addProperty("width", w);
        request.addProperty("height", h);
        request.addProperty("network", network);
        request.addProperty("speed", Double.toString(speed));

        SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);

        envelope.setOutputSoapObject(request);
        HttpTransportSE androidHttpTransport = new HttpTransportSE(URL);

        try {
            androidHttpTransport.call(SOAP_ACTION, envelope);

            SoapPrimitive  resultsRequestSOAP = (SoapPrimitive) envelope.getResponse();
```

```
        webview.loadUrl(resultsRequestSOAP.toString());



} catch (Exception e) {
        lblResult.setText(e.toString());

}

        }

        /**
         * Override method: Never used in this application
         */
        public void onProviderDisabled(String str1) {
                // TODO Auto-generated method stub

        }

        /**
         * Override method: Never used in this application
         */
        public void onProviderEnabled(String str1) {
                // TODO Auto-generated method stub

        }

        /**
         * Override method: Never used in this application
         */
        @Override
        public void onStatusChanged(String str1, int status, Bundle str2) {
                // TODO Auto-generated method stub

        }

        /**
         * Override method: Never used in this application
         */
```

```
	public void onAccuracyChanged(Sensor sensor, int accuracy) {
		// TODO Auto-generated method stub

	}

	/**
	 * Catch the sensor changed event:
	 * when the status of device sensors changes,
	 * this method catches the type of sensor, the time-stamp, the accuracy and finally the input value(s).
	 */
	public void onSensorChanged(SensorEvent event) {
		// TODO Auto-generated method stub
		int type = event.sensor.getType();
		int accuracy = event.accuracy;
		long ts = event.timestamp;
		float[] values = event.values;
//		output.setText(String.valueOf(values[0]));

		sendSensor(type, accuracy, ts, values);

	}

	/**
	 * Catch the location changed event:
	 * when the location of device changes,
	 * this method catches the 'location sensor' provider, the accuracy, the time-stamp and finally the location parameters (i.e. latitude and longitude).
	 */
	public void onLocationChanged(Location location) {
			// TODO Auto-generated method stub
			sendLocation(location.getProvider(), location.getAccuracy(), location.getTime(), location.getLatitude(), location.getLongitude());

		}

	/**
	 * Send location attribute to the web-service through the SOAP protocol:
	 * the message will contain:
	 * the 'location sensor' provider, the accuracy, the time-stamp and finally the latitude and longitude.
```

```
     *
     * @param provider  provider name
     * @param accuracy  signal accuracy
     * @param ts  time-stamp
     * @param latitude  device latitude
     * @param longitude  device longitude
     */
    public void sendLocation(String provider, float accuracy, long ts, double latitude, double longitude) {
        // TODO Auto-generated method stub

        MarshalDouble md = new MarshalDouble();
        SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME3);
        request.addProperty("provider", provider);
        request.addProperty("accuracy", String.valueOf(accuracy));
        request.addProperty("ts", ts);
        request.addProperty("lat", latitude);
        request.addProperty("lon", longitude);


        SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);
        md.register(envelope);
        envelope.setOutputSoapObject(request);
//      envelope.implicitTypes = true;

        envelope.addMapping(NAMESPACE, "lat",md.getClass());
        envelope.addMapping(NAMESPACE, "lon",md.getClass());
        HttpTransportSE androidHttpTransport = new HttpTransportSE(URL);

        try {
            androidHttpTransport.call(SOAP_ACTION3, envelope);

            SoapPrimitive resultsRequestSOAP = (SoapPrimitive) envelope.getResponse();




        } catch (Exception e) {
```

```
		}
	}

	/**
	 * Send a generic sensor information to the web-service through the SOAP protocol:
	 * the message will contain:
	 * the type of sensor, the accuracy, the time-stamp and finally the input value(s).
	 *
	 *
	 * @param type  type of sensor
	 * @param accuracy  signal accuracy
	 * @param ts  time-stamp
	 * @param values  sensor value array
	 */
	public void sendSensor(int type, int accuracy, long ts, float[] values) {
		// TODO Auto-generated method stub
		webview.clearCache(true);
SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME2);
String stringvalue = "";

for (int i=0; i<values.length; i++) {
	stringvalue+=values[i]+";";
}
request.addProperty("type", type);
request.addProperty("accuracy", accuracy);
request.addProperty("ts", ts);
request.addProperty("values", stringvalue);


SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);

envelope.setOutputSoapObject(request);
HttpTransportSE androidHttpTransport = new HttpTransportSE(URL);

try {
 androidHttpTransport.call(SOAP_ACTION2, envelope);

  SoapPrimitive  resultsRequestSOAP = (SoapPrimitive) envelope.getResponse();
```

```
        lblResult.setText(resultsRequestSOAP.toString());


    } catch (Exception e) {
        lblResult.setText(e.toString());

    }

        }

        protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(LightListener);
    mSensorManager.unregisterListener(AccelerationListener);
    mSensorManager.unregisterListener(ProximityListener);
  }

        private SensorEventListener LightListener = new SensorEventListener() {


                public void onSensorChanged(SensorEvent event) {
                        // TODO Auto-generated method stub
                        int type = event.sensor.getType();
                        int accuracy = event.accuracy;
                        long ts = event.timestamp;
                        float[] values = event.values;

                        sendSensor(type, accuracy, ts, values);
                }

                public void onAccuracyChanged(Sensor sensor, int accuracy) {
                        // TODO Auto-generated method stub

                }
        };

private SensorEventListener AccelerationListener = new SensorEventListener() {

                public void onSensorChanged(SensorEvent event) {
                        // TODO Auto-generated method stub
```

```
                int type = event.sensor.getType();
                int accuracy = event.accuracy;
                long ts = event.timestamp;
                float[] values = event.values;

                sendSensor(type, accuracy, ts, values);
        }

        public void onAccuracyChanged(Sensor sensor, int accuracy) {
                // TODO Auto-generated method stub

        }
};

private SensorEventListener ProximityListener = new SensorEventListener() {

        public void onSensorChanged(SensorEvent event) {
                // TODO Auto-generated method stub
                int type = event.sensor.getType();
                int accuracy = event.accuracy;
                long ts = event.timestamp;
                float[] values = event.values;

                sendSensor(type, accuracy, ts, values);
        }

        public void onAccuracyChanged(Sensor sensor, int accuracy) {
                // TODO Auto-generated method stub

        }
};

private LocationListener Listener = new LocationListener() {

        public void onStatusChanged(String provider, int status, Bundle extras) {
                // TODO Auto-generated method stub

        }

        public void onProviderEnabled(String provider) {
```

```
                // TODO Auto-generated method stub

            }

            public void onProviderDisabled(String provider) {
                // TODO Auto-generated method stub

            }

            public void onLocationChanged(Location location) {
                // TODO Auto-generated method stub
                sendLocation(location.getProvider(), location.getAccuracy(),
location.getTime(), location.getLatitude(), location.getLongitude());

            }
        };

        private class OffListener extends TimerTask {

            @Override
            public void run() {
                // TODO Auto-generated method stub
            mSensorManager.unregisterListener(LightListener);
            mSensorManager.unregisterListener(AccelerationListener);
            mSensorManager.unregisterListener(ProximityListener);
                }

        }

        private class OnListenr extends TimerTask {

            @Override
            public void run() {
                // TODO Auto-generated method stub

mSensorManager.registerListener(LightListener,light,SensorManager.SENSOR_DELAY_NORMAL);

mSensorManager.registerListener(AccelerationListener,accel,SensorManager.SENSOR_DELAY_NO
RMAL);

mSensorManager.registerListener(ProximityListener,prox,SensorManager.SENSOR_DELAY_NORM
AL);
```

```
                }

            }

}
```

b. Codice Sorgente Jess

i. jess_frasi.clp

```
; I template riportati di seguito definiscono la struttura degli attributi
; che possono essere ricevuti dal dispositivo.
; Una combinazione delle istanze di tali template, chiamati fatti,
; verranno sottoposti al motore a regole che individuerà una particolare azione per il presentation layer.

(import java.lang.Math)
(defglobal ?*accResult* = "" )
(defglobal ?*movingResult* = "" )
(defglobal ?*lightResult* = "" )
(defglobal ?*proximityResult* = "" )
(defglobal ?*connectionResult* = "" )
(defglobal ?*screenResult* = "" )

(deftemplate foo "dummy fact"
   (slot result))

(deftemplate accelerometer "accelerometro"
   (slot accuracy)
   (slot timestamp)
   (slot a_x)
   (slot a_y)
   (slot a_z)
   )

(deftemplate gyroscope "grado di rotazione"
   (slot accuracy)
   (slot timestamp)
   (slot g_x)
   (slot g_y)
   (slot g_z)
```

```
    )

(deftemplate light "luce ambientale"
    (slot accuracy)
    (slot timestamp)
    (slot lux)
    )

(deftemplate proximity "prossimita'"
    (slot accuracy)
    (slot timestamp)
    (slot distance)
    )


(deftemplate orientation "orientamento"
    (slot accuracy)
    (slot timestamp)
    (slot azimuth)
    (slot pitch)
    (slot roll)
    )


(deftemplate location "posizione del dispositivo"
            (slot accuracy)
    (slot provider)
    (slot timestamp)
    (slot lat)
    (slot lon)
    )

(deftemplate screen_size "dimensione dello schermo"
     (slot s_h)
     (slot s_w)
    )

(deftemplate connection_type "dati della connessione"
    (slot type)
    (slot speed)
```

```
    )
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(load-function frasi.engine.GetAllTemplateFacts)
(load-function frasi.engine.MakeHTML)

;query che mi restituiscono tutti i fatti di un determinato tipo

; mi restituisce tutte le istanze del sensore "location" in questo modo posso fare una analisi differenzale
(defquery find-all-instances-of-location "find all instances of location"
            (location )
            )

(defquery find-all-instances-of-accelerometer "find all instances of accelerometer"
    (accelerometer )
            )

; Regole tipo per la generazione dell'azione che caratterizza il presentation layer
; in questa fase, verranno utilizzati pochipresentation layer parametrizzati
; le regole qui di seguito sono in grado di selezionare quale presentation layer attivare
; e quali sono i parametri da passare per generare l'azione.

;Regola sulle dimensioni dello schermo
(defrule screenRule
    (screen_size (s_h ?height) (s_w ?width))
    =>
            (bind ?*screenResult* (str-cat "- Ottimizza la visualizzazione per la risoluzione " ?height " x "
?width))
)

;Regole sul tipo di connessione
(defrule connectionRule "cambia la qualita' dei contenuti accessibili in base al tipo di connessione"
    (connection_type (type ?type) (speed ?speed))
    =>
    (if (> ?speed 1) then
        (bind ?*connectionResult* (str-cat "la connessione e' " ?type " con velocita' " ?speed " Mbps.
utilizza contenuti completi"))
        else
         (bind ?*connectionResult* (str-cat "la connessione e' " ?type " con velocita' " ?speed " Mbps.
utilizza contenuti ridotti")))
  ;  (= (str-compare ?miss "true" ) 0)
```

```
   )

; REGOLE DI MOVIMENTO (agiscono sul contesto)

(defrule accRule "cambia la dimensione dei caratteri in funzione dello shaking"
        ?last_acc <- (accelerometer (timestamp ?ts2) (accuracy ?acc2) (a_x ?a_x2) (a_y ?a_y2) (a_z ?a_z2))
        ;?last_acc-1 <-  (accelerometer (timestamp ?ts1&:(and (<> ?ts1 ?ts2) (<= (abs (- ?ts1 ?ts2)) 6000000))) (accuracy ?acc1) (a_x ?a_x1) (a_y ?a_y1) (a_z ?a_z1))
   ?last_acc-1 <-  (accelerometer (timestamp ?ts1&:(< ?ts1 ?ts2)) (accuracy ?acc1) (a_x ?a_x1) (a_y ?a_y1) (a_z ?a_z1))
   =>
      (bind ?force_x2 (call Math pow (/ ?a_x2 9.81) 2))
          (bind ?force_y2 (call Math pow (/ ?a_y2 9.81) 2))
          (bind ?force_z2 (call Math pow (/ ?a_z2 9.81) 2))
          (bind ?totalForce2 (sqrt (+ (+ ?force_x2 ?force_y2) ?force_z2)) )

      (bind ?force_x1 (call Math pow (/ ?a_x1 9.81) 2))
          (bind ?force_y1 (call Math pow (/ ?a_y1 9.81) 2))
          (bind ?force_z1 (call Math pow (/ ?a_z1 9.81) 2))
          (bind ?totalForce1 (sqrt (+ (+ ?force_x1 ?force_y1) ?force_z1)) )

          (if (and (> ?totalForce2 0.95) (> ?totalForce2 ?totalForce1) )  then
          (printout t "shake" crlf)
          (bind ?*accResult* "- Utente in movimento, visualizza caratteri grandi.")
      else
          (bind ?*accResult* " ")
          (printout t "no shake" crlf)
      )

   (retract ?last_acc-1)

   )

 (defrule movingRule "identifica il tipo di movimento in base alle rilevazioni della location"
   ?last_loc <- (location (timestamp ?ts2) (accuracy ?acc2)(lat ?lat2) (lon ?lon2))
        ?last_loc-1 <-  (location (timestamp ?ts1&:(< ?ts1 ?ts2)) (accuracy ?acc1)(lat ?lat1) (lon ?lon1))

   =>
```

```
   (bind ?r 6372.795477598)
   (bind ?radlat1 (/ (*  ?lat1) 180))
   (bind ?radlon1 (/ (* 3.14159 ?lon1) 180))
   (bind ?radlat2 (/ (* 3.14159 ?lat2) 180))
   (bind ?radlon2 (/ (* 3.14159 ?lon2) 180))

   (bind ?phi (abs (- ?radlon1 ?radlon2)))

   (bind ?p (call Math acos (+  (* (call Math sin ?radlat1) (call Math sin ?radlat2) ) (* (* (call Math cos
?radlon1) (call Math cos ?radlon2) ) (call Math cos ?phi) )  ) ) )

   (bind ?distanzaKm (* ?p ?r))

   (bind ?deltaTnanosec (abs (- ?ts1 ?ts2)) )

   (bind ?deltaTore (* ?deltaTnanosec (* 2.78 (call Math pow 10 -13)) ) )

   (bind ?velocita (/ ?distanzaKm ?deltaTore))

   (printout t ?velocita crlf)

    (bind ?*movingResult* (str-cat "- Sei a piedi. La tua velocita' e': " ?velocita) )

          (retract ?last_loc-1)
   )

; REGOLE DI VISUALIZZAZIONE (agiscono sullo stile)

(defrule lightRule "identifica la variazione della luce"
   ?last_lux <- (light (timestamp ?ts) (accuracy ?acc)(lux ?lux))
 =>
; (retract ?foo)
(if  (>= ?lux 400 ) then
   (bind ?*lightResult* "- Setta i parametri per la visione diurna ad alto contrasto.")
   else
   (if (<= ?lux 10) then
      (bind ?*lightResult* "- Setta i parametri per la visualizzazione ad alto contrasto notturna.")
      else
      (if (and (> ?lux 10 ) (< ?lux 400)) then
         (bind ?*lightResult* "- Setta i parametri per la visualizzazione con contrasto normale.")
```

```
        )
      )
   )
 )

(defrule proximityRule "identifica se il cellulare è stato portato all'orecchio o è stato ribaltato"
   ?last_prox <- (proximity (timestamp ?ts2) (accuracy ?acc2) (distance ?dist2&:(< ?dist2 5)))
   ?last_lux <- (light (timestamp ?ts&: (<= (abs (- ?ts ?ts2)) 10000000) ) (accuracy ?acc)(lux ?lux&:(< ?lux 10) ))

                                                ;6000000
 =>

   (bind ?*proximityResult* "- Attiva la descrizione audio dei contenuti.")

   )

(defrule proximityStopRule "reset proximityRule"
   ?last_prox <- (proximity (timestamp ?ts2) (accuracy ?acc2) (distance ?dist2&:(> ?dist2 5)))
   =>

   (bind ?*proximityResult* "")

   )




(defrule doOutput "scrive il risultato delle inferenze"
  (declare (salience -1))
   ;(initial-fact)
   ?foo <- (foo (result ?result))
   =>
   (retract ?foo)
   ;assegna la variabile (stringa) da stampare
   (make-html ?*accResult* ?*movingResult* ?*lightResult* ?*proximityResult* ?*connectionResult* ?*screenResult*)
   )

;(watch activations)
;(watch rules)
```