UNIVERSITÀ DEGLI STUDI DI BOLOGNA
FACOLTÀ DI INGEGNERIA

Corso di laurea in Ingegneria Elettronica
Reti di Calcolatori

INFRASTRUTTURA PER DISCOVERY E ADATTAMENTO DI SERVIZI MULTIMEDIALI

Candidato: Roberto Luciani
Relatore: Chiar.mo Prof. Ing. ANTONIO CORRADI
Correlatori: Dott. Ing. LUCA FOSCHINI
Chiar.mo Prof. Ing. PAOLO BELLAVISTA

Anno accademico 2004 – 2005

Parole chiave

Sistemi multimediali

Adattamento di servizi

Discovery di servizi

Agenti Mobili

Gestione handoff

Ai miei genitori.

INDICE

INTRODUZIONE

Le nuove tecnologie si sono evolute verso possibilità fino a pochi anni fa impensabili, come la fruizione su di un dispositivo mobile di servizi multimediali, o la possibilità di adattare, a tempo di esecuzione, i flussi multimediali erogati verso un terminale in base alle sue caratteristiche. Queste nuove opportunità hanno portato a focalizzare l'attenzione sul problema della distribuzione di flussi multimediali in ambito mobile. Conseguentemente sono cresciute e si sono sviluppate tematiche connesse alla mobilità del dispositivo, la principale delle quali è certamente quella di trovare e fornire servizi all'utente in base alla località in cui si trova e automatizzarne una loro eventuale configurazione. Oltre a ciò, sempre più utenti vogliono oggi accedere ai proprio servizi indipendentemente dal terminale da loro utilizzato e mantenere aperte le proprie sessioni mentre si muovono fra terminali diversi, si pensi ad esempio ad un utente che voglia spostare una telefonata dal proprio telefono cellulare al proprio PC. L'insieme di queste nuove esigenze, ha portato gli sviluppatori di software a esplorare soluzioni innovative: ad adottare paradigmi di programmazione avanzati, a definire nuovi protocolli, e a creare infrastrutture distribuite che si interpongono tra il sistema sottostante e le applicazioni utente per fornire a queste ultime servizi di supporto in maniera trasparente.

L'intento che si vuole raggiungere in questa tesi è realizzare un sistema di supporto alla distribuzione di contenuti multimediali su dispositivi mobili che coadiuvato da un sistema di ricerca di servizi, gestisca la transazione da terminale mobile ad un terminale fisso mantenendo nel contempo la sessione di cui si stava usufruendo e riadattandola in base alle caratteristiche del nuovo terminale.

Il sistema di ricerca di servizi, detto anche sistema di discovery, si occuperà di ricercare nella località servizi o dispositivi, in base ad una descrizione dei loro attributi e alle preferenze degli utenti. In particolare, il sistema di discovery dovrebbe ricercare in modo automatico servizi che rispondano alle esigenze degli utenti (espresse attraverso le loro preferenze). Ad esempio, quando un servizio di visualizzazione, come un monitor, è disponibile nella località attraversata dall'utente il servizio di discovery segnala ciò al sistema di supporto. Al fine di garantire una maggiore apertura del

progetto, il sistema di discovery sarà realizzato in modo tale da interagire con qualsiasi standard di discovery esistente.

Il sistema di supporto dovrà essere in grado di gestire l'erogazione del flusso diretto al terminale anche durante lo spostamento del terminale fra località coperte da diversi punti di accesso wireless (Access Point, AP), senza che il flusso venga interrotto, questa procedura è detta *handoff*. Inoltre interagendo con il sistema di discovery, ricercherà un terminale fisso nel quale l'utente possa spostare la sessione, configurandolo pro-attivamente, ancor prima di informare l'utente, in modo tale che il cambio sia istantaneo. Lo spostamento della sessione su un altro terminale è detto *offload* e può includere una fase di riadattamento del flusso, in modo da sfruttare al meglio le caratteristiche del terminale.

Il risultato di tale studio sarà la proposta e lo sviluppo di un'architettura per la realizzazione di tale sistema che verrà progettato nell'ottica di arricchire un'infrastruttura di supporto allo sviluppo di applicazioni multimediali, realizzata presso il Dipartimento di Elettronica Informatica e Sistemistica (DEIS) dell'Università di Bologna.

La tesi sarà organizzata come segue. Nel primo capitolo verrà presentato lo scenario applicativo nel quale verrà inserito il progetto oggetto della tesi e una panoramica sui concetti di reti wireless, introduzione e problematiche del discovery di servizi ed infine l'introduzione al concetto di proxy. Il capitolo 2 sarà dedicato allo studio delle possibili strategie di discovery adottabili e alla discussione di alcune delle soluzioni più rappresentative presentate in alcuni recenti lavori di ricerca. Nel capitolo 3 verranno presentate gli standard attualmente più utilizzati per il discovery dei servizi. Nel capitolo 4 verranno presentate le tecnologie utilizzate per lo svolgimento della tesi, in particolare SOMA, MUM e JMF. Lo scopo del progetto di tesi è quello di arrichire SOMA e MUM con nuove funzionalità che verranno presentate nei capitoli 5, 6 e 7. In particolare il capitolo 5 sarà dedicato all'analisi del sistema di discovery e distribuzione di contenuti multimediali su terminali mobili e alla sviluppo della loro architettura. Nel capitolo 6 verranno illustrate le scelte progettuali che hanno guidato l'implementazione del sistema, presentata nel capitolo 7 unitamente ai risultati raccolti nella fase di sperimentazione.

CAPITOLO 1

DISCOVERY E ADATTAMENTO IN SISTEMI MOBILI

I recenti progressi tecnologici hanno reso i dispositivi portatili come cellulari e palmari, sempre più potenti e con maggiori capacità computazionali, contemporaneamente l'aumento della banda disponibile nei collegamenti wireless ha permesso operazioni multimediali sempre più evolute. Nonostante queste loro caratteristiche, i dispositivi portatili risultano essere ancora molto limitati rispetto a postazioni fisse, e non sono in grado di svolgere alcuni servizi attualmente richiesti dagli utenti.

Il servizio di discovery si inserisce in questo contesto proprio per sopperire a queste mancanze, fornendo a questi dispositivi la possibilità di accedere a servizi più evoluti in qualsiasi posto essi si trovino attraverso un sistema di localizzazione dei servizi nella rete. I servizi offerti non si limitano solo a elaborazioni complesse, come ad esempio il supporto per un middleware completo, ma anche a proprietà caratteristiche del dispositivo, come la possibilità di visionare un contenuto multimediale sopra uno schermo di dimensioni maggiori o la possibilità di stampare documenti, operazioni fisicamente impossibili su questi dispositivi. La distribuzione di servizi porterà lo sviluppo futuro verso uno scenario di *pervasive computing* dove le risorse sono distribuite e l'utente ha la possibilità di accedervi in qualsiasi momento e in qualunque luogo si trovi. Affinché sia possibile realizzare questo scenario è necessario, oltre a sviluppare un sistema di localizzazione dei servizi, riuscire a gestire le eterogeneità dei diversi dispositivi che partecipano, attraverso un sistema di adattamento dei servizi e dei contenuti che si occupa di eseguire le trasformazioni necessarie per il passaggio fra dispositivi con caratteristiche diverse.

In questo capitolo verranno presentate alcune nozioni utili per la comprensione dei capitoli successivi, in modo da creare un background di conoscenze necessarie alla comprensione del lavoro svolto.

1.1 Le Reti Wireless

Per reti wireless si intende in generale, un serie di dispositivi che vanno dai PC ai cellulari, collegati tra loro senza mezzi fisici come ad esempio cavi o fili di qualsiasi genere, in quanto la comunicazione tra i vari dispositivi avviene tramite apposite antenne che fungono sia da trasmittente che da ricevente.

Il vantaggio evidente di questa tecnologia è che non c'è bisogno di una postazione fissa per partecipare ad una rete, consente una certa mobilità e svincola dalla necessità di disporre di connettori e cavi. L'utente può connettersi alla rete ovunque egli ritenga più comodo tramite un dispositivo portatile, a condizione che si trovi all'interno di un'area coperta da una rete wireless. Infatti, sebbene non necessiti di cavi, la rete wireless ha comunque un'estensione massima che è data dalle portate delle antenne che trasmettono e ricevono il segnale e che fungono da punti di accesso alla rete stessa, i cosiddetti Access Point (AP).

Oltre alla mobilità le reti wireless hanno anche portato ad una più ampia estensione dell'accesso ad Internet: prima dell'avvento di questa tecnologia per accedere ai servizi legati ad Internet era necessario possedere una connessione privata oppure poter accedere a postazioni pubbliche che avevano comunque il limite di poter offrire un numero limitato di accessi. Ora invece l'accesso alla rete Internet può avvenire da qualsiasi punto che sia coperto dal segnale proveniente da un AP.

È possibile individuare due tipi di accesso ad un rete wireless: tramite una infrastruttura fissa ed accesso ad hoc. Nel caso di infrastruttura fissa, i terminali mobili possono venir a far parte di una rete fissa tramite l'accesso ad un AP collegato alla rete stessa, abbiamo quindi un sistema di accesso centralizzato dove tutti i terminali si riferiscono ad uno stesso AP, questa struttura è molto simile ad un sistema di telefonia cellulare, dove i terminali sono connessi con l'antenna più vicina. Nel caso di rete ad hoc, non esiste un punto di accesso uguale per tutti, in quanto gli stessi terminali possono fungere da punto di accesso per altri terminali, il sistema è distribuito rispetto al caso precedente e tutti i terminali sono pari fra loro. Il lavoro svolto si concentrerà sul primo sistema di accesso, in modo da concentrare tutti i problemi relativi ad eventuali disconnessioni e cali di segnale, solo nell'ultimo tratto, che collega AP e terminale.

1.1.1 Un confronto con le reti fisse

Oltre ai vantaggi precedentemente evidenziati che hanno guidato ad un'espansione sempre maggiore di queste reti, queste tecnologie comportano anche alcuni svantaggi non trascurabili rispetto alle reti fisse.

Il mezzo di trasmissione scelto comporta implicitamente una banda minore rispetto a quella ottenibile in una rete fissa per le proprietà stesse della trasmissione attraverso onde elettromagnetiche.

Garantire un accesso più esteso ad Internet vuol dire anche avere un maggior numero di utenti da servire, ma non implica avere più banda disponibile. Gli AP sono dispositivi spesso collegati ad una rete fissa capace di una certa velocità di trasferimento che l'AP utilizza per servire gli utenti ad esso collegati. Quindi, più utenti sono collegati ad un AP, maggiormente questo dovrà suddividere la banda disponibile equamente tra di loro. Questo problema in generale affligge tutte le reti, ma in particolare le reti wireless poiché, come detto, danno modo ad un maggior numero di utenti di collegarsi. Questo avviene se non si predispone un controllo degli accessi al servizio, ovvero se si consente a chiunque voglia di accedere alla rete; questo problema può essere limitato accettando solo un numero di utenti tale da poter garantire una certa qualità di servizio.

Oltre a ciò, tradizionalmente l'accesso alla rete Internet veniva effettuato da macchine con determinate caratteristiche che permettevano di accomunarle, ed in generale si parlava di computer dalle diverse potenze e prestazioni, ma caratterizzati da una struttura comune.

Questi nuovi tipi di reti, invece, prevedono l'accesso di dispositivi che presentano una maggiore varietà e vanno da potenti laptop a dispositivi più limitati come i Personal Digital Assistans (PDA). Questo fa si che difficilmente si possano fare ipotesi sul dispositivo con cui ci si trova a dialogare, è quindi necessario adattare il servizio alle caratteristiche del dispositivo. Inoltre, reti di questo tipo sono più inaffidabili rispetto alle fisse e la perdita di informazione può essere sensibilmente maggiore poiché il mezzo fisico (etere) è più soggetto ad interferenze ed errori. La comunicazione è anche sensibile ad eventuali ostacoli presenti sul cammino tra AP e utente, come muri e sensibile anche alle interferenza con altri tipi di strumentazione che comunicano utilizzando lo stesso mezzo trasmissivo.

Altro problema da considerare per completare il panorama è quello della sicurezza che, sebbene non centrale in questo elaborato, è invece piuttosto rilevante nella diffusione dell'uso di queste reti ad esempio in uffici o banche: questo problema è legata alla natura stessa del mezzo di comunicazione che, a differenza di un cavo, non porta l'informazione unicamente al destinatario, ma la diffonde a chiunque sia in grado di riceverla all'interno dell'area.

1.1.2 Handoff

Dai precedenti paragrafi si è visto che per poter accedere ad una rete Wireless si deve instaurare una comunicazione con un punto di accesso chiamato, l'AP. Ognuna di queste antenne ha però una portata limitata che varia a seconda della tecnologia utilizzata. La zona geometrica all'interno della quale l'AP riesce a comunicare con il dispositivo, è detta cella. La qualità del segnale aumenta con la vicinanza a tali AP e degrada quanto più ci allontaniamo dal centro della cella, fin quando non è più sufficiente a mantenere una comunicazione. Per questo motivo si dispongono più AP in modo da coprire l'area desiderata.

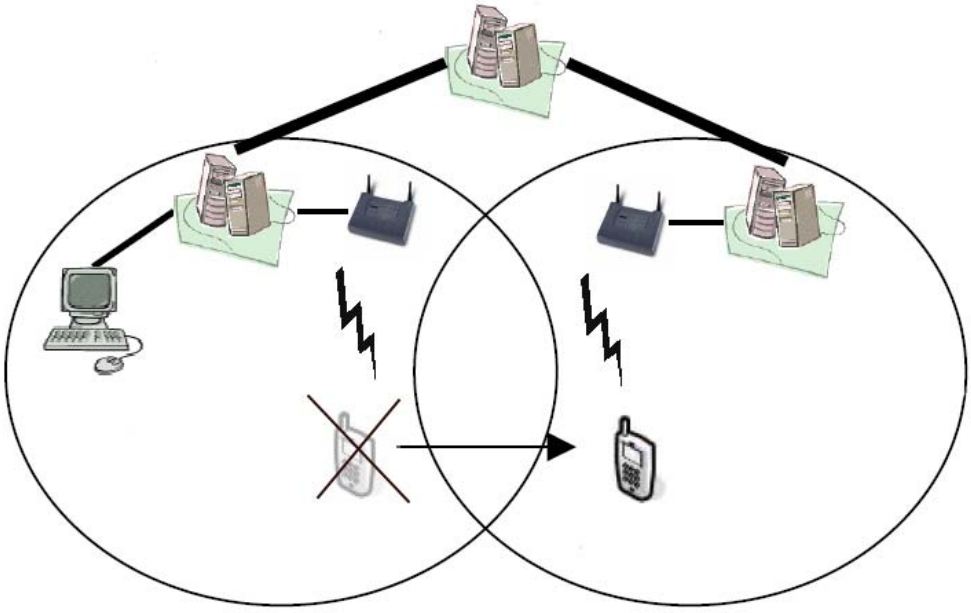


Figura 1.1: Cambio di AP, nel processo di handoff.

Accade così che mentre ci allontaniamo da un AP ci avviciniamo al successivo e ad un certo punto perdiamo la comunicazione con l'AP precedente per instaurarla con quello successivo, figura 1.1. Ad esempio, la rete per la telefonia cellulare funziona in questo modo, facendoci comunicare in ogni istante con l'antenna più vicina a noi. Questo processo di distaccamento e successiva riconnessione prende il nome di *handoff* ed in generale può essere diverso a seconda della tecnologia wireless utilizzata.

1.2 Discovery e naming: elementi introduttivi

Nel presente paragrafo introdurremo il concetto di discovery, ma prima di procedere è necessario sapere cosa si intende con middleware, in quanto verrà più volte menzionato.

Un middleware è un ambiente di supporto per integrare fra loro sistemi eterogenei, fornendo quindi ad un sistema la possibilità di usufruire dei servizi di un altro, in particolare fungendo da mediatore nel caso in cui i protocolli di comunicazione fossero diversi. Il termine middleware deve essere considerato nel modo più generale possibile, questo infatti potrebbe essere un infrastruttura distribuita, oppure una semplice libreria di codice incorporata insieme al servizio e ai clienti in modo che questi possano interagire con gli stessi protocolli. L'obbiettivo finale di un middleware è, quindi, quello di fornire servizi aggiuntivi ad un sistema, mantenendo il protocollo di comunicazione utilizzato dal sistema stesso. Un particolare servizio fornito dal middleware è rappresentato dal servizio di discovery.

Con il termine discovery, si vuole indicare un processo di localizzazione di servizi che può venire incontro a certi requisiti specificati dall'utente. Un modello generale di naming e discovery è mostrato in Figura 1, tratta da [NamDM]. In questo modello, un servizio si registra con il middleware, mentre il cliente fa richiesta al middleware per ottenere risultati sul servizio desiderato.

Il servizio registrandosi specifica un nome che lo descrive e alcune informazioni che specificano meglio le sue caratteristiche. Per riflettere gli eventuali cambiamenti del servizio durante la sua esecuzione, è necessario realizzare meccanismi per l'aggiornamento delle registrazioni. Una richiesta di servizio da parte del cliente può

essere persistente, cioè viene memorizzata e il middleware notificherà il cliente nel caso in cui il risultato di quella richiesta sia cambiata.

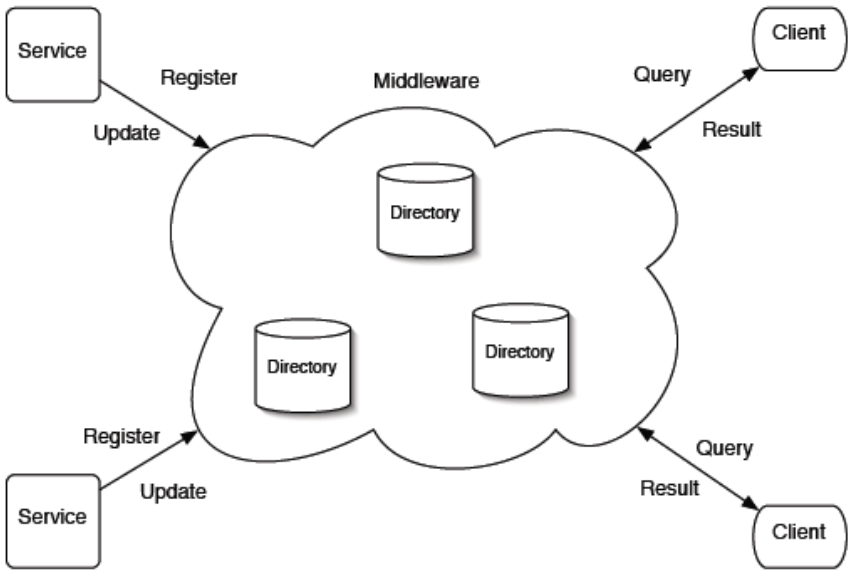


Figura 1.2: Un modello generale di un middleware per discovery e naming.

I nomi dei vari servizi formano quello che viene chiamato "spazio di nomi", mentre con il termine *directory* che compare nella Figura 1.2 si indica una struttura dati che contiene i nomi dei servizi ed eventualmente altre informazioni che riguardano il servizi stessi. Il servizio di directory può essere realizzato tramite una directory centralizzata che memorizza l'intero spazio di nomi, oppure un insieme di directory distribuite, ognuna della quali memorizza un proprio spazio di nomi completo o solamente una porzione di esso.

Il nome rappresenta la descrizione del servizio, la quale potrebbe essere una semplice stringa, un insieme di attributi, oppure un documento XML [ChenKotz]. Una richiesta da parte di un cliente è specificata con una sintassi simile a quella del nome codificato all'interno del sistema di discovery, quando il middleware riceve la richiesta restituisce tutti i nomi che fanno match con la richiesta. I criteri di *matching* potrebbero essere semplici test di uguaglianza, comparazioni con *wildcard*, un sottoinsieme di attributi o una valutazione complessa sfruttando le caratteristiche di XML.

Una volta che il middleware restituisce il nome che fa match con la richiesta del cliente, questo può comunicare direttamente con il servizio selezionato. Alternativamente la comunicazione può essere mediata attraverso il middleware, in

questo caso il middleware funge da intermediario passando i messaggi avanti e indietro tra il cliente e il servitore.

1.2.1 Problematiche per il discovery negli ambienti mobili

Uno degli obiettivi principali per il discovery e naming è fornire interfacce o protocolli per facilitare l'interazione tra i clienti e servizi minimizzando i costi di amministrazione e configurazione del sistema. Possono essere evidenziati quattro requisiti che devono essere presenti in un ambiente mobile: mobilità, scalabilità, consapevolezza del contesto e sicurezza.

1.2.1.1 Mobilità e discovery

La mobilità rappresenta uno dei primi problemi che si presentano quando si ha a che fare con il discovery. Durante la progettazione bisognerà considerare che sia il cliente che il servitore possano decidere di muoversi nell'ambiente in cui si trovano. Di seguito si considererà la mobilità del cliente, in quanto maggiormente rilevante ai fini della tesi, ma analoghe considerazioni possono essere fatte per la mobilità del servizio.

La mobilità causa principalmente tre problemi. Primo, il movimento di un device può causare un aggiornamento dell'indirizzo di collegamento interrompendo la comunicazione esistente fra cliente e servitore, di conseguenza si possono quindi avere cali di velocità dei dati sulla rete o disconnessioni temporanee. Un esempio di questo problema è il movimento da un AP ad un altro, la comunicazione fra il dispositivo e il servizio di cui stava usufruendo deve essere mantenuta, mentre l'indirizzo del dispositivo nella rete potrebbe cambiare. Secondo, il device potrebbe avere bisogno di registrarsi con una nuova directory nel sistema middleware, in questo caso si parla di *handoff* tra directories. Infine, il movimento di un device può anch'esso essere il motivo del cambiamento di nome di un cliente in modo che quest'ultimo possa riflettere il cambio di locazione.

Una volta che il cliente ha localizzato il servizio desiderato tramite il discovery del middleware, può iniziare la comunicazione con quel servizio. La comunicazione può avvenire in modo diretto o attraverso il middleware, che funge da mediatore per i messaggi. In entrambi i casi, bisognerà gestire la possibilità di un cambio di indirizzo di rete del cliente, originato da uno spostamento nella rete di quest'ultimo.

Nel momento in cui un cliente si muove da una regione ad un'altra, potrebbe essere necessario un *handoff* dalla precedente directory, dove era registrato, verso un'altra nella stessa regione. Il processo di *handoff* include due operazioni fondamentali: muovere le informazioni di registrazione dalla vecchia directory alla nuova e, nel caso in cui il servizio di cui stava usufruendo fosse dipendente dalla località, ristabilire la comunicazione tra cliente e servitore (figura 1.3), alternativamente nel caso in cui il servizio fosse indipendente dalla località, il middleware può fornire un nuovo servizio dello stesso tipo, nella località corrente. Il quantitativo di traffico generato dall'*handoff* è determinato dalla granularità delle regioni, dal numero di servizi con il cui il cliente comunica e dallo schema di mobilità del cliente.

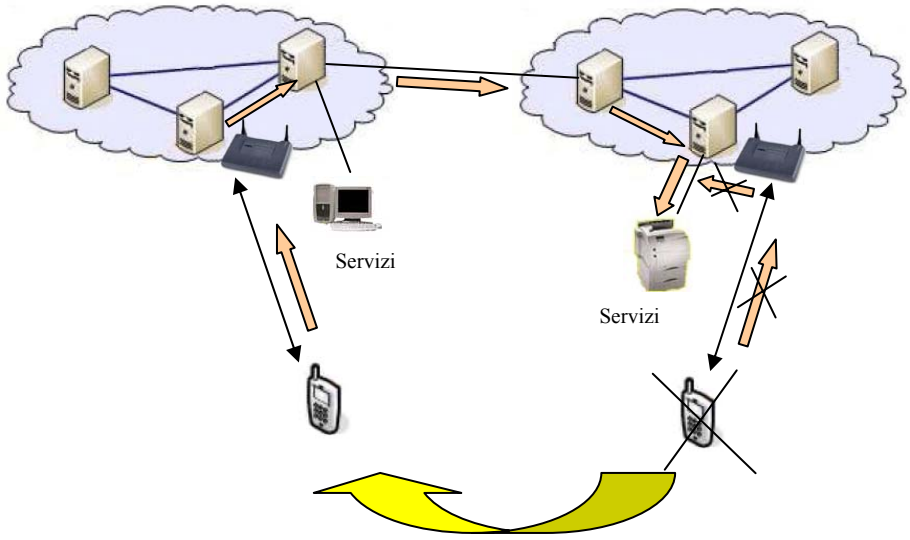


Figura 1.3: Movimento di clienti e comunicazione con i servizi.

È utile distinguere la mobilità fisica e la mobilità di rete, che rappresentano due problemi ortogonali. Un cliente che si muove fisicamente, può o meno cambiare il suo indirizzo di rete. Per molti protocolli di discovery un cliente non si è mosso finché è ancora nella sottorete. In altre parole, questi protocolli definiscono come regione una sottorete, mentre altri protocolli definiscono regione una zona geometrica che non ha nulla a che vedere con i limiti della rete.

In sintesi, la mobilità può causare cambiamenti nel sistema. Il middleware mobile può scegliere di notificare i movimenti del cliente solo al servitore e lasciare il trattamento ai servitori stessi. Oppure il middleware può scegliere di trattare la mobilità da sé e nascondere i cambiamenti completamente ai servitori. Il primo approccio cresce la complessità del servitore, mentre il secondo approccio accresce la complessità del sistema middleware. Una progettazione che segue il secondo approccio ad ogni modo, necessita anche di maggiore prudenza per quando riguarda le assunzioni fatte per i servitori, in quanto vengono prese alcune decisioni al di fuori dei servitori stessi. Per esempio se il middleware ricevesse un messaggio da un servitore mentre il cliente specificato si sta muovendo, il middleware potrebbe memorizzare in un buffer il messaggio per un determinato tempo, e allo scadere, dichiarare fallimento oppure potrebbe instradarlo non appena il cliente diventa disponibile, però queste decisioni vengono prese senza conoscere il tipo di utilizzo del servizio, quindi ad esempio un servizio che fornisce la temperatura della stanza in cui si trova il cliente perde significato quando il cliente si muove in un'altra località.

1.2.1.2 Scalabilità

Se noi consideriamo il rintracciamento e il monitoraggio di servizi che possono muoversi in una vasta area, la scalabilità è un inevitabile problema di progettazione per il middleware di discovery e naming.

Considerando lo scenario visto nel paragrafo 1.2, si può immaginare che migliaia di clienti mobili possano richiedere al middleware di trovare la locazione di servizi e il middleware possa anche aver bisogno gestire le registrazioni di migliaia di devices di servizio, come display, stampanti e sensori nell'ambiente. L'approccio generale è di partizionare lo spazio di nomi in diversi sottospazi, ognuno dei quali è gestito da una directory. Ci sono due ordini di problemi che devono essere analizzati, come partizionare lo spazio di nomi, e come instradare le registrazioni di servizi e le richieste di clienti alla corrispondente directory. Un approccio naturale è di dividere l'intera area in regioni geometriche e inserire una directory dedicata per ogni regione, in genere per servizi che si possono identificare con una locazione fisica. Ogni directory è responsabile per la registrazione dei nomi di tutti i servizi nella propria regione, questa

struttura è molto simile al sistema di nomi DNS. Vale a dire, se una richiesta di una stampante in un edificio B arriva nella directory di un edificio A, la richiesta viene rispedita verso la directory superiore all'edificio A, di cui rappresenta la directory radice e da li viene instradata alla directory dell'edificio B. Questa struttura gerarchica sfrutta la località della registrazione e richiede di partizionare l'area di lavoro, assumendo che le richieste di una regione siano maggiormente incentrate nella ricerca di servizi nella regione stessa. Le richieste che coinvolgono altre regioni possono essere ulteriormente ridotte attraverso l'utilizzo di una cache.

Invece di raggruppare i servizi in base alla loro locazione corrente, è anche possibile raggrupparli usando altre proprietà. Per esempio si può dividere lo spazio dei nomi in sottospazi, ognuno dei quali contenenti una singola directory per i nomi del corrispondente sottospazio. Ogni nome è poi mappato in uno o più di questi sottospazi ottenendo così una certa ridondanza.

In sintesi, il middleware di discovery e naming necessita di un partizionamento dello spazio di nomi in un insieme di directory distribuite per riuscire a processare su larga scala le registrazioni di servizi e le richieste dei clienti. Aumentando il numero di directory, il middleware di discovery e naming scala verso l'alto gestendo più servizi e clienti. Il compromesso per la crescita di scalabilità generalmente è un aumento della latenza per l'elaborazione di una richiesta, in quanto le registrazioni e le richieste hanno bisogno di essere instradate alla corrispondente directory, la quale può essere distante diversi salti dal punto in cui queste vengono inviate.

1.2.1.3 Consapevolezza di contesto

In scenari di *pervasive computing*, il contesto in cui si opera assume un ruolo centrale perché in genere la registrazione di un servizio e la richiesta di un cliente sono *context-dependent*, cioè dipendono dal contesto in cui si trovano. Questa dipendenza si riflette anche nel servizio di discovery, che dovrà essere anch'esso consapevole del contesto in cui si trova, cioè *context-aware*. Vale a dire, il middleware deve anche supportare il monitoraggio del contesto per adattare dinamicamente il processo di discovery e di naming, evitando di lasciare questo peso ai servizi e ai clienti. I clienti mobili devono riuscire a trovare e usare servizi basati sul contesto corrente. Il contesto è

la circostanza nella quale una applicazione esegue e potrebbe includere lo stato fisico (come il rumore e il livello delle luci), lo stato computazionale (come l'ampiezza e la latenza di una rete), e lo stato utente (come la locazione e i task correnti). Per esempio, il middleware deve essere in grado di restituire un appropriato servizio di locazione che meglio risolve la posizione corrente del dispositivo che si sta muovendo all'interno di una determinata area. Considerando lo scenario nel quale un utente ricerca servizi per le cui caratteristiche sia necessaria la vicinanza al dispositivo che fornisce il dispositivo stesso, come ad esempio display o stampanti, possiamo distinguere una richiesta di tipo *persistent query* ossia di tipo persistente, la quale è continuamente valutata nonostante lo spazio dei nomi cambi, e una *context-sensitive query* ossia dipendente dal contesto, la quale cambia se stessa in accordo con il contesto. Alternativamente, il nome del display e della stampante potrebbero cambiare in base alla locazione corrente, che dovrebbe essere aggiornata quando il devices si muove, questo richiede il cosiddetto *context-sensitive name* che cambia se stesso in base al contesto.

Questi scenari pongono diversi requisiti nel servizio di discovery e naming. Deve essere flessibile, così i nomi possano caratterizzare la risorsa e in questo modo le richieste possono esprimere le caratteristiche desiderate; devono essere scalabili, per trattare molti nomi e richieste; devono essere veloci, per supportare frequenti richieste e aggiornamenti di nomi; e deve essere in grado di reagire con prontezza, per notificare le applicazioni sui cambiamenti nel caso di richieste di tipo persistente. Questi scenari, comunque, pongono dei requisiti anche a clienti e servitori. I servizi devono attivamente tenere traccia del loro contesto, in modo che possano aggiornare il loro nome. I clienti devono anche loro tenere traccia del loro contesto, in modo che possano aggiornare le loro richieste persistenti. La raccolta e l'elaborazione delle informazioni di contesto rappresenta un problema non banale e potrebbe richiedere capacità computazionali maggiori di quelle disponibili nei devices mobili connessi ad una rete a banda limitata. Di conseguenza è desiderabile che il supporto *context sensitive*, sia per la registrazione dei nomi, sia per le richieste, venga gestito dall'infrastruttura in esecuzione sulla rete fissa. Il nome *context sensitive* registrato da un servizio e una richiesta *context sensitive* eseguita da un cliente, specificano come dovrebbero essere aggiornati in base al contesto. Il middleware è quindi responsabile di tener traccia delle sorgenti di dati di contesto ed eseguire tutti i calcoli necessari per aggiornare in modo opportuno nomi e richieste. Ogni volta che i nomi e le richieste sono aggiornate, le richieste devono anche

essere rivalutate per determinare se la risposta alla richiesta è cambiata. Attraverso lo sgravo di questi compiti dal servitore e dai clienti, il middleware di naming e discovery migliora le sue performance e facilita lo sviluppo sia dei clienti che dei servitori.

1.2.2 Sfide per la realizzazione di protocolli di discovery dei servizi in ambienti distribuiti

Gli ambienti di computazione distribuiti impongono nuove restrizioni che devono essere prese in considerazione quando viene definito un middleware per un servizio di discovery. Queste restrizioni possono essere divise in quelle che riguardano i servizi di discovery (in particolare il protocollo di discovery) e quelle che riguardano i servizi di definizione (come il servizio di descrizione dei servizi).

Dalla letteratura è noto che soluzioni di tipo legacy per servizi di discovery falliscono nel realizzare i requisiti imposti per ambienti pervasivi [SDD02]. Di seguito verranno riassunte alcune dei più importati sfide che devono essere risolte.

- Minimizzare le trasmissioni di rete: una delle principali sorgenti di consumo di potenza sono le trasmissioni di rete [IECWN01]. Questo implica che uno dei più importanti problemi quando si progetta un protocollo di discovery di servizi è minimizzare il numero di trasmissioni, specialmente per i device con risorse limitate.
- Non dipendere dall'infrastruttura fissa e adattarsi ai rapidi cambiamenti dell'ambiente: in una rete molto estesa di calcolatori, molti o tutti i device sono mobili. Nessuna infrastruttura fissa, incluso un servizio di directory, dovrebbe essere necessaria. Ad ogni modo, in certe località, ma non sempre, alcuni device saranno immobili e si troveranno sempre nello stesso posto. Un protocollo per un servizio di discovery deve adattarsi bene a tutte le possibili località.
- Massimizzare la cooperazione tra device: in ambienti saturati con device limitati, la cooperazione permette a questi di eseguire più task anche complessi. Questo concetto applicato ai protocolli per discovery, significa che device con lo stesso sistema devono cooperare per il discovery di servizi, nell'interesse del bene comune.

- Tener conto delle differenze necessarie alle applicazioni: alcune applicazioni potrebbero cercare un dispositivo che offra un determinato servizio all'interno della rete, ad esempio un sensore di temperatura nella stanza, mentre altre applicazioni potrebbero cercare tutti i dispositivi che offrono particolari servizi nella rete, ad esempio un editor di testi che ricerca stampanti e vuole mostrare tutte quelle disponibili all'utente. Tradizionalmente, i protocolli di discovery eseguono questi tipi di ricerche in modo non differenziato.

1.2.3 Problemi da affrontare nelle descrizioni di servizi

La proliferazione di servizi in ambienti di computazione distribuita richiede una descrizione flessibile del meccanismo di servizio, con una complessità tale da catturare ogni dettaglio in ogni servizio, ma non troppo elevata per non appesantire la sua gestione. I principali problemi che devono essere affrontati per la descrizione di servizi in questi ambienti possono essere riassunte in:

- *Semplicità*: molti device sono limitati in termini di potenza di calcolo, memoria e capacità di rete. La semplicità diventa un requisito fondamentale.
- *Flessibilità*: un meccanismo di descrizione di servizio adattato a ambienti distribuiti deve essere in grado di catturare ogni dettaglio in ogni servizio che potrebbe essere presente nell'ambiente. In genere si punta ad un compromesso tra semplicità e flessibilità.
- *Backward* e *forward compatibility*: è importante che un nuovo servizio sia disponibile non solo per le nuove applicazioni, ma anche per le applicazioni legacy, d'altro canto, nuove applicazioni devono essere in grado di usare i servizi legacy.

1.3 Distribuzione di contenuti multimediali basato su proxy

Le architetture proxy-based si stanno sempre più imponendo come scelta architetturale per i servizi pervasivi di prossima generalizzazione. Un proxy è un

elemento intermedio interposto sul percorso tra il server e i suoi clienti. I proxy sono usati per contenere l'ampiezza di banda, riducendo la latenza di accesso e riuscire a gestire eterogeneità di rete e di dispositivi.

Nel caso specifico di computazione mobile e comunicazione wireless, i proxy sono principalmente usati per superare i tre principali problemi di queste reti: la differenza tra il volume di dati e la latenza che c'è tra i collegamenti wired e wireless, la mobilità degli host, e i limiti imposti dagli hosts mobili (MH). In genere le funzioni svolte dai proxy richiedono molta memoria e alti carichi computazionali di conseguenza la loro esecuzione avviene su rete fissa.

Nella maggioranza dei casi, i proxy fungono da traslatori di protocollo, cache e contenitori di adattatori per clienti con reti o dispositivi limitati e sono posizionati sopra, o vicino al bordo tra la rete wired e quella wireless, come nei wireless AP, chiamati anche Base Station o Mobility Support Station. Oltre a queste funzioni tradizionali, i proxy possono svolgere un ampio ventaglio di altre operazioni complesse per conto dei clienti mobili, come l'*handoff*, controllo di sessione o consistenza, personalizzazione, autenticazione, discovery e molte altre.

I maggiori vantaggi di usare una architettura basata su proxy per clienti mobili, quando comparata con altri approcci punto-punto, sono le seguenti:

a) tutte le trasformazioni che derivano dalla mobilità e dal wireless, come traslazioni e transcodifiche, possono essere assegnate al proxy e non è necessario che siano trattate dal server, permettendo a server di tipo legacy di essere usati direttamente per accessi mobili;

b) tutta l'elaborazione necessaria per il protocollo e le trasformazioni del contenuto è distribuita agli altri nodi dove è necessaria, evitando un sovraccarico ai server;

c) posizionando i proxy prima, o vicino al nodo con l'interfaccia wireless, abilita un più agile ed accurato monitoraggio della qualità del collegamento wireless, una accurata rilevazione delle disconnessioni dei MH e una migliore selezione dell'adattamento necessario al cliente;

d) possono essere implementate trasformazioni ad ogni livello di comunicazione e sono adattabili su richiesta più facilmente, in accordo con le specifiche capacità del collegamento wireless.

In definitiva si può definire proxy come una entità che intercetta la comunicazione o realizza alcuni servizi per conto di alcuni clienti mobili [ER05].

1.3.1 Classificazione degli approcci basati su proxy

I proxy sono principalmente usati per collegare e facilitare le differenze fra reti e dispositivi e permettono la realizzazione di applicazioni specifiche per l'adattamento, in generale le loro funzioni sono sviluppate in accordo con:

- Le differenze caratteristiche fra una rete wired e wireless da collegare fra loro, come througthput, latenza, affidabilità, probabilità di disconnessioni, etc.
- Le caratteristiche specifiche di host mobili, come: dimensione dello schermo, meccanismi di input/output, capacità di calcolo, dimensione della RAM, persistenza della memoria, alimentazione limitata, etc.
- Il tipo di applicazione e i suoi requisiti specifici, come la velocità del tempo di risposta, il tempo di latenza di rete, l'affidabilità della comunicazione, mobilità o trasparenza della disconnessione, coerenza della cache, etc.

Questi aspetti danno un'idea dell'ampia gamma di adattamento e funzioni di management che possono essere assegnate ad un proxy. Possono trattare problemi di comunicazione tra protocolli, trasmissione di dati e codifica, personalizzazioni di specifici dispositivi, *handoff* e management della mobilità, sicurezza ed autenticazione, ripristino della comunicazione dopo una disconnessione etc.

A dispetto dell'ampia diversità di architetture basate su proxy e proposte, si possono identificare due forme ortogonali di classificazione e comparazione di tutti gli approcci basati su proxy. La prima dimensione prende in considerazione alcune caratteristiche generali di una architettura basata su proxy, mentre la seconda dimensione si concentra su i compiti, ossia le funzionalità assegnate ai proxy. Queste due classificazioni verranno viste nel prossimi paragrafi.

1.3.1.1 Classificazione basata sull'architettura

L'architettura a proxy può essere classificata in base ai seguenti punti di vista: livelli software, posizionamento e distribuzione, singolo/multiplo protocollo e comunicazione ed estensibilità, questi aspetti verranno analizzati singolarmente di seguito.

1.3.1.1.1 Livelli software

Da quando i proxy possono essere usati per trattare l'adattamento e la personalizzazione di vari livelli software, è possibile evidenziare una classificazione basata su tre generici livelli: livello comunicazione, livello middleware, livello applicativo.

Nel livello di comunicazione i proxy sono incaricati di trattare tutti i problemi relativi ai protocolli di comunicazione e astrazione. Il principale obiettivo è creare la mobilità del dispositivo e l'uso di collegamenti wireless trasparenti a livelli software più alti. Tipiche realizzazioni a questo livello sono protocolli traslatori od ottimizzatori per wired-wireless, con buffering, con handoff management etc.

Nel livello middleware, i proxy eseguono operazioni generali, non realizzate per una specifica applicazione e neppure per uno specifico protocollo. Esempi di questo solo l'adattamento dei contenuti, il management della consistenza della cache, il discovery di servizi o risorse, etc.

Nel livello applicativo alcune architetture basate su proxy sono incentrate su uno specifico tipo di applicazione come il *Web-browsing*, accesso a database, condivisione dati P2P, etc. In questo caso, i proxy eseguono compiti prestabiliti da specifici requisiti e funzioni di una applicazione. A titolo di esempio, si può analizzare la differenza tra il caching nel Web e nelle applicazioni di database, il primo tratta oggetti eterogenei e essenzialmente punta a ridurre il tempo di risposta, mentre il secondo di solito tratta dati omogenei, ma richiede il management della consistenza della cache.

1.3.1.1.2 Posizionamento e distribuzione

Per quanto concerne il posizionamento di proxy, in genere si adotta la classificazione di Pitoura e Samaras [PS98] per le architetture basate su proxy, la quale definisce le seguenti strutture principali: un proxy esegue solo su un nodo stazionario della rete (lato server); un proxy solo sul nodo mobile (lato cliente); un paio di proxy, uno esegue su un host stazionario e l'altro su un host mobile; e un proxy che può muoversi tra un nodo stazionario e il device mobile o tra nodi (proxy migratore o

agente). Mentre la maggioranza dei sistemi usano sia un proxy lato server o coppie di proxy, ci sono anche esempi di proxy puramente lato cliente. Quando si tratta la mobilità del cliente il proxy migratore assume un ruolo fondamentale, in quanto è questo proxy che segue i movimenti del cliente fornendogli un supporto alla mobilità e garantendo la continuità della sessione in corso. È necessario quindi che un proxy migratore sia in grado di adattarsi all'ambiente in cui il cliente si muoverà predisponendo gli elementi necessari a garantire tale continuità nella sessione.

Un altro aspetto da evidenziare è la distribuzione nell'architettura dei proxy con specifiche funzioni di adattamento e management. Possono essere centralizzati, quando tutte le funzionalità sono racchiuse in un unico proxy. Oppure sono decentralizzati, quando il sistema consiste di numerosi proxy cooperanti, dove ognuno è responsabile per un sottoinsieme delle funzioni.

1.3.1.1.3 Singolo e multi protocollo

Le architetture proxy ricadono in due gruppi a seconda del numero di protocolli di comunicazione che supportano. La maggioranza dei sistemi trattano un singolo protocollo , come TCP o HTTP, e supportano specifici adattatori di questi protocolli, puntando a colmare il gap fra reti wired e wireless. Comunque, ci sono anche altre architetture basate su proxy le quali adottano un approccio a multi protocollo, nel quale il proxy supporta la traslazione wire-wireless usando diversi protocolli (come UDP, SMTP, SMS) e sono in grado di cambiare dinamicamente tra questi protocolli, per inviare i dati verso l'utente indipendentemente da quale rete, ad esempio wireless o cellulare, esso sia direttamente connesso.

1.3.1.1.4 Comunicazione

Questo aspetto caratterizza l'architettura basata su proxy in base al modo con cui il proxy comunica con il cliente, il server e gli altri proxy.

Essenzialmente, un proxy può comunicare con entrambi gli *endpoint*, cioè il server e il cliente, in due modi: in modo sincrono, il proxy realizza una funzione di adattamento

e risponde al cliente in seguito ad una esplicita richiesta di quest'ultimo. Nel modo asincrono, il proxy esegue un compito a lungo termine per conto dell'utente in base alle sue preferenze e invia al cliente una notifica quando il lavoro è concluso. Il modo asincrono è comune quando i proxy svolgono il ruolo di agenti utente, cioè ricercano, collezionano e aggregano informazioni per conto dell'utente.

Alcune architetture supportano anche una comunicazione tra proxy, usata in genere per scopi di sessione e gestione dell'handoff. In base a questo la comunicazione può essere diretta o indiretta. Nel primo modo un proxy conosce, per esempio tramite il suo cliente, con quale altro proxy deve interagire. Nel secondo modo, il server (o un altro proxy) funge da *router* dei messaggi scambiati fra i proxy comunicanti.

1.3.1.1.5 Estensibilità

L'estensibilità dei proxy, cioè la capacità di adattare e personalizzare le loro funzioni, è un ulteriore criterio che differenzia le architetture. In molti sistemi, il proxy ha un predefinito comportamento adattabile, generalmente determinato dallo stato corrente dell'ambiente di esecuzione. Il primo passo per la realizzazione dell'estensibilità consiste nel realizzare un framework generico nel quale i proxy possano essere facilmente adattati alle specifiche necessità di una applicazione o middleware al momento dello sviluppo. Alternativamente si può sviluppare una infrastruttura che supporti il caricamento dinamico di filtri o nuovi moduli che implementano specifiche funzionalità.

1.3.1.2 Classificazione basata sulle funzioni dei proxy

In questo paragrafo verrà introdotta un'altra forma di classificazione, che è basata sulle principali funzioni eseguite dai proxy. Da notare che questa classificazione non separa l'insieme dei sistemi in categorie indipendenti, in quanto in genere le varie categorie di funzioni possono intrecciarsi fra loro. La lista delle funzioni applicabili ad

un proxy è elevata, di conseguenza verranno presentate solo quelle funzioni ritenute più significative e utilizzate.

1.3.1.2.1 Protocollo di traslazione e ottimizzazione

Siccome molti protocolli di comunicazione convenzionali per le reti wired sono di solito non adatti per reti wireless, a causa dei problemi come l'alto numero di errori, il basso throughput, l'alto costo e latenza, la connessione intermittente, etc., una delle funzioni più comuni dei proxy è di trattare questi problemi con un protocollo di traslazione e ottimizzazione.

I protocolli di traslazione wired-wireless sono richiesti in molti livelli del protocollo, ma di seguito si analizzerà solo i problemi del livello di trasporto e superiori.
In aggiunta al piano di traslazione tra i formati di protocollo, i proxy possono anche dover trattare con una lista di altri problemi specifici di comunicazione, come il controllo di flusso, controllo errori, etc., i quali essenzialmente puntano ad ottimizzare il trasferimento di dati sopra il collegamento wireless e riducono il *gap* dalla rete wired. Questo è particolarmente vero per protocolli orientati alla connessione, come TCP, di cui meccanismi per il controllo del flusso non reagiscono correttamente alle disconnessioni, ai blocchi di pacchetti persi o alla fluttuazione del ritardo di round trip.

Una operazione molto comune, assegnata ai proxy è quella di ottimizzare i trasferimenti di dati di un protocollo convenzionale sopra la rete wireless. L'ottimizzazione dei protocolli ha essenzialmente due obiettivi: ottenere un miglior sfruttamento della banda utilizzata e fornire un più piccolo ritardo di round trip. Le tecniche usuali di ottimizzazioni includono il caching dei dati, il *multiplexing* delle connessioni, una compressione dell'*header* e del payload, un controllo del flusso adattabile e una riduzione del volume dati.

Ovviamente ci sono molte altri tipi di tecniche di ottimizzazione della comunicazione, come quelle relative alle trasmissioni multimediali e le presentazioni. Per i contenuti multimediali sono state sviluppate numerose tecniche di codifica e scalabilità con condizioni di rete eterogenee e variabili.

1.3.1.2.2 Adattamento di contenuti

Mentre protocolli di traslazione si trattano con specifici protocolli di adattamento e ottimizzazione, l'adattamento di contenuti è principalmente un protocollo indipendente e punta a trasformare il payload per ottimizzare la trasmissione e la presentazione sul dispositivo mobile. Lo specifico tipo di adattamento usato è principalmente determinato tramite i requisiti dell'applicazione, la quale può considerare i seguenti problemi: qualità del collegamento wireless, le caratteristiche del dispositivo come la sua potenza computazionale (CPU, memoria) e le capacità di uscita (dimensione dello schermo, numero di colori) e i protocolli supportati.

Ci sono una ampio ventaglio di proposte di adattamento dei contenuti per differenti tipi di dati, di seguito verranno esposte quelle più generali che includono tecniche come la distillazione di dati o raffinamento, *summarization*, filtraggio intelligente e transcodifica.

La distillazione è una tecnica di compressione ad alta perdita, in tempo reale e specifica per dati, che tenta di eliminare la ridondanza o le informazioni non necessarie, mentre conserva il principale contenuto semantico dei dati. La distillazione è di conseguenza un termine generale per diverse forme di compressione dei dati, le quali possono o meno essere basate su codifiche standard. Un esempio di distillazione potrebbe essere una trasformazione dove le immagini sono ridotte nelle due dimensioni al fine di ridurre la loro dimensione totale, quindi riducendo anche la loro rappresentazione binaria. Un altro esempio di distillazione può essere la riduzione della profondità di colore. La rappresentazione risultante, nonostante risulti impoverita in risoluzione e colore rispetto all'originale, è ancora riconoscibile dall'utente ed è quindi utilizzabile. Alternativamente, l'utente può essere interessato al contenuto di alta precisione di alcune parti di un dato originale, per esempio attraverso lo zoom in una sezione di una immagine grafica. Il raffinamento è usato per riferire determinate parti di un documento in qualità originale. Di fatto è possibile definire uno spazio di distillazione e raffinamento per ogni tipo di dato, dove distillazione e raffinamento possono essere applicati ortogonalmente ai dati in modo da ridurre la dimensione del file binario.

Con il termine *summarization* si intende una sorta di compressione con perdita dove solo una specifica parte dei dati originali sono selezionati per la presentazione, puntando alla minima perdita di informazione. Il tipo di dati maggiormente utilizzati per dispositivi mobili e wireless sono testo e video. La *summarization* consiste nel catturare dai un file video o di testo le parti esenziali, tralasciando quelle che non sono ritenute importati al fine di preservare il contenuto del messaggio originale.

Il filtraggio intelligente è usualmente definito come un meccanismo per trasformare, ridurre o ritardare i dati spediti attraverso l'applicazione di filtri sul percorso dei dati, a seconda della rete o delle condizioni del dispositivo di destinazione.

La transcodifica è un processo generale di trasformazione del formato di rappresentazione del contenuto: i dati possono essere filtrati, trasformati, convertiti o riformattati per renderli accessibili ad una ampia varietà di dispositivi. La transcodifica è comunemente usata per la conversione di formati video (es. quicktime verso MPEG) o l'adattamento di documenti HTML e file grafici per a dispositivi mobili limitati (es. HTML verso WAP). È spesso usato quando le caratteristiche del terminale impediscono al contenuto di essere rappresentato nel suo formato originale.

1.3.1.2.3 Caching e consistenza

Il caching dei dati vicini sull'host mobile è una operazione molto comune assegnata ai proxy. Gli obiettivi principali e comuni di caching sono la riduzione del traffico verso e da la sorgente server, limitare la latenza percepita dall'utente, conservare la banda wireless e la batteria del dispositivo mobile, in oltre trattare le disconnessone del cliente.

Mentre per i primi due obiettivi, riduzione del traffico e della latenza, può essere sufficiente fare caching di dati in un nodo nel bordo della rete wired, per gli altri obiettivi è necessario un caching dal lato cliente, sull'host mobile.

In principio, il caching lato server per hosts mobili non differivano significativamente da un proxy basato su cache tradizionale per un accesso alla rete wired. Tuttavia, nei dispositivi mobili la differenza principale è che l'accesso avviene attraverso un ampio numero di dispositivi mobili (dai portatili ai cellulari) e attraverso collegamenti wireless con differenti capacità. Per far fronte a questa diversità, le

informazioni dei fornitori hanno iniziato a memorizzare i loro contenuti in differenti formati conformi all'originale, scegliendo il formato che meglio si adatta alle esigenze del cliente e in accordo con le sue preferenze. Questa pratica ha delle implicazioni sulla caching, poiché ogni richiesta è trattata indipendentemente, diversi oggetti possono essere messi in cache sul proxy nello stesso momento con diversi formati, sprecando molto spazio di memorizzazione sul proxy. Per risolvere questo problema diversi studi hanno proposto la combinazione di una transcodifica attiva e dinamica e una cache adattabile nel proxy, così da transcodificare i contenuti in vari formati vicini al cliente.

Per molti protocolli, il caching è anche una caratteristica chiave usata per ottimizzare i trasferimenti di dati su collegamenti con basso throughput, come le reti wireless. D'altra parte, il caching lato cliente punta ad abilitare alcune limitate forme di accesso ai dati, nelle quali il dispositivo non è sempre connesso. Il principale problema è trattare le disconnessioni involontarie e garantire la consistenza degli oggetti nella cache, specialmente quando gli oggetti in cache vengono modificati dai clienti e più di un cliente può avere in cache gli stessi oggetti oppure la copia originale sul server può essere modificata da altri utenti.

1.3.1.2.4 Management di sessione

Molte applicazioni sono basate sul concetto di sessione, che in generale consiste in un insieme o sequenza, di azioni coerenti svolte dall'utente. Sebbene una sessione possa differire molto da un servizio ad un altro, tutte queste si basano sul concetto di stato della sessione. In un ambiente di computazione wireless e mobile, il management della sessione riguarda il mantenimento di uno stato di sessione di una applicazione o servizio a dispetto di una disconnessione e migrazione dell'utente. Si deve considerare che, in questo contesto, una migrazione può avere diversi significati. Nella forma più semplice, un utente mantiene il suo dispositivo e si riconnette ad un differente AP nella stessa rete, o in una differente rete, cioè abbiamo una roaming mobility e si parla di utente mobile. La fornitura di un servizio ad un dispositivo mobile comporta in generale anche un adattamento dei contenuti, per permettere anche a dispositivi limitati di ricevere e presentare il servizio in base alle capacità disponibili sul terminale. Il lavoro svolto integra in questo scenario la possibilità di ricercare, all'interno dell'ambiente, un

sistema di visualizzazione alternativo con caratteristiche migliori attraverso l'utilizzo del discovery. Una più complessa migrazione si ha quando un utente cambia dispositivo, ma vuole continuare senza interruzioni ad usare lo stesso servizio da un nuovo dispositivo, questa viene detta nomadic mobility e si parla di utente nomadico. In questo caso, lo stato di sessione deve non solo essere trasferito, ma molto probabilmente deve essere adattato al nuovo protocollo di comunicazione e alle caratteristiche del nuovo dispositivo.

1.3.1.2.5 Management handoff

Fra tutti i vantaggi offerti da una rete wireless, la mobilità dell'utente è quella che apporta più benefici dato che abilita l'utente ad accedere ad informazioni da differenti locazioni, anche quando è in movimento. Comunque per riuscire a supportare questo, le reti mobili devono fornire un meccanismo per la gestione della mobilità detto management handoff. L'handoff, o handover, si ha quando un utente, precedentemente connesso a qualche rete, si riconnette alla stessa o ad una nuova rete. In entrambi i casi l'handoff deve dare continuità, ossia nessun dato deve essere perso. Il management handoff è responsabile principalmente di due compiti: aggiornare la locazione dell'host mobile, per assicurare che possa essere raggiunto, e trasferire lo stato della sessione dalla vecchia rete alla nuova. Quindi il management handoff si occupa di offrire una trasparenza alla mobilità alle applicazioni.

1.3.1.2.6 Discovery e auto configurazione

La dinamicità degli ambienti mobili di computazione impone requisiti più forti per i servizi di discovery che, tradizionalmente, sono distribuiti nell'ambiente di esecuzione. Per esempio, i servizi di discovery dovrebbero trattare i cambiamenti nella disponibilità di dispositivi o servizi e essere in grado di scegliere il servizio che meglio si adatta per ogni cliente, in accordo con il suo contesto.

I sistemi basati su proxy possono venire incontro a superare alcuni problemi nascondendo le eterogeneità e la dinamicità della rete, così pure riducendo la

complessità dei meccanismi di controllo per gestire tale dinamicità. Accedendo al servizio tramite un proxy, al posto di interagire direttamente con una particolare instanza dei servizi di discovery, può rimuovere dal cliente la necessità di scegliere il miglior servizio adatto alle sue esigenze e la necessità di riconfigurarlo quando il contesto di esecuzione cambia.

1.4 Scenario applicativo

Si consideri il seguente scenario: un utente dotato di un terminale mobile in grado di visualizzare contenuti multimediali di tipo video richiede la visualizzazione di un contenuto multimediale (ad esempio un filmato o un registrazione audio) presente su un server predisposto a questo scopo. I terminali mobili dispongono in genere di risorse molto limitate, di conseguenza il contenuto multimediale potrebbe non essere visualizzabile direttamente da tale dispositivo. Diventa quindi necessario trasformare il contenuto multimediale in modo tale che venga incontro alle caratteristiche fisiche dei terminali richiedenti, questa operazione viene denominata adattamento dei contenuti. Il contenuto multimediale così adattato viene inviato come un flusso di dati attraverso la rete wireless, permettendo così all'utente di muoversi durante la visione. Nel caso in cui questa eventualità dovesse verificarsi, l'infrastruttura proposta deve essere in grado di reagire e predisporsi affinché non ci siano interruzioni nel flusso che disturbino la visione. Si ipotizzi che l'utente arrivi in una zona nella quale sia presente un servizio di visualizzazione, cioè un particolare terminale fisso il cui scopo sia fornire uno schermo alternativo per la visualizzazione, e che questo presenti delle caratteristiche comparabili con le preferenze specificate dall'utente in un precedente momento. A questo punto, l'infrastruttura deve essere in grado, analizzando la zona in cui si trova il cliente, di rilevare questo terminale interagendo con quello che viene denominato servizio di discovery, il cui scopo è mantenere informazioni sui servizi presenti in una determinata zona. Rilevato questo terminale l'infrastruttura deve essere in grado di configurarlo per rendere possibile lo spostamento della sessione dal terminale mobile a questo terminale fisso. Questa particolare operazione deve essere eseguita in maniera del tutto trasparente all'utente e senza un suo intervento manuale, figura 1.4.

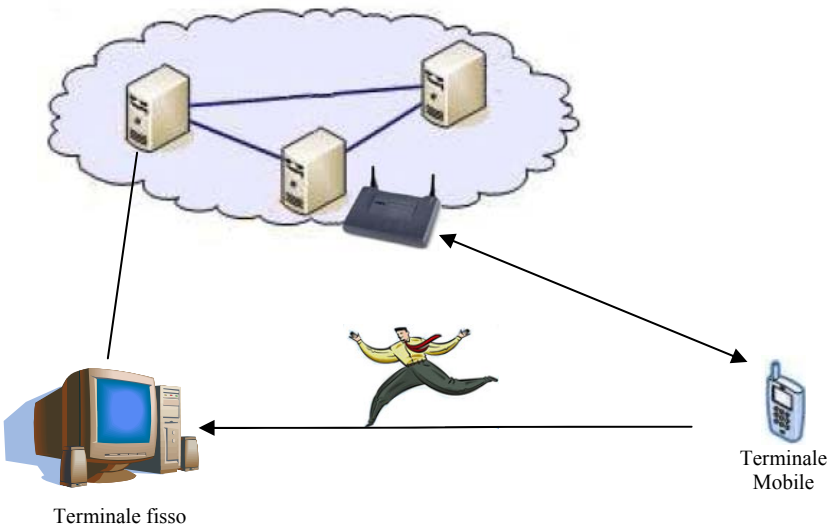


Figura 1.4: Spostamento della sessione.

Con la seguente tesi ci si propone di realizzare quanto esposto sopra, in particolare un sistema di distribuzione di contenuti multimediali verso terminali mobili che non solo fornisca un supporto alla mobilità, ma che sfrutti anche il discovery dei servizi per fornire la possibilità all'utente di spostare la sessione su un terminale fisso in maniera automatica.

1.5 Conclusioni

In questo capitolo sono state introdotte le nozioni che stanno alla base delle reti wireless, del discovery ed è stata presentata una panoramica sulle architetture a proxy. Nella prima parte è stato presentato il concetto di handoff e le differenze principali fra reti cablate e reti senza fili. Per quanto riguarda la parte relativa al discovery, sono state identificate le principali sfide che devono essere superate per la realizzazione un sistema middleware di discovery e naming: mobilità, scalabilità e consapevolezza di contesto e le sfide da superare in ambiente distribuito. Infine sono state presentate le due classificazioni che possono essere identificate per architetture basate su proxy in un sistema mobile, identificando le caratteristiche dell'architettura e le principali categorie di funzioni assegnate a questi intermediari.

Nel prossimo capitolo verranno esposti i lavori di ricerca che si occupano di realizzare una infrastruttura di discovery e come cercano di affrontare le sfide sopra esposte.

CAPITOLO 2

STATO DELL'ARTE

DEI

SISTEMI DI DISCOVERY ED ADATTAMENTO

Il crescente interesse per la mobilità dei dispositivi, ha spinto la comunità scientifica ad inserire negli ambienti di supporto alla mobilità, funzionalità di discovery, per abilitare l'accesso a servizi in qualsiasi luogo e avere una configurazione automatica dei dispositivi. Contemporaneamente, la forte diffusione di dati multimediali a cui si è assistito negli ultimi anni, dovuta principalmente alla nascita di dispositivi elettronici dalle capacità computazionali sempre più performanti, ha creato molto interesse anche verso problematiche di adattamento di contenuti multimediali.

Nel presente capitolo verranno evidenziati i principali studi che si prefiggono il fine ultimo di realizzare un sistema di distribuzione di contenuti multimediali verso sistemi mobili, sfruttando in maniera ottimale le risorse offerte dal dispositivo e minimizzando la configurazione di rete.

2.1 Stato dell'arte

Nei prossimi paragrafi verranno presentati alcuni lavori attualmente in fase di studio e che si propongono di realizzare infrastrutture di supporto al discovery di dispositivi e servizi.

2.1.1 INS/Twine

Sviluppato presso il MIT Laboratory for Computer Science di Cambridge, *INS/Twine* è un middleware le cui intenzioni sono di fornire un approccio scalabile alle risorse di discovery, dove esiste una collaborazione fra pari per la distribuzione di informazioni sulle risorse e per la risoluzione di *query* [INST02].

INS/Twine è stato realizzato basandosi sul Intentional Naming System indicato con INS [INS] e si fonda sull'idea che tutte le risorse sono ugualmente utili.

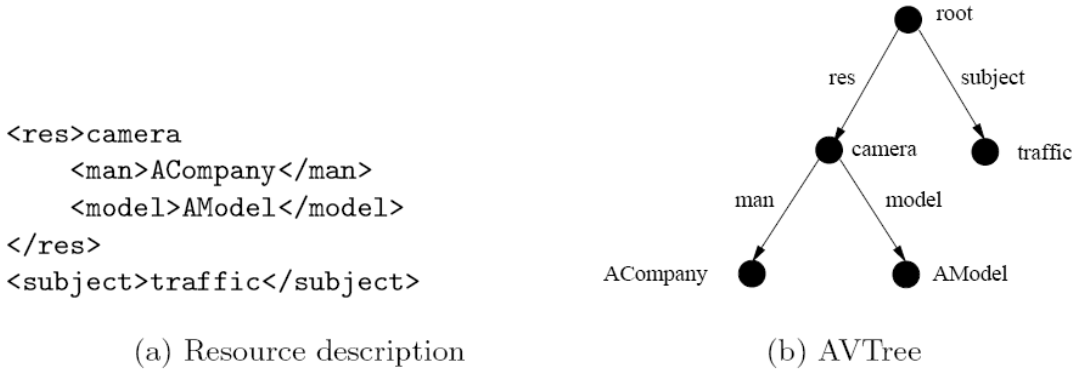


Figura 2.1: Esempio di una semplice descrizione di risorsa e il suo corrispondente AVTree.

Le risorse in *INS/Twine* sono descritte con una gerarchia di coppie di attributi, rappresentati con un qualsiasi linguaggio come XML, INS name-specifiers [1], etc. L'idea è di convertire qualsiasi tipo di descrizione in una forma canonica: un albero di attributi-valori chiamato *AVTree*. In figura 2.1 è mostrato un esempio di una semplice descrizione di risorsa e il suo *AVTree*. Tutte le risorse che possono essere rappresentate con una descrizione a metadati, possono anche essere rappresentate con *AVTree*. Ogni descrizione di risorsa punta ad un *name-record*, che contiene informazioni sulla locazione della rete corrente dove la risorsa è pubblicata, il suo indirizzo IP, il numero di porta, il protocollo di trasporto, e l'applicazione che ne fa uso. In *INS/Twine*, una risorsa coincide con la *query* di un utente se l'*AVTree* della *query* è lo stesso dell'*AVTree* della descrizione originale, con nessuna o più coppie attributi-valori mancanti. Per esempio il dispositivo di figura 2.1 coincide con la *query*:

<res>camera

<man>ACompany</man>

</res>

oppure anche con la *query*:

<res>camera</res>,

questo implica che un importate classe di query che INS/Twine deve supportare sono le *query* parziali, in aggiunta alle *query* complete che specificano la esatta pubblicazione fornita dal servizio e che ne rappresentano la descrizione. Perciò, come le descrizioni delle risorse, le *query* dei clienti sono descritte usando gerarchie di attributi-valori e sono convertite in *AVTree*. *INS/Twine* fornisce un modo per far raggiungere alle *query* il nodo che si occupa della risoluzione meglio attrezzato (*Resolver*) per gestirle, basandosi sugli attributi e valori cercati. I risultati definitivi di un confronto con la *query*, dipendono dall'elaborazione dell'algoritmo locale inserito all'interno di ogni *Resolver*. Esempi di questi algoritmi includono *INS's subtree-matching algorithm* [INS], *UnQL* [UnQL] oppure *XSet query engine for XML* [XSet].

Il confronto con le *query* si basa sul confronto sia dell'attributo che del valore, è quindi possibile una maggiore flessibilità nel confronto separando la stringa del valore in diverse coppie di valori ad esempio: *<model>AModel Camcorder 123</model>* divisa in *<modelw>AModel</modelw>*, *<modelw>Camcorder</modelw>* e *<modelw>123</modelw>*, permettendo in questo modo *query* anche con solo *<modelw>123</modelw>*.

2.1.1.1 Architettura

INS/Twine usa un insieme di *Resolvers* che si organizzano in una rete per instradare le descrizioni delle risorse l'un l'altro, al fine di memorizzarle e di collaborare insieme per la risoluzione delle *query* dei clienti. Ogni *Resolver* conosce un sottoinsieme degli altri *Resolver* sulla rete. Dispositivi e utenti comunicano con i *Revolver* per pubblicare risorse o inviare *query*. Quando una risorsa pubblica periodicamente se stessa attraverso un particolare *Resolver*, è considerata direttamente connessa con quel *Resolver*. Quando un cliente invia una *query* ad un *Resolver*, riceverà sempre la risposta dal quel *Resolver*.

La comunicazione tra applicazioni clienti e *Resolver* avviene sul bordo della rete di *INS/Twine*. La comunicazione tra *Resolver*, invece ha luogo nel cuore della rete.

L'architettura di *INS/Twine* ha tre *layer*, come mostrato in figura 2.2. Il *layer* superiore, il *Resolver*, funge da interfaccia tra l'*AVTree storage* e *query engine*, il primo contiene le descrizioni delle risorse e il secondo implementa l'elaborazione delle *query*, restituendo un insieme di *name-records* corrispondenti alle *query*. Quando un *Resolver* riceve una pubblicazione da una sua applicazione cliente, la memorizza localmente. Il *Resolver* poi divide le pubblicazioni di descrizioni di risorse in parti e le passa al *StrandMapper layer*.

Il *StrandMapper* mappa le parti in uno o più chiavi numeriche a m-bit usando una funzione *hash*. In seguito passa ogni chiave, insieme con la pubblicazione completa (il valore corrispondente della chiave), al *KeyRouter layer*. Infine, data una chiave, il *KeyRouter* determina quale *Resolver* nella rete dovrebbe ricevere il corrispondente valore e instradare le informazioni al corrispondente pari. Perciò, per la distribuzione dei dati, l'approccio utilizzato si riassume nell'inserire la descrizione delle risorse usando ciascun prefisso delle varie parti come chiavi separate.

La suddivisione delle descrizioni delle risorse in parti è una abilità di *INS/Twine* molto importante ai fini della sua scalabilità. Abilita i *Resolver* a specializzarsi nel mantenere le informazioni e rispondere alle *query* che riguardano il sottoinsieme delle risorse. La scelta di una adeguata tavola *hash* distribuita elaborata dal *KeyRouter* è critica per realizzare un altro tasso di successo nel rispondere alle *query* e minimizzare il numero di *Resolver* contattati per ogni *query*.

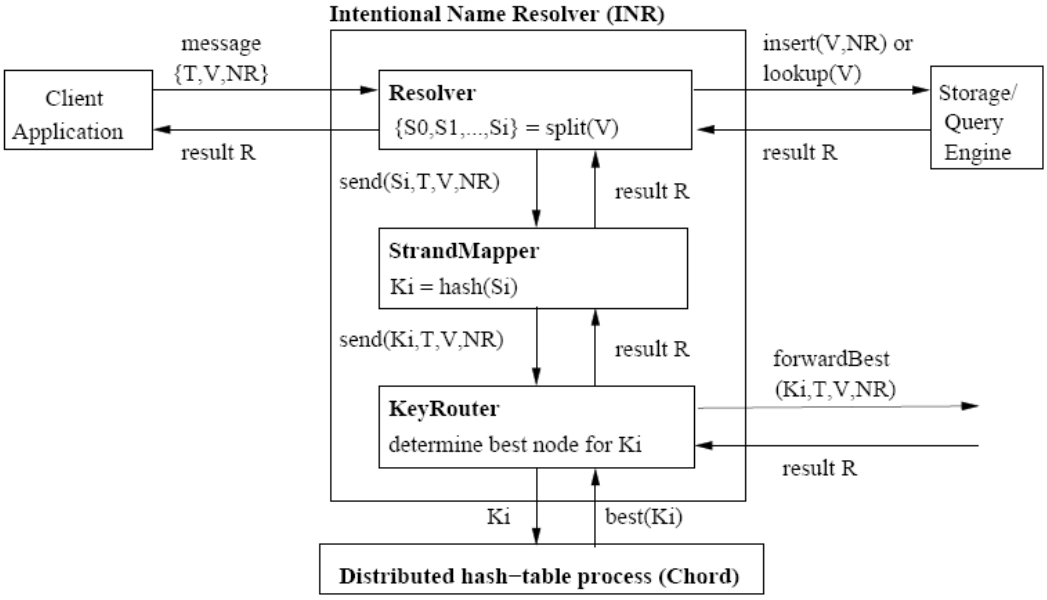


Figura 2.2: INS è composto da tre layer.

Di conseguenza il cuore del *INS/Twine Resolver* è l'algoritmo di *splitting* che suddivide in parti la descrizione di una risorsa. L'obbiettivo dell'algoritmo è spezzare la descrizione in pezzi significativi che il *Resolver* possa adattare al sottoinsieme di descrizioni. Nello stesso tempo, la suddivisione deve preservare la struttura della descrizione e supportare *query* parziali.

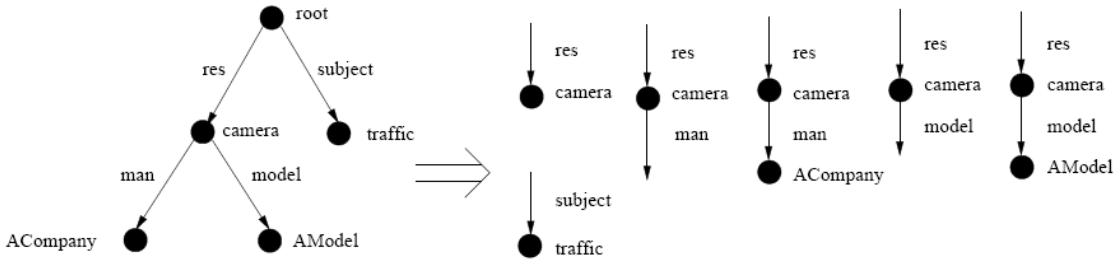


Figura 2.3: Suddivisione della descrizione in parti.

Un semplice metodo di suddivisione consiste nell'estrarre ogni coppia attributo-valore dall'*AVTree* e indipendentemente mapparle in una chiave. Comunque, questo schema potrebbe portare a la perdita di espressività della descrizione gerarchica e potrebbe non permettere la realizzazione di *query* espressive. Per esempio, l'attributo *format* nella sequenza *printer-paper-format* potrebbe diventare indistinguibile da quello in *video-cassette-format*. L'algoritmo *Twine* per lo *splitting* preserva la struttura della descrizione e supporta anche *query* parziali. Estrae ogni sottosequenza di prefissi unici

di attributi e valori dalla descrizione (sia dalle *query,* che dalle pubblicazioni) come illustrato in figura 2.3. Ad esempio:

Sequenza in ingresso: res-camera-man-ACompany

h1 = hash(res-camera)

h2 = hash(res-camera-man)

h3 = hash(res-camera-man-ACompany)

Chiavi in uscita: h1, h2 and h3

Ogni parte estratta dalla descrizione è indipendentemente passata allo *StrandMapper layer*, insieme con la descrizione della risorsa completa o la *query*. Lo *StrandMapper* è responsabile per l'associazione numerica di chiavi con ogni parte. Esegue questo tramite una concatenazione di attributi e valori di parti in una singola stringa e elabora per la stringa una chiave a 128-bit MD5 *hash*. Lo *StrandMapper* passa poi la chiave al *KeyRouter* il quale la usa per determinare quale altri *Resolver* dovrebbero memorizzare le informazioni sulla risorsa, o dovrebbero partecipare nella risoluzione della chiave. La scelta di uno schema appropriato per il *KeyRouter* è molto importate per le performance del sistema.

2.1.1.2 Management delle informazioni

I seguenti obbiettivi guidano il progetto del meccanismo di *management* delle informazioni sulle risorse in *INS/Twine*:

- quando una risorsa si aggiunge alla rete o si muove o modifica la sua descrizione, le informazioni di aggiornamento sono propagate al *Resolver* appropriato immediatamente. Le nuove informazioni sostituiscono quelle vecchie, assicurando che ne la vecchia descrizione e ne la vecchia locazione siano restituite come risultato di una *query*. Mentre le informazioni sulle risorse sono propagate attraverso la rete, è possibile che sia le vecchie che le nuove informazioni sia restituite come risultato di una *query*;
- quando una risorsa abbandona o fallisce, le sue informazioni sono cancellate da tutti i *Resolver*;

- i risultati di *query* non sono influenzati da nuovi *Resolver* aggiunti alla rete, o dal loro fallimento, tramite un indicatore di tolleranza ai guasti configurato tramite un parametro numerico.

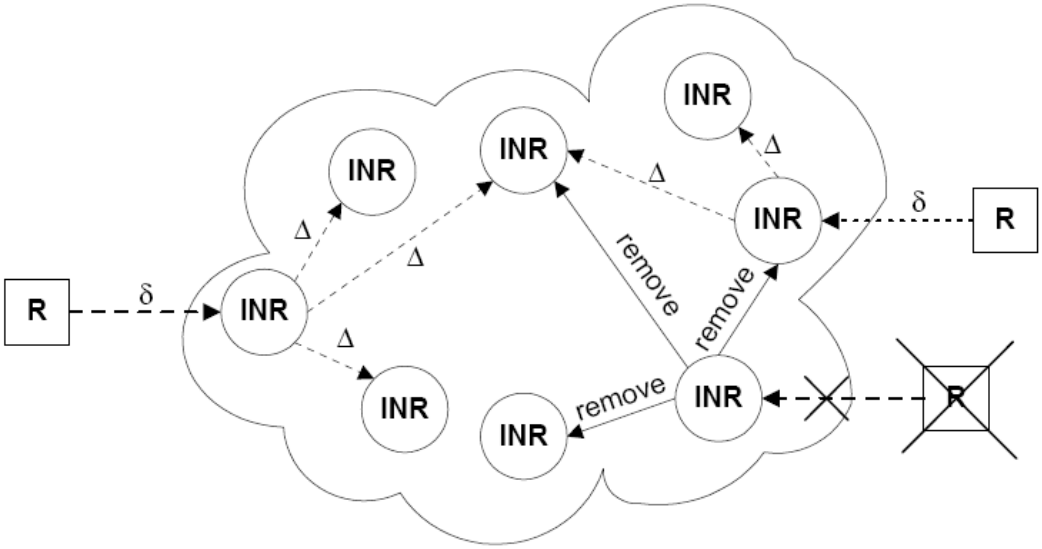


Figura 2.4: Mantenimento della consistenza

Ci sono diversi modi di raggiungere la consistenza esposta sopra, *INS/Twine* adotta uno schema ibrido che combina la semplicità di management di un *soft-state* con le basse richieste di banda di un *hard-state*, figura 2.4 mostra questo approccio. Ogni *Resolver* sul bordo della rete è responsabile per le risorse che comunicano direttamente con lui. I *Resolver* di bordo ricevono aggiornamenti sullo stato delle loro risorse a determinati intervalli di tempo. Questi poi propagano ogni cambiamento agli appropriati *Resolver*. Se la risorsa non contatta più il sistema per un certo periodo di tempo, allora i *Resolver* del bordo inviano un messaggio esplicito di rimozione agli appropriati *Resolver*.

2.1.2 Solar

Solar, sviluppato presso l'Institure for Security Technology Studies del college Dartmouth, è una middleware per il naming e il discovery che supporta nomi e richieste context-sensitive [NamDM].

Il cuore di *Solar* è un insieme di nodi con le medesime funzionalità, chiamate *Planets*, che sono connessi fra loro usando una rete peer-to-peer basata su DHT

[CLK04]. *Solar* impiega un sistema di naming basato su attributi ed ogni *Planets* ospita una directory di nomi. Utilizza un schema di attributi *hashing*, simile a INS/Twine [INST02], per partizionare lo spazio dei nomi in tutti i *Planets*.

È previsto che il contesto sia computato tramite aggregazione di dati da uno o più sensori fisici o virtuali. Questi sensori, come gli altri servizi, si connettono a un *Planets* e registrano il solo nome. Nella figura 2.5 è presentata l'architettura di *Solar*.

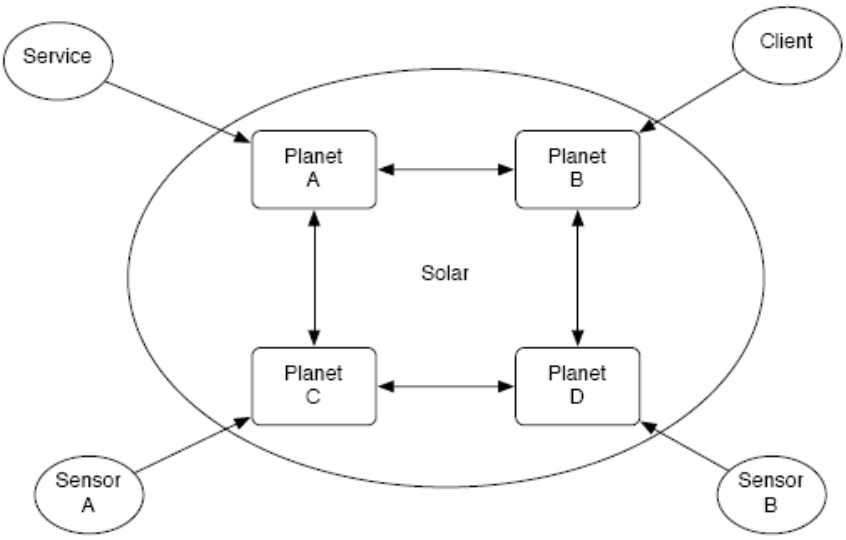


Figura 2.5 Architettura del sistema Solar.

Un servizio si registra con il *Planet* A, il quale instrada il nome del servizio verso gli appropriati *Planets* in base ad un *hashing* di attributi. Si immagini che arrivi una *query context-sensitive* a *Planet* B, cioè una richiesta che sia dipendente dal contesto in cui il richiedente si trovi, per esempio la ricerca di una stampante all'interno dell'edificio:

$campus-locator = [sensor=locator, scope=campus];
$user-locator = @filter("00022dd54817") <- $campus-locator;
query [service=printer, building=$user-locator:building].

La *query context-sesitive* è definita in modo che il valore di alcuni attributi sia in parte computata attivamente da altri sensori. A questo punto, *Planet* B prima definisce un identificatore della zona indicato con *$campus-locator*, che traccia tutti di device all'interno della rete. In seguito definisce un identificatore che traccia solo la posizione di colui che fa la richiesta, tramite un filtro che analizza i dati in uscita da *$campus-*

locator, usando l'indirizzo MAC del dispositivo che esegue la richiesta. Infine la *query* richiede tutte le stampanti nello stesso edificio della locazione corrente dell'utente, dove il valore dell'attributo building equivale all'unico nell'output del identificatore dell'utente.

Planet B tratta le *query context-sensitive* del cliente, e necessita di elaborare il contesto usando i dati di sensori che potrebbero essere connessi ad altri *Planets*, come su C e D. L'insieme dei *Planets* in questo modo serve anche come livello applicativo di routing per i dati dei sensori [CLK04]. Il filtro, accennato precedentemente, è chiamato *operator*. Un *operator* è un componente che processa dati, si occupa quindi di analizzare uno o più stream di dati provenienti da sensori e li ripropone in uscita ad altri. Un nome o una richiesta *context-sensitive* può specificare un grafico di *operators* per indicare la logica di combinazione di dati da sensori multipli [CK03]. *Solar* è responsabile del caricamento e dell'impiego di questi *operators* sui *Planets*.

Dagli esperimenti di valutazione del sistema *Solar* è risultato che un middleware che supporta i nomi e le richieste *context-sensitive* porta a dei vantaggi: riduce la latenza delle richieste e migliora la scalabilità del sistema.

2.1.3 TiniObj

TiniObj è un modello di discovery di servizi sviluppato presso la Graduate School of Frontier Science dell'università di Tokyo. Il modello include una parte realizzata in hardware e una parte software [TinyObj].

Il software è stato realizzato allo scopo di fornire un'interfaccia grafica uniforme, in modo che un utente possa facilmente aggiungere, rimuovere, pubblicare e fare il discovery di nuovi servizi. A differenza di altri sistemi di discovery, *TiniObj* non richiede una conoscenza pregressa di un qualche linguaggio di programmazione. Il modello di discovery utilizza broadcast di piccoli pacchetti su rete wireless, al fine di raggiungere gli utenti che si trovano nelle vicinanze.

Figura 2.6: il dispositivo Buoy

Il dispositivo hardware è denominato *Buoy*, si tratta di un dispositivo wireless che può essere usato come un servizio di discovery indipendente o come un accessorio per telefoni cellulari, come mostrato in figura 2.6, tale dispositivo è basato sul protocollo CSMA/CA MAC.

Il servizio di discovery *TiniObj* coinvolge tre elementi del base:

- *Service matchmaking*: il numero di fornitori di servizi in generale è superiore al numero di servizi effettivamente necessari all'utente. Il sistema prevede un meccanismo per ricevere risultati di ricerca focalizzati sull'obbiettivo della ricerca. Inoltre, il *service matchmaking* fornisce un modo generale di rappresentazione del tipo di servizio. Un utente può aggiungere e rimuovere facilmente servizi dal sistema.
- *User Interaction*: il servizio di discovery è predisposto a supportare due tipi di utenti mobili: un utente che utilizza un telefono cellulare e un utente che fa uso di un dispositivo wireless di discovery. Uno degli obbiettivi è fornire una facile interazione utente-dispositivo per entrambi i tipi di utenti.
- *Broadcast Protocol*: il broadcast è un componente importate del sistema e deve essere quindi molto efficiente. I dati inviati e il consumo di potenza devono essere ridotte al mimino possibile.

Il modello di discovery utilizzato in *TiniObj* è formato da quattro punti fondamentali:

1) inizializzazione dati;

2) scambio di dati in modalità wireless;

3) *service matchmaking*;

4) avvisi di discovery.

Come prima cosa, un utente esegue un inizializzazione dei dati. In generale l'utente può inizializzare due tipi di dati: inserzione del servizio da pubblicare e le preferenze per le caratteristiche di un servizio. Si consideri il caso di un utente che vuole trovare un servizio di tipo bancomat, come si vede in figura 2.7, l'utente specifica delle preferenze, che costituiranno un filtro per selezionare i possibili servizi, come ad esempio solo servizi bancomat che supportano carta di credito VISA con le spese di commissione al massimo di 1,5 $. Nel frattempo, un fornitore di servizi, rende disponibile il suo servizio, definendo un'inserzione del servizio come ad esempio: Servizio di bancomat di Citibank, carte di credito supportate VISA, MasterCard, e spese di commissione di 1,05 $.

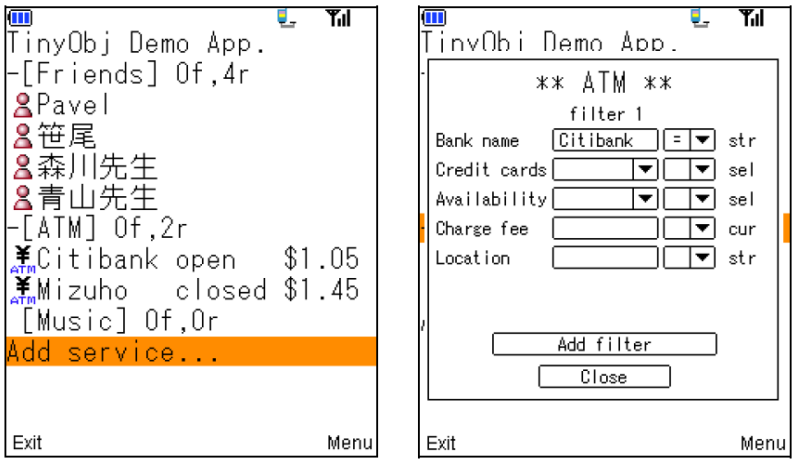


Figura 2.7: esempio di funzionamento

Secondariamente, dopo che i dati sono stati inizializzati, il processo di scambio di dati wireless può avere inizio. Lo scambio di dati wireless è basato sull'invio periodico di dati in broadcast, nonostante questo non è necessario che tutti i partecipanti trasmettano dati di tipo broadcast. I partecipanti possono essere essenzialmente di due tipi: passivi o attivi. I partecipanti attivi sono quelli che trasmettono in broadcast nelle zone limitrofe, mentre i passivi salvano i dati ricevuti da un altro dispositivo wireless senza fare nessun tipo di broadcasting.

Di norma, un inserzionista che intende pubblicizzare il suo servizio e un utente, possono selezionare se utilizzare una strategia di discovery di tipo passivo o attivo. Nello scenario visto in precedenza, la banca potrebbe utilizzare una strategia attiva

perché è interessata a consegnare il servizio all'utente, mentre l'utente potrebbe, invece, essere interessato ad utilizzare una strategia passiva ed inserire un filtro per selezionare i possibili servizi trovati, in questo modo il dispositivo ricevendo dati da tutti i servizi, considererà solo quelli che coincidono con le sue preferenze. Comunque, è possibile eseguire un broadcast non solo di pubblicazione, ma anche di preferenze. In questo caso, un dispositivo inserzionista riceve le preferenze, e le compara con tutti i servizi da lui stesso pubblicati, i quali sono memorizzati nel database del dispositivo stesso. Perciò, sia l'inserzionista del servizio, che l'utente che esegue la ricerca di servizi, possono realizzare il discovery. L'idea di fondo di *TinyObj* è di utilizzare solo trasmissioni broadcast. Di conseguenza, quando un dispositivo wireless trova un servizio, non ha un metodo per rispondere al fornitore del servizi e non può usare lo stesso mezzo di comunicazione. Un utente può accedere al fornitore solo attraverso altri tipi di comunicazione, come e-mail, telefono o web, usando le informazioni di contatto incluse nel pacchetto.

In terzo luogo, dopo che i partecipanti decidono se essere un elemento passivo o attivo, è il componente *service matchmaking* che sottopone a confronto tutti i pacchetti ricevuti tramite broadcast con i dati memorizzati. Quando un dispositivo riceve una pubblicazione, questa viene comparata con i filtri memorizzati nel dispositivo stesso. Analogamente, se un dispositivo riceve delle preferenze, questo verifica se quest'ultime coincidono con la sua pubblicazione, memorizzata sul dispositivo stesso. Il *service matchmaking* fornisce, a tal scopo, una funzione che prende come parametri di ingresso una pubblicazione e le preferenze, e restituisce come risultato di comparazione, vero o falso. Se il valore ritornato è vero allora il *service matchmaking* ha successo e il servizio è stato trovato.

Inoltre, dopo che il *service matchmaking* rileva il servizio che coincide con le preferenze, notifica l'utente con un avviso che può essere un suono, una vibrazione, etc. Dopo che l'utente ha avuto modo di visionare il contenuto del servizio di discovery, può accedere al fornitore del servizio utilizzando le informazioni incluse nel pacchetto dati.

2.1.4 PDP/GSDL

PDP/GSDL è un middleware di discovery per il supporto di interazione spontanee in ambienti pervasivi, sviluppato presso la University Carlos III di Madrid [PDPGDSL].

È costituito da due componenti fondamentali il *Pervasive Discovery Protocol* (PDP) e il *Generic Service Description Language* (GSDL). Il primo rappresenta un protocollo di discovery, mentre il secondo è un linguaggio per la descrizione di servizi.

2.1.4.1 PDP

Uno degli obbiettivi chiave del protocollo PDP è minimizzare il numero di trasmissioni necessarie per il discovery di servizi e limitare quindi il consumo di batterie, specialmente per la maggior parte di device mobili e dalle capacità limitate. Questo è realizzato attraverso l'uso del tempo di disponibilità, un parametro che verrà introdotto in seguito. PDP rende più prioritarie le risposte dei device più limitati, permettendo agli altri di abortire le loro risposte. PDP inoltre elimina la necessità di un server centrale, è quindi un protocollo pienamente distribuito che fonde le caratteristiche delle metodologie *push* e *pull*.

In PDP tutti i messaggi sono broadcast, e tutti i device cooperano attraverso la coordinazione delle loro risposte e condividendo le informazioni nelle loro cache. PDP tiene conto delle differenze necessarie alle diverse applicazioni, e questo permette inoltre di ridurre il consumo di potenza.

Si consideri il seguente scenario applicativo, composto da D device e che ogni device offra S servizi e preveda di rimanere disponibile in rete per T secondi. Questo tempo T è chiamato tempo di disponibilità ed è configurato nel device, in quanto dipende dalle sue caratteristiche di mobilità.

Ogni device ha un *PDP User Agent* (PDP_UA) e un *PDP Service Agent* (PDP_SA). Il PDP_UA è un processo attivo per conto dell'utente, il cui scopo è cercare informazioni riguardo i servizi offerti nella rete. Il PDP_SA è un processo attivo per pubblicizzare un servizio offerto da un dispositivo, nella sua pubblicizzazione include sempre il tempo di disponibilità del dispositivo.

Ogni dispositivo ha una cache contenente una lista dei servizi che ha appreso dalla rete. Ogni elemento della cache ha due campi: la descrizione del servizio e il tempo di scadenza del servizio. Il tempo di scadenza del servizio è il tempo di stima per cui il servizio rimarrà disponibile. Questo tempo è calcolato come il minimo di due valori: il tempo di disponibilità del dispositivo locale e il tempo di disponibilità promesso dal servizio. Le voci sono rimosse dalla cache quando il loro tempo è scaduto.

2.1.4.2 Descrizione del protocollo

PDP ha due messaggi obbligatori: *PDP_Service_request*, che è usato per inviare una richiesta di servizio, e *PDP_servcie_Reply*, che è usato per rispondere a un *PDP_Service_Request*, e per annunciare i servizi disponibili. In aggiunta, PDP ha un messaggio opzionale: *PDP_Service_Deregister*, che è usato per informare che un servizio non è più disponibile.

Quando una applicazione o l'utente finale di un dispositivo necessita di un servizio di un certo tipo, utilizza il suo PDP_UA. Al fine di supportare le necessità delle differenti applicazioni, PDP ha definito due tipi di *query*:

- *One query-one response* (1/1): l'applicazione è interessata a un servizio, ma non in quale device questo servizio è offerto.
- *One query-multiple responses* (1/n): l'applicazione vuole trovare tutti i dispositivi nella rete che offrono il servizio. In questo tipo di domanda, è stato introdotto un tipo speciale di servizio, chiamato *ALL*, allo scopo di permettere ad una applicazione di trovare tutti i servizi disponibili di tutti i tipi di rete.

Entrambi i tipi di *query* usano lo stesso messaggio *PDP_Service_Request*. Un *flag* nell'*header* del messaggio indica se esso è 1/1 o 1/n.

Nel tipo 1/1, il PDP_UA cerca un *service_type* nella lista dei servizi locale e nella cache. Se viene trovato, il PDP_UA fornisce all'applicazione la descrizione del servizio corrispondente, senza nessuna trasmissione di rete. Se non viene trovato, il PDP_UA invia un broadcast con un *PDP_Service_Request* per quel servizio, aspettando un *CONFIG_WAIT_RPLY* per rispondere. Se non arrivano risposte, il PDP_UA risponde

all'applicazione che il servizio non è disponibile della rete. Se alcune risposte arrivano il PDP_UA fornisce all'applicazione la descrizione dei servizi ricevuti. I PDP_UA in tutti i dispositivi sono continuamente in ascolto sulla rete per tutti i tipi di messaggi (richieste e risposte) e aggiornano le loro cache con i servizi contenuti in tali messaggi. Naturalmente, le cache dei device hanno una dimensione limitata, di conseguenza quando un PDP_UA sente uno nuovo annuncio, ma la cache è piena, cancella le voci dei servizi più vicini alla scadenza. Per quanto riguarda il PDP_SA, il suo compito è pubblicizzare i servizi offerti dal dispositivo. Elabora il messaggio *PDP_Service_Request* e genera il corrispondente *PDP_Service_Reply*, se necessario. Allo scopo di minimizzare il numero di trasmissioni il PDP_SA prende in considerazione il tipo di *query* fatta dal PDP_UA remoto. Quando un PDP_SA riceve un *PDP_Service_Request* 1/1, controlla se il servizio richiesto è uno dei suoi servizi locali, in tal caso ad un *PDP_Service_Reply* viene associato un tempo casuale, inversamente proporzionale alla disponibilità di tempo del dispositivo. Durante questo tempo, se un'altra risposta alla stessa *query* PDP da parte di un altro dispositivo, viene rilevata, la risposta è abortita, questo perché il PDP_UA remoto passerà alla applicazione solo il primo servizio e scarterà qualsiasi altro, in questo modo i messaggi di risposta saranno notevolmente ridotti. Se invece il timer scade e non sono partite risposte da altri dispositivi, allora questa viene inviata. L'algoritmo assegna automaticamente il dispositivo fornitore del servizio che è attualmente più scarico e quindi con più opportunità di rispondere alla richiesta. Di conseguenza l'algoritmo da maggiore priorità alla risposte provenienti da dispositivi con una disponibilità stimata più lunga.

Quando un PDP_SA riceve un *PDP_Service_Request* 1/n, controlla se la richiesta di servizio è un dei suoi servizi locali, o se è nella *cache*. In tal caso, è generato un tempo di attesa *random*, inversamente proporzionale al tempo di disponibilità del dispositivo e al numero di servizi conosciuti, durante questo tempo, il PDP_SA analizza i messaggi sulla rete per ogni *PDP_Service_reply* della stessa richiesta. Quando il timer scade, se il PDP_SA conosce altri device aggiuntivi che offrono questo tipo di servizio e che non sono ancora stati annunciati, invia un suo *PDP_Service_Reply*. Perciò, la maggior parte del tempo il device è in grado di offrire il servizio, e più grande è la cache, maggiore è la probabilità di rispondere prima. Tutto questo supponendo che il device con la

maggiore disponibilità di tempo e la più grande cache sia anche quello con il maggior dettaglio di quello che succede nell'ambiente.

In certi casi, un device potrebbe essere a conoscenza di quando sta per essere spento o sta per spostarsi in un'altra rete. In tal caso, il PDP_SA del device invia un *PDP_Service_Deregister*, elencando tutti i suoi servizi locali prima di spegnersi o spostarsi. Quando un altro dispositivo riceve il messaggio, deve rimuovere i servizi elencati nella sua cache. Talvolta, quando un device prova ad accedere ad un servizio elencato nella sua cache, si potrebbe verificare che il servizio sia down, di conseguenza il device può usare il messaggio *PDP_Service_Deregister* per informare il resto della rete che questo servizio non è più disponibile, così tutti i device che ricevono il messaggio possono cancellare la *entry* dalla *cache*.

2.1.4.3 Generic Service Description Language (GSDL).

GSDL fa uso di una gerarchia di servizi descritta in [MR04]. Questa gerarchia definisce servizi generali comuni a tutti gli ambienti delle reti di prossima generazione e nel primo layer, specificando un secondo layer con più servizi specifici che sono condivisi da gruppi o famiglie di device e infine incorpora, in un terzo e successivo layer, i servizi specifici per ogni device (di fatto, i layer 1 e 2 sono layer astratti di cui le funzioni sono implementate in servizi nei layer 3 e successivi). I servizi esportati da ogni device sono inserti in un albero gerarchico nel quale i dettagli specifici del layer 3 sono discendenti da un più generico servizio nel layer 2 e così via. Questo permette alle applicazioni in un ambiente distribuito di usare i servizi in altri device anche se essi non sono famigliari o non hanno conoscenze pregresse dei dettagli nel layer 3 per mezzo di una conoscenza degli antenati nei layer 1 o 2. Questa gerarchia di servizi soddisfa entrambi i requisiti di semplicità e scalabilità specificati nella sessione precedente siccome permettono, da una parte, la definizione di un servizio specifico e, dall'altra, un loro generico uso in device limitati attraverso i servizi noti nei layer 1 e 2. Inoltre, questa definizione di gerarchia di servizi provvede ad una compatibilità precedente e futura di servizi e applicazioni, a meno che i layer 1 e 2 non vengano modificati.

GSDL fa uso di servizi Web come meccanismo di accesso ai servizi, in quanto si adattano alla computazione distribuita. I Web service sono basati su XML e usano un

protocollo di internet come http, la loro semplicità è tale da renderli integrabili in piccoli device. GSDL è definito per essere un linguaggio di descrizione di servizi che è ben integrato con i Servizi Web, di fatto è definito per essere una descrizione complementare dell'interfaccia del servizio.

GSDL definisce una relazione di gerarchia fra servizi fornendo una strada semplice, scalabile e compatibile per definire nuovi servizi. GSDL cattura solo la relazione di gerarchia per la descrizione del servizio e usa il linguaggio WSDL per la descrizione dell'interfaccia.

GSDL è basato su XML ed è specificato usando uno schema XML. I primi elementi nello scema sono graficamente rappresentati in figura 2.8.

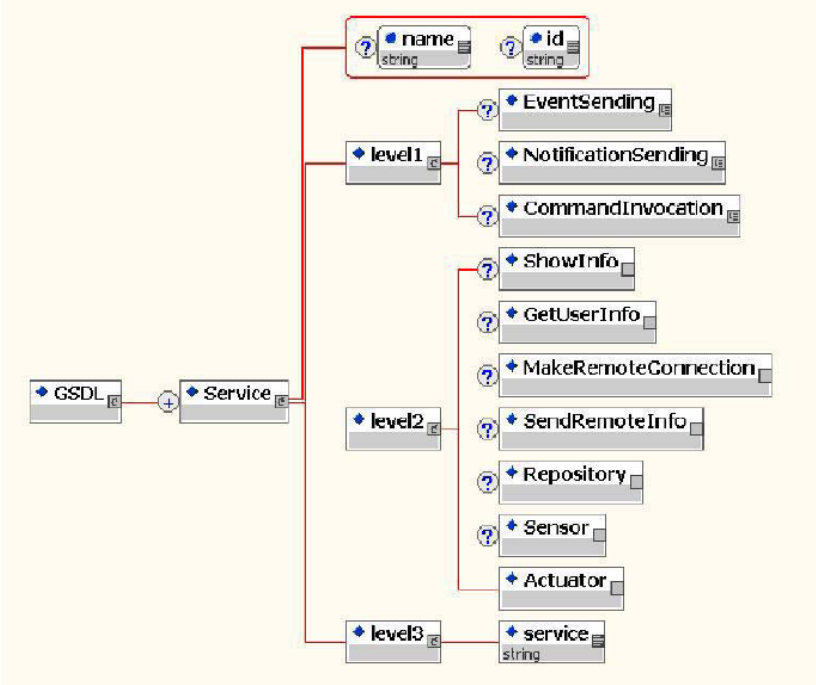


Figura 2.8: Schema GSDL: rappresentazione grafica.

Per ogni servizio specifico, GSDL descrive i servizi di livello 1 e 2 da cui questo specifico servizio eredita. Questi servizi di livello 1 e 2 sono servizi ben noti e rappresentano un generico modo di accedere ad ogni specifico servizio, definiti prendendo in considerazione i diversi scenari di rete. I dettagli specifici dei servizi di livello 3 sono ottenuti specificando l'URL di dove possono essere trovate le descrizioni delle interfacce in WDSL.

Al fine di ridurre la dimensione dei messaggi PDP, non viene inclusa la piena descrizione GSDL dei servizi, al suo posto, per uno specifico servizio, vengono inclusi i

servizi di layer 1 e 2 codificati con 2 byte (ci sono 3 servizi di livello 1 e 7 di livello 2) e l'URL dove il device può ottenere la descrizione del livello 3.

2.1.5 Psachno

Sviluppato presso l'Università School of Computer Science and Software Engineering Monash in Australia, Psachno è un frameworks che intende fornire un'architettura totalmente dinamica dove ogni componente può essere caricato via Web usando delle informazioni di metadati collegati al componente [Psachno]. L'ambiente Psachno è informato sui diversi componenti e identificatori allo scopo di fornire un ambiente di discovery. Un nodo è una singola entità che espone certe risorse. Una risorsa può essere un file, una stampante o qualche generica risorsa disponibile sul nodo. Ogni risorsa è assegnata a una chiave che descrive il suo accesso e livello di protezione.

2.1.5.1 Architettura

Per rendere il sistema generico e interoperabile, tutte le strutture dati fanno uso di un XML Schema Definition (XDS). Questa scelta progettuale è stata fatta per integrare e fornire flessibilità tra differenti *frameworks*. Nel *frameworks* Psachno, ogni tipo può essere serializzato in formato binario e XML. La decisione di supportare entrambi i modelli è stata presa allo scopo di fornire allo stesso tempo sia un modello generico, sia un modello efficiente e veloce.

Nel *frameworks* Psachno i sono quattro essenziali tipi: *Identifier*, *Key*, *DiscoveryItemBase*, *DiscoveryPackage*, *DiscoveryResultPackage*.
Un *Identifier* è un semplice identificatore utilizzato per descrivere l'identità di un elemento all'interno del framework, questo identificatore viene generato con l'algoritmo *SHA1 hash*. Questo garantisce che il valore generato sia totalmente univoco.

Il *DiscoveryItemBase* è costituito di diversi elementi: un identificatore univoco, un nome, una categoria e un proprietario. Un *identificare univoco* è obbligatorio in quanto

elementi del discovery possono essere simili o addirittura avere lo stesso nome. Una *categoria* è un raggruppamento che da la possibilità allo sviluppatore di posizionare elementi in un particolare gruppo comune. Di default, l'ambiente Psachno fornisce tre tipi di discovery:

- *DiscoveryItem*: questo è un parametro extra come la locazione, chiave per definire un elemento da trovare e fornire sicurezza.
- *Mime*: eredita da *DiscoveryItem* e fornisce informazioni extra come il *path* e il tipo di servizio, più attributi relativi solo a ricerche riguardati il tipo *Mime*.
- *Service*: eredita da *DiscoveryItem* e fornisce informazioni extra come il tipo e il proprietario riferiti solo al tipo di servizio.

Questi elementi sono di base e possono essere estesi. Essendo realizzati in XSD, uno sviluppatore può personalizzare ed estendere lo schema attualmente disponibile. Uno schema che deve essere riconosciuto dal sistema deve necessariamente ereditare da *DiscoveryItem* oppure esserne una super classe.

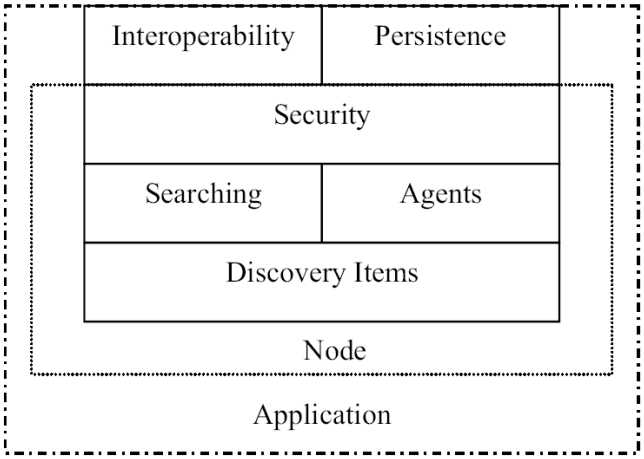


Figure 2.9: Psachno architettura dell'applicazione

La struttura di una applicazione Psachno fornisce due layer, il layer *Application* e il layer *Node* come si vede in figura 2.9. Il layer *Application* fornisce la funzionalità di interoperabilità e persistenza al layer *Node*. Questo approccio di progettazione è stato scelto principalmente in quanto tutti gli elementi nel layer *Node* possano essere persistenti e tutti i messaggi che vengono ricevuti possano essere inviati attraverso questo layer.

L'interoperabilità è fornita attraverso l'utilizzo di ascoltatori interoperabili che forniscono l'abilità di ricevere messaggi in altri protocolli e presentare i loro oggetti

come *DiscoveryItems*. Per questo fine vengono fornite un'interfaccia e una classe astratta per dar la possibilità di realizzare differenti modi di collegamento.

Nel layer *Node* è presente la maggior parte dell'elaborazione e delle funzionalità. Ogni messaggio inviato è elaborato come prima cosa da un verificatore di sicurezza, in seguito inviato al motore e agli agenti di ricerca, che si trovano nel layer inferiore. Il motore e gli agenti di ricerca incapsulano gli elementi di discovery.

Il layer *Node* in PSachno rappresenta l'incapsulamento di una risorsa. Come menzionato precedentemente le risorse possono essere files oppure servizi. Un nodo è molto più di un componente che incapsula una risorse, è necessario che sia in grado di cercare e trovare risorse connesse e includerle nel *Node* stesso. Ogni *Node* è capace di cercare usando diversi motori di ricerca. Ad ogni elemento di discovery è assegnato un attributo di ricerca *SearchAttribute* che specifica quale sia il migliore motore di ricerca da utilizzare. Differenti motori di ricerca possono essere sviluppati per fornire una ricerca ottimizzata per particolari tipi di ricerca. Di default, un motore di ricerca ne ha uno interno generico e può scaricarne uno migliore per aumentare le sue prestazioni. Di default, è presente anche un *CacheAgent*, che permette di avere una memoria sulle ricerche effettuate.

Ogni nodo ascolta su una porta di default, il framework la utilizza per ricevere le richieste di discovery da altri nodi. Nella corrente implementazione, Psachno usa il protocollo *Pastry* [Pastry] siccome offre una layer di trasporto con *context aware*. Questo può essere collegato con altri layer di trasporto come JXTA. Il layer *Node* supporta anche la proprietà *time to live* (TTL) per specificare quanti *Node* un *discovery item* può attraversare prima di diventare ridondante.

Psachno fornisce anche un modello di sicurezza di base. Ogni elemento nel framework Psachno è distribuito con una chiave. Una chiave è un tipo astratto e può essere implementata in molte maniere. Di default, è implementata una chiave *GUID* che permette di usare un generatore di chiavi uniche per l'autenticazione per *discovery items*. Attraverso la default *GUID key*, questa sicurezza può essere aumentata attraverso lo sviluppo di chiavi *hash* o usando un meccanismo di chiave pubblica / chiave privata. Ad ogni chiave è associato un *SecurityAttribute* per definire un motore di sicurezza che è necessario per autenticare una chiave. Questo è simile al *SearchAttribute* dove più motori di ricerca possono essere scaricati per fornire una autenticazione con le varie chiavi.

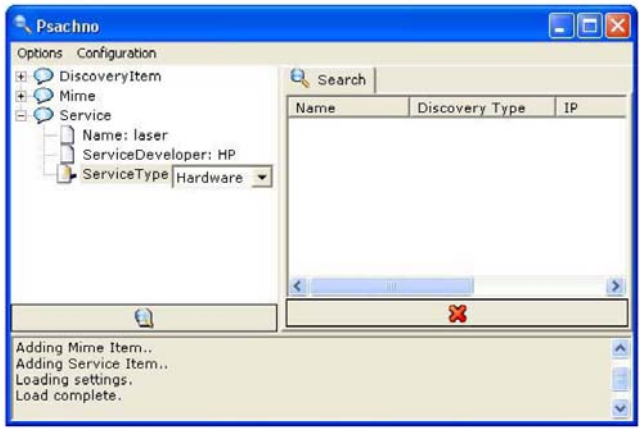


Figura 2.10: interfaccia grafica di Psachno

Psachno è implementato in C# tramite il *framework* .NET, è stata fatta questa scelta, per via dei metadata tags personalizzati che nel sistema vengono denominati attributi. Gli attributi personalizzati permettono allo sviluppatore di aggiungere *meta-informazioni* alle librerie.

In Psachno, le *meta-informazioni* sono usate per collegare i tipi alle librerie personalizzate. L'implementazione usa un *GUI front end* per fornire all'utente un'interfaccia grafica. Esistono attributi personalizzati grafici chiamati *PsachnoGuiPropertyAttribute* che possono essere aggiunti ai *DiscoveryItems* per fornire informazioni su come gli attributi possano essere mostrati sullo schermo come controlli come si vede in Figura 2.10.

2.1.6 ReMMoC (Reflective Middleware for Mobile Computing)

ReMMoC (Reflective Middleware for Mobile Computing) è un middleware framework adattativo indipendente dal particolare protocollo di discovery e di interazione, sviluppato presso il Computing Department alla Lancaster University delle Gran Bretagna [ReMMoC]. ReMMoC è in grado di trovare il servizio richiesto indipendentemente dal protocollo di discovery e interoperare con l'implementazione di tale servizio gestendo diversi tipi di interazioni. Il framework monitorizza l'ambiente e i tipi di servizi in uso e riconfigura se stesso per rispecchiare la configurazione corrente. ReMMoC usa l'astrazione dei servizi Web per permettere ai clienti di essere sviluppati

indipendentemente dalla specifica implementazione del servizio, questa astrazione è poi mappata nell'appropriato protocollo a *run-time*.

2.1.6.1 Il Framework ReMMoC

Questa sezione descrive la progettazione di ReMMoC, di cui l'operazione chiave è adattare dinamicamente i protocolli di interazione e discovery per venire incontro al ambiente del servizio mobile corrente, e quindi superare l'eterogeneità di piattaforme. È basato sulla filosofia di progettazione *OpenORB* che promuove l'utilizzo di riflessione, tecnologia a componenti e struttura a componenti, al fine di sviluppare un middleware riflettivo [OpenORB]. I componenti sono i blocchi principale con i quali è costituito il middleware, dove un componente è inteso come un elemento che specifica un'interfaccia, il quale può essere realizzato indipendentemente da terze parti. Questa tecnica promuove riuso e riconfigurabilità del livello middleware. La riflessione è poi utilizzata per fornire un meccanismo per analizzare e adattare dinamicamente i componenti della struttura. *OpenORB* a sua volta è costruito utilizzando *OpenCOM* [OpenCOM], che è un modello a componenti leggero, efficiente e basato sulla riflessione, che utilizza le funzioni chiave di Microsoft COM per rafforzare la sua implementazione, che includono: IDL di Microsoft, gli identificatori unici di COM e le interfaccie *IUnknown*, figura 2.11.

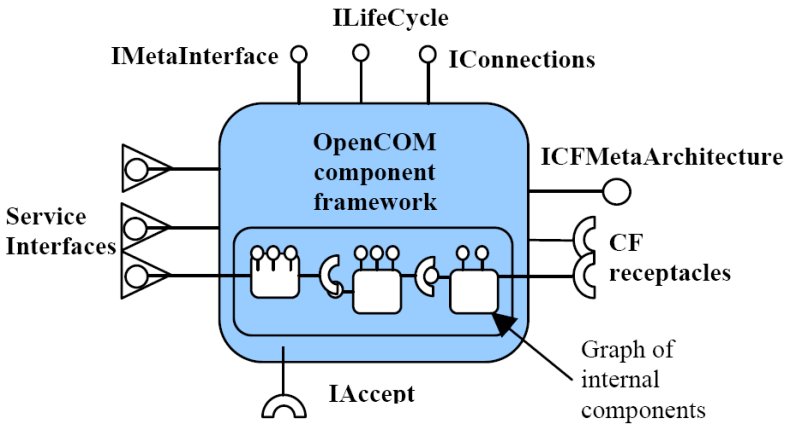


Figura 2.11: il modello OpenCOM

2.1.6.2 Architettura ReMMoC

ReMMoC è progettato sopra dispositivi mobili per lo sviluppo di applicazioni client. Quindi l'architettura di ReMMoC, illustrata in figura 2.12, è progettata come un insieme minimale componenti *OpenCOM* per ridurre l'utilizzo di risorse. ReMMoC è una architettura a due layer consistente di un livello superiore costituito strutture di componenti e singoli componenti fra loro interconnessi. Ci sono tre sezioni nel livello superiore:

1. La sezione del *concrete*, la quale è composta di due strutture di componenti, la prima struttura detta *Binding* realizzata per l'interoperabilità con servizi e che gestisce differenti tipi di interazione e una struttura chiamata *Service Discovery* per la ricerca di servizi forniti dai diversi protocolli di discovery di servizi. La struttura *Binding* è configurata tramite l'inserimento di differenti tipi di implementazioni come SOAP RPC, Event sunscriber etc. e la struttura service discovery è configurata in modo simile attraverso l'inserimento di differenti protocolli.
2. Il modello astratto di programmazione middleware, il quale implementa delle API per realizzare il servizio di discovery e interazione del servizio indipendente dall'implementazione del protocollo.
3. La sezione relativa al mappaggio dall'astratto al concreto, la quale consiste di un componente per mappare la richiesta astratta di un servizio collegandola all'implementazione del discovery relativa alla località corrente.

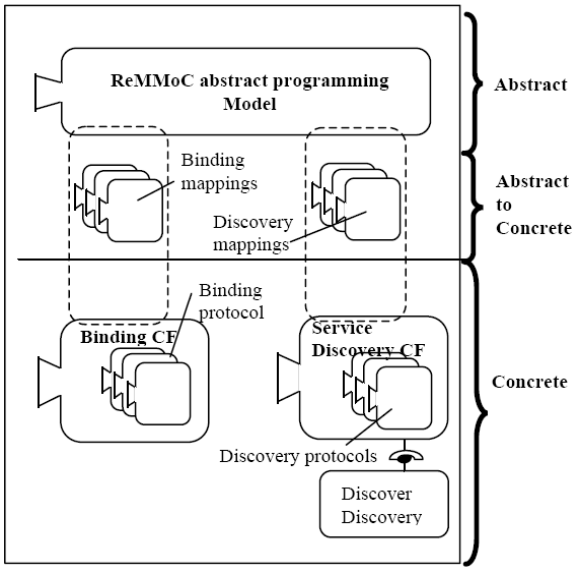


Figura 2.12: L'Architettura ReMMoC

ReMMoC è flessibile per venire incontro alle differenze di requisiti delle applicazioni. Per esempio, la piattaforma può essere configurata per precisare la sezione *concrete*, oppure anche solo una delle due struttura di componenti che la compongono. Questa richiesta viene incontro ad applicazioni su dispositivi dalle risorse limitate come i palmari; lo spazio occupato in memoria è significativamente minore ed inoltre è evitato l'overhead per elaborazioni extra.

In modo molto simile, la piattaforma è estensibile per permettere l'aggiunta di più strutture a componenti per altre proprietà non funzionali come la gestione della sicurezza e delle risorse.

2.2 Conclusioni

La numerosa documentazione sui lavori di ricerca e sviluppo che cercano di indirizzare il tema del discovery in generale dimostra l'attuale interesse della comunità scientifica verso questo tema. Sono stati proposti molti approcci e modelli per la realizzazione di servizi di questo genere nell'ambito delle applicazioni multimediali, poiché sembra che la comunicazione digitale stia sempre più allargando le proprie prospettive verso questo tipo di contenuti.

In questo capitolo sono state presentate 6 diverse ricerche nell'ambio del discovery mostrando, per ognuno di esse, i meccanismi inclusi, peculiarità e le nuove proposte con l'intento ti trarre da una loro analisi comparativa utili spunti per lo sviluppo di questo lavoro di tesi. L'attenzione è stata rivolta principalmente a PDP e a ReMMoC; il primo per il concetto di distribuzione del discovery e per la presenza della cache, mentre il secondo per le caratteristiche di interoperabilità nei diversi standard di discovery.

CAPITOLO 3

STANDARD

PER IL

DISCOVERY DI SERVIZI

Da diversi anni, il mondo dell'industria e della ricerca stanno svolgendo studi approfonditi per la realizzazione di sistemi di configurazione automatica, indicati genericamente con il termine servizi di discovery. Jini, Universal Plug and Play (UPnP), Salutation e il Service Location Protocol (SLP) sono i pionieri di questo tipo di ricerca [JINI, SLP, UPnP]. Tuttavia, la scelta di concentrarsi sul discovery di servizi, va oltre l'esigenza di trovare soluzioni plug-and-play o fornire un supporto facilitato per l'utente SOHO (small office/home office). Il vero potenziale dei servizi di discovery lo si può vedere nei dispositivi mobili e negli ambienti di calcolo distribuiti. Mobilità significa aver la possibilità di uscire da ambienti configurati ed addentrarsi in reti straniere e con infrastrutture sconosciute. Tuttavia, poiché un calcolatore mobile non può predire a priori lo spostamento in tali infrastrutture, potrebbe non sapere come sfruttarle o persino non avere la possibilità di interagire con loro. Per esempio, un dispositivo mobile potrebbe non essere in grado di utilizzare una stampate che si trova nelle vicinanze, perché non ha il driver adatto alla stampate stessa, oppure un PDA potrebbe avere un accesso lento al Web perché non è a conoscenza della presenza di una cache Web nell'ambiente. Mentre la computazione mobile si evolve oltre la capacità di collegarsi senza fili per leggere email o per navigare sul Web dovunque e su qualsiasi dispositivo, questa è limitata nello sfruttare risorse locali, eventuali dispositivi pari caratteristiche, servizi locali. Con l'arrivo di servizi basati sulla locazione ed la computazione peer-to-peer, il discovery di servizi sta assumendo un nuovo ruolo come middleware, cruciale per la computazione mobile. Il discovery di servizi può anche avvantaggiarsi di ambienti di calcolo distribuito, in cui numerosi elementi di calcolo ed sensori devono spesso interagire per realizzare la funzionalità e l'intelligenza volute. In tali ambienti, il

discovery del pari e la pubblicazione automatica può permettere allo spazio distribuito di cambiare ed evolversi dinamicamente e in modo automatico.

3.1 Protocolli di discovery di servizi

Nei prossimi paragrafi verranno esaminati gli standard di discovery di servizi disponibili attualmente,prima di ciò verranno presentati tre scenari che introdurranno le scelte fatte dalle diverse software house promotrici di questi standards.

Come primo scenario, si immagini di trovarsi in un taxi senza portafogli. Fortunatamente, si ha a disposizione un telefono cellulare e il fornitore del servizio cellulare sia dotato di tecnologia Jini, in modo che riesca a comunicare con la banca. Sul vostro schermo del telefono, sarà possibile vedere la compagnia del servizio di taxi, in modo da avere la possibilità di scaricare una applicazione di pagamento elettronico, per autorizzare il pagamento. Il sistema di pagamento dell'azienda di taxi immediatamente riconosce la transazione e trasmette una ricevuta alla stampate nel taxi. Arrivati a destinazione, il cellulare automaticamente informa l'utente, che nelle vicinanze si è liberata la camera in un hotel adatto alle proprie esigenze monetarie.

Come secondo scenario, si consideri un commesso di assicurazione che visita l'ufficio di un cliente. Il commesso desidera informare il cliente sui nuovi prodotti e sulle loro opzioni, le quali sono memorizzati in palmare con installato Windows CE e che questo palmare disponga di una connessione rete senza fili e supporta lo standard UPnP, in modo automatico viene a conoscenza ed utilizza una stampate collegata ad una rete Ethernet, senza alcuna configurazione e messa a punto di rete. A questo punto il commesso può stampare qualunque documento desideri che sia contenuto nel palmare e promuove i nuovi prodotti.

Infine, si consideri come terzo scenario un proiettore intelligente collegato in rete in modo da avere accesso ad documenti del cliente. Dopo essere stato autenticato, l'utente potrebbe selezionare un insieme di tabelle memorizzate o di altri documenti per la proiezione. Piuttosto che portare dei lucidi ad una riunione, l'utente accede a questi tramite il server della rete.

Il primo scenario è una tipica dimostrazione di Jini della Sun Microsystems, mentre il secondo è uno scenario di UPnP di Microsoft ed infine il terzo scenario è un esempio di Salutation. Come si può notare la problematica trattata è molto simile: utilizzo di dispositivi mobili, configurazione del sistema minima, accesso ad infrastrutture sconosciute, permesso tramite il protocollo di discovery di servizi e possibilità di collegarsi alla rete più vicina disponibile. I diversi protocolli di discovery di servizi, tuttavia hanno origini differenti, il problema viene affrontato da differenti punti di vista e viene risolto in modo diverso a seconda del produttore. Questi standards, comunque non sono necessariamente realizzati per ambienti mobili e non risolvono tutti i problemi visti nei capitoli precedenti. Dagli esempi sopra esposti si possono ricavare alcune informazioni basilari: la connessione del dispositivo con i servizi offerti dalla rete richiede un qualche tipo di meccanismo di ricerca, denominato Discovery. Il dispositivo è in grado di trovare la rete, ma è anche in grado di trovare i servizi che mette a disposizione, questo è il meccanismo viene chiamato Lookup. Gli esempi evidenziano anche la gestione degli eventi: la notifica istantanea sul dispositivo del verificarsi di una determinato evento.

3.1.1 Jini

Jini fu introdotto nel Luglio del 1998, è un'estensione naturale di Java che permette ai servizi di una rete di comunicare usando lo stesso linguaggio indipendente dalla piattaforma.

3.1.1.1 La visione di Jini

Un sistema Jini è un sistema distribuito basato su l'idea di federazione di gruppi di utenti e di risorse necessarie a questi utenti. L'obiettivo principale è trasformare la rete in un flessibile, semplice tool di amministrazione in cui le risorse possano essere trovate da utenti e da dispositivi. Le risorse della rete Jini possono essere dispositivi hardware, componenti software o la combinazione delle due cose.

L'obbiettivo del sistema è fare della rete una entità più dinamica che meglio rifletta la natura dinamica di un gruppo di lavoro attraverso la possibilità di aggiungere e cancellare servizi in modo flessibile. Jini è inoltre pensata per essere robusta e affidabile. Se alcune risorse non sono più raggiungibili, per via di una disconnessione o un crash, Jini provvede automaticamente ad eliminare i servizi che non funzionano correttamente, in modo che i servizi restanti funzionino in un ambiente coerente. La rete Jini, inoltre, si accorge automaticamente quando una risorsa è aggiunta o aggiornata. In breve, Jini definisce il modo in cui i vari servizi si connettono e si trovano tra loro, non come lavorano e che tipo di servizio implementano. Per questo motivo Jini può essere utilizzato per gestire qualsiasi tipo di situazione: da una rete casalinga di dispositivi Hi-Fi, fino ad una rete enterprise di gestione dati.

3.1.1.2 Jini estende Java

Il sistema Jini estende l'ambiente di lavoro delle applicazioni Java dalla singola virtual machine (VM) ad una rete virtuale di VM. L'ambiente di sviluppo Java permette di avere un buon supporto per i sistemi distribuiti, poiché mette a disposizione le seguenti caratteristiche:

- Dati e codice possono essere trasferiti in modo pressoché trasparente da una macchina all'altra.
- L'ambiente run-time incorpora i meccanismi necessari per garantire la sicurezza e quindi assicura garanzie prima di eseguire il codice scaricato da altre macchine.
- La forte tipizzazione degli oggetti, tipica di Java, permette di sapere se una classe potrà girare su un'altra macchina remota conoscendo solamente la sua interfaccia.

Questa soluzione si propone di evolvere l'idea di rete, da mero strumento di comunicazione a luogo dove risiedono servizi, utilizzando una piattaforma indipendente dall'hardware e dal sistema operativo.

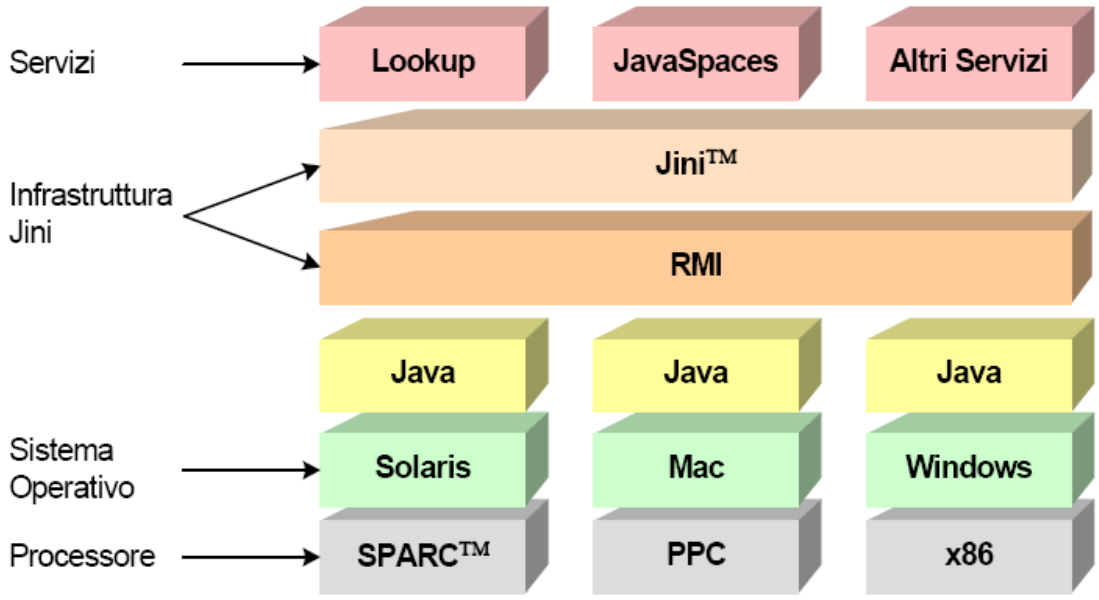


Figura 3.1 Infrastruttura Jini.

L'insieme di questi fattori permette agli oggetti componenti del sistema di configurarsi automaticamente ed essere eseguiti in JVM diverse da quella di origine. Gli obiettivi finali sono:

- Abilitare gli utenti a condividere risorse e servizi in rete.
- Fornire un accesso facile alle risorse a prescindere dalla collocazione topologica e permettere l'accesso agli utenti indipendentemente dalle proprie postazioni di lavoro.
- Semplificare i task di costruzione, di mantenimento e alterazione di una rete composta da dispositivi, software e utenti. Estende l'ambiente delle applicazioni Java da una singola VM ad una rete di VM.

	Infrastruttura	**Modello di programmazione**	**Servizi**
Java	Java VM RMI Java Security	Java API JavaBeans …	JNDI Enterprise Beans JTS …
Java + Jini	Discovery/Join Distributed Security Lookup	Leasing Transaction Events	Stampa Transaction Manager JavaSpaces

Tabella 3.1 Infrastruttura Jini . Differenti approcci in ambito di rete.

La forte tipizzazione garantisce l'identificazione delle classi degli oggetti che girano sulla macchina virtuale anche quando si tratta di oggetti non originati in locale. Il risultato è un sistema nel quale la rete supporta una configurazione fluida degli oggetti che possono essere spostati da un luogo ad un altro e possono interpellare qualunque servizio Jini presente nella rete per eseguire operazioni. L'architettura Jini espande queste caratteristiche per semplificare la costruzione di un sistema distribuito in cui dispositivi, servizi e client possono connettersi o scollegarsi dalla rete con meccanismi semplici ed automatici. Si presuppone che i dispositivi a cui si fa riferimento siano attivi, cioè dotati di memoria e processore. In caso contrario, ci deve essere almeno un dispositivo attivo che faccia da intermediario (proxy) con la periferica. La latenza della rete deve essere ragionevole e proporzionata alle esigenze dei dispositivi. Una stampante richiede una banda differente da una telecamera. Tutto questo avviene attraverso meccanismi che permettono la configurazione della rete Jini in modo naturale, automatico e dinamico senza che vi sia la necessità di intervenire manualmente sulla rete.

3.1.1.3 L'architettura Jini

Lo scopo di Jini è la creazione di una federazione di gruppi di dispositivi e componenti software in modo da creare un singolo e dinamico sistema distribuito.

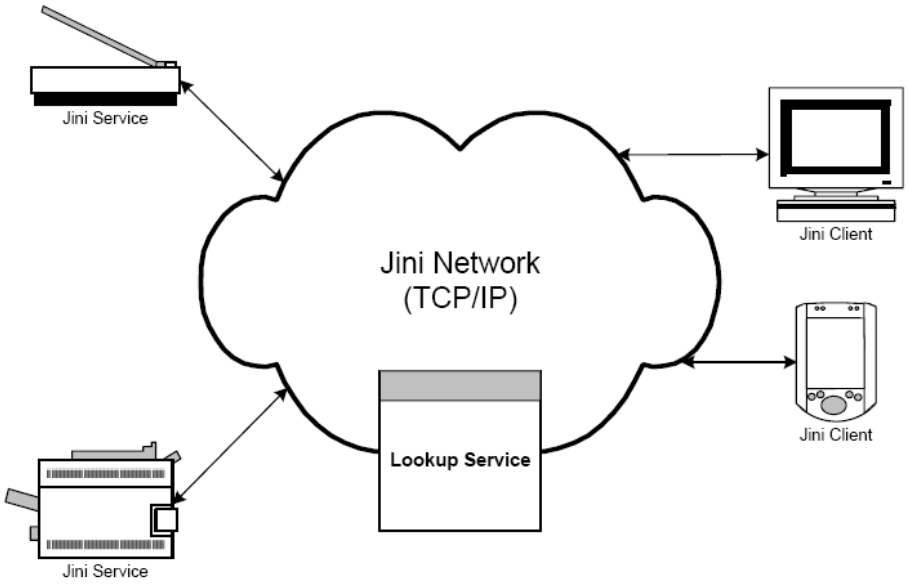


Figura 3.2 Esempio di architettura Jini.

Servizi (Services)

Il componente principale dell'architettura Jini è il service. Un service è un'entità che implementa un qualche tipo di servizio che può essere utilizzato da una persona, da un programma o da un altro service. Il service può essere un programma di calcolo, un'interfaccia per accedere a dati, un canale di comunicazione (bridge) verso altre architetture distribuite, un altro utente o un dispositivo hardware. Esempi di service sono dispositivi di stampa, scanner, macchina fotografiche digitali, o bridge per accedere ad altri servizi di discovery (Salutation, Upnp, etc).

Servizio di Lookup (Lookup service)

Il servizio di nomi di Jini è il Lookup Service. I servizi sono individuati tramite il Lookup service che è il sistema di nomi che permette di trovare altri servizi. Senza il Lookup service il network Jini non funzionerebbe, i vari servizi non riuscirebbero a trovarsi e ad interagire tra loro. Poiché questo componente è di vitale importanza, è previsto che, per aumentare la robustezza e l'affidabilità del sistema, vi possano essere più Lookup service in esecuzione contemporaneamente su macchine diverse.

Il servizio di Lookup è un sistema che tiene traccia dei servizi che hanno notificato la loro presenza sulla rete. I servizi sono tenuti a rinnovare la propria registrazione dopo un tempo stabilito al momento della registrazione stessa (leasing). Allo scadere del tempo, se il servizio non rinnova la registrazione, viene cancellato dalle entry del servizio di Lookup. Il processo di registrazione da parte di un servizio consiste nell'invio di un oggetto (service object) e di una serie di attributi descrittivi al Lookup. Il service object contiene l'interfaccia per l'utilizzo del servizio e quindi anche i metodi che utenti e applicazioni potranno invocare.

La ricerca di un servizio da parte di un'altra applicazione avviene attraverso il matching degli attributi: se gli attributi specificati dal client corrispondono ad un servizio registrato nel Lookup service, quest'ultimo risponde con il service object corrispondente. Un servizio si registra sul Lookup service utilizzando due protocolli: Discovery e Join. Il protocollo di Discovery permette di trovare il Lookup Service, mentre tramite il secondo si effettua la registrazione (Join). Sun Microsystems mette a

disposizione un'implementazione di riferimento del Lookup service chiamata Reggie contenuta nei files lib/reggie.jar e lib/reggie-dl.jar.

Clienti

Un client è, in generale, un'applicazione che utilizza un servizio. In Jini può essere, a sua volta, anche un servizio per un altro client. Nell'implementazione corrente di Jini, tutti i servizi possono essere visti come client, perché tutti hanno la necessità di utilizzare uno stesso servizio fondamentale: il Lookup service. I client, in generale, cercano servizi conoscendo a priori l'interfaccia che utilizzeranno (e quindi i metodi che mette a disposizione) oppure attraverso il matching degli attributi.

Comunità (Gruppi)

Tutti i servizi che fanno parte di una rete locale sono generalmente sempre raggiungibili. In Jini i servizi si possono unire in gruppi di entità più piccoli che sono chiamati groups. La grandezza massima dei gruppi non è specificata, ma idealmente è data dalla dimensione di un gruppo di lavoro. La comunità di default è la public. Durante la fase di Discovery, un servizio o un client può specificare il gruppo d'appartenenza del servizio che si cerca.

3.1.1.4 I concetti fondamentali

L'abilità di Jini di creare spontaneamente una federazione di servizi che sia robusta e tollerante ai guasti, è basata su una serie di concetti fondamentali:

- Discovery: processo con cui un client o un servizio localizza il sistema di nomi all'interno della rete.
- Join: meccanismo che permette di registrare un nuovo servizio sul sistema di nomi locale o remoto.
- Lookup: processo attraverso il quale il client interroga il sistema di nomi per trovare il servizio richiesto.

- Leasing: meccanismo disponibile per i servizi Jini che realizza un sistema di garbage collection distribuito, introdotto per risolvere le inconsistenze che intervengono nella rete di servizi Jini.
- Remote Event: estensione a livello di rete Jini del modello ad eventi Java.
- Transazioni: implementazione di un modello per gestire il parziale fallimento di operazione multiple, in modo da garantire affidabilità e robustezza al sistema.

Discovery

Il processo di Discovery è usato dai client e dai servizi per trovare un Lookup service. La ricerca avviene differentemente a seconda che si cerchi in una rete locale (LAN) o geografica (WAN). In una LAN, il client, appena è in esecuzione, utilizza il multicast Discovery per localizzare, in maniera attiva, un Lookup service. Passato un certo tempo, la fase di Discovery attiva termina e il client si mette in ascolto di pacchetti multicast provenienti dal Lookup service. Se il Lookup è situato in una WAN è necessario affidarsi ad un protocollo di localizzazione che non utilizzi la comunicazione di gruppo. La ricerca da parte del client avviene attraverso la specifica dell'indirizzo internet del Lookup service di cui si vogliono informazioni e l'utilizzo di un protocollo orientato alla connessione, l'unicast Discovery:

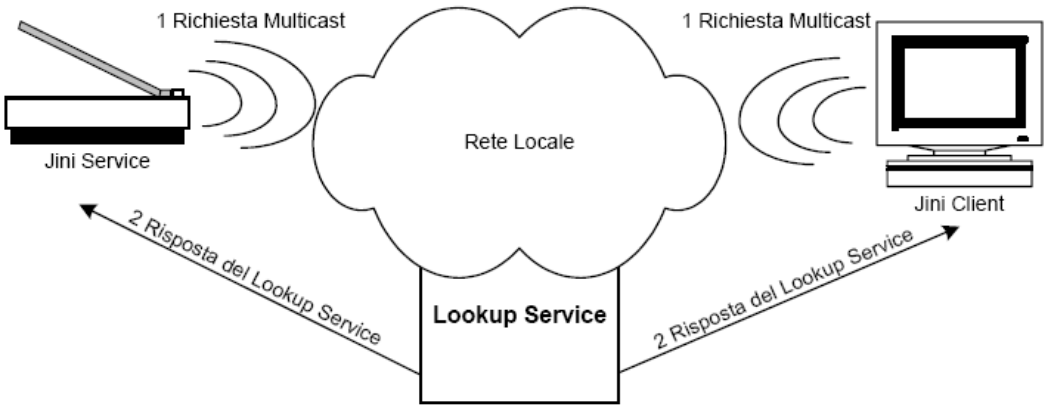


Figura 3.3 Multicast Discovery

Il multicast Discovery è utilizzato quindi per trovare un Lookup service che conterrà servizi che appartengono alla locale comunità Jini, mentre l'unicast Discovery permette di localizzare Lookup service che contengono servizi remoti. Questa dualità presente in

fase di Discovery è un approccio fondamentale per scalare le reti di servizi Jini a grandi dimensioni.

Join

Quando un servizio ha terminato la fase di Discovery e ha trovato un Lookup service, ha la necessità di registrasi sul Lookup service stesso. Il servizio invia un oggetto proxy (service object) e gli attributi ad esso associati. Il service object contiene la descrizione dell'interfaccia per l'utilizzo del servizio, nonché l'implementazione dei metodi che verranno utilizzati dalle applicazioni.

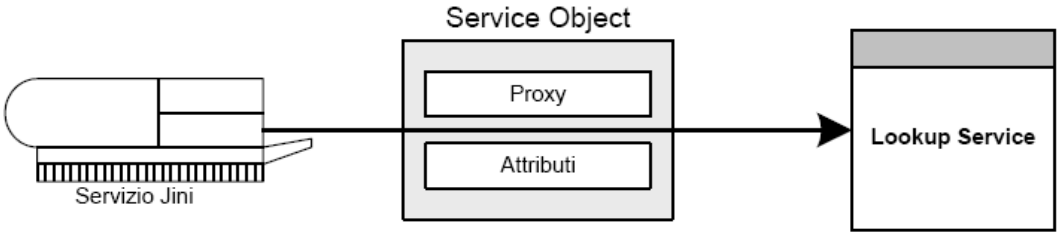


Figura 3.4 Join.

Il servizio si registra sul Lookup service inviando un service object. Il service object può essere implementato in maniera diversa a seconda del tipo di servizio che vogliamo offrire. Qui di seguito vengono riportate le implementazioni più comuni:

1. L'oggetto proxy è il servizio: Caso estremo è quando l'oggetto proxy contiene il servizio stesso: tipicamente ciò avviene quando non si ha più la necessità di comunicare con il service provider. Il service provider ha, come unico compito, una volta effettuata la registrazione, di rinnovare periodicamente la registrazione in modo che il proxy corrispondente, presente nel Lookup Service, non venga cancellato. In questa configurazione il servizio è eseguito completamente all'interno del client. Nella figura 3.5 si nota che l'oggetto proxy non ha necessità di comunicare con il servizio Jini che lo ha creato, il Jini Server si occupa solamente di rinnovare il Lease.

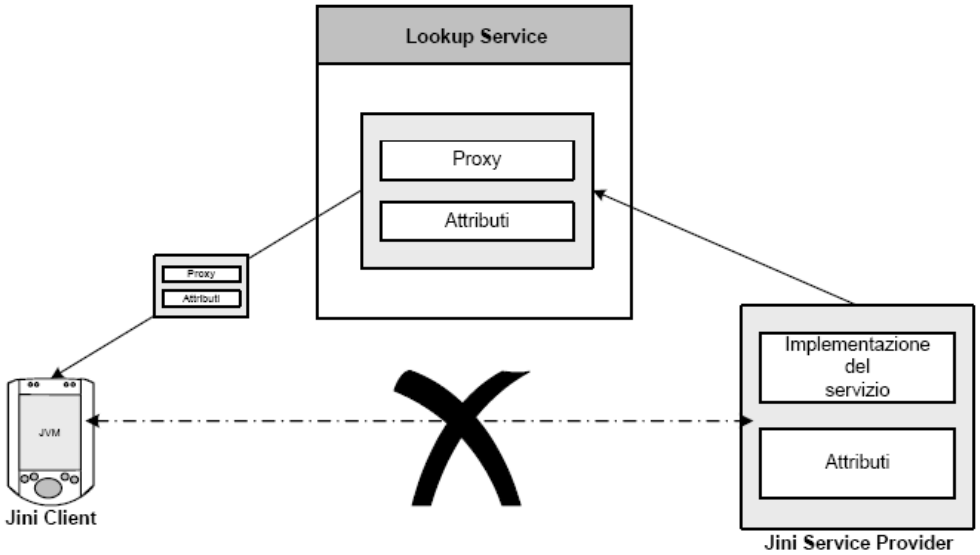


Figura 3.5 Il servizio è eseguito completamente nel client.

2. L'oggetto proxy è un proxy per comunicazione remota: L'implementazione opposta è invece quella in cui tutta l'elaborazione avviene da lato server, l'oggetto proxy inviato al client serve per raccogliere le invocazioni e chiamare i metodi corrispondenti dal lato server. L'utilizzo della RMI di Java permette di effettuare queste operazioni in modo del tutto trasparente. Questo approccio è utile quando l'elaborazione dal lato client non serve, ma è invece essenziale la comunicazione con il service provider. Questa soluzione è adottata per interrogare un database o per utilizzare servizi che sono legati strettamente ad un hardware remoto (lettura dello stato di apparecchiature, interazione con dispositivi hardware che non hanno un JVM, query SQL, etc.).
3. Altri oggetti proxy: Un altro approccio è il non utilizzare RMI per comunicare con un server remoto. In questo caso il service provider, all'atto della registrazione nel Lookup service, deve creare un oggetto proxy che abbia al suo interno la capacità di creare un qualche tipo di comunicazione con il server una volta che è esportato. Rispetto al caso precedente, si perde in trasparenza e non si ha a disposizione un servizio di nomi (rmiregistry) per localizzare il server una volta che l'oggetto proxy è in esecuzione nel client. Il metodo di comunicazione dell'oggetto proxy con il server non è specificato: si possono utilizzare delle socket o utilizzare un formato di comunicazione proprietario. Questo modello è quindi adatto per portare nel mondo

Jini applicazioni client-server di tipo legacy, cioè quando si utilizzano sistemi di comunicazione e rappresentazione dei dati estranei al mondo Java.

Lookup

Questa fase avviene quando un client ha già effettuato la fase di Discovery con successo e vuole richiedere un servizio Jini che abbia le caratteristiche richieste.

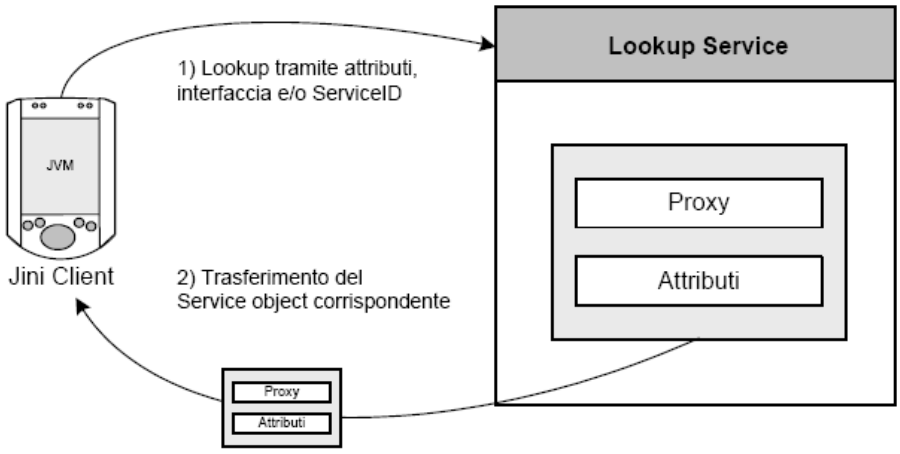


Figura 3.6 Lookup.

La richiesta al Lookup service avviene specificando uno o più dei seguenti campi:

- *Attributi*: viene effettuata una comparazione con gli attributi registrati all'atto del Join dei servizi.
- *Interfaccia*: si specifica un'interfaccia che il servizio poi implementa. Si può, ad esempio, specificare un'interfaccia Stampante per poi scegliere tra i service provider che la implementano.
- *ID number*: Ad ogni servizio Jini, all'atto della registrazione, è assegnato un numero identificativo di 128 bit (ServiceID) che può essere utilizzato nella fase di Lookup per richiedere un servizio ben preciso.

Leasing

Il concetto di lease permette a Jini di avere un meccanismo di garbage collection distribuito e di aumentare notevolmente l'affidabilità dell'intero sistema. La

registrazione di un servizio su un Lookup service dura solamente un breve periodo di tempo; se allo scadere di quest'ultimo l'iscrizione non è rinnovata, il servizio è cancellato e le risorse allocate vengono liberate. Questo meccanismo permette di eliminare delle entry nel Lookup service che fanno riferimento a servizi che non sono più raggiungibili per un problema di rete o del servizio stesso. Dal lato dell'utilizzatore del servizio i lease possono essere esclusivi o non esclusivi. Un lease esclusivo assicura che in un dato istante un solo utilizzatore abbia accesso ad un servizio. Con un lease non esclusivo più utilizzatori possono condividere l'uso di un servizio. Questo strumento è importante sia per il fault tolerance e sia per la prevenzione dei deadlock. Le interfacce lease definiscono il modo di allocare e liberare risorse con meccanismi quali la negoziazione e la durata base. Nel modello di programmazione Java è presente la nozione di mantenimento del riferimento ad una risorsa, con la possibilità di recuperarlo in caso di guasti sulla rete. Le interfacce lease aggiungono al modello il concetto di tempo. La durata di un lease è un compromesso nella progettazione del sistema: se da un lato lease brevi sono necessari per mantenere il sistema in uno stato consistente, dall'altro aumentano notevolmente il traffico di rete associato al processo di rinnovo del lease. D'altro canto lease troppo lunghi diminuiscono il traffico di rete, ma possono mantenere il network Jini in uno stato inconsistente per lungo tempo [Sin00].

Remote Event

Jini supporta gli eventi distribuiti. Un oggetto potrebbe permettere ad altri oggetti di registrare interessi verso eventi su oggetti e ricevere una notifica dell'occorrenza di quel tipo di evento. Questo permette a programmi distribuiti basati su eventi di essere scritti garantendo diversi gradi di affidabilità e scalabilità.

Transazioni

Una serie di operazioni, contenti un singolo servizio o comprendente più servizi, possono essere raggruppate in una transazione. Le interfacce di transazione di Jini forniscono un protocollo di servizio, necessario per coordinare un commit a 2 fasi. Come le transazioni siano implementate, sono lasciate al servizio attraverso l'uso delle interfacce sopra menzionate.

3.1.2 Salutation

L'architettura Salutation è stata creata per risolvere i problemi di service discovery e l'utilizzazione di questi in ambienti pervasivi e con alta mobilità delle componenti. Le soluzioni adottate sono indipendenti dal tipo di linguaggio implementativo, dal tipo di rete e dal protocollo di comunicazione utilizzato. L'architettura mette a disposizione di applicazioni, servizi e dispositivi, metodi per descrivere e rendere nota la loro presenza all'interno della rete. È inoltre possibile cercare applicazioni, servizi e dispositivi in base ad attributi e instaurare sessioni di lavoro tra loro.

La tecnologia, controllata dal consorzio Salutation, include membri dell'industria e accademici. I protocolli messi a disposizione sono aperti e non richiedono royalty per il loro utilizzo.

3.1.2.1 Architettura

L'architettura Salutation è composta di tre elementi fondamentali evidenziati in figura 3.7:

- Salutation Manager (SLM): ha un ruolo simile a quello del Lookup service di Jini: è quindi il sistema di nomi dell'architettura. Gestisce, inoltre, comunicazioni tra client e servizi ed è indipendente dal livello di trasporto sottostante.
- Transport Manager (TM): I TM isolano l'implementazione degli SLM dal livello di trasporto sottostante, rendono quindi l'architettura Salutation indipendente dal protocollo di trasporto. I TM si occupano infatti di interfacciare gli SLM con i protocolli specifici di ogni rete. C'è un TM per ogni protocollo (TCP/IP, IrDA,.). La gestione di un nuovo protocollo comporta, di conseguenza, l'implementazione di un nuovo TM. I TM hanno anche il compito di trovare altri SLM. La fase di discovery avviene utilizzando tecniche di broadcast, tabelle statiche d'indirizzi o sistemi di directory che contengono liste di SLM.
- Unità funzionali (FU): è il nome con cui il consorzio Salutation chiama i servizi. Le FU comunicano con gli SLM attraverso una serie di API chiamate Salutation Manager Application Interface (SLM-API). L'utilizzo delle SLM-API permette di

sviluppare applicazioni che sono indipendenti dal livello di trasporto, poiché i TM non sono visibili dalle FU.

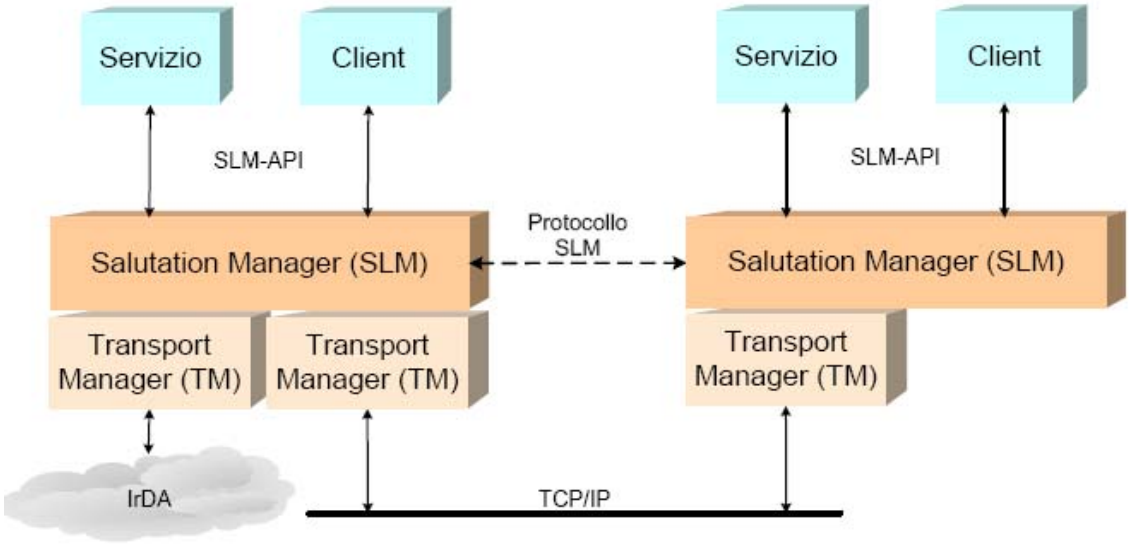


Figura 3.7 Architettura di Salutation

La maggior parte dei servizi ha un Salutation Manager locale a cui fare riferimento ed è utilizzato per accedere ad altre unità funzionali. Nel caso in cui non sia presente un SLM locale, è necessario accedere a Salutation Manager remoti. In questo caso si deve implementare, all'interno dei servizi, un protocollo di comunicazione. Il consorzio Salutation ha scelto il Remote Procedure Call Protocol (RPC) di Sun Microsystems per accedere a SLM remoti. I dettagli di funzionamento del SLM vengono descritti più in dettaglio nei paragrafi successivi evidenziando, in particolare, la memorizzazione dei descrittori dei servizi, la localizzazione delle risorse e la gestione dell'interazione tra client e server.

3.1.2.2 SLM: Service Registry

Tutti i Salutation Manager hanno una struttura dati che contiene le informazioni sui servizi registrati chiamata Registry. I servizi possono essere sia locali che remoti. Questi ultimi sono raggiungibili attraverso l'utilizzo di RPC (Remote Procedure Call). Un SLM locale può avere problemi di prestazioni se in presenza di un numero elevato di

servizi registrati. L'SLM, infatti, non è un componente specializzato per la gestione di directory e quindi un elevato numero di accessi può pregiudicare le sue prestazioni.

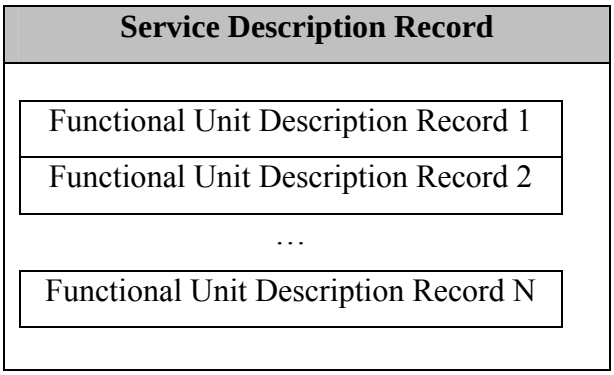


Figura 3.8 Struttura di un Service Description Record.

Ad ogni servizio presente nel Registry, è associato un descrittore chiamato Service Description Record (Figura 3.8) che contiene a sua volta una serie di vettori (Functional Unit Description Record) che descrivono i compiti significativi del servizio. Vi sono tre classi di Service Description Record che si differenziano per il contesto in cui sono utilizzate:

- *Registered Service Description Record*: è creato per ogni servizio che si registra nel SLM.
- *Requested Service Description Record*: è generato ogni volta che un client cerca un servizio che non è registrato nel SLM locale.
- *Reply Service Description Record*: è creato in risposta ad un altro SLM che cerca un servizio non presente in ambito locale.

Il *Funcional Unit Description Record* definisce in maniera più specifica il servizio che viene offerto. Il Consorzio Salutation specifica formalmente degli identificativi univoci (Functional Unit ID) attraverso i quali si risale al tipo di servizio svolto (Fax, Stampante, etc.).

3.1.2.3 SLM: Service Discovery

Si entra ora nel dettaglio di come un client può richiedere un servizio. Il processo di Discovery di un servizio da parte di un client, avviene attraverso la specifica di attributi

presenti nelle Functional Unit Description Record. La richiesta è inoltrata al SLM locale, ma se il servizio non è disponibile, è necessario coinvolgere nella ricerca altri SLM remoti. Le specifiche Salutation demandano il compito della localizzazione di altri SLM, al TM che può utilizzare per la ricerca, tecniche di broadcast, accedere a servizi di directory o utilizzare una lista statica di indirizzi che fa riferimento ad altri SLM.

Il protocollo utilizzato dai client per eseguire le query è chiamato Capability Exchange. Esso permette di scoprire i dettagli dei servizi messi a disposizione da server remoti. Il client ha la possibilità di ottenere informazioni inerenti la classe di servizio implementata (per una stampante si possono sapere i formati di stampa supportati), ma anche le informazioni necessarie per utilizzare il servizio. Sono disponibili tre metodologie di ricerca per trovare il servizio. La prima è chiamata ALL CALL e permette di recuperare tutte le unità funzionali presenti in un altri SLM, per poi fare una cernita locale di quelle che soddisfano la query. La seconda tecnica, chiamata TYPE CALL, permette di recuperare le unità funzionali che sono del tipo specificato. L'ultima tecnica, la MATCH CALL, permette di trovare solamente le unità funzionali che soddisfano una serie di attributi. Tutte le tecniche proposte non permettono di limitare lo spazio topologico di ricerca, rendendo critica la scalabilità del protocollo Salutation in reti di grandi dimensioni. Questa limitazione è anche riconosciuta dal consorzio Salutation che sta cercando di risolvere il problema in future versioni del protocollo.

3.1.2.4 Gestione di sessione

Una volta che il processo di Discovery è terminato, il client procede alla fase di utilizzo del servizio. Richiede al SLM locale di aprire un canale di comunicazione bidirezionale, chiamato Service Session, per comunicare con la risorsa. Lo scambio di messaggi avviene utilizzando un protocollo ben definito utilizzando tre modalità distinte:

- Native Personality: i dati sono scambiati utilizzando un protocollo proprietario; si ha interazione diretta tra client e server senza l'intervento di SLM.

- Emulated Personality: gli SLM gestiscono il flusso dei messaggi, svincolando così client e server dalla conoscenza del tipo di protocollo di trasporto sottostante. Gli SLM non alterano, in questo profilo, la rappresentazione dei dati.
- Salutation Personality: come nel caso precedente, agli SLM è demandata la gestione del flusso di dati; vengono inoltre definiti tipi di dati da utilizzare nella trasmissione. In questo tipo di comunicazione si seguono standard predefiniti per dialogare con le unità funzionali, permettendo così un aumento dell'interoperabilità tra client e servizi.

3.1.3 Service Discovery Protocol: Bluetooth

Bluetooth [BlueT] è uno standard per la comunicazione radio a breve distanza, pensata per sviluppare dispositivi piccoli e a basso costo. E. sviluppato da un consorzio d'industrie che ha come membri Ericsson, Nokia, IBM. Sono già disponibili sul mercato implementazioni del protocollo: cuffie, fotocamere e cellulari che, una volta in prossimità, si accorgono della presenza di dispositivi vicini e creano una connessione. Non sono invece disponibili applicazioni più complesse che richiedono sincronizzazione automatica di dati tra dispositivi diversi. Le frequenze di trasmissione sono nella banda di 2.4 Ghz, la distanza di funzionamento tra i dispositivi è di circa 10 metri e la velocità massima di trasmissione è di 720 Kbps.

3.1.3.1 Architettura

L'architettura Bluetooth (figura 3.9) presenta una suddivisione in livelli che parte dalla specifica dei livelli più bassi del modello ISO OSI. Questa scelta è necessaria poiché deve essere garantita l'interoperabilità tra dispositivi di produttori diversi. Applicazioni di produttori differenti possono utilizzare protocolli di trasporto differenti, ma devono essere assolutamente conformi allo standard Bluetooth per quanto riguarda i livelli inferiori (Data Link e Livello fisico). I livelli inferiori specificano le caratteristiche hardware del collegamento in radio frequenza e si occupano di garantire la correttezza dei bit trasmessi. Il livello superiore, che corrisponde al livello 3 del

modello ISO OSI, è l'LMP (Link Manager Protocol) che si occupa di stabilire la connessione tra diversi dispositivi e implementa servizi di autenticazione e cifratura.

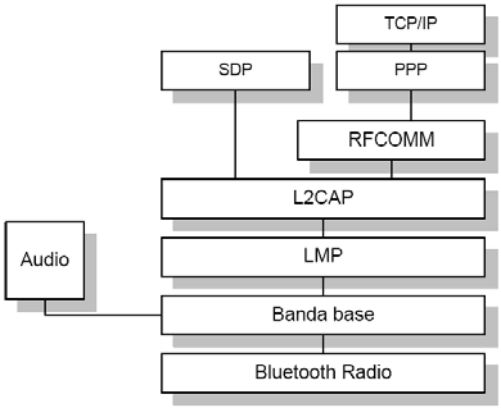


Figura 3.9 Architettura Bluetooth.

Il livello di trasporto è implementato utilizzando il protocollo proprietario L2CAP (Logical Link and Adaption Protocol) che provvede al multiplexing dei dati provenienti dai livelli superiori. È previsto che offra un supporto per connessioni virtuali (tipo TCP) o servizi senza connessione (tipo UDP). Corrisponde quindi, al livello 4 di trasporto del modello ISO OSI.

Il livello RFCOMM, noto anche come Cable Replacement Protocol, emula un'interfaccia RS-232 ed è stato introdotto per riutilizzare protocolli esistenti o per introdurne di nuovi. Grazie al livello RFCOMM e al supporto Point to Point Protocol (PPP), molte applicazioni già esistenti possono accedere subito ai vantaggi derivanti dall'utilizzo di hardware e software Bluetooth, senza il bisogno di essere riscritte. Il PPP permette di utilizzare tutta una serie di protocolli gia esistenti. Mentre sono direttamente supportati TCP, UDP ed OBEX.

3.1.3.2 Protocollo di Discovery SDP

Il Service Discovery Protocol (SDP) mette a disposizione delle semplici API per enumerare i dispositivi raggiungibili dal collegamento in radiofrequenza e per consultare i servizi disponibili. I programmi client possono utilizzare queste API per cercare i servizi specificando o la classe di servizio richiesta (una stampante o un dispositivo di memorizzazione), oppure gli attributi specifici di un dispositivo (il

numero di serie o i protocolli supportati). SDP mette anche a disposizione strumenti per scoprire nuovi servizi che diventano disponibili nelle prossimità del client e per cancellarli quando non più disponibili. È essenzialmente, un protocollo studiato appositamente per gestire le esigenze di prossimità o lontananza che si hanno in una rete Bluetooth. Una volta che un client ha effettuato una richiesta all'SDP, il protocollo provvede ad interrogare altri SDP per cercare il servizio che corrisponde alla classe o agli attributi specificati (figura 3.10).

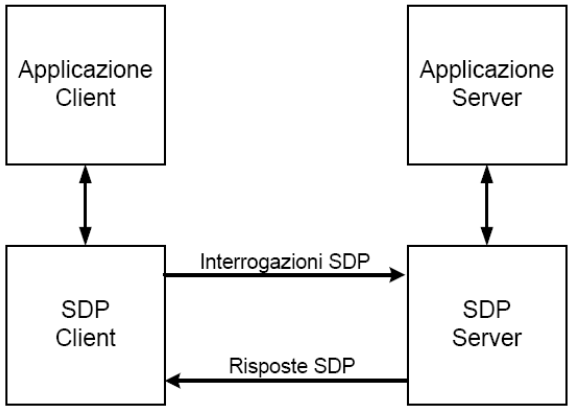


Figura 3.10 Protocollo SDP.

L'applicazione client effettua la richiesta al SDP client. Il processo di discovery avviene interrogando altri SDP server. Il server SDP mantiene una lista dei servizi registrati al suo interno e, se non trova il servizio richiesto, provvede ad inoltrare la richiesta verso altri SDP server. SDP si occupa solamente di trovare i servizi, ma non mette a disposizione un meccanismo per utilizzarli direttamente. Il client deve quindi aprire una connessione separata quando decide di utilizzare il servizio.

Un SDP mantiene tutte le informazioni su un particolare servizio in un descrittore chiamato service record. Esso è composto da una lista di attributi che descrivono le caratteristiche del servizio. SDP si differenzia molto, quindi, da altri protocolli di Discovery come Jini, le azioni specifiche per utilizzare un servizio devono essere messe a disposizione da un protocollo a più alto livello o devono essere hard coded all'interno dell'applicazione client e conoscere il protocollo da utilizzare per la comunicazione. È comunque presente un attributo, chiamato ProtocolDescriptorList, che descrive i protocolli supportati dal particolare servizio. Le specifiche Bluetooth stabiliscono che non vi possa essere più di un SDP server per dispositivo. Se un dispositivo decide di funzionare solamente come client, non ha bisogno di implementare SDP. Per via delle

caratteristiche estremamente variabili della topologia di una rete Bluetooth, l'insieme dei server SDP, disponibili per un client, può cambiare molto rapidamente. Per questo motivo un server SDP notifica continuamente la sua presenza: client vicini sentono così la sua presenza. Nel caso, viceversa, che il server si allontani e diventi irraggiungibile, è compito del client, attraverso il polling, accorgersi della sua scomparsa.

3.1.4 UPnP

L'UPnP (Universal Plug and Play) nasce con l'esigenza di estendere in ambito distribuito un meccanismo proprietario di Microsoft, il Plug and Play, già ampiamente utilizzato nei sistemi operativi Windows 95, Windows 98 e Windows 2000. Lo sviluppo del protocollo e delle API è affidato al UPnP Forum [UPnP]: un comitato tecnico diretto da Microsoft a cui aderiscono, tra gli altri, IBM, Ericsson, Cisco Systems ed Intel. UPnP si propone, quindi, come strumento per individuare e identificare risorse connesse alla rete in maniera automatica, senza la necessità di interventi di configurazione.

Il protocollo basa il suo funzionamento su protocolli aperti e ampiamente diffusi, in modo da rendere il meccanismo indipendente dal linguaggio di programmazione, ma soprattutto dal sistema operativo. Il funzionamento di UPnP si basa su protocolli internet, sono utilizzati l'IP, TCP, UDP, HTTP e l'XML. Il protocollo utilizzato dai dispositivi per rappresentare le struttura dati è di tipo dichiarativo è espresso utilizzando l'XML e usa l'HTTP come protocollo per comunicare. Il controllo dei dispositivi non richiede drivers specifici e il loro controllo avviene attraverso browser o applicazioni costruite ad hoc. L'UPnP Forum si occupa, inoltre, di standardizzare le interfacce di dispositivi comuni (stampanti, fax, fotocamere digitali e scanner) per garantire una maggiore interoperabilità tra dispositivi di produttori diversi e per diminuire i tempi di sviluppo. Appena un dispositivo è connesso alla rete, ottiene un indirizzo IP, rende disponibili i propri attributi e gli giunge notifica delle capacità degli altri dispositivi presenti. Successivamente i dispositivi possono comunicare direttamente tra loro. Non è richiesta la presenza di server DNS [RFC1034] e DHCP [RFC2131], ma sono usati se presenti.

3.1.4.1 Architettura

L'architettura UPnP basa il suo funzionamento su una serie di protocolli standardizzati a livello internazionale per garantire l'interoperabilità tra dispositivi di diversi produttori. Il compito dell'UPnP Forum è quello, di definire i livelli superiori dell'architettura (vedi figura 3.11). Il TCP/IP è utilizzato per garantire il livello base di connessione tra i vari dispositivi UPnP attraverso l'utilizzo del HTTP. Sono in particolare utilizzate delle sue varianti, HTTPU e HTTPMU [GolSh00], pensate per inviare messaggi usando datagrammi UDP invece che TCP. Questa scelta permette di avere un minor carico di lavoro per lo scambio dei messaggi poiché si rinuncia alla connessione virtuale garantita dal TCP.

Analizzando ancora la figura seguente si nota l'utilizzo di un protocollo per il Discovery dei servizi: Simple Service Discovery Protocol (SSDP). SSDP basa il suo funzionamento su HTTPU e HTTPMU e mette a disposizione metodi sia per trovare dispositivi in rete sia per permettere ai dispositivi di annunciare la propria presenza. L'uso combinato di queste due caratteristiche permette, secondo l'UPnP Forum, di eliminare l'overhead che si avrebbe utilizzando due meccanismi separati per il Discovery e il Join dei dispositivi. In realtà, questo vincolo di progetto limita enormemente la scalabilità del protocollo all'aumentare sia del numero di dispositivi di una rete locale sia all'estensione del suo utilizzo in una rete geografica. In quest'ultimo caso, in particolare, la necessità di non filtrare pacchetti multicast, necessari per il funzionamento dell'HTTPMU, crea problemi di traffico sull'intera rete. SSDP risulta molto simile al Service Location Protocol (SLP) (vedi paragrafo seguente), ma manca della possibilità di cercare i servizi specificando attributi.

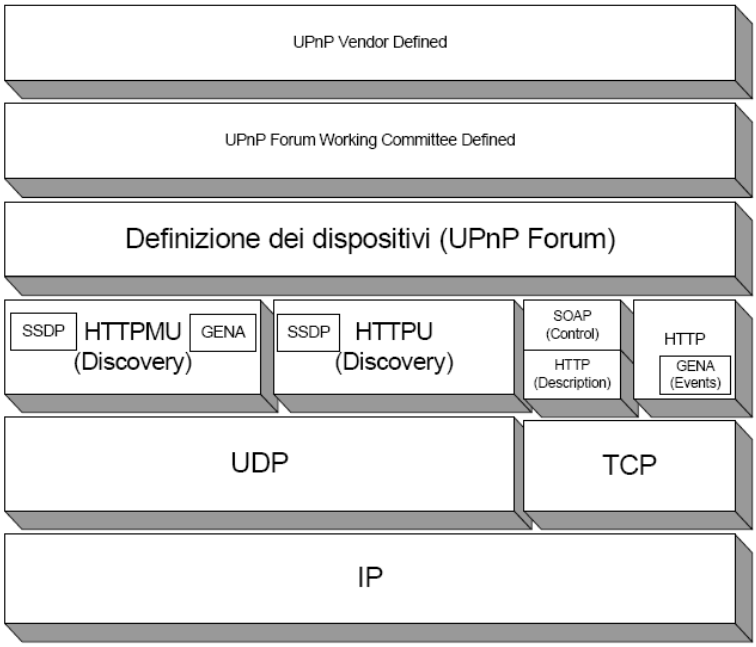


Figura 3.11 Architettura UPnP: Struttura a livelli.

La gestione degli eventi è affidata al protocollo GENEA: Generic Event Notification Architecture [Genea00]. Un client interessato alla notifica di eventi da un altro servizio, invia una richiesta, secondo il formato GENEA, indicando il servizio a cui è interessato e il tempo di durata del servizio. GENEA basa il suo funzionamento sull'uso dell'XML per la struttura dei dati e dell'HTTP come protocollo di trasporto.

Un altro protocollo utilizzato nello stack UPnP è SOAP: Simple Object Access Protocol [Dea00], utilizzato per eseguire chiamate a procedura remota (RPC). SOAP basa a sua volta il suo funzionamento su XML, per la descrizione dei dati, e su HTTP, per effettuare connessioni. Un indubbio vantaggio, rispetto ad altre tecnologie, è che l'uso del protocollo HTTP non richiede alcun tipo di riconfigurazione dei firewall eventualmente presenti ed inoltre c'è la possibilità di utilizzare SSL [FKK96] per comunicazioni sicure.

Di seguito verranno esposto come i precedenti protocolli intervengono all'atto della connessione di una nuova periferica UPnP alla rete locale. Sei fasi successive intervengono per la gestione del nuovo dispositivo:

1. Indirizzamento: Poiché il funzionamento dell'UPnP si basa sull'IP è necessario che la periferica ottenga un indirizzo IP valido. Questa operazione può svolgersi differentemente a seconda che nella rete vi sia o meno un server DHCP [RFC2131]. Nel caso in cui il server DHCP sia presente, la gestione e l'assegnamento degli

indirizzi è affidato a quest'ultimo, altrimenti UPnP utilizza il protocollo AutoIP [Tro00] che definisce il modo in cui un dispositivo sceglie un indirizzo IP tra una serie di indirizzi riservati.

2. Discovery: Una volta ottenuto un indirizzo IP, i dispositivi entrano a far parte della rete e hanno la necessità di rendere nota la propria presenza ai punti di controllo. Il protocollo SSDP è utilizzato per inviare un messaggio che contiene informazioni essenziali come l'indirizzo, il tipo del servizio messo a disposizione e una URL, che permette di ottenere maggiori informazioni sul servizio.
3. Descrizione: Dopo che un punto di controllo ha scoperto il dispositivo, il control point stesso si occupa di ricercare ulteriori informazioni utilizzando l'URL contenuta nel messaggio di Discovery. La descrizione di un dispositivo è espressa utilizzando XML e include informazioni quali il numero di serie e il nome del dispositivo e altre dati utili al funzionamento del servizio stesso.
4. Controllo: Ottenute le informazioni descrittive del dispositivo, il punto di controllo deve richiedere informazioni dettagliate per utilizzare effettivamente il servizio. Anche in questo caso la descrizione dettagliata è in formato XML e include i comandi, le azioni e i parametri per utilizzare il dispositivo. Sono anche utilizzate una serie di variabili che descrivono lo stato del servizio in tempo reale. Per controllare il dispositivo, il control point invia comandi all'URL del servizio utilizzando il protocollo SOAP.
5. Eventi: Un punto di controllo, se interessato al cambio di stato di una variabile di un servizio, può richiedere di ricevere notifiche per ogni cambiamento. Il formato dei dati anche in questo caso è l'XML, mentre il protocollo preposto per gestire questo compito è GENEA.
6. Presentazione: Un dispositivo può avere una URL di presentazione che permette al punto di controllo di caricare la pagina corrispondente in un browser. Un eventuale utente può, in questo modo controllare il dispositivo e osservare il suo stato utilizzando un browser.

Se lo stadio di presentazione è presente e mette a disposizione un'interfaccia utente, UPnP può essere visto come una tecnologia leggera dal lato utente che utilizza XML per controllare servizi. È possibile quindi implementare UPnP anche in piccoli dispositivi che lavorino su una rete IP e abbiamo il supporto XML. UPnP fa anche parte della

nuova architettura di Microsoft .NET [.NET] che include una serie di nuove tecnologie che mettono in competizione diretta UPnP con l'accoppiata Java/Jini.

3.1.5 Home Audio Video Interoperability (HAVi)

La specifica HAVi nasce dall'accordo di otto produttori di sistemi audio/video per interconnettere e coordinare l'utilizzo delle risorse messe a disposizione dai loro prodotti. È stato quindi specificato un insieme di API, servizi e protocolli per garantire queste funzionalità. Il campo di applicazione di HAVi è strettamente locale, mirato a soddisfare le necessità molto specifiche del campo audio/video: elevate velocità di connessione, trasmissioni isocrone, configurazione automatica dei dispositivi e dei servizi. Per garantire tutto questo è stato scelto come mezzo di trasporto dei dati lo standard IEEE 1394, nato per garantire un'elevata banda passante (400 Mbps) e il supporto di connessioni a banda garantita e isocrone, ideali per il supporto di trasmissioni audio/video (vedi figura 3.12). IEEE 1394 garantisce l'implementazione dei livelli 1 e 2 del modello ISO OSI. A livello superiore, il livello di connessione è assicurato dal functional control protocol [IEC61883.1] che garantisce tutte le operazioni che riguardano la creazione e la gestione di connessioni isocrone tra dispositivi HAVi.

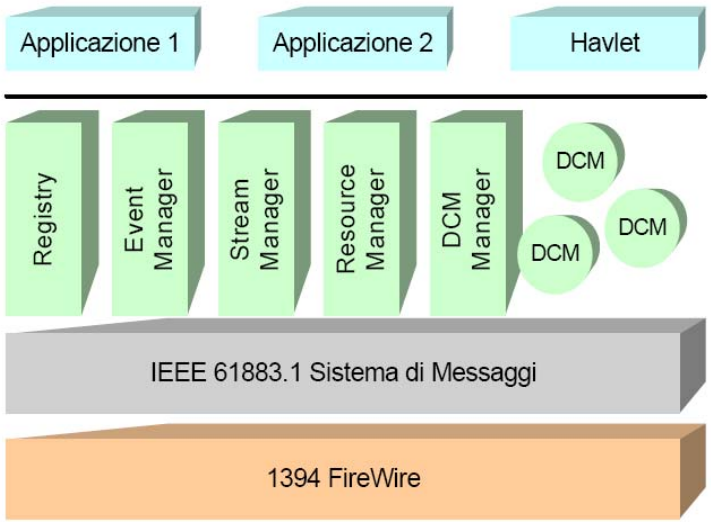


Figura 3.12 Architettura HAVi.

I livelli superiori dell'architettura non devono essere necessariamente tutti implementati all'interno della stessa periferica, HAVi differenzia i dispositivi a secondo dei servizi che sono in grado di contenere, stabilendo così a priori una differenziazione di complessità, e quindi anche di costo, all'interno dei dispositivi stessi. HAVi distingue i dispositivi in quattro categorie, suddivise tra loro dal livello di complessità e dalla capacità di controllare altri dispositivi:

- Full Audio/Video device (FAV): sistemi che contengono tutte le componenti software di un sistema HAVi. Sono in grado di eseguire bytecode Java e di distribuire codice ad altri dispositivi, aumentando così la capacità di controllo (DCM).
- Intermediate Audio/Video device (IAV): stesse caratteristiche dei precedenti, ma non hanno una Java Virtual Machine, costano quindi meno e non possono controllare eventuali nuovi dispositivi connessi.
- Base Audio/Video device (BAV): dispositivi che, per motivi di costo o risorse, non contengono alcun elemento dell'architettura HAVi. Hanno però la possibilità di eseguire bytecode Java.
- Legacy AV (LAV): sistemi che non supportano in alcun modo l'architettura HAVi. Utilizzano in genere protocolli proprietari che garantiscono il solo controllo.

I servizi messi a disposizione da un dispositivo FAV sono modellati come oggetti. Ogni elemento software è accessibile attraverso un'interfaccia specificata da HAVi e la sua esecuzione avviene sul dispositivo che lo mette in esecuzione. HAVi, inoltre, non specifica l'architettura hardware su cui devono girare i dispositivi, la scelta è lasciata ai produttori. I servizi necessari al funzionamento di HAVi sono però ben definiti (figura 3.12):

- Registry: un sistema di Directory che permette ad un elemento software di trovare altri componenti locali e determinarne capacità e attributi. Le informazioni sui servizi memorizzati nel Registry non sono permanenti, devono essere ricreati completamente all'accensione della periferica.
- Event Manager: implementa un sistema di invio eventi distribuito. Ogni volta che avviene un cambio di stato su una variabile di un sistema remoto, l'Event Manager controlla se un componente software locale è registrato per la sua gestione. Se ciò accade, viene attivato il sistema di messaggi per notificare il cambiamento.

- Stream Manager: componente responsabile del controllo di flussi di dati audio/video che intervengono tra unità funzionali.
- Resource Manager: permette di condividere risorse tra dispositivi e la programmazione di eventi futuri.
- DCM (Device Control Module): è un elemento software che rappresenta un dispositivo nella rete HAVi che specifica le interfacce di utilizzo del dispositivo stesso. Se un dispositivo viene inserito o rimosso dalla rete, il DCM per quel dispositivo deve essere istallato o rimosso. I DCM sono quindi componenti software dinamici e sono alla base della flessibilità e adattabilità che si vuole ottenere nei dispositivi FAV.
- DCM Manager: componente che è responsabile dell'installazione e rimozione dei DCM.

Le API di HAVi sono specificate utilizzando IDL (Interface Definition Language) in modo da permettere l'utilizzo di diversi linguaggi di programmazione. Il consorzio HAVi ha comunque selezionato Java come linguaggio preferenziale di sviluppo per la sua capacità di funzionare su differenti piattaforme hardware e per la sua diffusione. Naturalmente i dispositivi FAV, quelli più evoluti, devono avere una Java Virtual Machine (JVM) e implementare classi specifiche per il supporto ad HAVi. La filosofa di sviluppo delle applicazioni alla base di HAVi è molto simile a quella di Jini: rende possibile il trasporto di applicazioni su hardware differenti e permette la mobilità del codice tra i vari dispositivi. Quest'ultima caratteristica è fortemente utilizzata nelle applicazioni HAVi ed è utilizzata nelle seguenti componenti:

- DCM: Le DCM contenute all'interno delle ROM delle BAV sono in bytecode Java, sono così eseguibili all'interno dei FAV.
- Havlet: Un havlet è un'applicazione HAVi portabile ed è implementata nelle FAV e nelle DCM. Possono contenere un'interfaccia grafica che è utilizzata per controllare i dispositivi corrispondenti.
- Moduli applicativi: Un modulo applicativo è un wrapper necessario per istallare e rimuovere havlet. Possono essere scritti in Java.

3.1.6 Service Location Protocol (SLP)

SLP è uno standard della Internet Engineering Task Force (IETF) che gestisce il Discovery e l'advertising dei servizi, ma che non definisce alcun protocollo per la comunicazione tra client e i servizi stessi. SLP è, totalmente dipendente dal protocollo IP a differenza di Salutation (vedi paragrafo 3.1.2).

3.1.6.1 Architettura SLP

SLP è composto dalle seguenti entità, figura 3.13:

- Service Agents (SA): Un SA si occupa di pubblicizzare locazione e attributi dei servizi per conto delle componenti che li forniscono.
- User Agents (UA) . L'UA si occupa di trovare la locazione e gli attributi dei servizi richiesti dal software del client.
- Directory Agents (DA) . Hanno la stessa funzione del Lookup service di Jini, anche se in una rete SLP non è obbligatoria la loro presenza.

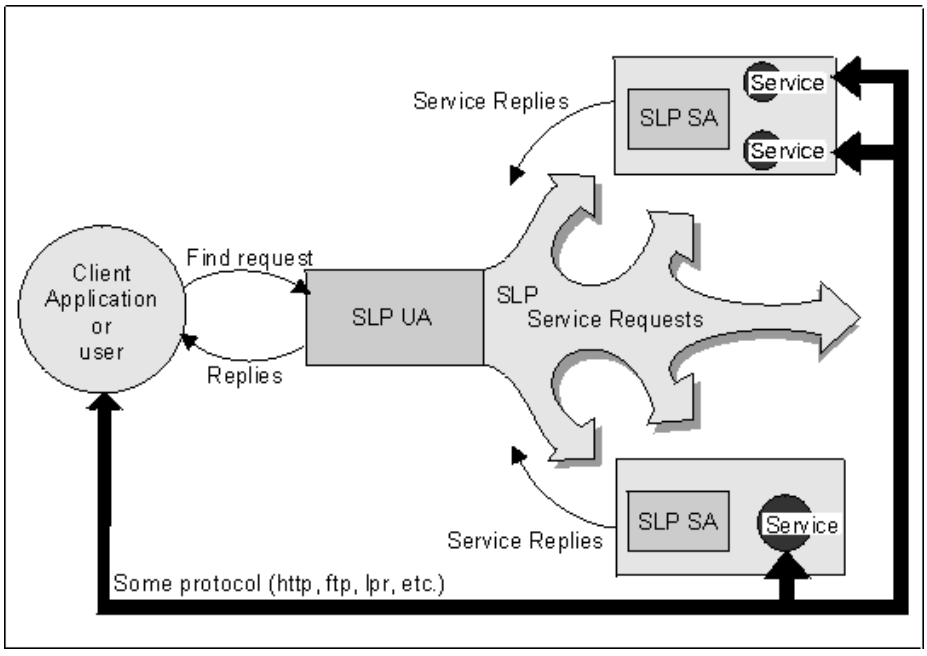


Figura 3.13: Struttura di SLP

La fase di Discovery di UA è iniziata dai SA. I SA interrogano un DA per avere le informazioni richieste, oppure ricorrono al multicast nel caso non vi sia alcun DA presente in rete. Nel primo caso i DA sono trovati attraverso lo sniffing di pacchetti multicast che periodicamente vengono inviati attraverso l'utilizzo del DHCP (Dynamic Host Configuration Protocol), oppure con l'invio di richieste SLP in multicast. Nel caso in cui non vi sia alcun DA, gli UA ricorrono alla comunicazione di gruppo per trovare i SA e ricevono riposte in unicast dai SA che controllano i servizi richiesti. Quest'ultimo modello aumenta l'utilizzo di banda, ma garantisce una fruizione più semplice delle risorse; è quindi molto adatto a reti poco estese.

Per quanto riguarda la sicurezza, SLP non definisce alcun protocollo per la comunicazione tra client e server, per cui SLP si preoccupa solamente di evitare la propagazione di false informazioni sulla locazione dei servizi. I Service Agents possono firmare elettronicamente quando registrano un servizio in un DA e gli UA possono verificare la loro identità.

3.2 Conclusioni e confronti

L'analisi delle tecnologie di Discovery e in generale di servizi e risorse svolta nei precedenti paragrafi permette di riassumere nella seguente tabella le caratteristiche distintive:

Nome	Produttore	Piattaforma	Linguaggio	Protocollo di rete
BlueTooth	Consorzio	Qualunque	Qualunque	LMP e IP
HAVi	Consorzio	Qualunque	Qualunque	FireWire
Jini	Sun Microsystems	Qualunque	Java	TCP/IP
Salutation	Consorzio	Qualunque	Qualunque	Qualunque
SLP	IETF	Qualunque	Qualunque	IP
UPnP	Microsoft	Qualunque	Qualunque	TCP/IP

Nome	Discovery	Announce	Registry	Interoperabilità	Sicurezza
BlueTooth	√	X	X	√	√
HAVi	√	√	√	√	√
Jini	√	√	√	√	√
Salutation	√	√	√	√	√
SLP	√	√	√	√	X
UPnP	√	√	X	X	X

Tabella 3.2 Relazioni tra le diverse tecnologie.

CAPITOLO 4

TECNOLOGIE UTILIZZATE: SOMA, MUM E JMF

In questo capitolo verranno introdotti gli strumenti che saranno utilizzati nella realizzazione della tesi, saranno evidenziati solo gli aspetti che maggiormente riguardano il progetto da realizzare, infatti non sarebbe possibile, in questo contesto, trattare in maniera completa tutte le caratteristiche delle tecnologie utilizzate, data la vastità degli argomenti.

Il capitolo è strutturato essenzialmente in tre parti. Nella prima parte verrà introdotto brevemente il concetto di mobilità di codice e degli agenti mobili, una volta noti questi concetti si prenderà in considerazione la piattaforma SOMA; questo ambiente rappresenta la base su cui poggia tutto il progetto. Nella seconda parte verrà introdotto MUM; un middleware costruito su SOMA, il cui intento è quello di fornire le infrastrutture e i servizi necessari per la realizzazione di applicazione per la consegna e la fruizione di materiale multimediale in un sistema distribuito. Tra i servizi offerti, MUM mette a disposizione delle applicazioni utente la possibilità di monitorare lo stato della sessione, cioè di verificare se i requisiti richiesti per la fruizione del materiale multimediale sono o meno rispettati e, in caso non lo siano, di attivare una fase di riconfigurazione. Nella terza parte si parlerà di JMF, che rappresenta un'estensione del linguaggio Java riguardante i tipi di dati multimediali e che permette di effettuare la distribuzione di video sulla rete in streaming.

4.1 SOMA: introduzione alla mobilità di codice e agenti mobili

In questo paragrafo ci poniamo l'obbiettivo di introdurre alcuni dei concetti fondamentali su cui si basano gli ambienti che verranno descritti nei successivi

paragrafi. Come prima cosa bisogna stabilire, cosa si intende per mobilità del codice. In linea generale si tratta di considerare il trasferimento non più dei soli dati, ma anche del codice, in modo che possa essere eseguito localmente dove è necessario. Questo permette di aggiungere adattabilità e flessibilità alle applicazioni distribuite. Per approfondire questi concetti è necessario introdurre alcune definizioni:

- Con *Execution Unit* (EU) intendiamo un flusso computazionale sequenziale, ovvero l'insieme del codice e del suo stato di esecuzione, come ad esempio un thread ed il suo contesto.
- Con *Computational Environment* (CE) intendiamo l'ambiente all'interno del quale le EU operano.

Occorre ora fare una distinzione tra mobilità debole e mobilità forte. Si parla di mobilità debole quando si ha la migrazione del solo codice tra due diverse EU che operano su diversi CE. Si parla invece, di mobilità forte, quando il trasferimento riguarda un'intera EU, cioè codice e stato di esecuzione, che migra da un CE ad un altro.

Il concetto di mobilità del codice coinvolge diversi paradigmi, anche se verrà considerato solamente quello relativo agli agenti mobili. Un agente mobile è in pratica un programma o un oggetto trasportabile, che, una volta lanciato da un'unità di partenza, si sposta all'interno dell'infrastruttura di rete fino a raggiungere il determinato nodo nel quale sono richiesti i suoi servizi. Dopo essere giunto a destinazione eseguirà il o i processi necessari per portare a termine il suo compito. Il requisito fondamentale di un agente mobile risulta essere, quindi, la sua mobilità. Un suo aspetto fondamentale è, inoltre, l'identificazione, effettuata tramite un sistema di nomi. Oltre agli agenti è necessario identificare i nodi, le risorse e gli utenti. La loro identificazione è fondamentale per realizzare una delle operazioni più importanti in queste tipologie di sistemi: la comunicazione fra gli agenti.

Perché sia possibile creare ed eseguire agenti mobili è necessaria la presenza di un sistema ad agenti mobili, un'infrastruttura che implementi il paradigma introdotto. Nel prossimo paragrafo andiamo a presentare l'ambiente che si occupa di questo aspetto.

4.1.1 L'ambiente SOMA

SOMA (Secure and Open Mobile Agent) [SOMA] è un ambiente ad agenti mobili realizzato presso il Dipartimento di Elettronica Informatica e Sistemistica (DEIS) dell'Università di Bologna. Di seguito verranno delineate solo le caratteristiche architetturali principali, senza entrare troppo nei dettagli realizzativi.

L'esigenza, a livello realizzativo, di SOMA è il superamento del problema dell'eterogeneità delle reti e la realizzazione di un meccanismo di migrazione per le applicazioni che su tale piattaforma intendessero appoggiarsi. Entrambi questi obiettivi sono stati realizzati tramite l'implementazione mediante il linguaggio Java.

Java è, infatti, uno dei linguaggi più adatti all'implementazione di agenti mobili, per diversi motivi:

- essendo interpretato, può eseguire su qualunque macchina possegga una Java Virtual Machine (JVM) in grado di trasformare il bytecode in istruzioni macchina, di conseguenza è portabile su diverse architetture software/hardware;
- è un linguaggio object-oriented che permette uno sviluppo modulare del codice per estensione ed ereditarietà, in questo modo il programmatore può scrivere i propri agenti con uno sforzo limitato a partire dalle classi messe a disposizione dalla piattaforma;
- prevede un meccanismo di caricamento dinamico delle classi anche da sorgenti remote e di collegamento dinamico all'applicazione corrente, permettendo così di caricare a tempo di esecuzione elementi del programma non previsti in precedenza;
- fornisce la possibilità di serializzare gli oggetti, cioè rappresentarli come uno stream di byte e di trasferirli sulla rete;
- ha caratteristiche di sicurezza built-in, la più nota è quella associata alle applet che, originariamente, potevano essere scaricate da un Web server, ma non avevano diritto di accedere alle risorse del sistema locale. Dalla versione 1.2 del linguaggio si ha un'architettura di sicurezza rivisitata, molto più flessibile ed espressiva, fatta di domini di protezione, controlli di accesso e permessi da associare sia a codice remoto che a codice locale.

Essendo un linguaggio interpretato, una parte dello stato di esecuzione rimane incluso nello stato dell'interprete, in particolare la posizione del program counter che punta

all'istruzione eseguita in quel momento, di conseguenza la cattura di tale stato diventa praticamente impossibile se non modificando l'interprete, ma questo, oltre a problemi intrinseci, porterebbe alla perdita della portabilità, che è una delle caratteristiche fondamentali di un sistema ad agenti.

Di conseguenza, siccome SOMA è scritto in Java, è un sistema a mobilità debole, in cui cioè, ciò che si muove non è l'immagine dello stato di esecuzione di un processo in un istante qualsiasi, ma un suo stato di esecuzione ben definito e creato a livello applicativo, facendo uso della terminologia già introdotta nel paragrafo precedente, si può dire che:

- il codice sono classi Java;
- le risorse sono oggetti Java;
- le execution unit (EU) sono thread Java;
- i siti (in SOMA chiamati place) corrispondono a macchine virtuali Java;
- le interazioni avvengono tramite chiamate di metodi, scambio di messaggi e, quando i componenti sono ospitati da JVM diverse, tramite scambio di comandi.

4.1.1.1 Astrazioni di località e topologia in SOMA

L'infrastruttura SOMA fornisce una gerarchia di astrazioni di località. Tali località, che vengono chiamate domini, sono adatte alla descrizione di diversi scenari di connessione e la loro idea è mutuata da Internet; esistono, infatti, diversi domini all'interno dei quali vengono distinti sotto-domini, fino a giungere al semplice nodo. Il mondo in cui vivono gli agenti è allora costituito di Place e Domini (vedi figura 4.1).

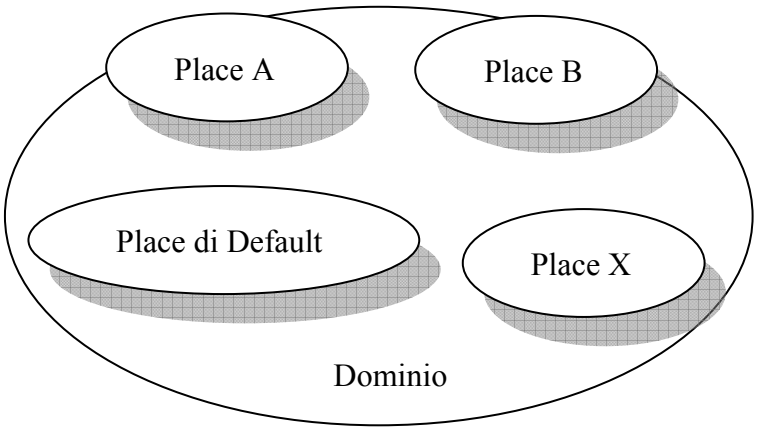


Figura 4.1 località logiche in SOMA.

I domini sono quindi organizzati secondo un'organizzazione gerarchica. Tutti gli elementi all'interno di un dominio sono infatti considerati "figli" di quel dominio, che di conseguenza ne è il padre. L'organizzazione che si viene a creare, è quindi quella di una struttura ad albero, in cui i Place normali costituiscono le foglie (cioè nodi terminali) mentre i Place di Default possono rappresentare sia nodi che foglie. Adottare questo tipo di topologia, permette l'identificazione univoca di un certo place/default-place, tramite il percorso che lo lega al default place "radice", cioè tramite la lista di DefaultPlace che devono essere attraversati, partendo dal DefaultPlace radice, fino al place/default place stesso. La struttura così delineata, deve corrispondere ad una rete reale, vediamo quindi ora qual è, in generale, la corrispondenza tra gli elementi della topologia e quelli di una struttura reale.

Il Place è il contesto di esecuzione dell'agente e sussume il concetto di nodo: il place può infatti corrispondere ad una macchina fisica, ma su un nodo possono convivere anche più place, permettendo, ad esempio, la definizione di località di protezione delle risorse. Anche se generalmente, viene assunto che un Place sia costituito da una postazione fissa, SOMA in realtà fornisce anche il supporto alle postazioni mobili; in particolare, tali postazioni sono rese tramite l'astrazione di un Mobile Place. Concettualmente il Mobile Place viene trattato come un Place normale quando si trova in un dominio ma è comunque necessario tenere presente che esso può spostarsi e registrarsi presso un altro dominio durante il corso della sua esistenza.

Il Dominio è un'aggregazione di place che può rispecchiare un contesto reale, come una LAN, oppure una caratteristica logica, ad esempio l'insieme dei dispositivi dello stesso tipo o appartenenti ad uno stesso dipartimento all'interno di un'organizzazione. Nell'implementazione, il concetto di Dominio si riscontra solo nella distinzione tra place generici e place cosiddetti di "default"; questi ultimi hanno conoscenza dei siti costituenti un dominio e sono punto d'accesso da e verso l'esterno.

Per quello che riguarda l'implementazione del sistema oggetto di questo lavoro di tesi, la corrispondenza fra place e nodi è di tipo 1:1. Il motivo di questa scelta è che in questo modo il place può essere utilizzato come utile astrazione di località fisica. Imponendo questa corrispondenza 1:1 il place rappresenta, per le applicazioni sviluppate al di sopra del middleware basato su SOMA, l'unico punto di accesso alle risorse di sistema di un certo nodo, risolvendo alcune delle problematiche di gestione

delle risorse. Ad esempio, come vedremo, alcuni servizi di base, il sottosistema per la prenotazione delle risorse, vengono realizzati a livello di place, in questo modo si ha, come è giusto, un unico gestore delle risorse per singola macchina. In particolare, nella presente tesi, determinati place di default saranno dotati di Access Point (AP) e fungeranno da punto di accesso per dispositivi mobili, sui quali, a causa delle loro limitate risorse, sarà presente un client leggero per l'accesso a SOMA. Inoltre, nei Default Place sarà inserito il sistema di discovery dei servizi presenti nel dominio, in modo che tutti i place figli possano venire a conoscenza dei servizi disponibili.

L'organizzazione gerarchica, adottando una topologia ad albero per i domini, permette inoltre l'identificazione univoca di un certo place/default place, tramite il percorso che conduce dal default place "radice" al place/default place stesso. In figura 4.2 riportiamo un esempio che utilizza una gerarchia di semplice comprensione. Il default place radice sarà il default place Mondo, e poi ci sono tutti i vari sotto-domini, ecc.; ad esempio il place "Toscana" è univocamente determinato dal percorso [Mondo, Europa, Italia, Toscana].

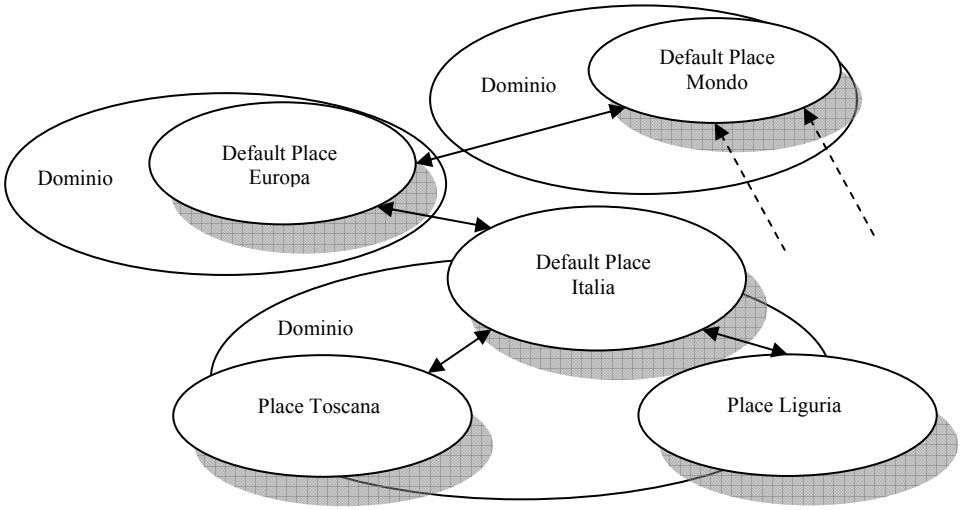


Figura 4.2 Organizzazione gerarchica dei Domini in SOMA

4.1.1.2 L' ambiente di esecuzione

Il place è l'ambiente di esecuzione dell'agente, realizzato mediante moduli di supporto che forniscono i servizi fondamentali e che si possono logicamente suddividere in:

1) *Agent Manager*: che gestisce gli agenti permettendone l'esecuzione, l'ingresso e uscita da un place; l'insieme delle funzionalità dell'Agent Manager non verrà sostanzialmente modificato nell'ambito di questo progetto; l'*Agent Manager* è tuttavia lo strumento attraverso il quale, l'agente ha accesso al ambiente del Place in cui si trova e pertanto, se si trova nel Default Place, anche alle informazioni relative al discovery, come accennato nel paragrafo precedente;

2) *Network Manager*: che gestisce le interazioni tra i place, mantenendo i necessari canali di comunicazione; la comunicazione avviene direttamente, esclusivamente tra Place che hanno visibilità diretta;

3) *Information Service*: che gestisce le informazioni sui place appartenenti ad un dominio e sui domini ad esso noti e si distingue in *DomainNameService* e *PlaceNameService*.

4.1.1.3 Identificazione di Agenti e Place

Ogni agente è associato ad un identificatore unico, *AgentID*, ricavato dall'identificatore del place sul quale nasce e da un intero progressivo. Questa scelta permette di conoscere sempre l'origine di un agente e, ad esempio, di poter inviarne la posizione corrente alla sua home, ovvero al place di instanziazione. Il naming di Place e Domini si basa sul *PlaceID*: identificatore frutto della concatenazione del nome del dominio e del nome del place. Nel caso di un place di default quest'ultimo sarà semplicemente nullo. Per quanto riguarda l'identificazione di un place mobile, questo viene preceduto da un prefisso costante che identifica tali place. Nel caso che tal place si sposti infine da un dominio ad un altro, il suo identificatore si modificherà tenendo conto del nuovo dominio in cui si trova.

4.1.1.4 Comunicazione in SOMA

La capacità di un agente di comunicare con altri agenti è una ulteriore caratteristica di base. Il meccanismo di base adottato per la comunicazione è quello dello scambio di messaggi. Tale scambio è possibile tra qualunque coppia di agenti, locali o remoti, anche nel caso che siano migrati in altri siti, purché però gli agenti stessi vengano creati come rintracciabili. Lo scambio di messaggi in sé, è basato sul concetto di MailBox e di Messaggio; un agente rintracciabile avrà quindi una propria MailBox da cui potrà leggere i messaggi (operazione bloccante) o anche solo controllare se ve ne sono, in modo da poter evitare un eventuale blocco. I messaggi, al loro volta saranno inviabili da qualunque agente che conosca l'identificatore dell'agente destinatario e in particolare ogni messaggio conterrà l'identificatore del mittente e del destinatario oltre, ovviamente al contenuto del messaggio che potrà essere un oggetto qualunque (ovviamente serializzabile date le caratteristiche di Java). Questa soluzione è molto flessibile, perché fornisce le basi per la creazione di schemi di comunicazione avanzati. Per quanto riguarda infine, la trasparenza, rispetto alla locazione di mittente e destinatario di un messaggio, essa è realizzabile grazie al servizio di localizzazione degli agenti offerto dal supporto e attivabile su necessità.

4.1.1.5 Sicurezza in SOMA

SOMA garantisce il rispetto di una politica di autorizzazione così che agenti maliziosi non interagiscano in modo incontrollato con le risorse e i servizi messi a disposizione dall'ambiente. La definizione di diverse astrazioni di località, inoltre, consente di intro-durre politiche di sicurezza nelle quali le azioni siano controllate sia a livello di dominio, sia a livello di place. Il dominio definisce una politica di sicurezza globale che impone autorizzazioni e proibizioni generali; ogni place, però, può applicare restrizioni ai permessi consentiti a livello di dominio. Questo modello di sicurezza si basa sui meccanismi offerti dal linguaggio Java. In particolare, il *ClassLoader* e la gestione delle politiche sono adattati alle specifiche esigenze di un sistema ad agenti; così, ad esempio, è stato definito un *AgentClassLoader* ad hoc. Agli agenti vengono

attribuiti specifici permessi in base alle azioni che devono compiere; ogni permesso è caratterizzato da un target, cioè una risorsa locale, e da un numero di azioni permesse. Esistono permessi predefiniti come, per esempio, quello di accesso ad un place (*PlaceAccessPermission*) e quello per ottenere il riferimento all'ambiente del Place (*AgentPermission*). Per consentire l'attribuzione dei permessi giusti ad ogni agente, il sistema deve identificare il relativo *Principal*, cioè l'entità che rappresenta il responsabile per l'agente (sia esso uno user, una compagnia ecc…). Tale *Principal*, in Java viene rappresentato mediante un *CodeSource* composto da un URL e da un insieme di chiavi crittografiche pubbliche. Le chiavi consentono la verifica delle credenziali associate all'Agente (che sono firme digitali applicate al codice). Una volta nota e autenticata l'identità del *Principal* di un agente, quest'ultimo possono essere associati i permessi previsti dalla politica del Place.

4.1.1.6 Dettagli della Piattaforma SOMA.

Si vuole ora brevemente illustrare come le principali caratteristiche di SOMA, evidenziate nei paragrafi precedenti, trovino una loro espressione a livello implementativo.

4.1.1.6.1 Gli Agenti

Un agente è realizzato come classe derivata dalla superclasse astratta Agent ed è un semplice oggetto passivo serializzabile, proprio perché la JVM non permette la migrazione di unità di esecuzione, quindi di thread. Così quando un agente nasce viene affidato ad un thread (*AgentWorker*) che gli associa un flusso di esecuzione, lo stesso avviene all'arrivo dell'agente su un nuovo nodo. L'attributo principale di un agente è il suo identificatore unico (*AgentID*) da cui è possibile dedurne l'origine, ovvero il place di instanziazione. Un altro attributo è la *Mailbox*, mezzo di invio/ricezione di messaggi ad altri agenti. L'agente può, infatti, essere creato *Traceable* oppure no. Da questo dipende la possibilità di rintracciarlo quindi di recapitargli i messaggi. Se un agente è

Traceable il gestore degli agenti del suo place di origine è responsabile della conoscenza della sua posizione corrente, l'informazione è mantenuta aggiornata grazie ad un meccanismo di notifica da parte di ogni place ricevente al place di origine. L'agente può interagire con l'ambiente di esecuzione solo tramite il campo *agentSystem* di cui è dotato: esso rappresenta l'interfaccia tra agente e sistema, cioè il punto di accesso a risorse e servizi. Si tratta di un campo *transient*, perché non sopravvive alla migrazione e viene assegnato all'agente dal place in cui di volta in volta si trova. Tutte le migrazioni avvengono tramite invocazione del metodo *go(PlaceID,String)* dove il programmatore deve specifica-re l'identificatore del place di destinazione e il nome del metodo da cui ripartirà l'esecuzione; in questo modo si riesce a simulare il controllo di flusso tipico della mobilità forte. La migrazione del codice dell'agente e di tutte le eventuali classi da questo riferite avviene ad opera di un *ClassLoader* specializzato, e cioè l' *Agent-ClassLoader*, che richiede trasferimento del codice da remoto solo su necessità (paradigma COD) e lo mantiene in una cache locale per utilizzi successivi.

4.1.1.6.2 L'accesso ai Servizi: l' Environment

Come anticipato il place è l'ambiente di gestione ed esecuzione degli agenti e, in quanto tale, deve rendere disponibili tutti servizi di comunicazione, naming, sicurezza,migrazione e discovery. Per consentire ciò, ogni Place fornisce un contenitore che raggruppa i gestori di tali servizi. Tale contenitore è l'*Environment* del Place (che è realizzato da un'omonima classe) e, di fatto si può identificare con il Place a tal punto che, a livello implementativo, l'instanziazione di un nuovo *Environment* comporta l'effettivo avvio di un place.

Possedere un riferimento all'*Environment* locale significa avere completo accesso alle funzionalità del supporto; ogni attributo di classe, infatti, corrisponde ad un fornitore di servizi logicamente correlati o ad un data base di informazioni (vedi figura 4.3).

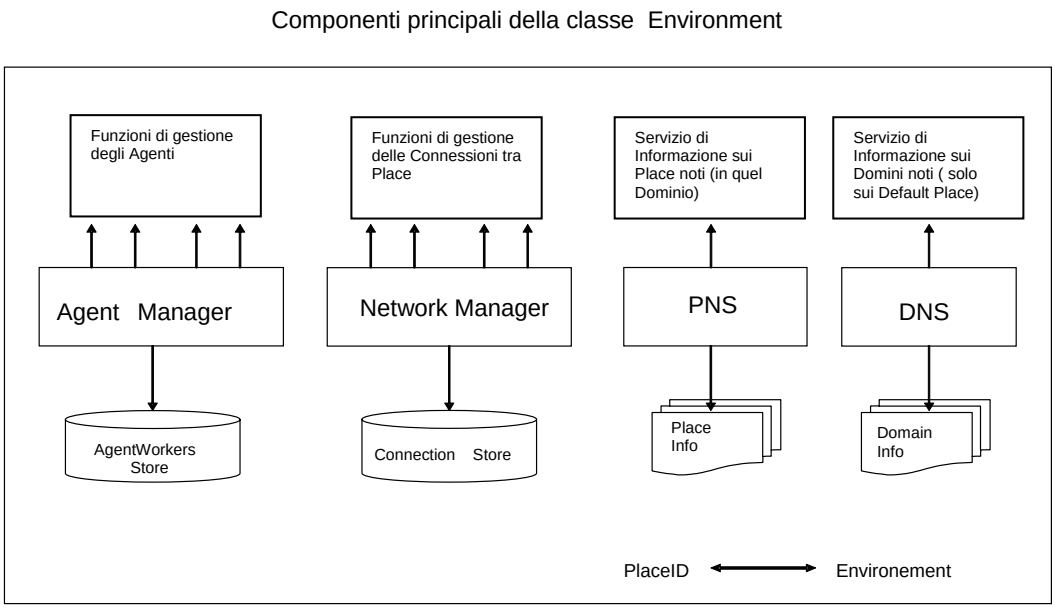


Figura 4.3: componenti principali dell'Environment

Risulta allora evidente perché non si sia stato reso direttamente accessibile dalla classe *AgentID*: è necessario discriminare gli accessi e lo si può fare a partire da *AgentSystem* che consente di creare "viste" differenziate dell'ambiente. Si vogliono ora descrivere gli attributi della classe *Environment* che corrispondono ad altrettanti moduli del supporto SOMA .

4.1.1.6.3 Gestore degli Agenti

L'attributo *AgentManager* dell'*Environment* riconduce al gestore degli agenti di un place, che permette di creare nuovi agenti, avviarli e risalire alla loro posizione attuale. È il responsabile dell'attribuzione di un flusso di esecuzione ad ogni agente presente sul place; si ricorda, infatti, che gli agenti sono oggetti passivi tali da poter essere serializzati e quando nascono o giungono su un place vengono assegnati dal supporto a un thread, che avvia l'esecuzione a partire da un metodo specificato e che termina quando si esce da tale metodo. L'*AgentManager* mantiene una struttura dati, instanza della classe *AgentWorkerStore*, in cui memorizza le associazioni tra *AgentWorker*, cioè thread, e *AgentID*; appoggiandosi alle funzionalità di questa classe permette anche di

serializzare gli agenti correntemente in esecuzione, rendendoli successivamente disponibili.

4.1.1.6.4 Informazioni su Place e Domini: l'Information Service

Come abbiamo anticipato, l'Information Service, è realizzato da due componenti: il *PlaceNameService* (PNS) e il *DomainNameService* (DNS); come è facilmente intuibile dai nomi stessi, il primo di questi componenti fornisce le informazioni su quelli che sono i place semplici che appartengono a un dominio del sistema mentre il secondo mantiene le informazioni su i Place che rappresentano l'astrazione di un Dominio e che vengono chiamati *DefaultPlace*. La presenza di due sistemi di nomi è dovuta, come abbiamo visto, all'organizzazione della topologia di SOMA. In particolare, si vuole mantenere limitata al solo dominio in cui si trovano, la visibilità per i Place normali. Per i *DefaultPlace* invece, la visibilità non viene estesa, tramite appunto al *DomainNameService*, a tutti gli altri *DefaultPlace* del sistema. Tale forma di visibilità, in realtà non rispecchia la struttura della topologia in quanto, in questo modo, ad un agente risultano accessibili, direttamente, tutti i *DefaultPlace* del sistema, come se vi fosse una connessione diretta tra questi (connessione che nella pratica può esistere ma che non è rispecchiata dalla topologia del sistema). In sostanza, ai Default Place viene impedita la vista degli elementi appartenenti a un dominio al di fuori del proprio, ma viene fornita la possibilità di comunicazione diretta con ogni altro *DefaultPlace* in una qualunque posizione nella topologia.

Entrambi il PNS e il DNS sono assimilabili a delle tabelle i cui elementi sono, nel primo caso, degli oggetti *PlaceInfo* mentre, nel secondo caso, degli oggetti *DomainInfo*; pur rappresentando entità concettualmente diverse, all'atto pratico *DomainInfo* e *PlaceInfo* sono sostanzialmente uguali in quanto le informazioni che mantengono sono: l'identificatore del Place a cui si riferiscono (il *PlaceID*) e l'indirizzo di rete (nel nostro caso indirizzo ip e porta) su cui il Place è in ascolto per le comunicazioni con gli altri Place. La differenza fondamentale quindi tra i due servizi di nomi è sul tipo di visibiltà che offrono della topologia di SOMA in quanto all'atto pratico le stesse informazioni descrivono sia un dominio che un Place comune. Affinché un place entri a far parte di un dominio è necessario che il suo PNS si registri presso il PNS del place di default,

non esiste ancora, infatti, un aggiornamento automatico all'atto della creazione del place.

I DNS racchiudono conoscenza relativa ai domini esistenti nel mondo degli agenti e, rappresentando un'astrazione di livello superiore, sono mantenuti solo su place di default. È prevista una gerarchia di DNS cui corrisponde una propagazione di informazioni dal basso all'alto e viceversa: ogni DNS ha un DNS padre presso cui si è registrato e può avere DNS figli che si sono registrati presso di lui. Oltre ai metodi per la registrazione iniziale, esistono metodi cosiddetti di refresh per sollecitare un aggiornamento delle tabelle di PNS/DNS. Tale gerarchia è visualizzata in figura 4.4.

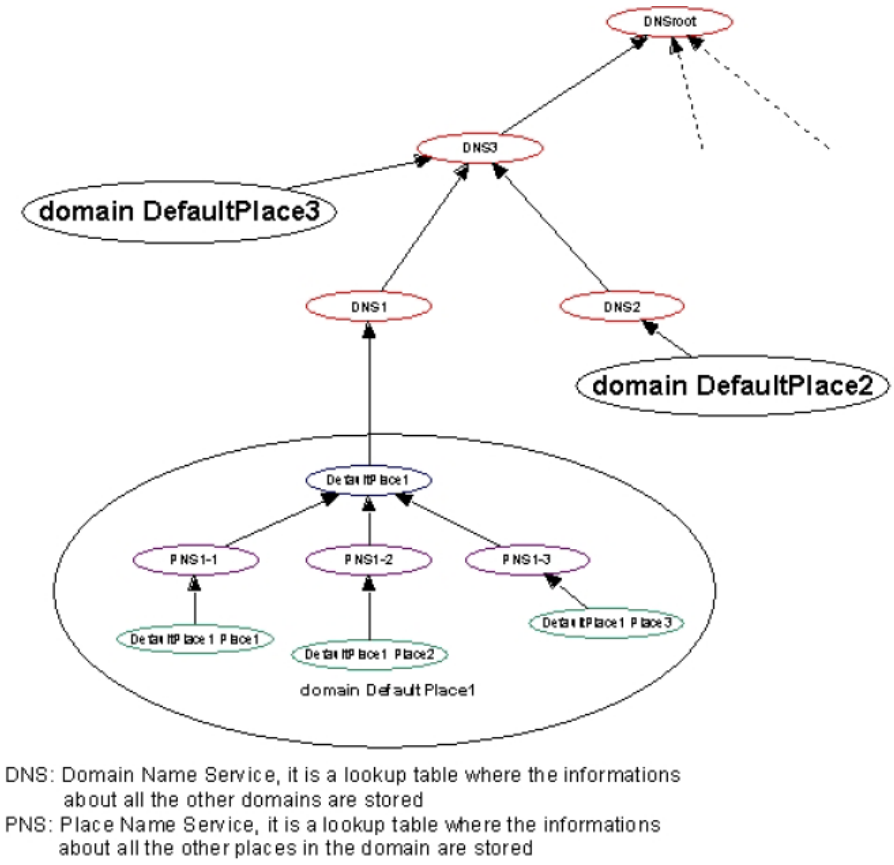


Figura 4.4 organizzazione di PNS e DNS.

4.1.1.6.5 Interazione tra Place: gli oggetti Command

Le interazioni tra place diversi avvengono tramite normali comunicazioni via socket, ma ciò che è particolare è l'oggetto dello scambio: un comando. I comandi derivano dalla classe astratta *Command* che ne presenta l'interfaccia di riferimento. Il metodo start() è quello invocato da un place all'atto della ricezione, al suo interno si ha la creazione di un nuovo thread cui si affida l'esecuzione del metodo run(), dove viene racchiusa la computazione associata al comando concreto (paradigma remote evaluation). La registrazione di un place presso il place di default può essere un esempio di utilizzo di comando: il PNS richiede al *NetworkManager* l'invio di un'istanza di una particolare sottoclasse di *Command*, il *PlaceRegisterCommand*; questo comporta il recupero o la creazione della *Connection* verso il place di default, per poi invocarne il metodo *send(Command)*. All'altro capo della comunicazione il demone responsabile (un'altra istanza di *Connection*) accetterà il comando e ne avvierà l'esecuzione che comporterà una chiamata di registrazione al PNS locale.

4.1.1.6.6 Gestore di Rete

La gestione delle comunicazioni relative ad un place è logicamente a sé stante ed anche nell'implementazione è stata associata ad una classe specifica: il *NetworkManager*, anch'esso raggiungibile tramite l'*Environment*. Il *networkManager* ha conoscenza delle connessioni stabilite con altri place dello stesso dominio grazie alla struttura *ConnectionStore*, una sorta di contenitore delle associazioni tra connessioni, istanze della classe Connection, e identificatori di place coinvolti. All'inizializzazione di un place viene anche creato un *ConnectionServer*, demone che attende richieste ed attiva connessioni. A loro volta le istanze di Connection non sono altro che demoni responsabili delle comunicazioni via socket con un altro place. Oggetto della comunicazione gestita dai networkManager sono i Comandi presentati nel paragrafo precedente. Ogni comando viene indirizzato verso un particolare Place di destinazione; il network manager reperisce le informazioni relative al Place tramite

l'*InformationService* del Place ed eventualmente richiede l'attivazione di una connessione con tale Place. Una volta stabilita la connessione invia quindi il comando.

Nel caso in cui il comando sia invece indirizzato verso un Place non conosciuto dall'*InformationService* (caso di un Place normale di un altro dominio), il *networkManager* genera un Comando di Trasporto (*TransportCommand*); tale comando, ingloba il comando originario e viene spedito verso il *DefaultPlace* proprietario del dominio in cui è presente il Place non noto all'*InformationService*. Una volta giunto sul *DefaultPlace*, il *TransportComand* provvede a rispedire il comando originario al Place giusto (questa volta sicuramente noto in quanto si è nel suo dominio).

4.2 MUM

MUM (Mobile agent-based Ubiquitous multimedia Middleware), sviluppato presso il Dipartimento di Elettronica Informatica e Sistemistica (DEIS) dell'Università di Bologna, è una infrastruttura nata per fornire supporto allo sviluppo di applicazioni multimediali fornendo ai suoi utenti accessibilità ai servizi non solo in modo indipendente dai loro movimenti e dal luogo in cui accedono alla rete, ma anche considerando i vincoli tecnici imposti dal terminale che stanno utilizzando [MUM].

La versione di MUM utilizzata per la presente tesi, incorpora un sistema di adattamento che permette di personalizzare i contenuti multimediali inviati ad un cliente in modo che il dispositivo utilizzato sia sfruttato in base alle caratteristiche di visualizzazione che è in grado di offrire. Non è presente invece un sistema di *handoff* ossia la gestione del movimento di un terminale da un AP ad un altro e nemmeno quello che viene chiamato *offload*, ossia la possibilità di spostarsi da un dispositivo mobile a uno fisso per migliorare la visione del contenuto multimediale, in particolare facendo ricorso ad un sistema di discovery che permetta al sistema un certo grado di decisione per la locazione fissa di destinazione.

Questo capitolo non ha l'obiettivo di presentare MUM in modo esaustivo, ma solo di esporre le sue caratteristiche principali al fine di comprendere ciò che la piattaforma mette attualmente a disposizione e che può dunque essere utilizzato per lo sviluppo del progetto di tesi.

4.2.1 Caratteristiche di MUM

L'architettura distribuita di MUM è basata su un modello computazionale misto che cerca di integrare i vantaggi del tradizionale modello cliente/servitore con quello ad agenti mobili. MUM infatti è stato costruito al di sopra della piattaforma SOMA, presentato nel paragrafo precedente. Tuttavia, pur disponendo di una tecnologia per il supporto degli agenti mobili, non tutte le entità di MUM sono state realizzate come tali. Non conviene, infatti utilizzare questo potente paradigma di comunicazione in modo generalizzato per vari motivi. Primo fra tutti la richiesta di un ambiente di esecuzione uniforme sul quale far eseguire il codice mobile. Inoltre, il fatto che lo stesso codice mobile dovrà accedere alle risorse del sistema verso cui è migrato pone non pochi problemi di sicurezza, monitoraggio e autenticazione. Molti dei servizi offerti da MUM sona basati sul principio di *location-awarness* cioè richiedono che la presenza di un supporto in grado di fornire la nozione di località fisica. A questo problema viene incontro SOMA che definisce una gerarchia di astrazioni di località adatta alla descrizione di qualunque scenario di connessione, da quello di un rete estesa e aperta come Internet a quello di una rete locale LAN (Local Area Network) come descritto nel paragrafo 4.1.1.1.

4.2.2 Architettura di MUM

In Figura 4.5 viene rappresentata l'architettura di MUM, evidenziando i servizi di supporto offerti. Essa è composta da due livelli:

1. Il *Mechanisms Layer*, che realizza i servizi di base;
2. Il *Facilities Layer*, che incapsula le strategie e i servizi middleware direttamente utilizzabili dalle applicazioni costruite al di sopra di MUM.

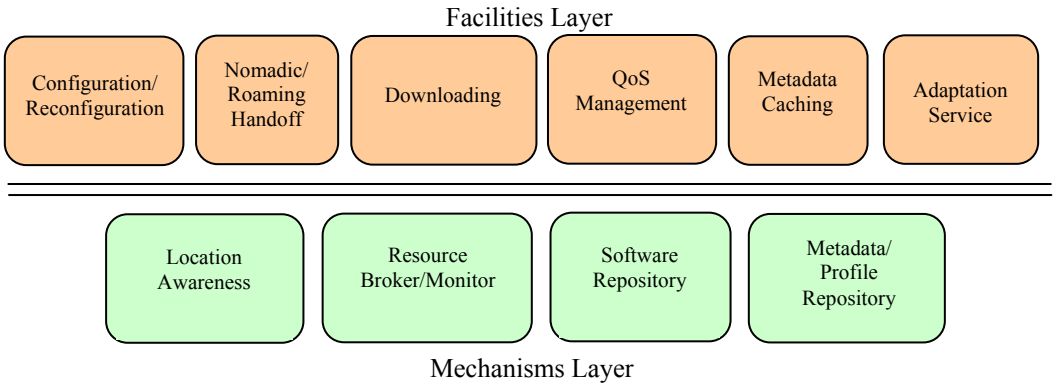


Figura 4.5: Architettura a livelli di MUM

Nel livello più basso, quello dei meccanismi, troviamo servizi essenziali come:

1. *Location Awareness*: Fornisce ai servizi del livello superiore visibilità sulla posizione dei nodi della rete, attraverso l'uso delle astrazioni di località di SOMA;
2. *Resource Broker/Monitor*: Ha il compito di effettuare il monitoraggio delle risorse;
3. *Software Repository*: Risorsa che mantiene il codice dei componenti che possono essere scaricati in fase di configurazione dei servizi;
4. *Metadata/Profile Repository*: Mantiene informazioni relative ai contenuti disponibili all'interno del sistema (ad esempio metadati sulle presentazioni) e ai profili dei clienti.

Questi meccanismi vengono utilizzati nello sviluppo dei servizi di supporto che MUM offre alle applicazioni e che si trovano nel *Facilities Layer*:

1. *Configuration/reconfiguration*: Servizio per la configurazione del cammino che permette al servitore di raggiungere il cliente. Nello scenario di fruizione il client viene collegato a un server che dispone del contenuto richiesto attraverso un percorso che può coinvolgere nessuno, uno o più nodi intermedi detti proxy. Questo percorso, chiamato Service Path (SP), attivo e riconfigurabile, è molto utile nella distribuzione dei vari componenti necessari alla elaborazione e al monitoraggio dei flussi inviati verso il cliente. Il servizio di configurazione si occupa quindi dell'inizializzazione dei nodi del SP affinché un certo servizio possa essere espletato. Inoltre, esso è anche in grado di gestire gli aspetti di negoziazione relativi alla gestione della qualità di servizio sia prima che durante l'erogazione del servizio richiesto;

2. *Nomadic/Roaming Session Handoff*: Questo servizio che verrà implementato nella tesi, ha lo scopo di realizzare il supporto necessario a garantire continuità alla sessione utente. Quando, durante l'erogazione del servizio, l'utente si muove col proprio terminale mobile o comanda esplicitamente lo spostamento della propria sessione su un altro terminale, il sistema deve compiere delle operazioni che il letteratura vengono chiamate operazioni di gestione del "*Session handoff*". Esse mirano a disconnettere i componenti applicativi dalle risorse del vecchio ambiente e a ricollegarle a quelle del nuovo in modo del tutto trasparente per l'utente. Il compimento di questa operazione è di fondamentale importanza per la preparazione dell'ambiente in cui la sessione dell'utente verrà spostata ed eseguita;
3. *QoS Management*: Incapsula le strategie per la gestione della qualità di servizio. Questo servizio collabora con il Monitor delle risorse, per verificare le condizioni generali della rete di comunicazione;
4. *Downloading*: Servizio per la ricerca e il dowloading del codice utilizzabile per la configurazione dei nodi che appartengono al Service Path;
5. *Metadata Caching*: Servizio per il caching distribuito dei metadati relativi ai contenuti multimediali presenti nel sistema;
6. *Adaptation Service*: Questo servizio adatta i contenuti multimediali alle caratteristiche del dispositivo di visualizzazione, interagendo con un database di profili di dispositivi per ricavare tali caratteristiche. Nella versione attuale di MUM non sono supportati l'*handoff* del terminale e l'adattamento dinamico nel caso si voglia passare ad un altro terminale con caratteristiche di visualizzazione diverse, obbiettivo della tesi è includere anche queste funzionalità.

4.2.3 Fruizione del materiale multimediale

Il modello di fruizione dei contenuti multimediali proposto da MUM introduce cinque entità fondamentali: *Client*, *Server* e *Proxy* realizzate come entità fisse, *ClientAgent* e *ProxyAgent* realizzate come entità mobili [Fos03]. Di seguito vengono analizzati brevemente il ruolo e le caratteristiche di ognuna di queste entità.

- *Client*: rappresenta l'end-point del *Service Path* che riceve i flussi multimediali richiesti. Per sua natura questa entità è fissa e risiede sulla macchina dalla quale l'utente effettua l'accesso al sistema. Il fatto che il Client non sia un agente mobile non implica che il software necessario per la sua esecuzione debba essere già presente sulla macchina dal quale verrà lanciato, infatti potrà essere utilizzato il servizio di *downloading* per scaricare a *run-time* tutti i componenti di cui si ha bisogno. Nella presente tesi i client saranno presenti in due versioni, una versione che si basa sull'infrastruttura sottostante SOMA e MUM ed una versione più leggera adatta ad un dispositivo portatile;
- *Server*: rappresenta l'altra estremità del *Service Path*, cioè l'entità che trasmette il contenuto multimediale comandato dall'utente. Così come il Client, anche il Server potrà usufruire del servizio di *downloading* per essere inizializzato con tutto il software di cui ha bisogno. Inoltre, una volta che un certo tipo di server viene lanciato su una macchina, viene utilizzato per servire tutte le richieste che da quel momento in poi arriveranno per quel tipo di server;
- *Proxy*: le due entità precedentemente presentate sono parte del ben noto modello Client/Server, il Proxy invece è un'entità non prevista da questo modello e ricopre un ruolo fondamentale nel modello computazionale di MUM. Il Proxy, posizionato sui nodi intermedi del *Service Path*, è un'entità che partecipa attivamente alla consegna del servizio richiesto. Nell'implementazione attuale di MUM, esso si occupa di:
    - Gestire l'intermittenza delle connessioni al Server. Questo problema, molto importante soprattutto per i dispositivi che accedono al sistema attraverso rete mobile (non sempre in grado di garantire una copertura permanente di tutto il territorio), viene risolto utilizzando il Proxy come una cache in cui memorizzare l'oggetto multimediale richiesto dall'utente: Partecipare attivamente alla gestione della qualità di servizio, monitorando le condizioni delle risorse e chiedendo, se necessario, l'eventuale riconfigurazione del servizio in caso di malfunzionamenti o in presenza di situazioni di sovraccarico della rete.
    - Servizio di adattamento dei contenuti multimediali, uno o in generale più Proxy sono responsabili della trasformazione del flusso multimediale in un formato adeguato alle caratteristiche del terminale cliente che lo ha richiesto.

- *ClientAgent*: Rappresenta l'entry-point del sistema dal lato cliente. Il suo compito è inizializzare la sessione per l'utilizzatore e poi gestire le interazioni con il resto del middleware. Svolge dunque un ruolo di mediatore che accetta le richieste fatte dall'utente attraverso un'opportuna interfaccia grafica permettendogli di interagire con MUM. Questa entità è stata realizzata come agente mobile per modellare il movimento dell'utente verso un altro terminale (nomadic user), e quello dell'utente insieme al proprio terminale verso un'altra area (roaming user).
- *ProxyAgent*: Agente mobile introdotto per supportare il movimento dei terminali che per la gestione vera e propria dei flussi si avvale dell'entità Proxy precedentemente introdotta.

4.2.4 Configurazione dinamica del sistema

In ambienti molto eterogenei non si può assumere che tutti i nodi che partecipano nel *Service Path* dispongano dei componenti necessari all'esecuzione di un certo servizio. La possibilità di scaricare codice solo al bisogno potrebbe portare notevoli benefici soprattutto nei casi in cui i nodi sono dispositivi con risorse limitate. Tuttavia, anche quando il *Service Path* è composto da macchine altamente performanti è utile disporre di un servizio di inizializzazione per evitare problemi legati all'indisponibilità dei componenti, sia di tipo applicativo che di tipo middleware, necessari per lo svolgimento del servizio richiesto. Per questi motivi, MUM offre un servizio di supporto per la configurazione dinamica del sistema. Quando viene richiesto un servizio, prima viene scaricato il software necessario alla sua trasmissione e ricezione, poi viene configurato il *Service Path* (scaricando, se necessario, il software indicato all'interno di un piano di inizializzazione) ed infine si inizia la sua erogazione.

In Figura 4.3 viene mostrata l'architettura del servizio di configurazione proposto evidenziando i meccanismi che vengono utilizzati per la sua realizzazione. In particolare, si può osservare che esso ha bisogno di definire e riconoscere le località dei partecipanti in modo da poter scaricare su ognuno il codice opportuno, inoltre si serve di un *Resource Manager* per supportare la riconfigurazione dinamica del sistema che viene

garantita mediante la generazione di più piani di inizializzazione alternativi che possono essere utilizzati a *run-time* per riorganizzare il sistema.

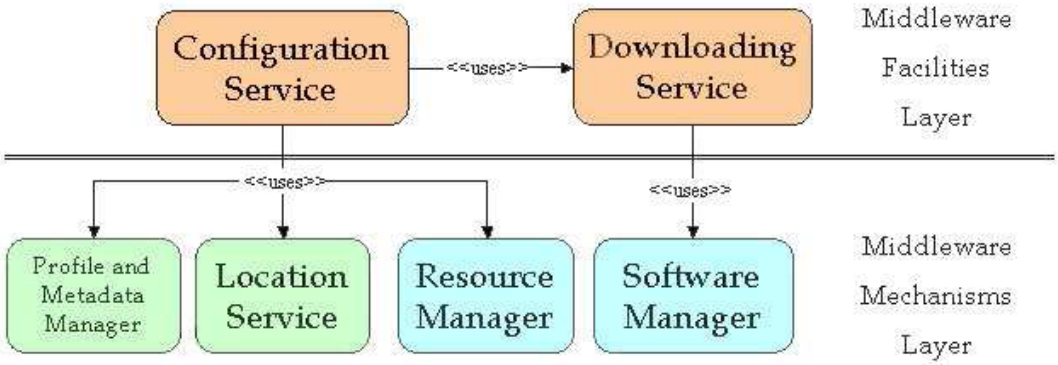


Figura 4.6 Architettura del servizio di configurazione proposto da MUM

Il servizio di configurazione viene realizzato attraverso due componenti middleware:

1. Il Decision Maker (DM), che incapsula le strategie che permettono di decidere come configurare il Service Path, producendo un piano che contiene una o più soluzioni corrispondenti a diversi livelli di qualità di servizio.
2. Il Plan Visitor Agent (PVA), agente mobile che viene spedito lungo il Service Path e che ha il compito di effettuare su ogni nodo la configurazione prevista dal piano generato e consegnatogli dal DM.

In Figura 4.7 viene mostrato lo scenario di configurazione: alle ricezione di una richiesta, il *ClientAgent* richiede l'inizializzazione del sistema indicando il DM da utilizzare per la generazione del piano di configurazione e il titolo che identifica il contenuto multimediale richiesto. Il servizio di configurazione, una volta ottenuto il piano dal DM, lo passa a un PVA che attraversa il *Service Path* dal *Client* al *Server* per operare su ogni nodo:

- La negoziazione e la prenotazione delle risorse;
- Il *downloading* del codice necessario;
- L'inizializzazione di tutti i componenti necessari;

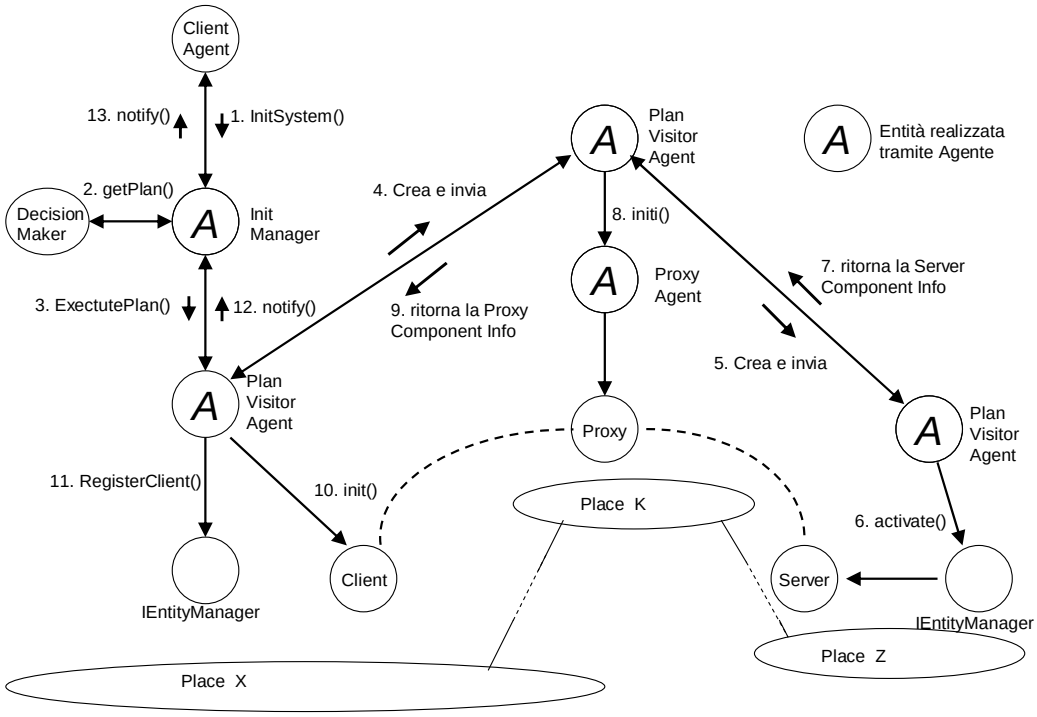


Figura 4.7: Schema di configurazione del sistema

Il servizio di inizializzazione, quindi, provvede anche alla gestione della qualità di servizio. Appena giunto in un nuova località, infatti, il PVA per prima cosa vede se ci sono risorse sufficienti per la presentazione richiesta. In caso affermativo continua il processo di inizializzazione inviando un altro PVA sul nodo successivo, e se ci sono sufficienti risorse su tutto il *path*, il processo di inizializzazione termina con successo. Se al contrario in un certo place non sono disponibili sufficienti risorse, il PVA consulta il proprio piano di inizializzazione per vedere se esistono presentazioni alternative, a qualità più bassa e tenta di instanziare il percorso per tali presentazioni.

4.2.5 Mobilità di utenti e terminali

MUM considera due tipi di mobilità:

1. *Nomadic-mobility*: mobilità degli utenti da un terminale all'altro;
2. *Roaming-mobility*: mobilità degli utenti insieme ai dispositivi che stanno utilizzando.

Nel primo caso, rappresentato in Figura 4.8, in risposta alla necessità dell'utente di cambiare il terminale dal quale sta consumando il servizio per spostarsi su un altro verso il quale si sta spostando fisicamente, MUM in modo trasparente all'utente trasferisce la sua sessione in modo che quando l'utente arrivi sul nuovo terminale possa continuare a fruire il materiale multimediale senza alcun ulteriore intervento da parte sua, in particolare il lavoro svolto si concentrerà sul cambio da un terminale mobile ad uno fisso e il tutto verrà integrato con un sistema di discovery che permetterà al sistema di scegliere il terminale di destinazione in base alle preferenze dell'utente e alla sua posizione attuale.

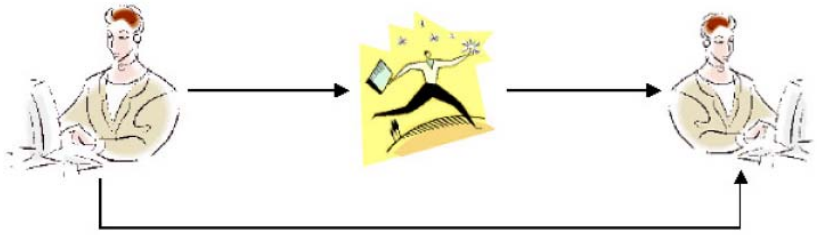


Figura 4.8: Modello di Nomadic-mobility

Nel secondo invece, rappresentato in Figura 4.9, MUM assiste il movimento del cliente facendo in modo che l'infrastruttura si modifichi a *run-time* in modo da seguire i movimenti dell'utente.

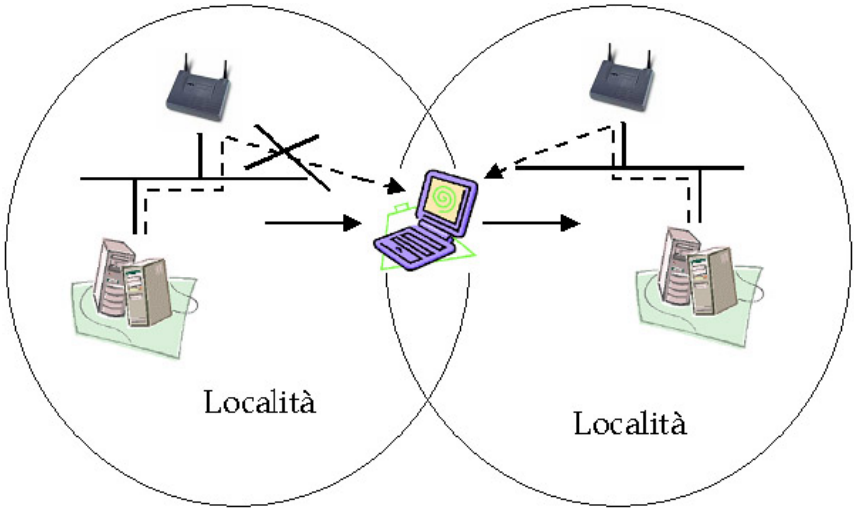


Figura 4.9: Modello di Roaming-mobility

In entrambi i casi è necessario effettuare un *re-binding* dinamico dei componenti di servizio, ed è compito del middleware compiere questa operazione in modo trasparente

all'utente. L'approccio utilizzato da MUM nella realizzazione del servizio di *re-binding* per il caso di utente nomadico è rappresentato in Figura 4.10: la richiesta dell'utente, catturata dal *ClientAgent*, viene passata all'*InitManager*, un componente middleware presente su ogni nodo, che ottiene dal DM un piano che descrive le azioni da compiere per spostare la sessione. Questo piano viene consegnato a un PVA che invia un altro PVA verso il nodo di destinazione della sessione. Una volta che quest'ultimo è arrivato scarica, se necessario, il codice ed inizializza il nuovo Client collegandolo al Proxy. Quando il "*Session handoff*" termina viene rispedito al place di partenza un messaggio che avvia la migrazione fisica del *ClientAgent* verso il nodo di destinazione. Questo protocollo garantisce continuità di sessione, infatti quando avviene la migrazione il nuovo *Client* è già pronto permettendo di effettuarla senza interrompere l'erogazione del servizio. La gestione dei *roaming-users* non è presente nella versione di MUM utilizzata, la tesi si occuperà di aggiungere anche questa funzionalità usando un approccio molto simile a quello esposto precedentemente per i *nomadic-users*, solo che questo tipo di mobilità non verrà modellata come migrazione del *ClientAgent*, ma di uno o più *ProxyAgent* situati nelle vicinanze del *Client*.

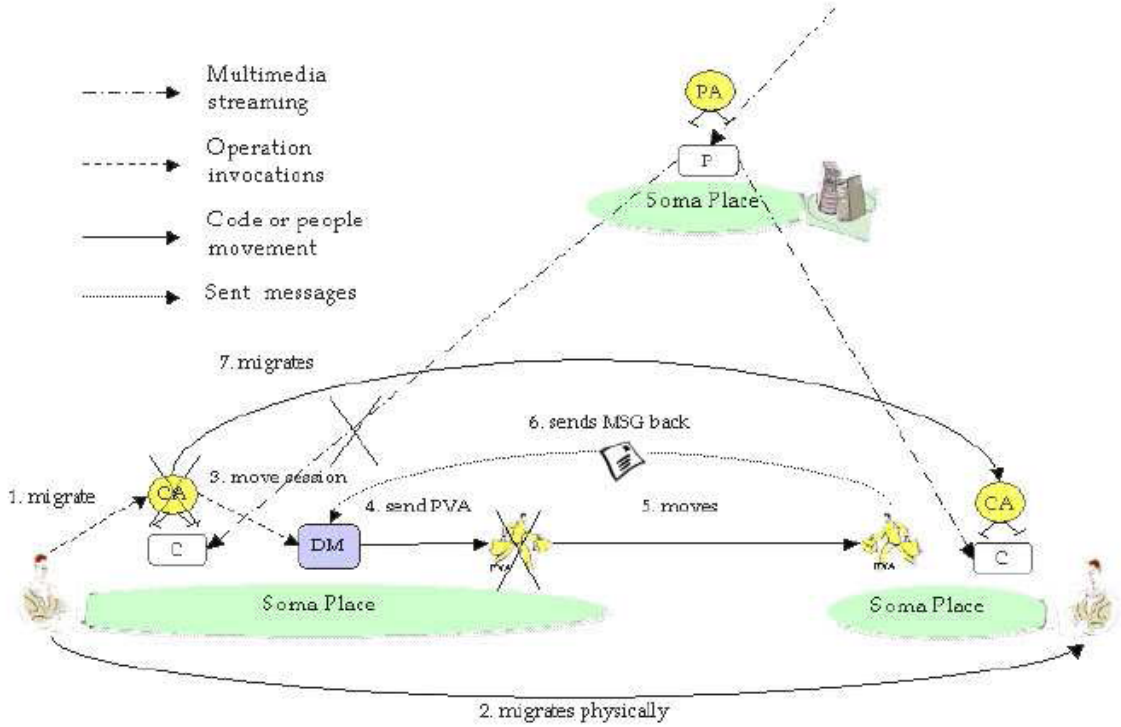


Figura 4.10: Protocollo per garantire continuità di sessione in MUM

4.2.6 MUM e Supporto all'Eterogeneità

MUM fornisce un supporto per adattare le presentazioni multimediali alle esigenze e limitazioni imposte dai terminali utenti. Il supporto di adattamento dei contenuti opera in accordo con altri due servizi, la gestione dei metadati e la gestione dei profili utente. La gestione dei metadati consiste nel reperimento delle presentazioni multimediali presenti nel sistema, i suoi compiti principali sono:

- Fornire al sistema di adattamento un set di presentazioni sui cui lavorare sufficientemente ampio, in modo che le sue scelte vengano svolte sulla base del maggior numero di alternative possibili.
- La considerazione dell'importanza delle risorse di rete. L'accesso alle cache dei metadati dell'intero sistema deve essere regolato da politiche anche quando si fanno operazioni di recupero.

Il servizio per la gestione dei profili si occupa invece di fornire:

- Una rappresentazione standard dei profili, in modo che questi siano interpretabili in modo univoco da qualunque altro sistema.
- Un database all'interno del quale memorizzare i profili degli utenti e sul quale si possa effettuare operazioni per il loro inserimento, eliminazione e recupero.
- Tradurre il profilo espresso in formato standard e di estrapolare da esso le informazioni richieste dal sistema di adattamento.

Il servizio di adattamento svolge quindi le seguenti operazioni:

- Sceglie, fra tutte le presentazioni corrispondenti al titolo richiesto che sono state recuperate dal sistema, quelle che considera convenienti da adattare in base a una politica di adattamento predefinita dallo sviluppatore che ha costruito la propria applicazione multimediale al di sopra di MUM.
- Sulla base delle condizioni di carico del sistema, sceglie il place, o in generale i place, sul quale effettuare l'adattamento.
- Sceglie gli eventuali adattatori da utilizzare per trasformare il flusso multimediale richiesto nel formato considerato più idoneo alle caratteristiche del dispositivo utilizzato dal client.
- Effettuare la trasformazione vera e propria del flusso.

I tre moduli sopra evidenziati sono riassunti in figura 4.11.

Figura 4.11: Moduli che si occupano dell'adattamento.

Il Modulo principale, quello che si occupa dell'adattamento, dovrà svolgere essenzialmente due tipi di attività: di decisione e di codifica. Questo modulo è diviso in due componenti:

- L' *Adaptation Engine*, rappresentato nello schema di Figura 4.12, è il componente che si occupa di elaborare le informazioni sui metadati e sui profili, per decidere in base alla politica di adattamento corrente su quali presentazioni operare (eventualmente) l'adattamento.

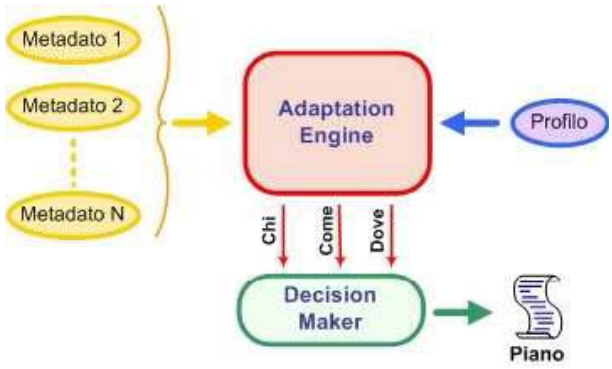


Figura 4.12: Ruolo dell'Adaptation Engine nel sistema di adattamento

  Esso gioca un ruolo fondamentale all'interno del sistema di personalizzazione poiché, imporrà al DecisionMaker le proprie scelte per la generazione dei piani di configurazione del servizio di erogazione. Le sue scelte inoltre riguardano anche la scelta degli adattatori da utilizzare per la eventuale trasformazione del flusso e le definizione di dove tale operazione dovrà essere svolta.

- L' *Adaptor* è invece il componente che si occupa di effettuare la traduzione di un flusso multimediale da un certo formato ad un altro scelto dall'Adaption Engine.

I due componenti lavorano su livelli diversi, infatti mentre il primo opera su metainformazioni (profili e metadati), il secondo elabora le informazioni vere e proprie (flussi multimediali).

4.3 Java Media Framework (JMF)

Come accennato nell'introduzione di questo capitolo si tratta di un prodotto opzionale che estende il linguaggio Java, ed in particolare la piattaforma JAVA2SE™, consentendo le gestione di tipi di dati multimediali. Il Java Media Framework è sostanzialmente costituito da un insieme di API (Application Program Interface) creato appositamente per consentire di incapsulare dati di tipo multimediale in applicazioni o applet Java. Sun e IBM hanno sviluppato questo componente con l'idea di fornire supporto ai più comuni standard di memorizzazione di contenuti multimediali, come ad esempio: MPEG-1, MPEG-2, QuickTime, AVI, WAV, AU, MIDI e AIFF. Una delle caratteristiche principali che sono necessarie quando si vuole gestire materiale multimediale, ad esempio tramite un lettore multimediale, è la velocità di computazione. È richiesta un'elevata velocità computazionale per garantire che operazioni quali, ad esempio, la decompressione delle immagini o il rendering vengano svolte in maniera soddisfacente. Sappiamo che Java utilizza una Java Virtual Machine, che interpreta il byte-code generato dal compilatore del linguaggio. Questo meccanismo, che ha come suo punto di forza la garanzia di portabilità, impone, allo stesso tempo, seri vincoli in termini di prestazioni, qualora si richieda elevata velocità computazionale. Per ovviare a questi problemi, e voler trattare dati multimediali, è necessario ricorrere all'utilizzo di codice nativo della piattaforma su cui si opera. Questo comporta che il programmatore abbia una buona conoscenza delle funzioni native e, cosa ancora più importante, pone un serio vincolo alla portabilità dell'applicazione realizzata. Le API JMF cercano di ovviare a questi problemi.

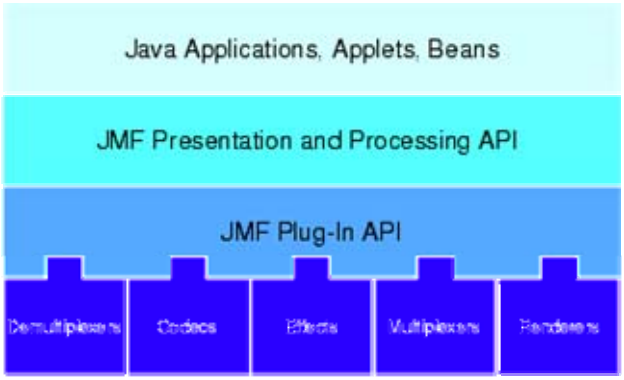


Figura 4.13:Architettura del Java Media Framework.

Il componente JMF mette a disposizione degli sviluppatori un insieme di chiamate ad "alto livello" che consentono la gestione di codice nativo. L'applicazione o l'applet, che integra JMF, non ha bisogno di sapere se e quando deve ricorrere al codice nativo per soddisfare una determinata richiesta. La versione 2.1.1 di JMF rende disponibili classi che consentono di sviluppare applicazioni che catturino dati multimediali e che forniscano la possibilità di controlli ulteriori sull'elaborazione e la riproduzione del materiale stesso. In particolare il componente JMF 2.1.1 è stato progettato per ottenere i seguenti vantaggi:

- facilitare la programmazione;
- mettere a disposizione del programmatore un player JMF per la riproduzione di dati multimediali;
- semplificare l'integrazione di sorgenti multimediali in applet o applicazioni fornendo classi e metodi che consentano la gestione temporale di stream di dati;
- consentire la connessione a host remoti e l'instaurazione di sessioni http o RTP/RTCP (Real Time Control Protocol) o RTSP (Real Time Streaming Protocol);
- permettere la realizzazione di applicazioni in audio e video conferenza in linguaggio Java;
- consentire a programmatori avanzati di sviluppare soluzioni basate sulle API esistenti e di integrare le nuove caratteristiche nella struttura esistente;
- permettere lo sviluppo di demultiplatori, codificatori, elaboratori, multiplatori e riproduttori personalizzati (JMF plug-in);
- garantire la compatibilità con le sue versioni precedenti.

Di seguito verranno presentati brevemente alcuni dei componenti che costituiscono questo prodotto.

4.3.1 Il componente Player

Definendo con media stream il dato multimediale ottenuto da un file locale, acquisito dalla rete oppure da un dispositivo di input (videocamera, microfono, ecc.), possiamo considerare il *Player* come quella struttura che ne consente la riproduzione. È possibile paragonare il *Player*, per le funzioni di controllo che rende disponibili, ad un

semplice videoregistratore. Grazie ad esso, gli sviluppatori di software, possono disinteressarsi delle chiamate al codice nativo, ed allo stesso tempo possono occupare risorse necessarie per la riproduzione e rilasciarle quando non più necessarie. Queste chiamate a metodi e classi di alto livello rendono trasparente al programmatore la connessione che viene stabilita tra la JVM e le routine specifiche di sistema. Il dato multimediale in ingresso al *Player* viene collegato ad esso mediante una struttura denominata *DataSource*, che fa riferimento al dato vero e proprio. Quest'ultima struttura potrebbe essere considerata alla stregua di una videocassetta per un videoregistratore, il *Player*. Nella figura 4.14 viene rappresentato quanto appena descritto.

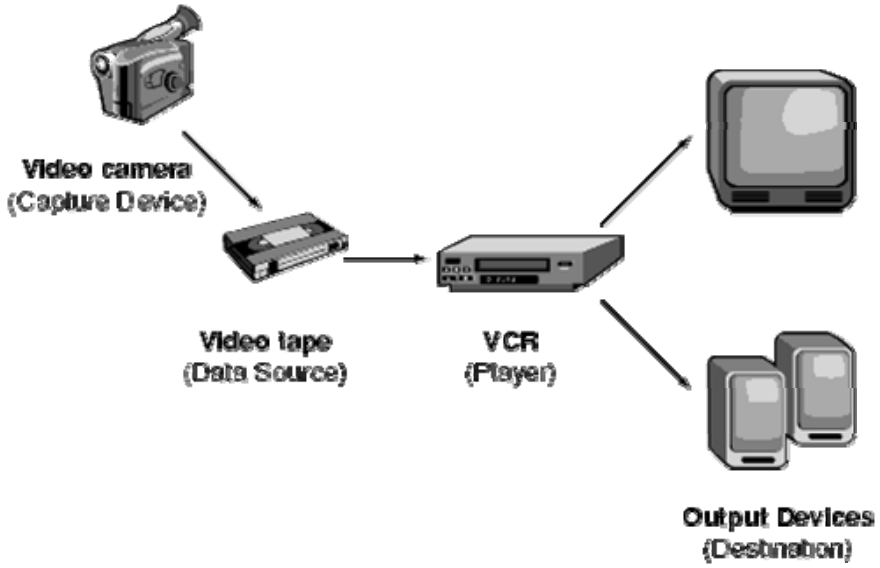


Figura 4.14: ricezione e riproduzione di flussi multimediali

Come si può notare da questa figura, il media stream riprodotto potrebbe essere di tipo audio o di tipo video o audio e video. Infatti un *media stream*, anche se identificato da un'unica traccia, potrebbe contenere più canali, come ad esempio un canale audio ed uno video. Un *Player* può trovarsi, operando normalmente, in uno dei suoi sei stati possibili. Ho cinque di questi stati che individuano una comune situazione di interruzione della riproduzione, vuoi perché è stata interrotta, vuoi perché non è ancora iniziata. Solitamente vengono attraversati questi primi stati fino a giungere al sesto stato, quello dell'effettiva riproduzione del contenuto multimediale. La transizione fra questi stati avviene solitamente in seguito al verificarsi di determinati eventi. In generale, il *Player*, una volta inizializzato, può fornire, se presente, un componente

visuale utilizzato per la fruizione del dato multimediale. All'interno dell'ambiente MUM il *Player* è quell'entità che permette al client di ricevere e riprodurre i flussi multimediali. Nell'ambito di questa tesi, per il cliente mobile, prima del componente *Player* verrà inserito un *buffer* circolare che accumulerà dati prima dell'effettiva visualizzazione, in modo che durante l'*handoff*, si possa garantire una continuità del flusso video. Questo *buffer* si comporterà come il componente *DataSource* che verrà esaminato più avanti, in modo da rendere invisibile al *Player* la sua presenza.

4.3.2 Il componente Processor

Per la riproduzione di media stream può essere utilizzata anche la struttura denominata Processor. Questa, infatti, risulta essere un'estensione del componente Player. Per quanto riguarda l'utilizzo del processor nell'ambiente MUM ed in questo progetto di tesi, possiamo dire che le sue funzioni sono quelle di ricezione ed invio dei flussi multimediali richiesti. Infatti, oltre a metodi di gestione e trasformazione relativi ai tipi di dati multimediali, questa struttura offre la possibilità di ottenere un *DataSource* in uscita.

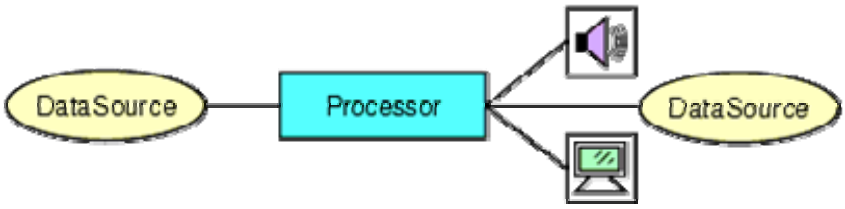


Figura 4.15: ricezione ed invio di flussi multimediali.

Il vantaggio di poter avere un output reindirizzato direttamente su di un *DataSource* è rappresentato dal fatto che questa uscita può essere presentata direttamente all'ingresso di un Player, di un *DataSink* (che descriveremo nel prossimo paragrafo) o di un altro processor. Facendo riferimento alle caratteristiche di questo componente fino a qui descritte è intuibile come sia stato utilizzato all'interno dell'ambiente MUM: come parte integrante delle entità server e proxy. A differenza del Player, il processor può trovarsi in uno dei suoi otto stati possibili. Questi otto stati vengono suddivisi in due gruppi principali che individuano due condizioni del processor: *unrealized* e *realized*. Durante

le sue fasi di inizializzazione e configurazione è definito *unrealized*, mentre al termine di queste fasi viene definito *realized* ed è pronto per avviare la trasmissione dei dati. La fase di configurazione di un processor prevede la connessione del *DataSource*, la demultiplazione dello stream di ingresso e il reperimento delle informazioni sul formato dei dati multimediali in ingresso. La demultiplazione consiste sostanzialmente nell'analisi del flusso multimediale in ingresso per l'individuazione di eventuali tracce multiple. Nel qual caso si verifichi la presenza di tracce multiple queste vengono automaticamente suddivise ed inviate separatamente in output. Se l'output è indirizzato verso un *DataSource*, le diverse tracce individuate devono essere raggruppate nuovamente in modo da ottenere un unico flusso multimediale d'uscita (come evidenziato nella figura 4.16).

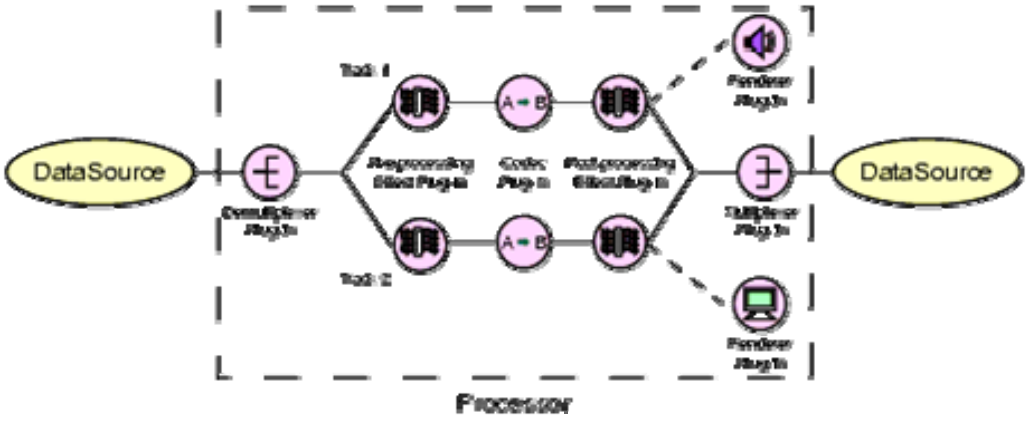


Figura 4.16: demultiplazione e multiplazione del flusso multimediale.

Anche nel caso del processor, le transizioni da uno stato ad un altro sono determinate dal verificarsi di alcuni eventi precisi.

4.3.3 Il componente Buffer

In JMF ogni contenuto multimediale, sia audio che video, è costituito da una serie di elementi denominati *frame*. I *frame* rappresentano la più piccola porzione di contenuto multimediale che è possibile memorizzare, figura 4.17.

Il componente Buffer svolge appunto la funzione di memorizzare un singolo *frame*, incapsulando tutti i dettagli che lo riguardano come il tempo nel quale il *frame* si

posiziona nel contenuto multimediale, la sua velocità di esecuzione, la sua durata, il numero di sequenza all'interno del flusso e il formato, ossia il tipo di dato che rappresenta e la sua codifica. Questo componente è importate per la realizzazione di un Buffer circolare, in quanto utilizzando una serie di questi elementi è possibile memorizzare il flusso in arrivo e rileggerlo in un secondo momento.

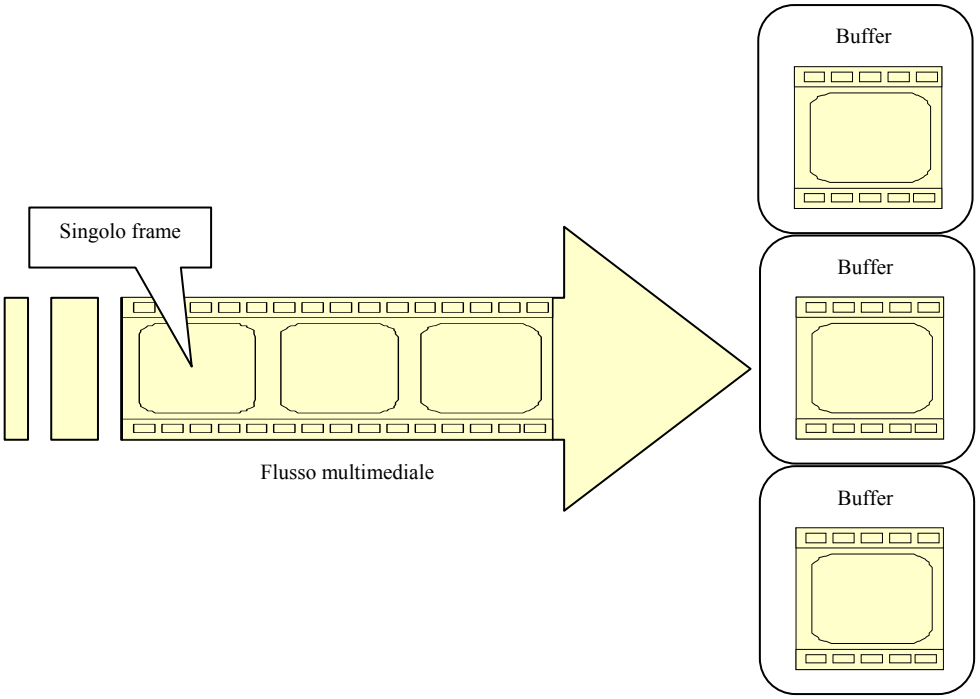


Figura 4.17: I frame del flusso multimediale.

Il Buffer Circolare è un elemento passivo che contiene un numero fissato di componenti Buffer. È strutturato in maniera tale che lettura e scrittura avvengano in modo controllato, ossia il produttore cioè colui che scrive nel Buffer Circolare rimane bloccato in scrittura solo nel caso in cui sia non presente spazio libero e rimane in attesa che si liberi, mentre il consumatore, cioè colui che legge dal Buffer Circolare, rimane bloccato in lettura solo se non sono presenti *frame* da leggere, in attesa che il produttore li inserisca. L'entità attiva che si occuperà del riempimento di questo elemento, dovrà anche controllare il flusso in arrivo, infatti nel caso in cui il buffer circolare sia vuoto, per accelerare il riempimento dovrà richiedere un aumento della velocità, mentre nel caso in cui il buffer sia prossimo al riempimento decelerare o al limite fermare i dati in arrivo per evitare di perdere dei *frame*.

4.3.4 Il componente DataSource

Un altro dei componenti integrati nel framework proposto dalla SUN, ed importante per la realizzazione del nostro sistema, risulta essere il *DataSource*. Come abbiamo accennato in precedenza, può essere inteso come una videocassetta, ovvero come l'oggetto che contiene al suo interno il multimedia stream. In particolare contiene al suo interno sia le informazioni riguardo l'ubicazione dell'oggetto multimediale che quelle riguardanti i protocolli ed il software interessato.

Una fondamentale caratteristica relativa a questo componente è il suo essere clonabile. Un *DataSource* definito *clonable*, può essere utilizzato per creare altri *DataSource* cloni di quello originario. I cloni creati possono essere controllati direttamente dal clonable *DataSource* utilizzato per generarli. In particolare, la chiamata di metodi (ad es.: connect, disconnect, start o stop) relativi al *DataSource* originario, viene propagata anche ai suoi cloni.

4.3.5 Il componente DataSink

L'ultima struttura del Java Media Framework che presentiamo è l'interfaccia denominata *DataSink*. Questo componente viene utilizzato per leggere un flusso multimediale da un *DataSource* e reindirizzare questo contenuto verso altra destinazione. Solitamente non viene utilizzato per la riproduzione di media stream. I tipici utilizzi in cui viene impiegato un *DataSink* sono la scrittura del flusso in entrata in un apposito file o attraverso l'infrastruttura di rete. Un *DataSink* permette di iniziare il trasferimento delle informazioni multimediali, di interromperlo e di reagire di conseguenza al verificarsi di determinate condizioni, corrette o errate che siano.

4.4 Jini e UPnP API

Nel codice relativo alla tesi, sono state utilizzate due librerie per l'accesso a standard di discovery, in particolare a Jini e UPnP.

Per accedere alle funzionalità di Jini esistono opportune librerie Java che permettono di interagire con i servizi offerti dalla piattaforma, distribuite insieme al sistema stesso [Jini].

Le API di Jini sono organizzate in un vasto numero di package, quelli impiegati nella tesi sono i seguenti:

- *net.jini.discovery*: contiene le classi e le interfacce per accedere al sistema di nomi di Jini Lookup Service;
- *net.jini.lookup*: contiene le classi e le interfacce per effettuare le ricerche e le registrazioni al Lookup Service;
- *net.jini.lease*: contiene le classi e le interfacce per la gestione del leasing dei servizi.

Per l'accesso a UPnP è stato utilizzata, invece, una libreria *open source* sempre scritta in Java che realizza tutte le fasi di comunicazione attraverso il protocollo http e permette di effettuare l'analisi dei dati scambiati con il server UPnP, tramite il parsing del formato XML [Konno].

Le API proposte sono organizzate nei seguenti package:

- *org.cybergarage.http*: contiene le classi e le interfacce per l'accesso a UPnP tramite il protocollo http;
- *org.cybergarage.soap*: contiene le classi e le interfacce per la gestione del protocollo SOAP utilizzato da UPnP per eseguire chiamate a procedura remota e per interagire con i servizi;
- *org.cybergarage.upnp*: fornisce le classi di base per accedere a UPnP e per la creazione di servizi;
- *org.cybergarage.xml*: definisce le classi per gestire il formato XML, utilizzato nella descrizione dei servizi e nell'interazione tramite il protocollo SOAP.

4.5 Conclusioni

In questo capitolo l'attenzione è stata focalizzata sulle tecnologie coinvolte nella realizzazione del progetto. Per ognuna di esse è stata fatta una panoramica introduttiva, cercando di fornire informazioni generiche di inquadramento. Successivamente sono stati approfonditi quegli argomenti che risultano direttamente coinvolti nel lavoro di tesi. Alcuni di questi verranno ripresi anche nei capitoli successivi, trattando le loro più specifiche caratteristiche tecniche ed implementative. In particolare, nel prossimo capitolo verrà affrontata l'analisi del sistema realizzato.

CAPITOLO 5

ANALISI DEL SISTEMA

DI

DISCOVERY E ADATTAMENTO

I precedenti capitoli hanno cercato di creare il background di conoscenze necessario per comprendere le problematiche legate allo sviluppo della presente tesi.

La tesi punta ad inserire nuove funzionalità sia all'infrastruttura SOMA, sia al middleware MUM, che come abbiamo visto nei capitoli precedenti, si appoggia a SOMA per il suo funzionamento.

Per quanto riguarda SOMA l'obbiettivo è introdurre un sistema di accesso all'infrastruttura MUM da parte di dispositivi dalle capacità limitate e quindi non in grado di sostenere una ambiente complesso come SOMA, gestire l'handoff fra un place dotato di accesso wireless ed inoltre inserire un sistema di discovery di servizi, che permetta all'utente che utilizza il dispositivo mobile di selezionare una postazione fissa dove effettuare uno spostamento nomadico e quindi usufruire delle caratteristiche del nuovo terminale.

Per quanto riguarda MUM, si predisporrà l'ambiente allo *streaming* su un dispositivo mobile, che comporta anche una adattamento dei contenuti per poterlo visualizzare utilizzando al meglio le caratteristiche che tale dispositivo offre, MUM dovrà essere anche in grado di interagire con il servizio di discovery da inserire in SOMA, per poter autoconfigurarsi su una postazione fissa, accelerando i tempi di spostamento della sessione su un computer fisso.

La suddivisione della fase progettuale in produzione dell'architettura logica e del progetto concreto è una prassi ben consolidata dell'ingegneria del software, e viene qui adottata perché si ritiene di fondamentale importanza non asservirsi da subito alle tecnologie, per concentrarsi unicamente sulle reciproche interazione tra le parti e per facilitare apertura e reingegnerizzazione futura.

5.1 Scenario applicativo

Riprendiamo brevemente lo scenario visto nel capitolo 1. Un utente che dispone di un terminale mobile connesso ad una rete wireless decide di visualizzare un filmato disponibile su un server predisposto a questo scopo; una volta selezionato il genere di filmato da visualizzare, invia la richiesta al server, il quale dopo aver verificato i dati dell'utente e le caratteristiche del suo dispositivo, inizia l'invio in *streaming* del filmato verso il dispositivo mobile. L'utente durante la visualizzazione potrebbe aver bisogno di muoversi in un'altra zona, anch'essa coperta dal servizio wireless, ma su una differente rete, il dispositivo a questo punto è in grado di fornire una previsione sulla locazione di destinazione e di conseguenza l'infrastruttura sottostante reagisce predisponendo le entità necessarie pro-attivamente, evitando che il filmato venga interrotto. Quando l'utente arriva nella zona di destinazione, l'infrastruttura trova uno schermo che dispone di caratteristiche simili a quelle specificate dell'utente nelle sue preferenze e come prima cosa inizializza le entità necessarie, poi informa il cliente della possibilità di passare dal terminale mobile ad un terminale fisso, migliorando l'esperienza di visualizzazione, a questo punto in base alla scelta del cliente si avrà uno spostamento o meno della sessione.

Questo scenario mostra tutti gli obbiettivi che si ha intenzione di raggiungere in questa tesi, inoltre emergono tutti gli elementi che devono essere sviluppati affinché il sistema sia in grado di fornire tale servizio. Innanzitutto, come più volte evidenziato, un terminale mobile ha delle limitazioni rispetto ad un terminale fisso, principalmente la capacità di elaborazione, le piccole dimensioni dello schermo e la durata delle batterie, l'infrastruttura sottostante deve sempre tenere conto di questi vincoli. Di conseguenza, per sfruttare nel modo migliore il terminale mobile, il contenuto multimediale deve essere adattato alle particolari caratteristiche del dispositivo, in particolare la dimensione dell'immagine dovrà essere uguale o inferiore alla risoluzione effettiva dello schermo del terminale mobile.

Lo spostamento del terminale da una rete ad un'altra comporta una serie di operazioni che vanno sotto il nome di handoff. In generale, il dispositivo mobile è in grado di fornire una previsione del possibile Access Point (AP) di destinazione, fornendo all'infrastruttura la possibilità di predisporre lo spostamento della sessione

riducendo il tempo di passaggio fra due reti e di conseguenza riducendo le probabilità che il filmato venga interrotto.

La ricerca di dispositivi alternativi di visualizzazione, comporta l'utilizzo di un sistema di discovery, in grado di ospitare le informazioni sulle caratteristiche di tali dispositivi. Il successivo passaggio dal terminale mobile ad dispositivo di visualizzazione fisso, è denominata *offload*. L'*offload* è costituito da una serie di operazioni, in specifico l'inizializzazione del percorso fino al dispositivo e l'operazione di riadattamento del video, necessaria per sfruttare le caratteristiche del dispositivo di destinazione.

5.2 Requisiti

È solitamente considerata buona norma nello sviluppo di un progetto software, e di qualunque altro genere di progetto in generale, individuare i suoi requisiti.

Possiamo distinguere i requisiti in funzionali e non funzionali. I primi esprimono le funzionalità che effettivamente il sistema costruito dovrà saper svolgere. I secondi, invece, indicano le linee guida strutturali che devono essere tenute in considerazione durante lo sviluppo dell'applicazione, per ottenere un prodotto di buona qualità, ben costruito e facilmente reingegneralizzabile.

5.2.1 Requisiti funzionali

Come è stato accennato in precedenza, SOMA è un ambiente di supporto per agenti mobili, in generale ambienti di questo tipo risultano essere molto onerosi in termini di prestazioni richieste al dispositivo in grado di ospitarli, di conseguenza i terminali mobili, che in genere dispongono di una quantità di memoria e capacità computazionali limitate, non sono in grado di eseguirli. Il sistema che si vuole realizzare, lavorando in cooperazione con SOMA deve fornire un punto di accesso al dispositivo mobile semplificando l'interazione con quest'ultimo, permettendo di riconoscere le caratteristiche del dispositivo dell'utente in base al suo profilo utente.

Il profilo utente è un contenitore di informazioni che descrive le caratteristiche del dispositivo utilizzato dall'utente e che il sistema considererà nello svolgimento del proprio lavoro. Alcune caratteristiche interessanti sul dispositivo potrebbero essere la dimensione dello schermo, la bit-rate disponibile, il numero di colori che esso è in grado di gestire e così via. Le informazioni contenute nel profilo utente potrebbero contenere anche le preferenze sullo schermo nel caso voglia spostare la sessione su un terminale fisso, in modo che il sistema possa filtrare fra le possibili alternative quelle che vengono incontro alle preferenze dell'utente. Il vantaggio di conoscere in precedenza le possibili scelte dell'utente permette inoltre di poter predisporre il cammino che deve essere instaurato per fornire il servizio video, ottenendo quindi una sensibile riduzione del tempo di inizializzazione.

Quando si progetta un'applicazione per terminali mobili bisogna sempre considerare che questi dispositivi sono dotati di una batteria, di conseguenza ogni trasmissione di rete porta ad una riduzione della durata complessiva, è necessario quindi che queste trasmissioni siano ridotte al minimo indispensabile.

Affinché un terminale mobile che si muove fra due AP diversi sia in grado di mantenere costante il flusso video e non produrre salti nella visualizzazione, percepiti dall'utente finale come un calo di qualità, è necessario che il filmato sia memorizzato in un'apposita zona di memoria che previene le interruzioni del flusso video.

5.2.2 Requisiti non funzionali

I principi a cui si cercherà di attenersi il più possibile nello sviluppo del sistema in esame sono i seguenti:

1. *Modularità*: Si cercherà di suddividere il sistema in parti, ognuna in grado di svolgere determinate attività, al fine di separare le varie logiche che guidano ognuna di esse. Il notevole vantaggio di un sistema sviluppato secondo questo principio è quello di assicurare incapsulamento, in modo che, se le cose sono state sviluppate nel modo corretto, sarà possibile modificare il sistema semplicemente sostituendo la vecchia versione di un modulo con la nuova.

2. *Flessibilità*: Essendo il sistema in esame parte di un middleware, è molto importante attenersi a questo principio e non effettuare scelte architetturali e di progetto legate a un particolare contesto di utilizzo del sistema, ma invece cercare di risolvere i problemi sempre nel modo più generale possibile. Ovviamente questo potrebbe richiedere un aumento di complessità considerevole durante le fasi di sviluppo e implementazione del sistema. Ci si riserva, dunque, di discutere di volta in volta la possibilità di effettuare opportune semplificazioni.
3. *Scalabilità*: Nell'ottica di inserire il componente che verrà realizzato in un ambiente distribuito, è di fondamentale importanza che il degrado delle sue prestazioni non lo rendano inutilizzabile all'aumentare delle dimensioni del sistema all'interno del quale verrà utilizzato. Tuttavia, solitamente, i sistemi multimediali per l'alto fabbisogno di risorse richiesto, sono intrinsecamente poco scalabili.

5.2.3 Analisi dei requisiti

Da una lettura attenta dei requisiti funzionali del sistema che si vuole sviluppare, possiamo distinguere quattro attività fondamentali: servizio di discovery di servizi, inizializzazione del sistema di distribuzione video tramite adattamento, la gestione dell'handoff ed infine la migrazione della sessione su un terminale fisso.

Come accennato nell'introduzione, le modifiche da apportare al sistema preesistente coinvolgeranno sia l'ambiente SOMA sia l'infrastruttura MUM. In SOMA possono essere individuati due moduli principali mostrati in figura 5.1.

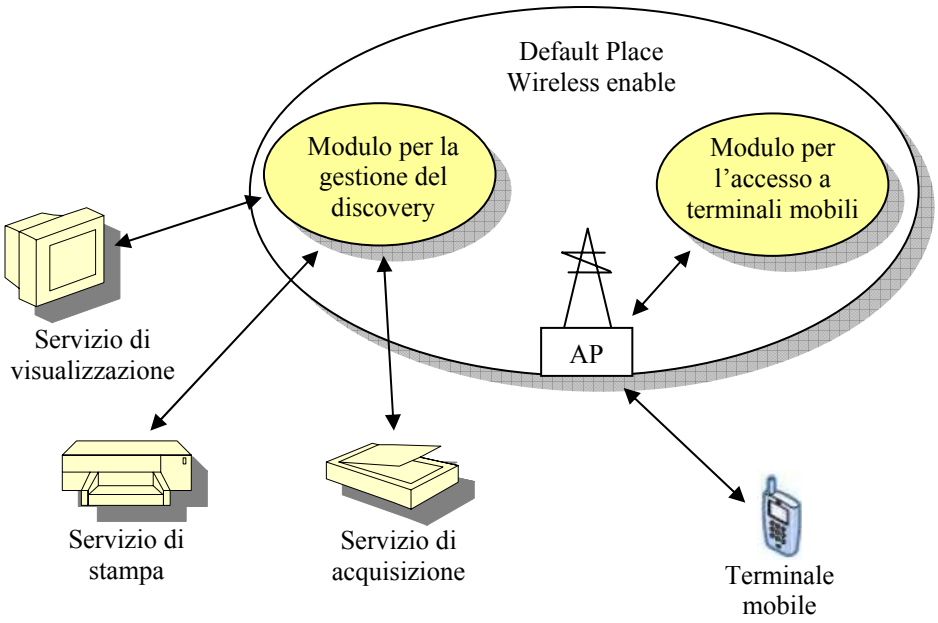


Figura 5.1: Moduli nell'ambiente SOMA

Il modulo di gestione del discovery si occuperà di memorizzare tutti i servizi forniti della località a cui riferisce il Default Place ed permetterà di fare ricerche in base a determinate caratteristiche. I Default Place dotati di un AP permetteranno un accesso a dispositivi mobili, questo accesso avverrà tramite un modulo apposito che interpreterà i pacchetti di richiesta come si vede nella figura 5.1.

Nel paragrafo 4.2.3 sono state evidenziale le cinque unità fondamentali di MUM: il *Client*, il *Server*, il *Proxy*, il *ProxyAgent* e il *ClientAgent*. L'infrastruttura in progetto estenderà la struttura preesistente, aggiungendo due nuove entità denominate *ProxyManagerAgent* e *ClientMobile*, la prima realizzata come un agente mobile, la seconda realizzata come entità fissa sul terminale mobile.

Il *ProxyManagerAgent* sarà un entità di gestione direttamente accoppiata con il *ClientMobile*. Si occuperà principalmente di inizializzare tutto il percorso, interagendo con il *ClientMobile*, al fine di riconoscere le caratteristiche del dispositivo portatile utilizzato dall'utente, decidendo in questo modo il metodo di adattamento dei contenuti che meglio si applica a tale dispositivo. Secondariamente si occuperà della gestione dell'handoff e di interagire con il gestore del servizio di discovery per ricercare un'eventuale postazione fissa che venga incontro alle esigenze dell'utente,

informandolo della possibilità di trasferimento su un nuovo terminale, sul quale in seguito verrà utilizzata l'entità *ClientAgent*.

Il *ClientMobile* riprende alcune funzioni del *ClientAgent*, come l'interfaccia grafica e l'interazione vera e propria con il sistema, ma a differenza di quest'ultimo sarà realizzato come una entità fissa, in quanto rimarrà sempre sul dispositivo mobile ed interagirà con la piattaforma MUM limitando il più possibili il quantitativo di dati scambiati per preservare le batterie consumate dal dispositivo.

Quindi si può notare come il *ProxyManagerAgent* e il *ClientMobile* siano elementi legati alla sessione su terminale mobile, mentre il *ClientAgent* sia un elemento che riguarda la sessione su terminale fisso.

L'adozione di un approccio a moduli ed entità permetterà di analizzare, progettarli e realizzarli separatamente concentrandosi di volta in volta solo sugli aspetti caratteristici del componente considerato. In un approccio di tale genere, però, è di fondamentale importanza definire e tenere d'occhio le interazioni fra le varie parti che permetteranno loro di collaborare correttamente al fine di offrire i servizi richiesti.

5.3 Analisi

In questa sezione verrà effettuata l'analisi dei vari componenti del sistema e delle loro reciproche relazioni al fine di produrre un'architettura logica robusta che guiderà le successive fasi di progettazione e implementazione del sistema.

5.3.1 Servizio di discovery di servizi

Il servizio di discovery di servizi va ad aggiungersi ai servizi offerti dalla piattaforma SOMA, verrà inserito all'atto della creazione del *Environment* e quindi nel momento di creazione del Place. Trattandosi di un servizio che raccoglierà al suo interno i servizi offerti in una determinata località, sarà di conseguenza posizionato in un Default Place, in questo modo tutti i Place figli possano accedere al servizio indirettamente, senza nessuna operazione di ricerca della posizione effettiva.

Nel capitolo 3 sono stati introdotti i diversi standard che permettono il discovery di servizi in una rete, fra questi sicuramente Jini è il più vicino alle caratteristiche del sistema preesistente, in quanto è anch'esso scritto in Java. Si è preferito adottare una soluzione proprietaria, perché possa essere realizzata venendo incontro alle esigenze dell'infrastruttura sottostante SOMA e MUM, e contemporaneamente realizzare un sistema di accesso generalizzato a qualsiasi standard di discovery esistente. La struttura del servizio di discovery è presentata in figura 5.2.

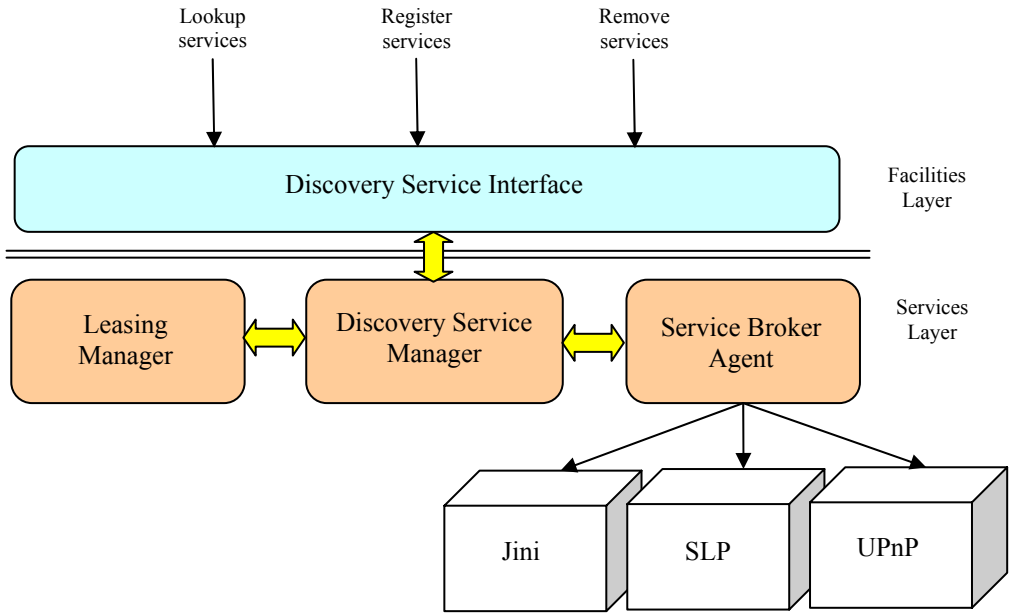


Figura 5.2: Struttura del servizio di Discovery

Seguendo l'architettura già proposta per MUM, nella quale vengono presentati tre livelli indicanti diverse categorie di servizi, il discovery di servizi è stato anch'esso suddiviso in livelli: il *Facilities Layer* e il *Service Layer*. Il *Facilities Layer* contiene l'interfaccia attraverso la quale gli applicativi che eseguono sopra SOMA e MUM, accedono al servizio di discovery, permettendo a loro di cercare, aggiungere e rimuovere servizi. Il *Services Layer* contiene invece i servizi che costituiscono il sistema di discovery, denominati *Discovery Manager*, *Leasing Manager* e *Service Broker Agent*, che verranno introdotti nei paragrafi successivi.

5.3.1.1 Discovery Manager

Il Discovery Manager è il componente che gestisce il database che contiene le descrizioni dei servizi. Dato in ingresso la descrizione di un servizio fornisce in uscita un elenco di tutti i servizi contenuti nel database la cui descrizione collima con quella fornita in ingresso. Questo componente interagisce con gli altri due componenti del *Services Layer* per gestire l'eliminazione automatica dei servizi e per garantire l'interoperabilità con altri standard di discovery, come verrà chiarito nel rispettivi paragrafi.

5.3.1.1.1 Descrizione di un servizio

Affinché un sistema di discovery possa essere considerato efficiente deve disporre di un sistema di descrizione dei servizi che permetta di sintetizzare le caratteristiche che può offrire un servizio all'utente finale, senza però perdere ricchezza espressiva tralasciando informazioni importati. Questa descrizione è la base del confronto con le richieste dell'utente.

Come è stato più volte evidenziato, il termine servizio racchiude in se un significato molto generico, infatti per servizio può essere inteso sia un dispositivo hardware, sia un componente software, il cui compito è fornire una qualche utilità all'utente.

La descrizione del servizio potrebbe essere una semplice stringa, un insieme di attributi, oppure un documento XML [ChenKotz]. Nel caso si usi una stringa, questa potrebbe possedere o meno un struttura sintattica, come ad esempio "monitor 1024x768" oppure "/device/monitor/1024x768". La rappresentazione a stringa è tipicamente molto limitata dal punto di vista espressivo e la sintassi utilizzata potrebbe risultare scomoda nel caso di una descrizione particolarmente complessa. Un'altra rappresentazione comunemente utilizzata è il naming basato su attributi. Un attributo è utilizzato come una chiave di ricerca, un nome a sua volta può contenere un insieme di attributi; la struttura di un attributo può essere semplice oppure di tipo gerarchico cioè un attributo può essere figlio di un altro attributo. Riprendendo l'esempio precedente si

possono specificare anche alcune caratteristiche proprie della stampante ed eventualmente anche la sua locazione fisica:

[servizio=monitor, risoluzione=1024x768, edificio=deis [piano=1 [stanza=6.2]]].

I nomi basati su XML sono anch'essi molto espressivi e la loro sintassi facilita l'interoperabilità fra servizi e clienti. In aggiunta alle proprietà del servizio, XML può anche essere usato per specificare la sintassi e la semantica dell'interfaccia di un servizio, permettendo al cliente di invocare direttamente questi metodi analizzando il documento XML. Uno svantaggio del servizio di nomi basato su XML è la sua sintassi molto estesa, di conseguenza è necessaria una scansione del documento (parsing) per la ricerca delle informazioni, che porta ad un aumento dell'overhead rispetto ai casi precedenti.

Un altro sistema di descrizioni di servizi, è quello che rappresenta i singoli attributi come oggetti. Questo sistema, presente in Jini, permette una descrizione molto potente, in quanto è possibile creare attributi con una struttura molto articolata e, allo stesso tempo, grazie alla serializzazione di Java permette di memorizzare i dati in forma compatta permettendo di risparmiare memoria.

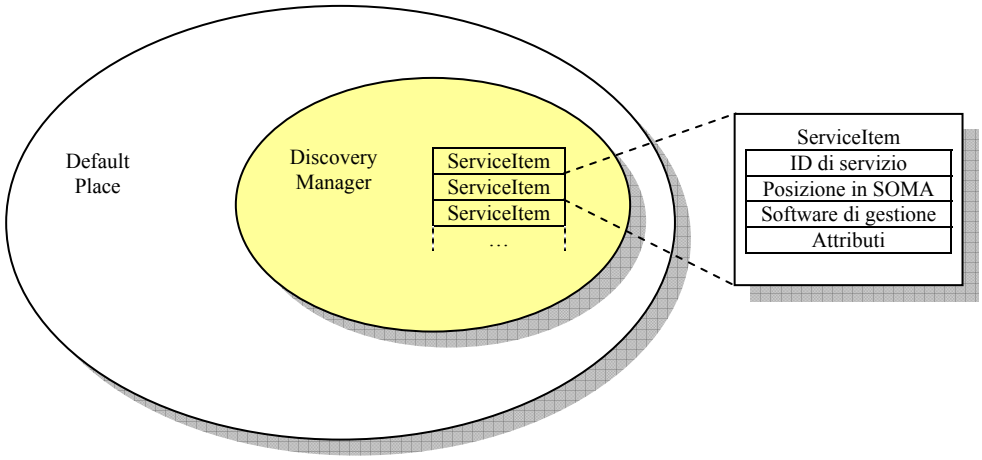


Figura 5.3: Struttura di un ServiceItem

Nella presente tesi, la descrizione del servizio sarà rappresentata dall'entità *ServiceItem* (figura 5.2), la quale conterrà tutte le caratteristiche che definiscono un determinato servizio, queste ultime sono rappresentate da:

- *un identificativo di servizio*: ogni servizio deve essere identificabile all'interno del sistema in modo univoco, questo per evitare che un servizio possa essere confuso con un altro che riporta caratteristiche simili o identiche; siccome il sistema di discovery progettato poggia sul sistema di astrazione di località di SOMA (si veda paragrafo 4.1.1.1 per dettagli), l'identificatore sarà costituito da una parte dipendente dal Default Place in cui il servizio di discovery è localizzato e dall'identificativo vero e proprio, ottenendo in questo modo una univocità su tutto il sistema, similmente come avviene per gli identificatori degli agenti in SOMA;
- *posizione in SOMA*: rappresenta la posizione del Place che ospita il servizio o che interagisce con esso, una volta noto questo valore è possibile inviare agenti che interagiscano con il servizio;
- *software di gestione*: sfruttando il sistema di distribuzione di codice presente in MUM (si veda a tal proposito il paragrafo 4.2.2) è possibile aver a disposizione il codice per la gestione del servizio da qualsiasi Place attraverso il quale accede al servizio;
- *attributi caratteristici*: rappresentano tutte le informazioni base del servizio che si sta descrivendo.

Lo stesso sevizio di discovery rappresenta esso stesso un servizio descrivibile tramite un ServiceItem, questo accorgimento permette di raggruppare i servizi di discovery in una organizzazione gerarchica ha il vantaggio di essere molto decentralizzata, in quando il servizio di una certa zona è affidato ad un default place, ma allo stesso tempo permette di essere a conoscenza dell'intera struttura ripercorrendo la gerarchia.

I vari *ServiceItem* rappresentanti tutti i servizi del Default Place a cui il servizio di discovery si riferisce, verranno memorizzati in un database sul quale verranno eseguite tutte le ricerche.

5.3.1.1.2 Metodo di ricerca

Il metodo di ricerca si baserà semplicemente sul confronto degli attributi dei servizi presenti nel database con quelli richiesti. L'oggetto *ServiceTemplace* rappresenta la

struttura base che il servizio da ricercare deve avere, al suo interno sono presenti tutti gli attributi che saranno presi come mezzo di confronto con i servizi nel database.

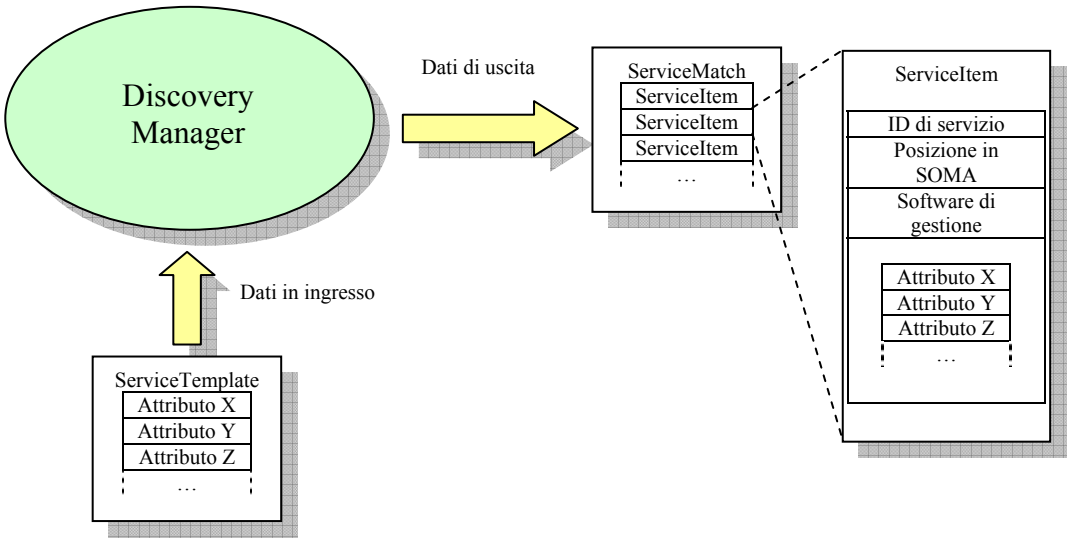


Figura 5.4: Ricerca di servizi

Alla fine della ricerca il sistema di discovery restituisce l'oggetto *ServiceMatch*, che racchiude al suo interno tutti i servizi i cui attributi sono uguali a quelli richiesti, come mostrato in figura 5.3. Nel caso la ricerca coinvolga delle valutazioni che vadano oltre la semplice uguaglianza, come ad esempio valori all'interno di un determinato *range*, allora è necessario effettuare un ulteriore raffinamento dei valori trovati, tramite un confronto più accurato degli attributi.

5.3.1.2 Service Broker Agent

Questo componente è l'elemento base per garantire l'interoperabilità fra diversi standard di discovery. I diversi standard di discovery offrono meccanismi di recupero e descrizione dei servizi differenti, affinché sia possibile fornire un'interfaccia comune al sistema è necessario interporre fra il sistema di descrizione utilizzato e il sistema di discovery nel quale si intende ricercare un servizio, un mediatore che traduca le richieste da un sistema ad un altro e viceversa per le risposte. Il *Service Broker Agent* rappresenta fondamentalmente un particolare tipo di servizio il cui scopo è tradurre il *ServiceTemplate* del Discovery Manager in una richiesta di servizio per il particolare

standard a cui riferisce. Esistono quindi un numero di *Service Broker Agent* pari al numero di standard di discovery con i quali si vuole garantire l'interoperabilità figura 5.5.

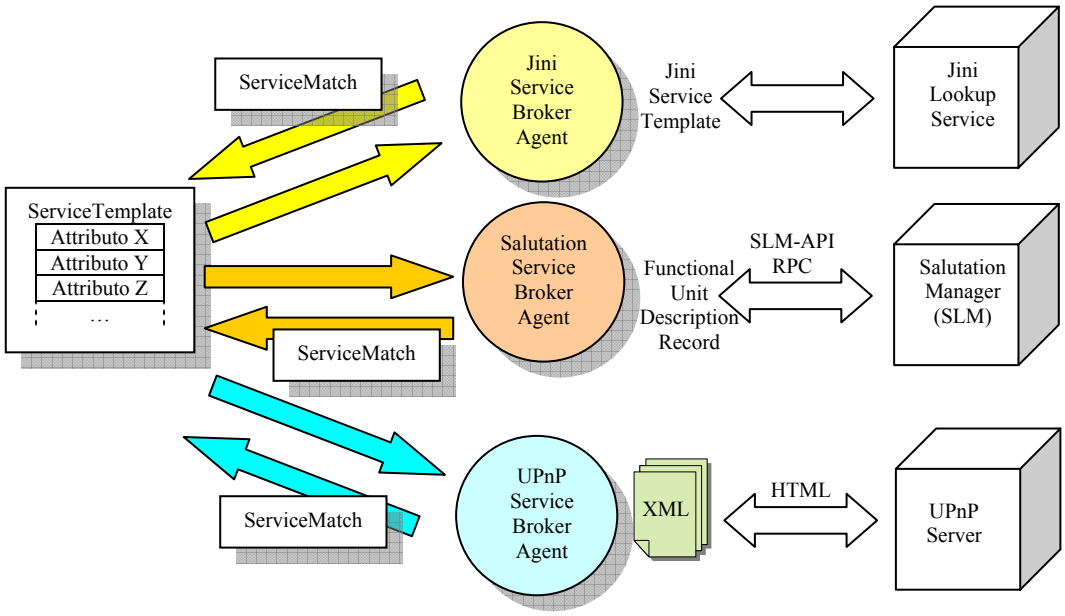


Figura 5.5 Service Broker Agent

La flessibilità di questo approccio è costituita dal fatto che il *Service Broker Agent* ha una struttura generica che accetta in ingresso un *ServiceTemplate* e restituisce in uscita un *ServiceMatch*, e quindi è possibile aggiungere al Discovery Manager nuovi *Service Broker Agent* senza dover modificare il componente *Discovery Manager*. Questa flessibilità si paga in termini di prestazioni in quanto ogni ricerca estesa ad un servizio aggiuntivo comporta l'instanziazione di un *Service Broker Agent* che funga da mediatore. Per garantire una migliore risposta, le ricerche effettuate tramite *Service Broker Agent* verranno memorizzate nel database locale che fungerà da cache, quindi inizialmente la ricerca coinvolgerà il database locale e nel caso in cui non fosse trovato nessun servizio che soddisfi i criteri di ricerca, allora la ricerca verrà estesa ai *Service Broker Agent*.

5.3.1.3 Leasing Manager

Al fine di facilitare la gestione dell'aggiornamento dello stato dei servizi, si inserirà nel sistema di discovery il concetto del leasing. La registrazione di un servizio presso il sistema di discovery ha una durata di tempo finita, indicata con il valore *leasetime*, di conseguenza al termine del *leasetime* se il servizio non rinnova la sua registrazione viene automaticamente cancellato. Infatti, nel caso in cui un servizio smetta di funzionare per un problema interno o un problema di rete, il database dei servizi conterrebbe dei dati errati che non rifletterebbero lo stato attuale dei servizi presenti, invece grazie al *lease*, il disallineamento fra lo stato memorizzato e quello effettivo sarebbe solo temporaneo dipendendo dal tempo *leasetime*. Quindi il Leasing Manager gestisce la permanenza delle registrazioni all'interno del database locale, sfruttando il valore di *leasetime* associato alla registrazione stessa. Quando viene inserito un servizio in seguito ad una ricerca del *Service Broker Agent*, viene anche registrato il tempo di scadenza del servizio relativo allo standard di discovery considerato. Il Leasing Manager dovrà quindi gestire la rimozione delle descrizioni servizi dal database interagendo con i *Service Broker Agent* relativi agli standard con cui erano stati registrati, al fine di verificare se un servizio sia o meno stato rimosso anche dal servizio standard di discovery di cui faceva parte.

5.3.2 Inizializzazione del sistema di distribuzione video

Come evidenziato al paragrafo 4.2.2, la fruizione di materiale multimediale attraverso MUM necessita dell'inizializzazione del percorso dal cliente al servitore, chiamato *Service Path* (SP). Sul SP verranno instanziati vari elementi, presenti in figura 5.6.

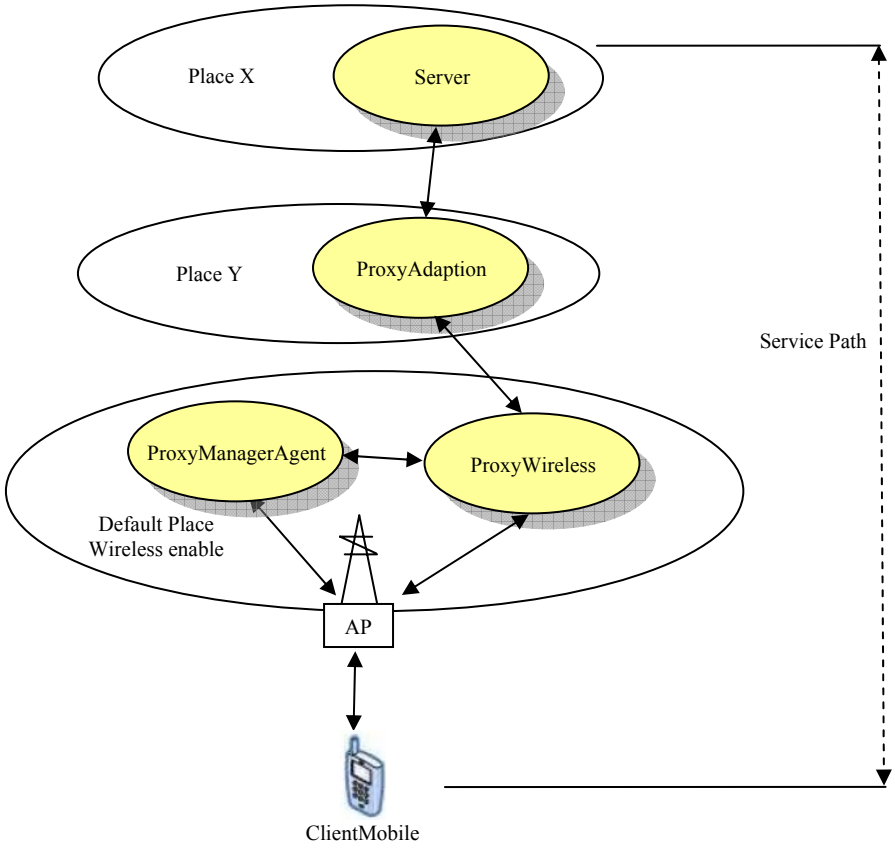


Figura 5.6: il Service Path e i vari elementi presenti.

Il *Server*, è l'entità che gestisce l'accesso al contenuto multimediale, rappresenta il punto di partenza del flusso multimediale. Il *ProxyAdaptor*, è un'entità intermedia che adatta il flusso multimediale in base alle esigenze dei terminali che si occupano della ricezione. Il *ProxyWireless* è un'entità intermedia che funge da ripetitore del flusso, cioè instrada il flusso verso l'utilizzatore finale in base alla locazione corrente di quest'ultimo. Il *ProxyManagerAgent* è un elemento mobile che si occupa del device a cui è direttamente associato, agendo da controllore sugli spostamenti di quest'ultimo. Il *ClientMobile* è un'entità che risiede sul terminale mobile, interagisce con il *ProxyManagerAgent* per la gestione dell'*handoff* e *offload* e con il *ProxyWireless* per la fruizione del materiale multimediale.

Di seguito verranno date delle panoramiche sul comportamento del sistema in caso di *handoff* e *offload*, per poi approfondire le descrizioni degli elementi coinvolti.

5.3.2.1 Gestione dell'handoff

Quando il *ClientMobile* entra nelle vicinanze di uno o più AP diversi da quello a cui è attualmente collegato, in base alla potenza emessa da questi AP, è in grado di stabilire con una certa probabilità verso quali di questi AP potrà collegarsi in futuro. Questa previsione viene inviata dal *ClientMobile* al *ProxyManagerAgent* con un certo anticipo rispetto all'effettiva connessione con il nuovo AP. La previsione arrivata al *ProxyManagerAgent* da origine ad un piano d'azione per stabilire gli elementi da instanziare lungo il percorso, in figura 5.7 viene mostrata questa prima fase di previsione e creazione del piano d'azione.

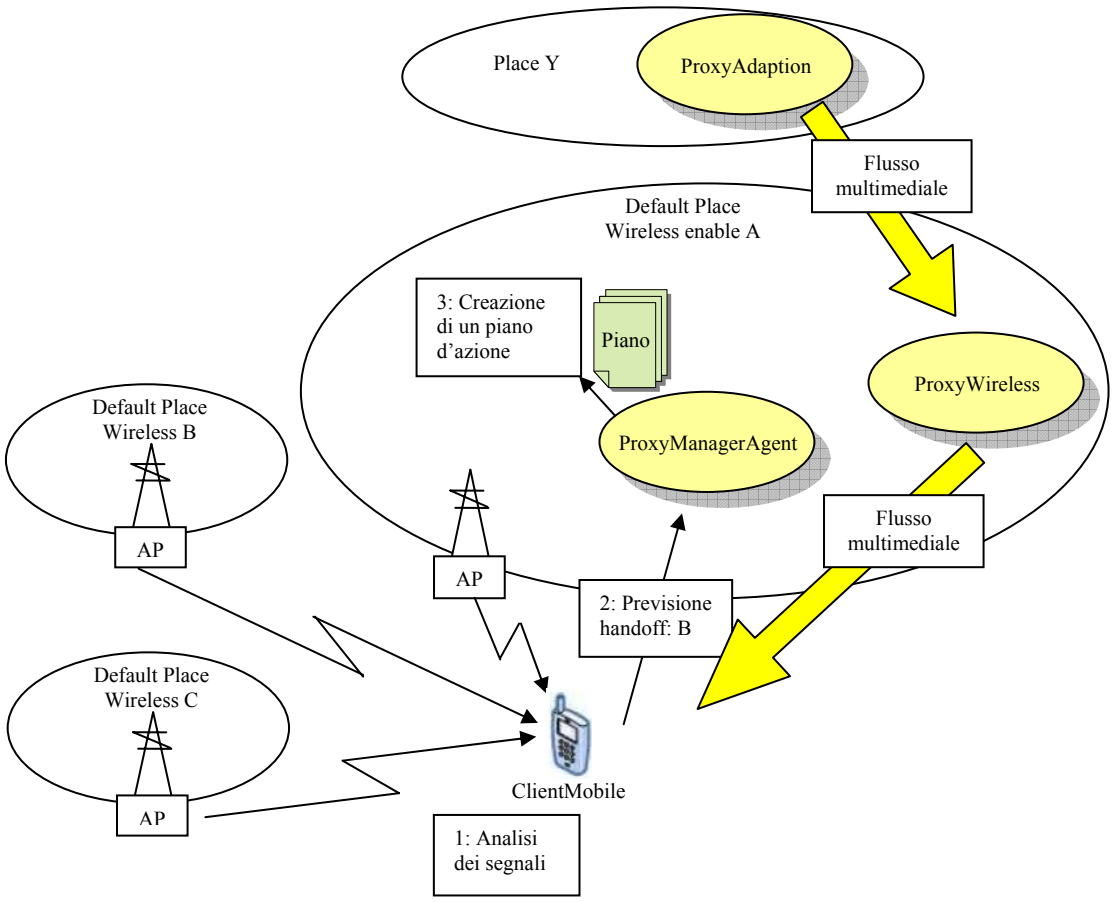


Figura 5.7: Prima fase: la previsione e la generazione di un piano per la gestione dell'handoff

Il piano include, oltre agli elementi da instanziare sulla nuova località, tutte le informazioni della sessione corrente, necessarie a ricostruire una nuova sessione sulla località di destinazione.

In una seconda fase, si ha l'inizializzazione della destinazione prevista. Il *ProxyManagerAgent* crea un *PlanVisitorAgent* il quale, come spiegato nel paragrafo 4.2.4, rappresenta un entità mobile che si occupa di instanziare ed attivare gli elementi indicati nel piano. Lo scopo del piano consiste nella creazione, sulla destinazione prevista, di un *ProxyWireless*, che una volta inizializzato richiederà al *ProxyAdaptor* la duplicazione del flusso video, la quale manterrà nel contempo la stessa politica di adattamento sul contenuto multimediale, figura 5.8.

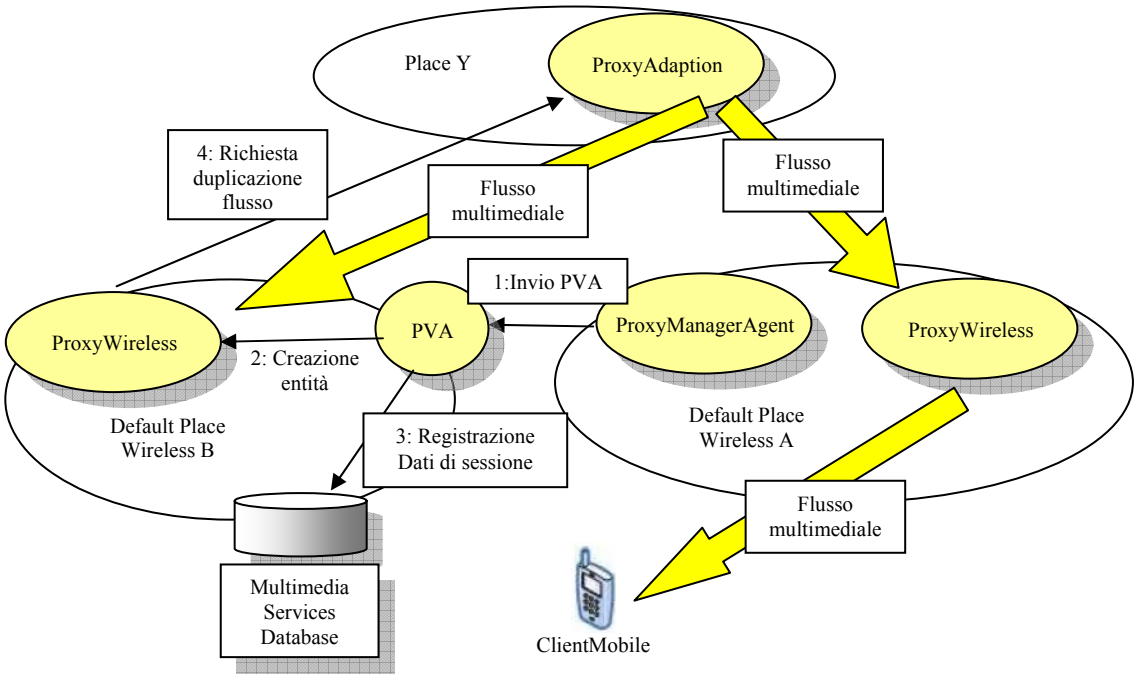


Figura 5.8 Seconda fase: duplicazione del flusso video mantenendo la politica di adattamento.

La creazione di un *ProxyWireless*, comporta anche la registrazione dei dati della sessione nel *Multimedia Service Database* di MUM, questi dati conterranno le informazioni sul cliente, sulla sessione precedente e i dati di collegamento al *ProxyWireless* della nuova località.

Nel terza fase il *ClientMobile*, dopo essersi mosso fisicamente nella nuova località, si riconnette all'infrastruttura. Il *ClientMobile* invia una richiesta di ripresa del contenuto multimediale fornendo l'identificativo di sessione che gli era stato fornito dal sistema al momento del *login* e l'identificativo dell'entità *ProxyManagerAgent* che gestiva la sessione sul nodo precedente. Alla ricezione di questo messaggio

l'infrastruttura verifica se è presente una registrazione di sessione il cui identificativo sia uguale a quello fornito dal cliente.

Nel caso in cui la richiesta abbia esisto positivo, nei dati di sessione sono già presenti i valori di collegamento al nuovo *ProxyWireless*, in quanto sono stati registrati dal *PlanVisitorAgent*, come visto sopra. Di conseguenza l'infrastruttura può già fornirli al *ClientMobile*, che ha quindi la possibilità di riprendere la sessione immediatamente. A questo punto l'infrastruttura conclude il suo compito generando un nuovo *ProxyManagerAgent*, passandogli tutti i dati di sessione e il nuovo indirizzo del *ClientMobile*, figura 5.9

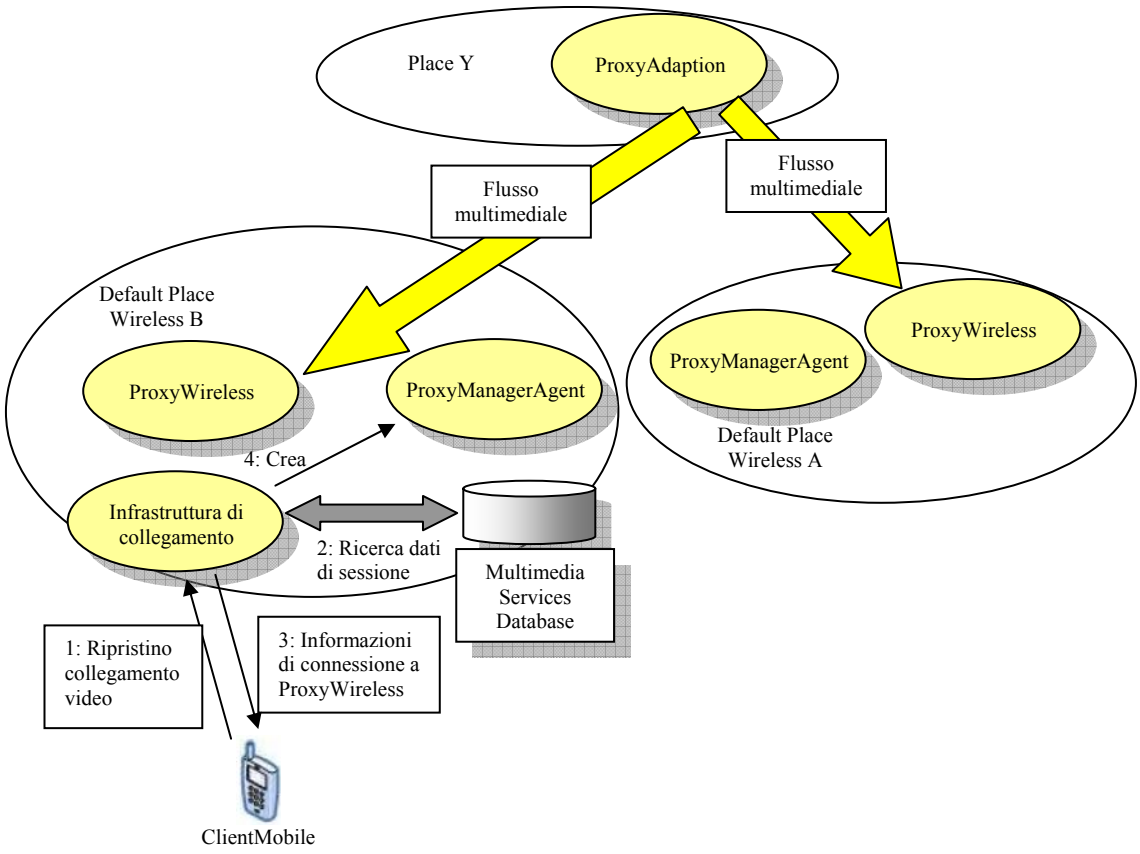


Figura 5.9: Ripristino sessione in caso di previsione corretta.

Il *ProxyManagerAgent* appena creato invierà al *ClientMobile* i suoi dati di collegamento, concludendo così la procedura di handoff da parte del terminale. Durante tutto questo lasso di tempo il ClientMobile non riceve nessun flusso multimediale, ma nonostante questo, la presentazione multimediale è sostenuta lo stesso dalla presenza di un buffer, come spiegato nel paragrafo 4.3.3. A questo punto, nella vecchia locazione,

saranno presenti un *ProxyManagerAgent* e un *ProxyWireless*, che rimarranno ancora attivi per un tempo fissato, nell'eventualità che il terminale mobile si sposti nuovamente nella vecchia locazione, gestendo così la possibilità di movimento ravvicinato fra due AP (problema del ping-pong). Scaduto questo tempo il nuovo *ProxyManagerAgent* distruggerà il *ProxyWireless* e il *ProxyManagerAgent* nella vecchia località, figura 5.10.

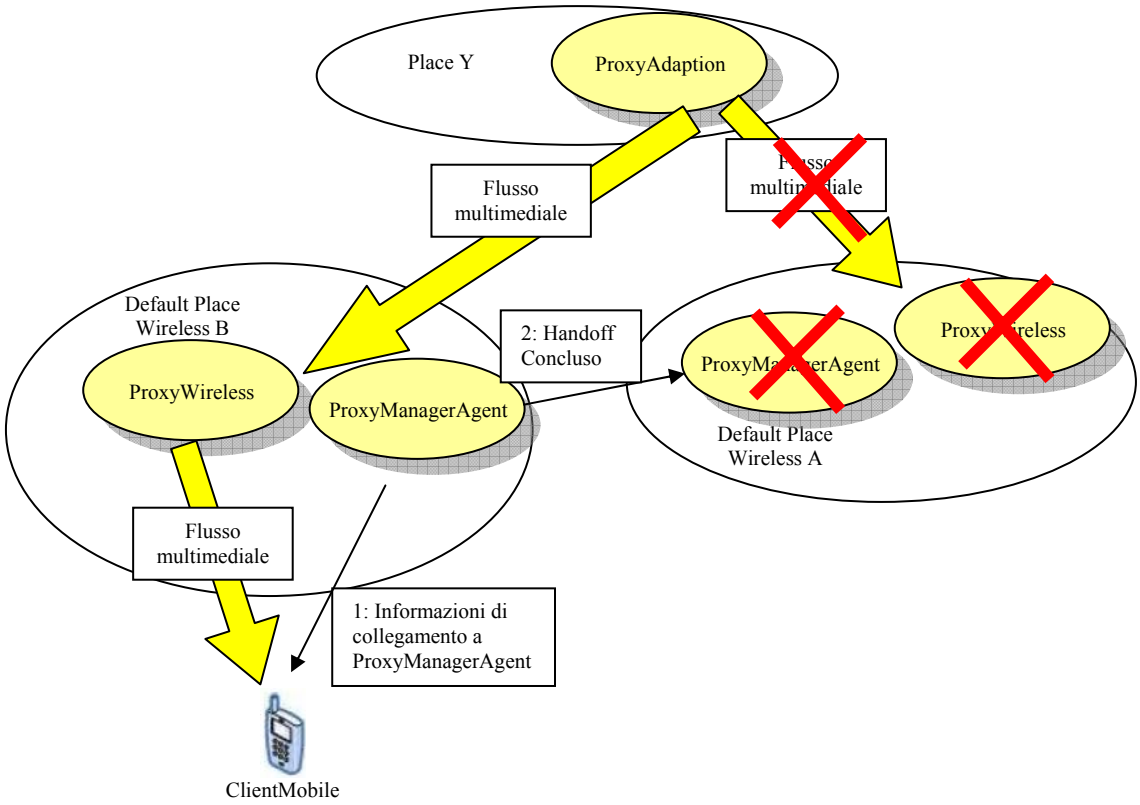


Figura 5.10: Handoff terminato.

Nel caso in cui la richiesta desse esito negativo, possono esserci due casi; un primo caso in cui la previsione sia corretta, ma il *PlanVisitorAgent* non sia stato abbastanza veloce nella creazione della sessione e secondo caso in cui la previsione sia effettivamente errata. In entrambi l'infrastruttura invia al *ProxyManagerAgent* nella vecchia locazione, i dati relativi al collegamento e alla locazione del *ClientMobile*, in modo che il *ProxyManagerAgent* sia in grado da questi dati di sapere se il *ClientMobile* è arrivato nella locazione prevista o in un'altra locazione e reagire di conseguenza. Se la destinazione è quella corretta, allora il *ProxyManagerAgent* aspetta che il *PlanVisitorAgent* completi il suo compito, poi migra nella nuova locazione, e una volta

arrivato, informa subito il *ClientMobile* sui dati del *ProxyWireless* ed elimina la sessione sul nodo precedente.

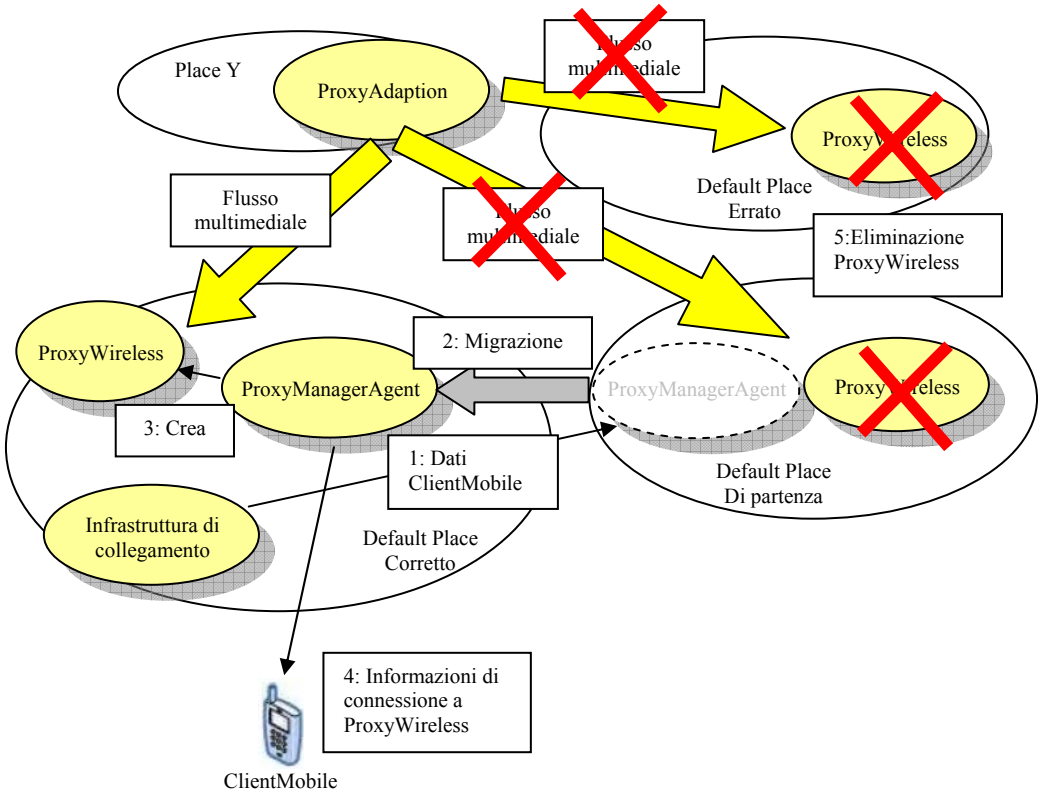


Figura 5.11: Caso di previsione errata.

Se la destinazione prevista non coincide con quella effettiva, allora il *ProxyManagerAgent* come prima cosa migra nella nuova località, ricrea il *ProxyWireless* ed invia le relative informazioni al *ClientMobile*, concluse queste operazioni elimina i due *ProxyWireless* sulla locazione precedente e su quella errata, figura 5.11.

5.3.2.2 Gestione dell'offload

In modo analogo al paragrafo precedente verrà presentata una panoramica della procedura di *offload*. Durante la sessione di *streaming*, il *ProxyManagerAgent* effettua periodicamente delle ricerche nel database del servizio di discovery del Default Place in cui si trova, al fine di trovare una postazione fissa dotata di uno schermo che

venga incontro alle preferenze dell'utente contenute nel suo profilo, nel caso in cui non la trovi la ricerca continua nell'eventualità che un servizio di quel tipo di liberi o si aggiunga alla lista dei servizi disponibili. Una volta ricavate le preferenze, viene creato un *ServiceTemplate* al cui interno sono specificati degli attributi che identificano dei particolari terminali fissi il cui unico scopo è fornire una postazione di visualizzazione per contenuti multimediali, indicati con il termine generico di Servizio di *Display*. Il servizio di discovery fornisce una lista di tali servizi, che andrà poi filtrata, ottenendo solo gli schermi che presentano la risoluzione e la dimensione comprese nei valori specificati dall'utente. Dalla lista verrà poi scelto lo schermo dalle caratteristiche migliori e dalla sua descrizione, cioè dal suo *ServiceItem*, verrà ricavato il Place nel quale sarà localizzato, figura 5.12.

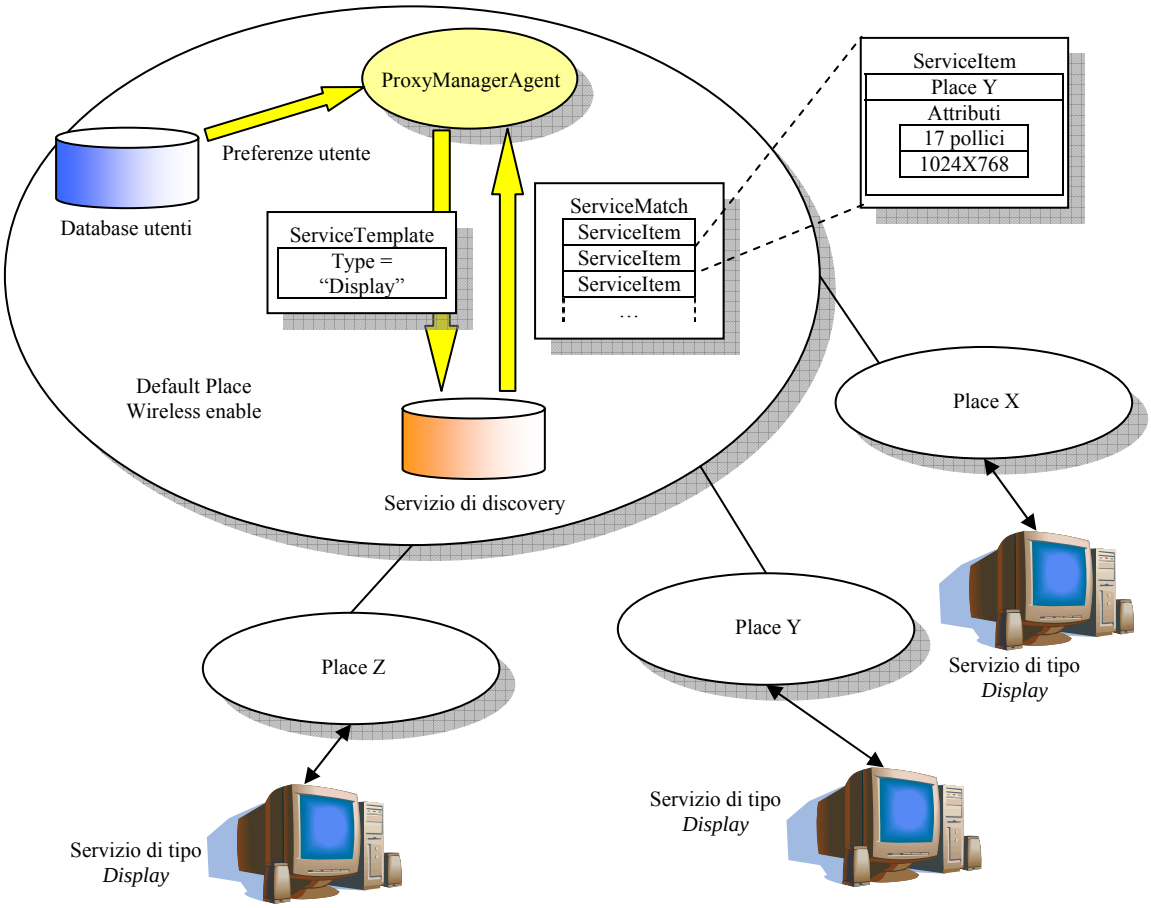


Figura 5.12: Ricerca preferenze e servizi.

A questo punto *ProxyManagerAgent* da inizio alla procedura di inizializzazione preventiva del nodo, in modo che se l'utente decidesse di spostare la sessione, il cambio sarà molto più veloce perché già avviato. Analogamente al caso dell'*handoff*, il

ProxyManagerAgent crea un piano per l'*offload*. Il piano oltre a contenere i componenti da instanziare, potrebbe contenere anche dei nuovi adattatori da inviare al *ProxyAdaptor*, nel caso in cui il Servizio di *Display* non fosse in grado di supportare la piena qualità del flusso, riadattando il contenuto multimediale. Tali adattatori potrebbero non essere necessari nel caso in cui il Servizio di *Display* fosse in grado di supportare la piena qualità. Il piano così ottenuto viene passato al *PlanVisitorAgent*, quando questo arriverà sul Place contenente il servizio, instanziarà tutti i componenti presenti nel piano, in particolare un *Client Video* necessario per la visualizzazione sul Place di destinazione, figura 5.13.

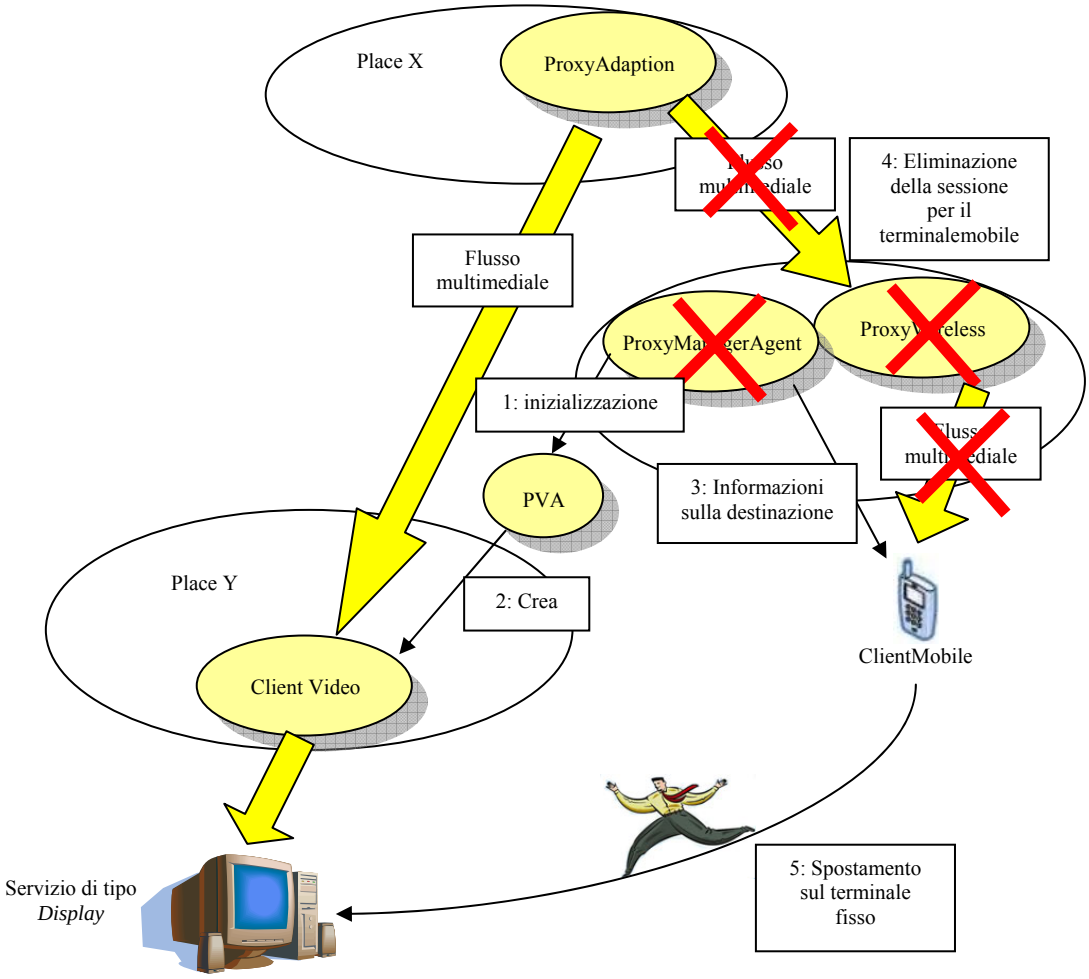


Figura 5.13: Inizializzazione del Place contenente il servizio e riadattamento.

Una volta che il Place è stato inizializzato, viene inviato un messaggio al *ClientMobile* che informa l'utente della possibilità di cambiare terminale, insieme alle informazioni sulla posizione e sulle caratteristiche di quest'ultimo. Nel caso di conferma positiva da parte dell'utente allora inizierà lo *streaming* sul nuovo terminale e la sessione sul

ClientMobile sarà distrutta. Nel caso di conferma negativa, gli elementi instanziati nel Place saranno distrutti, liberando le relative risorse impegnate.

5.4 Analisi Entità principali

Nei prossimi paragrafi verranno esaminate le entità che costituiscono il Service Path presentato in figura 5.6.

5.4.1 ClientMobile

Come già ampliamente spiegato nelle sezioni precedenti un dispositivo mobile non ha la possibilità di eseguire un'infrastruttura complessa come SOMA. *ClientMobile* rappresenta l'end-point del SP che riceve i flussi multimediali dall'infrastruttura MUM. È composto da una parte relativa alla comunicazione con SOMA e MUM, una semplice interfaccia grafica e un gestore del protocollo RTP.

Prima di procedere alla distribuzione effettiva del video, è necessario che sia effettuato un *login* dell'utente. Ogni utente è registrato all'interno dell'infrastruttura MUM in un database dei profili utente, che contiene tutti i dati e le preferenze dell'utente. Al fine di includere le informazioni di dispositivi mobili è stato creata una nuova categoria di profili, il *MobileUserProfile*, che contiene i seguenti dati:

- identificativo dell'utente, nome e cognome, dati ereditati dal profilo standard di MUM;
- lista dell'identificativo e le rispettive caratteristiche dei terminali mobili utilizzati dall'utente, dove le caratteristiche sono descritte secondo lo standard CC/PP;

preferenze dell'utente in caso di *offload* di sessione su un nuovo terminale, queste preferenze riguardano la risoluzione e la dimensione dello schermo di tale terminale minima e massima, questa scelta è dovuta al fatto che nello scenario presentato all'inizio del capitolo, diversi servizi potrebbero avere costi diversi. Una volta riconosciuto l'utente ed individuato il dispositivo mobile utilizzato, gli viene assegnato un identificativo di sessione, utilizzato per riprendere la sessione video in caso di

handoff, come specificato nel paragrafo 5.3.2.1. L'utente a questo punto si mette in attesa della fine della procedura di inizializzazione delle varie entità che sono necessarie per la fornitura del servizio video, una volta che questa procedura è stata portata a termine sul *ClientMobile* viene avviato il gestore del flusso RTP ed inizia la fase di buffering, figura 5.14.

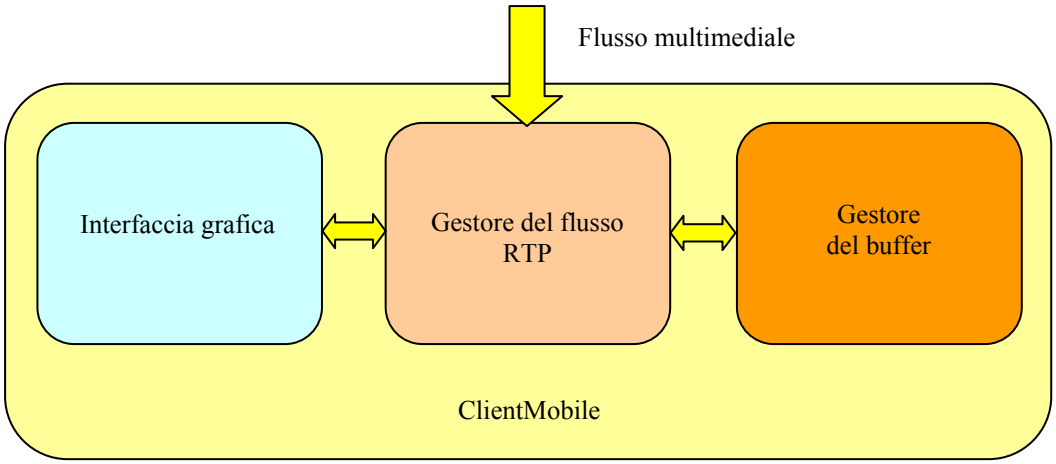


Figura 5.14: Struttura del ClientMobile

Durante la riproduzione il dispositivo potrebbe muoversi da un AP ad un altro AP, prima di eseguire questa operazione, il dispositivo è in grado effettuare con una buona approssimazione, la previsione dell'AP con il quale si collegherà, come evidenziato al paragrafo 5.3.2.1. Questa previsione verrà spedita al *ProxyManagerAgent*, in modo che possa reagire di conseguenza duplicando preventivamente la sessione sulla possibile destinazione in modo da diminuire i tempi che intercorrono per lo spostamento della sessione su un diversa rete. La previsione potrebbe essere anche errata, di conseguenza il sistema deve essere in grado anche di gestire questa possibilità, eliminando la sessione creata sul nodo errato e ricreare la sessione nella locazione effettiva del cliente.

5.4.2 ProxyManagerAgent

L'accesso al sistema SOMA e MUM da parte del *ClientMobile* che risiede sul terminale mobile, avviene attraverso un Default Place collegato alla rete wireless tramite un AP, che d'ora in poi verrà chiamato Default Place Wireless. Sul Default Place Wireless è presente il componente *MobileClientListener*, la cui funzione è quella

di ascoltare e gestire le richieste di terminali mobili. La procedura iniziale di *login* è affidata a questo componente, il quale interagendo con il database dei profili degli utenti, riconosce i dati, le preferenze e il dispositivo mobile che sta utilizzando attualmente l'utente. Una volta che l'utente è stato autenticato, la fase di *login* termina e il *MobileClientListener* avvia il *ProxyManagerAgent*, che rappresenta l'elemento fondamentale del sistema di distribuzione video, in quanto si occuperà di tutte la procedura di inizializzazione, di quelle di *handoff* e di quelle di *offload* su un terminale fisso, in figura 5.15.

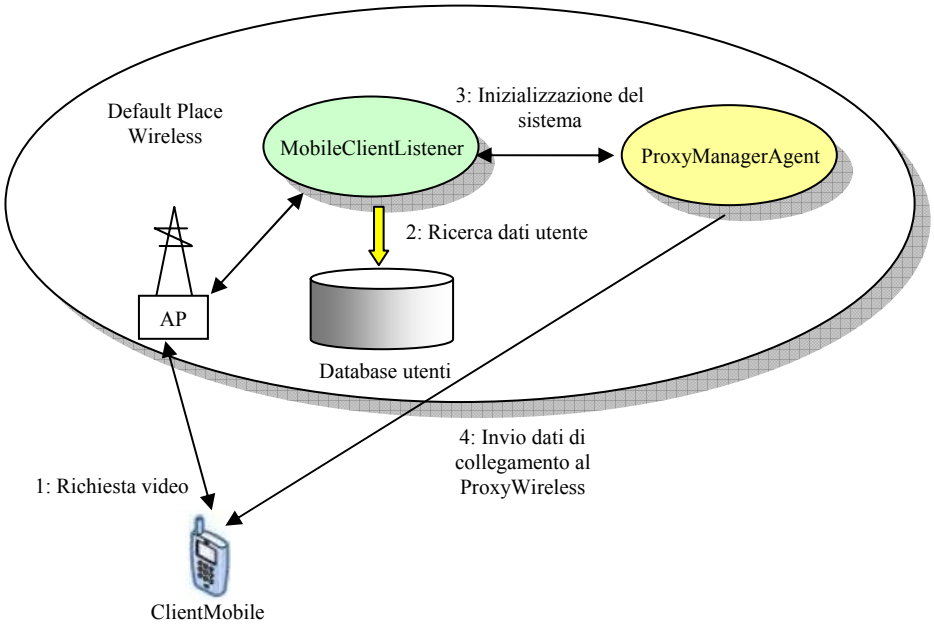


Figura 5.15: Procedura iniziale di login

A questo punto il *ProxyManagerAgent* da inizio alla procedura di configurazione del SP. Come prima cosa richiede al *DecisionMakerWireless* un piano per l'inizializzazione dei vari elementi lungo il SP. Il *DecisionMakerWireless* è una versione modificata del *DecisionMaker* di MUM (si veda paragrafo 4.2.4), in grado di fornire un piano specifico per il *ClientMobile*. Il *ClientMobile*, come evidenziato nel paragrafo precedente, risiede su un terminale mobile, di conseguenza il flusso video che partirà dalla sorgente ad una qualità alta, dovrà essere adattato per venire incontro alle ridotte possibilità del dispositivo. Il piano conterrà quindi delle informazioni relative all'adattamento del contenuto multimediale. Queste informazioni verranno ricavate dal profilo utente, che contiene i dispositivi utilizzati e le loro caratteristiche, descritte attraverso lo standard CC/PP e il vocabolario UAProf, come evidenziato in [Mon04]. Il

piano ottenuto verrà passato al *PlanVisitorAgent*, che si occuperà di instanziare su i vari Place che compongono il percorso gli elementi del piano, verificando che le risorse da impiegare sui vari Place del percorso siano sufficienti. Quando l'inizializzazione sarà completata il *ProxyManagerAgent* invierà al *ClientMobile* i dati necessari per la connessione della sessione RTP e il conseguente *streaming* del video.

Nel caso in cui il *ClientMobile* decida di spostarsi, come è stato evidenziato nel paragrafo precedente, lo spostamento sarà preceduto da una previsione dell'AP nel quale il *ClientMobile* potrebbe arrivare. Nota questa previsione il *ProxyManagerAgent* procede richiedendo al *DecisionMakerWireless* il piano di handoff, che verrà passato al *PlanVisitorAgent* in modo che possa instanziare gli elementi necessari sulla destinazione prevista. Nel caso in cui la previsione risultasse errata, allora sarà necessario eliminare gli elementi instanziati sulla destinazione prevista e instanziarli sulla destinazione effettiva. Alla fine di questa procedura il *ProxyManagerAgent*, eliminerà gli elementi presenti sulla locazione precedente, liberando di conseguenza le risorse impegnate.

Il *ProxyManagerAgent* durante la fruizione del materiale multimediale, si occupa anche di effettuare una ricerca utilizzando il servizio di discovery che si trova nella località attuale del *ClientMobile*. Tale ricerca, basata sulle preferenze contenute nel profilo dell'utente, riguarda la presenza di un terminale fisso che permetta, grazie alle caratteristiche del suo monitor registrate nel sistema di discovery, una migliore visualizzazione del filmato. Nel caso in cui la ricerca abbia esisto positivo, il *ProxyManagerAgent* richiede al *DecisionMakerWireless* un piano per l'*offload* dal terminale mobile a quello fisso. Quando gli elementi sul Place saranno operativi, il *ProxyManagerAgent* informerà il *ClientMobile* della possibilità di *offload* della sessione su questo nuovo terminale.

5.4.3 ProxyWireless

L'architettura del Proxy di MUM è organizzata in modo da separare la gestione della sessione, ottenuta mediante l'invio dei comandi provenienti dal client ed inoltrati al server, da quella dei flussi che al contrario provengono dal server per essere ridiretti

sul client, figura 5.16. Il *ProxyWireless* è l'elemento che nella catena di distribuzione dei flussi si trova subito prima del *ProxyAdaptor*, il cui compito è quello di inoltrare il flusso proveniente da quest'ultimo al *ClientMobile*. Questo elemento assume un ruolo fondamentale nel processo di *handoff*. All'arrivo della previsione di *handoff* da parte del *ClientMobile*, il *ProxyManagerAgent* crea il nuovo piano da passare al *PlanVisitorAgent*, il quale istanzia sul nodo di arrivo il *ProxyWireless*, che permetterà di riprendere il flusso video. Una volta che questa procedura è completata, *ProxyWireless* attende di essere contattato dal *ClientMobile* nel caso in cui la previsione sia corretta oppure dal *ProxyManagerAgent* per essere distrutto nel caso in cui la previsione fosse errata.

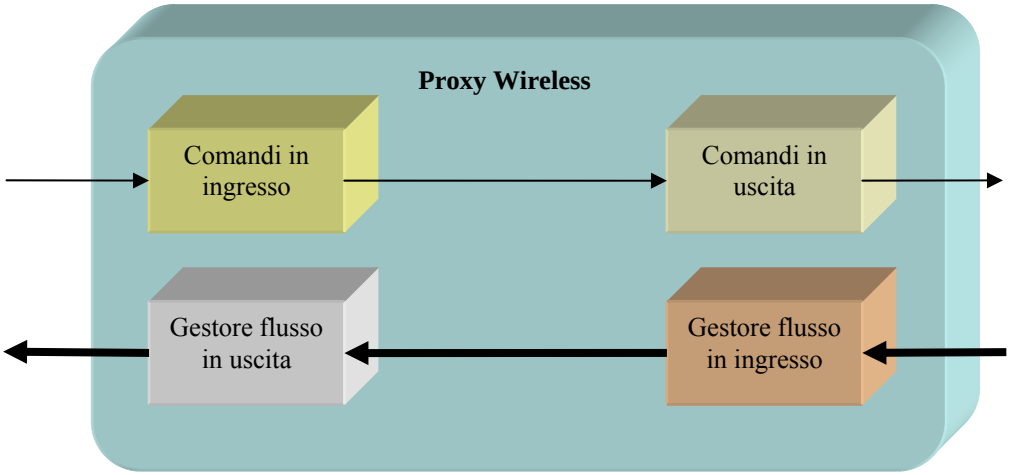


Figura 5.16: Struttura di un ProxyWireless

5.4.4 ProxyAdaptor

Come accennato al paragrafo precedente, i proxy hanno una parte relativa alla gestione dei comandi e una parte relativa all'invio del flusso video verso il cliente. Il *ProxyAdaptor* prima dell'invio del flusso verso l'elemento che lo precede nel SP, inserisce degli adattatori il cui compito è quello di trasformare il contenuto multimediale in modo da venir incontro alle caratteristiche del dispositivo in cui il contenuto multimediale sarà visualizzato. In figura 5.17 è presentata la struttura di *ProxyAdaptor*.

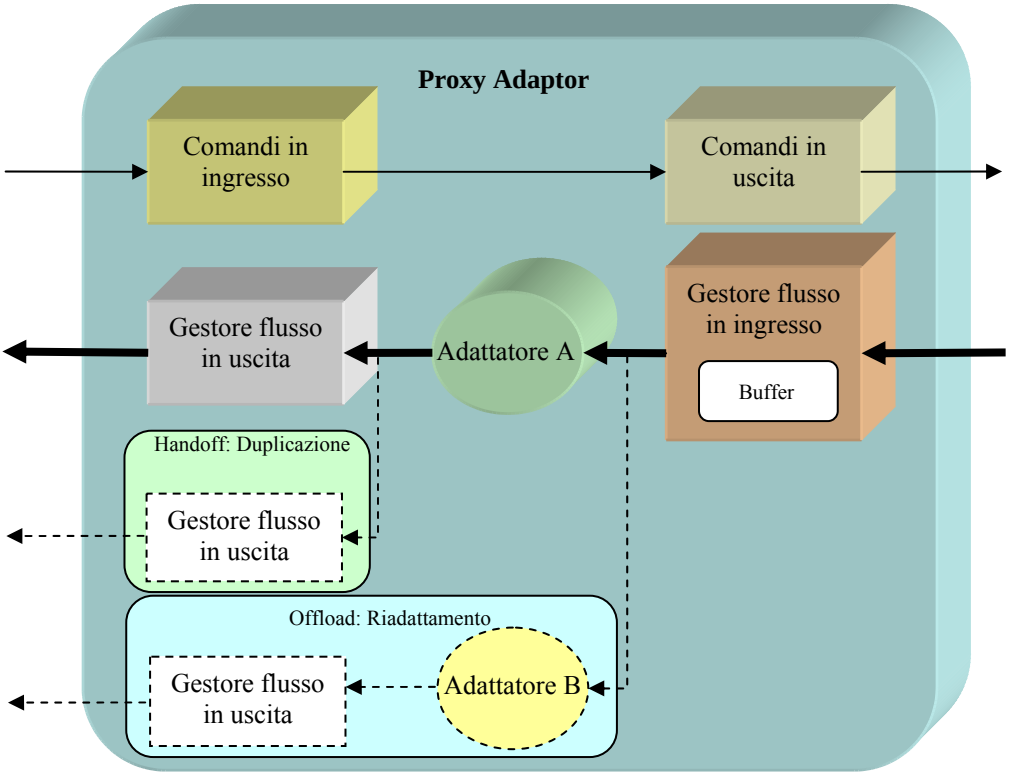


Figura 5.17: Struttura di un ProxyAdaptor

Nella figura 5.17 è evidenziato anche cosa succede quando si presenta un *handoff* e quando si presenta un *offload*: in caso di *handoff*, il flusso video viene clonato dopo la parte relativa all'adattamento, mantenendo quindi la stessa politica di adattamento del flusso diretto verso il *ProxyWireless* che viene duplicato sulla destinazione prevista dall'*handoff*, in caso di *offload*, il flusso viene clonato prima dell'adattamento, in modo da avere la piena qualità ed in seguito viene applicato un altro adattatore diverso da quello presente in origine, questo adattatore può anche non essere presente nel caso in cui il terminale fisso sia in grado di gestire il flusso a qualità piena.

Come è stato evidenziato nei paragrafi precedenti, il *ClientMobile* dispone di un buffer utilizzato nei momenti in cui il flusso viene interrotto per via dell'handoff. Anche il *ProxyAdaptor* dispone di un buffer, ma con una funzione diversa. Nella prima fase di riempimento del buffer del *ClientMobile*, questo buffer permette al *ProxyAdaptor* di accelerare la velocità del flusso inviato, in modo che il *ClientMobile* possa riempire il suo buffer ad una velocità maggiore. Nella fase di handoff, invece, il buffer del *ProxyAdaptor* permette di riavvolgere il flusso evitando al *ClientMobile* di perdere quei *frame* che durante lo spostamento sono stati spediti, ma non sono stati ricevuti perché persi nel canale di comunicazione.

5.4 Conclusioni

In questo capitolo sono stati fissati i requisiti che il sistema dovrà soddisfare. Dopo una loro prima analisi, il sistema in esame è stato suddiviso in moduli fondamentali e per ognuno di essi l'analisi ha permesso di individuare servizi e comportamenti che dovranno realizzare per assolvere alle loro responsabilità.

Per quanto riguarda il servizio di discovery è stata individuata una struttura logica che guiderà le fasi successive dello sviluppo del progetto.

Mentre per quanto riguarda il sistema di distribuzione di contenuti multimediali su dispositivi mobili, sono state individuate le entità fondamentali che intervengono nelle fasi di inizializzazione, di *handoff* e di *offload*, e che verranno approfondite nei capitoli successivi.

Nel prossimo capitolo verranno esposti il progetto e l'implementazione delle entità costituenti il sistema realizzato.

CAPITOLO 6

PROGETTO DELL'INFRASTRUTTURA

DI

DISCOVERY E ADATTAMENTO

Questo capitolo sarà dedicato alla progettazione dei componenti e dei servizi evidenziati precedentemente nella fase di analisi. L'obiettivo della fase di progettazione è quello di stabilire come possono essere realizzate ed assolte funzionalità e responsabilità delle varie entità precedentemente analizzate, studiando in dettaglio i comportamenti e le interazioni fra le varie parti del sistema. In generale, in questa fase dello sviluppo, sarebbe auspicabile rimanere ancora slegati dall'ambiente in cui il sistema verrà implementato, tuttavia essendo esso parte di un Middleware costruito sulla piattaforma SOMA, realizzata in linguaggio Java, si orienterà lo sviluppo dell'architettura verso il modello ad oggetti.

La struttura del capitolo segue quella dell'architettura logica prodotta in fase di analisi che divide le varie entità che compongono il sistema.

6.1 Progetto del servizio di discovery

6.1.1 Discovery Manager

Come visto in fase di analisi, il sistema di discovery è costituito da due layer, il *Facilities layer* e il *Services layer*. Il *Facilities layer* permette di accedere al sistema di discovery fornendo funzioni base del sistema, rappresenta quindi un'astrazione di alto livello del servizio, rivolta agli sviluppatori che intendono solo ricercare dati e inserire servizi direttamente nella cache per uso interno all'infrastruttura. Il *Services layer* contiene invece le funzioni di più specifiche, rappresentando quindi un'astrazione di

basso livello, rivolta agli sviluppatori che intendono cimentarsi nella creazione di un *ServiceBrokerAgent*. In figura 6.1 è presentata l'interfaccia del Discovery Manager, chiamata *IDiscoveryManager*, tale entità incapsula la logica di accesso ai sistemi di discovery standard e al database contenente i servizi per sistema di discovery locale, ma non interessa in fase progettuale stabilire come sarà realizzato tale database, rimandando il dettaglio nella fase implementativa.

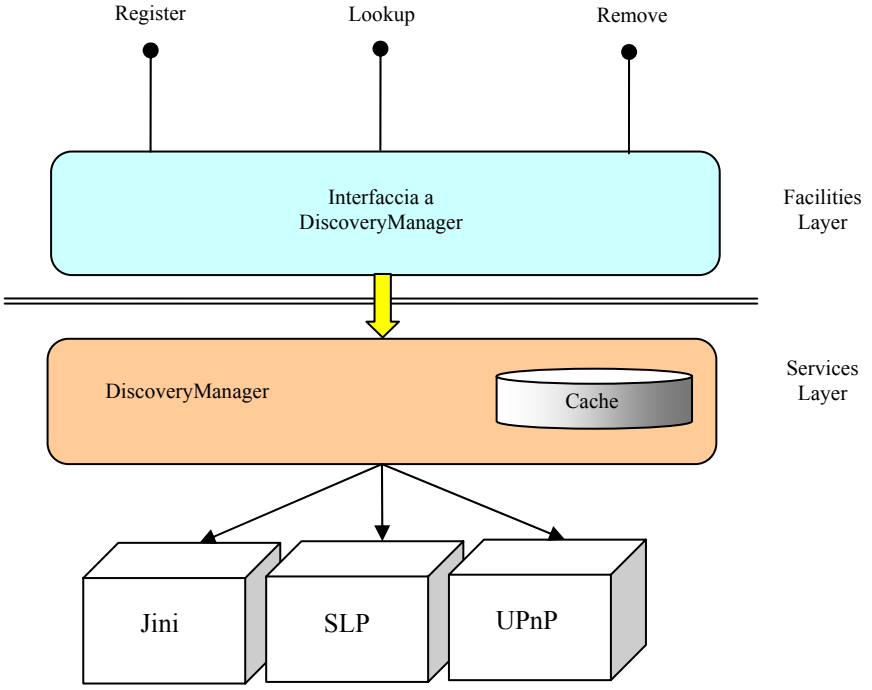


Figura 6.1: Interfaccia IDiscoveryManager

Con la costante *LEASE_INFINITY* viene indicato il caso di una registrazione con tempo di lease infinito. Il metodo register accetta in ingresso la descrizione del servizio rappresentata da *ServiceItem* e una variabile *tlease* rappresentante il tempo di durata del lease espresso in millisecondi. I metodi *getItem* e *remove_service* agiscono sul database rispettivamente aggiungendo od eliminando i servizi specificati come parametri attraverso l'identificativo o attraverso il *SeviceItem*. Il metodo *lookupservices* permette di recuperare dal database tutti i servizi che presentano le caratteristiche specificate nel *ServiceTemplate*, la ricerca avviene solo nel Default Place della località corrente, in quanto come specificato nella fase di analisi, ogni Default Place contiene solo i dati dei Place figli che possono ospitare un servizio o fungere da mediatori con questo, se la ricerca dovesse essere estesa a domini superiori, è possibile creare un agente che muovendosi fra i vari domini raccolga i possibili risultati.

Nel *Services layer* sono presenti invece gli elementi che realizzano il servizio di discovery vero e proprio, figura 6.2.

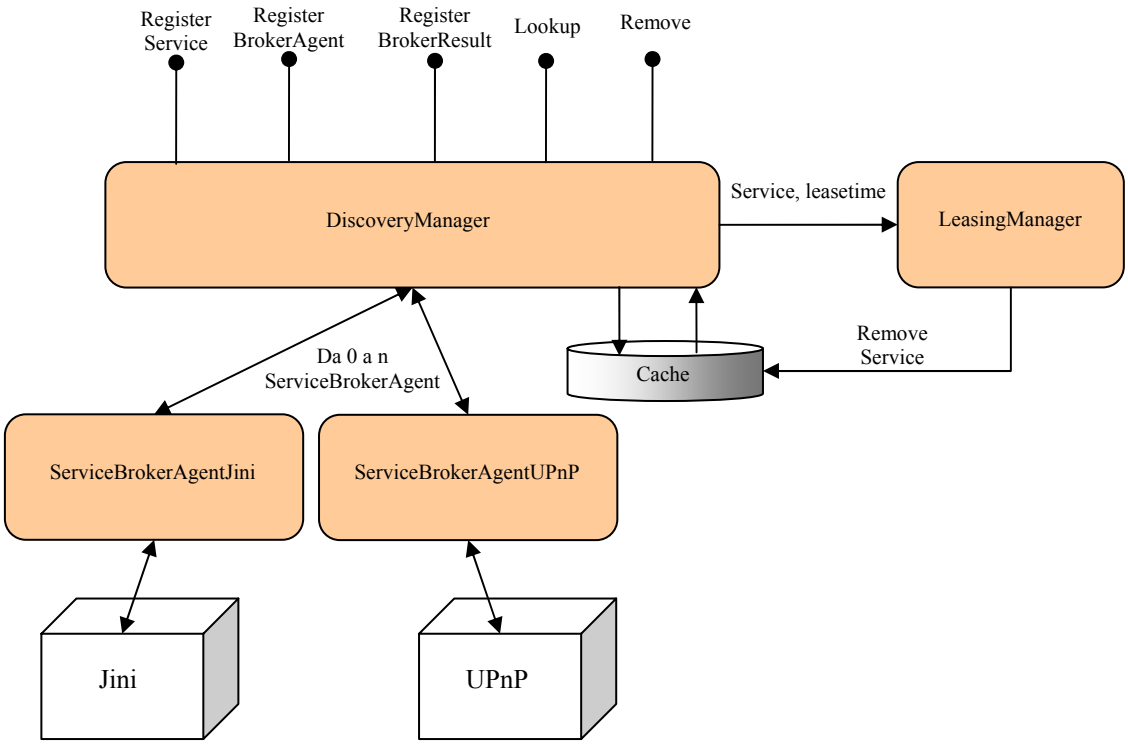


Figura 6.2: Struttura del Services Layer

Il *DiscoveryManager* registra i servizi in un database, che ha come chiave di ricerca l'identificativo assegnato al servizio, generato ad ogni inserimento in modo sequenziale, e come elemento l'oggetto *ServiceItem*. I metodi che operano sulla tabella agiscono in mutua esclusione, in modo che non ci siano accessi di lettura e scrittura contemporanei nel database.

I *ServiceBrokerAgent* al momento della creazione devono registrarsi presso il *DiscoveryManager*, quest'ultimo interagisce con loro per la ricerca nello standard di discovery con il quale fungono da mediatori, inviando un messaggio di richiesta di lookup dove viene indicato il *ServiceTemplate* del servizio che si intende ricercare, i *ServiceBrokerAgent* dopo aver effettuato la ricerca, rispondono registrando i loro risultati nel *DiscoveryManager*.

Nel momento della registrazione del servizio, se è stato specificato un valore di tempo di lease diverso da *LEASE_INFINITY*, quel particolare servizio viene associato con il *LeasingManager*, al quale verrà passato l'identificativo del servizio e il tempo di lease.

6.1.2 Descrittore dei servizi

Come evidenziato nella fase di analisi la descrizione dei servizi avviene per mezzo di oggetti *ServiceItem*. L'oggetto *ServiceItem* è serializzabile di modo che sia possibile trasformarlo in un flusso di byte per poterlo trasferire sulla rete, questa caratteristica è importante perché in questo modo descrizioni di servizi possono migrare insieme agli agenti ed essere trasferite da un Place ad un altro, requisito necessario affinché servizi, che si trovano su una località diversa dal *Default Place* dove è collocato il *DiscoveryManager*, possano registrarsi.

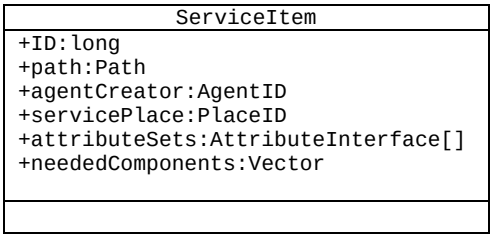


Figura 6.3 Il ServiceItem

I campi contenuti all'interno di *ServiceItem* (figura 6.3) che rappresentano il servizio sono:

- *ID*: un campo numerico che rappresenta l'identificativo del servizio, questo valore è lo stesso che viene utilizzato dal *DiscoveryManager* come indice per il database;
- *path*: è un oggetto che rappresenta il *Path* in SOMA che rappresenta il cammino dal nodo radice al nodo che contiene il servizio o un mediatore per quest'ultimo;
- *agentCreatore*: è l'identificativo dell'agente SOMA che ha creato il servizio e che ne è il mediatore, questo valore potrebbe essere nullo, nel caso in cui non sia presente un mediatore per quel specifico servizio;
- *servicePlace*: è un oggetto di tipo *PlaceID* che indica il Place dove è reperibile il servizio, questo valore potrebbe essere nullo nel caso in cui il servizio non risieda su un Place SOMA e quindi indipendente da quest'ultima piattaforma;
- *attributeSets*: è un array di oggetti di tipo *AttributeInterface* che rappresentano attributi del servizio generici;
- *neededComponents*: è un vettore di stringhe, che rappresentano le classi che devono essere caricate per interagire con il servizio, può essere vuoto nel caso in cui non sia

necessario software particolare per la gestione del servizio. La capacità di scaricare il codice al momento del bisogno tramite il *SoftwareRepository* di MUM, rende questo campo molto versatile, permettendo di avere a disposizione codice sempre aggiornato per quel specifico servizio.

Al momento della creazione del servizio, il campo *ID* può essere settato con il valore *NOT_ASSIGNED* nel caso in cui la registrazione avvenga per la prima volta, oppure con il valore della chiave nel database se si tratta di un aggiornamento della descrizione del servizio.

6.1.3 Attributi dei servizi

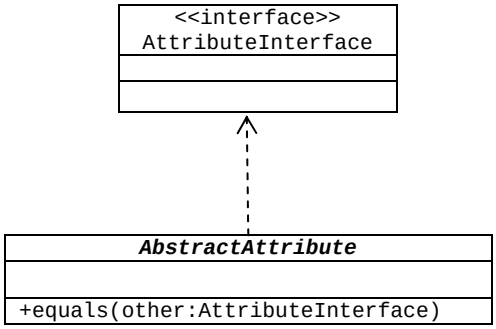


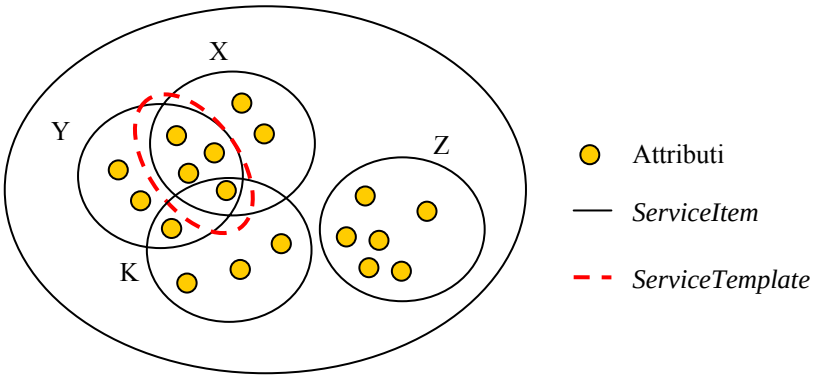
Figura 6.4 Attributi

Ogni attributo deriva dalla classe astratta di base *AbstractAttribute* che a sua volta implementa dall'interfaccia *AttributeInterface*, come si vede in figura 6.4. L'interfaccia *AttributeInterface* funge solo da marcatore per indicare il generico attributo riferito dal *ServiceItem*, mentre *AbstractAttribute* definisce il metodo *equals*, che sfruttando la riflessione fornita dai linguaggi ad oggetti, permette di facilitare la creazione di attributi, fornendo la possibilità di analizzare la struttura del generico attributo che deriva da *AbstractAttribute* leggendo tutti i suoi possibili campi dati. In questo modo il metodo *equals* confronta tutti i campi che costituiscono l'attributo senza necessariamente realizzare un metodo personalizzato per ogni specifico attributo.

6.1.4 Ricerca di servizi

La ricerca di un servizio avviene tramite l'utilizzo della classe *ServiceTemplate*, che rappresenta il modello di un generico servizio. Al suo interno contiene esclusivamente gli attributi che saranno confrontati con tutti i servizi presenti nel database, sfruttando il metodo *equals* visto nel paragrafo precedente. Il servizio di ricerca restituisce come risultato l'oggetto *ServiceMatch* che contiene al suo interno una lista di *ServiceItem*, nei quali gli attributi coincidono con quelli del *ServiceTemplate* fornito.

L'algoritmo di ricerca nella cache agisce leggendo dal database ogni singolo servizio presente ed effettuando un confronto con gli attributi del modello fornito come parametro. Quando il tipo di attributi presenti nel modello è un sottoinsieme di quelli presenti nel servizio analizzato e i valori contenuti negli attributi è lo stesso allora il servizio è compatibile con il modello e viene inserito nella lista che verrà restituita a colui che ha effettuato la richiesta, figura 6.5.



X, Y fanno match con il *ServiceTemplate*, mentre K e Z no

Figura 6.5: Rappresentazione grafica della ricerca.

Indicando con *TA* gli attributi del modello, con *SA* gli attributi del servizio considerato e indicando con *TA(n)* e con *SA(n)* il generico attributo n-esimo, SA fa parte dei risultati se $\forall n$ indice indicante gli attributi di *TA* vale:

$$\begin{cases} TA \subseteq SA \\ TA(n) = SA(n) \quad \forall n \end{cases}$$

Qualsiasi altro confronto diverso dall'uguaglianza deve essere effettuato in un secondo momento utilizzando la lista dei servizi ottenuta.

Entrambe le classi *ServiceTemplate* e *ServiceMatch* sono serializzabili, di modo che tali informazioni possano essere trasferite tra host diversi.

6.1.5 Descrizione di un servizio di visualizzazione video

Nella presente tesi, la descrizione di servizi video, ossia dispositivi il cui scopo è fornire uno schermo dalle migliori prestazioni, per la visualizzazione di contenuti multimediali assumono un ruolo rilevante rispetto a qualsiasi altro servizio. La descrizione di tali servizi avviene utilizzando i seguenti attributi:

- *Name*: è un attributo che specifica il nome assegnato al servizio, come ad esempio il modello del monitor;
- *Location*: indica la posizione del dispositivo di visualizzazione, in particolare è costituita dai campi *floor*, *room* e *building*, tre variabili che indicano rispettivamente piano, stanza e l'edificio dove è localizzato il monitor;
- *Type*: è una variabile indicate il tipo di dispositivo;
- *DisplaySize*: una variabile che rappresenta la dimensione dello schermo in pollici;
- *DisplayFrameSize*: un oggetto *Dimension* che rappresenta la risoluzione orizzontale e verticale dello schermo ;
- *DeviceState*: un *flag* che indica lo stato corrente del dispositivo, i valori assunti possono essere *STANDBY* o *WORKING*, nel caso in cui rispettivamente il dispositivo sia libero od occupato.

6.1.6 LeasingManager

Il *LeasingManager* è un oggetto attivo, il cui compito è verificare la scadenza dei servizi registrati nel database locale del *DiscoveryManager*. Al momento dell'inizializzazione di un servizio, il *LeasingManager* registra il valore dell'identificativo e il tempo ad esso associato in una tabella.

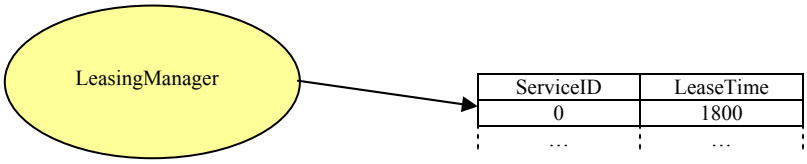


Figura 6.5: LeasingManager e la tabella contente i tempi di lease

Periodicamente aggiorna i valori dei tempi all'interno della tabella e al momento della scadenza, rimuoverà il servizio associato all'identificativo fornitogli in precedenza dal *DiscoveryManager*, nel caso in cui il servizio fosse rinnovato, il valore di lease viene aggiornato automaticamente, ritardano l'operazione di eliminazione.

6.1.7 ServiceBrokerAgent

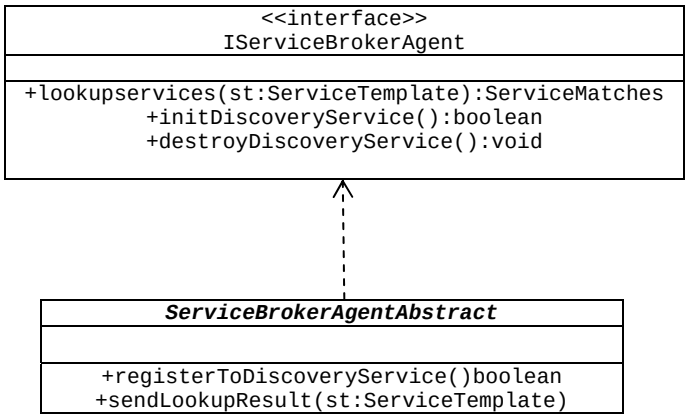


Figura 6.6 ServiceBrokerAgent

Il *ServiceBrokerAgent* rappresenta il mediatore per accedere in ricerca a servizi di discovery diversi da quello di default inserito nel sistema, rappresentato da *DiscoveryManager*. Per ogni specifico standard di discovery deve essere creato un opportuno *ServiceBrokerAgent* che ne tratti la mediazione. Il *ServiceBrokerAgent* è realizzato come un oggetto attivo, al quale è associato un identificativo univoco. Una volta avviato il *ServiceBrokerAgent* come prima cosa questo si registra al *DiscoveryManager*, così che quest'ultimo abbia la possibilità di contattarlo al momento del bisogno utilizzando l'identificativo. Per facilitare la creazione di un mediatore, si

utilizza la classe astratta *ServiceBrokerAgentAbstract* che integra al suo interno il sistema di registrazione e comunicazione con il *DiscoveryManager*, per l'invio dei risultati di ricerca. Per creare il mediatore si dovrà solo realizzare il metodo per l'inizializzazione e il metodo per la ricerca all'interno dello standard di discovery, figura 6.6.

Per esempio nel caso del UPnP, il mediatore una volta ricevuto il *ServiceTemplate*, lo deve convertire in una descrizione in formato compatibile con UPnP e poi contattare il sistema di lookup di UPnP, fornendogli la descrizione, una volta ottenuti i risulta dovrà effettuare un'ulteriore conversione inversa per il formato interno. Risulta quindi necessario utilizzare un parser XML per l'analisi della risposta fornita dallo standard UPnP e il base agli identificativi dei campi contenuti in tale documento XML è possibile ricavare a quali attributi si riferiscono, inoltre è necessario realizzare un sistema di comunicazione basato sul protocollo http, utilizzato in UPnP.

6.2 Sistema di distribuzione video

L'accesso da parte del terminale mobile avviane tramite *MobileClientListener*, un oggetto attivo creato per i Default Place dotati di Access Point e parte integrante dell'*Environment*. Lo scambio di messaggi fa ricorso al protocollo UDP, per avere un maggiore controllo sui messaggi spediti. Per gestire l'accesso concorrente di più utenti, all'arrivo di un messaggio da parte di un terminale mobile, viene generato l'oggetto attivo *PacketManager* incaricato di analizzare e gestire i pacchetti del *ClientMobile*, figura 6.7.

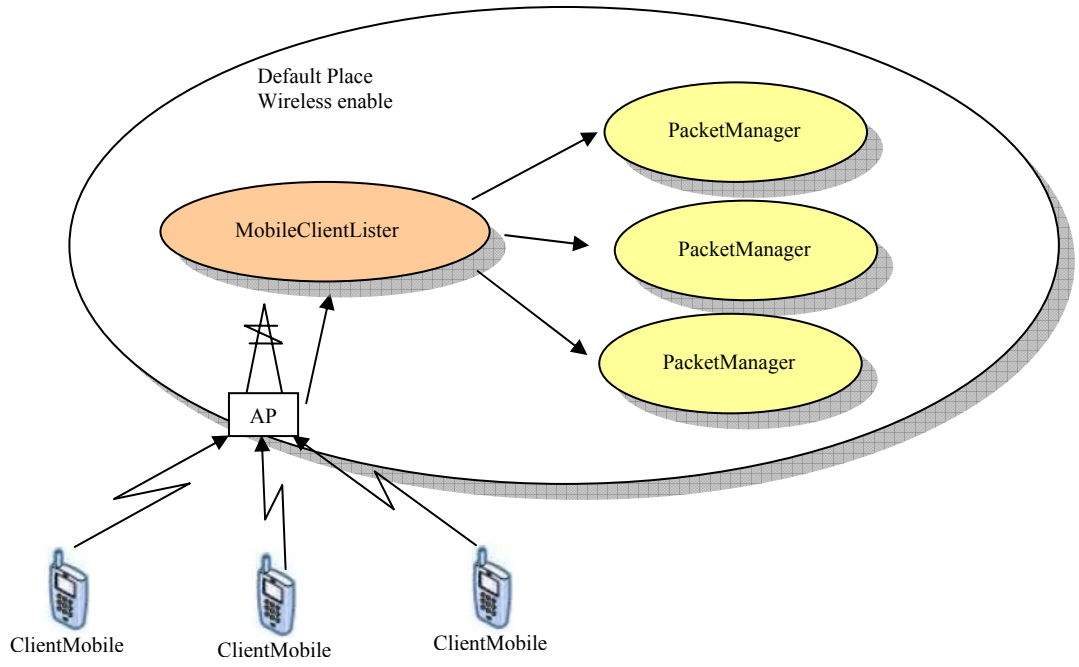


Figura 6.7: MobileClientListener e PacketManager

6.2.1 Il terminale mobile e il ClientMobile

L'elemento *ClientMobile* è un oggetto attivo che coordina tutte le attività sul dispositivo mobile e che interagisce con il *ProxyManagerAgent* per la gestione dell'*handoff* e dell'*offload*. All'interno del terminale mobile sono presenti, oltre al *ClientMobile*, gli oggetti *MobileClientVideoManager* e *MobileClientVideoProtocolUnit* legati allo *streaming* video, in particolare *MobileClientVideoManager* inizializza e gestisce la sessione RTP, ricevendo i dati di controllo della sessione inviati dal *ProxyWireless*, come il *FrameRate* effettivo e il *BitRate*, mentre *MobileClientVideoProtocolUnit* si occupa della parte di comunicazione vera e propria con il *ProxyWireless*, gestendo il flusso video e i comandi di controllo del flusso. Il flusso in arrivo viene poi elaborato dall'oggetto attivo *ReceiveStreamReader* il cui scopo è leggere i *frame* che compongono il contenuto multimediale ed inserirli in *QueableCircolarBuffer*, un oggetto che memorizza i *frame* in una zona di memoria realizzando un buffer circolare. Il *MobileClientVideoProtocolUnit* interagisce con il *QueableCircolarBuffer*, e a seconda dello stato di riempimento del buffer, regola la

velocità dei dati in arrivo tramite comandi di flusso inviati al *ProxyWireless*, più precisamente, quando il buffer è pieno la velocità dei dati deve essere diminuita, viceversa quando il buffer è quasi vuoto aumentata, cercando in questo modo di mantenere costante lo stato di riempimento e nel caso in cui si verifichi un handoff, riavvolgere il flusso video per recuperare i *frame* che vengono persi durante il periodo in cui si ha la riconnessione al nuovo AP. Il *QueableCircolarBuffer* deve essere poi convertito nuovamente in un flusso video per poter essere visualizzato, questa operazione è eseguita dall'oggetto attivo *BufferToDataSource*. Infine il *ClientFrame* rappresenta la *GUI* con la quale l'utente interagisce per il controllo del filmato e dove viene visualizzato il contenuto multimediale. La struttura di *ClientMobile* è presentata in figura 6.8:

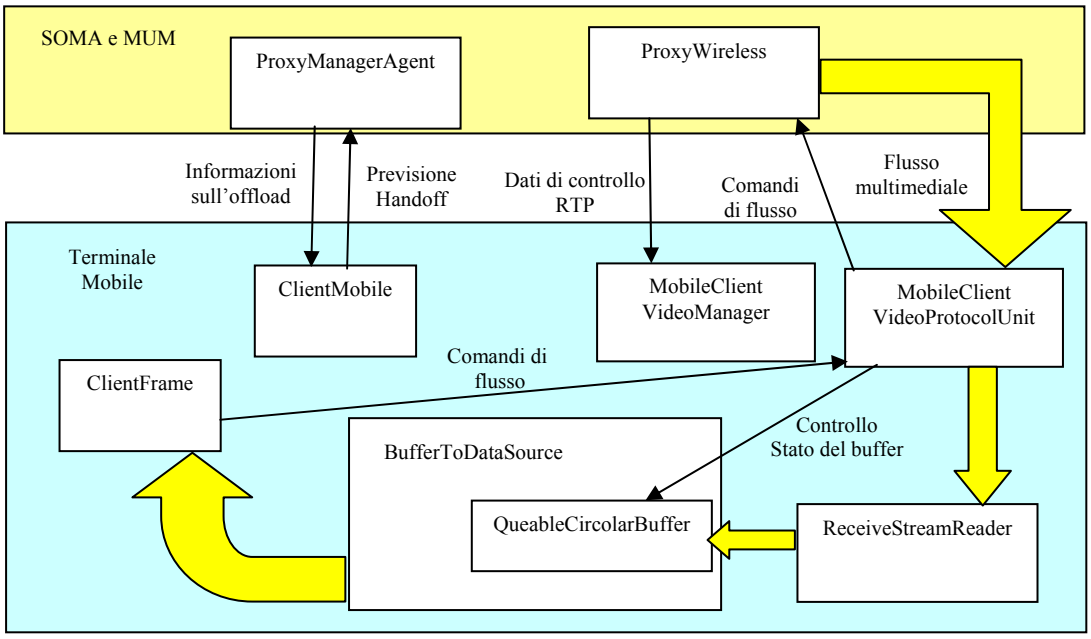


Figura 6.8: Struttura del ClientMobile

6.2.2 ProxyManagerAgent

È l'elemento che inizializza la sessione video per il *ClientMobile* e gestisce tutte le fasi di handoff e *offload*, di conseguenza per gestire i movimenti fra i vari *DefaultPlace* è realizzato come un agente mobile. Interagisce con il database dei profili utente di MUM per ricavare le caratteristiche del terminale utente e le preferenze in caso di *offload*. Queste informazioni vengono passate ad un oggetto attivo interno a

ProxyManagerAgent il cui scopo è interagire con il *DiscoveryManager* per ottenere la destinazione di *offload*.

6.2.2 ProxyWireless

Questo elemento si occupa di inviare il contenuto multimediale al terminale mobile, svolgendo solo una funzione di routing del flusso verso il *ClientMobile*. Quando si ha un handoff, questo elemento viene duplicato nella destinazione dell'utente creando un altro flusso utilizzabile dal *ClientMobile*. In caso di *offload*, invece, non viene utilizzato in quando in questo caso non è necessario avere mobilità del flusso multimediale.
Gli elementi principali che costituiscono il ProxyWireless sono:

- *ProxyInProtocolUnit*: gestisce i comandi per avviare o bloccare il flusso video provenienti dal *ClientMobile*;
- *ProxyOutProtocolUnit*: gestisce i comandi per avviare o bloccare il flusso video diretti al *ProxyAdaptor*;
- *ServerVideoAgent*: invia il flusso video verso il *ClientMobile*;
- *StreamReciever*: riceve il flusso dal *ProxyAdaptor*.

La struttura è rappresentata in figura 6.9:

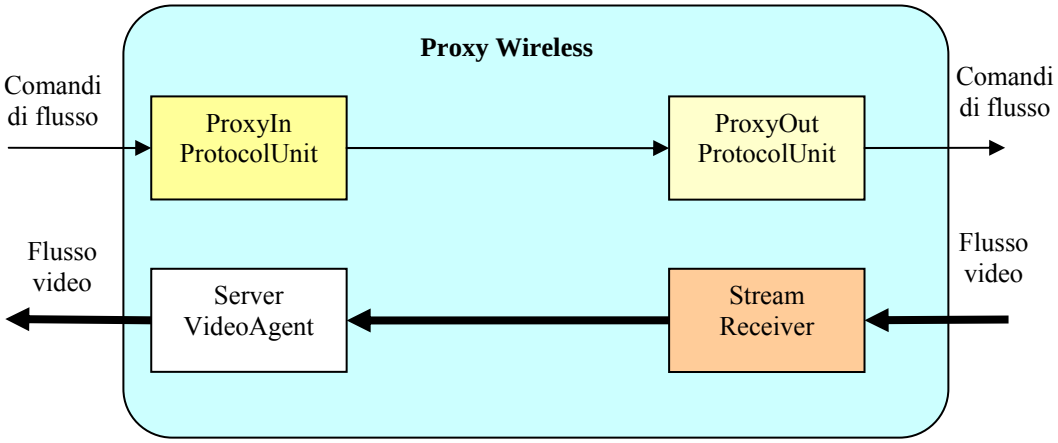


Figura 6.9: Struttura di un ProxyWireless.

6.2.3 ProxyAdaptor

Questo elemento rappresenta un componente intermedio della distribuzione del contenuto multimediale, la cui posizione rimane immutata durante lo *streaming*. La struttura interna è molto simile a quella del *ProxyWireless*, ma la funzione principale è quella di adattare i contenuti multimediali per venire incontro alle esigenze dei dispositivi che lo utilizzano. Gli adattatori necessari a questo compito vengono scaricati al momento dell'inizializzazione o nella fase di riadattamento, sfruttando la distribuzione del codice di MUM. Affinché possa gestire l'*handoff* e l'*offload* del terminale è necessario che sia in grado di duplicare il flusso e di modificare gli adattatori applicati al contenuto multimediale che riceve dal server. Durante la fase di *handoff* parte dei *frame* spediti verso il *ClientMobile* vengono persi, di conseguenza è necessario riuscire a riportare lo stato del flusso indietro in modo da poter riprendere i *frame* persi. Per realizzare questa funzione all'interno del *ProxyAdaptor* è stato inserito all'interno dell'elemento che gestisce il flusso in ingresso, un buffer circolare che abilità la funzione di riavvolgimento del flusso figura 6.10.

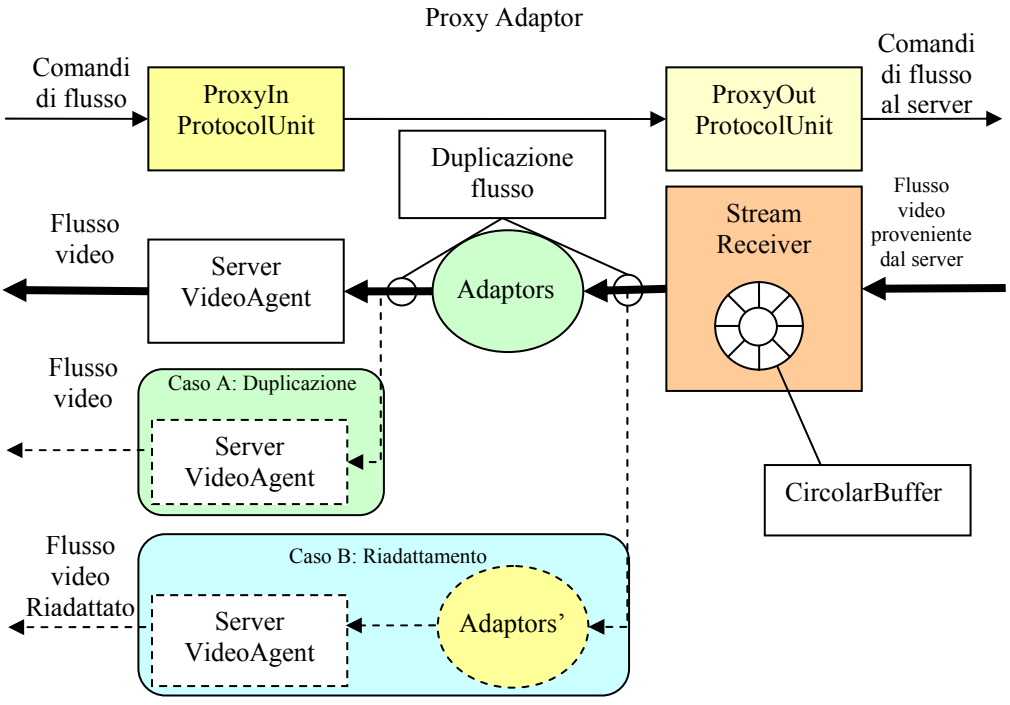


Figura 6.10 Struttura del ProxyAdaptor

6.2.4 Inizializzazione del sistema di distribuzione video

Il pacchetto relativo alla richiesta video è evidenziato in figura 6.11. La prima parte del pacchetto contiene sempre il tipo di pacchetto inviato, in questo caso *VIDEO_REQUEST*, nella seconda parte il corpo del messaggio costituito da:

- *user*: è il nome utente contenuto nel database dei profili utenti di MUM, dove sono contenute tutte le informazioni sull'utente e i dati relativi a tutti i dispositivi mobili posseduti;
- *device*: rappresenta l'identificativo del device attualmente utilizzato dall'utente, come ad esempio il modello di cellulare, è contenuto all'interno del profilo dell'utente, quest'ultimo contenuto a sua volta nel *Repository* dei profili di MUM e permette di ricavare la descrizione del dispositivo in formato CC/PP, ottenendo quindi risoluzione, profondità di colore e dimensione dello schermo, in modo che il sistema possa ricavare gli adattatori necessari al quel particolare terminale;
- *presentation*: è il nome della presentazione multimediale richiesta dall'utente;
- *user data port*: è una porta aperta sul terminale mobile per la ricezione di messaggi di aggiornamento per il *ProxyManagerAgent* e per messaggi di gestione dell'*offload* su un terminale fisso.

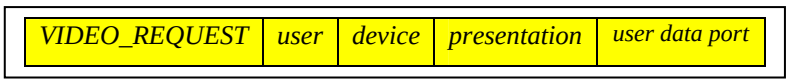


Figura 6.11: Pacchetto di richiesta video.

A seguito della richiesta di un nuovo video, PacketManager verifica l'identità dell'utente e il dispositivo utilizzato, nel caso in cui la verifica abbia successo invia al *ClientMobile* un messaggio di conferma che risponde a sua volta dando origine ad un *handshake* a tre fasi, in caso di errore viene inviato un *ack* negativo nel quale è indicato il motivo del fallimento, figura 6.12.

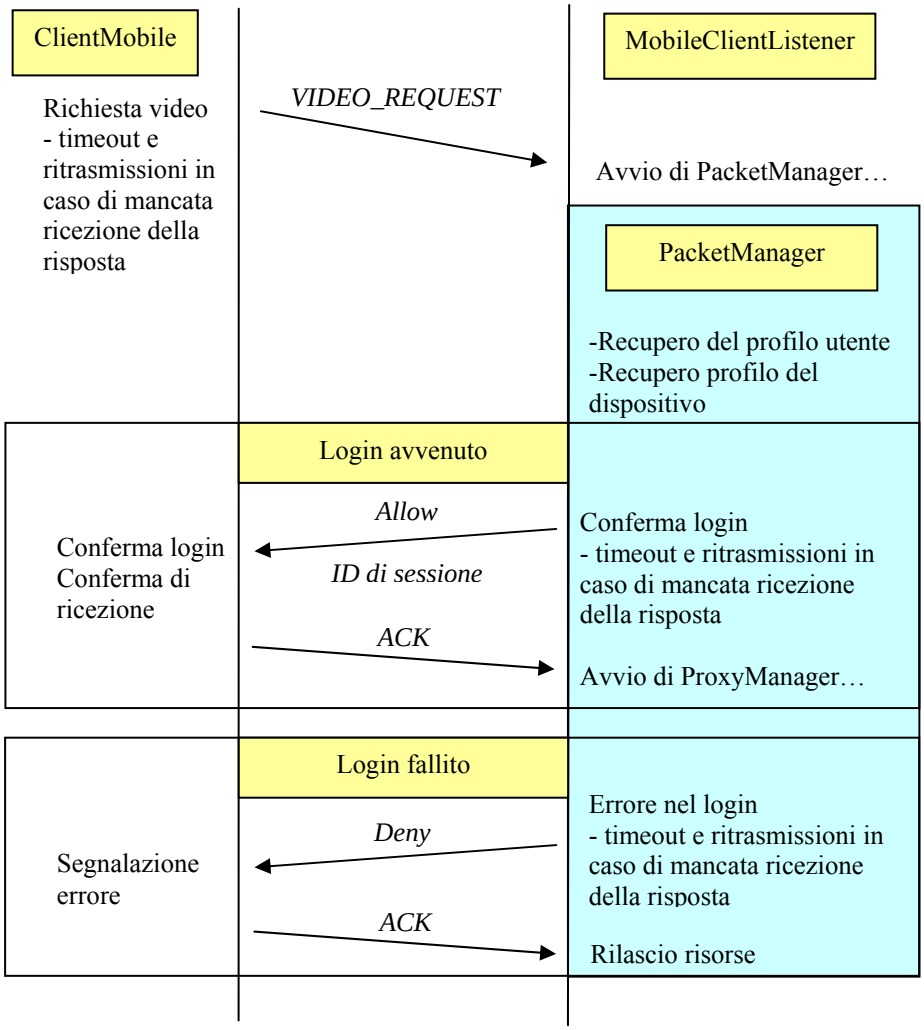


Figura 6.12: Fase iniziale della richiesta video.

A *login* avvenuto, *ClientMobile* si mette in attesa dell'inizializzazione del sistema, mentre *PacketManager* genera un identificativo numerico che rappresenta la sessione e crea l'agente SOMA *ProxyManagerAgent* a cui viene passato come argomento l'identificativo, i dati del profilo dell'utente e i dati di collegamento a *ClientMobile* sul terminale mobile. Quando l'inizializzazione è conclusa il *ProxyManagerAgent* invia i dati di collegamento al *ProxyWireless* e i dati di sessione al *ClientMobile* che può iniziare lo *streaming*. I dati di sessione sono rappresentati dall'identificativo dell'agente *ProxyManagerAgent* e dalla porta sulla quale ascolta i messaggi provenienti dal *ClientMobile* entrambi necessari nelle fasi di *handoff*.

6.2.5 Progetto per la gestione dell'handoff e offload

Quando il *ClientMobile* riceve dal sistema di previsione i dati del prossimo AP al quale prevede di collegarsi, viene creato un pacchetto di previsione, con indicato il valore della previsione, la struttura del pacchetto è presentata in figura 6.13.

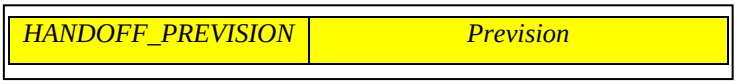


Figura 6.13: Pacchetto di handoff.

Nel momento in cui inizia la fase di handoff, il terminale rimane disconnesso per un determinato periodo di tempo che dipende da quando impiega la scheda wireless a ricollegarsi al nuovo AP.

Arrivato nella nuova località, il *ClientMobile* in base ad una lista ottenuta in precedenza dalla rete (in questa sede non verrà analizzato il metodo di recupero) è in grado di conoscere gli *end-point* dell'infrastruttura associati ai diversi AP, a questo punto ottenuto l'indirizzo dell'infrastruttura, invia un pacchetto di richiesta per il ripristino video, anche qui, come nel caso dell'inizializzazione, interviene il *MobileClientListener* che avvia il *PacketManager* per la gestione del pacchetto. Il pacchetto relativo alla richiesta di ripristino è presentato in figura 6.14. In questo pacchetto, il valore principale è costituito da *agent_session_ID* che rappresenta la sessione video attuale e *oldPMA* che è l'identificativo del *ProxyManagerAgent* che gestiva la sessione video sul nodo precedente.

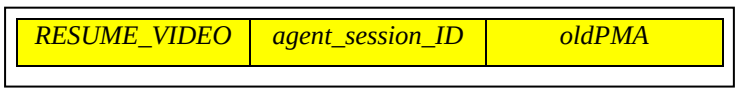


Figura 6.14: Pacchetto di richiesta video.

A questo punto il *ClientMobile* si mette in attesa dei dati di connessione al *ProxyWireless*, figura 6.15.

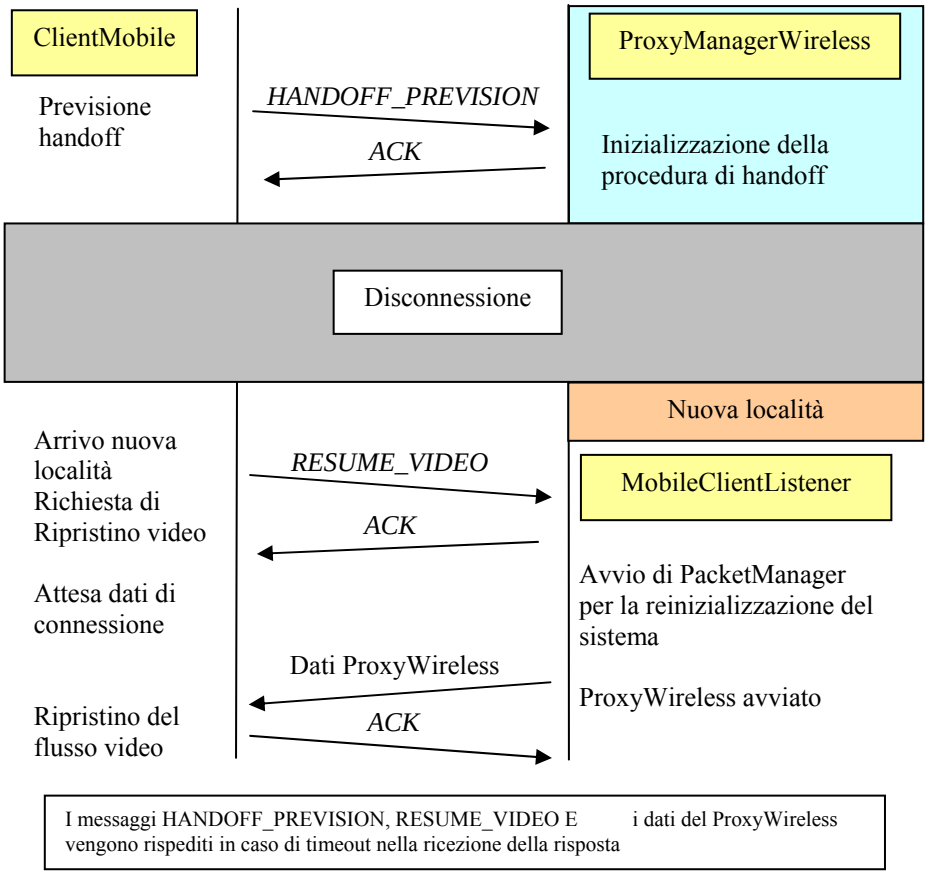


Figura 6.15: Scambio di messaggi Client-Proxy in caso di handoff.

In caso di *offload*, il *ClientMobile* rimane in ascolto su un porta fissata al momento dell'inizializzazione, indicata in figura 6.11 come *user data port*. L'infrastruttura in maniera del tutto trasparente rispetto all'utente ricerca delle postazioni fisse per effettuare il cambio di terminale. Nel caso in cui trovi un terminale o un terminale presente si aggiunga o si liberi, l'infrastruttura invia un pacchetto al *ClientMobile*, per informarlo della possibilità di cambiare terminale, in caso di conferma si ha lo spostamento della sessione, viceversa in caso di rifiuto vengono liberate le risorse sul terminale, figura 6.16.

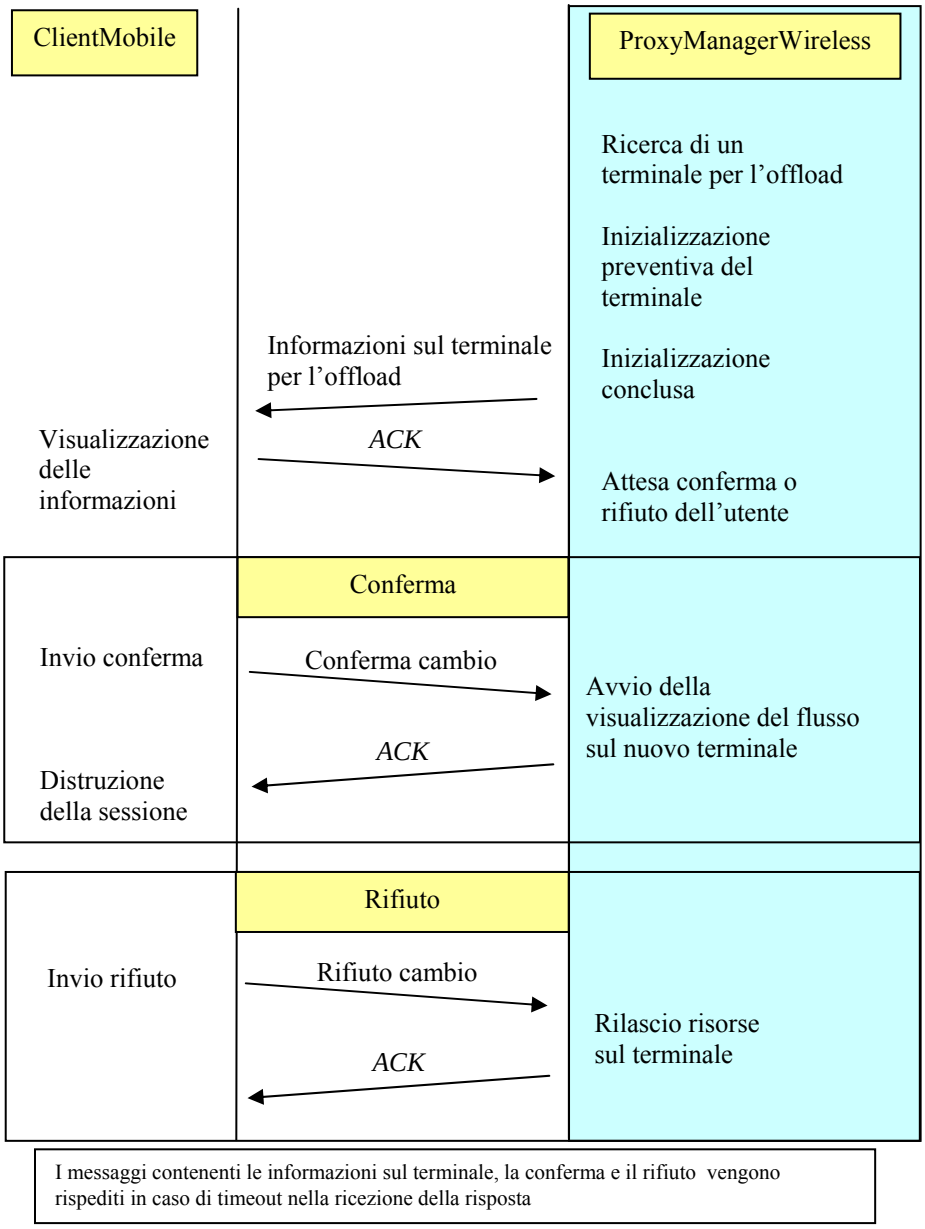


Figura 6.16: Scambio di messaggi Client-Proxy in caso di offload.

6.2.5.1 Gestione dell'handoff da parte dell'infrastruttura

Il *ProxyAgentManager* alla ricezione del pacchetto, fa richiesta al *DecisionMakerWireless* per la creazione di un piano di handoff, che porterà all'instanziazione di un nuovo *ProxyWireless* nella destinazione prevista dal *ClientMobile*. Il piano è contenuto all'interno dell'oggetto *WirelessProxyPlan* che

contiene, inoltre, i dati della sessione corrente rappresentati dall'oggetto *session_infos*, tali dati sono:

- *identificativo della sessione corrente*: il valore univoco assegnato alla sessione video corrente;
- *user profile*: il profilo dell'utente attuale, che contiene tutte le sue preferenze che verranno utilizzate per l'*offload*;
- *device name*: il nome del dispositivo attualmente utilizzato dall'utente, utilizzato per ricavare le caratteristiche funzionali del dispositivo stesso;
- *presentation metadata*: sono i dati relativi alle presentazione video e che sono necessari per ricavare le informazioni sulle risorse impegnate sul nodo dal flusso video;
- *user data port*: porta di servizio aperta sul *ClientMobile*, utilizzata per ricevere le informazioni di aggiornamento per il *ProxyWirelessManager* e le informazioni per l'*offload*;
- *indirizzo del ProxyAdaptor*: è il proxy che si trova immediatamente sopra il *ProxyWireless* e a cui si deve collegare per ricevere il flusso video.

In seguito, richiede la creazione e l'esecuzione dell'agente *PlanVisitorAgent* di MUM al quale viene passato il *WirelessProxyPlan*. Quando il *PlanVisitorAgent* termina l'instanziazione sul nodo destinazione del *ProxyWireless* registra i dati di sessione nel *multimediaServicesManager* di MUM, indicando i dati di connessione al proxy appena creato, in questo modo il nodo di destinazione ha a disposizione tutti i valori necessari per poter ricostruire una nuova sessione video.

Quando il *ClientMobile* arriva nella nuova località, viene creato un *PacketManager* per la gestione del pacchetto. Il *PacketManager* come prima cosa controlla se esiste nel *multimediaServicesManager* una sessione video corrispondente all'identificativo ricevuto, in caso positivo ricava i dati relativi al *ProxyWireless* per il collegamento video e li invia immediatamente al *ClientMobile* tramite il pacchetto *RESUME_REPLY*, in figura 6.17 è presentata la struttura del pacchetto dove sono contenuti l'indirizzo e la porta del *ProxyWireless* relativi alla sessione RTP. Successivamente il *PacketManager* ricrea un nuovo agente *ProxyManagerAgent* sul nodo corrente passandogli tutti i dati di sessione.

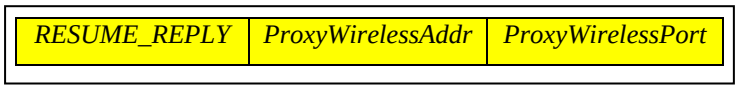


Figura 6.17: Pacchetto di risposta ad una richiesta video.

Il *ProxyManagerAgent* appena creato invia un pacchetto di *UPDATEPMA*, per aggiornare i dati che lo riguardano presenti sul *ClientMobile*, che sono rappresentati dall'identificativo del *ProxyManagerAgent* e la porta aperta per l'ascolto dei dati in caso di *handoff*, figura 6.18.

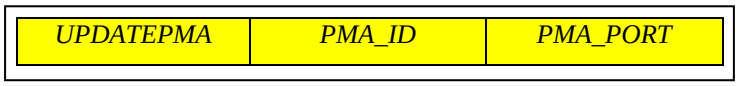


Figura 6.18: Pacchetto di aggiornamento dei dati del ProxyManagerAgent.

A questo punto la procedura di *handoff* può ritenersi conclusa, ma per poter gestire la possibilità che il *ClientMobile* ritorni nella precedente posizione innescando una nuova procedura di *handoff*, caso che si verifica quando ci si trova al confine fra due AP con segnale simile, la sessione nella locazione precedente viene eliminata allo scadere di un tempo fissato a priori, in questo modo per un determinato periodo il *ClientMobile* troverà nelle due locazioni sempre un proxy disponibile.

Nel caso in cui non sia presente nessuna voce registrata nel *multimediaServicesManager* possiamo avere due sottocasi: il *PlanVisitorAgent* è arrivato sul nodo corretto, ma non è riuscito ad reinizializzare la sessione prima dell'arrivo del *ClientMobile* e non ha registrato nessun dato, oppure il *PlanVisitorAgent* è in una locazione sbagliata. In entrambi i casi il *PacketManager* invia al *ClientMobile* un messaggio che lo notifica del ritardo di inizializzazione, mentre in base al campo *oldPMA* ricavato dalla richiesta (figura 6.14), contatta il *ProxyManagerAgent* per informarlo della posizione effettiva del *ClientMobile* e i suoi dati di collegamento. Il *ProxyManagerAgent* conoscendo la previsione è in grado di sapere se si tratta di un ritardo nell'inizializzazione o se invece la previsione è errata. Nel caso di inizializzazione ritardata aspetta la conferma da parte del *PlanVisitorAgent* del termine dell'inizializzazione e poi migra sul nodo di destinazione, in questo modo sul nodo di destinazione sono presenti tutti gli elementi necessari allo *streaming* e si può quindi inviare al *ClientMobile* le informazioni di collegamento al *ProxyWireless* e le

informazioni di aggiornamento sul *ProxyManagerAgent* come nel caso precedente, figura 6.19. Nel caso di inizializzazione errata, sulla destinazione non è presente nessun elemento di conseguenza il *ProxyManagerAgent* migra sulla destinazione portando quindi con se tutti i dati necessari e fa ripartire il *PlanVisitorAgent* per la locazione attuale, quando l'inizializzazione sarà terminata procederà all'invio delle informazioni al *ClientMobile* come summenzionato.

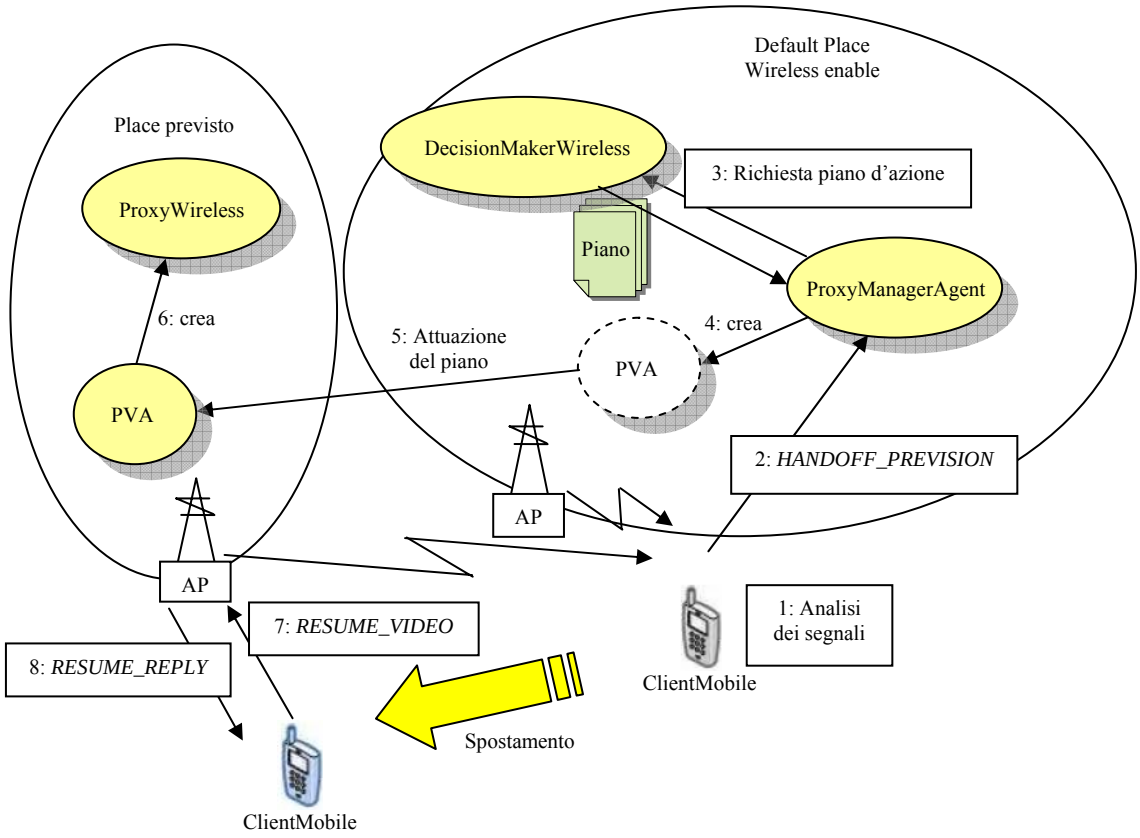


Figura 6.19: Gestione handoff sull'infrastruttura.

6.2.5.2 Gestione dell'offload da parte dell'infrastruttura

Ad ogni inizializzazione il *ProxyManagerAgent* instanzia un oggetto attivo denominato *OffloadFinder* il cui unico scopo è cercare nel sistema di discovery un terminale fisso per effettuare l'handoff. In particolare dopo aver ricavato la lista di tutti i possibili terminali disponibili per l'*offload*, la filtra in base alle caratteristiche specificate nelle preferenze del profilo dell'utente cioè in base alla dimensione in pollici e alla risoluzione. Tra tutti i possibili terminali le cui caratteristiche rientrano nella

categoria evidenziata sopra, sceglie quello dalle caratteristiche migliori e da avvio alla procedura di inizializzazione.

Come nel caso dell'*handoff* si utilizza il *DecisionMakerWireless*, richiedendogli un piano di *offload*. Il piano consiste nell'instanziazione sul nodo di destinazione di un *ClientVideo* che rappresenta un entità in grado di visualizzare il contenuto multimediale che viene erogato direttamente dal *ProxyAdaption*. Il flusso video tipicamente necessiterà di un riadattamento, in modo che la visualizzazione sul terminale fisso possa essere sfruttata nel migliore dei modi. I dati relativi al riadattamento sono contenuti nella descrizione del servizio e sono inseriti insieme al piano nell'oggetto *OffloadClientWirelessPlan*, che rappresenta appunto il piano utilizzato dal *PlanVisitorAgent* per l'inizializzazione. Il *PlanVisitorAgent* esegue una pre-inizializzazione che consiste nell'instanziare gli elementi coinvolti nella ricezione dei dati, richiedere il riadattamento del video e ricezione del flusso video, senza però dare avvio alla visualizzazione effettiva. In seguito invia al *ProxyManagerAgent* un messaggio di conferma utilizzando l'infrastruttura per i messaggi di SOMA e si mette in attesa per la conferma a procedere alla visualizzazione da parte di *ProxyManagerAgent*. Il *ProxyManagerAgent* ricevuto il messaggio prepara per il *ClientMobile* un pacchetto informativo, figura 6.20.

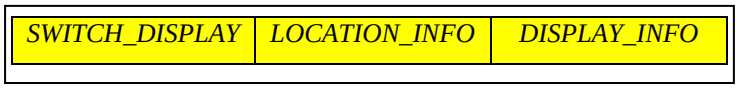


Figura 6.20: Pacchetto informativo sul offload.

Il pacchetto contiene la locazione effettiva dello schermo all'interno dell'edificio, in modo che l'utente che utilizza il terminale mobile possa spostarsi fisicamente sul terminale e altre informazioni sullo schermo come la risoluzione e la dimensione in pollici. Dopo l'invio del pacchetto il *ProxyManagerWireless* aspetta la decisione dell'utente ad effettuare l'*offload*. I dati di *SWITCH_DISPLAY* verranno visualizzati sullo schermo del dispositivo mobile e il *ClientMobile* a seconda della risposta del cliente invierà un messaggio *SWITCH_OK* nel caso l'utente decida di spostare la sessione o *SWITCH_CANCEL* nel caso contrario. Solo nel primo caso il *ProxyManagerAgent* distrugge il *ProxyWireless* e invia al *PlanVisitorAgent* in attesa un messaggio di avvio della visualizzazione, concludendo la procedura di *offload*.

6.3 Conclusioni

In questo capitolo è stato presentato dettagliatamente il progetto sviluppato nel lavoro di testi. Riprendendo l'architettura proposta in fase di analisi, considerando la parte relativa al sistema di discovery composta dai due layer *Services* e *Facilities* e la parte relativa alla distribuzione dei video, trattando la gestione dell'handoff e dell'*offload*.

Nel prossimo capitolo verrà presentata l'implementazione di quanto esposto nel capitolo di progettazione.

CAPITOLO 7

IMPLEMENTAZIONE E TEST DEL SISTEMA

L'ultima fase del processo di sviluppo del sistema è l'implementazione, in cui l'architettura e le soluzioni individuate nella fase di progettazione vengono realizzate attraverso l'utilizzo degli specifici strumenti che la piattaforma di sviluppo scelta mette a disposizione. In realtà, dopo l'implementazione, è di fondamentale importanza sottoporre il sistema creato a una fase di test per verificare che tutte le funzionalità progettate si comportino nel modo previsto e per valutare l'efficienza delle scelte effettuate in fase di progetto.

La prima parte del capitolo sarà dedicata alla presentazione di alcune caratteristiche implementative del sistema, soffermandosi su alcune parti ritenute maggiormente interessanti; nella seconda, invece, si focalizzerà l'attenzione sui risultati ottenuti dalla fase di test del sistema cercando di capire meglio i limiti della soluzione proposta e trarre dalla loro analisi considerazioni e utili spunti per lo sviluppo di eventuali miglioramenti del sistema.

7.1 Strumenti per l'implementazione

Come anticipatamente segnalato in fase di progettazione, la piattaforma utilizzata per la realizzazione del sistema progettato è quella JAVA, linguaggio di programmazione in cui è stato sviluppato SOMA e quindi conseguentemente utilizzato per l'implementazione di MUM. Per quanto riguarda l'implementazione della parte software sul del terminale mobile, è stato scelto anche qui Java per omogeneità con la restante parte e soprattutto perché in questo modo è possibile in futuro una facile conversione verso Java MicroEdition che rappresenta una versione di Java studiata appositamente per terminali mobili.

Nei successivi paragrafi, verranno illustrate alcune delle scelte implementative svolte durante la realizzazione del prototipo, con riferimento in particolare alle funzioni di ricerca nel sistema di discovery e le procedure di *handoff* e *offload*.

7.2 Implementazione del sistema di discovery

L'obbiettivo di questa parte di capitolo non è quello di illustrare la completa implementazione del sistema di discovery, per la quale si rimanda al codice realizzato, ma ha lo scopo di evidenziare solo i punti cruciali del lavoro svolto mettendo in risalto le peculiarità implementative del prototipo realizzato.

7.2.1 Sistema di discovery

Il servizio di discovery, è stato concepito come un'estensione della piattaforma SOMA e come è stato esposto nei capitoli precedenti, risiede nei Default Place, in quanto questi identificano una certa località ed è quindi naturale identificare i servizi della località in questo Place. L'avvio del servizio avviene al momento della creazione del Default Place, in particolare quando viene creata la classe *Environment*, la quale permette poi di accedere a questo servizio. Come evidenziato in fase di analisi, i servizi vengono memorizzati in un database locale che funge da cache per la ricerca. Questo database è realizzato con la classe *HashTable* di Java, in quanto permette di inserire elementi identificandoli con una chiave alla quale è associata il valore che si intende memorizzare. Al fine di garantire la mutua esclusione nell'accesso al database ogni metodo che gestisce l'inserimento, la cancellazione e la ricerca è stato dichiarato *synchronized*.

Per quanto riguarda l'interazione con i *ServiceBrokerAgent*, all'interno del *DiscoveryManager* sono presenti due liste rappresentate dalla classe *Vector* di Java. La prima lista è dominata *brokerAgents* e contiene tutti gli identificativi ai *ServiceBrokerAgent* della corrente località. Questi identificativi permettono di risalire al Place dove è fisicamente allocato il *ServiceBrokerAgent*. Perché ciò sia possibile è

necessario che i *ServiceBrokerAgent* si registrino al *DiscoveryManager*. La registrazione avviene tramite la classe comando di SOMA *RegisterServiceBrokerAgent*, nella quale è specificato il Place e l'identificativo del Thread *ServiceBrokerAgent*. Nel momento in cui il *DiscoveryManager* deve effettuare una ricerca utilizza la classe comando *LookupCommand* per richiamare i metodi di ricerca sui *ServiceBrokerAgent*, che a lavoro completato invieranno la risposta al *DiscoveryManager* utilizzando la classe comando *RegisterResultCommand*. *RegisterServiceBrokerAgent*, *LookupCommand* e *RegisterResultCommand* ereditano dalla classe *Command* di SOMA che utilizza il paradigma remote evaluation (REV) per poter eseguire del codice in remoto su un determinato Place (si veda paragrafo 4.1.1.6.5), in questo modo sia il *DiscoveryManager* che il *ServiceBrokerAgent* possono interagire fra loro come se agissero in locale, figura 7.1.

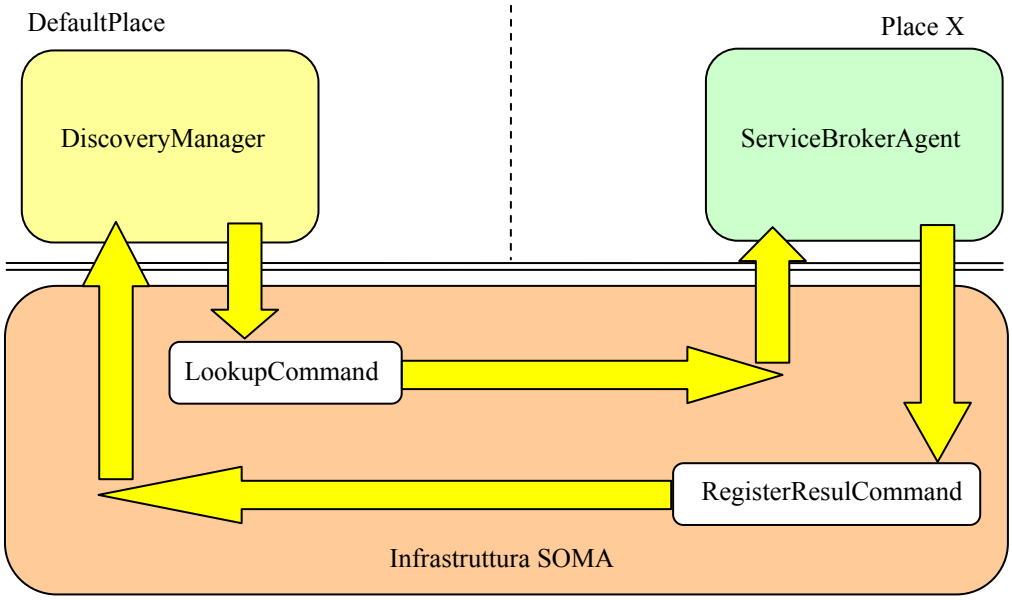


Figura 7.1: Comunicazione fra DiscoveryManager e ServiceBrokerAgent

L'utilizzo dei comandi sopra summenzionati è dettato anche da una questione di omogeneità con l'infrastruttura SOMA, alternativamente si sarebbero potute impiegare delle socket in ascolto sui *ServiceBrokerAgent*, ma questa possibilità è stata scartata al fine di ridurre le porte occupate dal sistema.

Il comando *LookupCommand* incapsula al suo interno la richiesta di ricerca, in quando i *ServiceItem*, come evidenziato nella fase di progetto, sono serializzabili e quindi possono essere inviati sulla rete come flusso di byte, giunto a destinazione

LookupCommand passa al *ServiceBrokerAgent* il *ServiceTemplate* che identifica la richiesta. Analogamente il *RegisterResultCommand*, incapsula al suo interno il *ServiceMatch* che rappresenta la risposta, i servizi in esso contenuti verranno aggiunti alla cache locale. La seconda lista presente in *DiscoveryManager* è denominata *brokerResults*, il suo scopo è memorizzare i risultati inviati dai *ServiceBrokerAgent*, quando il numero dei risultati corrisponderanno al numero di *ServiceBrokerAgent*, allora la ricerca sarà conclusa.

7.2.1.1 ServiceBrokerAgentUPnP

Al fine di realizzare un accesso al servizio UPnP sono state utilizzate le librerie introdotte nel paragrafo 4.4.. Per creare un *ServiceBrokerAgent* è necessario ereditare dalla classe astratta *ServiceBrokerAgentAbstract*, la quale fornisce in automatico tutta la parte relativa all'inizializzazione e alla comunicazione con il *DiscoveryManager*, rimane quindi solo da implementare il metodo di ricerca. Nel caso di UPnP, una volta inizializzate le librerie tramite il metodo *UPnPControlPoint.start()* e registrato il *ServiceBrokerAgentUPnP* come *listener* delle risposte, è possibile effettuare le ricerche nello standard, in figura 7.2 è presentato il metodo di ricerca:

```
public ServiceMatches lookupservices(ServiceTemplate st) {

    ServiceMatches sm = new ServiceMatches();
    this.sync = new Synchronizer(this.out);
    //ricerca dei dispositivi nella rete
    this.UPnPControlPoint.search(ST.ROOT_DEVICE);
    //attesta del risultato
    this.sync.waitAGo();
    DeviceList rootDevList = this.UPnPControlPoint.getDeviceList();
    int nDevs = rootDevList.size();
    for (int n=0; n<nDevs; n++) {
        //singolo dispositivo
        Device dev = rootDevList.getDevice(n);
        //converte il dispositivo, se non corrisponde alle richieste il valore
        è null
        ServiceItem si = convertDevice(dev,st);
        if(si!=null){
            sm.addServiceItem(si);
        }
    }
    this.sync = null;
    return sm;
}
```

Figura 7.2: Ricerca nello standard UPnP

Come si vede dalla figura, viene inviata una richiesta di ricerca di servizi e poi si mette in attesa dei risultati, una volta ottenuti vengono convertiti in *ServiceItem* ed inseriti in una struttura *ServiceMatch*, infine in automatico sfruttando i metodi di *ServiceBrokerAgentAbstract* i risultati saranno inviati al *DiscoveryManager*.

7.2.2 Ricerca dei dati nella cache

In figura 7.3 è presentato il metodo che esegue la ricerca dei servizi nel database locale, che rappresenta la cache per gli standard di discovery:

```
//recordService è la hashtable che contiene la cache dei servizi
Enumeration elements = this.recordService.elements();

while(elements.hasMoreElements()){

    ServiceItem si = (ServiceItem)elements.nextElement();
    // attributeSets è l'insieme degli attributi del servizio
    AttributeInterface[] attributesSets = si.attributeSets;
    //variabile dove verranno conteggiati il numero di attributi compatibili
    int numbequal=0;

    for(int i=0;i<attributesSets.length;i++){
        for(int j=0;j<st.attributeSetTemplates.length;j++){

            if(st.attributeSetTemplates[j].equals(attributesSets[i])){
                numbequal++;
            }
        }
    }
    //solo se il numero di attributi compatibili è lo stesso del template
    //la ricerca ha esito positivo
    if(numbequal == st.attributeSetTemplates.length){
        sm.addServiceItem(si);
    }
}
```

Figura 7.3 : Ricerca in cache.

Come evidenziato nel paragrafo 6.1.4, la ricerca confronta gli attributi del *template* con quelli dei servizi, se tutti gli attributi del *template* sono contenuti anche nel servizio allora il servizio fa *match* con il *template*. Questa modalità di ricerca, è stata scelta in quanto ai fini della tesi modalità più complesse non avrebbero portato a significativi miglioramenti, inoltre un tipo di ricerca simile è utilizzata anche in Jini. Come si vede dalla figura per ogni elemento contenuto nella cache, rappresentato da un *ServiceItem*, viene eseguita una comparazione di tutti gli attributi in esso contenuti con gli attributi presenti nel *ServiceTemplate*, indicato con *st*. Nel caso in cui tutti gli attributi del

ServiceItem analizzato in quel momento, siano uguali al *ServiceTemplate*, allora il *ServiceItem* verrà inserito nel *ServiceMatch*, indicato in figura con *sm*.

7.2.3 Attributi dei servizi

Gli attributi derivano dalla classe astratta *AbstractAttribute*, che come evidenziato in fase di analisi, implementa un metodo che esegue un confronto sfruttando la riflessione di Java, figura 7.4.

```
public static boolean equals(AttributeInterface a1, AttributeInterface a2) {
    //stesso riferimento stesso attributo
    if (a1 == a2)
        return true;
     //diversa classe
    if (a1.getClass() != a2.getClass())
        return false;

          //metodo per l'estrazione degli attributi tramite la riflessione di Java
          //esclude i modificatori statici, final e transient
    Field[] fields = fieldInfo(a1);
    try {
    // viene comparato ogni campo
    // di conseguenza se si tratta di un altro attributo per il polimorfismo
    // viene analizzato nei dettagli utilizzando sempre questo metodo
    for (int i = 0; i < fields.length; i++) {

        // f works for other since other is the same type as this
        Field f = fields[i];
        Object ov = f.get(a1);
        Object tv = f.get(a2);

        if (tv == ov)           // stesso riferimento o entrambi nulli
            continue;
        if (tv == null || ov == null)   //caso in cui solo uno è nullo
            return false;

        if (!tv.equals(ov))     //comparazione se si tratta di attributi
            return false;           //viene nuovamente richiamato questo metodo
    }
```

Figura 7.4 : Il metodo equals per il confronto di attributi.

Come si vede dalla figura 7.4 gli attributi coinvolti nel confronto implementano l'interfaccia *AttributeInterface*, nel caso in cui i riferimenti siano gli stessi il confronto termina immediatamente, mentre in caso contrario si utilizza il metodo *fieldInfo()* che ricava dall'attributo passato tutti i campi Java che ne costituiscono la struttura interna, escludendo dalla ricerca i valori *static*, *transient* e *final* in quanto non significativi, inoltre affinché la ricerca vada a buon fine è necessario che i campi oggetto del

confronto siano dichiarati *public*, perché la riflessione di Java non permette l'accesso a campi privati.

7.3 Implementazione del sistema di distribuzione video

L'implementazione del sistema di distribuzione video può essere concettualmente diviso in due parti, una prima parte rivolta all'implementazione del servizio in MUM e una seconda parte riguardante l'implementazione sul cliente mobile. Come nel paragrafo precedente verranno esposte solo le parti ritenute più significative.

7.3.1 Profilo utente mobile

Come ampliamente evidenziato nei capitoli precedenti, al momento dell'accesso da parte del cliente, si ha una fase di *login* dove vengono verificati i dati dell'utente, vengono ricavati i dati sul dispositivo utilizzato e sulle sue preferenze in caso di *offload* attraverso quello che viene chiamato *MobileUserProfile*.

I dati utente contenuti nel *MobileUserProfile* sono gli stessi che vengono ereditati dal profilo standard di MUM, mentre i dati del dispositivo rappresentano le informazioni sulle caratteristiche fisiche del dispositivo mobile. Queste caratteristiche sono memorizzate in una tabella interna, che può contenere tante voci quanti sono i dispositivi utilizzati dall'utente. La tabella è realizzata come una Hashtable dove la chiave di ricerca è costituita da un identificativo di tipo Stringa che rappresenta il nome e il modello del dispositivo mobile, mentre il valore associato alla chiave rappresenta l'identificativo del profilo CC/PP, quest'ultimo valore sarà quello che verrà passato al sistema di adattamento, che interagendo con il database dei profili CC/PP otterrà infine le informazioni sulla dimensione dello schermo e profondità di colore che utilizzerà per adattare il flusso multimediale.

7.3.2 ProxyManagerAgent

Quando il cliente ha effettuato la prima fase di login ed è stato riconosciuto dal sistema, il thread PacketManager che gestisce questa fase, crea un nuovo *ProxyManagerAgent* che darà avvio all'inizializzazione di tutto il percorso video fino al *ClientMobile*. Il *ProxyManagerAgent* eredita dalla classe Agents di SOMA, rendendolo di fatto un agente mobile. Questa caratteristica è necessaria nelle fasi di *handoff*, in particolare nel caso di previsione errata, perché permette al *ProxyManagerAgent* di migrare portando con se i dati delle sessione. I dati sessione, rappresentano tutte le informazioni della sessione corrente e sono registrati dal *ProxyManagerAgent* presso il *multimediaServicesManager* al momento dell'inizializzazione del sistema quando tutti i dati sono noti, in figura 7.5 è presentato il metodo che si occupa di tale registrazione.

```
private void register_session() {

    //creo gli elementi di sessione
    session_infos = new Object[11];
    session_infos[0] = this.agent_session_id; //identificativo di sessione
    session_infos[1] = this.profile; //profilo mobile del cliente
    session_infos[2] = this.devicename; //identificativo del dispositivo
    session_infos[3] = this.userAddress; //indirizzo attuale del cliente
    session_infos[4] = new Integer(this.userPort); //porta attuale del cliente
    session_infos[5] = this.presentationmetadatalist; //elenco delle presentazioni
    session_infos[6] = new Integer(this.userDataPort); //porta dati del cliente
    session_infos[7] = this.getID(); //identificativo del ProxyManagerAgent
    session_infos[8] = this.proxyadaption; //proxy che esegue l'adattamento
    session_infos[9] = this.currentproxyvideo; //proxywireless corrente
    //planvisitoragent che ha creato il proxy wireless
    session_infos[10] = this.pvaCurrentProxy;

    //registrazione dell'identificativo di sessione e dati di sessione
    //presso il multimediaServicesManager

    this.agentSystem.getEnvironment().multimediaServicesManager.
        addSessionClientMobileReference(this.agent_session_id,this.session_infos);

}
```

Figura 7.5: Metodo per la registrazione della sessione.

I dati di sessione vengono memorizzati in un array di *Object*, in quanto quando l'agente *ProxyManagerAgent* viene ricreato a seguito di un *handoff*, potranno essere passati direttamente al costruttore dell'agente la cui l'interfaccia accetta appunto un array di *Object*. Il *multimediaServicesManager* era già presente in MUM, è quindi stato modificato affinché contenga una Hashtable per la registrazione della sessione, utilizzando come chiave di ricerca l'identificativo di sessione *agent_session_id*. Quando avviene l'*handoff* parte delle voci che costituiscono *session_infos* vengono aggiornate e

registrate nel Place di destinazione, in particolare la voce *userAddress* che rappresenta l'indirizzo del cliente cambia per via del cambio di Access Point, *currentproxyvideo* che rappresenta l'indirizzo del *ProxyWireless* è diverso in quanto questo elemento viene ricreato sul Default Place di destinazione e il *pvaCurrentProxy* che rappresenta il *PlanVisitorAgent* creato per l'instanziazione del *ProxyWireless* è creato nel momento della richiesta di *handoff*.

Per quanto riguarda la gestione dell'*offload*, è presente una *inner class* creata al momento dell'instanziazione del *ProxyManagerAgent*. Questa *inner class* non è altro che un thread che periodicamente controlla il *DiscoveryManager* locale, al fine di trovare un terminale fisso, sul quale risieda la piattaforma SOMA e MUM e le cui caratteristiche vengano incontro alla preferenze dell'utente, presenti nel profilo.

7.3.3 ProxyWireless e PlanVisitorAgent

Il *ProxyWireless* è l'elemento che invia dati verso il cliente ed è quindi presente in rapporto uno a uno con il *ClientMobile*. Quando il cliente si muove a seguito di un *handoff* , viene ricreato nel Default Place nel quale il cliente verrà a trovare. Affinché possa essere eliminato nel momento in cui non è più utile per lo *streaming*, è direttamente collegato al *PlanVisitorAgent*, il quale dopo averlo creato instanzia una *MailBox* per la ricezione del messaggio relativo alla eventuale distruzione.

7.3.4 ProxyAdaptor e buffer circolare

Durante il processo di handoff il client ha un periodo nel quale il segnale wireless è del tutto assente. In quel momento tutti i pacchetti rappresentanti il flusso video vengono persi, di conseguenza è presente nel *ProxyAdaptor* un buffer circolare, la cui funzione è quella di portare indietro il flusso dei *frame* persi, evitando che il cliente abbia un salto nella riproduzione del flusso multimediale. Il buffer circolare è rappresentato da un vettore di elementi Buffer, che vengono riempiti dal Thread *ProxyBufferReceiveStreamReader*, questo Thread legge i frame direttamente dal

datasource proveniente dal Server. La bufferizzazione è stata inserita prima della fase di adattamento, in quanto in caso di riadattamento l'intero contenuto del buffer sarebbe stato inutile e quindi scartato. Affinché sia possibile inviare i dati contenuti nel buffer come se si trattasse di un normale *datasource* interviene la classe *ProxyBufferToDataSource*. Questa classe eredita da *PushBufferDataSource* delle librerie JMF e permette di costruire un *datasource* per l'invio di dati in *streaming*. Al suo interno è presente un ulteriore classe, il *LiveStream* che eredita da *PushBufferStream* ed implementa *Runnable*. È *LiveStream* che legge direttamente dal buffer per simulare lo stream multimediale.

Il *ProxyAdaptor* esegue anche l'adattamento e il riadattamento del flusso multimediale. Rispetto a quanto già presente in [Mon04], per eseguire il riadattamento sono state effettuate alcune modifiche. Innanzi tutto è stato necessario creare un apposito metodo che fornito in ingresso il *datasource* originale, restituisca in uscita un *datasource* adattato, figura 7.6.

```
private DataSource applyAdaptor(Vector adaptors, DataSource ds) throws Exception {
    // array che contiene gli adattatori che saranno applicati
    ArrayList dsList = new ArrayList();
    // array che contiene i player per l'adattamento
    ArrayList players = new ArrayList();
    DataSource adaptDS = null;
    if (adaptors != null) {
        adaptors.add(0, h263Adaptor);
        dsList.add(ds);
        for (int i = 0; i < adaptors.size(); i++) {
        Adaptor currT = (Adaptor) adaptors.get(i);
        adaptDS = this.initProcessing((DataSource) dsList.get(i), currT, players);
        dsList.add(adaptDS);
    }
    // dopo il ciclo for currDS contiene il datasource con tutte
    // le trasformazioni applicate
    for (int j = 0; j < players.size(); j++) {
        Player currP = (Player) players.get(j);
        waitForState(currP, Player.Started);
        ((Adaptor) adaptors.get(j)).startTranscoding();
    }
    return adaptDS;
}
```

Figura 7.6: Il metodo per applicare gli adattatori al datasource

Secondariamente è stato necessario creare prima dell'adattamento un *datasource* clonabile tramite il metodo fornito dalle JMF *createClonableDataSource()*, in modo tale che fosse possibile creare un duplicato del flusso multimediale ottenendo lo *streaming* su due Place contemporaneamente, molto importate nella fase di *offload* per non bloccare l'esecuzione del contenuto multimediale. Anche il flusso adattato è stato reso

clonabile, in quanto in questo modo al momento dell'*handoff*, il sistema ha gli adattatori già disponibili e non è necessario un riavvio.

7.3.5 Codice sul terminale mobile

L'implementazione del codice lato cliente è l'unica parte del progetto che non fa ricorso all'infrastruttura SOMA per eseguire. È costituito da diverse classi che svolgono le funzioni di comunicazione con la piattaforma MUM, la gestione dei messaggi con il *ProxyManagerAgent*, il buffering del flusso video e visualizzazione vera e propria del filmato.

Per quanto riguarda le caratteristiche del codice, si prenderà in esame solo il buffer sul cliente. Questo buffer differisce rispetto alla versione del proxy per quanto riguarda la gestione dell'inserimento dei *frame*, infatti quando si verifica un *handoff* il gestore buffer, rilevando un salto nei *frame* inviati dal Proxy, richiede un riavvolgimento, in modo da ripristinare quanto perso. Un ulteriore differenza rispetto al buffer del proxy è nella conversione in *datasource*, in quanto non essendo necessario l'invio, la classe che si occupa della conversione, denominata *BufferToDataSource*, deriva dalla classe astratta *PullBufferDataSource* di JMF, derivando da questa classe si permette al player contenuto nel cliente una gestione del flusso di dati in lettura.

7.4 Test del sistema

Quest'ultima parte del capitolo è dedicata alla presentazione dei risultati sperimentali e a una loro interpretazione e discussione, allo scopo di individuare eventuali inefficienze e carenze del sistema implementato. Essa è dunque di estrema importanza perché rappresenterà il punto di partenza di eventuali sviluppi futuri del sistema.

I test riguarderanno le principali funzioni svolte dal sistema in particolare:

- ricerca nel sistema di discovery e valutazione della funzione di caching;
- inizializzazione del sistema di distribuzione video;

- valutazioni delle tempistiche di *handoff* in caso di previsione corretta ed errata;
- valutazioni delle tempistiche per l'inizializzazione dell'*offload*;
- scalabilità del sistema al variare del numero di utenti.

7.4.1 Sistema di discovery

Al fine di valutare le prestazioni del sistema di discovery, sono stati fatti diversi test con un numero variabile di elementi all'interno della cache e un *ServiceBrokerAgent* per l'accesso ad un sistema di discovery standard. Il *ServiceBrokerAgent* preso come riferimento per la ricerca in servizi standard è *ServiceBrokerAgentUPnP* che si riferisce a UPnP. Questo test, è stato realizzato su macchine dotate di sistema operativo Windows XP, per via della presenza del servizio UPnP. La rete utilizzata in questo test è costituita come segue:

- Un Place di default dove risiede il servizio di discovery, eseguito su un Celeron 1 GHz con 512 mega di RAM;
- Un Place normale che effettua le ricerche, eseguito su un Celeron 1,6 GHz con 256 mega di RAM.

I risultati espressi in millisecondi, sono stati riportati in tabella 1:

	Senza l'utilizzo della cache			**Con utilizzo della cache**		
	Prova 1	**Prova 2**	**Prova 3**	**Prova 1**	**Prova 2**	**Prova 3**
10 servizi	660	481	499	10	<1	<1
100 servizi	4881	982	872	10	<1	<1
150 servizi	7289	1210	1072	20	<1	<1

Tabella 1: Ricerca nel servizio di discovery.

Come si può notare dalla tabella l'utilizzo della cache permette un notevole risparmio di tempo, dovuto principalmente al fatto che la cache è contenuta in memoria centrale e l'accesso avviene direttamente, mentre la ricerca negli standard avviene indirettamente dovendo passare per il *ServiceBrokerAgent* che comporta un ulteriore scambio di messaggi, inoltre la ricerca nella cache non necessita di una conversione degli attributi. Il tempo di conversione degli attributi è dipendente dal tipo di ricerca e dal tipo di standard, nel caso del UPnP consiste nella conversione dai dati XML: per il test sono

stati considerati gli attributi relativi alla descrizione del servizio di visualizzazione. Si può infine notare che la differenza di tempo dalla prova 1 rispetto alle successive è dovuta al primo caricamento delle librerie in memoria da parte della JVM.

7.4.2 Distribuzione contenuti multimediali

I test relativi alla distribuzione di contenuti multimediali sono stati eseguiti utilizzando una topologia di rete rappresentata in Figura 7.7:

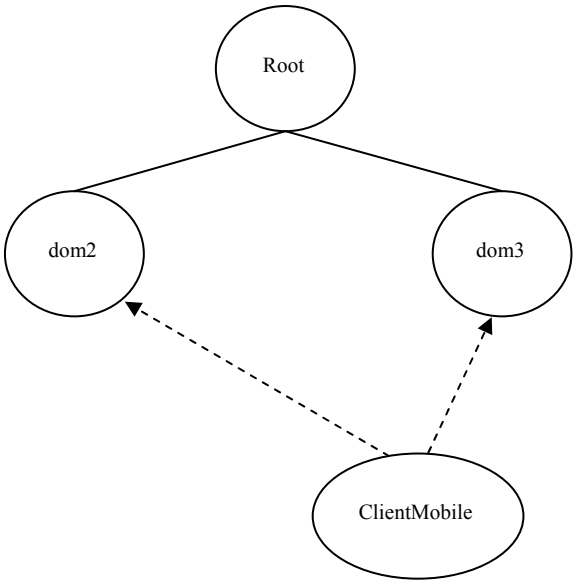


Figura 7.7: Tipologia di rete usata per i test

Il dispositivo mobile sarà emulato utilizzando un computer all'interno della rete, durante la fase di handoff il flusso in arrivo sarà scartato, in modo da simulare la perdita di dati causata della riconnessione al nuovo Access Point.

I computer coinvolti nell'esecuzione dei test, connessi attraverso una rete Ethernet LAN a 100 Mbps, sono equipaggiati nel seguente modo:

Configurazione Hardware:

- Sun Blade 2000 workstation con processore a 900 Mhz e 1024 MB di RAM.

Configurazione Software:

- Sistema operativo SunOS 5.9;
- Java Virtual Machine (JVM) versione 1.5.0;

- Java Media Framework (JVM) Performance Pack per Solaris versione 2.1.1e.

7.4.2 Inizializzazione dello streaming

Con inizializzazione dello *streaming* si intende tutto il periodo che va dal momento della richiesta del contenuto multimediale del cliente fino al momento in cui la visualizzazione risulta possibile. In questo lasso di tempo è possibile individuare tre misure che possono essere rilevate: la fase di *login* con la relativa ricerca dei dati utente, il tempo necessario a ricevere i dati di collegamento al *ProxyWireless* e il tempo che impiega il *datasource* ad arrivare una volta che il collegamento è avvenuto. Per la misurazione di questo tempo è stato esclusa la bufferizzazione sul *ProxyAdaptor* essenzialmente per due motivi, primo perché porterebbe ad un ritardo costante dipendente dalla grandezza del buffer e quindi non rilevante ai fini del test e secondo perché questo ritardo può essere eliminato estendendo il progetto attuale con il sistema di caching locale del flusso video, già presente in MUM.
Nella tabella 2 sono evidenziati i tempi in millisecondi relativi a 50 prove:

	Fase di login	**Arrivo dati collegamento**	**Arrivo datasource**
Caso peggiore	99	4818	13761
Media	41±5	498±7	7246±124

Tabella 2: Inizializzazione del sistema.

Nella tabella è evidenziato il caso peggiore ottenuto all'inizializzazione del sistema, nella quale devono essere configurati i nodi che partecipano allo *streaming*, questo comporta lo scaricamento lungo il *ServicePath* di tutto il software tramite il servizio di *downloading* del codice offerto da MUM. Di norma questo valore non è rilevante ai fini dei test, in quanto nella realtà si considerano sempre sistemi a regime, ma è stato inserito per completezza. La media delle prove è stata eseguita con il sistema a regime, quindi nel caso in cui Java abbia caricato tutte le librerie necessarie.

7.4.3 Gestione dell'handoff

Nella gestione dell'handoff si sono considerati i tempi necessari all'inizializzazione sul nodo di destinazione del *ProxyWireless*, il tempo che intercorre per l'arrivo del *datasource* e nel caso in cui la previsione fosse errata il tempo, lato cliente, prima che arrivino i dati di collegamento, tabella 3 e tabella 4 i tempi espressi in millisecondi relativi a 50 prove:

	Inizializzazione ProxyWireless	**Arrivo datasource**
Caso peggiore	7369	465
Media	5000±39	492±23

Tabella 3: tempi di handoff corretto

	Ritardo errata previsione
Caso peggiore	7887
Media	5800±271

Tabella 4: tempi di handoff errato.

Anche in questo test la prova peggiore è riferita al caso di movimento verso un Place dove non è presente il *software* e di conseguenza il maggiore tempo riscontrato è dovuto al ritardo dovuto al *download* del codice. Nella media dei tempi ottenuta, realizzata con il sistema a regime, la parte più consistente del tempo è dovuta all'inizializzazione del *ProxyWireless*, causata principalmente dalla gestione del flusso RTP, cioè le fasi di ricezione del *datasource* dal *ProxyAdaptor* e l'inizializzazione degli elementi relativi all'invio del flusso verso il *ClientMobile*, in quanto la sola instanziazione dell'entità e il suo avvio richiede mediamente un tempo di 200 millisecondi, in figura 7.8 è evidenziata in rosso la parte di flusso inizializzato nell'*handoff*.

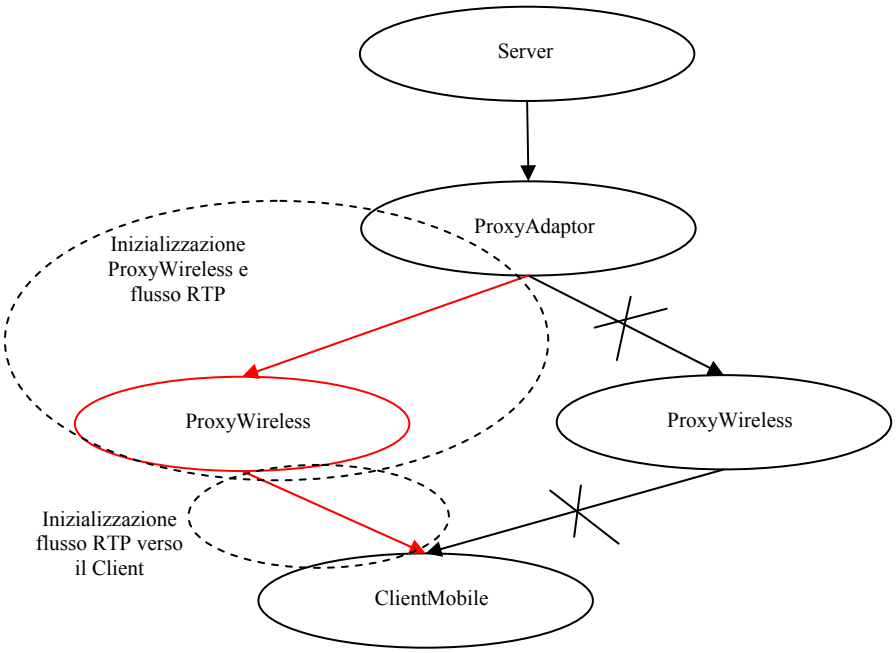


Figura 7.8: Parte di flusso inizializzata durante l'handoff

Il dimensionamento del buffer del cliente dovrà essere effettuato in modo da coprire la somma del tempo nel quale il terminale rimarrà disconnesso e il tempo necessario al sistema per reinizializzare il flusso verso il cliente cioè il tempo di arrivo del *datasource*. Dalla letteratura si viene a conoscenza che il tempo di previsione di un *handoff* può avvenire anche 6 secondi prima dell'effettivo spostamento del terminale, di conseguenza i tempi riscontrati permettono al sistema di essere già predisposto ad accogliere il cliente senza introdurre ritardi ulteriori, mentre la disconnessione a seguito di un *handoff*, ha una durata variabile a seconda della copertura del segnale e del modello di scheda wireless, che spazia da un minimo di 500 millisecondi al un massimo di 7000 millisecondi, di conseguenza per assicurarsi di non perdere *frame* il buffer dovrà essere dimensionato in base alla somma del massimo tempo di *handoff* in caso di previsione errata e il tempo necessario per l'arrivo del *datasource*.

7.4.4 Gestione dell'offload

Nella tabella 5 sono riportati i tempi in millisecondi relativi a 50 test di *offload*, che possono essere divisi in quattro parti: il tempo per la scaricare sul nodo di destinazione il codice necessario, il tempo di inizializzazione di tale del codice, il tempo per l'arrivo del *datasource* e il tempo di *switch* ossia il tempo dal momento in cui avviene la conferma del passaggio sul terminale mobile e il momento il cui avviene la visualizzazione.

	Download del codice	**Inizializzazione del client**	**Arrivo del datasource**	**Tempo di switch**
Caso peggiore	602	142	784	281
Media	<1	73±2	462±18	121±32

Tabella 5: Tempi di offload

Se sul nodo il codice è già presente allora il tempo necessario allo scaricamento è pressoché nullo. I tempi sono minori rispetto al caso di handoff in quanto è presente solo il *ProxyAdaptor* che è già inizializzato e deve solo avviare il flusso verso il cliente, in quanto il *ProxyWireless* non è più necessario, figura 7.9.

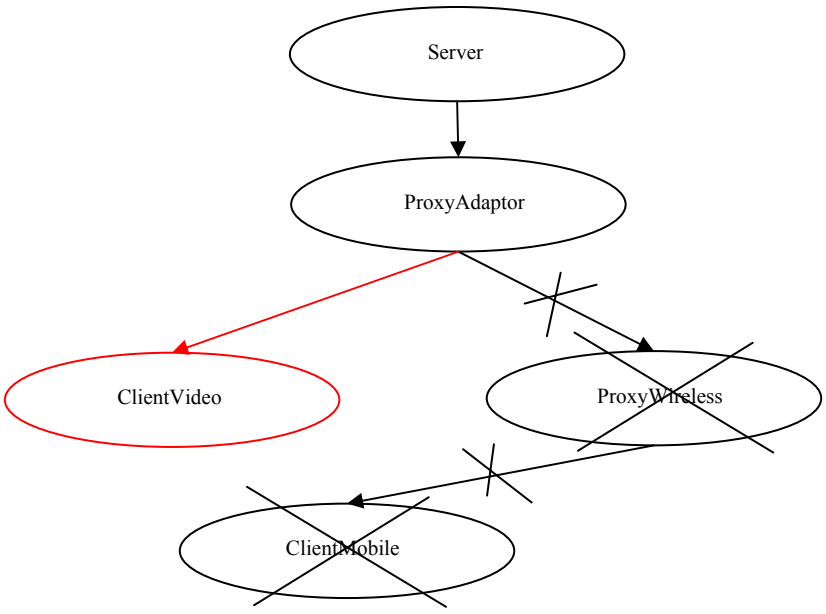


Figura 7.9: Inizializzazione dell'offload

7.4.5 Scalabilità del sistema

Questo test ha lo scopo di verificare la scalabilità del sistema di distribuzione video, valutando il comportamento del sistema al variare del numero di utenti connessi contemporaneamente. I test riguardano i tempi relativi all'arrivo dei dati di collegamento al *ProxyWireless*, il tempo che impiega il flusso video ad arrivare ed infine il carico della CPU sul nodo *ProxyAdaptor* che rappresenta il nodo più caricato in quanto responsabile dell'adattamento, tabella 6.

Numero di richieste	Tempo medio di arrivo dati sul proxy	Tempo medio di arrivo del datasource	Utilizzo della CPU
1	498	7246	1,2%
2	526	7697	4,2%
3	1004	10489	7,0%
5	4061	22090	13,3%
10	6059	34241	20,5%
15	6852	44968	22,4%

Tabella 6: Scalabilità del sistema.

Come si può notare la scalabilità non è molto elevata e cala notevolmente all'aumentare dei clienti (figura 7.10a, 7.10b e 7.10c), in quanto la richiesta di flussi video richiede molte risorse, ma intendendo il servizio proposto come un servizio all'interno di una località non troppo estesa è logico aspettarsi un numero di richieste non troppo elevato, ponendo quindi la scalabilità in secondo piano.

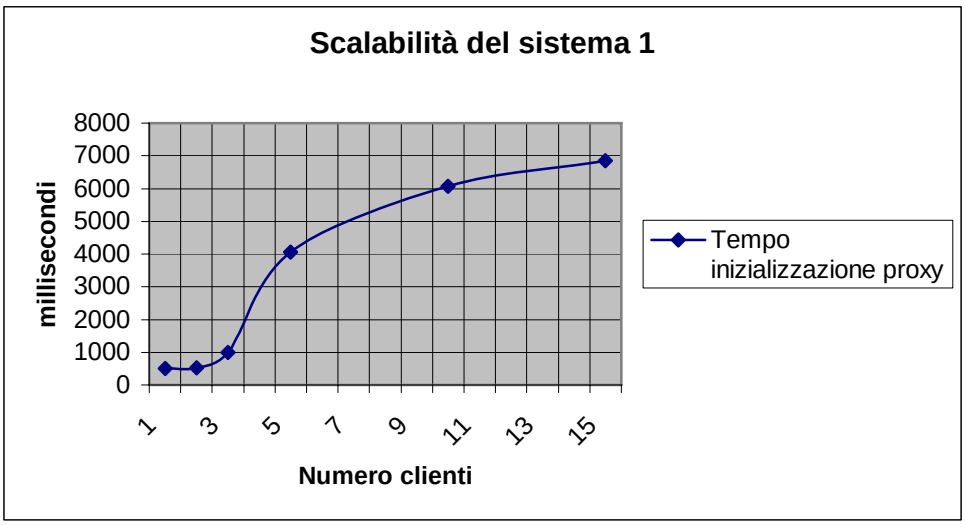


Figura 7.10a : Tempo medio di arrivo dati di connessione.

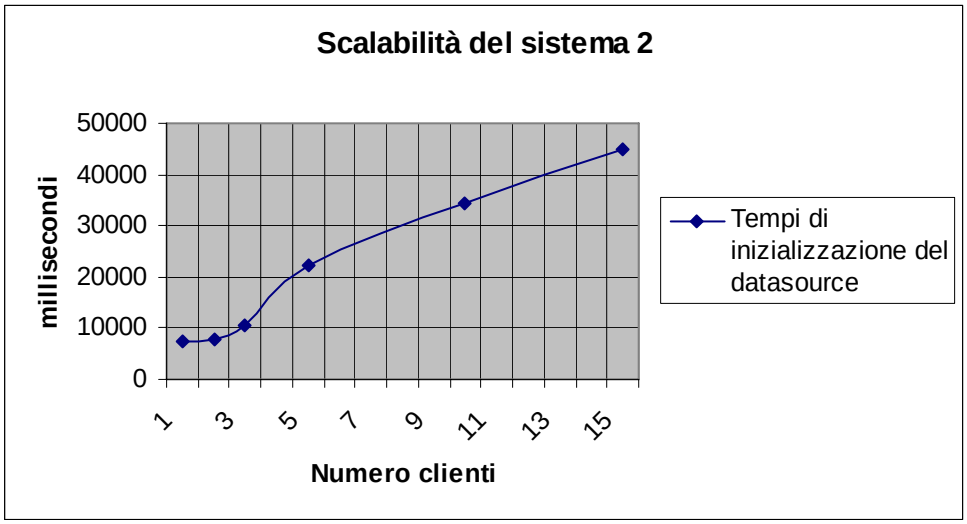


Figura 7.10b : Tempo medio di inizializzazione del datasource.

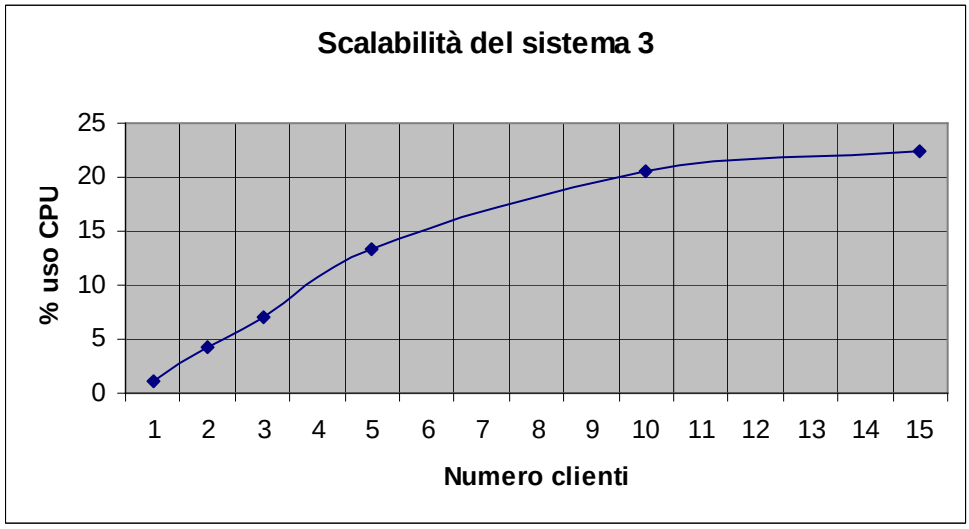


Figura 7.10c :Utilizzo della CPU

7.5 Conclusioni

In quest'ultimo capitolo sono stati affrontati gli aspetti legati all'implementazione del servizio di discovery e il sistema di distribuzione video, evidenziando le scelte e le ipotesi effettuate durante la realizzazione di un prototipo. Inoltre sono stati presentate le prove fatte sul sistema durante la fase di testing.

Il sistema di discovery si rivelato molto efficiente grazie al sistema di caching inserito, ed inoltre l'accesso a standard tramite mediatori, fornisce un ritardo trascurabile rispetto all'accesso diretto allo standard stesso.

Dai test relativi all'inizializzazione del sistema di distribuzione video, si sono ottenuti tempi di inizializzazione dell'ordine dei 7 secondi, inoltre i test hanno evidenziato come il sistema sia limitato in scalabilità, a causa dell'elevato consumo di risorse necessario al flusso RTP, nonostante questo i tempi si possono essere considerati molto buoni, in quanto l'attesa è presente solo nella fase iniziale e dal punto di vista dell'utente finale non arreca troppo disturbo.

Dai test relativi alla gestione dell'*handoff* si è ottenuto un tempo dell'ordine dei 5 secondi, si è quindi trovato che la velocità di movimento del Proxy è limitata dall'inizializzazione del flusso RTP che deve essere ridiretto verso il client, nonostante questo il tempo ottenuto può ritenersi molto soddisfacente considerando che il codice è realizzato interamente in Java, inoltre le previsioni hanno un anticipo tale da garantire la sicurezza di avere in tempo un'inizializzazione preventiva e di conseguenza a seguito di un handoff si ha solo un ritardo circa di 400 millisecondi prima dell'arrivo del flusso multimediale. Sarà questo tempo che dovrà essere aggiunto al tempo nel quale il terminale rimarrà disconnesso dalla rete a seguito dell'*handoff*, costituendo il tempo totale nel quale il buffer dovrà agire garantendo continuità.

CONCLUSIONI

Durante lo sviluppo del progetto di tesi sono state approfondite diverse tematiche che hanno spaziato dalla ricerca di servizi all'interno di una rete e loro configurazione automatica, alla gestione del processo di handoff, fino all'adattamento di servizi multimediali per la distribuzione di contenuti su terminali mobili.

Una prima considerazione che può essere ottenuta da quanto studiato è la mancanza di uno standard unico per la ricerca e la descrizione dei servizi presenti nella rete, rendendo di conseguenza difficile l'interoperabilità fra dispositivi che fanno uso di standard diversi. La soluzione proposta ha cercato di arginare questo problema introducendo una elemento intermedio che funga da mediatore, realizzando inoltre il tutto su una piattaforma indipendente dalla macchina sottostante. Il sistema di discovery così ottenuto è quindi in grado di operare su qualsiasi standard previa realizzazione di un mediatore appropriato.

Per quanto riguarda la distribuzione di contenuti multimediali su terminali mobili, la presenza di una infrastruttura di supporto che consente ai nodi mobili di prevedere la rete wireless alla quale si collegheranno, ha permesso la realizzazione di un sistema in grado di riconfigurarsi pro-attivamente ed in modo automatico creando e attivando gli elementi necessari alla distribuzione del contenuto multimediale nella locazione prevista accelerando così i tempi richiesti per il completamento dell'handoff.

Il sistema realizzato è stato progettato per essere integrato in MUM, una infrastruttura di supporto, volta alla semplificazione dello sviluppo applicativo delle applicazioni multimediali, che attraverso questa tesi è stata arricchita con un servizio di discovery che si occupa della ricerca di dispositivi e servizi ed inoltre l'infrastruttura è stata ampliata per gestire la distribuzione di contenuti multimediali su terminali mobili. In particolare, la parte relativa alla gestione dei dispositivi mobili, utilizzando il servizio di discovery, è in grado di rilevare eventuali servizi di visualizzazione nei quali sia possibile continuare la propria sessione di lavoro e configurarli prima di informare il cliente, in modo da rendere il cambio istantaneo.

L'architettura proposta può essere concettualmente divisa in tre parti, la prima, dedicata alla ricerca di servizi nella rete tramite interoperabilità con diversi standard, la seconda relativa alla gestione della distribuzione di flussi multimediali verso terminali

mobili, in grado di adattare il flusso in base alle caratteristiche del terminale e di gestire i movimenti del terminale, e la terza, la realizzazione di un componente in grado di eseguire su terminali mobili senza richiedere troppe risorse.

I risultati ottenuti dai test vengono ritenuti significativi, soprattutto se si considera che l'ipotesi sulla quale si basa l'inizializzazione del sistema è che tutto il software sviluppato venga scaricato e inizializzato a tempo di esecuzione. In particolare, l'inizializzazione del flusso RTP lungo il *Service Path* è risultata l'operazione più pesante, con tempi dell'ordine di 7-8 secondi, tempo molto soddisfacente in quanto si tratta di una inizializzazione e quindi presente solo all'avvio. Nondimeno, la possibilità di avere una previsione della località di destinazione del client in caso di *handoff*, garantisce che il sistema possa reagire con un certo anticipo assicurando così una presentazione dei contenuti senza salti (*seamless*). Infatti i risultati sperimentali sono molto incoraggianti, in quanto il tempo necessario all'*handoff* risulta essere inferiore alla somma dell'anticipo fornito dalla previsione e del tempo di disconnessione e quindi garantisce un inizializzazione degli elementi prima dell'arrivo del dispositivo mobile, di conseguenza il tempo di arrivo del flusso, di soli 400 millisecondi, rappresenta l'unico tempo che deve essere conteggiato all'interno del buffer per consentire una visualizzazione senza salti. Per quanto riguarda l'*offload*, grazie alla completa configurazione del servizio su rete fissa, si sono ottenuti tempi ancora migliori, dell'ordine dei 200 millisecondi; conseguentemente, dal punto di vista dell'utente, il cambio di terminale è pressoché istantaneo.

Sviluppi futuri del sistema potranno estendere l'infrastruttura attuale con l'intento di offrire un supporto alla continuità non di flussi multimediali, ma anche di flussi dati; per esempio, per gestire elaborazioni distribuite eseguite da agenti mobili, che elaborino flussi di informazioni provenienti da un server centrale, e che per svolgere questo compito debbano migrare in diverse località a seconda della presenza o meno di determinati servizi necessari per la loro elaborazione, ottenendo queste informazioni dal servizio di discovery locale. Inoltre potrebbe essere interessante sfruttare il progetto esistente per gestire multicast o broadcast, ad esempio per facilitare la realizzazione di servizi di radio/news sul web, utilizzando uno stesso proxy adattatore ed un'unica entità server per più utenti, accelerando così le fasi di inizializzazione e migliorando l'occupazione globale di risorse.

BIBLIOGRAFIA

[**.NET**] Informazioni su .NET si trovano all'indirizzo: http://msdn.microsoft.com/net/

[**BlueT**] Informazioni su BlueTooh al sito: http://www.bluetooth.com

[**ChenKotz**] Guanling Chen and David Kotz. Context aggregation and dissemination in ubiquitous computing systems. In Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications, pages 105–114. IEEE Computer Society Press, June 2002.

[**CK03**] Guanling Chen and David Kotz. Context-Sensitive Resource Discovery. In Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, pages 243–252, Fort Worth,TX, March 2003.

[**CLK04**] Guanling Chen, Ming Li, and David Kotz. Design and Implementation of a Large-Scale Context Fusion Network. In Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, pages 246–255, Boston, MA, August 2004.

[**Dea00**] N.Deason, .SIP and SOAP.,IETF,2000. http://search.ietf.org/internet-drafts/draft-deason-sip-soap-00.txt

[**ER05**] proxy-based adaption for mobile computing Markus Endler, Hana Rubinsztejn, Ricardo C. A. Da Rocha, Vagner Sacramento Dipartimento de Informatica, PUC-Rio, Brazil April 1, 2005

[**FKK96**] Alan O. Freier, Philip Karlton, Paul C. Kocher, .The SSL Protocol., Netscape Communications Corporation, 1996. http://home.netscape.com/eng/ssl3

[**Fos03**] Foschini L., "Gestione di flussi multimediali in reti integrate fisse e mobili", tesi di laurea 2003.

[**Genea00**] J. Cohen, S. Aggarwal, Y. Y. Goland, .General Event Notification Architecture Base: Client to Arbiter., Microsoft, 2000 http://www.upnp.org/draft-cohen-gena-client-01.txt

[**GolSh00**] Y.Y.Goland, J.C.Schlimmer, .Multicast and Unicast UDP http Messages., Microsoft, 2000. http://www.upnp.org/draft-goland-http-udp-03.txt

[**IEC61883.1**] Consumer Audio/Video Equipment . Digital interface, parti 1-5,International Electrothecnical Commission, Ginevra, 1998.

[**IECWN01**] L. M. Feeney and M. Nilsson. Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment. In IEEE INFOCOM, 2001.

[**INS**] Adjie-Winoto, W., Schwartz, E., Balakrishnan, H., Lilley, J.: The design and implementation of an intentional naming system. In: Proc. ACM Symposium on Operating Systems Principles. (1999) 186–201

[**INST02**] Magdalena Balazinska, Hari Balakrishnan, and David Karger. INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery. In Proceedings of the First International Conference on Pervasive Computing, Zurich, Switzerland, August 2002.

[**JINI**] Informazioni su Jini al sito http://www.jini.org

[**Konno**] CyberLink, package for UPnP devices, prodotto da Satoshi Konno, reperibile al sito http://www.cybergarage.org/net/upnp/java/index.html

[**Mon04**] Monaco L., "Infrastruttura per la profilazione e l'adattamento di servizi multimediali", tesi di laurea 2004.

[**MR04**] M. Muňoz and C. Garcìa-Rubio. A New Model for Service and Application Convergence In B3G/4G Networks. IEEE Wireless Communications, 11(5):6–12, Oct. 2004.

[**MUM**] MUM Home Page. http://www.lia.deis.unibo.it/Research/MUM/

[**NamDM**] Naming and Discovery in Mobile Systems Guanling Chen, Kazuhiro Minami, David Kotz. http://www.cs.dartmouth.edu

[**OpenCOM**] M. Clarke, G. Blair, G. Coulson and N. Parlavantzas, "An Efficient Component Model for the Construction of Adaptive Middleware", Proceedings of Middleware 2001, Heidelberg, Germany. November, 2001.

[**OpenORB**] G. Blair, G. Coulson, A. Andersen, L. Blair, M. Clarke, F. Costa, H. Duran-Limon, T. Fitzpatrick, L. Johnston, R. Moreira, N. Parlavantzas and K. Saikoski, "The design and implementation of Open ORB 2", IEEE Distributed Systems Online, 2(6), Sept 2001.

[**Pastry**] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems". IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pages 329-350, November, 2001.

[**PDPGDSL**] Celeste Campo, Mario Muñoz, José Carlos Perea, Andrés Marìn, Carlos Garcìa-Rubio, Conf. on Pervasive Computing and Communications Workshops IEEE 2005

[**Perkins**] Charles E. Perkins. Mobile networking through mobile IP. IEEE Internet Computing, 2(1), January 1998.

[**Psachno**] Nicholas Nicoloudis, Christine Mingine School of Computer Science and Software Engineering Monash University Australia, Psachno: A Dynamic and Generic Discovery Framework within a Peer-to-Peer Network Model, Tenth Asia-Pacific Software Engineering Conference 2003

[**PS98**] E. Pittura and G.Samaras Data management for mobile computing. Kluver Acedemic press, 1998

[**ReMMoC**] ReMMoC (Reflective Middleware for Mobile Computing) A Reflective Framework for Discovery and Interaction in Heterogeneous Mobile Environments, Paul Grace, Gordon S. Blair, Sam Samuel.

[**RFC1034**] P.Mockapetris, .Domain Names . Concepts and facilities., IETF, RFC1034, 1987. http://www.rfc-editor.org/rfc/rfc1034.txt

[**RFC2131**] R. Droms, .Dynamic Host Configuration Protocol., IETF, 1997. http://www.rfc-editor.org/rfc/rfc2131.txt

[**SDD02**] S. Helal. Standards for Service Discovery and Delivery. IEEE Pervasive Computing, pages 95–100, jul/sep 2002.

[**Sin00**] Sing Li, Professional Jini, Wrox 2000, Pag 267.

[**SLP**] Informazioni su Service Location Protocol SLP sul sito http://srvloc.sourceforge.net/

[**SOMA**] Bellavista P., Corradi A., Stefanelli C., "Mobile Agent Middleware to Support Mobile Computing", IEEE Computer, Vol. 34, No. 3, pages 73-81, March 2001

[**TinyObj**] Pavel Poupyrev, Takahiro Sasao, Shunsuke Saruwatari, Hiroyuki Morikawa, Tomonori Aoyama, Service Discovery in TinyObj: Strategies and Approaches IEEE April 2005.

[**Tro00**] R.Troll, .Automatically Choosing an IP Address in an Ad-Hoc IPv4 Network., IETF, 2000. http://search.ietf.org/internet-drafts/draft-ietf-dhc-ipv4- autoconfig-05.txt

[**UnQL**] Abiteboul, S.: Querying semi-structured data. In: ICDT. Volume 6. (1997) 1–18

[**UPnP**] Informazioni su Universal Plug & Play Protocol (UPnP) al sito: http://www.unpnp.com

[**XSet**] Zhao, B., Joseph, A.: Xset: A lightweight database for internet applications. http://www.cs.berkeley.edu/~ravenben/publications/saint.pdf10 (2000)

RINGRAZIAMENTI

Vorrei ringraziare i miei genitori per l'appoggio ricevuto durate tutto il mio percorso universitario.
Infine, un sentito ringraziamento al prof. Antonio Corradi per la sua disponibilità e gentilezza e al dott. Ing. Luca Foschini per i suoi preziosi consigli e per la costante presenza.