**UNIVERSITÀ DEGLI STUDI DI BOLOGNA**
FACOLTÀ DI INGEGNERIA

Corso di Laurea in Ingegneria Informatica
Reti di Calcolatori

# GESTIONE DI FLUSSI MULTIMEDIALI IN RETI INTEGRATE FISSE E MOBILI

Relatore: Chiar.mo Prof. Ing. ANTONIO CORRADI
Correlatori: Dott. Ing. PAOLO BELLAVISTA
Chiar.mo Prof. Ing. CESARE STEFANELLI

Candidato: LUCA FOSCHINI

ANNO ACCADEMICO 2002–2003

*Parole chiave*
Qualità di Servizio
Sistemi Multimediali
Mobilità utente e terminale
Location Awareness
Agenti Mobili

## INDICE

# Introduzione

La rapidità che ha contraddistinto in questi ultimi anni la diffusione delle tecnologie wirless e la sempre maggiore domanda da parte degli utenti di servizi accessibili dovunque, richiede la progettazione di una piattaforma che assista i continui movimenti delle persone e dei terminali da loro utilizzati. Più che di una innovazione tecnologica si tratta di una vera e propria rivoluzione del modo di concepire i sistemi distribuiti ed i modelli di progettazione degli stessi.

A tutt'oggi la maggioranza delle applicazioni si richiama infatti al modello di comunicazione cliente/servitore ed alle sue evoluzioni. In tale nuovo scenario, tuttavia, l'utilizzo di questo strumento di progettazione sembra non essere più sufficiente e proprio da ciò nasce l'interesse ad esplorare nuovi schemi di progettazione più dinamici, quali quelli basati sulla mobilità di codice.

D'altro canto assistiamo anche ad una sempre maggior diffusione di applicazioni multimediali che ogni giorno diventano più pervasive fino ad approdare, com'è recentemente accaduto, alla telefonia mobile. Grosso limite alla larga diffusione di tale applicazione rimane comunque l'ingente richiesta di risorse e la carenza di adeguati strumenti per la gestione delle stesse. Inoltre pur esistendo molte proposte e molti studi per ciò che riguarda lo streaming su rete fissa, non esistono ancora architetture ben assestate per quello che riguarda il supporto alla mobilità.

In questo contesto si colloca il nostro lavoro di tesi con l'intento di costruire una piattaforma per la gestione dei flussi multimediali, che offra un valido supporto all'utilizzo delle risorse e faciliti lo sviluppo di applicazioni di nuova generazione, ispirate a modelli di ubiquitous computing. Si intende in particolare esplorare l'utilizzo del paradigma ad agenti mobili per la realizzazione di un middleware, che dovrà offrire supporto alla mobilità degli utenti e dei terminali, occupandosi anche della gestione delle risorse e della configurazione del sistema distribuito. Verranno perciò implementati una serie di utili servizi, che saranno resi accessibili alle applicazioni sviluppate al di sopra del supporto in progetto, nell'intento di dare maggior visibilità e controllo della piattaforma sottostante allo sviluppatore dell'applicativo.

La tesi sarà organizzata come segue. Nel capitolo 1 vengono date alcune definizioni fondamentali per la successiva comprensione e vengono fissati i principi architetturali che dovrebbero guidare lo sviluppo di un suppoto multimediale. Nel capitolo 2 vengono considerati i meccanismi per l'implementazione di tali principi e vengono presentate sei architetture per il supporto allo streaming multimediale, con l'intento di trarre utili suggerimenti per il successivo sviluppo della piattaforma multimediale di supporto. Il capitolo 3 è dedicato alla presentazione dei diversi modelli di comunicazione distribuita, del paradigma ad agenti mobili; viene poi presentata la piattaforma SOMA, utilizzata come ambiente ad agenti per lo sviluppo del lavoro di tesi. Nel capitolo 4 viene presentato lo standard 802.11 per la realizzazione di reti wireless e le Wireless API, una libreria Java per la gestione delle schede wireless. Il capitolo 5, 6 e 7 sono poi dedicati allo sviluppo del progetto. Nel capitolo 5 viene presentato lo scenario d'utilizzo del sistema e l'architettura logica del middleware insieme ai servizi offerti per lo sviluppo di applicativo. Nel capitolo 6 viene dettagliato il progetto della piattaforma, e nel capitolo 7 viene descritta una applicazione per lo streaming video realizzata al di sopra della piattaforma stessa.

CAPITOLO 1

# 1 Middleware di supporto per applicazioni multimediali

In questo capitolo verranno esaminate le principali problematiche legate allo sviluppo di un'infrastruttura che supporti applicazioni multimediali (*multimedia middleware*). Questa area di ricerca è assai ampia, e molto lavoro è stato svolto negli ultimi anni. Si cercherà inoltre di delineare i principi architetturali che sottendono lo sviluppo di questo particolare tipo di infrastruttura.

*Multimedia middleware* è infatti un termine generico che racchiude al suo interno problematiche varie legate ai diversi livelli di progetto (ci riferiamo in questo contesto ai 7 livelli definiti dallo stack OSI [OSI]).

Questo lavoro di tesi cerca di coniugare due aspetti che oggi più che mai vengono ritenuti strategici nella progettazione di un'infrastruttura di nuova generazione (si veda a proposito [GEK01]): il supporto per la mobilità, sia dell'utente che del terminale, e il supporto per lo streaming che avverrà su rete fissa e mobile.

Questo capitolo, così come i seguenti, intende creare il background necessario per la comprensione dell'architettura realizzata, proponendo inoltre una classificazione delle principali problematiche e soluzioni presenti oggi in questo campo. Il lavoro, naturalmente, si concentrerà poi su alcuni degli aspetti considerati.

## *1.1 Alcune definizioni basilari*

Prima di trattare le problematiche legate alla realizzazione di multimedia middleware, diamo in questa sezione alcune definizioni fondamentali per la comprensione degli argomenti che saranno affrontati in seguito. In particolare vengono definiti il concetto di **Flusso** e alcuni parametri che saranno richiamati nel corso della trattazione.

### 1.1.1 Flusso

Il flusso è un'astrazione che viene introdotta per specificare la **fruizione** di un singolo oggetto multimediale. Con fruizione si intende recapito e il rendering dell'oggetto multimediale richiesto.

Per singolo *oggetto multimediale* si intende un particolare tipo di dato, caratterizzato dal fatto di *variare con continuità* e di essere *soggetto a vincoli di tempo*; tale oggetto multimediale potrebbe essere per esempio un video, un audio, o un generico flusso dati.

La prima definizione data serve per considerare le tre attività sopra elencate come integrate e legate l'una all'altra. Tale definizione, come vedremo, è di importanza fondamentale, nel momento in cui si inizi a esigere qualità di servizio (Quality of Service, in seguito indicata con l'acronimo QoS). Il singolo flusso può infatti essere considerato come unità elementare cui riferire la QoS.

I flussi sono l'entità minima identificabile per la trasmissione di oggetti multimediali, e possono essere sia di tipo *unicast* (da 1 sorgente a 1 destinatario) che di tipo *multicast* (da 1 sorgente a N destinatari). Il flusso attraversa diverse entità (transitando dalla sorgente al destinatario/destinatari) ognuna delle quali, in un'ottica di garanzia di QoS, dovrà riservare una parte delle risorse disponibili per processare il flusso stesso.

Vediamo ora quale sia la semantica che regola la trasmissione dei flussi. La definizione di oggetto multimediale data sopra sottolinea il fatto che esso è per sua natura continuo; questo aggettivo si riferisce alla percezione che l'utente finale ha dello streaming multimediale. In realtà però è noto che, per poter essere processati da un elaboratore, gli oggetti multimediali devono essere discretizzati e quantizzati.

Ne segue che, se consideriamo il flusso ad un livello di astrazione più basso, lo possiamo pensare come una sequenza discreta di dati soggetti a vincoli di tempo: l'istante di ricezione degli stessi da parte del ricevente ne determina la validità. In altri termini, se un dato viene ricevuto in tempo utile per il suo utilizzo da parte del ricevente, verrà consumato, altrimenti sarà scartato e non sarà ritenuto valido.

Una semantica di questo tipo viene detta isocrona. Perciò diciamo che i flussi multimediali sono per loro natura **isocroni**.

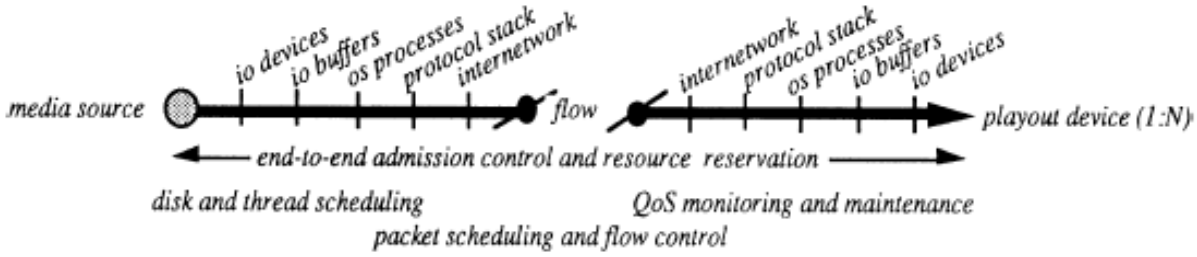


**Figura 1.1:** risorse coinvolte nel delivery di materiale multimediale.

La figura 1.1, tratta da [CAA98], rappresenta sinteticamente le varie risorse coinvolte nel delivery di un qualsiasi flusso multimediale. In particolare viene sottolineata la necessità di assicurare la QoS su base end-to-end, coinvolgendo cioè in questa attività le diverse entità che processano flusso stesso.

### 1.1.2 Larghezza di banda

La larghezza di banda è il parametro che misura la portata di un canale trasmissivo digitale, poiché definisce il numero massimo di dati che possono transitare nella sezione nell'unità di tempo. Solitamente l'unità di misura è data da bit/secondo (bps), parametro importante perché rappresenta un vincolo assai stringente per la trasmissione di oggetti multimediali. Un flusso multimediale, infatti, richiede solitamente una banda molto ampia e ormai da diversi anni sono allo studio algoritmi di compressione che hanno come scopo quello di diminuire i requisiti di banda per la trasmissione di oggetti multimediali, soprattutto di tipo video.

L'aspetto negativo dell'uso di tali algoritmi è che richiedono (solitamente dal lato della sorgente) una computazione ulteriore piuttosto consistente, occupando la risorsa più costosa, la CPU. Tale problematica non verrà approfondita in questa tesi, anche se si terrà conto di alcuni ordini di grandezza determinati dall'uso di un algoritmo di compressione piuttosto che di un altro, in termini di utilizzo del processore e di occupazione di banda.

### 1.1.3 Parametri caratterizzanti i flussi: latenza, jitter, loss rate e skew

La **latenza** è il parametro che misura il ritardo introdotto dalla linea di trasmissione, cioè il tempo impiegato dai dati discretizzati e impacchettati per giungere dalla sorgente al ricevente.

Tale parametro è rilevante soprattutto nelle applicazioni che richiedono l'interazione con un sistema remoto, o con altri utenti a distanza. A tale proposito è stato valutato che il ritardo massimo tollerabile per una conversazione sia di 150 ms, mentre per una applicazione di tipo Video on Demand (VoD), nella quale si interagisce con un “videoregistratore remoto”, sia al massimo di 500 ms. Ovviamente la latenza non è costante, ma soggetta a variazioni, sarà quindi rappresentata da una variabile aleatoria.

Un parametro ancora più interessante per la caratterizzazione dei flussi è il **jitter**. Tale parametro misura la varianza della latenza, cioè lo scostamento rispetto al valor medio della stessa. In figura 1.2, tratta da [BAF01], si può osservare la rappresentazione analitica di quanto espresso fin qui a parole. Viene

rappresentata la latenza dei pacchetti dati come densità di probabilità, e il jitter. La freccia contrassegnata con "Latenza" rappresenta il valor medio della latenza

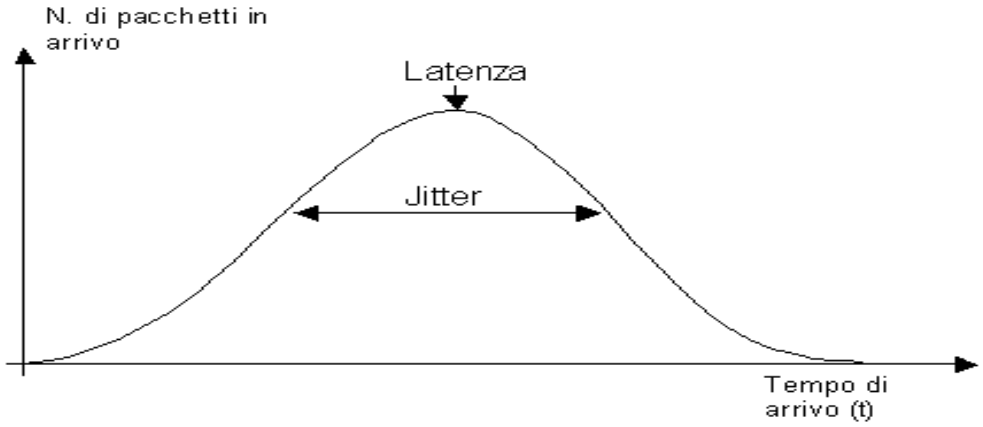


**Figura 1.2:** Latenza dei pacchetti.

La situazione ideale si verificherebbe se tutti i pacchetti giungessero al ricevente nello stesso ordine nel quale sono stati generati, e venissero processati con la stessa velocità con la quale arrivano. In tal caso non ci sarebbe bisogno di bufferizzare i dati arrivati al ricevente. Nella realtà però si pongono due problemi:

- i pacchetti possono giungere con un ordine diverso da quello col quale sono stati spediti;
- la latenza è una variabile aleatoria: bisognerà allora bufferizzare i dati in modo da tollerare una certa varianza della latenza stessa.

In relazione al secondo problema delineato si comprende l'importanza del jitter nel dimensionamento del buffer di ricezione. Il jitter fornisce inoltre un'informazione importante per valutare lo stato corrente del carico del canale trasmissivo. Per quanto visto sopra al punto 1.1.1, poiché i flussi multimediali sono isocroni, l'arrivo con troppo ritardo di un pacchetto, ne determina l'inutilità. Fortunatamente però molte applicazioni multimediali possono tollerare delle perdite di dati.

La **loss rate** è il parametro che quantifica la percentuale di dati “perdibile” in un flusso. Tale parametro è ortogonale agli altri, possono cioè esistere flussi diversi con la stessa occupazione di banda e jitter, ma loss rate molto diversa.

L'ultimo parametro che consideriamo è lo **skew**. Fino ad ora abbiamo sempre considerato un singolo flusso, con i relativi problemi di sincronizzazione. Esiste però la possibilità che più oggetti multimediali costituiscano insieme un unicum, che verrà definito *presentazione multimediale*. In tal caso parliamo di sincronizzazione inter-oggetto, contrapponendola alla sincronizzazione intra-

oggetto che si riferisce al singolo flusso. Lo skew quantifica (come intervallo di tempo) lo sfasamento nell'arrivo di flussi diversi facenti parte della stessa presentazione, ad esempio un flusso audio e video per un film.

Con quest'ultima definizione è conclusa questa sezione nella quale sono state date alcune definizioni di base sul concetto di flusso, oggetto multimediale, presentazione multimediale e la loro caratterizzazione.

## *1.2 Applicazioni multimediali: problematiche e linee evolutive*

Da quando nei laboratori della Xerox di Palo Alto, poi alla Apple, sono state sviluppate le prime interfacce grafiche (Graphical User Interface, GUI) sono passati molti anni. Da allora le applicazioni multimediali, di cui le GUI erano forse la prima elementare espressione, hanno raggiunto oggi un'ampia diffusione.

Molti di noi ne fanno esperienza scaricando musica dal Web, o anche ascoltando la radio via Internet, o sfruttando la rete per fare telefonate intercontinentali pressoché gratis. Questo solo per indicare alcune applicazioni oggi largamente diffuse. Le applicazioni a più alto valore aggiunto sono poi quelle che prevedono un'interazione in tempo quasi reale con il sistema o con altri interlocutori. Si pensi per esempio alle prospettive aperte da applicazioni di videoconferenza, fino ad arrivare all'e-learning. Abbiamo cioè un ampio ventaglio di applicazioni che vanno da quelle “di intrattenimento” a quelle cosiddette “mission critical”.

Denominatore comune di tutti questi tipi di applicazioni è l'elevata richiesta di risorse e di garanzie di qualità di servizio che aumentano all'aumentare della criticità dell'applicazione.

Ci si potrebbe ragionevolmente chiedere per quale motivo questo tipo di applicazioni non siano ancora disponibili dappertutto e con accesso da un qualsiasi device. Le risposte sono da ricercare principalmente nell'architettura del software di base al di sopra del quale è sviluppata la maggior parte dei sistemi distribuiti, e nel modo in cui Internet e i suoi protocolli siano stati pensati sin dall'inizio della loro esistenza. Come vedremo, i limiti con i quali ci si confronta sono sia di tipo fisico (carenza di risorse), sia dovuti al software.

Per esempio, l'odierna comunicazione di rete si appoggia largamente sulla suite di protocolli TCP/IP, che non dà alcuna garanzia di qualità di servizio. Il middleware, come vedremo, è quella parte dell'architettura software che cerca di sopperire a molte di queste mancanze, e fornisce alle applicazioni multimediali che vengono sviluppate al di sopra di esso garanzie in termini di QoS e utili servizi per portare a termine la loro missione.

### 1.2.1 Carenza di risorse fisiche

Uno dei fattori che maggiormente limitano la diffusione delle applicazioni multimediali è la grande richiesta di risorse. In questa sezione vogliamo indicare quali sono le risorse fisiche più sollecitate nel recapito di un flusso multimediale (la figura 1.1 presentata in precedenza sintetizza bene il quadro che stiamo analizzando).

Consideriamo tre risorse: la CPU, la banda trasmissiva e la memoria.

- La **CPU** è la risorsa più contesa perché viene coinvolta in diverse fasi del recapito della presentazione multimediale:
  - impacchettamento/spacchettamento dei dati;
  - codificazione/decodificazione del flusso (cioè l'esecuzione di algoritmi di compressione/decompressione);
  - rendering dell'oggetto multimediale.

  Ciò solo per citare alcune delle operazioni più costose operate durante la trasmissione di un flusso multimediale. Solitamente i requisiti, per un dato tipo di CPU, vengono espressi come frazioni di tempo di un periodo. Per esempio un'applicazione potrebbe dichiarare che necessita di 10 ms di CPU, ogni 100 ms; oppure, sotto forma percentuale che necessita del 10% dei cicli del microprocessore.
- La **banda trasmissiva** è un'altra risorsa che solitamente viene fortemente richiesta. Basti pensare che un filmato non compresso con qualità video TV standard richiede approssimativamente una banda pari a 120 Mbps (Mega bit per second), che eccede la capacità di molte delle Ethernet oggi utilizzate, pari a 100 Mbps. Come già accennato sopra (al punto 1.1.2), l'utilizzo degli algoritmi di compressione riduce notevolmente la richiesta di banda. Per esempio, utilizzando la prima delle codifiche studiate dal Moving Picture Experts Group (MPEG), cioè MPEG-1, possiamo abbassare questo valore a 1,5 Mbps. Naturalmente queste trasformazioni richiedono un costo, cioè pesano maggiormente sulla risorsa CPU.
- L'ultima risorsa che consideriamo è la **memoria**. La trasmissione e visualizzazione dei flussi multimediali richiede un'elevata quantità di memoria. Come già visto, il ricevente dovrà bufferizzare i dati in arrivo per poter continuare la visualizzazione dell'oggetto multimediale, anche in presenza di momentanei problemi di trasmissione. Inoltre ci sarà bisogno di altri buffer per salvare i dati durante la loro trasformazione ad opera dei CODEC (cioè i moduli di codifica/decodifica, che operano le

compressioni/decompressioni dei dati), e per operare il rendering degli oggetti multimediali in arrivo.

Delle tre risorse considerate questo lavoro di tesi prenderà in esame le prime due. L'infrastruttura in progetto fornirà cioè, a livello applicativo, delle opportune funzionalità per trattare la prenotazione e il monitoraggio delle risorse CPU e banda trasmissiva. Questa scelta è stata fatta perché queste due risorse vengono ritenute le più sensibili ai fini della fruizione dei flussi multimediali.

### 1.2.2 Mancanza di espressività nelle attuali piattaforme

Un secondo problema che si affronta quando si sviluppano applicazioni multimediali è quello della carenza di supporto, nelle attuali piattaforme, per le nuove tipologie di applicazioni Realtime e Soft-Realtime, perciò anche delle applicazioni multimediali. Finora infatti la scelta solitamente operata sia a livello di infrastruttura di rete che a livello di sistemi operativi, è stata quella di supportare traffico e applicazioni con architetture di tipo best-effort, che tuttavia non offrono particolari garanzie di QoS. Questa scelta, giustificata da un principio di trasparenza, secondo cui si volevano nascondere all'utilizzatore alcuni dettagli demandandoli ai livelli sottostanti, risulta oggi un vincolo eccessivo. In queste particolari tipologie di applicazioni, infatti, emerge una forte esigenza di poter interagire in modo più diretto con le risorse, si ha cioè bisogno, più che di trasparenza, di maggiore espressività.

Dal momento che non esistono ancora standard *de facto*, si possono seguire due strade diverse. La prima consiste nell'estendere i sistemi operativi per supportare anche questo nuovo tipo di applicazioni, la seconda consiste nello sviluppare uno strato di middleware che realizzi il supporto necessario per la gestione delle risorse. Descriveremo ora brevemente i vantaggi e gli svantaggi di queste due alternative, anche se si può notare da subito come le due soluzioni proposte siano fra loro ortogonali e possano coesistere all'interno dello stesso sistema.

Nella prima ipotesi si va ad agire a livello di sistema operativo (SO) mettendo a disposizione dello sviluppatore opportune API (Application Programming Interface), che estendano il set di API usato solitamente. In questo modo i processi con vincoli di tipo Realtime (RT) o Soft-Realtime (SRT) possono usare il set di API esteso, mentre gli altri processi di tipo Best-Effort possono continuare ad usare il vecchio set di API. Nella seconda ipotesi si

inserisce uno strato di middleware, che si occuperà di gestire in modo opportuno le varie richieste fra il sistema operativo e le applicazioni. La figura 1.3, rappresenta graficamente le due alternative. Naturalmente per ognuna di queste due soluzioni esistono lati positivi e lati negativi.

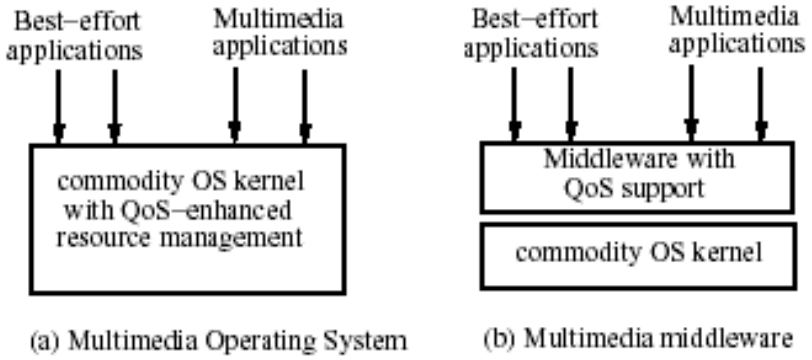


**Figura 1.3:** due possibili alternative architetturali per supportare applicazioni multimediali.

La prima soluzione richiede di intervenire sul kernel, modificando, o meglio estendendo lo scheduler dei processi e quello di input/output (I/O), mentre la seconda no. Ragionando in un'ottica di accrescimento dei sistemi preesistenti, l'adozione della prima via non richiede alcuna modifica dell'applicativo preesistente potendo esso continuare ad appoggiarsi sulle vecchie API; la seconda, invece, a garanzia che i contratti fatti dalle applicazioni multimediali vengano onorati, presuppone che tutte le richieste passino attraverso il middleware, e ciò potrebbe portare alla modifica di molte delle applicazioni scritte in precedenza. Infatti, invece di utilizzare le API preesistenti, offerte dal SO, tali applicazioni dovrebbero essere nuovamente scritte in modo da utilizzare il set di API fornito dal middleware. Un'ultima differenza è che l'utilizzo della seconda alternativa consente di cambiare il sistema operativo sottostante senza che ciò influisca in alcun modo sulle applicazioni (basterà infatti che esistano delle implementazioni del middleware multi-piattaforma). Un interessante confronto fra i pro e i contro di queste scelte architetturali si può trovare in [SHP02].

Finora abbiamo considerato la gestione delle risorse per singolo nodo, vediamo ora invece cosa accade nel distribuito quando si voglia garantire QoS per flussi in transito nella rete.

Per ciò che concerne i sistemi distribuiti, la situazione che si riscontra attualmente non è molto migliore di quella vista finora nel concentrato. Sebbene lo stack OSI (si veda [OSI]) preveda una fase di negoziazione della QoS ai vari livelli, nondimeno l'attuale stack TCP/IP ossia lo standard de facto, non prevede

alcun controllo particolare per assicurare qualità. Al fine di colmare questa lacuna sono state avanzate varie proposte. Per esempio i Servizi Integrati (IntServ, [RFC2210], si veda inoltre il ReSerVation Protocol, RSVP, [RFC2205]) e i Servizi Differenziati (DiffServ, si veda [RFC2475]) proposti da IETF, sono stati pensati con l'intento di assicurare qualità di servizio per il singolo flusso (i primi) e per classi di flussi appartenenti alle stesse classi di servizio (i secondi). Inoltre, per quello che riguarda più specificatamente il dominio multimediale, sono stati proposti due protocolli, e cioè il RealTime Protocol (RTP, si veda [RFC1889]) e il RealTime Streaming Protocol (RTSP, si veda [RFC2326]).

Le ultime soluzioni sono quelle oggi utilizzate per assicurare QoS nella trasmissione di flussi multimediali, vengono sviluppate a livello applicativo. Risulta infatti più semplice agire in questo modo, vista anche la grande inerzia della comunità degli amministratori di rete, restia a ogni nuovo cambiamento che vada a modificare il software attualmente utilizzato.

La nostra decisione per quello che riguarda la gestione delle risorse nel concentrato è stata di optare per la produzione di un middleware a livello applicativo, che offra le funzionalità di cui le applicazioni multimediali necessitano. Sicuramente questa decisione costerà di più, in termini di overhead, rispetto ad una soluzione che privilegi l'utilizzo di SO con supporto realtime. Nondimeno i vantaggi, anche in un'ottica di adattabilità e flessibilità del sistema, ci sembrano evidenti. Inoltre la scelta di appoggiare lo sviluppo dell'applicativo sul middleware fa sì che, qualora i sistemi operativi vengano estesi per supportare le applicazioni RT, non sia necessario variare il codice dell'applicativo. La progettazione del middleware potrà invece beneficiare delle nuove funzionalità offerte dal SO per offrire maggiori garanzie alle applicazioni.

Come visto, la maggior parte degli sforzi compiuti nel distribuito per garantire QoS, sono sviluppati a livello applicativo, data la fattibilità di questa soluzione.

### 1.2.3 Eterogeneità delle piattaforme e dei device

Un altro problema è quello dell'enorme eterogeneità delle piattaforme e dei device coi quali si vuole accedere al contenuto multimediale presente all'interno del sistema distribuito.

Alla luce di quanto visto nel paragrafo precedente, l'adozione di un middleware può facilitare notevolmente le cose, nascondendo agli sviluppatori dell'applicativo i dettagli della piattaforma sottostante. Questa soluzione è valida quando accediamo alla rete tramite device con una capacità computazionale tale

da sostenere il carico computazionale richiesto per l'esecuzione del middleware stesso, ma oggi ciò non si verifica sempre.

È infatti evidente come negli ultimi anni si sia diffusa l'esigenza di accedere ai contenuti multimediali dovunque e da qualsiasi tipo di device. Il problema si pone quando le capacità computazionali del device siano veramente limitate, per esempio quando esse siano insufficienti per ospitare il middleware, di cui abbiamo parlato sopra. Si esaminano ora due possibili alternative architetturali per la soluzione di questo problema.

La prima è quella di progettare un middleware che sia configurabile e modulare, in modo che sia possibile togliere a tempo di esecuzione tutte le funzionalità che non servono per l'esecuzione di un determinato tipo di applicazione, riducendo così al minimo la necessità di memoria. Questa soluzione adottata in [KOF00] e in [BLG01], prevede middleware organizzati a moduli che fanno uso riescono a configurarsi in automatico.

Quando si lavora con macchine che abbiano sufficienti risorse per ospitare il middleware si utilizza questa prima soluzione, sicuramente più elegante e portabile, quando invece ciò non sia possibile, sarà necessario progettare soluzioni ad hoc per ogni singolo device. Questa seconda via prevede l'uso, dal lato cliente, di software proprietari, con relativi formati per la codifica del contenuto multimediale. In questa architettura si frapporrà quindi fra il produttore del contenuto multimediale e il cliente un'entità che chiameremo *proxy*.

Essa svolge una funzione di ponte tra il middleware multimediale e i software proprietari. Conoscerà cioè i dettagli della piattaforma dello specifico device e trasformerà opportunamente il contenuto multimediale nel formato richiesto. Aggiungiamo che, siccome quasi sempre questi device sono di tipo wireless, cioè si collegano alla rete fissa attraverso una connessione senza fili, tipicamente si può incapsulare nel proxy un'ulteriore funzionalità che consiste nella negoziazione del protocollo utilizzato per la trasmissione, in modo da sfruttare al meglio le caratteristiche del device e dell'ambiente circostante (es.: potremmo utilizzare IEEE 802.11 piuttosto che Bluetooth).

Nel progetto di tesi, sebbene il prodotto finale sia stato sviluppato solo per device con una certa capacità computazionale, è stata adottata un'architettura che va nella direzione della seconda presentata qui.

### 1.2.4 Mancanza di supporto per la mobilità dell'utente e del terminale

La maggior parte dei multimedia middleware oggi esistenti raramente include anche un supporto per la mobilità degli utenti e dei terminali.

Le ragioni possono essere diverse, ne indicheremo qui due che riteniamo rilevanti.

In primo luogo solo da pochi anni i device portabili hanno capacità computazionali che rispondono alle caratteristiche richieste dalla presentazione di contenuti multimediali. In secondo luogo la comunità che ha lavorato sulla computazione pervasiva e quella che ha lavorato sul supporto multimediale, non hanno interagito in modo efficace l'una con l'altra.

Questo lavoro di tesi tenta di coniugare queste due aree di ricerca proponendo un multimedia middleware che supporti la mobilità, e sviluppando al di sopra di esso una applicazione per lo streaming di video e audio.

## *1.3 Principi architetturali di un supporto per applicazioni multimediali*

Nella sezione precedente abbiamo fatto una carrellata dei principali problemi che devono essere affrontati durante lo sviluppo delle applicazioni multimediali, e dovrebbe essere abbastanza chiaro, ora, perché riteniamo fondamentale sviluppare un middleware che offra i necessari servizi e faccia da mediatore fra le applicazioni e le piattaforme sottostanti. In questa sezione vengono esaminati i principi architetturali che indicano le linee da seguire per risolvere i problemi visti nella sezione precedente. I principi architetturali proposti derivano da uno studio che è stato fatto su un campione composto da vari middleware multimediali, e che verrà esposto nel capitolo seguente.

### 1.3.1 Modularità

Consiste nel suddividere il middleware in sottoparti (moduli), sia nella direzione verticale che in quella orizzontale. La modularità verticale consiste nel suddividere verticalmente il middleware così che solo un ristretto sottinsieme di moduli sia dipendente dalla piattaforma sottostante. In questo modo un cambiamento di tale piattaforma o il porting verso una nuova piattaforma richiedono il minimo sforzo. La modularità orizzontale invece suggerisce di suddividere le funzionalità offerte dal middleware, raggruppando in uno stesso modulo funzionalità simili. Sarà così possibile caricare solo alcuni moduli e non tutto il supporto.

Da ultimo strutturare un sistema in moduli assicura incapsulamento e, se le cose sono state sviluppate in modo corretto, sarà possibile modificare il supporto a runtime, semplicemente sostituendo vecchie versioni di un certo modulo con le nuove. Naturalmente questo principio è oggi largamente adottato, basti pensare

che tutti i moderni sistemi operativi sono basati su un'architettura di tipo micro-kernel.

### 1.3.2 Flessibilità

Consiste nello strutturare il supporto in modo che possa essere riconfigurato a seconda delle necessità e del contesto esterno. Per esempio, come si diceva sopra, l'utilizzo di un middleware opportunamente riconfigurato (in modo da includere solo le funzionalità realmente utili) potrebbe essere una soluzione ottimale quando utilizziamo un palmare dotato di una discreta capacità computazionale.

Naturalmente l'adozione di questo principio presuppone l'avere seguito il principio di modularità durante la strutturazione del middleware e comporta un costo, sia in termini di progetto di un'architettura che applichi questo principio, sia in termini di overhead dovuto ai tempi di inizializzazione e riconfigurazione del supporto stesso.

### 1.3.3 Scalabilità

È la capacità del sistema di rimanere utilizzabile all'aumentare delle dimensioni dello stesso. È un principio architetturale di grande importanza quando si progettino sistemi che devono servire una utenza molto ampia. I sistemi multimediali, anche per l’alto fabbisogno di risorse, non sono solitamente assai scalabili.

### 1.3.4 Prevedibilità

Consiste nel fare in modo che i contratti stipulati fra il supporto e l'applicativo per richiedere l'uso delle risorse, vengano rispettati.

Questo principio è alla base di tutti i meccanismi di tipo **pro-attivo**. Essi prevedono la prenotazione e occupazione preventiva delle risorse, negoziando inoltre col sistema le variazioni della disponibilità delle stesse che l'applicazione può sopportare. Tutto ciò avviene **staticamente**, cioè prima dell'inizio dell'erogazione, attraverso un contratto stipulato fra l'applicazione e il sistema. Tale principio inoltre comporta che il sistema si occupi della gestione della qualità di servizio su più livelli, anche se le soluzioni esistenti sono perlopiù sviluppate al livello applicativo. Un supporto che risponda a questo principio deve essere progettato in modo che possa gestire non un unico tipo di risorsa, ma possibilmente tutte le risorse attraversate dal flusso.

Naturalmente, come detto sopra, non sarà possibile realizzare un sistema che sia completamente prevedibile, nel quale cioè si abbia la certezza che ogni

risorsa richiesta e prenotata da una certa applicazione sarà disponibile per tutta la durata dell'erogazione del contenuto multimediale.

### 1.3.5 Minima Intrusione

Questo principio stabilisce che il monitoraggio, che è elemento fondamentale di ogni approccio reattivo, sia il meno intrusivo possibile.

È noto che ogni volta che si opera una misurazione si modifica lo stato del sistema in misura. Nel caso del monitoraggio di un sistema distribuito, tale problema è aggravato dal fatto che il monitor utilizza, esso stesso, parte delle risorse che sta monitorando.

Ne segue che l'attività di monitoraggio, se attuata quando la situazione sta peggiorando, può peggiorare ulteriormente le cose.

### 1.3.6 Adattabilità

È un principio di fondamentale importanza quando si vogliano implementare approcci di tipo **reattivo**. Il solo utilizzo di un approccio di tipo pro-attivo, infatti, non garantisce che, una volta stipulato un contratto, il sistema sia in grado di onorarlo. Infatti ci possono essere grandi fluttuazioni nella disponibilità di risorse del sistema distribuito, si deve affiancare perciò a questo approccio un secondo, cioè quello reattivo. In questo caso, con azioni svolte **dinamicamente**, il supporto controlla come stia procedendo la trasmissione monitorando il sistema e, in caso di degrado delle risorse disponibili, richiede una nuova fase di negoziazione della QoS.

In pratica bisognerebbe strutturare il supporto in modo che sia possibile, dopo aver rilevato un degrado delle risorse, rinegoziarle l’occupazione delle risorse con l'applicazione, e adattare il servizio alla nuova situazione che si è venuta a verificare.

### 1.3.7 Visibilità

**Q**uesto principio stabilisce che, nell'ambito delle applicazioni multimediali, il supporto deve offrire mezzi per raccogliere informazioni circa l'uso corrente delle risorse e la gestione delle stesse. Questo principio, che è facilmente comprensibile, apre però un'infinità di possibilità per lo sviluppatore del supporto. In realtà il problema è fissare il livello di trasparenza e l'espressività che si vuole offrire allo sviluppatore dell'applicativo. Nel seguito della trattazione verranno esaminati diversi meccanismi attraverso i quali si può concretizzare questo principio.

### 1.3.8 Consapevolezza del contesto (Context-awareness)

Principio architetturale secondo il quale l'architettura del sistema è strutturata in modo che esso sia in grado di assumere consapevolezza delle piattaforme computazionali con le quali si trova ad interagire, e dei servizi che esse offrono.

È necessario cioè che il supporto conosca le configurazioni delle piattaforme con le quali l'utente opera, in modo da adattare adeguatamente il servizio, e offra la dei servizi per il discovery dei servizi, che sono di fondamentale importanza per i terminali mobili che, giungendo in una nuova località non sanno a priori quali siano i servizi disponibili in tale localotà.

### 1.3.9 Consapevolezza della località (Locality-awareness)

**C**onsiste nel fatto che il supporto offra informazioni di località fisica all'applicativo sviluppato al di sopra di esso. È auspicabile conformare il supporto multimediale a tale principio poiché le applicazioni multimediali richiedono la trasmissione di ingenti quantità di dati.

Se questo principio è stato rispettato infatti il supporto può scegliere il servitore più vicino in base alla nozione di località introdotta.

## *1.4 Conclusione*

In questo primo capitolo abbiamo introdotto alcune nozioni basilari, definendo il concetto di flusso e alcuni parametri ad esso connessi. Abbiamo poi presentato le principali problematiche che si presentano oggi a chi voglia sviluppare applicazioni multimediali. Nella stessa sezione abbiamo indicato le principali evoluzioni richieste all'architettura dei sistemi distribuiti per risolvere tali problemi, con l'intento di dimostrare come l'introduzione di uno strato di middleware sia di fondamentale importanza per trattare questa tipologia di applicazioni. Infine, l'ultima sezione è stata dedicata alla trattazione dei principi che dovrebbero guidare la costruzione di tale middleware.

Il secondo capitolo sarà dedicato all'analisi di lavori di ricerca che si occupano del progetto di piattaforme middleware rispondenti ai principi sopra esposti. Verranno dapprima considerati i principali meccanismi inclusi in tali piattaforme, poi si analizzeranno più nel dettaglio sette architetture, con l'intento di mettere in evidenza i loro pregi e le loro carenze.

CAPITOLO 2

# 2 Stato dell'arte dei multimedia middleware

Dall'analisi delle sei piattaforme considerate sono stati estrapolati i seguenti meccanismi di base che verranno divisi in tre categorie.

## *2.1 Meccanismi di base implementati da i multimedia middleware*

### 2.1.1 Stesura e trattamento delle specifiche

Per dare la possibilità agli sviluppatori dell'applicativo di specificare i requisiti della propria applicazione molte delle architetture considerate pongono particolare attenzione ad introdurre meccanismi per la specifica di qualità di servizio e dipendenze fra le entità distribuite che andranno a realizzare l'applicazione.

#### 2.1.1.1 Specifica della qualità di servizio

Si vuole da subito sottolineare come la gestione della qualità di servizio dell'applicazione andrebbe gestita in modo dichiarativo piuttosto che imperativo. Si dichiarano cioè le necessità dell'applicazione lasciando al supporto il compito di tradurle in vere e proprie richieste di risorse, sollevando quindi lo sviluppatore da questo compito. Vengono solitamente definiti tre livelli di specifica della QoS: il livello utente, il livello applicativo e il livello delle risorse.

- **Livello utente**: l'utente finale può specificare la qualità di servizio; ad esempio, si può immaginare uno scenario nel quale un servizio è offerto con diversi standards qualitativi e, in base alla qualità, si definisce il costo del servizio.
- **Livello applicativo**: i requisiti, in questo caso, vengono fissati dagli sviluppatori dell'applicativo. Questa tematica è stata particolarmente approfondita nell'intento di fornire adeguate astrazioni agli sviluppatori, in modo da sollevarli dalla progettazione di quella parte non funzionale di applicativo che si occupa della gestione della qualità del servizio.
- **Livello delle risorse**: a questo livello le richieste applicative, troppo astratte per essere utilizzate direttamente nella gestione delle risorse del sistema, vengono convertite in richieste di risorse che possono essere presentate direttamente al sistema; ad esempio la richiesta fatta a livello applicativo per

un filmato, che ne indichi il formato, la risoluzione e il frame rate, verrà tradotta in una richiesta di memoria, cicli di CPU e banda passante.

#### *2.1.1.1.1 Strumenti per il supporto del processo di specifica*

Alcuni sistemi provvedono lo sviluppatore di opportuni strumenti per facilitare il processo di stesura delle specifiche di QoS; ad esempio alcuni middleware mettono a disposizione dello sviluppatore un tool visuale per specificare la QoS desiderata, e la connessione fra i vari componenti coinvolti nel delivery del servizio (si veda [NAK01]). In altri sistemi invece è possibile specificare separatamente QoS desiderata e logica funzionale del sistema, ed il middleware mette a disposizione una sorta di compilatore che unisce queste due specifiche, a formare un unico programma ( si veda [SCD01]).

Una metrica considerata nella successiva analisi, è stata quindi verificare se la relativa piattaforma divida la specifica dei requisiti di QoS dalla logica funzionale dell'applicazione.

#### *2.1.1.1.2 Traduzione/Compilazione delle specifiche*

Come abbiamo visto è possibile suddividere le specifiche di QoS in tre livelli; è perciò fondamentale implementare a livello di middleware opportuni meccanismi per la traduzione delle specifiche da un livello ad un altro. Inoltre, in alcuni middleware (vedi [NAK01]), le dichiarazioni fatte ai tre livelli vengono fuse insieme e compilate da un vero e proprio compilatore, in modo da poter essere utilizzate come codice eseguibile dalla piattaforma middleware distribuita. Tale "codice" viene distribuito alla partenza dell'applicazione e serve per la configurazione dei componenti distribuiti che devono supportare la applicazione stessa. Nella successiva analisi si considererà:

(a) se il middleware automatizza il processo di traduzione dei requisiti;
(b) se viene effettuata alcuna verifica di consistenza; ad esempio è importante verificare che i vincoli di QoS fra due componenti successivi in un grafo di servizio vengano rispettati.

### 2.1.1.2 Specifica delle entità che compongono l'applicazione

Per rispondere ai principi di modularità e flessibilità, esposti nel capitolo precedente, bisogna che il middleware offra la possibilità di specificare come le varie entità distribuite che compongono l'applicazione siano organizzate e interagiscano fra loro. Ciò è molto importante per il supporto alla configurazione iniziale del sistema e alla sua riorganizzazione a tempo di esecuzione. Queste

specifiche sono poi tradotte per provvedere supporto a tempo di esecuzione attraverso opportuni oggetti di configurazione come i *Bindings* (vedi [BLG99]), o i *Component Graphs* (vedi [BLG01]).

#### *2.1.1.2.1 Supporto alla configurazione*

Una volta specificate le dipendenze delle varie entità che compongono una applicazione è però fondamentale che il middleware implementi meccanismi per la configurazione iniziale del sistema e la sua riconfigurazione dinamica. In questa ottica molti middleware realizzano meccanismi per lo scaricamento di code on demand (COD, si veda anche il capitolo seguente), e la configurazione dinamica dei componenti distribuiti, come ad esempio i component repository. Nella successiva analisi si verificherà:

- se il middleware offre supporto allo scaricamento dinamico del codice;
- se il middleware offre la possibilità di modificare il grafo di servizio a runtime.

#### *2.1.1.2.2 Supporto alla riflessione*

Con riflessione si intende la capacità di introspezione e riconfigurazione del middleware. La riflessione è uno strumento da tempo utilizzato con successo nel design dei linguaggi di programmazione. L'idea avanzata da alcuni (vedi ad esempio [BLG01]) è quella di utilizzare questo strumento anche per la configurazione e riconfigurazione dinamica del supporto. Ad esempio se il supporto si accorge che una certa richiesta arriva da un terminale con basse capacità computazionali, si può riconfigurare in modo da frapporre opportuni filtri per adattare i flussi multimediali, oppure, se vengono rilevate particolari condizioni della rete, similmente il supporto si può riconfigurare richiedendo l'utilizzo di protocolli diversi, che meglio rispondano alla nuova situazione verificatasi. Ovviamente perché tutto questo sia possibile devono essere presenti, all'interno del sistema, dei meta-dati che descrivano le varie entità facenti parte del sistema stesso.

### 2.1.2 Gestione delle risorse e dei dati

I meccanismi di gestione delle risorse sono di fondamentale importanza nella realizzazione di software multimediali. In particolare, nella loro progettazione, bisogna tenere presenti i principi di prevedibilità, minima intrusione, adattabilità e visibilità introdotti al capitolo 1. Oltre alla gestione delle

risorse fisiche si considera anche, in questo sottopunto, come avvenga la gestione dei dati all'interno del sistema.

#### 2.1.2.1 Astrazioni delle risorse e dei dati

Nella gestione delle risorse un aspetto molto importante, che si rifà al principio della visibilità, quello di decidere quali astrazioni delle risorse del sistema il middleware debba offrire al livello applicativo soprastante. Se poi si considera la particolare natura delle applicazioni multimediali, si capisce ben presto che si ha a che fare con un immenso numero di risorse, sia hardware che software.

Altrettanto importante è stabilire quale astrazione offrire allo sviluppatore dei dati presenti all'interno del sistema. Come già detto l'entità prima che si considera è il singolo flusso multimediale. Molte architetture multimediali offrono poi meccanismi per trattare aggregazioni di più flussi (sessioni), inoltre in alcuni middleware, basati su componenti, al livello più alto lo sviluppatore dell'applicativo si può unicamente concentrare sul collegamento dei diversi componenti, demandando al livello sottostante la gestione dei flussi e della sessione.

#### 2.1.2.2 Meccanismi per la gestione delle risorse

Innanzitutto è fondamentale che il sistema disponga di meccanismi per la **prenotazione delle risorse**, inoltre, come già accennato nel primo capitolo, si dovranno anche implementare servizi per il **monitoring** delle risorse stesse. Insieme al monitoring vengono poi solitamente implementati meccanismi per l'**adattamento** del servizio. Quando cioè il sistema si rende conto che le cose non vanno più tanto bene, cerca di adattare il servizio.

Questi servizi fondamentali sono implementati da quasi tutte le architetture, alcune architetture aggiungono però altri meccanismi, ad esempio implementano protocolli per il **probing** in modo da testare le condizioni del sistema distribuito, scegliendo la configurazione migliore per una certa applicazione, prima di farla partire.

Un interessante parametro di cui si terrà conto nell'analisi delle piattaforme è l'ambito d'azione (scope) di questi meccanismi che possono agire **localmente** oppure **globalmente**.

### 2.1.3 Gestione dell'eterogeneità e della mobilità

Si considera da ultimo come le varie architetture gestiscano l'eterogeneità di device e piattaforme computazionali, e la mobilità dei terminali e degli utenti.

Questi meccanismi si rifanno ai principi di location-awareness e context-awareness. Si nota che, per quello che riguarda il multimedia middleware, mentre i meccanismi considerati finora sono generalmente implementati in molte architetture, per quello che riguarda quest'ultima categoria di meccanismi essi sono sviluppati solo in un numero esiguo di architetture, inoltre le soluzioni finora sviluppate sono soluzioni piuttosto "ad hoc", e non esiste ancora un chiaro framework all'interno del quale muoversi. La motivazione principale è che il problema di supportare la mobilità dell'utente, per quello che riguarda il delivery di materiale multimediale, è un problema piuttosto recente, e molto lavoro deve essere ancora fatto.

#### 2.1.3.1 Application hand-off

Quando si abbia a che fare con movimenti dell'utente o del terminale, bisogna che l'applicazione sia organizzata in modo da poter assistere tali movimenti. In entrambi i casi, cioè, devono essere implementati meccanismi che si occupino della realizzazione dell'hand-off della sessione aperta. Nel caso di movimento dell'utente la sessione verrà spostata da terminale ad un altro, nel caso invece di spostamento del terminale si vuole solitamente spostare la sessione, in modo che l'ultimo dei componenti della catena di servizio che va dal client al server sia collocato nella stessa località nella quale si trova il terminale.

#### 2.1.3.2 Tailoring

Da ultimo si considerano i meccanismi per il tailoring del servizio. Quando un servizio deve essere fruito in un ambiente altamente eterogeneo non sono solitamente stabilibili a priori le caratteristiche di tutte le periferiche che vorranno accedere al servizio stesso. Questo è tanto più vero se si pensa che, per quanto riguarda il multimedia streaming, si utilizzano poi vari formati e protocolli per lo scambio dei dati. È quindi importante prevedere la realizzazione di meccanismi per la conversione/rimaneggiamento dei dati e l'eventuale utilizzo di protocolli diversi, in modo da permettere a nuovi device di partecipare al servizio.

Tale servizio può ad esempio effettuare il downscaling di un certo filmato che non possa essere direttamente visualizzato da un device a bassa capacità computazionale.

## *2.2 Architetture a confronto*

### 2.2.1 GOPI

GOPI (Generic Object Platform Infrastructure) è una piattaforma sviluppata all'Università di Lancaster. L'aspetto principale del progetto è il supporto alla gestione di flussi multimediali e alla specifica e gestione della QoS, attraverso la progettazione di un middleware orientato agli oggetti.

Il modello computazionale cui la progettazione della piattaforma si ispira è quello definito dal modello di riferimento ISO per l'Open Distributed Processing (Reference Model for Open Distributed Processing, RM-ODP [BLG97]). Perciò, seguendo tale modello, vengono definite come *"first class" entities*, tutti i tipi di interazione. In particolare in questo progetto vengono definiti tre tipi di interazione: le operazioni con invocazioni di tipo request/reply, gli stream, e gli eventi. Inoltre la proposta di GOPI è di integrare tutti questi tipi in un'unica piattaforma, che non richieda l'utilizzo di estensioni, come accade per esempio in CORBA, dove è stato proposto, come estensione, l'A/V Streams Service per semplificare la coordinazione e l'interazione fra i diversi tipi e facilitare la programmazione. Ciò significa che quando l'IDL (Interface Definition Language) interface viene compilata per tutti gli eventi e le interazioni fra gli stream, gli skeleton vengono automaticamente generati. Interface Definition Language è un linguaggio astratto per la descrizione di operazioni remote, che deve consentire l'identificazione del servizio e la definizioni dei dati di ingresso e uscita. In questo caso ci si riferisce ad un IDL ispirato a quello utilizzato in CORBA. Il modello computazionale adottato in GOPI offre poi la possibilità di effettuare *explicit binding* (binding espliciti) fra le entità, dando allo sviluppatore maggiore visibilità dell'ambiente distribuito.

Un *binding* è un oggetto che incapsula una lista di endpoint e una specifica di qualità di servizio per il binding stesso; è perciò un meccanismo per la specifica delle entità e delle loro interazioni, per come sono stati sopra definiti questi concetti. Questa astrazione è un mezzo per isolare le risorse necessarie ad un certo binding, inoltre i binding possono essere organizzati in topologie più complesse, come pipeline e alberi. Questo mezzo serve per attuare una gestione statica delle varie entità e non può essere utilizzato per riconfigurazioni a tempo di esecuzione di tutta la struttura. Infatti, definito un binding, la struttura distribuita del sistema è definita e non è possibile passare ad esempio da un certo binding ad un altro dinamicamente mentre è possibile adattare il servizio. Come già accennato, infatti, il binding rappresenta un mezzo per l'incapsulamento e la

gestione delle risorse; ad esempio un binding di gruppo potrebbe generare un evento quando la QoS degrada e potrebbe provvedere operazioni per aggiungere o rimuovere, a tempo di esecuzione, dei partecipanti dipendentemente dalle condizioni del sistema distribuito. Alcuni dei limiti per la riconfigurazione dinamica del sistema sono stati superati da OpenORB, un altro middleware, basato su componenti, sviluppato dallo stesso gruppo di ricerca.

Il supporto alla QoS permette che i vari punti di interazione possano essere annotati con delle specifiche di QoS che dichiarano come debba essere adattato il punto di interazione, considerando i parametri caratteristici di tale interazione (es. latenza, periodicità, ecc.) offrendo l'astrazione di "schema". Ricollegandoci ai concetti sopra introdotti, lo *schema* può essere considerato come una specifica fatta a livello applicativo; una parte del middleware è perciò dedicata alla traslazione di queste richieste dal livello applicativo a quello delle risorse. Vi è poi il supporto per la prenotazione, monitoring e adattamento delle risorse.

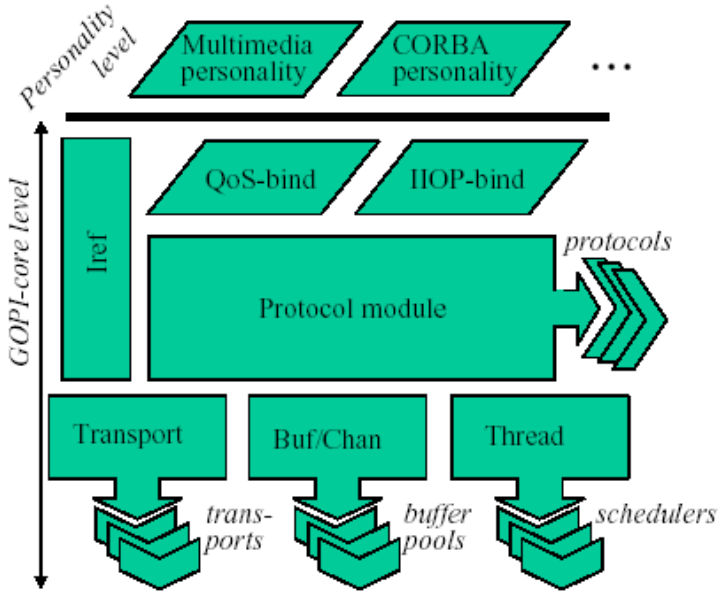


**Figura 2.1**: architettura di GOPI.

L'architettura del sistema, che viene mostrata in figura 2.1, è ispirata a quella dei sistemi operativi a micro-kernel. Riferendosi alla figura, l'idea è quella di offrire nel core level le funzionalità di base, mentre nel personality level le funzionalità specifiche di un certo dominio applicativo. Ogni livello è poi internamente strutturato in modo modulare. Esistono moduli per la gestione dei thread (scheduling), per il management dei buffer, e per la gestione dei protocolli di trasporto, in particolare viene implementato uno strato in modo da astrarre i diversi protocolli di trasporto.

Nella realizzazione del core level una scelta significativa fatta in GOPI è stata l'utilizzo di *plug-in*. Viene definita plug-in ognuna delle entità che si rifanno ad una interfaccia d'uso comune, implementando però internamente diversi comportamenti; ad esempio i vari scheduler utilizzati dal thread module sono dei plug-in. La gestione della QoS è strettamente collegata alla scrittura dei diversi plug-in. Per ogni plug-in implementato vengono infatti definite, attraverso un *QoS schema*, le varie operazioni che sarà necessario eseguire per adattare il servizio, il tutto dipendentemente dai parametri dello specifico plug-in. Ogni plug-in è quindi responsabile dell'interpretazione e della traslazione delle specifiche di QoS. Il lato negativo di questa soluzione è che lo sviluppatore del plug-in diventa perciò responsabile di queste due operazioni. Ad esempio lo sviluppatore di un nuovo Protocol plug-in per il Protocol module, non deve solo definire gli aspetti funzionali del protocollo che progetta, ma deve anche occuparsi della traslazione delle specifiche di QoS, della gestione degli eventi, e di tutte quelle operazioni richieste dall'utilizzo dei binding sopra introdotti, che riguardano la gestione della QoS e non la progettazioni in sé del protocollo.

Per quanto concerne il personality level esso offre diversi modelli di programmazione; ad esempio il multimedia personality level estende il CORBA IDL offrendo, come tipi primitivi, gli eventi, gli stream, i QoS-group e astrazioni per la composizione e la gestione degli oggetti multimediali.

### 2.2.2 TAO

TAO è l'implementazione di un ORB (Obejct Request Broker) scritto in C++ compatibile con la maggior parte dei servizi e delle caratteristiche definite nella specifica CORBA 2.6 (Common Obejct Request Broker Architecture). TAO è costruito al di sopra di ACE (Adaptive Communication Environment), un framework orientato agli oggetti, che implementa molti pattern per la comunicazione concorrente di processi distribuiti.

Entrambi i sistemi sono stati sviluppati nella sede di St. Louis dell'Università di Washington, in particolare, per ciò che riguarda questa tesi, siamo interessati al servizio di supporto allo streaming audio/video incluso in TAO che implementa lo standard definito nella specifica promulgata da OMG (Object Management Group) "Control and Management of Audio/Video Stream Specification", servizio che è stato poi integrato e testato insieme al QuO framework (vedi [KAD01]).

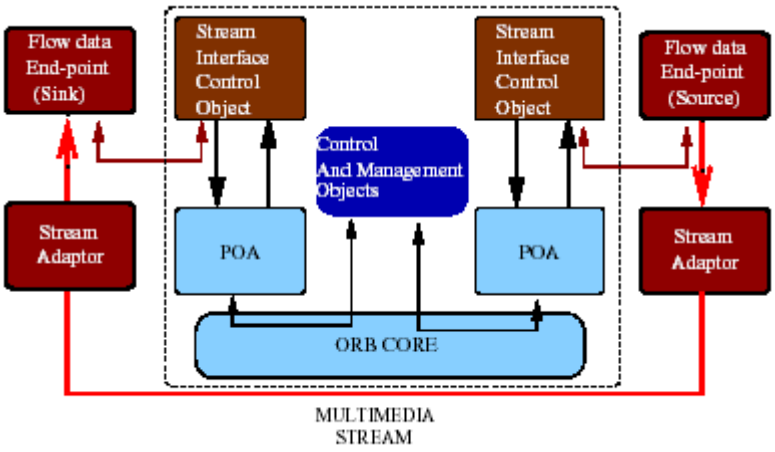


Figura 2.2: architettura per gestione flussi audio e video proposta da OMG

Il progetto del servizio di audio/video streaming implementa gli oggetti definiti dalla specifica OMG. L'architettura OMG, per quello che riguarda la gestione dei flussi (OMG Sreams-Architecture), viene mostrata in figura 2.2. E' importante notare che, mentre le operazioni di controllo e segnalazione sono implementate al di sopra dell'ORB, per il flusso dati ci si serve di un percorso alternativo per assicurare la necessaria modularità e flessibilità. Infatti il trasferimento dei flussi multimediali può essere realizzato utilizzando il protocollo di basso livello più adatto (ad esempio il protocollo ATM oppure UDT, TCP o RTP). Questa soluzione è simile a quella vista per GOPI quando si è introdotto il concetto di plug-in per il protocol-module. Per rispondere al principio di prevedibilità nella distribuzione del contenuto multimediale viene utilizzato il Real-time event service incluso in TAO, in modo da implementare un'architettura di tipo produttore/consumatore. In questo sistema non esiste supporto per la riconfigurazione e la gestione degli oggetti a tempo di esecuzione, cioè, come visto in GOPI, una volta definita una certa topologia per il delivery dei flussi essa viene mantenuta. Anche il supporto per lo scaricamento del codice non è presente, vengono invece definite astrazioni di basso livello per stabilire e controllare i flussi e supportati diversi tipi di sorgenti (source) e pozzi (sink).

Il QuO framework è una piattaforma costituita da oggetti distribuiti, concepita alla BBN (si veda [BBN]) per il supporto di applicazioni distribuite con necessità di garanzia per ciò che riguarda la qualità di servizio. In particolarte si possono definire i requisiti di QoS, quali entità debbano essere monitorate e quali adattamenti debbano essere attuati in risposta a fluttuazione delle risorse. L'aspetto innovativo di questo framework è che attraverso l'uso del QDL (QoS Description Language), istanza degli Aspect Language (vedi [LOJ98]), possono

essere distinti in fase di programmazione gli aspetti funzionali da quelli non funzionali (gestione della qualità di servizio) nella progettazione e scrittura dell'applicativo. Infatti i cosiddetti "sviluppatori di QoS" (QoS Develoters) sono incaricati unicamente della definiziomne delle specifiche di QoS e degli oggetti che interagiranno in modo da monitorare ed adattare il sistema a tempo di esecuzione; altre persone, invece, definiscono la logica dell'applicativo. Si nota che, in questo modo, il processo di programmazione, tenendo separati questi due aspetti, permette anche il riutilizzo del QuO framerwork in diversi domini applicativi. Come accade in generale nello sviluppo basato sugli Aspect Language il generatore di codice prenderà poi in ingresso i codici che, come visto sopra, definiscono l'aspetto funzionale e non funzionale dell'applicativo unendoli insieme.

### 2.2.3 MAQS

MAQS (Management Architecture for Qualità of Service) è un progetto sviluppato dalla "Fakultat fur Elektrotecnik und Informatik" di Berlino ed ha come fine lo sviluppo di un framework per il supporto alla QoS in sistemi costituiti da oggetti distribuiti, con particolare attenzione alla possibilità di riutilizzo del framework in ambiti diversi.

Questo lavoro è strettamente connesso al QuO sopra citato. La principale differenza tra i due framework è che QuO si concentra sulla suddivisione dei diversi aspetti di progettazione, mentre MAQS studia le necessarie estensioni da introdurre nella piattaforma CORBA, per fornire un supporto che garantisca QoS. Come in QuO il progetto si basa sull'idea di dirigere i diversi aspetti di progettazione, in particolare ci si riferisce al paradigma della programmazione orientata agli aspetti (Aspect Oriented Programming). Secondo tale paradigma la soluzione di un problema attraverso il solo utilizzo di oggetti non è possibile, viene perciò introdotto il concetto di *aspetto*. Un *aspetto* può toccare diversi oggetti che partecipano all'architettura e tipicamente riflette caratteristiche non funzionali del sistema. Come già visto sopra, sarà poi possibile, attraverso l'uso degli Aspect Language, modellare ed esprimere le varie proprietà di un *aspetto* (nel caso in questione le proprietà concernenti la qualità di servizio) e unire tali specifiche con quelle degli oggetti distribuiti, attraverso l'utilizzo di un compilatore che agisce come un compilatore di IDL per gli *aspetti*. Questo gruppo di ricerca propone l'introduzione di: (a) QoS Interface Definition Language (QIDL), che è un'estensione del CORBA IDL ed rappresenta l'Aspect Language utilizzato, (b) l'estensione del compilatore di IDL, in modo che compili

insieme alle interfacce standard anche gli *aspetti,* (c) due diversi livelli di astrazione, in modo che gli sviluppatori della QoS (già definiti nel punto precedente) possano progettare sia a livello applicativo, al di sopra dell'ORB, sia a livello di trasporto, in modo da garantire la necessaria flessibilità per la gestione della qualità di servizio.

Lo stesso gruppo di ricerca ha poi, da poco, intrapreso lo sviluppo di un Middleware, basato su componenti, chiamato QCCS (Quality Controlled Componenti-Based Software), che si occuperà delle medesime problematiche per ciò che riguarda la programmazione basata su componenti, riguardo al quale non è stato possibile, per il momento, trovare ulteriore documentazione.

### 2.2.4 $2K^{Q+}$

Questo Middleware è stato sviluppato nell'Università dell'Illinois con sede ad Urbana-Champaign ed in figura 2.3 è rappresentata l'architettura logica del sistema. Si presenta come uno dei più completi multimedia MW e si concentra sulle problematiche di specifica e gestione della qualità di servizio per applicazioni sviluppate sia al di sopra delle reti fisse che al di sopra delle reti mobili. Questo progetto vuole cioè offrire un frame-work per la gestione unificata della QoS nell'ambito di applicazioni distribuite, basate su componenti.

L'architettura può essere divisa in quattro parti: (a) l'ambiente di programmazione della qualità di servizio, (b) il compilatore delle specifiche della QoS (Q-Compiler), (c) il middleware di supporto a tempo di esecuzione ($2K^{Q+}$) e (d) l'estensione per il multimedia streaming verso device con limitate capacità computazionali.

*L'ambiente di programmazione* per la QoS offre allo sviluppatore dell'applicativo un mezzo per fissare le specifiche di QoS. La soluzione proposta consiste nell'introduzione di un tagged-language ispirato all'XML (E**x**tensible Markup Language) come linguaggio per la specifica della QoS. Inoltre è stato anche implementato un tool visuale in modo da facilitare il processo di specifica della QoS. L'ambiente per la specifica della QoS è uniforme, gli sviluppatori possono cioè servirsi dello stesso insieme di specifiche di QoS per fissare i requisiti di applicazione appartenenti a diversi ambiti. Utilizzando questo strumento lo sviluppatore specifica la QoS desiderata a livello applicativo e può anche fissare le dipendenze tra i vari componenti che interagendo realizzeranno l'applicazione vera e propria (cioè può specificare un grafo coi componenti che contribuiranno al delivery del servizio). Lo sviluppatore è inoltre incaricato di dichiarare le specifiche possibili a livello utente e le necessarie traduzioni per

trasformare tali specifiche in specifiche a livello applicativo. Riassumendo viene richiesta allo sviluppatore l'introduzione di quattro tipi di informazioni: la descrizione dell'applicazione, la specifica del grafo delle dipendenze dei componenti formanti l'applicazione, la descrizione dei componenti e la traslazione dalle specifiche a livello utente alle specifiche a livello applicativo.

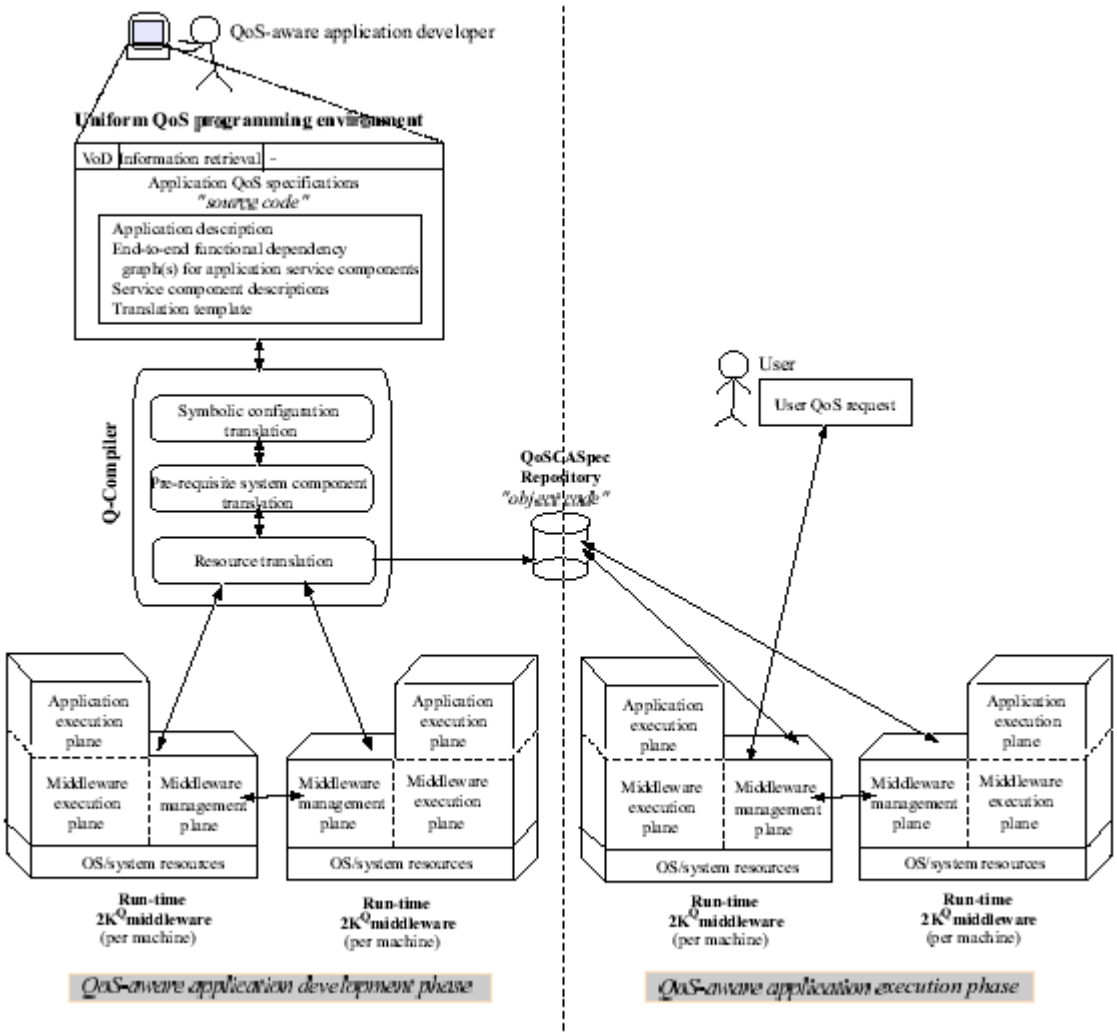


**Figura 2.3**: architettura logica di $2K^{Q+}$

Il *Q-Compiler* è un'entità fondamentale per la gestione unificata della qualità di servizio. Esso compila le specifiche di QoS producendo una sorta di codice eseguibile che viene distribuito e può essere utilizzato a tempo di esecuzione dal Middleware per la gestione della QoS. Questo compilatore realizza cioè la traduzione dalle specifiche a livello applicativo alle specifiche a livello di risorse. Un aspetto interessante di questa architettura è che tale processo è dinamico nel senso che vengono utilizzati protocolli distribuiti per testare le capacità disponibili del sistema e conseguentemente utilizzare tali informazioni

nel processo di traduzione. Il processo di compilazione produce in uscita per ogni specifica fatta a livello applicativo diverse specifiche fatte a livello di risorse. Ciascuna specifica, chiamata QoSCASpec (Component Based Application Specification) contiene una configurazione del sistema a seconda delle disponibilità delle risorse. Tutte queste specifiche sono poi salvate in un repository, pronte all'uso.

Il $2K^Q$ Middleware è la piattaforma basata su componenti e riconfigurabile a tempo di esecuzione, che viene utilizzata per supportare le applicazioni multimediali. L'obiettivo di questa parte del sistema è quello di offrire alle applicazioni soprastanti un comportamento del tipo: "quello che ti serve è quello che ottieni" (What you need is what you get), rispondendo ai requisiti di flessibilità e di modularità. L'idea alla base di $2K^Q$ è di dividere i componenti specifici dell'applicazione dai componenti di MW, in modo da potere riutilizzare i componenti di MW in diversi ambiti. Vengono perciò definiti due sottosistemi per l'esecuzione: uno per MW e uno per l'applicativo, chiamati rispettivamente MW plane e application plane. Nel piano di MW vengono eseguiti i broker e i monitor per le varie risorse (CPU, banda, ecc.), mentre nell'altro i componenti costituenti l'applicazione vera e propria. Le dipendenze tra i vari componenti sono incapsulate, come già accenato, nelle QoSCAAspec. Un terzo sottosistema si occupa poi (a) della gestione della configurazione del sistema distribuito, offrendo servizi per il discovery e la registrazione dei componenti, (b) della realizzazione dei protocolli per il coordinamento delle entità preposte a monitoring e adattamento del sistema distribuito.

E' stata poi anche implementata un'estensione della piattaforma in modo da supportare applicazioni che possano essere distribuite in reti mobili e ambienti eterogenei. In particolare è stata inclusa la *gestione dell'hand-off* a livello applicativo per i movimenti del terminale.

### 2.2.5 Open ORB2

Open ORB2 è una piattaforma a componenti, realizzata nell'università di Lancaster (vedi [BLG01]), basata su precedenti lavori del medesimo gruppo di ricerca, come il sopra citato GOPI.

L'architettura si concentra principalmente sugli aspetti di configurazione e riconfigurazione del sistema a tempo di esecuzione, sia in risposta a fluttuazioni delle risorse, sia per effettuare i dovuti rimaneggiamenti della piattaforma MW in dipendenza dal profilo del cliente finale. Ad esempio, se il cliente è un PDA (Portable Digital Assistant), solo una configurazione minimale del MW può

essere ivi installata. Un'importante caratteristica di questa architettura è la definizione di un modello computazionale che introduce un meta-livello per supportare meccanismi di riflessione del MW come definiti in 2.1.1.2.2 (la figura 2.4 visualizza questa situazione). Questo modello inoltre può essere applicato in maniera ricorsiva perciò, oltre al meta-livello, possono essere definiti un meta-meta-livello, ecc. Ogni meta-livello, poi, viene utilizzato come strumento di accesso a diversi aspetti dei componenti costituenti il sistema.

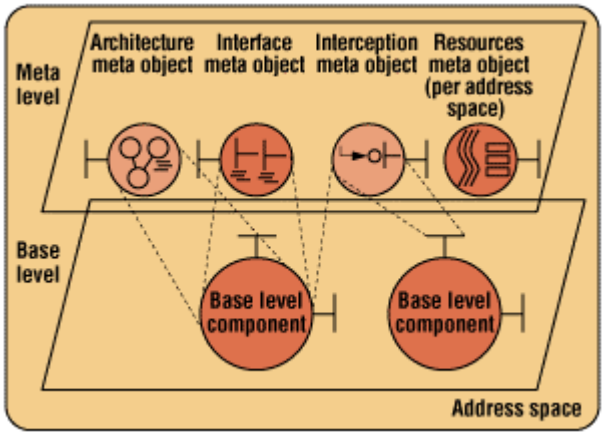


**Figura 2.3**: architettura di OpenORB2.

L'*interface Meta Object* offre l'accesso alla rappresentazione esterna del componente, agendo in modo assai simile al supporto per l'introspezione offerto dalle API per la riflessione in Java. L'*Architectural Meta Object* fornisce l'accesso al grafo dei componenti e ai vincoli che lo governano, offrendo uno strumento per la specifica delle dipendenze dei componenti stessi. Il terzo oggetto, l'*Interception Meta Object*, prevede la possibilità di inserire dinamicamente alle varie interfacce, moduli di pre/post elaborazione. Questo meccanismo viene, ad esempio, utilizzato per aggiungere dinamicamente moduli per la gestione della QoS. Viene infine introdotto il *Resource Meta Object* che è impiegato per la gestione delle risorse del sistema.

L'implementazione di questo sistema è sviluppata come un'estensione dell'architettura COM di Microsoft in modo da supportare il meta-livello cui abbiamo accennato sopra. L'aspetto chiave di tutta l'architettura è la gestione dinamica delle qualità di servizio. L'utilizzo della riflessione e del meta-livello introdotto assicurano flessibilità e riconfigurabilità a tempo di esecuzione per il supporto al monitoring ed all'adattamento del servizio. Possono essere definite diverse politiche di gestione a livello applicativo tramite l'utilizzo di un linguaggio di script e di un interprete. Attraverso l'uso di questo formalismo

possono essere dipinti degli automi a stati finiti, chiamati automi-temporizzati (Timed-automata) e viene anche introdotto un tool per la verifica formale di tali automi ([BLG00]).

Grazie all'utilizzo dei meta oggetti è possibile: (a) aggiungere e rimuovere componenti di gestione della QoS dal grafo dei componenti a tempo di esecuzione, (b) definire dei meta-manager, istituiti attraverso opportune meta-politiche, che possono monitorare e adattare i manager stessi, realizzando così un meccanismo per la riconfigurazione dinamica del sistema.

### 2.2.6 f-Desktop

f-Desktop è un framework proposto dall'università di Keio (vedi [TAK01]). Questo middleware, pur non essendo un MW multimediale, è stato tuttavia da noi considerato per il supporto che offre alla mobilità utente e alla migrazione delle applicazioni. La soluzione proposta è basata sull'utilizzo di agenti mobili (per una più dettagliata descrizione di tale paradigma si veda il capitolo successivo) per assistere la mobilità degli utenti.

In particolare il sistema si concentra su: (a) localizzazione della posizione dell'utente, (b) specifica del profilo dell'utente stesso e la sua gestione, (c) adattamento dell'applicazione dipendentemente dal profilo dei terminali e delle risorse disponibili, (d) supporto alla continuità della connettività.

Per la *localizzazione della posizione* dell'utente è possibile, in generale, utilizzare i due diversi approcci: il primo è tracciare i movimenti dell'utente attraverso l'uso di complessi sistemi di sensori, il secondo, adottato in questo progetto, prevede che l'hand-off della sessione sia avviato grazie ad un dispositivo che l'utente trasporta sempre con sè. Possiamo, ad esempio, pensare ad un PDA (Personal Digital Assistant) connesso attraverso una rete wireless, per mezzo del quale viene stimolato l'inizio del processo di hand-off, nel momento in cui l'utente si avvicina al terminale obbiettivo.

Il *profilo dell'utente* ha un ruolo centrale nell'architettura: oltre ad identificare l'utente stesso esso contiene sia una lista delle applicazioni le cui sessioni possono essere mosse tra i diversi terminali, sia i dati necessari per realizzare questo processo di hand-off in modo sicuro.

Per quanto riguarda *l'adattamento dell'applicazione* ciascuna applicazione che voglia partecipare a questa architettura può registrarsi presso una o più sorgenti di eventi in modo da ricevere, attraverso l'utilizzo di questo meccanismo, informazioni circa variazione nello stato delle risorse, e informazioni circa la capacità del terminale obiettivo verso il quale l'utente vuole

muovere la sessione. Internamente tali informazioni vengono utilizzate per adattare l'applicazione, ad esempio per adattare la grandezza di una GUI di interazione con l'utente se il terminale obiettivo ha uno schermo troppo piccolo rispetto a quello del terminale dal quale la sessione è stata mossa.

Per la *continuità della connessione*, infine, f-Desktop utilizza una libreria Java precedentemente sviluppata dal medesimo gruppo (Mobile Socket, MSocket), che introduce delle opportune API per interrompere le comunicazione aperte prima della migrazione e riprenderle a migrazione avvenuta, in modo trasparente allo sviluppatore dell'applicativo.

A conclusione dell'analisi delle sei architetture considerate si riporta in figura 2.4 una tabella che riassume le caratteristiche dei diversi MW.

| Middleware | Traduzione Automatica | Specifica del grafo di servizio | Download del Codice | Riconfigurazione dinamica | Linguaggio di Specifica della QoS | Specifica di Profili Terminali/Utenti | Ambito della gestione delle risorse | Supporto per l'Application Hand-off |
|---|---|---|---|---|---|---|---|---|
| GOPI | no | no | no | no | no | no | locale | no |
| TAO/QuO | no | no | no | no | sí | no | dipende dalle politiche | no |
| MAQS | no | no | no | no | sí | no | dipende dalle politiche | no |
| 2K$^{Q+}$ | sí | sí | sí | sí | sí | sí | global | sí |
| Open ORB 2 | no | sí | no | sí | sí | no | depends on politics | no |
| f-Desktop | no | no | sí | no | no | sí | locale | sí |

**Figura 2.4**: tabella riassuntiva.

## *2.3 Conclusione*

In questo capitolo sono stati analizzati i diversi meccanismi che un multimedia middleware dovrebbe includere per rispondere ai principi

architetturali fissati nel capitolo 1. Sono state poi presentate sei architetture, evidenziando, per ciascuna di esse, i meccanismi inclusi e le peculiarità, nell'intento di trarre da questa analisi, e dal successivo confronto, utili spunti per lo sviluppo del progetto di tesi.

Nel prossimo capitolo vengono presentati i modelli di comunicazione distribuita, il paradigma ad agenti mobili ed infine SOMA, l'ambiente ad agenti che verrà utilizzato per lo sviluppo della piattaforma per la gestione dei flussi multimediali.

CAPITOLO 3

# 3 Sistemi distribuiti: modelli di comunicazione e paradigma ad agenti mobili

In questo capitolo vengono dapprima esaminati i principali modelli di comunicazione che adottati nella progettazione dei sistemi distribuiti. Negli anni si è infatti avuta un'evoluzione nel settore passando da semplici modelli cliente-servitore, a modelli più evoluti che affrontano le problematiche legate ad una sempre più larga diffusione dei sistemi distribuiti, caratterizzati da una maggiore eterogeneità di piattaforme computazionali e da nuove necessità (applicazioni soft-realtime e realtime).

Dopo una panoramica generale sui diversi modelli viene affrontata la tematica della mobilità del codice, prendendo in esame i suoi diversi paradigmi. La loro applicazione apre nuovi orizzonti nello sviluppo del middleware distribuito per il supporto alla computazione in ambienti altamente dinamici; ne esamineremo pregi e problematiche annesse. In particolare ci concentreremo poi sul paradigma ad agenti mobili, che viene utilizzato per la realizzazione di questo progetto di tesi. Verrà poi presentato il sistema usato per lo sviluppo del progetto: SOMA. In particolare verranno considerati il supporto al movimento degli agenti e l'astrazione di località introdotta da SOMA, utilizzata per la realizzazione del prototipo finale.

## *3.1 Modelli di comunicazione*

In questa sezione si considerano tre diversi modelli di comunicazione, cercando di evidenziarne pregi e difetti. I modelli basati sulla mobilità di codice non vengono dettagliati eccessivamente, perché verranno ripresi successivamente.

### 3.1.1 Modello cliente/servitore

Il primo modello che consideriamo è il classico modello cliente/servitore. Questo modello prevede l'interazione di due entità: il cliente, che richiede un servizio e il servitore che offre il servizio stesso. In figura 3.1 viene rappresentata questa situazione. Nella sua accezione più semplice il modello prevede che il

cliente conosca direttamente il servitore, mentre il servitore non conosce direttamente il cliente che sta richiedendo il servizio e l'interazione è *sincrona* e *bloccante*.


**Figura 3.1:** classica architettura client-server.

Si utilizza cioè un modello **pull** in cui il cliente è caricato della responsabilità dell'ottenimento delle informazioni a cui è interessato ed esiste, a livello logico, un unico thread che viene eseguito, in parte sul lato client, e in parte sul lato server. L'esempio forse più noto di questo modello sono le chiamate di procedura remota (Remote Procedure Call) e il corrispettivo per la programmazione Object Oriented, cioè le chiamate di metodo remoto (Remote Method Invocation). Tali meccanismi permettono la comunicazione tra processi (le prime) ed oggetti (le seconde) che si trovano su macchine diverse, gestendo la conversione dei dati fra le diverse piattaforme e i dettagli di comunicazione in modo trasparente al programmatore, che si può perciò disinteressare di questi dettagli. Il problema di questi modelli di programmazione distribuita è che lo stile di interazione è solitamente sincrono, richiede perciò al cliente di attendere la terminazione del servizio prima di continuare la computazione e non è possibile stabilire a priori il tempo che occorrerà per il completamento del servizio stesso.

Per rispondere a questi problemi viene oggi spesso utilizzato un modello **push**, in cui il cliente richiede un servizio, ma è il servitore ad avere la responsabilità del recapito del risultato. Esempi dell'applicazione di questo modello sono i servizi di notifica degli eventi, che gestiscono l'invio messaggi, disaccoppiando gli interessati. Tali servizi vengono spesso utilizzati nei multimedia MW per l'implementazione del resource monitoring distribuito.

Inoltre oggi, spesso, l'interazione non avviene più direttamente tra cliente e servitore, ma esistono catene di servizio. Si parla perciò più propriamente di modelli **n-tier**, cioè a più livelli, composti da diversi **agenti** che collaborano tra loro per offrire un servizio. Possiamo pensare ad un agente come ad un'entità che svolga sia il ruolo di server nei confronti di alcune entità, sia quello di client, nei confronti di altre. L'ultima evoluzione di questo modello è rappresentata dalle

architetture **peer-to-peer**, che sono basate su entità "bifronti", che, a seconda della situazione, agiscono da client o da server. In queste architetture abbiamo collaborazione fra pari e non è più possibile distinguere tra chi offre un servizio chi lo fruisce (con ovvie problematiche di sicurezza, accounting e controllo dell'informazione condivisa, solo per citarne alcune).

### 3.1.2 Modello a memoria condivisa

Questo secondo modello di comunicazione che consideriamo unisce l'astrazione di memoria condivisa e quella di comunicazione. L'interazione tra i processi che risiedono sulle diverse macchine avviene accedendo all'informazione condivisa attraverso uno **spazio delle tuple**, cioè un insieme strutturato di relazioni intese come attributi e valori. Il middleware è strutturato in modo da dare ai processi l'illusione di memoria condivisa, cui si può accedere attraverso le primitive di lettura e scrittura delle tuple, come avverrebbe nel concentrato.

Si vuole qui notare come, quando si considerino scenari di mobile computing, tale modello offra alcuni notevoli vantaggi, ad esempio durante le momentanee disconnessione che si possono verificare.

### 3.1.3 Modelli basati sulla mobilità di codice

I modelli basati sulla mobilità di codice aggiungono flessibilità e adattabilità alle applicazioni distribuite e possono evitare la trasmissione di ingenti moli di dati, nei casi in cui si sia interessati solo ad un piccolo sottoinsieme di essi [GEK01]. Questo nuovo paradigma prevede infatti la spedizione non più dei soli dati, ma anche di codice in modo da "spostare l'intelligenza" là dove sia necessario eseguire una certa computazione. Esempi di applicazione di questo modello sono: il movimento di processi per operare bilanciamento di carico, la distribuzione di aggiornamenti software per quegli apparati che debbano sempre rimanere in funzione, senza dover mai essere spenti, l'information retrieval, l'esecuzione di task di controllo e monitoraggio su reti remote, ecc. A tutt'oggi non è però ancora stata trovata una cosiddetta "killer application" che abbia definitivamente stabilito l'adozione generalizzata di questo modello.

Infatti, anche se il modello offre allo sviluppatore notevoli potenzialità, e la possibilità di gestire interazioni che vanno ben al di là sia di quelle del modello client/server che di quello a memoria condivisa, esistono diverse problematiche che ne hanno finora limitato l'utilizzo generalizzato.

Prima fra tutte è la richiesta di un ambiente di esecuzione uniforme al di sopra del quale fare eseguire il codice mobile. Tale richiesta rappresenta oggi una assunzione molto forte, vista l'enorme eterogeneità di piattaforme computazionali. Ad esempio per quello che riguarda SOMA l'assunzione è che tutte le macchine che vogliono partecipare all'ambiente distribuito ospitino la macchina virtuale Java (Java Virtual Machine).

Ci sono poi forti resistenze all'utilizzo generalizzato di questo modello computazionale perché l'utilizzo di codice mobile, che in molti casi dovrà accedere alle risorse di sistema e in generale dovrà comunque utilizzare la risorsa CPU, pone non pochi problemi di sicurezza, monitoraggio e accounting. Per quello che riguarda la sicurezza si vuole fare in modo che ogni agente non possa agire in modo indisturbato, ma possa compiere solo alcune azioni, accedendo solo alle risorse a cui è autorizzato. Per quello che riguarda il monitoraggio, si vuole poi che il codice mobile, che verrà eseguito al di sopra di una macchina che ospita tale codice mettendo a disposizione le proprie risorse computazionali, sia soggetto a monitoraggio. Se la macchina non è di proprietà dell'utente che ha creato l'agente si deve poter quantificare l'ammontare delle risorse utilizzate dal codice mobile, in modo da addebitare tale costo a carico di chi abbia lanciato il codice stesso.

Da ultimo, in particolare per ciò che riguarda gli agenti mobili, l'utilizzo di questo modello computazionale apre un'infinità di possibili di interazioni fra molte entità indipendenti, ciascuna delle quali esegue in modo parallelo un certo task sulla piattaforma distribuita. Il problema è che non esistono ancora dei paradigmi di programmazione atti a guidare lo sviluppo di applicazioni che facciano uso di questo nuovo modello.

## *3.2 Mobilità di codice*

In questa sezione vengono considerate più in dettaglio la mobilità di codice e i diversi paradigmi di tale mobilità. Un ottimo articolo sull'argomento è [FUA98], dal quale sono anche tratte alcune delle immagini seguenti. L'architettura di riferimento che verrà considerata per la mobilità di codice è quella mostrata in figura 3.2.

Al di sopra dell'hardware si hanno i diversi sistemi operativi, qui chiamati Core Operating System (COS), e il Network Operating System (NOS), che offre servizi per una comunicazione non trasparente. Le applicazioni sviluppate direttamente al di sopra dei NOS devono nominare direttamente l'host col quale vogliono comunicare (l'astrazione di socket appartiene ai NOS). Infine, al di

sopra di questi due strati, risiede il Computational Environment che ospita l'esecuzione dei Component, che rappresentano il codice mobile.

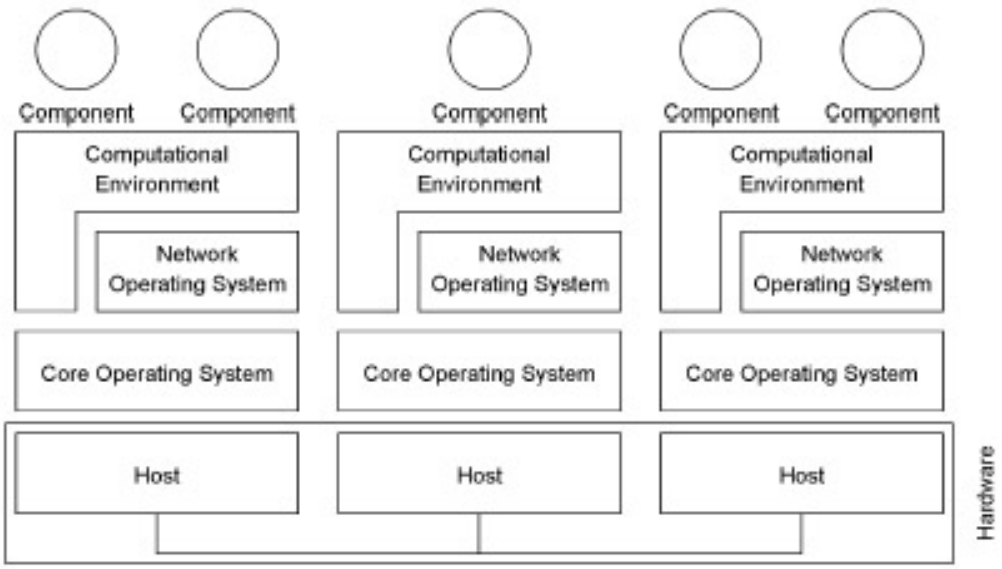


**Figura 3.2:** architettura di riferimento per la mobilità di codice.

Questa figura mette in rilievo una prima caratteristica fondamentale degli ambienti di computazione che ospitano il codice mobile (Component). I Computational Environment (CE) possono ospitare i component, offrendo loro un ambiente di esecuzione uniforme, ma sono logicamente tra loro divisi. Permane cioè una nozione di località anche se il livello applicativo viene sollevato dalla gestione dei canali di comunicazione, gestita più a basso livello dai CE interagendo coi NOS. Quando vengono utilizzati middleware che offrono trasparenza alla locazione, invece, tale informazione non è disponibile al livello applicativo, dato che tutti i componenti vengono percepiti come locali. Esempi di queste piattaforme sono i riferimenti ad oggetti remoti. In questi casi il programmatore si può disinteressare della fisica locazione dei componenti e dell'organizzazione della rete sottostante perdendo però in capacità espressiva. Questa prima proprietà messa in risalto, cioè il fatto che i modelli basati su mobilità del codice siano location aware, è uno dei principi che, come visto nei capitoli precedenti, è anche alla base della realizzazione dei multimedia middleware.

### 3.2.1 Mobilità dello stato di esecuzione

I Component introdotti nella sezione precedente possono essere distinti in: risorse e execution unit (EU). Una EU rappresenta un singolo thread con un

proprio stato di esecuzione e uno spazio dati e può condividere delle risorse con altre EU (la struttura interna di una EU è rappresentata in figura 3.3). Relativamente al fatto che lo stato di esecuzione di una EU venga mosso o meno vengono distinti due tipi di mobilità: **mobilità forte** e **mobilità debole**.

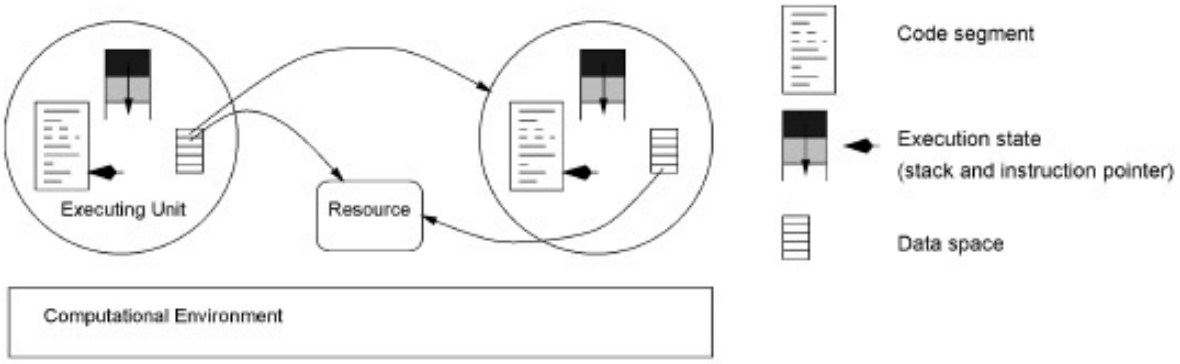


**Figura 3.3:** struttura interna di una EU.

La **mobilità forte** supporta la migrazione sia del codice della EU che dello stato di esecuzione. Affinché tale tipo di mobilità sia attuabile deve essere possibile salvare lo stato di esecuzione cioè, come mostrato in figura, lo stack e l'instruction pointer di una certa EU. All'arrivo nel nuovo CE lo stato di esecuzione verrà ristabilito e la EU potrà continuare la propria esecuzione da dove era stata sospesa. Naturalmente rimane il problema di come gestire i riferimenti alle risorse, aspetto che affronteremo nei prossimi punti. La **mobilità debole** invece supporta la migrazione del codice ed eventualmente di uno stato. Non un'immagine dello stato di esecuzione, bensì uno stato costruito al livello applicativo, in modo che, quando la computazione riprenda all'arrivo nel nuovo CE, tale stato possa essere riutilizzato per inizializzare la EU e continuare la computazione.

Si vogliono fin da ora fare due osservazioni. La prima è che la scelta del linguaggio per la realizzazione del supporto alla mobilità incide fortemente sulla possibilità o meno di realizzare l'uno o l'altro tipo di mobilità; infatti scegliendo un linguaggio che permetta il salvataggio dello stato di esecuzione sarà possibile implementare la mobilità forte, altrimenti no (cosa che accade per esempio nel caso dell'utilizzo di Java). La seconda è che, dal punto di vista dello sviluppatore finale, sistemi che supportino la mobilità forte lo sollevano dal compito di salvare per ogni salto lo stato in una apposita struttura dati, e di ripristinarlo all'arrivo nella nuova CE.

### 3.2.2 Mobilità del codice

A seconda di come avvenga la mobilità del codice, nel caso della mobilità debole, vengono solitamente distinti vari casi considerando le diverse dimensioni elencate di seguito.

- Direzione nella quale avviene il trasferimento di codice: una EU può richiedere il trasferimento di codice (**fetch the code**) da remoto per poi eseguirlo localmente, oppure può spedire (**ship the code**) essa stessa del codice ad un'altra CE.
- Natura del codice che deve essere mosso: il codice può essere **stand-alone code**, cioè codice che sia autocontenuto e, una volta arrivato nella nuova località, venga utilizzato per istanziare una nuova, indipendente EU, oppure può rappresentare un frammento di codice (**code fragment**) che verrà utilizzato da una EU già in esecuzione per essere collegato (linked) ed eseguito insieme ad altro codice.
- Sincronizzazione: dipendentemente dal comportamento della EU che richiede il trasferimento del codice, vengono distinti due casi: il caso sincrono quando l'EU si sospende in attesa che il codice trasferito sia stato completamente eseguito e il caso asincrono, se non c'è sospensione.

### 3.2.3 Gestione dello spazio dei dati

Ritornando a quanto detto sopra, rimane da affrontare un'ultima problematica, cioè come venga gestito lo spazio dei dati (data space, si veda la figura 3.3), durante la migrazione di una EU. Ogni risorsa può venire referenziata con tre modalità principali: con un'identificatore unico all'interno del sistema, o per valore o per tipo. A seconda di quale, o quali modalità vengano utilizzate, esistono diverse possibilità per la riallocazione delle risorse e la riconfigurazione dei binding alle risorse, dopo il salto di una EU. In [FUA98], cui si rimanda per ulteriori approfondimenti, viene riportato uno studio di questa problematica. Si vuole solo sottolineare che l'applicazione sviluppata utilizza principalmente il meccanismo **re-binding**, le risorse sono cioè referenziate per tipo e, dopo una migrazione, durante la fase di inizializzazione, si ricostruisce lo spazio dei dati, procedendo al re-bind delle varie risorse di cui una certa EU necessita nel nuovo CE.

### 3.2.4 Paradigmi di mobilità di codice

In relazione a come avviene l'esecuzione del servizio e alla posizione delle EU e delle risorse dopo la terminazione del servizio si distinguono diversi

paradigmi di mobilità del codice. In figura 3.4 viene riportata una tabella che riassume le diverse possibilità. Con A viene indicata una generica entità computazionale, collocata nel sito $S_A$, che è interessata alla ricezione del risultato di un certo servizio. Si assume poi l'esistenza di un sito $S_B$ che sarà coinvolto nella esecuzione di tale servizio. A e B sono due entità computazionali e in grassetto viene indicata l'entità che esegue il codice.

MOBILE CODE PARADIGMS

| Paradigm | Before | | After | |
|---|---|---|---|---|
| | $S_A$ | $S_B$ | $S_A$ | $S_B$ |
| *Client-Server* | A | know-how<br>resource<br>B | A | know-how<br>resource<br>**B** |
| *Remote Evaluation* | know-how<br>A | resource<br>B | A | *know-how*<br>resource<br>**B** |
| *Code on Demand* | resource<br>A | know-how<br>B | resource<br>*know-how*<br>**A** | B |
| *Mobile Agent* | know-how<br>A | resource | – | *know-how*<br>resource<br>***A*** |

**Figura 3.4**: paradigmi per la mobilità di codice [FUA98].

Nel classico modello client/server ovviamente non c'è mobilità di codice, e sia la conoscenza per effettuare la computazione, sia le risorse necessarie, sono presenti sul sito $S_B$.

#### 3.2.4.1 Remote Evalutation (REV)

Nel caso della remote evaluation il componente A possiede la conoscenza necessaria all'esecuzione del servizio, ma non dispone delle risorse necessarie per portare a termine tale computazione, dato che si trovano sul sito $S_B$. Perciò A passa all'entità B il necessario know-how, che B utilizza per portare a termine il servizio e spedire, in un secondo tempo, il risultato ad A. Casi pratici dell'applicazione di questo paradigma, sono, ad esempio, l'esecuzione di operazioni che richiedano un intenso uso di risorse computazionali, come calcoli scientifici complessi, oppure un intenso uso di dati presenti sul sito $S_B$ dai quali si vogliano ricavare risultati di dimensioni ridotte, rispetto a quelle dell'informazione da processare. Applicando tale paradigma si evita in questi casi di trasmettere attraverso la rete un'ingente quantità di dati.

#### 3.2.4.2 Code On Demand (COD)

Nel caso del code on demand invece l'entità A si trova collocata sul sito $S_A$ insieme alle risorse necessarie alla computazione, tuttavia non dispone della conoscenza necessaria al compimento del servizio, richiede perciò tale conoscenza all'entità B, e non appena dispone del know-how A porta a termine il proprio compito. Applicazioni di questo paradigma sono gli aggiornamenti automatici di codice.

#### 3.2.4.3 Mobile Agent (MA)

L'ultimo paradigma considerato è quello degli agenti mobili. In questo caso l'entità A possiede la conoscenza necessaria per portare a termine il servizio, ma necessita di alcune risorse che sono presenti solo sul sito $S_B$, migra perciò su tale sito e termina l'esecuzione del servizio. Esiste una differenza fondamentale fra questo paradigma e quelli presentati in precedenza, e cioè che, mentre nei casi precedenti ciò che veniva spostato era principalmente codice, qui si muove l'intera EU, col proprio stato, codice, e risorse. Nel sistema in progetto gli agenti mobili vengono utilizzati anche per la configurazione del sistema distribuito e per assistere la mobilità dell'utente e del terminale. Gli agenti non si spostano cioè là per accedere a risorse altrimenti non disponibili, ma piuttosto per essere sempre vicini all'utente/terminale che stanno assistendo.

Un importante aspetto legato alla realizzazione di questo paradigma è la decisione su come realizzare la **comunicazione fra gli agenti**. Bisogna innanzitutto supportare l'**identificazione** degli agenti stessi, in modo che sia possibile per un certo agente esprimere la volontà di aprire la comunicazione con un altro agente. La comunicazione può poi essere realizzata in molti modi. Possono essere utilizzate operazioni per la spedizioni di semplici messaggi (ad esempio oggetti serializzabili che vengono passati fra gli agenti), che si ispirino al message-passing, ma possono anche essere implementate comunicazioni di tipo strema-based fino all'utilizzo. Particolare rilevanza riveste poi l'utilizzo del modello a memoria condivisa, per la realizzazione di comunicazioni di gruppo fra più agenti.

### 3.2.5 Sicurezza

Da ultimo si vogliono accennare alcun problematiche legate alla sicurezza, infatti se in una rete locale possiamo ipotizzare di avere un elevato grado di fiducia, e quindi la mobilità di codice non implica particolari problemi, quando invece si considerino ambiti più ampi la sicurezza rappresenta un problema.

La paura degli amministratori di sistema è infatti quella di ricevere e far eseguire sulla propria macchina un codice non autorizzato che possa andare a minare le basi del sistema stesso, o anche, più banalmente, lo renda inutilizzabile, appropriandosi di tutte le risorse. Tale problema è stato uno dei fattori che hanno limitato la diffusione dei paradigmi sopra esposti, ed in particolare del paradigma ad agenti mobili. Elenchiamo qui alcuni dei requisiti di sicurezza richiesti ad un sistema che voglia supportare la mobilità di codice (si veda inoltre [TRA98]).

- **Riservatezza ed integrità**: è spesso necessario mantenere riservate alcune parti dello stato di un agente mentre attraversa la rete, così come bisogna implementare metodi per verificare l'integrità di un agente e più in generale di codice mobile, al suo arrivo in un nuovo sito.
- **Autenticazione**: ogni pezzo di codice mobile deve essere autenticato, in modo che i CE che ospitano tale codice possano determinare l'identità dell'utente che ne richiede l'esecuzione, questo anche per permettere l'addebito delle risorse utilizzate.
- **Autorizzazioni e controllo degli accessi**: si vuole dare la possibilità ai proprietari delle risorse di specificare le politiche di accesso alle risorse stesse, basando tali politiche su diverse dimensioni, non solo sull'identità dell'utente che ha progettato il codice mobile, ma anche sul suo ruolo, ecc., inoltre si vuole dare la possibilità agli sviluppatori degli agenti di specificare delle restrizioni ai diritti del proprio agente. Questa funzionalità sarebbe ovviamente molto utile in fase di debugging dell'agente stesso.

## *3.3 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. Qui se ne vogliono delineare semplicemente le caratteristiche architetturali principali, mantenendo come riferimento la teoria generale esposta nei paragrafi precedenti.

SOMA utilizza Java come piattaforma di implementazione e ne sfrutta tutte le potenzialità. Java è 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; è, cioè, portabile su diverse architetture software/hardware;

- è un linguaggio object-oriented che permette uno sviluppo modulare del codice per estensione ed ereditarietà, così che l'utente possa 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;
- fornisce la possibilità di serializzare gli oggetti, cioè di rappresentarli come 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 d'accesso e permessi da associare sia a codice remoto che a codice locale.

D'altra parte, come evidenziato in precedenza, il linguaggio obbliga alla scelta di un determinato modello di mobilità. Essendo un linguaggio interpretato, una parte dello stato di esecuzione rimane incluso nello stato dell'interprete; 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.

SOMA è allora un sistema a mobilità debole e, facendo uso della terminologia già introdotta al paragrafo 3.2, 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.

### 3.3.1 Caratteristiche di SOMA

#### 3.3.1.1 Astrazioni di località in SOMA

SOMA fornisce una gerarchia di astrazioni di località, adatta alla descrizione di ogni scenario di connessione, mutuata da Internet: esistono 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 3.5).

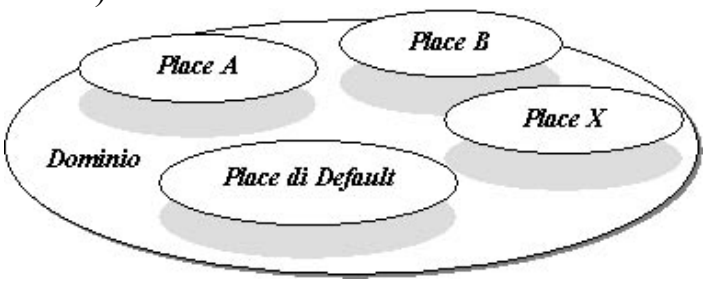


**Figura 3.5**: località logiche in SOMA.

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. 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 primo motivo di questa scelta è che in questo modo il place può essere utilizzato come utile astrazione di località fisica, sulla cui importanza non ci dilungheremo oltre, avendone già più volte parlato. L'altro motivo è che 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 affrontate in 1.2.2. Ad esempio, come vedremo, alcuni servizi di base, come la realizzazione del 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.

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 3.6 riportiamo un esempio che utilizza una gerarchia di semplice comprensione. Il default place root 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]. Tale astrazione di località verrà utilizzata per la realizzazione del servizio di località incluso nel supporto in progetto.

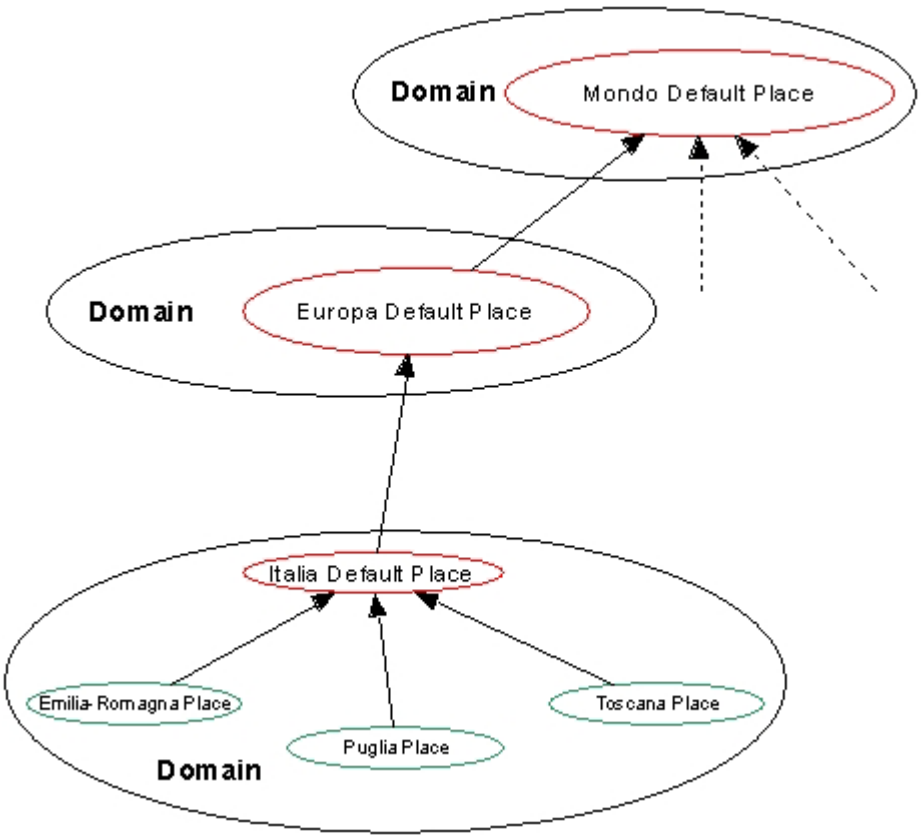


**Figura 3.6**: organizzazione gerarchica delle astrazioni di località in SOMA.

### 3.3.1.2 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:

- *Agent Manager*, che gestisce gli agenti permettendone l'esecuzione. l'ingresso e uscita da un place;
- *Network Manager*: che gestisce le interazioni tra i place, mantenendo i necessari canali di comunicazione;
- *Information Service*: che gestisce le informazioni sui place appartenenti ad un dominio e sui domini ad esso noti.

#### 3.3.1.3 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 istanziazione. 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.

#### 3.3.1.4 Comunicazione in SOMA

La capacità di un agente di comunicare con altri agenti è un'ulteriore caratteristica di base. L'ultima versione del sistema prevede funzionalità di comunicazione basate sul meccanismo dello scambio di messaggi, possibile tra qualunque coppia di agenti, locali o remoti. In realtà questa soluzione è molto flessibile, perchè fornisce le basi per la creazione di schemi di comunicazione avanzati. La trasparenza, rispetto alla locazione di mittente e destinatario di un messaggio, è realizzabile grazie al servizio di localizzazione degli agenti offerto dal supporto e attivabile su necessità.

#### 3.3.1.5 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 introdurre 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.

### 3.3.2 Particolari di Implementazione

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

#### 3.3.2.1 Gli Agenti

Un agente è realizzato come classe derivata dalla superclasse astratta *Agent* ed è un semplice oggetto passivo serializzabile, proprio perchè 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 istanziazione. 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*, perchè 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 specificare 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, l'*AgentClassLoader*, che richiede trasferimento del codice da remoto solo su necessità (paradigma COD) e lo mantiene in una cache locale per utilizzi successivi.

#### 3.3.2.2 La Classe 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 e migrazione. A livello implementativo il contenitore di tutto ciò è la classe *Environment* e la sua istanziazione 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 3.7). Risulta allora evidente perchè 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.

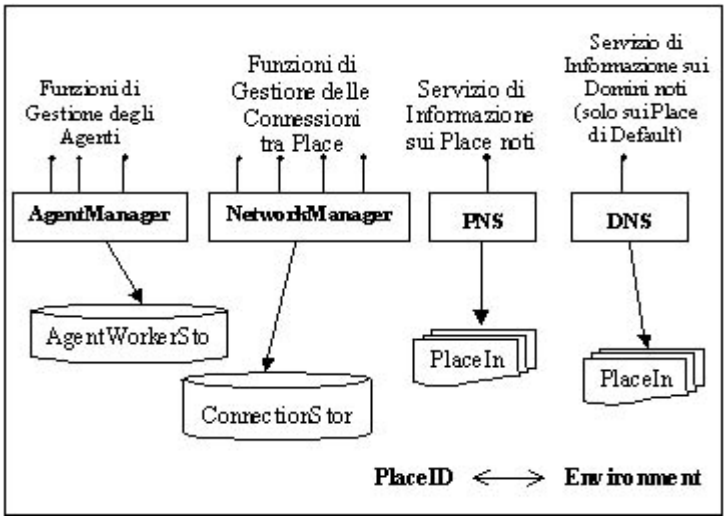


**Figura 3.7**: componenti principali dell'Environment

#### *3.3.2.2.1 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, istanza 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.

#### *3.3.2.2.2 Gestore di Rete*

La gestione delle comunicazioni relative ad un place è logicamente a sè 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.

#### *3.3.2.2.3 Informazioni su Place e Domini*

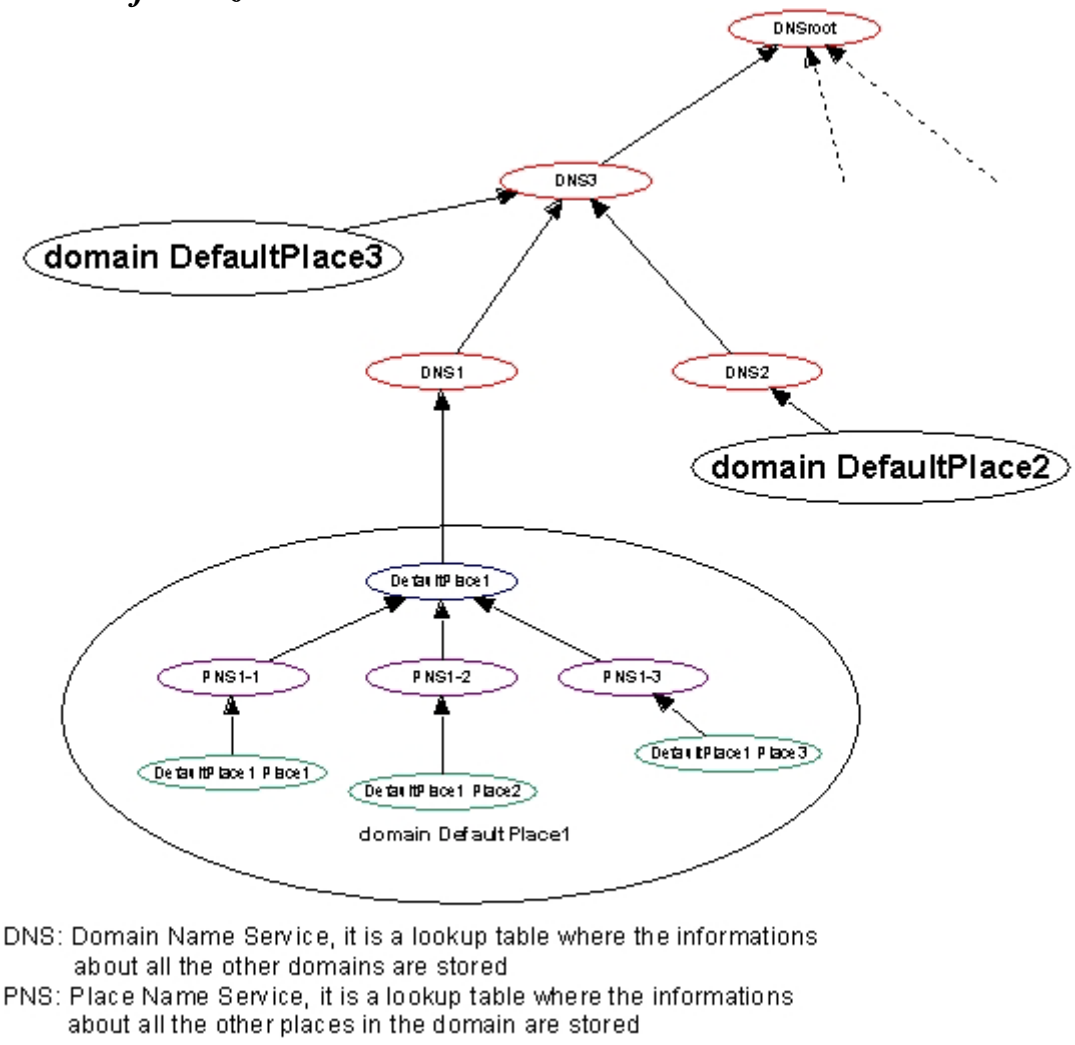


**Figura 3.8**: organizzazione di PNS e DNS.

I due attributi dell'Environment che rimangono da considerare costituiscono un accesso al servizio di informazioni sull'ambiente stesso, congiuntamente realizzano l'Information Service: il *PlaceNameService* gestisce il naming dei place, il *DomainNameService* quello dei domini. La struttura dei due è sostanzialmente analoga: sono la realizzazione di un data base di informazioni accessibili mediante chiave unica, il PlaceID, che nel primo caso è identificatore di un place generico, nel secondo individua un place di default. Affinchè 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 3.8.

#### 3.3.2.3 Comunicazione tra Agenti

La posizione degli agenti non è importante dato che può esserci comunicazione locale o remota purchè, come anticipato, l'agente sia stato creato come rintracciabile e se ne conosca l'AgentID. Il messaggio (Message) contiene gli identificatori di mittente e destinatario e come corpo ha un oggetto qualsiasi purchè, ovviamente, serializzabile. La ricezione di messaggi (*getMessage()*) è un'operazione bloccante a default, ma è anche possibile verificare lo stato della mailbox (*isMessage()*) e prelevare così messaggi solo se presenti.

#### 3.3.2.4 Interazione tra Place

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 REV). 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.

#### 3.3.2.5 Sicurezza

Il modello di sicurezza in SOMA si basa sui meccanismi offerti dal linguaggio Java, in particolare dell'ultima versione che, come si è già detto in precedenza, è molto ricca e flessibile. Il ClassLoader e la gestione delle politiche devono essere adattati alle specifiche esigenze di un sistema ad agenti; così, ad esempio, è stato definito un *AgentClassLoader* ad hoc. Per proteggere l'ambiente di esecuzione è necessario attribuire agli agenti specifici permessi (sottoclassi di *Permission*) in base alle azioni che devono compiere; ogni permesso è infatti caratterizzato da un target, cioè una risorsa locale, e da zero o più azioni. Esistono permessi predefiniti: per esempio, quello di accesso ad un Place (*PlaceAccessPermission*) e quello per ottenere il riferimento ad un Environment (*AgentPermission*). Affinchè si possano attribuire i giusti permessi ad un agente deve essere identificato il suo *Principal* che, in Java, viene rappresentato tramite un *CodeSource* composto da un URL e da un insieme di chiavi crittografiche pubbliche.

Le chiavi pubbliche permettono 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, a quest'ultimo possono essere associati i permessi previsti dalla politica.

## *3.4 Conclusione*

In questo capitolo sono stati analizzati diversi modelli di comunicazione, dettagliando in particolar modo il quello basato sulla mobilità di codice e il paradigma ad agenti mobili. È poi stato presentato SOMA, l'ambiente per il supporto degli agenti mobili utilizzato per lo sviluppo del progetto di tesi, mettendone in luce i principi architetturali e i servizi offerti.

Nel prossimo capitolo verrà presentato lo standard IEEE 802.11, che è la tecnologia adottata per la realizzazione della rete wireless. Dopo la presentazione del quadro tecnologico nel quale il sistema verrà sviluppato, vengono introdotte le Wireless API, una libreria Java sviluppata presso questo stesso dipartimento per fornire allo sviluppatore Java la visibilità diretta di alcuni servizi di basso livello, solitamente nascosti al livello applicativo (si veda a tale proposito la tesi [BEM02]). In particolare, con riferimento alle infrastructure local area network, che verranno esaminate nel prossimo capitolo, a livello applicativo si sfruttano tali API per ottenere visibilità dei diversi BSS (Basic Service Set).

CAPITOLO 4

# 4 Standard IEEE 802.11 e Wireless API

In questo capitolo si presenta la tecnologia wireless 802.11, presentando dapprima la suite di protocolli, e poi le due tipologie di architetture, ***ad-hoc local area network*** e ***infrastructure local area network***, dettagliando in particolar modo quest'ultima sulla quale si concentra questo lavoro di tesi. Verranno poi prese in considerazione le Wireless API, presentando brevemente le architetture per l'implementazione di tali API in ambienti Linux e Windows e le funzionalità offerte allo sviluppatore Java.

## *4.1 IEEE 802.11*

Nel campo delle tecnologie wireless sono stati numerosi gli standard nati per implementare sistemi di questo tipo:

- ***HyperLan 2:*** standard per comunicazioni wireless, specializzato per applicazioni multimediali di tipo home;
- ***HomeRF:*** standard per la trasmissione in radiofrequenza di dispositivi domestici;
- ***DECT(Digital Enhanced Cordless Telecommunications):*** standard digitale criptato per telefonini cordless;
- ***IEEE 802.11:*** standard per WLAN sviluppato dall'*Institute of Electrical and Electronic Engeneering* [80211].

Nel 1997 nasceva il primo standard di riferimento IEEE 802.11, che dettava le specifiche a livello fisico e a livello di datalink per l'implementazione di una LAN wireless (WLAN); tale standard ha avuto successive versioni, ognuna delle quali ha cercato di sopperire alle mancanze delle precedenti. Si riporta di seguito in figura 4.1 una tabella parziale che mostra le evoluzioni di questo standard.

| *Standard* | *Frequenza / Data rate* |
|---|---|
| IEEE 802.11 | 2,4 GHz / 1-2 Mbps |
| IEEE 802.11b (Wi-fi) | 2,4 GHz / 5,5-11 Mbps |
| IEEE 802.11a (Wi-fi 5) | 5-40 GHz / fino a 54 Mbps |
| IEEE 802.15.1 | nuovo standard |

**Figura 4.1**: evoluzioni dello standard IEEE 802.11.

Per brevità, sono state riportate solo le successive due versioni dello standard 802.11 (la “a” e la “b”): come si nota dalla tabella, le nuove specifiche hanno portato un incremento sulla velocità di trasmissione e variazioni sulla frequenza della banda utilizzata: oltre a questi aspetti fisici della comunicazione, gli standard hanno anche dettagliato e modificato le specifiche di implementazione di una WLAN, ad esempio prendendo in considerazione la qualità di servizio (QoS).

### 4.1.1 La suite di protocolli 802.11

Come tutti i protocolli appartenenti alla famiglia degli standard 802.x, anche lo standard 802.11 copre due layer, il ***MAC layer*** e il ***physical layer[MAC99]***; mostriamo in figura 4.2 la suite dei protocolli 802.11.

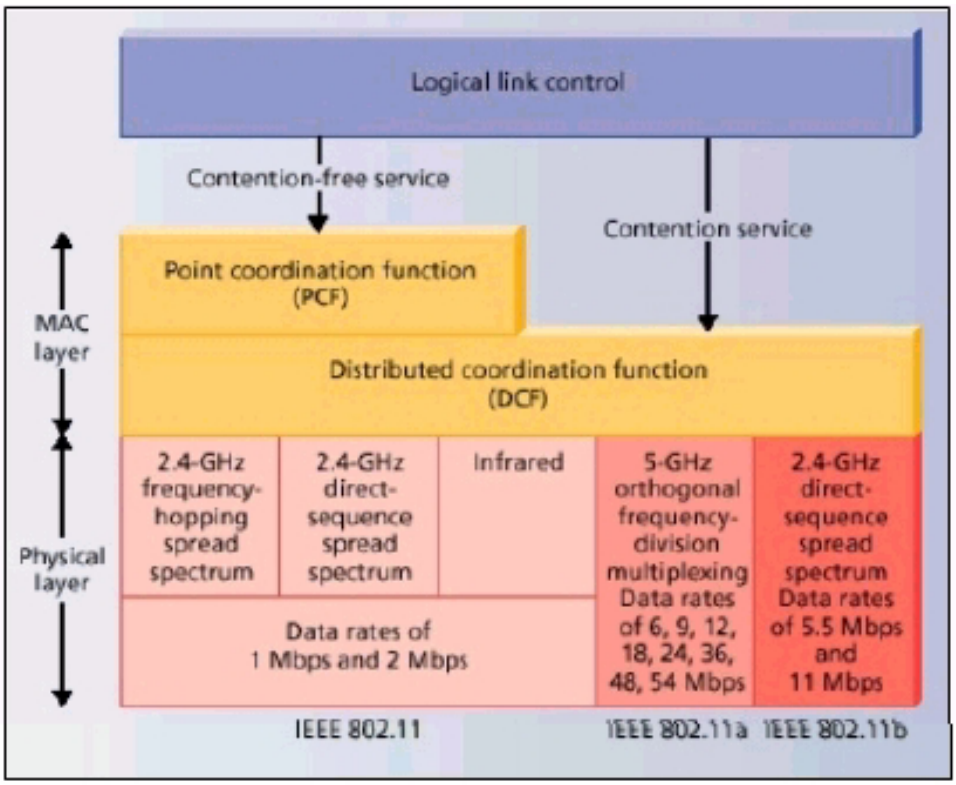


**Figura 4.2**: suite di protocolli 802.11.

E’ possibile rapportare questi due livelli introdotti dallo standard 802.11 con i layer definiti dal classico modello OSI, come evidenziato in figura 4.3.

#### 4.1.1.1 Il MAC layer

Facendo riferimento alla figura 4.2, il MAC layer è composto da due sub-layer: il *Point Coordination Function* e il *Distributed Coordination Function*, che definiscono due differenti metodi di accesso.

Il Distributed Coordination Function (DCF) definisce il meccanismo di accesso base al canale di comunicazione wireless: fondamentalmente, tale meccanismo è regolamentato da un protocollo di tipo *CSMA (Carrier Sense Multiple Access)* con strategia *Collision Avoidance*.

Tali tipi di protocollo offrono buoni risultati nel caso in cui il mezzo trasmissivo non sia pesantemente caricato: ad ogni modo c'è sempre la possibilità di collisione, nel caso in cui due stazioni trasmettano contemporaneamente dati, credendo che il canale sia libero.

Tale situazione di collisione deve essere identificata e trattata in maniera opportuna: a tal fine, il MAC layer potrebbe provvedere alla ritrasmissione dei pacchetti collisi, non appesantendo i livelli superiori: nel caso di reti Ethernet, il problema della collisione è affrontato e superato con la ritrasmissione dei pacchetti persi, secondo un algoritmo a *back off esponenziale.*

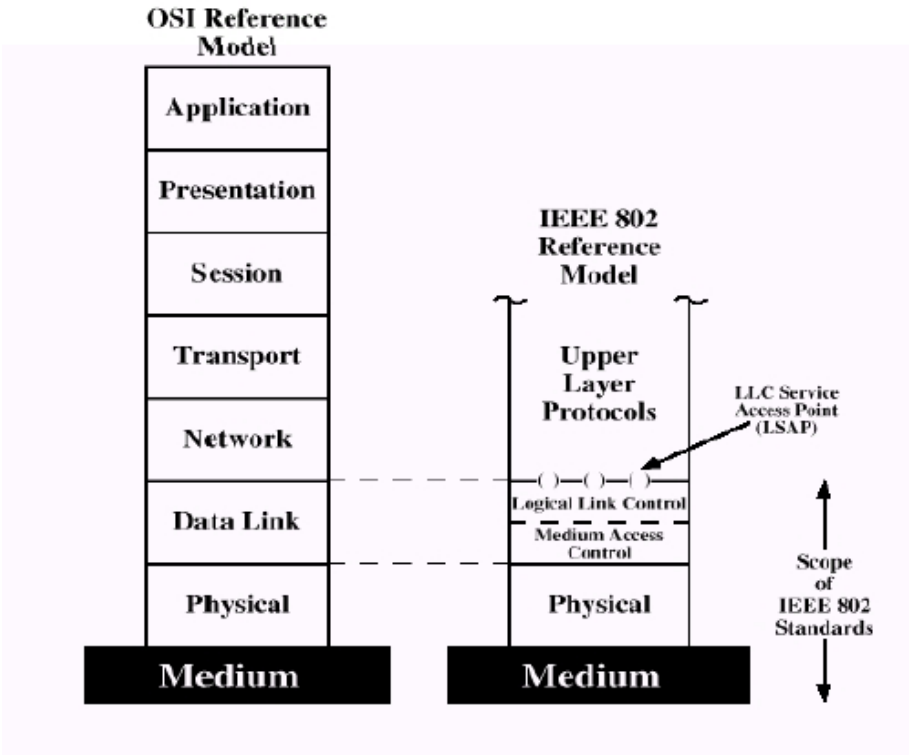


**Figura 4.3**: il modello OSI e lo standard 802.11.

Tale soluzione (*Collision Detection*) non può essere sfruttata in un ambiente wireless, in quanto:

1. il meccanismo di Collision Detection richiederebbe un supporto radio necessariamente full-duplex, incrementando i costi in modo significativo;
2. in un ambiente wireless non possiamo assumere che tutte le postazioni siano in visibilità reciproca diretta (assunzione che invece è tipica degli schemi basati sulla Collision Detection); quindi, se un terminale percepisce il mezzo trasmissivo come libero, ciò non implica che il mezzo sia ugualmente libero nelle circostanze dell'area del ricevente.

Alla luce di quanto esposto, lo standard 802.11 propone una strategia *Collision Avoidance,* di cui mostriamo di seguito le peculiarità.

Se Tx osserva che il mezzo è impegnato, allora dovrà ritardare la fase di trasmissione, viceversa, se il canale resterà libero per un tempo predeterminato (chiamato *DIFS, Distibuted Inter Frame Space*), Tx invierà ad Rx un pacchetto di CRC (*Cyclic Redundant Check*). Se effettivamente il canale è libero anche nell'area del ricevente, Rx riceverà il CRC e potrà rispondere con un acknowledgement (*ACK*): ricevendo il messaggio di ack, Tx saprà che non sono avvenute collisioni durante questa prima fase e quindi potrà tentare la trasmissione dei dati.

Per prevenire ulteriori collisioni, si definisce un meccanismo di *Virtual Carrier Sense*: con questo meccanismo, basato sullo scambio di altri due messaggi dal trasmittente al ricevente e viceversa (RTS Request To Send, CTS Clear To Send) si impegna volontariamente il mezzo trasmissivo ancor prima di iniziare la trasmissione, mantenendolo quindi occupato per tutta la durata della comunicazione.

In questo modo si evita che altre stazioni impegnino il canale, avendo l'esclusiva sull'utilizzo del canale durante la sessione di interazione tra i due comunicanti: in tale modo si tenta di prevenire eventuali collisioni.

Altra importante funzionalità svolta a livello MAC è la *frammentazione* e il *riassemblaggio* dei pacchetti: tipicamente i protocolli utilizzati in wired LAN fanno uso di pacchetti di dimensione elevata (centinaia di byte), ma ciò è sconsigliato in WLAN, infatti:

1. la probabilità che un pacchetto si corrompa durante la sua trasmissione aumenta con il crescere della dimensione del pacchetto stesso;

2. in caso di corruzione, più piccolo è il pacchetto e minore è l'overhead introdotto dalla sua ritrasmissione;
3. in sistemi a radio frequenza, il mezzo trasmissivo viene periodicamente interrotto per ottenere sincronizzazione (frequency hopping system): in tal caso pacchetti corti presentano maggiore probabilità di essere trasmessi prima del "salto" della portante rispetto a pacchetti lunghi, ottenendo benefici in termini di prestazioni.

Perciò il MAC layer introduce un meccanismo di frammentazione/riassemblaggio, proprio per evitare di avere pacchetti di dimensione troppo elevata; mostriamo con la seguente figura come un MSDU (MAC Service Data Unit) venga suddiviso in più frammenti, come mostrato in figura 4.4.

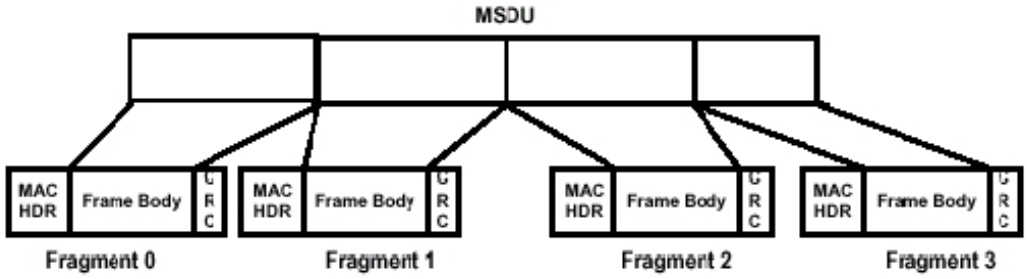


**Figura 4.4**: frammentazione introdotta dal MAC layer.

Come si nota, ad ogni singolo frammento il livello di MAC associa un header ed un trailer per garantire la consegna.

### 4.1.1.2 Il Physical Layer


**Figura 4.5**: i tre livelli fisici.

Lo standard definisce 3 differenti layer, come evidenziato in figura 4.5. Due di questi layer fisici si riferiscono a trasmissioni radio, mentre la terza riguarda trasmissioni a raggi infrarossi: sia il layer *FHSS* (*Frequency Hopping Spread Spectrum*) che quello *DSSS* (*Direct Sequence Spread Spectrum*) riescono

a supportare velocità di trasmissione di 1 Mbps e 2 Mbps, in funzione dello specifico tipo di modulazione adottato.

Lo standard 802.11 suddivide logicamente il physical layer in 2 sublayer; si evidenzia in figura 4.6 la presenza di questi due sottolivelli.


**Figura 4.6**: sublayer logici del livello fisico.

- ***Physical Medium Dependent Sublayer:*** tale livello definisce le modalità attraverso cui i dati possono essere trasmessi attraverso il canale wireless. Competono a questo livello la scelta del tipo di modulazione da usare e la codifica dei dati;
- ***Physical Layer Convergence Protocol:*** tale livello comunica col precedente e offre ai livelli superiori un'interfaccia indipendente dagli aspetti più fisici della comunicazione, incapsulati nel PMD.

### 4.1.2 Architettura 802.11

Lo standard 802.11 propone due architetture differenti di reti wireless: le ***ad-hoc local area network*** e le ***infrastructure local area network***. Analizziamo di seguito queste due tipologie di schemi.

#### 4.1.2.1 Ad-hoc local area network

In questa tipologia di reti sono gli stessi terminali mobili a costituire la rete: l'intera rete è così vista come mobile, in quanto non esistono punti fissi all'interno di essa.

Questo schema non prevede quindi alcuna infrastruttura fissa che fornisca servizi di supporto, ma i terminali, che saranno in visibilità diretta, potranno direttamente comunicare tra loro, scambiandosi dati, senza aver alcun

intermediario prestabilito: sottolineiamo in questa sede che le reti ad-hoc non escludono la presenza di eventuali servitori, ma se questi sono presenti, sono entità mobili anch'essi. La seguente figura 4.7 rappresenta una tipica ad-hoc LAN.

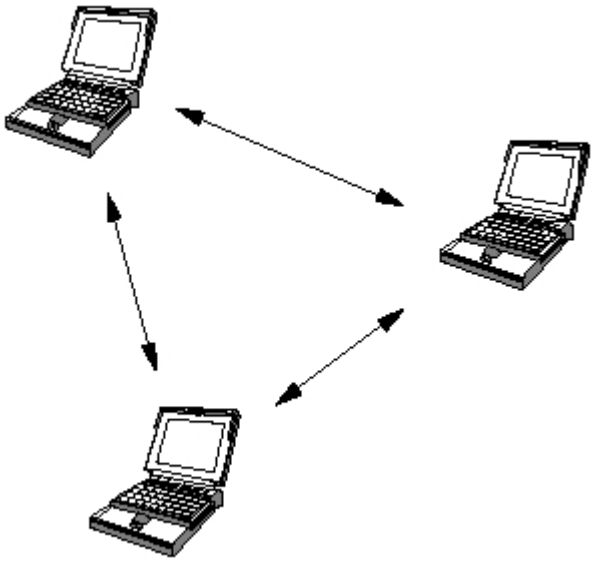

**Figura 4.7**: modello di una rete ad hoc.

Una rete ad-hoc è assolutamente indipendente, ovvero non è legata in alcun modo ad un'altra eventuale rete ad-hoc remota o a qualunque altra rete wired, questo proprio per l'assenza di un'infrastruttura: per evidenziare quest'indipendenza intrinseca di queste reti, lo standard 802.11 definisce una rete ad-hoc come un *Indipendent Basic Service Set* (*IBSS*), identificando una cella composta da più terminali mobili.

#### 4.1.2.2 Infrastructure local area network

In contrapposizione alle ad-hoc local area network, troviamo le *infrastructure local area network*: ovviamente anche in questo caso parliamo di reti wireless, in cui abbiamo terminali mobili che possono comunicare in maniera reciproca, ma incontriamo la presenza di un'infrastruttura fissa che funge da supporto alla parte mobile. Analizziamo dapprima quali sono le entità che costituiscono questa tipologia di reti wireless:

- ***stazione:*** è quel componente mobile che si connette al canale di comunicazione wireless;

- ***Basic Service Set (BSS):*** un BSS è un insieme di stazioni che possono comunicare reciprocamente. Ricordiamo che le ad-hoc LAN sono particolari BSS indipendenti (IBSS). Nelle infrastructure LAN invece di rilevante importanza sono gli ***access point*** (***AP***), stazioni radio fisse che consentono ai dispositivi wireless di comunicare: quando infatti un terminale mobile vuole comunicare con un'altra stazione wireless, invia i dati all'AP, e quindi questi vengono ridiretti all'effettivo destinatario.
- ***Extended Service Set (ESS):*** è un insieme di infrastrutture BSS, dove i relativi AP si coordinano tra di loro, consentendo il movimento di una stazione mobile da un BSS all'altro (***roaming***) e la comunicazione reciproca. E' possibile permettere una comunicazione tra i BSS attraverso un canale astratto, chiamato ***Distribution System*** (***DS***): un ESS, con tutte le sue stazioni mobili, appare come un singolo MAC-layer di rete, dove tutti i terminali sembrano essere fissi. Quindi un osservatore esterno non si rende conto della mobilità che in realtà esiste all'interno di un ESS;
- ***Distribution System (DS):*** è il canale astratto anticipato in precedenza. Grazie al DS un AP può comunicare con un altro, garantendo la connettività tra terminali appartenenti a BSS differenti: lo standard 802.11 non specifica come questo debba essere implementato, dunque il DS potrebbe essere sia un'infrastruttura wired che un'infrastruttura wireless;

Si mostra in figura 4.8 la topologia di una infrastructure LAN, evidenziando tutti i suoi componenti.

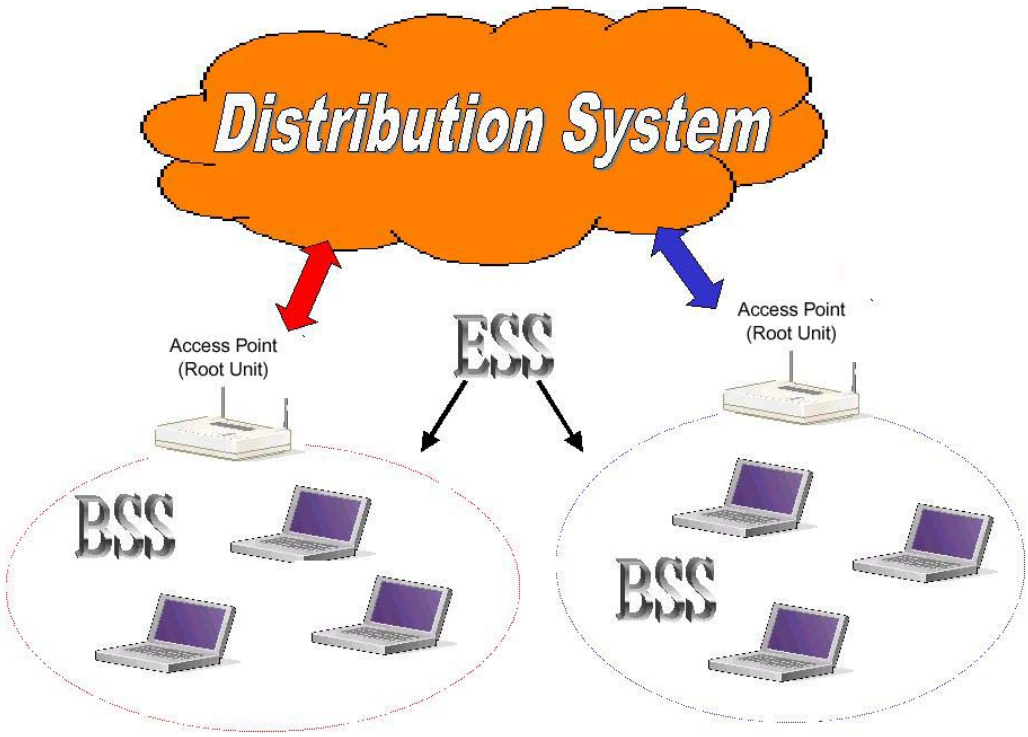


**Figura 4.8**: architettura di una infrastructure local area network.

#### *4.1.2.2.1 Associazione*

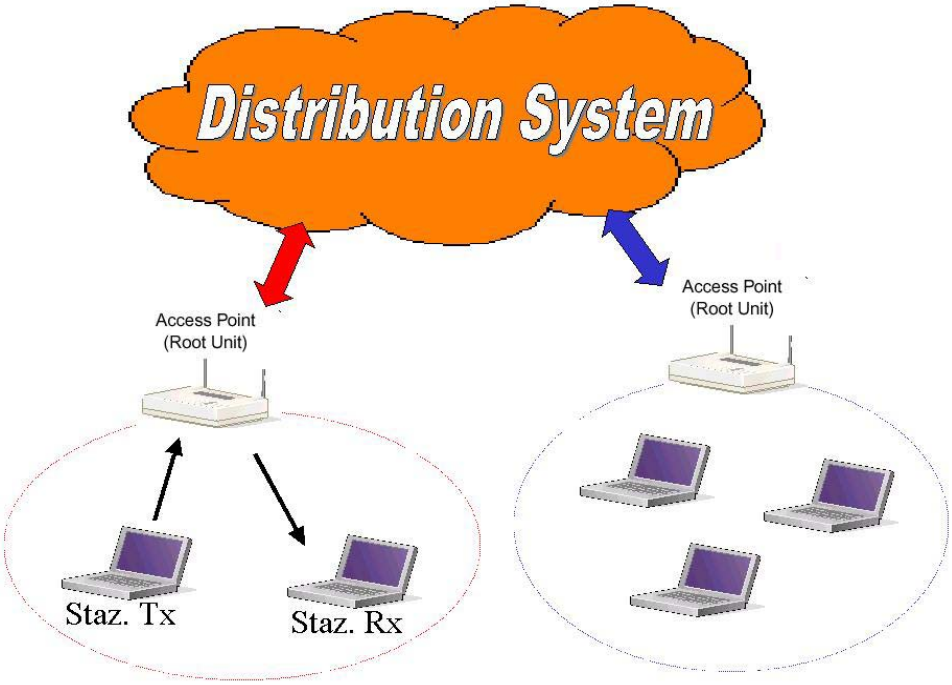


**Figura 4.9**: trasmissione di un frame all'interno dello BSS.

Di primaria importanza per lo standard 802.11 è il concetto di *associazione* (*association*): l'associazione è il meccanismo attraverso il quale lo standard fornisce una mobilità trasparente alle stazioni. Quando un terminale mobile richiede di connettersi ad una WLAN, invia una richiesta di associazione (*association request*) ad un AP: tale richiesta include informazioni riguardo le proprietà della stazione, come il supporto alla crittografia, le velocità di trasmissione che supporta ed altre informazioni. Le politiche e gli algoritmi utilizzati dagli AP per decidere se accettare o no le richieste di associazione non vengono descritti dallo standard: quando l'AP risponde (*association response*), comunica alla stazione informazioni riguardanti il suo stato, in particolar modo specificando l'esito della richiesta e, in caso di rifiuto, il motivo.

Se la richiesta di associazione ha esito positivo, a partire da quel momento l'AP è responsabile di tutto il traffico di rete prodotto dal terminale in questione: tutti i frame che la stazione invierà, dovranno passare attraverso l'AP relativo. Se un frame inviato dal terminale ha come destinazione un punto all'interno del BSS, l'AP non farà altro che trasmettere i dati al destinatario, come mostrato in figura 4.9.

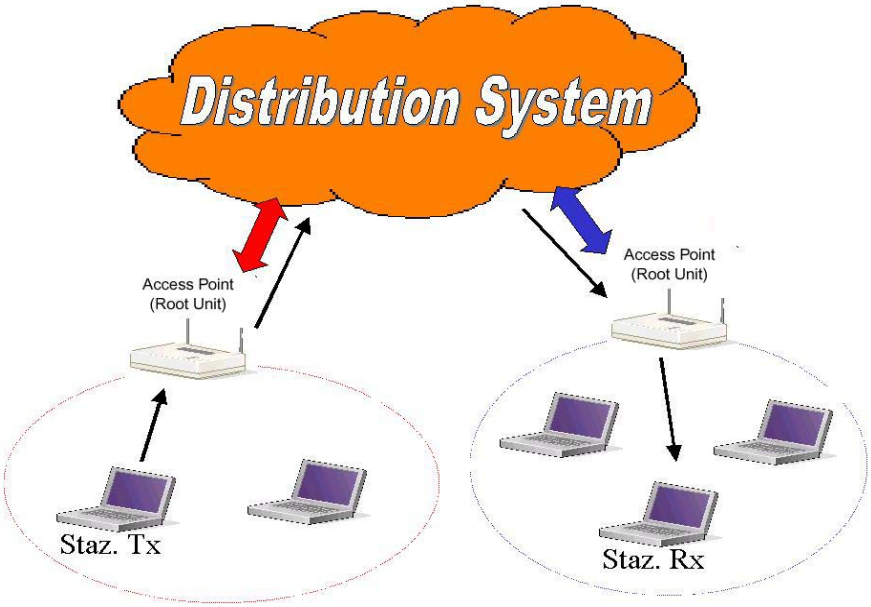


**Figura 4.10**: trasmissione di un frame ad un BSS differente.

Se il frame ha invece come destinazione una stazione appartenente ad un altro BSS , l'AP invierà, tramite il DS, i dati all'AP corrispondente al BSS destinatario, il quale, a sua volta, consegnerà i dati alla stazione interessata, si veda la figura 4.10.

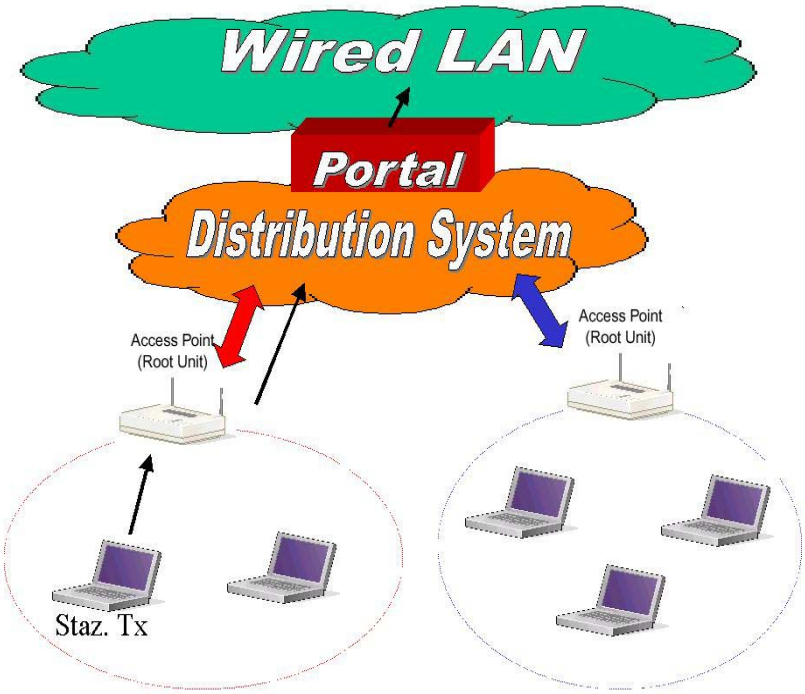


**Figura 4.11**: trasmissione di un frame ad una wired LAN.

Infine, se il destinatario è un'entità esterna all'ESS, allora il frame, tramite il DS, giungerà ad una nuova entità, chiamata *portal*, che funge da ponte tra una rete 802.11 ed un'altra rete wired.

Più concretamente, un'associazione può essere identificata dalla seguente coppia:


**Figura 4.12**: elementi costituenti un'associazione.

dove il "*MAC address stazione*" coincide con l'indirizzo MAC associato ad una stazione e il "*BSS ID*" coincide con l'indirizzo MAC associato all'AP rappresentante il BSS in questione. In questo modo un'associazione identifica univocamente una postazione all'interno di una rete wireless.

Basandosi sul concetto di associazione appena descritto, lo standard propone due tipi di servizi: gli *station services* e i *distribution services*.

#### *4.1.2.2.2 I servizi di stazione*

Tali servizi devono essere forniti da una qualunque postazione mobile: elenchiamo di seguito gli station services:

- authentication;
- deauthentication;
- privacy;
- data delivery.

A livello di MAC vengono utilizzati alcuni frame per implementare tali servizi: il processo di *authentication* è necessario in quanto in una rete wireless non si conoscono a priori i partecipanti , dunque, quando una stazione mobile vuole intraprendere una comunicazione con un altro terminale, deve precedentemente autenticarsi. Tale fase di autenticazione avviene mediante lo scambio di tre frame (*authentication algorithm number, auyhetnication transaction sequence number* e *status code*); ci sono due algoritmi di autenticazione:

- **open system authentication:** questo metodo di autenticazione è piuttosto debole e consiste nella presentazione reciproca delle due stazioni, senza contemplare una fase di verifica:
- **shared key authentication:** in questo caso viene usato un algoritmo che adopera la criptazione WEP (Wired Equivalence Privacy), di cui parleremo in seguito. Il protocollo utilizzato è riconducibile ad uno dei tradizionali protocolli "a sfida", in cui si utilizza una chiave di sessione condivisa per criptare un testo di sfida: se il processo di decriptazione ha successo, allora l'autenticazione ha esito positivo.

Il servizio di *deauthentication* viene erogato qualora una postazione richieda ad un'altra postazione di autenticarsi nuovamente: tale processo richiede l'invio di un solo frame, rappresentante la ragione della deautenticazione.

Per quanto riguarda il servizio di *privacy*, lo standard 802.11 si pone come obiettivo di raggiungere un livello di sicurezza comparabile a quello che si può raggiungere in una tradizionale wired LAN: per raggiungere tale scopo, si usa il meccanismo WEP (Wired Equivalency Privacy), che si appoggia su un algoritmo di criptazione RC4, sistema di criptazione a chiave simmetrica di lunghezza variabile.

Lo standard propone solamente l'uso degli algoritmi appartenenti alla famiglia RC4, senza dettare specifiche sulle metodologie di distribuzione e negoziazione delle chiavi: tale problematica non è quindi presa in considerazione dallo standard.

#### *4.1.2.2.3 I servizi di distribuzione*

I distribution services offrono quelle funzionalità per consentire alle stazioni mobili di muoversi liberamente all'interno di un ESS ed inoltre permettono alla WLAN di connettersi ad una wired LAN; riportiamo di seguito quali sono i servizi messi a disposizione da questo livello:

- **association service:** consente una connessione logica (*association*) tra una stazione mobile ed un AP. Tale connessione serve al DS per sapere come e dove inoltrare i dati verso una stazione mobile; la connessione logica inoltre è importante anche per gli stessi AP, i quali devono accettare dati dalle postazioni mobili e riservare risorse per supportare le stazioni wireless.

L'association service è invocato una sola volta, quando una stazione entra per la prima volta all'interno della wireless LAN. Menzioniamo che l'association service può essere espletato solo dopo aver eseguito l'authentication service;

- **reassociation service:** questo servizio include informazioni riguardo l'AP presso cui una stazione mobile si è associata. Tale servizio è invocato ogni qualvolta una stazione cambi BSS all'interno del proprio ESS (operazione di *roaming*): in tale situazione, gli AP corrispondenti devono coordinarsi e aggiornare il proprio stato circa i terminali ad essi associati. In figura 4.12 viene mostrato il processo di roaming di una postazione.

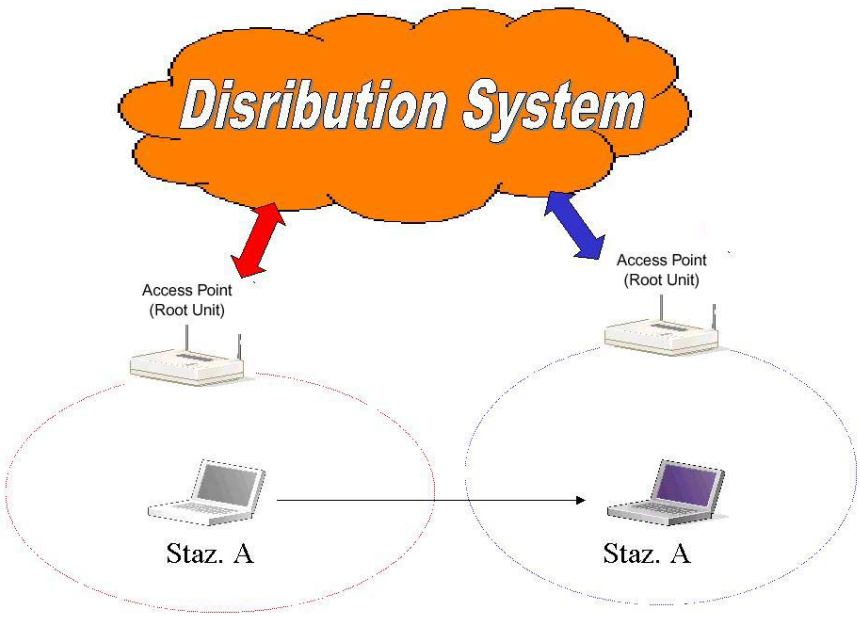


**Figura 4.12**: roaming di una stazione da BSS a BSS.

- **deassociation service:** con questo servizio viene a tutti gli effetti cancellata l'associazione esistente tra un AP ed una stazione mobile. Ciò può dipendere da una volontà propria della stazione di abbandonare l'AP, oppure può dipendere da una prolungata inattività da parte della stazione;
- **distribution service:** questo servizio è invocato dall'AP per determinare se un particolare data frame è destinato ad un stazione appartenente al BSS di competenza oppure è destinato ad un BSS esterno. In tal caso il frame è

inoltrato verso il DS, che provvederà a redirigere il pacchetto in questione all'appropriato AP;

- **integration service:** tale servizio è richiesto qualora vi sia interazione tra una wireless LAN e una wired network. Mediante esso i frame 802.11 vengono traslati in frame con formato conforme alla wired network in questione e viceversa: così facendo, garantiamo interoperabilità tra reti di tipologia differente.

#### *4.1.2.2.4 Sincronizzazione*

All'interno di una infrastructure local area network ci può essere la necessità di mantenere sincronizzazione tra i vari terminali appartenenti ad uno stesso BSS: per far ciò, le stazioni si sincronizzano su un clock comune, quello dell'AP corrispondente al BSS, tramite un semplice meccanismo [WIRN]. Periodicamente l'AP invia ai terminali ad esso associati un particolare frame, detto *Beacon frame*. Al ricevimento di tale dato, le stazioni aggiornano il loro clock logico, sincronizzandolo con quello dell'AP: bisogna considerare però che la trasmissione effettiva del beacon frame potrebbe essere ritardata a causa di traffico sulla rete o eventuale collisione dello stesso. In tal caso i clock logici degli AP e delle stazioni non sarebbero perfettamente allineati.

Il Beacon frame di cui parlato trova interesse anche nel concetto di *scanning*: ricevendo questi frame, le stazioni percepiscono gli AP presenti attorno a loro, ottenendo conoscenza sull'ambiente circostante. In questo modo possono scegliere un particolare AP tra quelli che hanno inviato dei Beacon frame cui associarsi: questa modalità di percepire gli AP presenti è nota come ***passive scanning***.

D'altra parte vi è la possibiltà di effettuare un ***active scanning***, con il quale una stazione mobile trasmette un frame (*Probe request*) per trovare gli AP presenti nell'area circostante: gli AP raggiungibili, in risposta, invieranno una *Probe response*. Il passive scanning fornisce prestazioni inferiori, ma consente un risparmio energetico rispetto all'active scanning: esiste quindi un trade-off prestazione/consumo.

#### *4.1.2.2.5 Power saving*

Le reti wireless sono tipicamente relazionate ad apparati mobili, provvisti di una batteria interna soggetta ad inevitabile esaurimento: proprio per questa ragione, lo standard 802.11 definisce un completo meccanismo che consente alle

stazioni di entrare in una modalità operativa a basso consumo (***sleeping mode***) per lunghi periodi di tempo, senza perdita di informazioni.

L'idea alla base di questo meccanismo propone di preservare risorse per ogni stazione in sleeping mode all'interno dell'AP: quindi l'AP bufferizza e mantiene tutti i dati indirizzati verso un determinato terminale mobile finché quest'ultimo non ritorni in modalità di lavoro normale oppure non faccia esplicita richiesta di scaricare dall'AP i dati memorizzati.

Periodicamente l'AP invia alle stazioni in sleeping mode dei frame, per portarle informazioni circa l'arrivo di frame ad esse indirizzate: sulla base di ciò, un terminale mobile potrà essere stimolato a scaricare i dati ad esso relativi.

## *4.2 Wireless API*

Queste API offrono al programmatore la possibilità di conoscere le impostazioni di una scheda wireless, configurarne i principali parametri nonché ottenere un feedback circa lo stato di funzionamento, ma realizzano anche uno strumento con cui accedere ad alcuni servizi di basso livello (il roaming e la scansione dell'ambiente circostante costituiscono un esempio in tal senso) solitamente visibili unicamente al driver del dispositivo. Come già sottolineato più volte lo sviluppo di un multimedia middleware trarrà beneficio dalla maggiore espressività resa disponibile da tali API.

### *4.2.1* Wireless API in Linux e Windows

Per il momento sono disponibili due implementazioni delle Wireless API, una sviluppata per le piattaforme Linux (si veda [BEM02]), che si appoggia sulle Wireless Extension, ed un'altra per la piattaforma Windows (si veda [FEA02]), che si appoggia sulle API stabilite dall'interfaccia NDIS (Network Device Interface Specification) e sulla libreria NDISUIO (Network Device Interface System User-mode I/O), sviluppando al di sopra di esse uno strato che offre tutti i servizi offerti dalle Wireless Extensions di Linux utilizzati nell'implementazione delle Wireless API. Si riporta un confronto delle due architetture realizzate in figura 4.13, dove nella parte destra, ove viene considerata l'architettura utilizzata per la piattaforma Windows, non viene messo in evidenza lo strato finale che interfaccia le Windows Wireless Extension con il mondo Java, cosa che viene invece evidenziata nella parte sinistra dell'immagine.

Il progetto *Linux Wireless Extensions* si occupa proprio di fornire un insieme di procedure per l'interfacciamento e la comunicazione con i driver di dispositivi wireless fisici, anche prodotti da case diverse, fornendo una interfaccia

comune; è, inoltre, in grado di reperire un certo numero di informazioni sullo stato della comunicazione, nonché settare varie opzioni del dispositivo stesso.

Il *Network Driver Interface Specification (NDIS)* è una libreria facente parte delle DDK (Driver Development Kit) di Microsoft per Visual Studio per standardizzare lo sviluppo di network driver per Windows ed avere un'interfaccia comune alla quale il sistema operativo possa accedere. Le maggior parte delle schede di rete wired/wireless al momento sul mercato (tra cui anche la Cisco Arionet 350 utilizzata nel progetto) implementano driver compatibili con le specifiche *NDIS* e sono quindi accessibili le funzionalità messe a disposizione da questa interfaccia: questa libreria permette di estrarre e gestire dati e funzionalità a partire dall'hardware delle schede di rete fino al livello network driver.

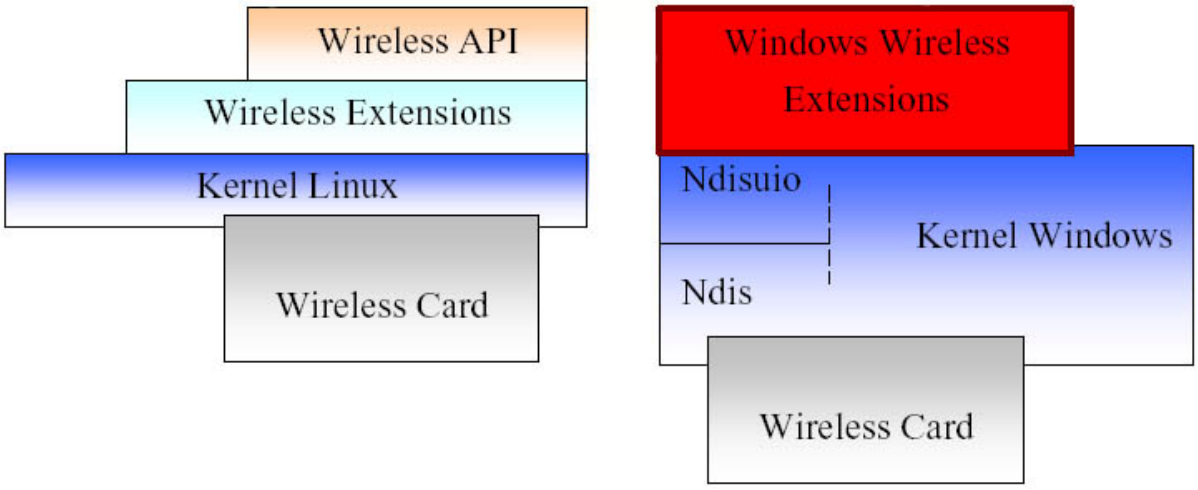


**Figura 4.13**: confronto fra le architetture per la realizzazione delle Wireless API in Linux e Windows

Non vogliamo scendere in questa sede nei dettagli di implementazione delle Wireless API per le due piattaforme, sottolineiamo solamente il fatto che grazie ad esse il middleware , che verrà sviluppato in Java, potrà essere eseguito al di sopra di entrambe le piattaforme.

### 4.2.2 Interfaccia esposta dalle Wireless API

L'intefaccia WirelessInterface mette a disposizione i seguenti metodi, invocabili da un qualsiasi programma Java:

- ***elencoBSSDisponibili***: restituisce l'elenco degli identificatori associati ai BSS, ovvero le celle presenti nell'ambiente circostante il terminale;
- ***getBSSID***: fornisce l'identificativo della cella di cui fa attualmente parte la postazione;
- ***getESSID***: restituisce il valore del parametro (denominato secondo lo standard IEEE 802.11, Extended Service Set) di configurazione utilizzato

per caratterizzare l'insieme delle celle appartenenti ad una stessa rete mobile;

- ***setAutomaticRoaming***: attiva (disattiva) il meccanismo di roaming automatico; il driver di un dispositivo wireless può automaticamente attivare il roaming all'occorrere di determinati eventi (tra cui: potenza del segnale utile percepito al di sotto di una certa soglia, perdita di sincronizzazione, ecc.) oppure per sottostare ad una certa politica di gestione (come, ad esempio, il loadbalancing, che ha come obiettivo la distribuzione uniforme delle postazioni mobili tra le diverse celle costituenti le rete, in modo da assicurare a ciascuna di esse il massimo della banda); qualora il roaming automatico venga disabilitato, l'ingresso in una nuova cella potrà avvenire unicamente attraverso l'invocazione del metodo *setBSSID;* tale metodo non è per ora implementato per le architetture Windows;
- ***setBSSID***: forza l'operazione di roaming; in particolare, il metodo obbliga il driver della scheda wireless ad approntare un'interazione con la stazione base responsabile della cella indicata al momento dell'invocazione, al fine di consentirvi l'ingresso del terminale mobile; spesso, la sua invocazione è preceduta dalla chiamata al metodo *elencoBSSDisponibili*, in modo da individuare l'insieme delle celle tra cui selezionare quella oggetto della fase successiva di roaming;
- ***setESSID***: imposta il parametro di configurazione omonimo;
- ***setStationName***: assegna un nome logico (*nick name*) alla postazione; tale metodo non è per ora implementato per le architetture Windows;

## *4.3 Conclusione*

In questo capitolo abbiamo analizzato le tecnologie che verranno utilizzate nello sviluppo della parte fruibile in ambiente wireless dell'applicazione. In particolare abbiamo dapprima introdotto lo standard 802.11 utilizzato per la realizzazione della rete wireless e le Wireless API, un'interfaccia per dare visibilità all'applicativo sviluppato in Java dell'Infrastructure Local Area Network.

CAPITOLO 5

# 5 Analisi del middleware per la gestione dei flussi multimediali su rete fissa e mobile

In questo capitolo si presenta l'architettura del multimedia middleware proposto. Nei capitoli precedenti abbiamo analizzato diversi tipi di multimedia middleware, rilevando quali siano i requisiti architetturali che sottendono la costruzione di questi prodotti software e i meccanismi implementati per rispondere a tali requisiti, mettendo in rilievo pregi e carenze delle diverse soluzioni. L'architettura proposta è una personale risposta ad alcune delle problematiche rilevate.

Nella sezione seguente si considerano i requisiti che hanno guidato lo sviluppo dell'architettura proposta. Il resto del capitolo è dedicato all'analisi del sistema e alla presentazione della sua architettura logica. Questo capitolo non entrerà nei dettagli di progetto, si vuole infatti che l'architettura logica proposta sia un'architettura astratta che fissa quali siano i punti focali che guidano il progetto, le interazioni fra le parti e la suddivisione del sistema in sottosistemi e in parti. Non interessano cioè a questo livello dettagli tecnologici, che verranno analizzati nel prossimo capitolo.

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 ad un certo linguaggio o ad una certa tecnologia, per concentrarsi unicamente sulle reciproche interazioni fra le parti e per facilitare apertura e reingegnerizzazione futura del prodotto.

## *5.1 Requisiti*

### 5.1.1 Scenario

Si vuole supportare la pubblicazione e fruizione di **presentazioni multimediali** di varia natura e qualità diverse su rete fissa e mobile. Col termine "presentazione multimediale" intendiamo un aggregato di uno o più oggetti multimediali dotato di un attributo di qualità. Per **oggetto multimediale** si

intende l'astrazione di un'istanza di un certo tipo di dato multimediale all'interno del sistema, è cioè un descrittore di tale istanza. Ogni oggetto multimediale incapsula un solo tipo di media, cioè può essere solo di tipo audio, video, ecc... A ciascun oggetto multimediale corrisponderà un file, giacché siamo interessati alla condivisione di presentazioni multimediali già registrate e non alla condivisione "in diretta", dove per esempio potrebbe non esistere neppure nel sistema un file ove sia salvato per intero tutto l'oggetto, ma esso potrebbe essere solo un flusso continuo di dati spediti a uno o più destinatari.

La **presentazione multimediale** è un metadato che viene utilizzato per aggregare diversi oggetti multimediali specificando anche la qualità della presentazione (intesa come qualità dell'aggregato). La presentazione multimediale conterrà anche un attributo che ne identifichi la posizione (come aggregato) all'interno del sistema. Consideriamo per esempio un film, cioè un aggregato di un oggetto multimediale di tipo audio e di uno di tipo video. Per lo stesso film potremmo avere nel sistema tre presentazioni multimediali diverse. Solitamente, all'aumentare della qualità aumentano le risorse richieste per poter fruire della presentazione stessa.

Il **contenuto multimediale** è un'astrazione introdotta per partizionare le presentazioni multimediali in sottoinsiemi mutuamente esclusivi. In ciascun sottinsieme saranno presenti le presentazioni multimediali che portano lo stesso contenuto informativo, vale a dire che le presentazioni multimediali che presentano la stessa struttura, intesa come struttura dell'aggregato, e lo stesso titolo. Naturalmente le presentazioni multimediali saranno differenziate per qualità, anche se notiamo fin d'ora che nello stesso sottoinsieme potrebbero esserci più presentazioni multimediali con la medesima struttura e qualità. Definiamo **titolo** l'identificatore del contenuto multimediale all'interno di tutto il sistema.

Gli **utilizzatori** del sistema potranno **richiedere il delivery** dei titoli presenti all'interno del sistema distribuito, **comandare** la presentazione, attraverso una interfaccia simile a quella di un video registratore, **richiedere lo spostamento della sessione** verso un terminale diverso da quello che stanno attualmente utilizzando. Quando l'accesso avvenga da rete mobile, cioè attraverso un device di tipo wireless, si vuole che l'infrastruttura supporti il delivery delle presentazioni, **riorganizzandosi per seguire i movimenti dell'utente**.

L'infrastruttura in progetto deve inoltre occuparsi della **gestione delle risorse** supportando prenotazione e monitoraggio delle stesse, nonché della

**scelta della presentazione** più adatta alla piattaforma computazionale dalla quale un certo utente sta accedendo al sistema. Le caratteristiche delle diverse piattaforme computazionali saranno salvate in dei descrittori, e i profili utenti manterranno delle coppie con locazione della piattaforma dalla quale si accede e riferimento al descrittore della piattaforma. Da ultimo si vuole fare in modo che sia possibile, avendo installato le parti basilari dell'infrastruttura, supportare **il downloading e l'inizializzazione di software a runtime**.

Il resto della sezione metterà in evidenza i requisiti che hanno guidato lo sviluppo dell'architettura.

### 5.1.2 Fruizione del materiale multimediale

L'architettura sarà realizzata in ambiente distribuito. Bisognerà perciò prima di tutto decidere quale modello computazionale adottare. La scelta fatta è stata quella di integrare il tipico modello client-server col modello ad agenti mobili, cercando di utilizzare il meglio di questi due approcci. In particolare si è deciso di introdurre quattro entità: ClientAgent, Client, ProxyAgent e Server. Esse saranno tra loro collegate a formare un percorso che va dal client verso il server, attraversando uno o più proxy. Chiameremo tale percorso **path**. Prima di considerare più in dettaglio i requisiti di queste entità, si vuole sottolineare che particolare attenzione è stata posta nel decidere quali entità dovessero essere realizzate come **entità fisse** e quali come **agenti mobili**. Si sottolinea questo aspetto perché, avendo a disposizione una tecnologia che supporta gli agenti mobili, una delle prime idee è stata quella di realizzare tutte le entità elencate sopra come agenti mobili. Ad una valutazione più attenta si è però capito che questo potente modello computazionale va utilizzato solo là ove ve ne sia necessità e non in modo generalizzato. Ogni qualvolta introduciamo un agente mobile verrà sottolineato il perché della scelta di questo modello computazionale.

- **ClientAgent**: è l'entry point del sistema dal lato client. È realizzato come agente mobile. Il compito del ClientAgent è quello di inizializzare la sessione per l'utilizzatore e gestire poi le interazioni col resto del middleware distribuito. Possiamo dire che è una sorta di *mediator* (con questo termine si fa riferimento al mediator pattern, si veda [MedPat]) che accetta le richieste fatte dall'utente attraverso un'opportuna GUI, per interagire col middleware in progetto e richiedere l'esecuzione dell'operazione richiesta. Questa entità verrà realizzata come agente per facilitare l'interazione con gli altri agenti, e

per permettere il movimento di questa entità, in modo da supportare gli utenti che si muovono fra diversi terminali fissi (utenti nomatici), muovendosi là dove essi si vogliano muovere.

- **Client**: il client è l'end-point che riceve i flussi multimediali richiesti. Non si vuole per ora entrare nei dettagli di progettazione del client, ma si vuole sottolineare che per sua natura il client appare come un'entità fissa sulla macchina dalla quale l'utente effettua l'accesso al sistema. Il fatto che il client non sia un agente non implica poi che il software richiesto per l'esecuzione del client stesso debba già essere presente sulla macchina dalla quale il client verrà lanciato, potrà infatti essere scaricato a runtime, seguendo il paradigma COD.
- **Server**: il discorso che si fa per il server è molto simile a quello fatto per il client. Infatti anche in questo caso si è optato per la realizzazione del server come entità fissa scaricabile secondo necessità (COD). Inoltre un certo tipo di server, una volta che sia stato lanciato su una certa macchina, verrà utilizzato per servire tutte le richieste che da quel punto in poi arriveranno a quella macchina per quel tipo di server. Come è facilmente intuibile, il Server è l'altro end-point della comunicazione dei flussi multimediali: è cioè colui che spedisce i flussi multimediali richiesti.
- **ProxyAgent**: per quello che riguarda il proxy, esso è stato realizzato come agente mobile, in modo da supportare il movimento dei terminali e verrà dettagliato più sotto (vedi punto 5.1.4). Per la gestione vera e propria dei flussi il Proxy Agent si avvarrà di una entità che chiameremo **Proxy**.

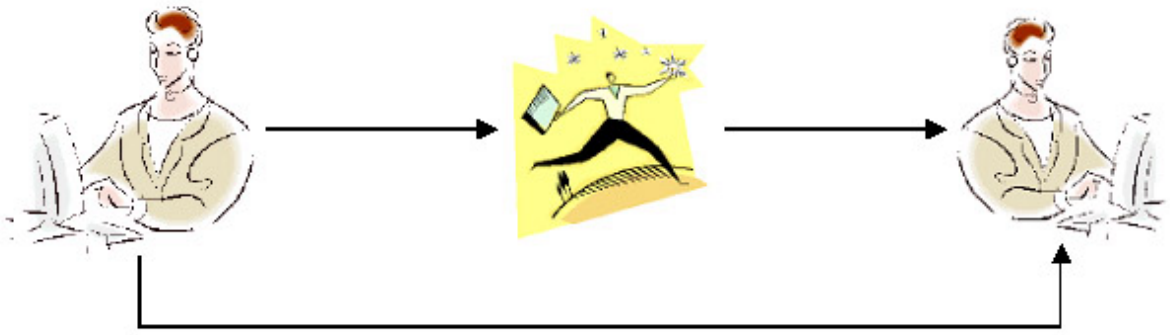

**Figura 5.1**: scenario di utilizzo del sistema per utilizzatori nomatici.

Lo sviluppo dell'applicazione per la fruizione dei contenuti multimediali sarà basato su specializzazioni di queste quattro entità.

### 5.1.3 Movimento degli utenti

L'architettura dovrà supportare il movimento dei clienti fra diversi terminali. Un utente può cioè esprimere il desiderio di muoversi. L'infrastruttura in risposta a questa richiesta deve muovere la sessione dell'utente verso il nuovo terminale. Per supportare questa funzionalità dovremo quindi costruire un supporto che risponda ai principi di ***location awareness*** e ***context awareness*** (vedi capitolo 1).

La figura 5.1 mostra lo scenario che si è appena delineato a parole. L'utente inizia la sessione su un terminale, poi richiede lo spostamento della sessione su di una altro terminale, verso il quale si sposta fisicamente. In modo trasparente all'utente stesso l'infrastruttura provvederà allo spostamento della 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. Gli utenti che compiono questo tipo di movimento vengono solitamente detti **nomadic users**, e sono contrapposti ai **roaming users**, che sono invece quegli utenti che si muovono insieme al dispositivo che stanno attualmente utilizzando. I ***requisiti*** richiesti per il supporto della mobilità utente sono i seguenti:

- sarà necessario offrire un'astrazione di località, si vogliono infatti rendere le applicazioni sviluppate al di sopra del middleware *location aware* (consapevoli della località). Verrà perciò introdotto un **servizio di località** che, interrogato, ritorni un identificatore della località nella quale ci si trova;
- bisogna fare in modo che il supporto possa reperire informazioni circa i vari terminali cui accederà l'utente, per gestire l'hand-off della sessione tra due terminali. Si rende quindi necessario un **servizio per la gestione dei profili** degli utenti. In particolare, ogni **profilo** utente, oltre a contenere dati e preferenze dello stesso, conterrà un elenco dei terminali dai quali egli può accedere al sistema, e la loro locazione espressa in termini dell'astrazione di località introdotta dal servizio di località.
- bisogna **gestire l'hand-off** fra i due terminali. A tale scopo verrà utilizzato un agente mobile che, spedito alla macchina verso la quale l'utente si vuole muovere, si occuperà dell'inizializzazione delle entità necessarie per la fruizione del materiale multimediale e passerà il controllo a tali entità, che, coordinandosi con i propri pari sulla macchina dalla quale si è partiti,

finiranno la gestione dello spostamento della sessione, richiamando sulla nuova località il ClientAgent.

### 5.1.4 Movimento dei terminali

Si vuole inoltre supportare la mobilità dei terminali. Ipotizzando cioè che il cliente si muova insieme al proprio terminale, si vuole assistere il movimento del cliente, facendo in modo che l'infrastruttura si modifichi a tempo di esecuzione, in modo da seguire i movimenti dell'utente.

Si deve dunque rispondere, anche in questo caso, ai principi di ***location awareness*** e ***context awareness***. Quando un cliente si muove bisogna che il terminale mobile interagisca con la rete fissa, in modo da richiedere il movimento del proxy, dalla località dalla quale l'utente si è mosso alla località verso la quale l'utente si sta muovendo. E' necessaria quindi una conoscenza dell'ambiente circostante che come visto al capitolo precedente è disponibile al livello applicativo grazie all'uso delle Wireless API. Questo scenario è mostrato in figura 5.2.

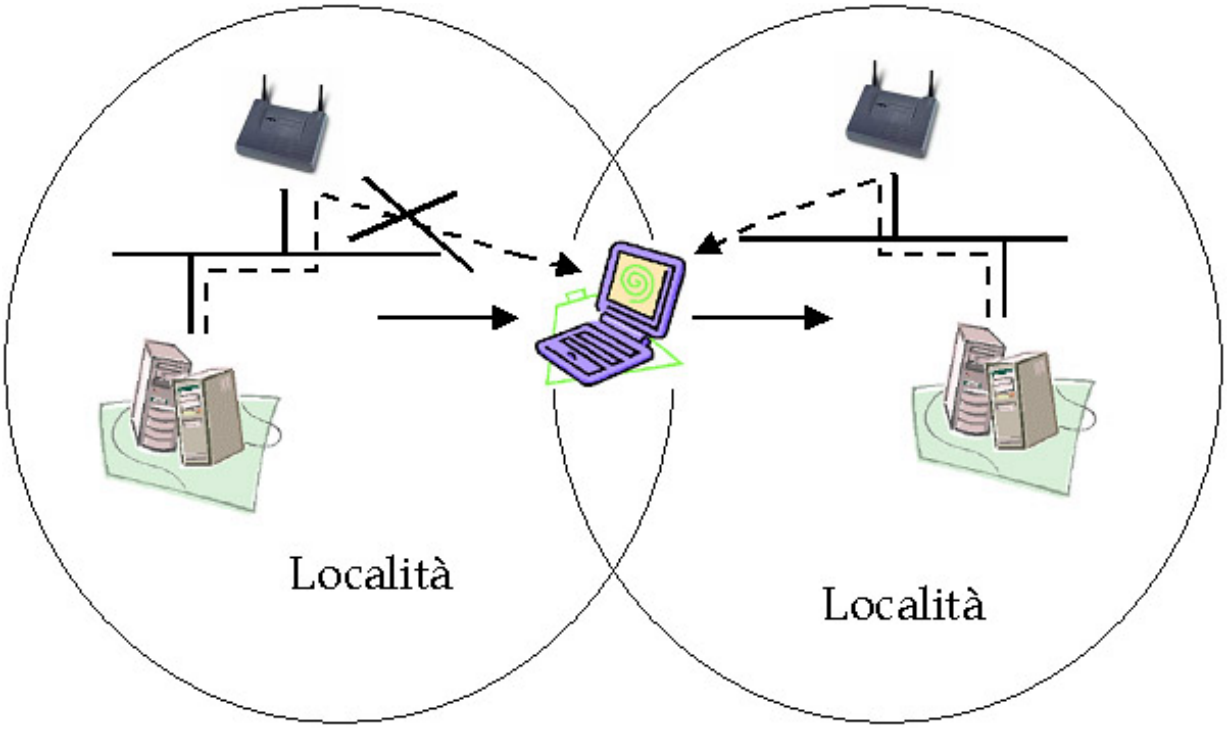


**Figura 5.2:** roaming users, scenario di utilizzo del sistema

L'utente si muove insieme al proprio terminale, e l'infrastruttura, interagendo col terminale stesso, al verificarsi del movimento si modifica in modo da effettuare lo streaming non più dalla vecchia località, ma dalla nuova

località. Si vuole che il tutto avvenga in modo trasparente nei confronti dell'utente finale. I ***requisiti*** in questo caso sono i seguenti:

- occorre che il terminale mobile possa capire quando entra nella nuova località. Inseriamo perciò un **servizio di notifica di ingresso in una nuova località**. Tale servizio, residente sul terminale mobile, all'ingresso nella nuova località notificherà l'evento a tutte le entità che si sono registrate presso di lui. Naturalmente, le astrazioni di località dipenderanno direttamente dalla tecnologia utilizzata per la realizzazione della rete wireless;
- occorre che l'infrastruttura si modifichi in modo da seguire i movimenti del terminale mobile. Si introduce perciò nel path, come ultimo elemento prima del client, il **ProxyAgent**, che sarà un agente mobile e si muoverà sulla rete fissa seguendo i movimenti del terminale nella rete wireless. Tale entità viene introdotta per i seguenti motivi:
    a) perché, anche in un'ottica di futuri sviluppi, il proxy potrebbe incapsulare tutti quei cambiamenti che si rendessero necessari per far dialogare il client mobile con il resto dell'infrastruttura sulla rete fissa. Ad esempio il protocollo utilizzato dal client, per la comunicazione nella rete wireless, potrebbe non essere necessariamente il TCP/IP, e in futuro potrebbero venire introdotti nuovi protocolli. Il proxy agisca da disaccoppiatore fra il client e l'infrastruttura sulla rete fissa garantendo *flessibilità* e *modularità* per futuri sviluppi riguardanti soprattutto il lato client;
    b) per trasformare (qualora ce ne fosse bisogno) i dati multimediali in arrivo. I ragionamenti fatti al punto a) possono infatti essere applicati anche ai formati dei dati trasmessi (tipicamente audio e video) richiesti dalla piattaforma software per lo streaming multimediale presente sul terminale mobile. Potrebbe infatti essere necessario un adattamento dei dati multimediali, in particolare quando essi vengano trasmessi ad un terminale mobile con basse capacità computazionali;
- bisogna che il terminale mobile acquisisca informazioni circa la nuova località nella quale è appena entrato. Si introduce perciò un **target localizer** (da non confondere col servizio di località introdotto sopra). Tale servizio viene interrogato, quando si entri in una nuova località, per scoprire la locazione della macchina che, nella località obiettivo verso cui ci si sta muovendo, può accettare l'agente mobile proxy che sta migrando. In pratica cioè si richiede quale macchina nella nuova località ospiti il middleware per il supporto degli agenti mobili.

### 5.1.5 Inizializzazione e riorganizzazione dinamica del sistema

Si vuole inoltre offrire supporto per l'inizializzazione e la riorganizzazione dinamica del sistema. Quando si inizia una sessione di streaming, dapprima (se non è già presenti) verrà scaricato il software necessario ad effettuare la trasmissione stessa, poi verrà configurato il path dal server al client, dopodiché verrà iniziata la trasmissione vera e propria.

Questo servizio è stato incluso nel supporto perché si vuole realizzare un middleware che possa rispondere il più possibile ai principi di ***modularità*** e ***flessibilità*** considerati nel capitolo 1. Come abbiamo visto nel capitolo 2, molti dei moderni multimedia middleware sono basati sui componenti. Il sistema oggetto di questo lavoro di tesi, sebbene non sia basato su componenti, tuttavia è stato progettato tenendo presenti come linee guida due concetti fondamentali che stanno alla base della progettazione a componenti.

Il primo è la ***progettazione per interfacce***, cioè i vari oggetti interagiscono tra di loro attraverso interfacce ben definite. Il secondo è lo sviluppo del servizio descritto sopra che si occupi di scaricare e attivare le entità necessarie per lo svolgimento dello streaming. Non si è sviluppato un vero e proprio ***container***, inteso come ambiente che ospita e supporta il ciclo di vita dei vari componenti; nondimeno ci si è ispirati a questo modello di progettazione facendo in modo che la fase di inizializzazione del sistema possa prevedere downloading e attivazione dei diversi oggetti che si trovano sul percorso dal server al client. Il sistema di inizializzazione deve rispondere ai seguenti ***requisiti***:

- bisogna che il sottosistema di inizializzazione sia in grado di scaricare e inizializzare le varie entità che partecipano al multimedia streming lungo tutto il path, ma senza conoscere a priori tali entità, che saranno realizzate dallo sviluppatore dell'applicazione, rispondendo a delle interfacce prestabilite;
- bisogna perciò introdurre un oggetto che incapsuli un **piano di inizializzazione**. Ciascun specifico piano di inizializzazione sarà una specializzazione di tale entità;
- bisogna conseguentemente introdurre un **interprete dei piani di inizializzazione**;
- la riorganizzazione del sistema viene supportata mediante la generazione di più piani di inizializzazione alternativi; a runtime il sistema, utilizzando tali piani, potrà riorganizzarsi;
- bisogna introdurre un **decisore** che interrogato possa ritornare il piano di inizializzazione. Il decisore è una funzione che prende in ingresso una lista di

presentazioni rispondenti al titolo richiesto dall'utente, la locazione da cui l'utente accede al sistema, ritornando un piano di inizializzazione.

### 5.1.6 Gestione delle risorse

Si è deciso di includere nel sistema un sottosistema di gestione delle risorse che attui entrambi gli approci pro-attivo e reattivo (vedi capitolo 1). Si ritiene infatti necessario prenotare preventivamente le risorse, negoziando inizialmente la qualità di servizio con una entità che si occupi della ***prenotazione*** delle risorse, e poi monitorando l'andamento della sessione, rinegoziando eventualmente a tempo di esecuzione la qualità di servizio. Ci si potrebbe chiedere perché introdurre supporto per il ***monitoraggio***, infatti se tutte le richieste di risorse passassero attraverso il middleware il monitoraggio stesso sarebbe inutile, dal momento che tutta la negoziazione iniziale servirebbe a fissare una qualità di servizio che verrebbe poi mantenuta, in modo pro-attivo. Dal momento che però ciò non è possibile, poiché tutti gli applicativi sviluppati direttamente al di sopra del sistema operativo, continueranno ad accedere direttamente alle risorse attraverso le opportune chiamate di sistema, si è ritenuto opportuno includere supporto per il monitoraggio.

Si è deciso di concentrare l'attenzione su due risorse, che vengono considerate come le più sensibili ai fini della fruizione di contenuti multimediali: ***banda trasmissiva*** e ***CPU***. L'altra risorsa particolarmente sensibile è la memoria, ma solitamente quest'ultima costituisce un vincolo meno stringente. Si vuole inoltre notare che, effettuando una gestione delle risorse di tipo multidimensionale (cioè considerando molte risorse insieme), aumentano sia la complessità della realizzazione del sottosistema di gestione delle risorse, sia l'overhead dovuto al monitoring. Per questi motivi si è deciso di circoscrivere la gestione delle risorse alle due considerate sopra.

Una decisione importante da prendere a questo livello è anche quella di stabilire come definire le qualità di servizio e come debbano essere adattate le sessioni in corrispondenza del verificarsi di un degrado delle risorse. Le scelte possibili sono due. La prima è quella di definire due valori che rappresentino il minimo e il massimo della qualità, e poi effettuare degli adattamenti fra questi due valori. Agendo in questo modo si opera una gestione della qualità di servizio che agisce in un dominio continuo. La seconda scelta possibile è invece quella di definire diversi livelli di qualità. In questo caso la gestione della qualità di servizio va ad agire in un dominio discreto, nel quale si "salta" da una qualità ad un'altra in modo discontinuo. In questo lavoro di tesi si propone una soluzione

che va in questa seconda direzione. I perché di questa scelta sono due. Il primo è che una gestione delle risorse che agisca nel continuo e quindi ad un livello di granularità assai fine, può spesso non portare sostanziali giovamenti, in caso di degrado delle risorse; solitamente invece, definendo delle classi di qualità di servizio che siano abbastanza differenziate fra loro, in termini di richiesta delle risorse per classe di QoS, si possono ottenere risultati migliori, evitando di incorrere in continui micro-adattamenti che portino solamente ad un ulteriore carico del sistema. Il secondo è la constatazione che un adattamento che agisca nel dominio del continuo non è sempre attuabile. Infatti, se si considerano oggetti multimediali di tipo video, tale soluzione può essere attuabile, ad esempio "croppando" un filmato codificato in MJPEG (Motion Joint Photographic Experts Group, dal nome del gruppo che ha definito lo standard JPEG), oppure scartando dei fotogrammi se si utilizza una codifica MPEG (Moving Picture Experts Group). Se però si considerano degli oggetti multimediali audio, solitamente si opta per un vero e proprio salto di qualità di servizio passando dallo streming di un certo file alla trasmissione di un altro file con lo stesso contenuto, ma a qualità più bassa, giacchè lo scarto di parti del flusso risulterebbe in un salto vero e proprio del sonoro udibile. La scelta di definire delle classi va quindi anche nella direzione di uniformare per quanto possibile anche il trattamento di oggetti multimediali diversi che facciano parte della stessa sessione. Verranno perciò genericamente definite N classi di qualità di servizio.

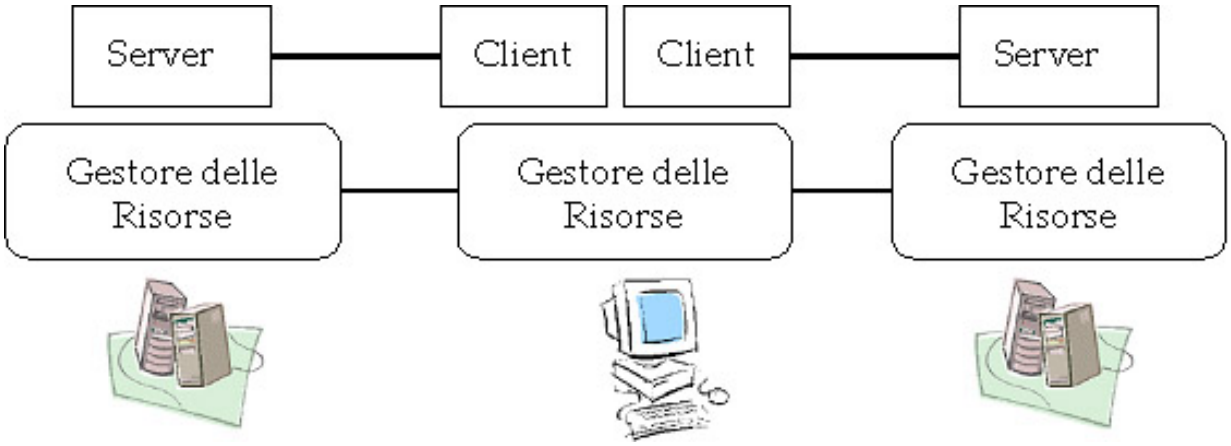


**Figura 5.3:** si ha un unico gestore delle risorse per ogni singola macchina

Sarà realizzato un gestore delle risorse per ogni singola macchina. Più clienti che siano lanciati sulla stessa macchina faranno perciò riferimento allo stesso gestore delle risorse. Sul PC al centro della 5.3 sono stati lanciati due client

per la fruizione di materiale multimediale, entrambi fanno riferimento allo stesso gestore delle risorse. Al di sopra del gestore viene poi realizzata la gestione della qualità di servizio. Abbiamo parlato nel capitolo 2 di meccanismi per il supporto alla dichiarazione e traslazione delle specifiche di qualità di servizio. Il problema della traslazione delle specifiche si presenta anche in questo caso. La soluzione proposta è quella di incapsulare nel metadato *presentazione multimediale* la classe di QoS corrispondente e le richieste di CPU e banda trasmissiva, in modo che non sia necessaria una vera e propria fase di traslazione delle specifiche dal livello applicativo a quello delle risorse, ma siano direttamente disponibili le richieste di risorse. I ***requisiti*** per quello che riguarda la gestione risorse sono i seguenti:

- bisogna che tutte le entità facenti parte della stessa macchina facciano riferimento allo stesso gestore delle risorse;
- bisogna che il gestore risorse mantenga lo stato delle risorse della propria macchina;
- bisogna definire le entità che si occuperanno della gestione delle risorse,e i protocolli di interazione;
- bisogna definire come avverrà il monitoraggio delle risorse, la frequenza di tale processo, lo scope del monitoraggio e cosa fare in caso di degrado delle risorse stesse.

### 5.1.7 Vocabolario di sistema

Prima di passare all'analisi del sistema si propone un vocabolario del sistema che definisca alcune entità e tipi di dati, in modo da rendere più chiara la successiva trattazione.

**Oggetto Multimediale (OM):**

descrittore di una istanza di dato multimediale. Ogni oggetto multimediale si riferisce ad un unico tipo di media (audio o video) e contiene informazioni di formato, compressione e durata. Nel nostro sistema a ciascun oggetto multimediale corrisponde un file che viene univocamente determinato da un locatore (non vogliamo definire come sarà fatto per ora).

**Contenuto Multimediale (CM):**

il Contenuto Multimediale, è un'astrazione introdotta per indicare un set di presentazioni multimediali che portino tutte la medesima informazione. Cioè un sottoinsieme di presentazioni multimediali che abbiano tutte la

medesima struttura, intesa come struttura dell'aggregato di oggetti multimediali, ma differiscano per caratteristiche di formato e qualità.

**Titolo:**

il titolo rappresenta un identificatore unico all'interno del sistema per un contenuto multimediale. Ad uno stesso titolo saranno legate una o più Presentazioni Multimediali.

**Presentazione multimediale (PM):**

metadato che viene utilizzato per aggregare diversi oggetti multimediali i cui singoli contenuti facciano parte dello stesso contenuto multimediale, specificando anche quale sia la qualità della presentazione (intesa come qualità dell'aggregato).

**Utilizzatore:**

un qualsiasi utente, preventivamente registrato, che interagisca col sistema.

L'utente può interagire col sistema nei seguenti modi:

1. richiedendo l'inizio della trasmissione di un titolo;
2. comandando il delivery del contenuto multimediale;
3. richiedendo lo spostamento della sessione ad un altro terminale (su rete fissa);
4. muovendosi, assistito dal middleware che gestisce l'hand-off della sessione (su rete mobile).

Alcune volte potrà capitare di utilizzare anche i termini *utente* o *cliente* al posto di Utilizzatore.

**Profilo:**

il profilo contiene informazioni circa l'utilizzatore e i terminali dai quali l'utente accede al sistema (la loro posizione espressa in termini dell'astrazione di località usata nel sistema e il tipo di piattaforma ospitato.

## *5.2 Analisi*

In questa sezione verrà effettuata l'analisi delle varie parti costituenti il sistema, e delle loro reciproche interazioni, e si produrrà un'architettura logica che guiderà la successiva fase di progetto del sistema. Per quello che riguarda il linguaggio di modellazione utilizzeremo **UML** e gli **stereotipi** proposti da Jacobson. Lo Unified Modeling Language è un linguaggio di tipo visuale utilizzato per visualizzare costruire e documentare sistemi orientati agli oggetti. Il meccanismo di stereotipazione viene utilizzato in UML per indicare che un particolare elemento del modello segue un pattern o un comportamento ben preciso dello stereotipo a cui si riferisce. La stereotipazione è approdata in UML grazie ai

primi lavori di Jacobson sulla modellazione orientata agli oggetti che si avvale di una notazione simbolica a icone per rappresentare gli stereotipi di entità di Boundary, Control e Entità, che si riferiscono ad un sistema organizzato secondo una architettura di tipo BCE.

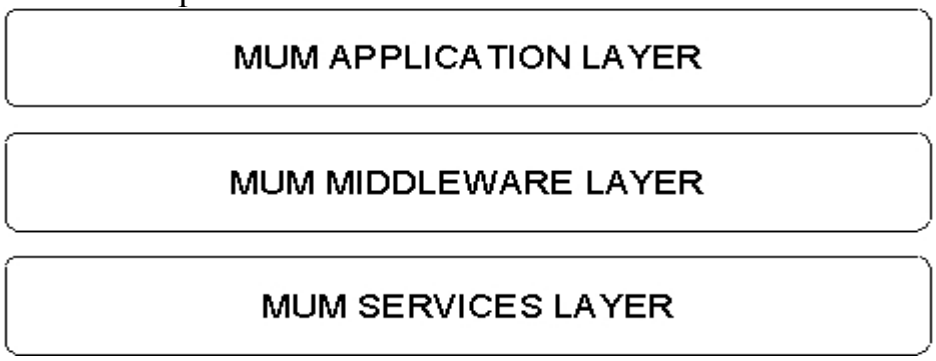


**Figura 5.4**: suddivisione in layer

Nei modelli visuali si è cercato di fare uso di nomi che siano il più possibile auto esplicativi. Il sistema in progetto verrà per brevità spesso referenziato con l'acronimo MUM (Multimedia agent based Ubiquitous Multimedia middleware).

Utilizzeremo una metodologia di tipo top-down che suddividerà il sistema in layer e sottosistemi. Particolare attenzione è stata posta nel decidere quali dettagli siano rilevanti o meno al fine della produzione dell'architettura logica del sistema, tenendo sempre presente il requisito di *robustezza* dell'architettura logica. Col termine robustezza si intende la proprietà dell'architettura logica di rimanere il più possibile invariata al variare del contesto tecnologico nel quale essa verrà poi tradotta in progetto e implementata. A questo livello non ci preoccupiamo eccessivamente di delineare come i sottosistemi verranno partizionati e organizzati in layer, poiché ci occuperemo di ciò successivamente.

In figura 5.4 è rappresentata la suddivisione in layer dell'architettura del sistema. Il sistema viene organizzato in tre livelli. Il **MUM Services Layer** implementa i servizi base, delle sorte di primitive che possono essere accedute dagli oggetti facenti parte del **MUM Middleware Layer**. Questo secondo livello, utilizzando i servizi offerti realizza funzionalità più complesse che mette a disposizione dello sviluppatore dell'applicativo. Il terzo ed ultimo livello (**MUM Application Layer**) racchiude invece tutte le entità che realizzano l'applicazione vera e propria, e l'interazione con l'utilizzatore del sistema. Il MUM Application Layer, con la presentazione della applicazione realizzata per testare il sistema, verrà considerato in un capitolo successivo.

### 5.2.1 MUM Services Layer

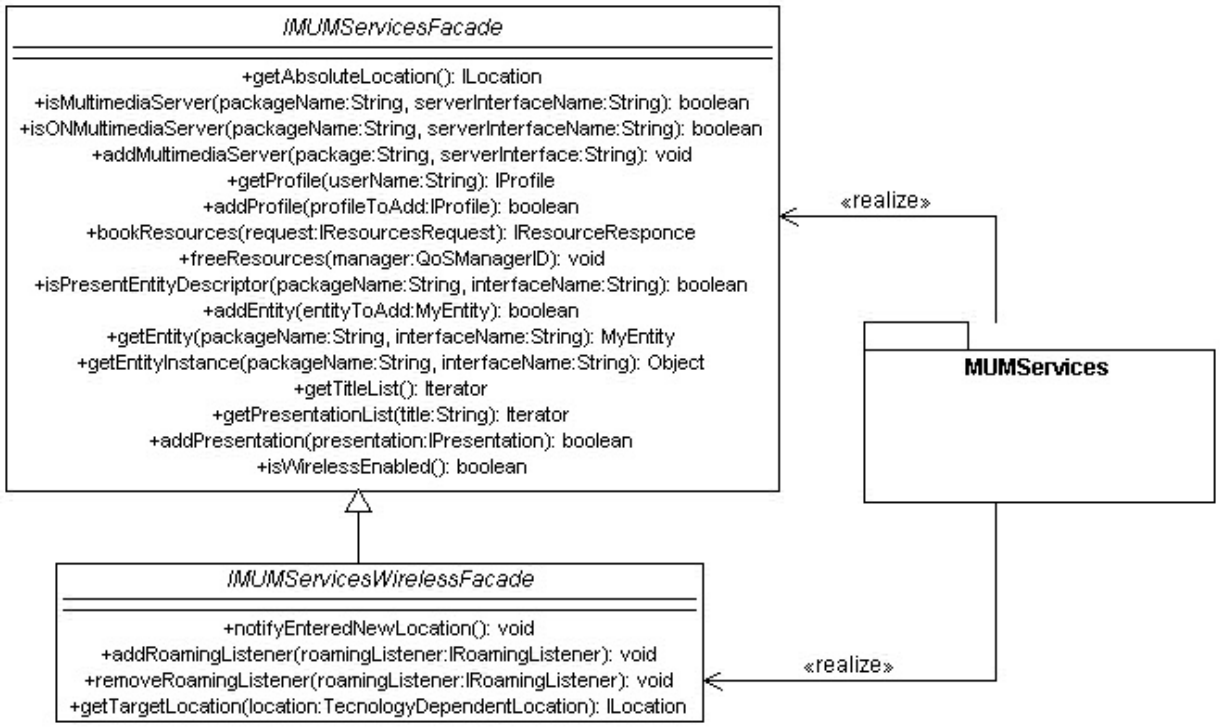


**Figura 5.5**: interfaccia del MUM services Layer

Prima di suddividere questo layer nelle varie parti che lo compongono, si vuole fissare l'interfaccia che questo layer offre al livello superiore, visualizzando, in 5.5, le signature dei principali servizi messi a disposizione. Si è deciso di realizzare un insieme di servizi comuni a tutte le piattaforme che faranno parte dell'architettura, sia quelle fisse che quelle mobili. Come si vede dal diagramma per le piattaforme mobili, i servizi offerti sono stati estesi, in modo da supportare la mobilità del terminale. Come suggerito dai nomi delle interfacce viene applicato il pattern Facade (si veda [FacPat]), in modo da offrire al livello superiore un'interfaccia unica, e non dover invece gestire i molti oggetti che costituiranno questo layer, semplificando l'interazione fra i livelli.

Vediamo ora i vari oggetti e sottosistemi che costituiscono il layer, e che realizzano i servizi messi in evidenza dall'analisi dei requisiti. Il diagramma di collaborazione in figura 5.6 evidenzia i vari oggetti che concorrono alla realizzazione dei servizi esposti dal layer. Nel seguito si esamina nel dettaglio l'architettura logica dei singoli servizi.

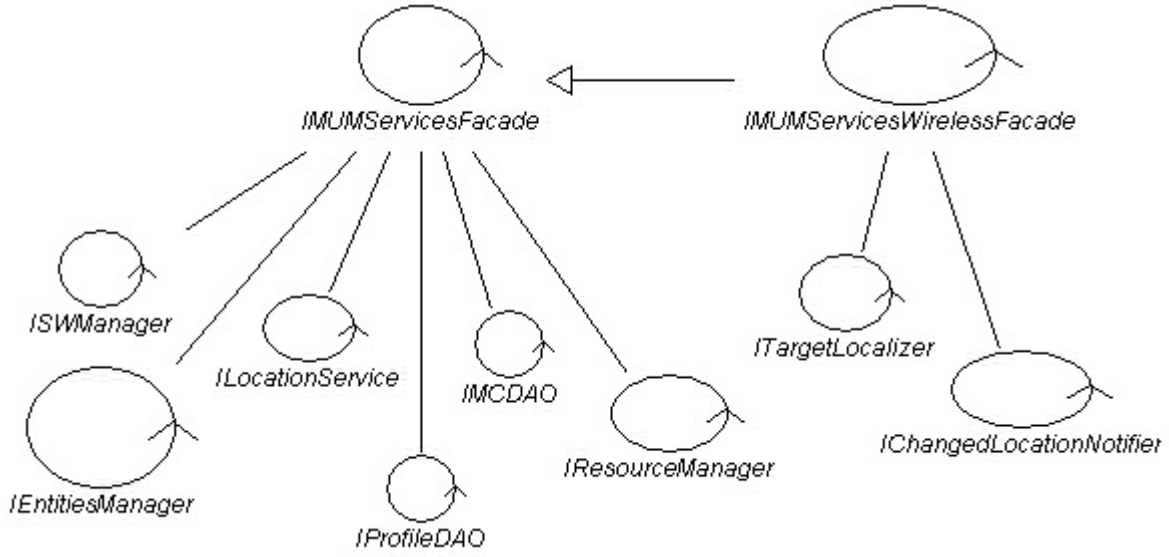


**Figura 5.6**: MUM Services Layer, oggetti costituenti il layer

#### 5.2.1.1 Location Service

Questo servizio offre un'astrazione di località. In particolare si vuole fare in modo che ogni nodo che prende parte al sistema sia univocamente localizzabile all'interno del sistema stesso e tale informazione di località rispecchi la località fisica dei vari nodi. L'intento è quello di offrire al middleware layer e all'application layer consapevolezza dell'ambiente nel quale avverrà lo streaming, in modo che possano essere fatte scelte "intelligenti", che per esempio, a fronte della richiesta di un certo titolo favoriscano la scelta di una presentazione multimediale fisicamente più vicina nei confronti di una più lontana, in modo da non occupare inutilmente banda trasmissiva. Il servizio offerto deve essere in grado, all'aggiungersi di un nodo al sistema, di registrare tale nodo presso un nodo già esistente, e di costruire un identificatore unico all'interno del sistema.

Non entreremo nei dettagli dell'architettura logica di questo servizio perché per questo servizio viene utilizzata l'astrazione di località introdotta da SOMA, di cui abbiamo già parlato nel capitolo 3. I nodi verranno cioè organizzati logicamente in una struttura ad albero e l'identificatore unico è rappresentato dal percorso che parte dalla radice dell'albero e raggiunge il nodo che si vuole identificare (Path). Sulla rete fissa ogni **place** corrisponde ad una macchina e ogni rete locale ad un dominio, naturalmente per ogni dominio verrà definito un **default place**. Per quello che riguarda le reti wireless, ogni ESS sarà logicamente accoppiato ad un **default place wireless enabled**, che sarà una macchina

collegata alla rete fissa alla quale sono collegate anche le base station. Tale default place differisce da un classico default place per il fatto che ospiterà il Target Localizer per la rete wireless (si veda il punto 5.2.1.7). Ad ogni AP viene poi fatta logicamente corrispondere una macchina, che rimarrà fissa nel BSS dell'AP al quale viene fatta corrispondere, e ospiterà un **place wireless enabled**. Tali place sono identici a quelli sopra definiti per le reti fisse, però quando si registrano presso il default place (che sarà un un default place wireless enabled) devono comunicare il Path che le contraddistingue e l'indirizzo MAC dell'AP al quale corrispondono. Sul terminale mobile sarà poi presente un ultimo place detto **place wireless** che ospiterà il MovementSensor (si veda il punto 5.2.1.7).

### 5.2.1.2 Sottosistema di gestione del software

Il sottosistema offre i servizi per il download e l'upload di **Entità** e per l'interrogazione del database contenente le entità e i loro descrittori. Ogni Entità è definita da una precisa interfaccia, facente a sua volta parte di un certo package ed è composta, oltre che da tale interfaccia, anche da una o più classi che servono per implementarla. L'Entità risulta univocamente determinata dal nome del package e dell'interfaccia. Con riferimento alla tecnologia Java, che verrà utilizzata per l'implementazione del sistema, la richiesta di una certa entità produce quindi il download dei file ".class", dell'interfaccia e delle altre classi utilizzate per implementare tale interfaccia, secondo il paradigma COD.

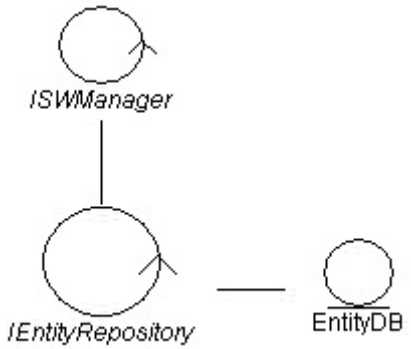


**Figura 5.7**: sottosistema di gestione del software, architettura logica

Ogni nodo (Place) metterà a disposizione un oggetto, chiamato **Software Manager** per la gestione del software presente sul nodo. Per ogni nodo vengono memorizzati i descrittori delle entità presenti sul nodo e le classi costituenti le entità. Ovviamente tali dati sono salvati in memoria persistente. Se in futuro venisse cambiata l'implementazione di una certa entità, l'interfaccia, cioè il contratto di funzionamento di tale entità dovrà essere comunque rispettato.

Quindi può essere utilizzata senza alcun problema l'entità già presente in memoria. In un qualsiasi momento il gestore del nodo può poi decidere di cancellare il database; tale operazione corrisponde ad una sorta di cancellazione di una cache. Per quello che riguarda il servizio per lo scaricamento del software esso viene implementato nel MUM middleware layer, e viene considerato più avanti. L'architettura logica del sottosistema è mostrata in 5.7. L'IEntityRepository potrebbe apparire come un oggetto inutile, ma è stato introdotto per rendere l'ISWManager indipendente dal DB utilizzato, incapsula cioè la logica di accesso al DB.

#### 5.2.1.3 Servizio di attivazione delle entità multimediali

Questo servizio viene utilizzato per attivare i server multimediali in uno dei nodi facenti parte del sistema, e, una volta attivato il server, per mantenere un riferimento al server, in modo da non dovere attivare un nuovo server per ogni nuova richiesta, ma attivare invece una sola istanza di server per ogni tipologia di server multimediale. Ogni nodo mette a disposizione il servizio. Il servizio viene anche utilizzato per salvare i riferimenti ai Client istanziati, mantenendo una tabella che fa corrispondere ad ogni ClientAgent il proprio Client. In caso di movimento dell'utente infatti, si vuole fare in modo che il Client venga inizializzato sulla macchina target, prima che il ClientAgent "salti" su tale macchina, in modo all'arrivo dell'agente tutto sia già pronto per far ripartire il rendering, senza che si debbano aspettare i lunghi tempi dovuti all'inizializzazione.

Questo servizio necessita di una struttura dati per la memorizzazione dei riferimenti ai server e ai client attivi. Tale struttura dati verrà conservata in memoria, e sarà distrutta ad ogni disattivazione del nodo. L'architettura logica di questo servizio è assai semplice, è infatti costituita dall'unico oggetto IEntityManager che realizza le funzionalità descritte e mantiene aggiornata la struttura dati.

#### 5.2.1.4 Sottosistema di gestione delle risorse

In questo layer realizziamo il servizio per la prenotazione e il rilascio delle risorse. La richiesta di risorse, la IResourceRequest, contiene le richieste di CPU e banda. In risposta ad una richiesta di risorse il gestore delle risorse risponde ritornando al richiedente una IResourceResponce. Chi gestirà le risorse per conto delle diverse entità che partecipano all'architettura distribuita sono i QoSManager. La IResourceResponce contiene informazioni circa la riuscita della

prenotazione. Potrebbe essere una risposta affermativa, in caso siano disponili le risorse, negativa, se non ci sono risorse disponibili, oppure positiva con riserva, se il gestore è stato informato di un prossimo rilascio di risorse. In questo caso cioè la risorsa viene per così dire "anticipatamente prenotata", e solo quando è realmente disponibile ciò viene comunicato al QoSManager che aveva effettuato la prenotazione. Se il QoSManager decide di non attendere la liberazione delle risorse dovrà disdire la propria prenotazione anticipata. Internamente il servizio manterrà perciò una tabella che per ogni QoSManager memorizzerà l'ammontare di risorse richiesto. Al termine dello streaming è dovere del QoSManager procedere alla liberazione delle risorse, che avverrà in due fasi: dapprima viene comunicata una prossima liberazione delle risorse. Poi, al termine vero e proprio dello streaming, le risorse vengono effettivamente liberate.

Il sottosistema di gestione delle risorse permette inoltre ai QoSManager di registrarsi per ottenere informazioni riguardo l'utilizzo della CPU sul nodo corrente, mediante la notifica di eventi di monitoring. Le decisioni riguardanti la frequenza di campionamento del monitoring sono rimandate alla fase di progetto.

#### 5.2.1.5 Gestione dei contenuti multimediali

Il servizio di gestione dei contenuti multimediali si basa su una classica architettura cliente-servitore. Quando un nuovo nodo di aggiunge alla gerarchia dei nodi, richiede al padre, presso cui si registra, un riferimento alla base di dati (servitore) che memorizza le informazioni relative alle presentazioni multimediali presenti all'interno del sistema. Localmente ogni nodo mette poi a disposizione uno stub (cliente) per l'interrogazione del DB remoto. Tale stub realizza un Data Access Object, implementa cioè il pattern DAO, si veda [DaoPat], in questo modo si disaccoppia cioè l'accesso al DB dalla logica applicativa. Vi sarà un unico servitore centralizzato a cui accederanno tutti i nodi, e che viene collocato sul nodo root. Questa soluzione non è ovviamente scalabile, nondimeno rileviamo che in futuro si può pensare di implementare un servizio di caching in modo che una volta scaricata una lista di presentazioni rispondenti ad un certo titolo, sia possibile per nuovi accessi non dover interrogare direttamente la root, ma utilizzare il dato presente in cache e inoltrare la richiesta al server centralizzato solo in caso non sia possibile trovarlo in cache.

Definiamo di seguito il tipo di dato *contenuto multimediale* e *presentazione multimediale*. Useremo a tale scopo una pseudo grammatica. Il formalismo utilizzato vuole essere unicamente uno strumento per analizzare meglio cosa si intenda per Presentazione Multimediale, anche se ovviamente non si può parlare

di vera e propria grammatica formale, anche perché in alcuni casi utilizziamo definizioni date per mezzo di frasi del linguaggio naturale.

Vogliamo solo notare che esiste un vincolo fra il Titolo e la struttura del MultimediaObjectsAggregate. Cioè tutte le presentazioni multimediali aventi lo stesso titolo devono anche avere la stessa struttura per il MultimediaObjectsAggregate.

<MultimediaContent> ::=
<Title> <Description> {<MultimediaPresentation>}
<Title> ::= stringa alfanumerica
<Description> ::= un testo che descrive brevemente il contenuto multimediale di questa presentazione
<MultimediaPresentation>::=
<Quality> <CPURequest> <BandwidthRequest>
<MultimediaObjectsAggregate> <TotalFrameDimension>
<AbsoluteLocation>
<AbsoluteLocation> ::= il Path che identifica univocamente il nodo dove è mantenuta la presentazione multimediale
<Quality> ::= low | medium | high
<CPURequest> ::= un intero fra 1 e 100 rappresentante la richiesta di CPU in forma percentuale
<BandwidthRequest> ::= un intero che definisce la richiesta di banda in bps
<TotalFrameDimension> ::= <FrameDimension>
<MultimediaObjectsAggregate> ::=
< MultimediaObject > |
< MultimediaObject > < MultimediaObjectsAggregate >
< MultimediaObject > ::=
audio <AbsoluteFileName> |
video <FrameDimension> <Position> <AbsoluteFileName>
<FrameDimension> ::= width <width> height <height>
<width> ::= un intero che indica in pixel la larghezza
<height> ::= un intero che indica in pixel l'altezza
<Position> ::= outRight <right> outTop <top>
<right> ::= un intero che indica la distanza in pixel dal lato sinistro del frame dove sono contenuti tutti gli OM video
<top> ::= un intero che indica la distanza in pixel dal lato superiore del frame dove sono contenuti tutti gli oggetti multimediali

<Duration> ::= un intero che dichiara la durata in secondi dell'oggetto multimediale
<AbsoluteFileName> ::= il nome assoluto del file nella località referenziata da AbsoluteLocation

#### 5.2.1.6 Gestione dei profili

Dal punto di vista architetturale il servizio è organizzato come il servizio di gestione dei contenuti multimediali. Esaminiamo però più nel dettaglio la struttura del tipo di dato profilo utente.

Il profilo utente, come già detto, dovrà contenere le informazioni riguardanti le piattaforme utilizzate dall'utente per accedere al sistema e le loro locazioni. I descrittori delle piattaforme sono un dato che rimane piuttosto stabile nel tempo e viene trattato separatamente rispetto ai profili utente, che invece possono cambiare più di frequente, perciò si è deciso di dividere logicamente i due database che conterranno queste informazioni. Il profilo utente conterrà una tabella che farà corrispondere ad ognuno dei terminali da cui l'utente accede al sistema (identificato univocamente dalla sua locazione espressa in termini dell'astrazione di località introdotta) l'identificatore unico del descrittore della piattaforma. I descrittori delle piattaforme stabiliscono le caratteristiche delle delle piattaforme. Il descrittore di piattaforma contiene come la grandezza dello schermo e la classe della piattaforma. All'interno del sistema in progetto vengono genericamente definite **M classi di piattaforme**. Queste definizioni vengono introdotte con l'intento di gestire al meglio la prenotazione delle risorse e la complessità dovuta all'eterogeneità di piattaforme presenti all'interno del sistema. Come visto per ogni metadato che descrive le presentazioni contiene le richieste di CPU e banda trasmissiva. Si nota però che mentre la richiesta di banda trasmissiva non è dipendente dalla specifica piattaforma con la quale si ha a che fare, ma è fissata dalla mole di dati da trasmettere nell'unità di tempo, la richiesta di CPU è invece fortemente dipendente dalla specifica piattaforma partecipante al multimedia streaming e vengono per questo motivo definite le varie classi. La percentuale di utilizzo di CPU presente nel metadato delle presentazioni si riferisce alla piattaforma con capacità computazionale massima partecipante al sistema; a tale piattaforma sarà attribuita la classe M. Quando una richiesta di risorse viene presentata al gestore delle risorse sarà poi presente una funzione per la traslazione della richiesta di CPU presente nel metadato in una richiesta di CPU per la specifica piattaforma (in realtà questa funzione altro non fa che moltiplicare per un valore maggiore di uno la percentuale richiesta, in

modo da ricavare la richiesta di CPU per la specifica piattaforma di classe M). In figura 5.8 viene riportata l'architettura di questo sottosistema.

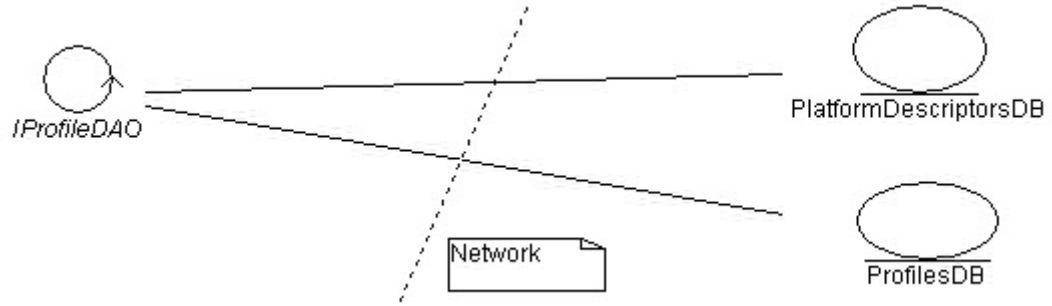


**Figura 5.8**: sottosistema di gestione dei profili utenti

### 5.2.1.7 Servizi di supporto alla mobilità del terminale

I servizi offerti sono due, uno per la notifica dell'ingresso in una nuova località, e l'altro per l'interrogazione del TargetLocalizer. L'architettura logica del sottosistema è mostrata in 5.9.

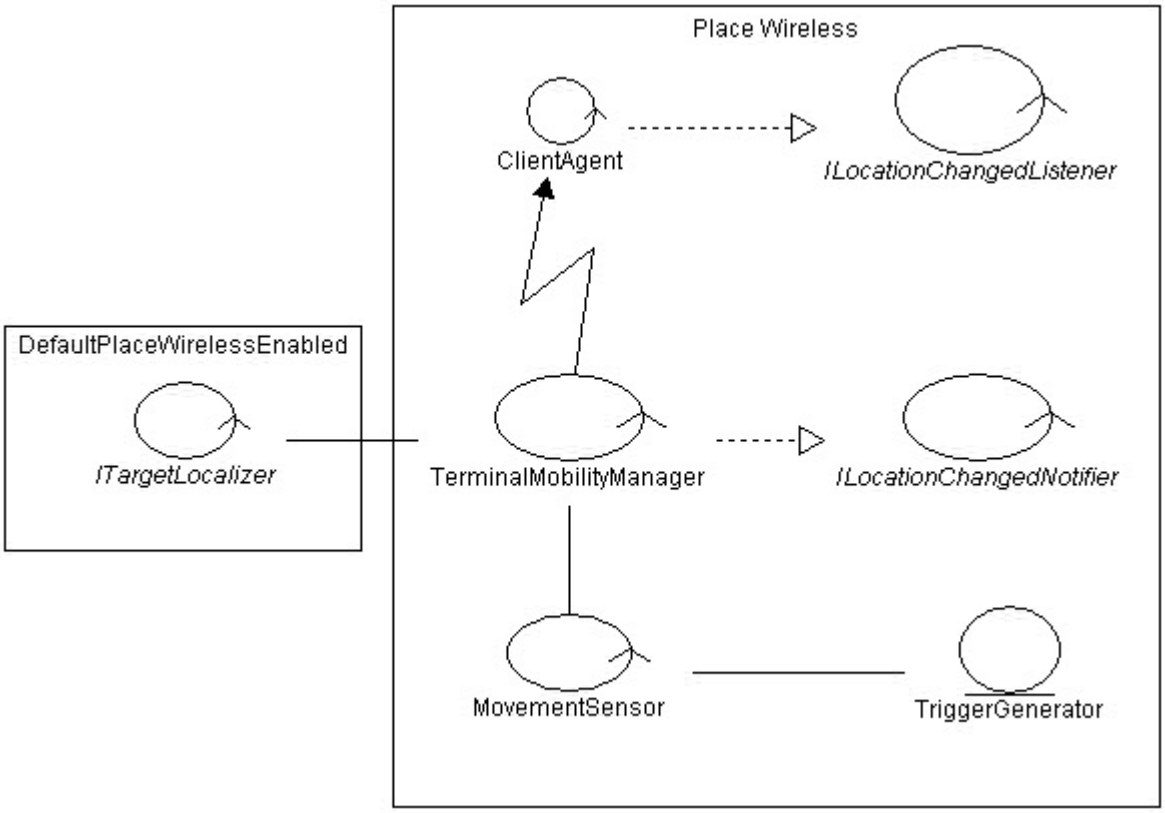


**Figura 5.9**:supporto alla mobilità del terminale

All'ingresso in una nuova località il Movement Sensor (che sarà costruito al di sopra delle Wireless API) richiama il metodo per la notifica dell'ingresso offerto dal TerminalMobilityManager. Vengono perciò notificati tutti i listener in

attesa di questo evento. L'evento contiene informazioni riguardanti i vari Place BSS sui quali è stato previsto che si potrebbe saltare, riportando per ciascuno di essi il Path che li identifica univocamente. Esisterà un unico Target Localizer per ogni rete wireless, e sarà un servizio offerto dal Default Place Wireless Enabled. Vi è poi un unico TerminalMobilityManager per ogni Place Wireless. Il TriggerGenerator incapsula le politiche per la decisione di quando notificare il movimento al livello superiore. Ad esempio l'utilizzo delle WirelessAPI, in particolare nell'implementazione per piattaforme Windows permette di estrarre informazioni circa la potenza con la quale il segnale proveniente dai vari AP viene ricevuto. Se i movimenti sono molto piccoli, cioè se la potenza dei segnali non cambia considerevolmente, possiamo quindi evitare di notificare al livello superiore ciò. Verranno invece fissate alcune soglie, superate le quali scatterà la notifica. Tali soglie sono incapsulate nel TriggerGenerator.

### 5.2.2 MUM Middleware Layer

Nel middleware layer vengono realizzati servizi più complessi che si basano sulle funzionalità offerte del MUM Services Layer. Continuiamo perciò l'analisi del sistema delineando l'architettura logica di tali servizi.

#### 5.2.2.1 Servizio per l'istanziazione di entità

Il servizio per l'istanziazione delle entità è una factory per la creazione di oggetti che debbano essere passati all'application layer, o devono essere creati a livello del MUM Middleware Layer.

#### 5.2.2.2 Servizio di downloading del software

Per la realizzazione di questo servizio si è deciso di utilizzare un agente mobile. Questa scelta è stata fatta perché realizzando come agente mobile questo componente possiamo facilmente cambiare le politiche di downloading, semplicemente agendo sul codice dell'agente. Inoltre si è sfruttato il supporto alla mobilità offerto da SOMA, unitamente al location service introdotto, in modo da rendere il servizio di downloading dinamico e scalabile. Non vi sarà alcuna necessità di configurazioni aggiuntive rispetto a quelle richieste dall'ambiente SOMA, per poter utilizzare questo servizio, inoltre dipendentemente dalle politiche adottate per la realizzazione dell'agente si può rendere il servizio più o meno scalabile. L'interazione fra il cliente di questo servizio e il servitore sarà basata sul modello a eventi. Il cliente richiede cioè il downloading di un certo SW al servitore e il servitore, quando il SW sia stato scaricato, notifica l'avvenuta esecuzione del servizio scatenando un evento al cliente che si era

preventivamente registrato presso di lui. Per ogni richiesta di SW viene passato al servitore l'elenco delle entità da scaricare (un elenco di tuple [nomePackage, nomeInterfaccia]), e numero che funga da identificatore per la richiesta, e che servirà, in caso vengano fatte più richieste da parte del medesimo cliente, per capire a quale precisa richiesta si riferisca un certo call-back.

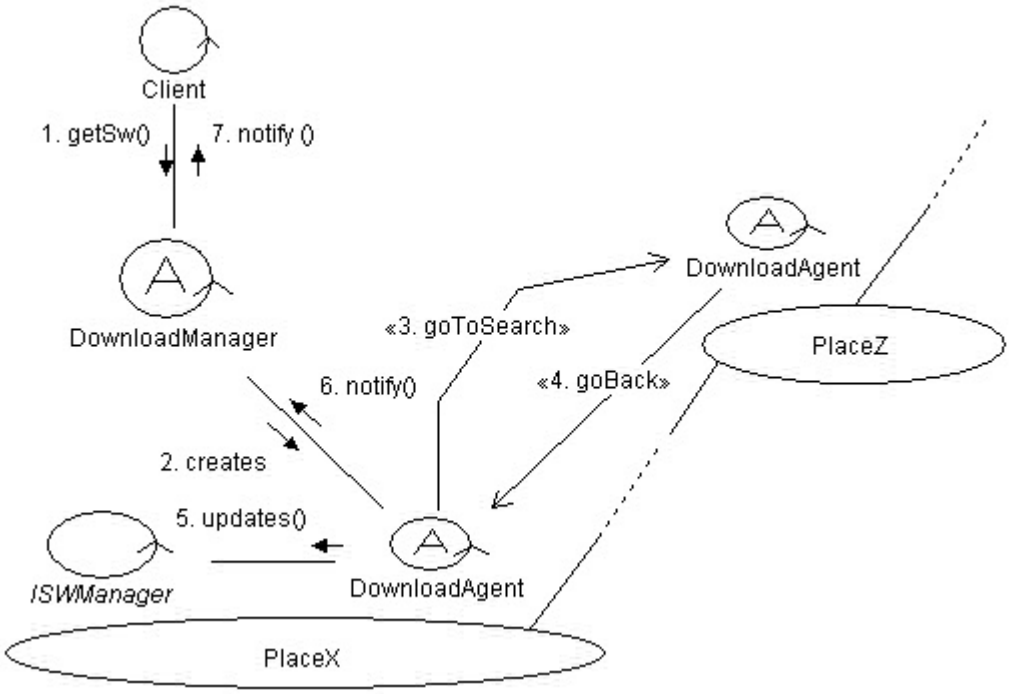


**Figura 5.10**: sottosistema di downloading del software

Ogni oggetto interessato al downloading di codice istanzierà, utilizzando il servizio la factory introdotta al punto precedente, un Download Manager. Il Download Manager è un agente degenere che rimane fisso sul place nel quale viene lanciato, e implementa i metodi per registrare il listener e per la notifica dell'avvenuto downloading. Per ogni nuova richiesta il DownloadManager dapprima controlla se il SW richiesto è già presente sul place, e in tal caso emette subito un evento di avvenuto download, altrimenti crea un agente per il downloading del codice e lo fa partire. Ipotizzeremo che tutto il codice scaricabile (COD) all'interno del sistema sia presente almeno nel nodo radice. Compito dell'agente è quello di andare alla ricerca del codice richiesto, ritornare al place di partenza, aggiornare l'EntityDB (si veda il punto 5.2.1.2), quindi spedire un messaggio al DownloadManager per notificare l'avvenuta terminazione dell'operazione richiesta. Il manager a sua volta invierà un evento al proprio cliente. In figura 5.10 rappresentiamo l'architettura proposta. La notazione di Jacobson è stata leggermente modificata introducendo un nuovo

stereotipo per modellare gli agenti mobili. L'entità Client in figura rappresenta un generico cliente del servizio.

Non abbiamo definito le politiche di download operate dall'agente, che fisseremo in fase di progetto di questo sottosistema.

#### 5.2.2.3 Architettura delle entità che effettuano lo streaming

Prima di considerare il sottosistema di inizializzazione vogliamo dettagliare meglio l'architettura di una generica entità che si occupa di gestire lo streaming per una determinata sessione. Tali entità, come già detto, sono client, proxy e server, ognuno dei quali può gestire uno o più flussi, agendo a livello di sessione. Ognuna di esse è un aggregato dei seguenti oggetti:

- Gestore della sessione;
- Uno o più agenti di gestione dei singoli flussi;
- Un protocollo per gestire il controllo di sessione;
- Un manager della qualità di servizio;

In figura 5.11 riportiamo l'architettura di un generico client.

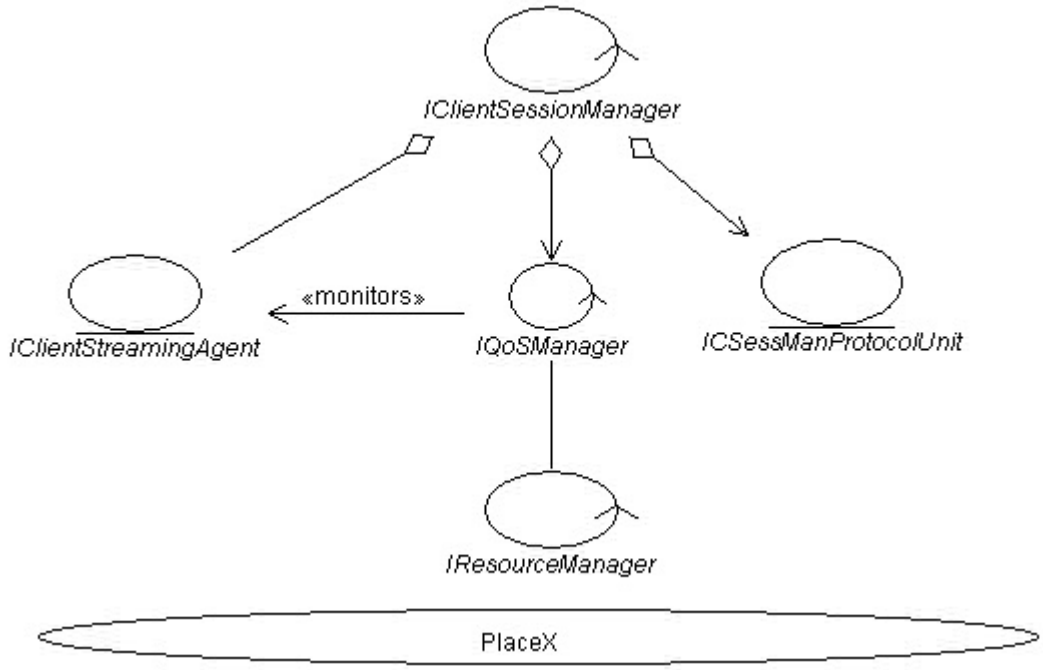


**Figura 5.11**: architettura client

Le architetture del proxy e del server sono assai simili a quella introdotta per il client. L'unica differenza sta nel fatto che, mentre gli *Streaming Agent* del client dovranno gestire solo flussi in ingresso, quelli del server dovranno gestire unicamente flussi in uscita, e quelli del proxy dovranno riportare in uscita tutti i

flussi che arrivano in ingresso. Il *QoS manager* interagisce col *Resource Manager* presente sui vari place, per la prenotazione delle risorse, inoltre monitora la situazione dello streaming osservando gli streaming agent. Il resource manager può poi notificare al QoS manager eventuali problemi dovuti all'eccessiva occupazione di CPU, che potrebbero richiedere un adattamento della sessione. Da ultimo le *Protocol Unit* incapsulano i protocolli di comunicazione fra le varie entità distribuite. In particolare notiamo che le protocol unit vengono utilizzate sia per gestire il controllo della sessione, sia per comunicazioni legate necessarie per la gestione della QoS.

#### 5.2.2.4 Servizio di inizializzazione e riconfigurazione dinamica del sistema

Il servizio di inizializzazione del sistema ha rappresentato uno dei punti nevralgici della progettazione del sistema. Partendo dai requisiti si è cercato di realizzare un supporto che fosse il più possibile flessibile e modulare. Progettando questo servizio si è in pratica progettato anche un vero e proprio framework per l'introduzione di entità di tipo client, proxy e server che vengano progettate in futuro. Il servizio di inizializzazione impone un modello per il collegamento fra le varie entità, e impone che in tali entità siano implementati alcuni metodi, lasciando al progettista dell'applicativo il compito di concentrarsi sulla realizzazione delle entità stesse, sollevandolo dal compito di progettare un sottosistema per la distribuzione e l'inizializzazione di tali componenti.

Le varie entità come già detto saranno distribuite su un Path che parte dal cliente e termina con un server. Tutte le entità introdotte agiranno a livello di sessione, gestendo tutti i flussi appartenenti a una certa sessione. L'inizializzazione sarà portata a termine da agenti mobili detti Plan Visitor Agent. Alla ricezione di una richiesta infatti il ClientAgent (vedi punto 5.1.2) richiede all'IInitManager l'inizializzazione del sistema indicando il DecisionMaker (il *decisore* introdotto nei requisiti al punto 5.1.5), da utilizzare e passando la richiesta dell'utente (il titolo richiesto). Fatto ciò attende, come accade per il servizio di downloading, la notifica di avvenuta inizializzazione. Il servizio di inizializzazione spedisce lungo il percorso i PlanVisitorAgent, passando ad essi un Plan, che è un piano di inizializzazione che contiene le informazioni riguardanti i componenti da scaricare e inizializzare lungo il percorso. Per l'interpretazione dei Plan, anche per permettere future estensioni, si è fatto uso del pattern Visitor [VisPat]. I Visitor Agent si coordinano tra loro, quando cioè il server è stato inizializzato le informazioni riguardanti l'end-point del server vengono passate indietro agli altri visitor agent sul percorso che

possono inizializzare le altre entità fino al client, come visualizzato dal collaboration diagram in figura 5.12.

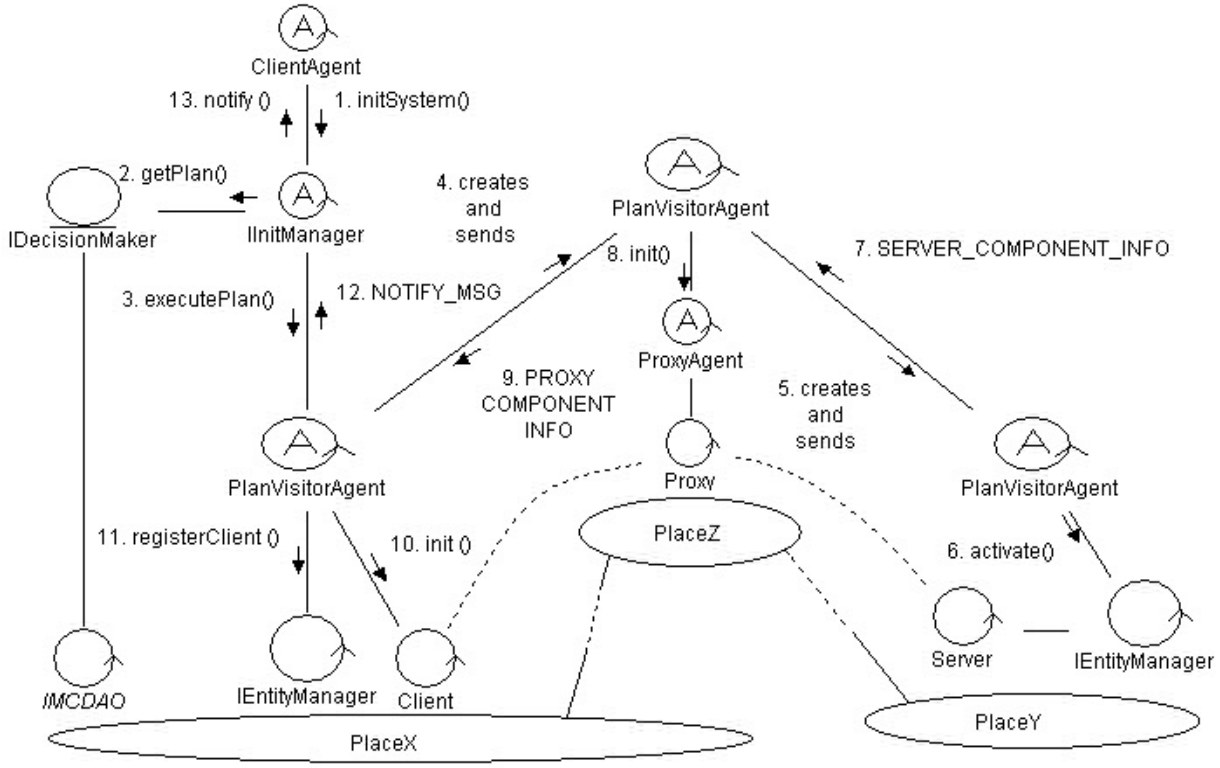


**Figura 5.12**: inizializzazione del sistema.

Si nota che il DecisionMaker incapsula, oltre alle politiche di scelta del software da istanziare, le politiche di scelta delle presentazioni multimediali. Per cambiare le politiche di inizializzazione e riconfigurazione del sistema basterà dunque modificare il DecisionMaker, in modo che produca un diverso piano di inizializzazione, senza che ciò richieda alcuna modifica nel resto del sistema. Lo sviluppatore dell'applicativo può dunque implementare nuove politiche unicamente riscrivendo il DecisionMaker, con l'unico accorgimento di introdurre nei Plan richieste di *Entità* disponibili all'interno del sistema, e richiedendo inizializzazioni valide. Non si può per esempio inizializzare un percorso pretendendo di far dialogare tra loro due client.

Il servizio di inizializzazione provvede anche ad effettuare la fase di negoziazione iniziale della QoS e prenotazione delle risorse. Appena giunto in una nuova località infatti il Plan Visitor per prima cosa vede se ci sono risorse sufficienti per la presentazione richiesta. In caso affermativo continua il processo di inizializzazione inviando un altro Plan Visitor Agent sul prossimo place, e se ci sono sufficienti risorse su tutto il Path, il processo di inizializzazione termina

con successo, come mostrato in figura 5.12. Se invece in un certo place non sono disponibili sufficienti risorse, il Plan Visitor consulta il proprio piano di inizializzazione per vedere se esistono presentazioni alternative a qualità più bassa e tenta di istanziare il persorso per tali presentazioni. In caso non si riesca a inizializzare il sistema per nessuna delle presentazioni incluse nel Plan ciò viene notificato all'indietro, fino al ClientAgent.

#### 5.2.2.5 Sottosistema di gestione della qualità di servizio

La prima questione da sciogliere è quale sarà lo scope della gestione delle risorse nel sistema. Come visto al punto 2.1.2.2 le possibilità attuabili sono due, la prima è quella di gestire la qualità di servizio agendo su base globale, considerando cioè il sistema nel suo complesso, l'altra è quella di agire localmente, cercando di capire come stiano andando le cose in una certa località, e cercando di reagire agendo su base locale. Nel progetto di questo sistema è stata scelta questa seconda via perché si ritiene richieda meno overhead, e sia intrinsecamente più scalabile. Per la gestione della qualità di servizio vengono fissate N classi di qualità di servizio. Quando l'utente richiede un certo titolo, vengono richieste tutte le presentazioni multimediali che corrispondono a tale titolo, si procede quindi ad una analisi delle stesse, in modo da costruire dei Plan per l'inizializzazione e la riorganizzazione del sistema in caso di degrado delle risorse. I criteri adottati sono i seguenti.

- Fra le presentazioni per ogni classe vengono ricercate le presentazioni che si trovano più vicine al client, in termini dell'astrazione di località introdotta. Si noti che è facile, avendo i percorsi assoluti dalla root ad ogni nodo considerato, concatenare il path che identifica il client col path che identifica la piattaforma, in modo da avere il path sul quale saranno istanziate le varie entità. In figura 5.13 viene visualizzato tale processo. Vengono indicati in rosso i path dalla radice della gerarchia al nodo ove si trova il client, in blu i percorsi dalla radice ai nodi ove si trovano le presentazioni e in verde i percorsi costruiti concatenando gli altri due, lungo i quali verranno istanziate le varie entità.
- A questo punto rimane da decidere come disporre le entità lungo il percorso. La scelta fatta è stata quella disporle in modo che in caso di richiesta di riconfigurazione il percorso già stabilito cambi il meno possibile. Per ogni piano sul primo nodo dovrà essere istanziato il client, e sull'ultimo il server. Per i nodi intermedi, verranno ricercati tutti i nodi non terminali, che sono in comune a due o più percorsi alternativi, istanziando

su tali nodi i proxy. Il Visitor Agent cioè, al momento di istanziare i proxy vedrà se sono presenti per quel medesimo nodo che sta considerando piani alternativi. Se ci sono istanzierà un proxy, che potrà essere utilizzato per stabilire i percorsi alternativi fissati da tali piani, nel caso invece non siano presenti piani alternativi non verrà istanziato un ulteriore proxy, per evitare un eccessivo consumo delle risorse del sistema. Sempre considerando la 5.13 perciò, ipotizzando che il piano per la presentazione qualità maggiore sia quello avente come percorso il Path 1, verrà istanziato il client sul nodo D5, un proxy sul nodo D2 e il server sul nodo D8, in modo che, se per esempio non vi fossero sufficienti risorse sul nodo D8, il proxy sul nodo D2 possa comunque essere utilizzato per stabilire il percorso alternativo, con server in D6.

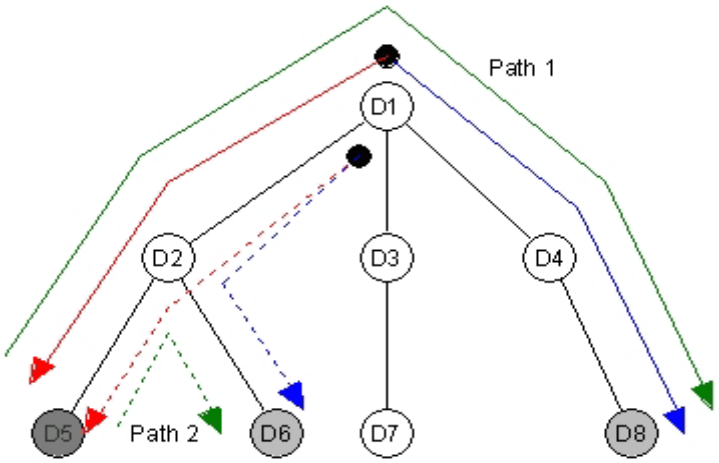


**Figura 5.13**: costruzione dei path

- La scelta adottata per la disposizione dei proxy influenza la fase di adattamento del sistema, infatti nel caso in cui un QoSManager lungo il percorso rilevi una oscillazione delle risorse tale da richiedere un adattamento, viene spedito all'indietro lungo il percorso un messaggio per richiedere il salto ad una qualità di servizio più bassa e l'ultima delle entità in comune al percorso attuale e al prossimo che verrà inizializzato durante la fase di riconfigurazione, ricevendo tale messaggio capisce che è stato richiesto un adattamento. Procede perciò alla riconfigurazione, senza inoltrare alle precedenti entità il messaggio stesso. Nel nostro esempio perciò dal proxy D2 partirà una fase di inizializzazione della parte del Path 2 non ancora inizializzata, e non appena finita tale fase di inizializzazione, viene spedita la presentazione di qualità media al posto dell'altra, e vengono rilasciate le risorse sul Path 1.

#### 5.2.2.6 Supporto per la mobilità utente

Il supporto per la mobilità utente è basato sul sottosistema di inizializzazione e riconfigurazione. Quando l'utente decide di muoversi il Plan Visitor Agent viene spedito sulla nuova località dove istanzia un nuovo Client, inizializzandolo in modo venga collegato alla seconda entità presente sul Path già stabilito e, a inizializzazione finita richiama il ClientAgent. Nel capitolo di progetto dettaglieremo questo protocollo, e le entità che vi prenderanno parte.

#### 5.2.2.7 Supporto per la mobilità del terminale

Il supporto per la mobilità del terminale viene costruito sfruttando i servizi presentati sopra. Se per la mobilità utente l'entità che veniva inizializzata nel target place era il Client, questa volta invece sarà il primo Proxy che si trova sul Path dal client al server. Secondo le politiche incapsulate nel Trigger Generator, viene notificato ai vari Client dapprima un *potenziale ingresso* in una nuova località, poi se effettivamente si entra nella nuova località, verrà notificato un *ingresso nella nuova località*. I Client possono perciò alla ricezione del primo evento cominciano l'inizializzazione del Proxy sui possibili place remoti, inviando su tali Place un Plan Visitor Agent. Una volta terminata l'inizializzazione, se effettivamente il terminale è entrato nella nuova località, verrà fatta partire la riconfigurazione dell'ultima parte del Path, stimolando il salto dell'ultimo ProxyAgent dal vecchio Place Wireless Enabled al PlaceWirelessEnable target, dove attraverso il servizio di attivazione delle entità multimediali (vedi 5.2.1.3) potrà recuperare il riferimento al Proxy già istanziato e inizializzato.

## *5.3 Conclusione*

In questo capitolo sono stati fissati i requisiti che hanno ispirato la progettazione del sistema per il supporto allo streaming. È quindi stata eseguita l'analisi di due dei tre layer, nei quali è stato suddiviso il sistema.

Nel prossimo capitolo verranno esposti progetto e implementazione delle entità costituenti i due strati qui analizzati.

CAPITOLO 6

# 6 Progettazione del middleware

In questa sezione presenteremo la progettazione dei layer analizzati nel capitolo precedente. Ipotizzeremo di utilizzare un linguaggio generico, orientato agli oggetti, definendo, per quanto possibile in modo generale, le varie entità. Nondimeno, dal momento che, per la realizzazione del progetto viene utilizzato il paradigma ad agenti mobili, dobbiamo anche riferirci a concetti e modelli di programmazione propri di questo paradigma. Sarebbe auspicabile, in tal senso, posticipare la scelta di un ambiente di sviluppo ad una successiva fase di implementazione, tuttavia non riteniamo possibile tale alternativa dal momento che ogni architettura per il supporto di agenti mobili presenta peculiarità proprie, tali da suggerirci di improntare, fin dalla fase di progetto, lo sviluppo dell'architettura, riferendoci all'ambiente SOMA, che verrà perciò esteso in modo da offrire i servizi fissati in fase di analisi.

Fatte queste premesse passiamo quindi alla presentazione del progetto.

## *6.1 Progetto del MUM Services Layer*

### 6.1.1 Location Service

Come stabilito in fase di analisi, il location service interrogato deve ritornare il path che identifica univocamente un certo place. La realizzazione del servizio consisterà nell'implementazione di un metodo facente parte dell'oggetto MUMServicesFacade, la cui interfaccia è stata introdotta in 5.2.1, che, leggendo il Path che verrà salvato come attributo dell'oggetto Environment, lo riporterà in uscita. Il problema da risolvere è come inizializzare i vari Path alla partenza del sistema o, in generale, quando un nuovo place si unisce all'architettura.

Per risolvere tale problema si è fatto uso della comunicazione tra place, fornita da SOMA (vedi 3.3.2.4), viene cioè utilizzato il paradigma REV in modo da inviare un comando al Default Place, presso il quale ci si registra in fase di inizializzazione; la root istanzierà invece semplicemente un Path contenente il nome del Default Place Root. L'esecuzione di tale comando, denominato GetPathCommand, similmente a quanto accade nel processo di registrazione dei place, una volta recuperato presso il default-place il Path che lo identifica, invoca la spedizione all'indietro di un RegisterPathCommand che, giunto presso il place

dal quale era stato lanciato il GetPathCommand, viene messo in esecuzione dal demone proposto dall'architettura SOMA, producendo l'inizializzazione della variabile Path. Tale variabile è la prima estensione della piattaforma SOMA e, come detto, viene salvata come campo dell'oggetto Enviroment (si veda 3.3.2.2). In figura 6.1 viene rappresentato questo processo.

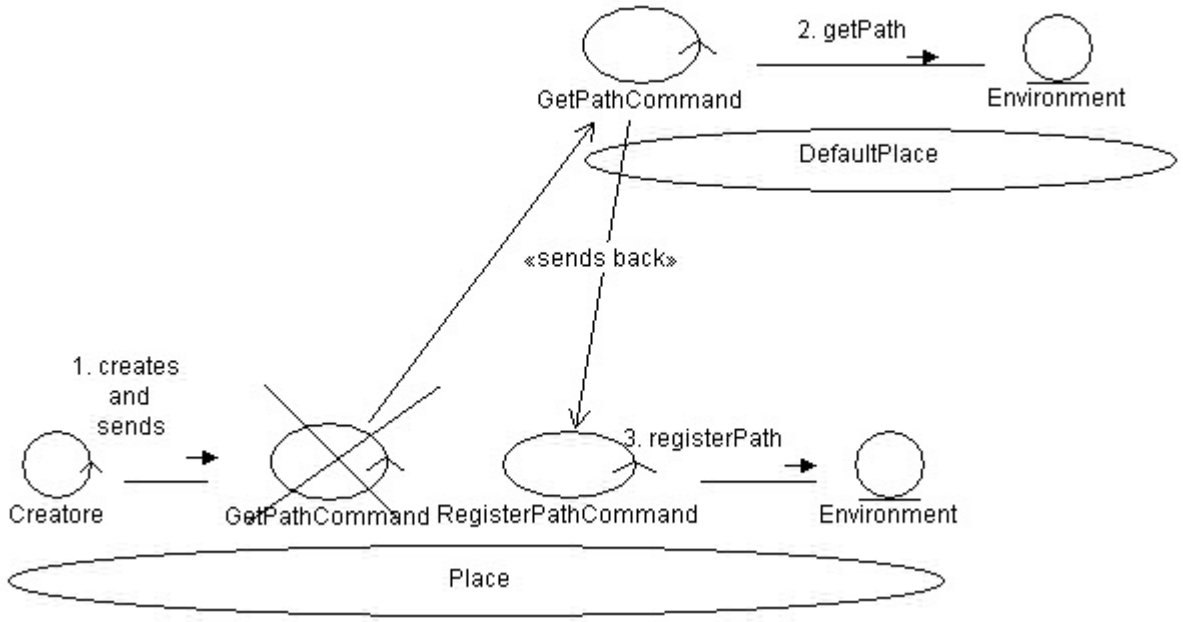


**Figura 6.1**: Location Service, inizializzazione del Path.

### 6.1.2 Sottosistema di gestione del software

Come fissato in fase di analisi, ciascun Place implementerà il software manager. Gli oggetti trattati dal software manger sono i *Descrittori delle Entità* e le *Entità*. Abbiamo introdotto il concetto di Entità in 5.2.1.2, e in figura 6.2 vengono riportati i diagrammi delle classi degli ADT (Abstract Data Type), Descrittore di Entità e Entità. Il primo, come suggerisce il nome, è un metadato che definisce l'Entità in termini delle varie classi che la compongono, mentre il secondo incapsula tutto il codice necessario alla realizzazione dell'entità. In realtà un'entità potrà fare riferimento a classi che non vengono scaricate insieme all'entità stessa, come ad esempio le librerie per la gestione dei flussi multimediali, che dovranno essere presenti su tutti i nodi che vorranno partecipare alla piattaforma. La verifica dei vincoli fra le diverse entità, e fra le entità e librerie presenti sui vari nodi partecipanti all'architettura, esula dagli scopi del middleware in progetto; si vuole infatti semplicemente offrire un mezzo per lo scaricamento dinamico del codice.

I metodi esposti dal Software Manager per l'upload/download delle entità presso un certo place, sono presentati in figura 6.3. I descrittori delle entità non vengono trattati direttamente dal Software Manager, ma sono utilizzati dal Download Agent (che considereremo più avanti) per richiedere nei singoli place visitati se sono presenti alcune delle classi che servono per la creazione di una certa entità. Infatti, sebbene abbiano una signature simile, i metodi getSingleClasses e getClassesList, differiscono per il seguente motivo: il primo ritorna una struttura dati contente il codice cercato, mentre il secondo ritorna unicamente una lista di coppie [packageName, className], che identificano univocamente le varie classi che compongono una certa entità.

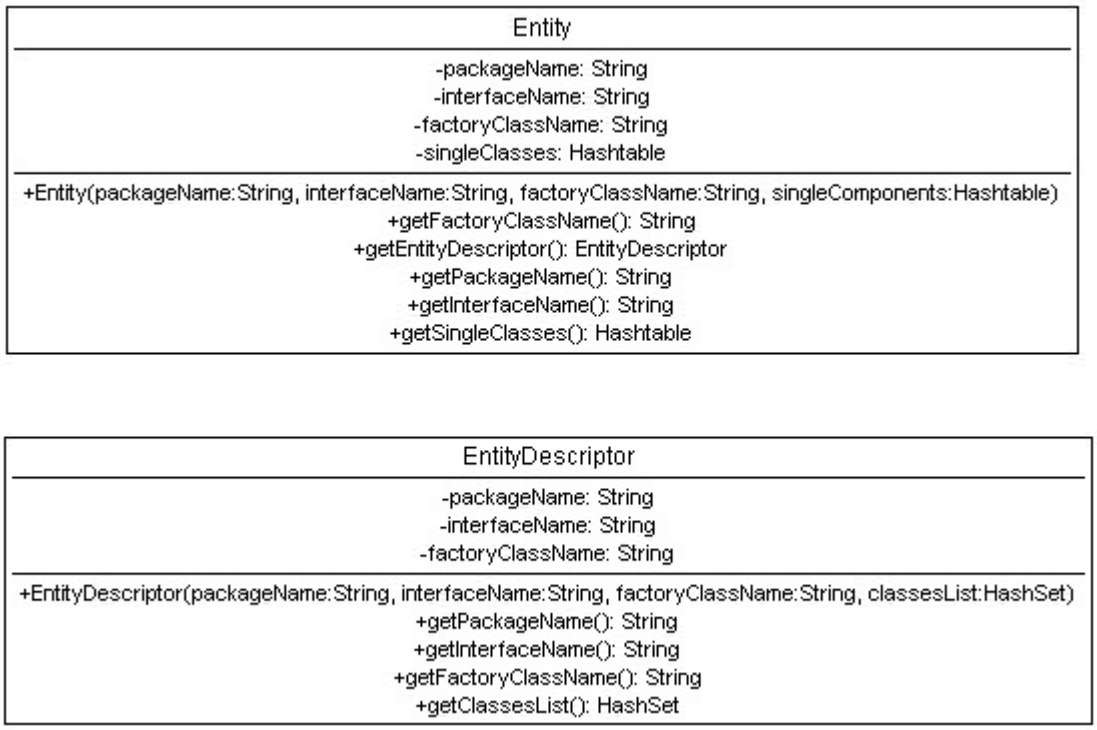


**Figura 6.2**: Entità e Descrittore dell'Entità

Il Software Manager poi, agendo da controllore, coordina due entità che raffinano l'IEntityRepository, introdotto in 5.2.1.2 e vengono usate, una per la gestione dei descrittori, ed una per la gestione del codice vero e proprio cioè l'IEntityDescriptorRepository e l'ICodeRepository. Tali entità incapsulano la logica di accesso alle basi di dati contenenti codice e descrittori, come raffigurato sempre in figura 6.3. Non interessa, in fase progettuale, stabilire come saranno realizzate tali basi di dati, rimandiamo perciò questo dettaglio alla fase implementativa.

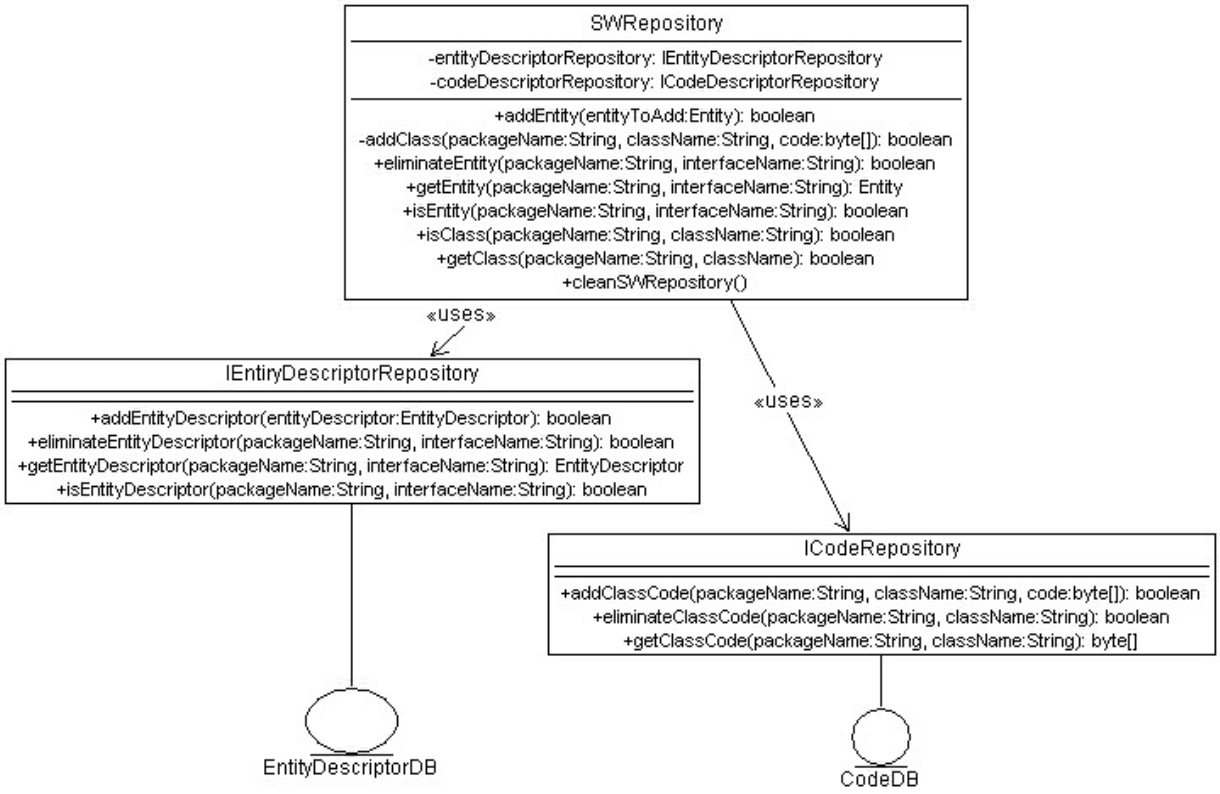


**Figura 6.3**: software repository.

### 6.1.3 Servizio di attivazione delle entità multimediali.

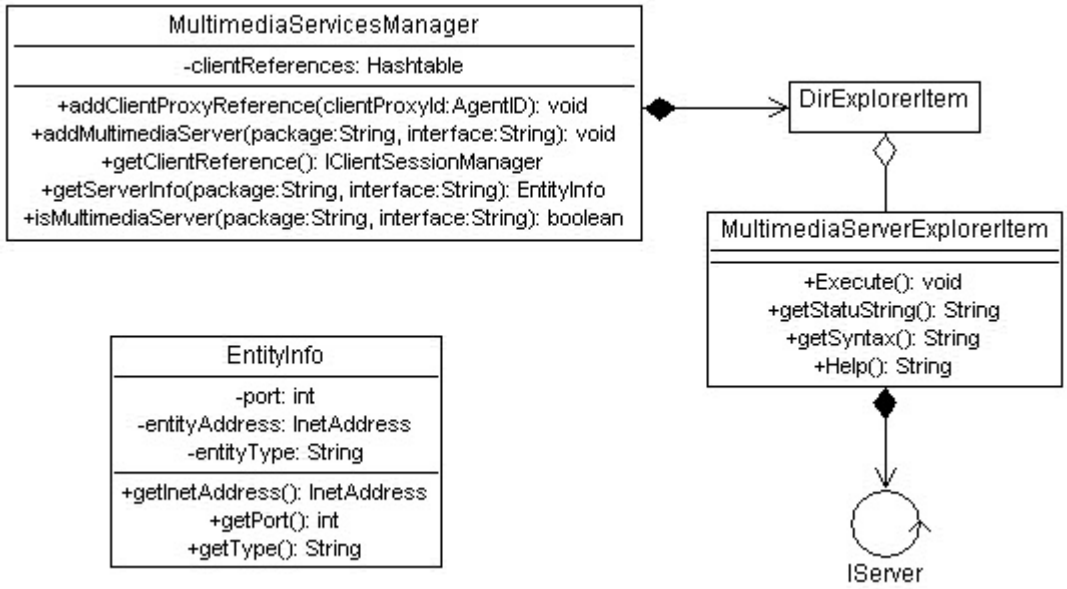


**Figura 6.4**: servizio di attivazione delle entità multimediali.

Questo servizio, come detto in fase di analisi, sarà costituito da un unico oggetto, che viene utilizzato per attivare i Server, mantenere le loro EntityInfo, e i

riferimenti ai Client istanziati (l'interfaccia del servizio viene mostrata in figura 6.4).

Il metodo addMultimediaServer riceve in ingresso l'identificatore unico dell'unità server da avviare, costituito dal nome del package e dell'interfaccia, e avvia il server richiesto. La semantica di utilizzo di tale metodo prevede che il codice richiesto sia già caricato prima che il metodo venga invocato. Dopo aver invocato questo metodo, sarà possibile ottenere un riferimento (indirizzo IP e porta) al server stesso, incapsulato nell'oggetto EntityInfo. Si nota che per il salvataggio delle varie entità server si è fatto uso, come evidenziato dal diagramma delle classi, di un DirExplorerItem, che racchiude vari MultimediaServerExplorerItem. Un DirExplorerItem è un menù introdotto dall'architettura SOMA, accedibile a riga di comando, che può contenere diverse voci di menù, in questo caso i vari MultimediaServerExplorerItem. Ognuna di queste voci permette di avviare i Server anche da riga di comando da parte di un amministratore di sistema e questi ultimi possono essere fermati e riavviati, come un qualsiasi demone facente parte dell'architettura SOMA, essendo, essi stessi, dei demoni. Di solito però saranno avviati dai PlanVisitorAgent, attraverso l'utilizzo dei metodi offerti dal servizio che stiamo esaminando; questa scelta progettuale è stata fatta per integrare la soluzione proposta con le soluzioni preesistenti, suggerite dal sistema SOMA.

L'altro metodo, offerto da questo servizio, è quello per l'aggiunta di un riferimento ad un Client che sia stato avviato su richiesta di un Plan Visitor Agent (definito in 5.2.2.4). La chiave per il recupero del riferimento al Client è rappresentata dall'identificatore del ClientAgent per il quale il client è stato istanziato, che corrisponde all'AgentID di tale agent, definito in SOMA.

### 6.1.4 Sottosistema di gestione delle risorse.

Il sottosistema di gestione delle risorse è costituito da un oggetto che espone metodi visualizzati in figura 6.5 ed è presente in ogni Place. Il metodo isResources accetta, come parametro di ingresso, una generica IResourceRequest ritornando, in uscita, una IResourceResponse. Come detto in fase di analisi tale risposta può essere: positiva, negativa o positiva con riserva. Nel caso in cui la risposta sia positiva le risorse vengono direttamente prenotate, nel caso in cui sia positiva con riserva, verrà notificata al momento dell'effettiva liberazione della risorsa, è invece negativa quando le risorse non sono disponibili.

Se il richiedente non è interessato ad attendere la liberazione di tali risorse, è suo dovere informare il gestore delle risorse attraverso il metodo

notInterestedToWait. Gli identificatori utilizzati per la prenotazione e il rilascio delle risorse devono essere unici per il singolo gestore, vengono perciò utilizzati, a tale scopo, gli AgentID dei ClientAgent.

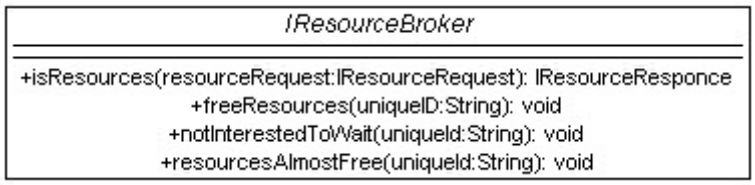


**Figura 6.5**: interfaccia del Resource Broker.

Per quanto riguarda la liberazione delle risorse in due fasi, alla quale si accennava in 5.2.1.4, sono resi disponibili due metodi: resourcesAlmostFree e freeResources. Richiamando freeResources le risorse verranno istantaneamente liberate, se invece viene richiamato il metodo resourceAlmostFree si comunica al gestore che le risorse saranno liberate entro breve. Si stabilisce a tale riguardo un accordo fra i clienti del servizio e il sottosistema di gestione delle risorse, secondo il quale il metodo resourcesAlmostFree dovrà essere richiamato N secondi prima della terminazione dello streaming, con N fissabile a piacere.

Naturalmente l'ipotesi forte sulla quale si basa questo sottosistema di gestione delle risorse, è che i clienti rispettino gli accordi stabiliti, altrimenti (se ad esempio non venissero liberate le risorse che sono state occupate), si giungerebbe ben presto ad esaurimento delle stesse.

### 6.1.5 Gestione dei contenuti multimediali.

Anche in questo caso, come nel caso dell'ottenimento del Path, considerato più sopra, viene utilizzato il paradigma REV per ottenere, alla creazione dei vari Place, un riferimento al servitore utilizzato per l'accesso ai metadati delle presentazioni multimediali. Tale servitore è unico per tutto il sistema distribuito.

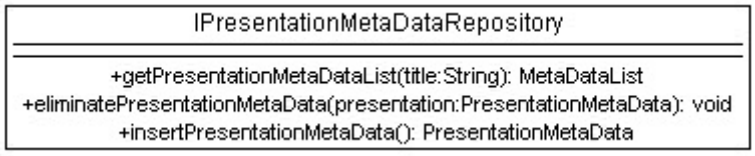


**Figura 6.6**: interfaccia di accesso alla base di dati contenente i metadati delle presentazioni.

In ogni Place viene poi creato uno stub per realizzare la comunicazione con tale server, in modo da ottenere i metadati delle presentazioni cui si è interessati. L'interfaccia implementata sia dallo stub che dal repository dei dati multimediali, è rappresentata in figura 6.6, mentre l'architettura di tutto il servizio è rappresentata in figura 6.7.

Per la comunicazione tra gli stub ed il server viene utilizzato il protocollo TCP ed il server è multithreaded; per ogni nuova richiesta viene cioè creato un nuovo thread che gestirà la richiesta stessa. I comandi, passati dal Client al server sono, così come le risposte, oggetti serializzati. I metadati riguardanti la presentazione descritta in 5.2.1.5 saranno perciò realizzati come oggetti serializzabili, in modo da poter essere spediti indietro dal server in risposta alle richieste ricevute dai Client-stub.

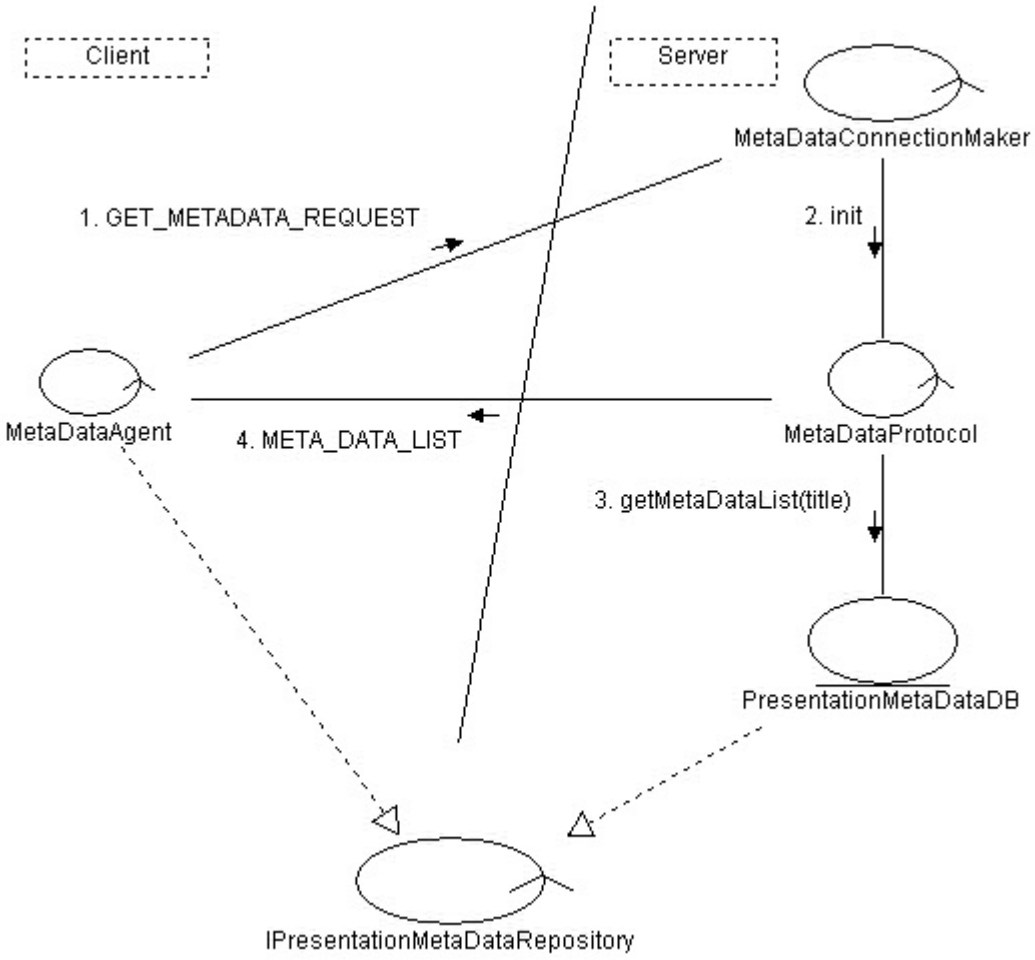


**Figura 6.7**: realizzazione del servizio per lo scaricamento dei metadati

### 6.1.6 Gestione dei profili

Per la gestione dei profili viene utilizzata un'architettura assai simile a quella considerata al punto precedente, con l'unica differenza che i dati che vengono trattati sono, in questo caso, i profili degli utenti e i profili delle diverse piattaforme. In figura 6.8 riportiamo i diagrammi delle classi dell'oggetto per l'accesso al servizio (che implementa cioè i metodi per il salvataggio e il

reperimento dei dati) e i diagrammi delle classi del profilo utente e dei profili delle piattaforme.

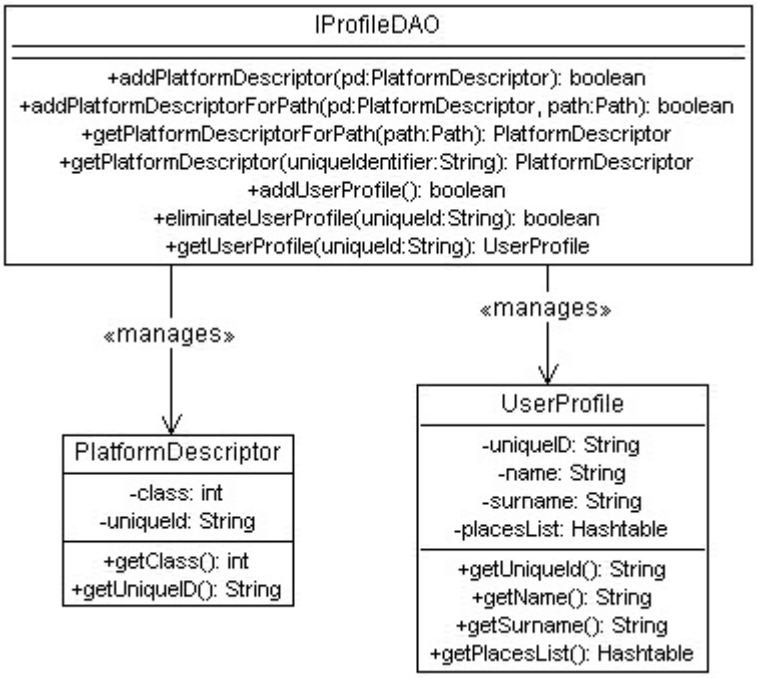


**Figura 6.8**: gestione dei profili utenti e dei descrittori delle piattaforme.

### 6.1.7 Servizi di supporto alla mobilità del terminale

Il servizio implementa i diversi oggetti evidenziati in 5.2.1.7; in figura 6.9 rappresentiamo i diagrammi delle classi dei diversi oggetti che realizzano il servizio. Il TargetLocalizer viene interrogato all'ingresso del PlaceWireless nell'architettura e serve per stabilire le corrispondenze fra gli indirizzi MAC, ricavati attraverso l'utilizzo delle WirelessAPI, e i Path dei Place Wireless Enbled. Vogliamo a questo punto specificare meglio quanto stabilito in fase di analisi per l'invio di eventi, in risposta ai movimenti del terminale.

Il TriggerGenerator, stimolato dal MovementSensor può generare i seguenti eventi:

- *Nessun evento*: questo è il caso in cui il terminale non si è spostato tanto da richiedere una notifica del fatto; come già detto ci si baserà sull'analisi della potenza dei segnali, stabilendone delle soglie.
- *Potenziale ingresso* in un nuovo BSS: questo evento viene lanciato quando si pensa che il terminale stia entrando in un nuovo BSS, cioè quando vengono superate una o più soglie. In questo caso l'evento viene notificato a tutti i ClientAgent, che possono perciò iniziare l'inizializzazione del Proxy sul/sui Place indicati.

- *Ingresso avvenuto* in un nuovo BSS: questo evento viene lanciato quando, verificando il BSS identifier del terminale, si vede che effettivamente la stazione risulta associata ad un nuovo BSS.
- *Ingresso non avvenuto*: questo evento viene lanciato quando l'ingresso che ci si aspettava in un nuovo BSS non è avvenuto.

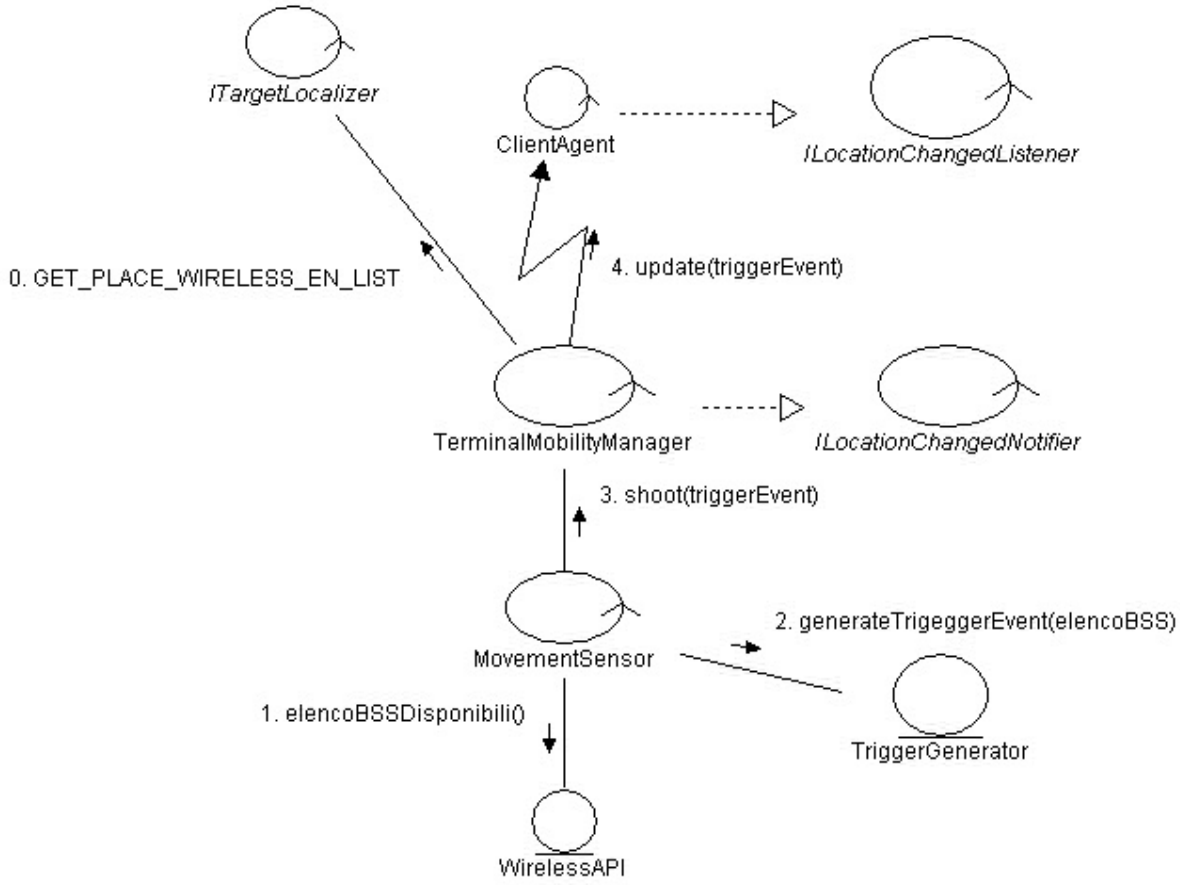


**Figura 6.9**: servizio per il supporto alla mobilità del terminale.

Perciò, quando si pensa che sia possibile l'ingresso in una o più nuove località, si predispone tutto in modo da rendere poi possibile il salto nella nuova località e si stimola il salto vero e proprio solo nel momento in cui effettivamente si è certi che tale evento si è verificato. Per generare gli eventi in questo modo il TriggerGenerator viene progettato come un automa a stati finiti, costituito da due stati: uno stato di riposo, nel quale si è stato stabilito il BSS al quale si è associati e non si sono verificati potenziali ingressi, e uno stato di potenziale ingresso, nel quale, una volta transitato dallo stato di riposo, l'automa a stati finiti rimane fino a quando:

- Non venga effettivamente rilevato l'ingresso in un BSS target diverso da quello di partenza;

- Oppure non scatti un timeout, che indica la terminazione di questa fase decisoria e riporta l'automa nello stato di riposo, dal quale potrà nuovamente iniziare una nuova fase decisoria.

Ipotesi forte sulla quale si basa la realizzazione della gestione dell'hand-off della sessione è che non ci siano disconnessioni per quanto riguarda la fruizione del servizio sulla rete wireless. Si vuole infatti unicamente gestire il roaming del terminale, mentre la gestione di eventuali disconnessioni durante la fruizione del servizio viene lasciata a successivi approfondimenti. Rimane un problema aperto: cosa fare dei Proxy che siano stati istanziati, senza poi venire effettivamente utilizzati. La soluzione che si propone è la distruzione degli stessi a carico del PlanVisitorAgent; infatti, una volta istanziato il Proxy, il PlanVisitorAgent aspetta dal ClientAgent la conferma di un salto del ProxyAgent, oppure una richiesta di distruzione del Proxy. L'uno o l'altra di queste alternative verrà scelta in base all'evento ricevuto dal ClientAgent, come stabilito sopra.

## *6.2 Progetto del MUM Middleware Layer*

### 6.2.1 Servizio di downloading del codice

Per quello che riguarda questo servizio rimangono da fissare le politiche di downloading; nel progetto finale è stata implementata la politica che esponiamo nella trattazione successiva.

Il DownloadAgent parte dal place presso il quale è stato istanziato e procede verso l'alto della gerarchia SOMA. Ad ogni place incontrato durante la risalita, richiede se presso tale place siano presenti una o più delle Entità che deve scaricare; se sono presenti le memorizza, quindi procede verso l'alto finché non riesce a trovare tutte le entità richieste (eventualmente arrivando fino alla root). A questo punto ritorna al place dal quale era stato lanciato, aggiornando il SWRepository. Nel caso in cui il place dal quale l'agente viene lanciato non sia un default place, durante la via del ritorno l'agente si ferma anche presso il default place, in modo da salvare il codice anche su di esso.

La politica è stata scelta cercando di ottimizzare, per quanto possibile, il processo dello scaricamento di codice. In particolare la scelta di fare risalire nodo per nodo l'agente è stata fatta per rispondere ad un principio di scalabilità del sistema. Si vuole cioè, per quanto possibile, fare in modo che, dopo una prima fase in cui in tutto il sistema i diversi SW repository sono vuoti, non appena sia disponibile del codice ai livelli più bassi dell'albero, le richieste vengano servite "il più vicino possibile" al nodo richiedente, in termini dell'astrazione di località

introdotta. In questa stessa direzione va anche la scelta di salvare comunque sul default place più vicino al richiedente il SW richiesto, in modo che le richieste di tutti gli altri place appartenenti al medesimo dominio possano essere servite direttamente.

### 6.2.2 Architettura delle entità che effettuano lo streaming

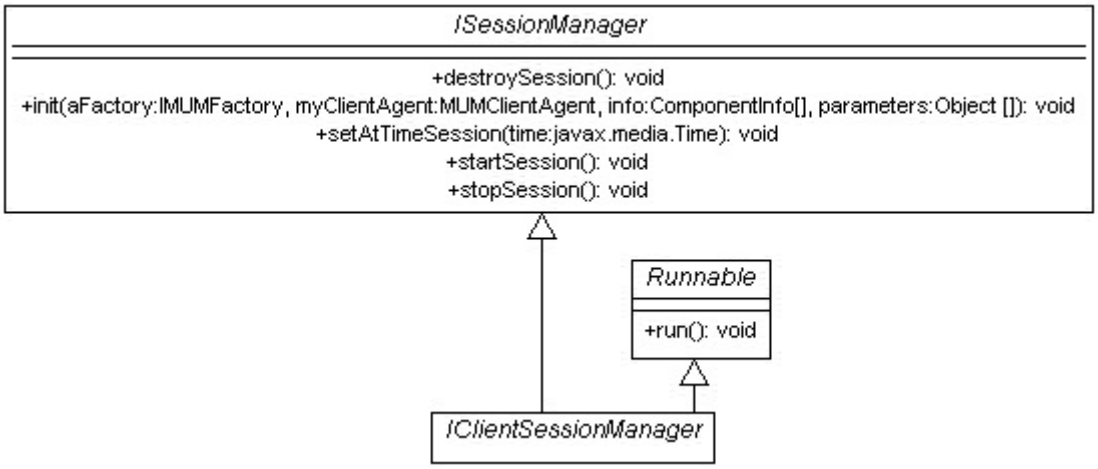


**Figura 6.10**: interfaccia del generico Client.

Tali entità verranno dettagliate nel capitolo che segue, nel quale si presenta un'applicazione per lo streaming video sviluppata al di sopra dell'architettura proposta. A questo livello ci interessa però fissare le interfacce delle diverse entità. Il design del PlanVisitorAgent dovrà infatti essere abbastanza generico in modo che, come detto in fase d'analisi, abbia la possibilità di istanziare le diverse entità pur non conoscendole, o meglio conoscendo unicamente le loro interfacce che vengono riportate di seguito.

Ciascuna delle entità implementa l'interfaccia Runnable (facendo riferimento al linguaggio Java), è cioè costituita da almeno un thread. Vedremo come avviene l'inizializzazione delle entità nella sezione seguente. Un importante osservazione è sulla gestione delle interazioni fra due oggetti uno dei quali sia un agente. Si è deciso di gestire l'interazione da un generico oggetto ad un agente attraverso l'uso delle interfacce in modo che le entità realizzate non siano necessariamente legate alla piattaforma SOMA, ma possano essere riutilizzate in diversi ambiti. Come evidenziato nel metodo di inizializzazione in figura 6.10, viene cioè passato un riferimento ad una generica interfaccia MUMClientAgent, e non un riferimento meno generale ad un Agent.

Riportiamo in figura 6.11 anche le interfacce di Server e Proxy. Si nota che mentre il Server viene progettato come demone, che una volta attivato resterà

attivo nel place di attivazione, i Proxy invece hanno corrispondenza 1:1 con gli ProxyAgent ed hanno un ciclo di vita che è legato al loro.

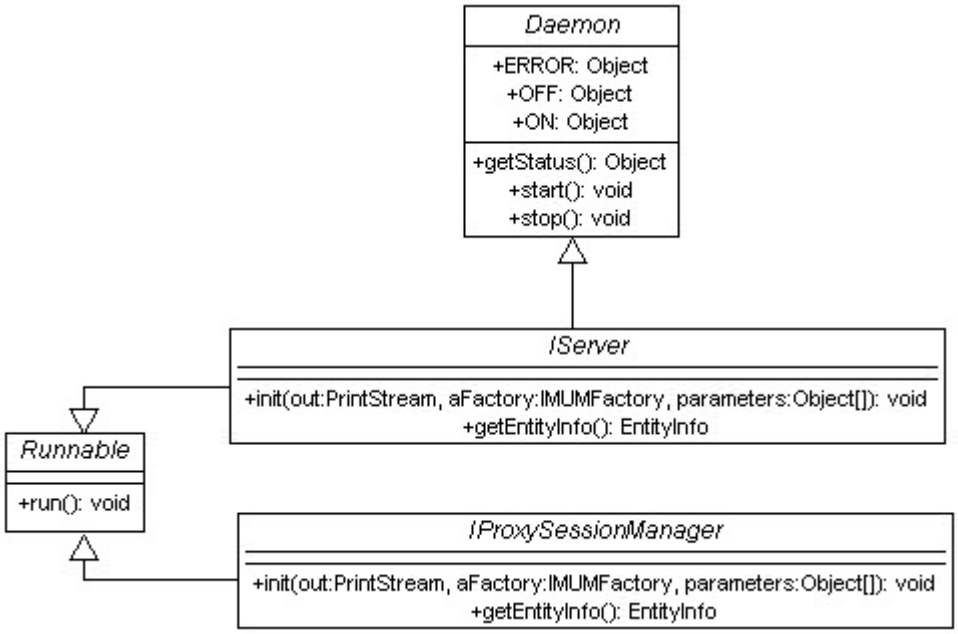


**Figura 6.11**: interfaccia del generico Proxy.

### 6.2.3 Servizio di inizializzazione e riconfigurazione dinamica del sistema

La progettazione del sottoinsieme di inizializzazione e riconfigurazione richiede la progettazione dei seguenti oggetti:

- Il DecisionMaker
- Il Plan
- Il PlanVisitorAgent
- L'InitManager.

Il **DecisionMaker** espone i metodi mostrati in fig. 6.12 in particolare, a parte il metodo utilizzato per l'inizializzazione del DecisionMaker, sono implementati anche altri tre metodi.

- Il primo, getPlanForPresentation, ritorna un generico Plan di inizializzazione del sistema; tale piano verrà poi passato al PlanDigitorAgent che dovrà effettuare l'inizializzazione del sistema.
- Il secondo metodo esposto è getPlanForClientMovement. A tale metodo viene passato il Path che identifica univocamente il TargetPlace verso il quale si vuole muovere l'utente nomadico e ritorna un piano che, passato al PlanVisitorAgent, viene appunto utilizzato per gestire l'hand-off della sessione verso il suddetto TargetPlace.

- Il terzo metodo, cioè il getPlanForTerminalMovement serve per richiedere l'inizializzazione di un Proxy nella località verso la quale si sta muovendo il terminale mobile.

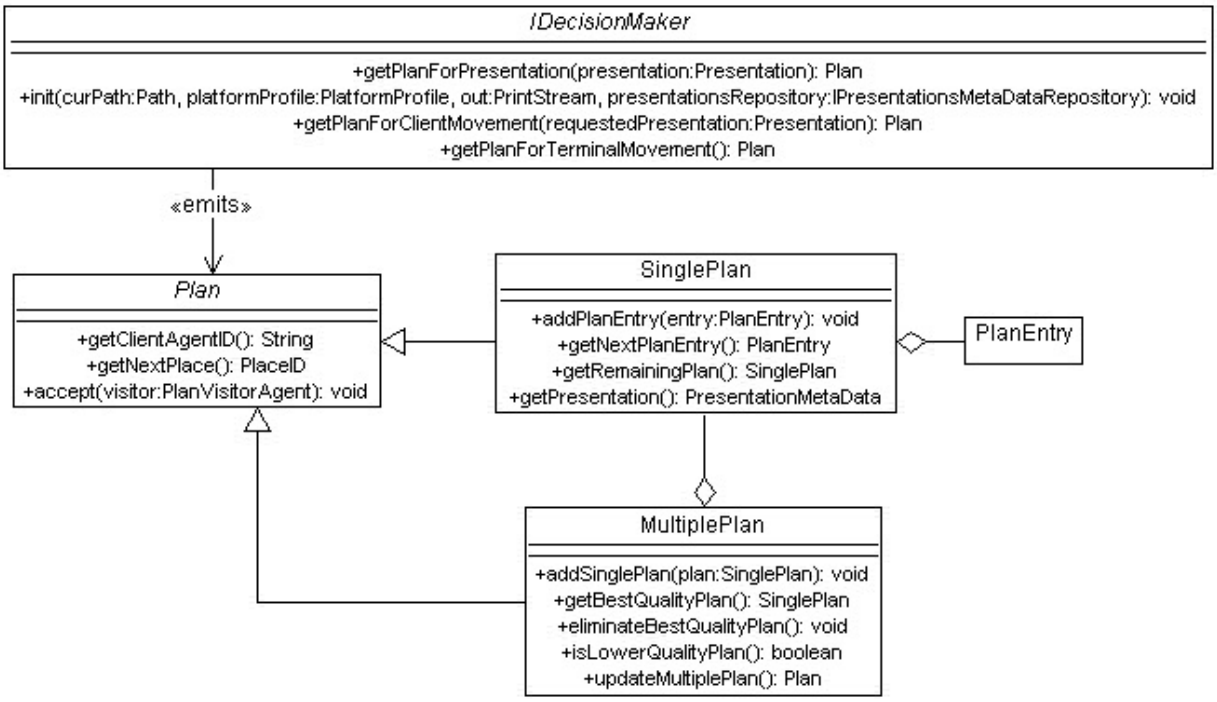


**Figura 6.12**: il DecisionMaker e i piani di inizializzazione.

Il **Plan** è il dato utilizzato per incapsulare le informazioni che riguardano l'inizializzazione e la riconfigurazione del sistema. Esso espone tre metodi:

- getClientAgentID, che ritorna il ClientAgent per il quale è stato emesso il Plan.
- getNextPlace, che ritorna il prossimo Place presente nel Plan, se c'è, oppure con puntatore a null.
- accept, questo metodo accetta un PlanVisitorAgent. Viene cioè applicato il pattern Visitor per l'interpretazione dei diversi piani di inizializzazione, in modo da facilitare l'introduzione di nuovi piani e l'estensione dei PlanVisitorAgent.

Sono poi implementati due tipi di Plan: il primo tipo, **SinglePlan**, prevede l'inizializzazione di un unico percorso senza la possibilità di inizializzare percorsi alternativi, in caso di carenza di risorse sul percorso considerato, il secondo, **MultiplePlan**, prevede, invece, la possibilità di avere diversi percorsi di inizializzazione in modo tale che l'Agente possa scegliere il percorso che meglio si addice alla situazione delle risorse nel sistema distribuito.

Per un MultiplePlan si cercherà cioè di inizializzare il percorso per la presentazione a qualità migliore e, nel caso in cui si verifichi la mancanza di risorse per tale presentazione, si cercherà di inizializzare la presentazione a qualità inferiore e così via fino a che non si riesce a concludere un'inizializzazione o finiscono le alternative. In tal caso verrà comunicata al ClientAgent l'impossibilità di fruizione della presentazione per carenze di risorse nel sistema.

Il MultiplePlan è costituito da due o più SimplePlan, mentre il SimplePlan è costituito da più **PlanEntry**. Ciascuno di essi incapsula informazioni concernenti il prossimo Place sul quale devono transitare i PlanVisitorAgent, il software da scaricare e l'identificatore dell'entità da inizializzare (se c'è), univocamente determinata dalla tupla [packageName, interfaceName].

Man mano che il PlanVisitorAgent procede, vengono poi richiesti al Plan degli aggiornamenti dello stesso; il Plan verrà aggiornato in modo che vengano passati in avanti solo quei SinglePlan che hanno il prossimo Place del percorso in comune col prossimo Place del percorso attualmente scelto, realizzando così le politiche di posizionamento dei Proxy come fissato al punto 5.2.2.5. Compito del DecisionMaker è quindi costruire tali piani. Si nota che questo meccanismo è notevolmente flessibile: basta infatti cambiare l'implementazione del DecisionMaker per variare le politiche di inizializzazione e riconfigurazione dell'intero sistema, tale meccanismo inoltre, anche grazie all'utilizzo del pattern Visitor, si presenta come intrinsecamente estendibile.

Il **PlanVisitorAgent** è l'Agente utilizzato per l'interpretazione dei piani. Considereremo qui il caso dell'interpretazione di un MultiplePlan, essendo l'interpretazione di un SinglePlan un caso particolare di quest'ultimo.

Ricordiamo, come già detto in fase di analisi, che il PlanVisitorAgent è l'entità che compie la negoziazione statica della QoS. Vediamo cosa accade. Il ClientAgent richiede l'inizializzazione del sistema all'InitManager (che descriviamo più sotto) e si pone in attesa di un evento di terminazione di inizializzazione che, come detto, potrà avere esito positivo o negativo. L'InitManager, quindi, richiede, attraverso il metodo getPlanForPresentation, un piano di inizializzazione al DecisionMaker. Quest'ultimo, consultando il metadata Repository, ricava una lista di tutte le presentazioni rispondenti a tale titolo; tra queste presentazioni se ne scelgono poi alcune che verranno incluse nel piano di inizializzazione stesso.

Il DecisionMaker, attualmente implementato, adotta questa politica: viene esaminata la lista delle presentazioni e, per ogni qualità di servizio disponibile,

viene scelta la presentazione che si trova più vicina al cliente (secondo l'astrazione di località introdotta); vengono poi costruiti i diversi piani di inizializzazione per ognuna di tali presentazioni.
Il risultato di questo processo è la creazione di un MultiplePlan. Una volta in possesso del MultiplePlan l'InitManagent crea un PlanVisitorAgent e glielo passa facendo poi partire l'Agente. Il PlanVisitorAgent inizia perciò a visitare il piano nel modo descritto sopra, cercando di istanziare il percorso per la migliore delle entità (risorse permettendo), ed eventualmente optando per presentazioni sempre peggiori. Quando anche il Server, ultima entità del percorso, è stato istanziato, vengono passati all'indietro gli EntityInfo, utilizzando lo scambio di messaggi fra agenti, offerto da SOMA. Gli EntityInfo incapsulano gli end-point necessari per la comunicazione fra le varie entità, che sarà sempre iniziata dal Client alla volta del Server. Quando l'end-point del primo Proxy del percorso viene comunicato al primo PlanVisitorAgent istanziato, quest'ultimo può terminare l'inizializzazione del Client, passandoglielo, quindi comunicare l'avvenuta terminazione dell'inizializzazione all'InitManager, che la notifica al ClientAgent, che da questo momento in poi può utilizzare il Client per gestire lo streaming.

L'ultima entità che consideriamo è l'**InitManager**. Tale entità viene utilizzata ogni qualvolta si renda necessaria una riconfigurazione del sistema, quindi, per la realizzazione dell'inizializzazione, per la riconfigurazione del sistema in caso di degrado della QoS e per le riconfigurazioni necessarie per il supporto della mobilità utente e del terminale.

### 6.2.4 Sottosistema di gestione della qualità di servizio.

Per quanto riguarda la negoziazione iniziale della QoS e la prenotazione delle risorse, non ci dilungheremo oltre, dal momento che questo argomento è stato ampiamente trattato al punto precedente. Vogliamo invece qui presentare il progetto degli oggetti che realizzano la traslazione delle specifiche di QoS e la gestione dinamica della stessa. I QoSManager, infatti, non interagiranno direttamente col Servizio di Prenotazione Risorse, ma le varie richieste verranno traslate dal ResourceTranslator.

Come detto in fase di analisi (vedi 5.2.1.6) sono definite M classi di piattaforme. Per ciascuna di esse viene fissato un coefficiente di conversione in modo che la richiesta di CPU, presente all'interno del metadato, e che si riferisce ad una piattaforma di classe M (classe con le maggiori potenzialità), sia ricondotta ad una più realistica richiesta di CPU per la piattaforma sulla quale

avviene la richiesta stessa. Le altre due entità per la gestione delle risorse da introdurre sono il QoS Manager ed il CPU Monitor.

Inizieremo descrivendo il CPUMonitor. Tale oggetto realizza il monitoraggio della CPU e, nel caso in cui l'utilizzo della CPU nell'ultimo intervallo considerato superi una certa soglia, lancia un evento alla volta dei QoSManager, i quali, in risposta a tale evento, cercheranno di passare ad una presentazione a qualità più bassa, in modo da liberare delle risorse di sistema, senza dover però chiudere la sessione. Per la realizzazione del CPUMonitor ci si appoggia su di una libreria sviluppata presso questo stesso dipartimento per il monitoring di sistema (si veda la tesi di laurea [ARL99]); in particolare viene utilizzato l'oggetto ProcessMonitor per ottenere informazioni circa l'utilizzo totale della CPU da parte dei diversi processi.

Il CPUMonitor è quindi un Thread che, con una prefissata frequenza, monitora la situazione. Tale oggetto dovrà inoltre rendere disponibili metodi per l'aggiunta e la rimozione di Listener ai quali, come visto sopra, verrà notificata la situazione critica di utilizzo della CPU. In particolare vengono definite alcune soglie, al superamento di ciascuna delle quali viene notificato l'evento con l'indicazione della soglia superata.

Si ribadisce, come già detto in fase di analisi, che ciascuna delle entità partecipanti all'architettura avrà un proprio QoSManager che è incaricato della gestione della QoS a tempo di esecuzione. Oltre a ricevere gli eventi relativi all'utilizzo della CPU dal CPUMonitor realizza esso stesso il monitoraggio della banda trasmissiva, come vedremo nel prossimo capitolo, e dipendentemente da queste due informazioni può decidere di richiedere una fase di riconfigurazione del sistema. La rilevazione di una situazione critica può riguardare il Client, così come il Proxy o anche il Server; si sottolinea però che la riconfigurazione del sistema dovrà necessariamente partire da uno dei Proxy (e tipicamente avverrà così) o, nel peggiore dei casi, dovremo ristabilire tutto il percorso e le varie entità, partendo dal Client.

Per la fase di riconfigurazione del sistema verrà utilizzato il protocollo descritto successivamente, che sfrutta le capacità di comunicazione degli agenti, ed in particolare la possibilità di spedirsi messaggi. Per prima cosa si vogliono evitare conflitti tra due entità che richiedano contemporaneamente una riconfigurazione del sistema. Viene perciò stabilito che, se il QoSManager comunica la necessità di una configurazione al proprio ProxyAgent quest'ultimo, per prima cosa, richieda al ClientAgent il permesso di iniziare una fase di riconfigurazione. Il ClientAgent può rispondere in modo affermativo, se non è già

stata fatta una simile richiesta, oppure negativamente, nel caso in cui tale richiesta sia già stata fatta (in tal caso il richiedente, almeno in questo primo prototipo, abbandona il proprio proposito fino a quando il QoSManager non chieda una nuova riconfigurazione). Il ProxyAgent che riceve risposta affermativa, può fare partire la fase di riconfigurazione. In tale fase sarà utilizzato un piano alternativo, che sarà certamente presente dal momento che i Proxy sono stati posti solo nei place per i quali fosse disponibile almeno un piano alternativo; tale piano viene passato all'InitManager per la riconfigurazione. Una volta terminata la riconfigurazione del sistema viene notificata la terminazione della riconfigurazione al ClientAgent, in modo che possa rispondere affermativamente a nuove richieste di riconfigurazione, e con ciò viene conclusa la fase di riconfigurazione.

Un caso particolare si ha quando la richiesta giunga da un QoSManager che stia monitorando un Server; in questo caso infatti non abbiamo nessun agente associato al manager, si decide perciò che il QoSManager possa creare un agente, in modo da utilizzare il meccanismo di comunicazione fra agenti (basato sui messaggi) per comunicare la volontà di fare partire una riconfigurazione al ProxyAgent che lo precede nel percorso ed agirà in sua vece.

### 6.2.5 Supporto alla mobilità utente

In figura 6.13 viene rappresentata la gestione dell'hand-off della sessione per il movimento degli utenti.Il protocollo seguito è il seguente:

- il ClientAgent, dopo aver richiesto all'utente il place verso il quale si vuole muovere, ed aver ricavato il Path dallo UserProfile, richiede all'InitManager l'inizio dell'hand-off, passandogli il Path che localizza il place verso il quale l'utente si vuole muovere e l'AgentID dell'ultimo Proxy nel percorso;
- l'InitManager crea un PlanVisitorAgent e gli passa un nuovo tipo di Plan, il ClientMovementPlan, per la configurazione del Client sul nuovo place. In questo piano sono contenute informazioni relative al Client da istanziare e al primo ProxyAgent nel percorso;
- all'atto dell'inizializzazione sul Target Place, il PlanVisitorAgent contatta il primo ProxyAgent nel Path, in modo da inizializzare una sessione fra il Proxy e il Client e ritorna un messaggio per notificare l'avvenuta inizializzazione (operazione 6 nel Collaboration Diagram);
- l'InitManager notifica poi la terminazione dell'hand-off della sessione stimolando il movimento del ClientAgent alla volta del Target Place;

- giunto sul nuovo Place, utilizzando il servizio di attivazione delle entità multimediali, il ClientAgent può procurarsi il riferimento al proprio Client.

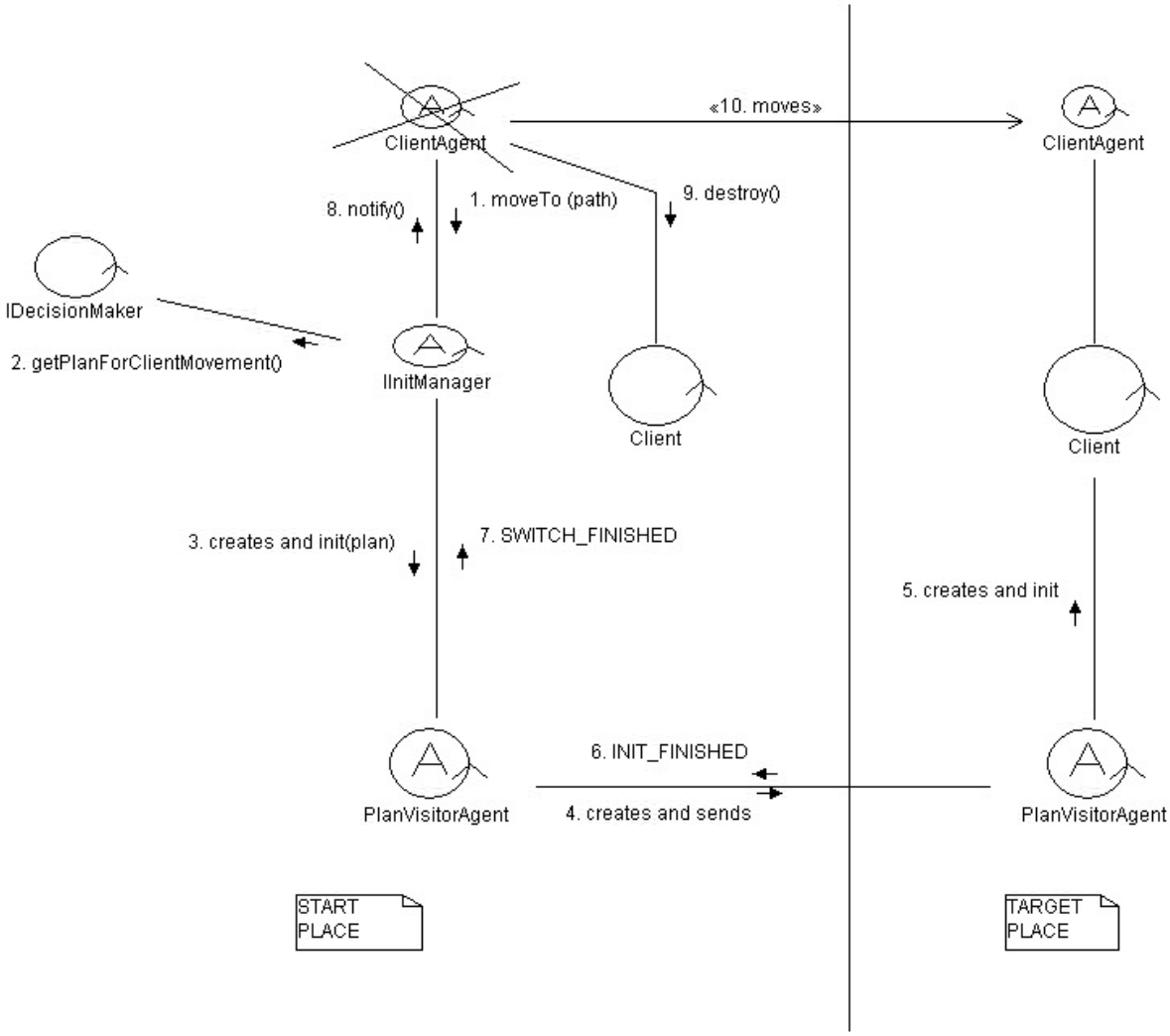


**Figura 6.13**: hand-off della sessione per utente nomadico.

### 6.2.6 Supporto alla mobilità terminale

Il supporto per la mobilità del terminale agisce in modo assai simile a quello per la mobilità utente, con la differenza che questa volta l'agente che deve essere mosso è il ProxyAgent e non il ClientAgent. In figura 6.14 riportiamo schematicamente, per non complicare troppo la figura, il diagramma di collaborazione relativo alla gestione dell'hand-off durante il movimento del terminale.

In questo caso la richiesta di inizio dell'hand-off viene stimolata dall'evento lanciato dal TerminalMobilityManager analizzato sopra. Viene quindi messa in atto una fase di inizializzazione del Proxy nella nuovo place wireless enabled, verso il quale ci si sta ipoteticamente muovendo, e solo all'arrivo di un

nuovo evento che confermi l'effettiva associazione ad un nuovo BSS, viene stimolato il movimento del ProxyAgent, che può ricavare il riferimento al Proxy richiedendolo al PlanVisitorAgent che lo ha istanziato. In caso invece la previsione fosse errata viene richiesta la distruzione del Proxy sull'ipotetico nuovo place attraverso un messaggio spedito al PlanVisitorAgent che l'ha istanziato.

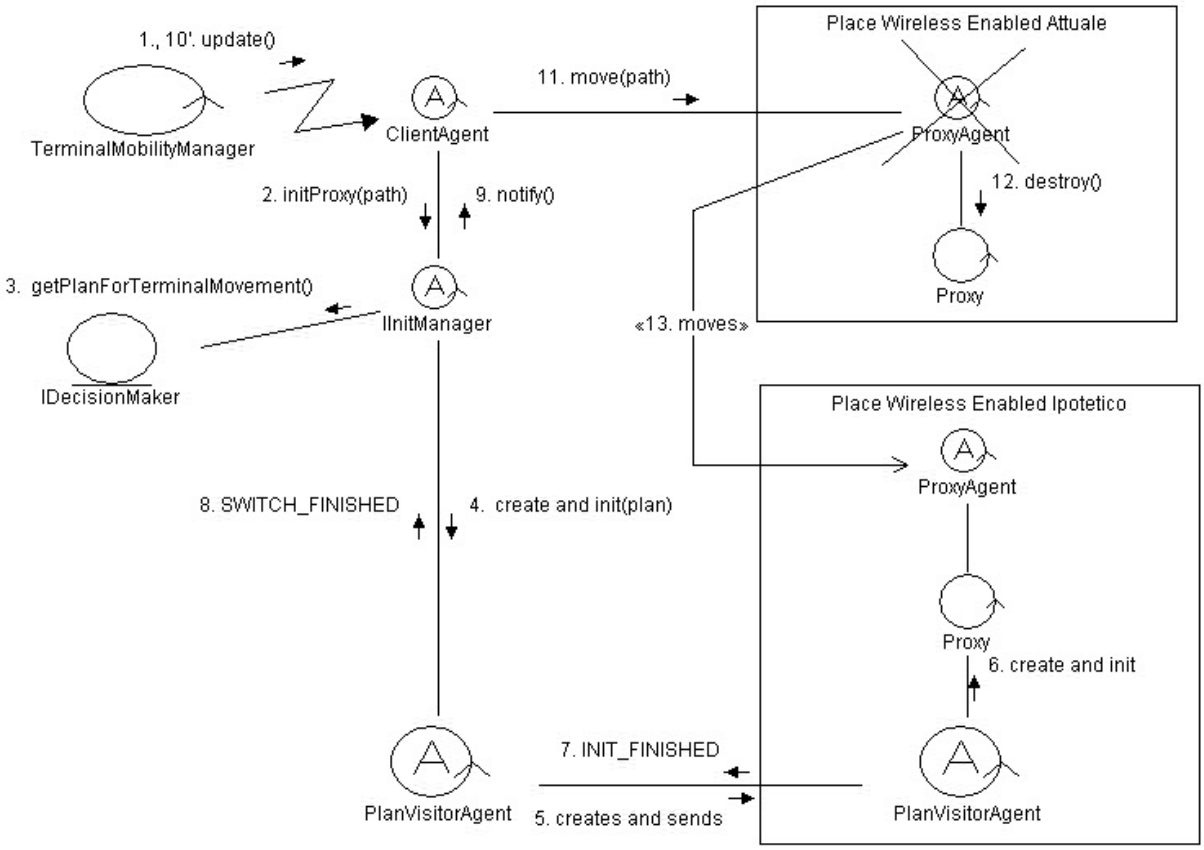


**Figura 6.14**: gestione dell'hand-off durante il movimento del terminale.

## *6.3 Alcune tracce sulle scelte implementative*

Per quello che riguarda l'implementazione del sistema si è optato per la scelta di Java come linguaggio di programmazione. Molti dei discorsi fatti in fase di progetto restano comunque validi indipendentemente dal linguaggio scelto.

Ovviamente alcune scelte sono state obbligate dagli strumenti utilizzati. La scelta di Java, infatti, è risultata naturale dal momento che l'ambiente di supporto degli agenti mobili SOMA, è sviluppato in tale linguaggio così come le Wireless API e le librerie utilizzate per il monitoring.

La scelta di questo linguaggio assicura comunque la portabilità del sistema realizzato al di sopra di diverse piattaforme con l'unico vincolo di avere a

disposizione il porting di tutte le librerie utilizzate per tali piattaforme. Come visto le Wireless API sono già disponibili per piattaforme Linux e Windows XP, così come le librerie per il monitoring.

L'implementazione finale consta di 13 package nei quali sono organizzate le 140 classi costituenti il middleware progettato.

## *6.4 Conclusione*

Con questo capitolo abbiamo presentato dettagliatamente il progetto del middleware oggetto di questo lavoro di tesi. Abbiamo seguito l'organizzazione proposta in fase di analisi considerando dapprima il progetto del MUM Services Layer, e poi quello del MUM Middleware Layer.

Nel prossimo capitolo si presenta un'applicazione per lo streaming video sviluppata al di sopra dell'architettura proposta, e il risultato dei primi test operati sul sistema.

CAPITOLO 7

# 7 Progettazione di un'applicazione per video streaming

In questo capitolo presentiamo un'applicazione che realizza lo streaming di un flusso video al di sopra del middleware proposto. Dismettiamo perciò i panni dello sviluppatore di middleware per "indossare" quelli di un potenziale sviluppatore di applicativo che si avvalga del supporto da noi progettato.

Le entità che dobbiamo sviluppare sono tre:

- il Client;
- il Proxy;
- il Server.

Una volta progettate tali entità dovremo poi implementare il DecisionMaker in modo che realizzi dei piani che prevedano l'utilizzo di queste tre entità. La gestione della configurazione iniziale del sistema, della sua eventuale riconfigurazione, della mobilità dell'utente e del terminale viene invece completamente gestita dal supporto. Ne segue che lo sviluppatore non dovrà, perciò, occuparsi di questi aspetti. Come stabilito nel capitolo precedente al punto 6.2.4 l'ultima entità che lo sviluppatore dovrà implementare è poi il QoSManager. I motivi principali di questa scelta sono due: il primo è che si vuole lasciare la libertà di realizzare il monitoraggio della banda sfruttando le caratteristiche dello specifico protocollo utilizzato per la trasmissione del flusso multimediale che verrà deciso all'atto dell'implementazione delle diverse entità, il secondo è che si vuole lasciare la libertà di decidere le politiche di adattamento da parte degli sviluppatori.

Il capitolo è organizzato come segue: dapprima si presentano brevemente il Java Media Framework (JMF) e il protocollo Real Time Protocol (RTP, vedi [RFC1889]) che sono rispettivamente la libreria usata per lo sviluppo delle diverse entità ed il protocollo utilizzato per lo streaming, poi viene presentato il design delle singole entità. Da ultimo si presentano alcuni risultati sperimentali riguardanti il processo di inizializzazione del sistema.

## *7.1 Java Media Framework*

Il Java Media Framework sono API (*application programming interface*) create per permettere di incorporare tipi di dati Multimediali in applicazioni o

applet Java; si tratta di un pacchetto opzionale, installabile per estendere le funzionalità della piattaforma JAVA2SE™. Questo componente è stato sviluppato congiuntamente da Sun e IBM, ed è nato con l'intento di fornire un supporto per i più comuni standard di memorizzazione dei dati multimediali, quali: MPEG-1, MPEG-2, Quick Time, AVI, WAV, AU e MIDI.

Com'è ben noto, la peculiarità di Java è l'utilizzo di una Java Virtual Machine (JVM) che interpreta il byte-code generato dal compilatore Java. Questo meccanismo permette la portabilità del codice Java su più piattaforme ma, nel caso in cui sia richiesta un'elevata velocità di esecuzione, impone dei seri vincoli di prestazioni. Il trattamento di dati Multimediali è uno dei casi in cui è richiesta un'elevata velocità computazionale (decompressione delle immagini, rendering, ecc.), e quindi, dove non è sufficiente la sola emulazione della CPU per ottenere delle prestazioni ottimali.

Ogni sviluppatore che desideri implementare un lettore multimediale in Java, e voglia ottenere delle prestazioni eccellenti deve, necessariamente, ricorrere a codice nativo della piattaforma alla quale è interessato. Questo procedimento comporta due problemi:

- Occorre una specifica conoscenza delle funzioni native da parte del programmatore.
- Un programma Java che utilizza codice nativo non è più trasportabile su piattaforme diverse da quella originaria.

L'API JMF tenta di risolvere questi problemi, mettendo a disposizione del programmatore una serie di chiamate ad "alto livello" per la gestione del codice nativo. Usando JMF, l'applicazione o applet non ha necessità di conoscere quando e se deve sfruttare particolari metodi nativi per svolgere una determinata azione. JMF 2.1.1 rende disponibili delle classi che permettono lo sviluppo di applicazioni per la cattura di dati multimediali, fornendo inoltre ai programmatori un controllo addizionale sull'elaborazione e la riproduzione dei dati stessi. JMF 2.1.1 è stato progettato per:

- Facilitare la programmazione
- Mettere a disposizione del programmatore un player JMF per la riproduzione dei dati multimediali.
- Semplificare notevolmente l'integrazione di sorgenti multimediali in applet o applicazioni fornendo tutta una serie di classi e metodi per la gestione temporale di stream di dati.

- Connettersi a host remoti e instaurare sessioni http, piuttosto che RTP/RTCP o RTSP (Real Time Streaming Protocol).
- Permettere lo sviluppo di applicazioni di audio e video conferenza in Java.
- Permettere a programmatori avanzati di implementare soluzioni personalizzate 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*)
- Mantenere la compatibilità con JMF 1.0

### 7.1.1 La ricezione dei dati sul Client: il Player

Il Player è la struttura che le API di JMF mettono a disposizione del programmatore per la riproduzione di dati multimediali. Come già accennato, le funzioni messe a disposizione dal Player consentono agli sviluppatori di software di non interessarsi direttamente delle chiamate al codice nativo, di impegnare le risorse necessarie per la riproduzione e di effettuarne un eventuale rilascio, quando queste non siano più necessarie.

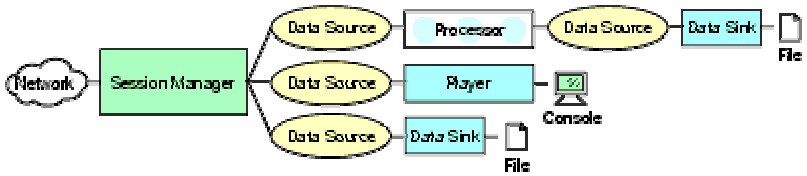


**Figura 7.1**: ricezione di flussi multimediali con il JMF.

Le chiamate a metodi e classi di "alto livello", rendono quindi trasparente al programmatore la connessione che si stabilisce tra Java Virtual Machine e routine specifiche di sistema (proprio secondo la filosofia generale di Java). In figura 7.1 viene rappresentato un tipico scenario di utilizzo del JMF, per la ricezione di dati dalla rete. Questa figura è stata tratta da una documentazione che purtroppo è piuttosto datata (vedi [JMFPG]): il SessionManager nell'attuale versione 2.1.1 è stato sostituito dall'RTPManager, comunque concettualmente non è cambiato molto. Per ogni flusso multimediale ricevuto viene impiegato un RTPManager che gestisce la sessione RTP. Ad alto livello dal lato del ricevente possiamo, attraverso l'uso di opportune API ricavare, all'arrivo dei dati, un DataSource che verrà utilizzato per l'inizializzazione del Player. Una volta inizializzato il Player potremo poi ricavare dal Player stesso un componente visuale (se presente), che utilizzeremo per la fruizione del materiale

multimediale; nel nostro caso, dal momento che realizzeremo uno streaming video, tale componente sarà, ovviamente, presente.

### 7.1.2 L'invio dei dati su Server e Proxy: il Processor

Per quanto riguarda invece l'invio dei dati viene utilizzato un Processor. Il Processor estende il Player ed in più di quest'ultimo offre metodi per la gestione e la trasformazione dei flussi multimediali e per l'ottenimento di un DataSource. Tale metodo è di fondamentale importanza per la successiva inizializzazione dell'RTPManager; infatti, all'atto della creazione di un SendStream, che rappresenta appunto l'astrazione del flusso dati che verrà trasmesso in rete, bisognerà passare all'RTPManager un DataSource dal quale attingere il contenuto multimediale che sarà, per l'appunto, ottenuto dal Processor. In figura 7.2 rappresentiamo la situazione che si ha per l'invio di dati in rete.
Questa figura, come la precedente, mette in evidenza diversi casi di possibile utilizzo delle API. Per quello che riguarda questo lavoro di tesi si fa riferimento al caso in cui il materiale multimediale (il video) già salvato all'interno di un file, viene poi spedito attraverso la rete e alla fine del percorso viene direttamente visualizzato sul terminale video.

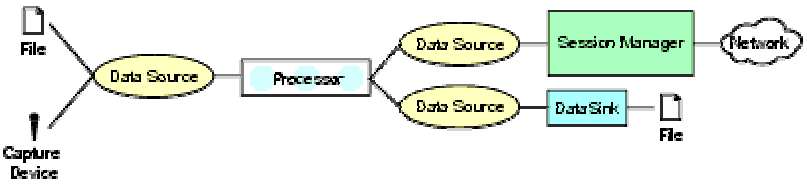


**Figura 7.2**: spedizione di flussi multimediali con il JMF.

### 7.1.3 Supporto al protocollo RTP/RTCP

L'ultima caratteristica che si vuole mettere in risalto del JMF è il fatto che offre, insieme al set di API utilizzato messo a disposizione per la riproduzione e il rendering di materiale multimediale, anche un set di API per stabilire sessioni RTP, offrendo al livello applicativo tutte le astrazioni necessarie a gestire tale protocollo in modo piuttosto semplice.

Tale caratteristica fa di queste API uno strumento ideale per lo sviluppo di applicativi di distribuzione dei flussi multimediali, ed inoltre, anche per lo sviluppo di applicazioni di tipo audio/video conferenza. La realizzazione dei server e dei client è poi piuttosto semplificata dall'utilizzo di tali API, una volta

capita la logica alla base del loro utilizzo. In figura 7.3 si riporta una figura che mostra l'architettura del JMF.

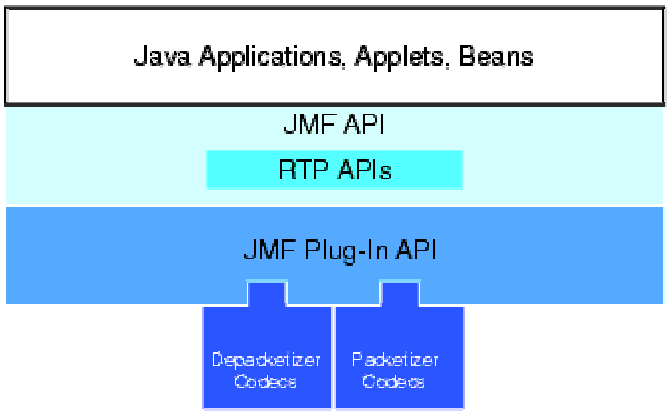


**Figura 7.3**: architettura del JMF.

Non ci dilungheremo oltre sulla trattazione di questa libreria. Per avere ulteriori informazioni è possibile consultare la suddetta documentazione, oppure fare riferimento alla home page [JMFH].

## *7.2 Protocollo RTP*

Come già accennato nel capitolo 1 nella maggior parte delle applicazioni tradizionali è importante che i dati giungano a destinazione integri e, pur di ottenere questa garanzia, è tollerabile che vi siano dei ritardi nella consegna. Nelle applicazioni che devono garantire un servizio in tempo reale, invece, è più facile compensare la perdita di una parte dei dati, rispetto ad un ritardo troppo elevato. Di conseguenza i protocolli che ben si adattano alla trasmissione di dati statici, non si adattano altrettanto bene al supporto di applicazioni multimediali. In questa sezione verrà brevemente descritto il *Real Time Protocol* (RTP) [RFC1889] che rappresenta lo standard proposto dall'*Internet Engineering Task Force* per la distribuzione di flussi multimediali su Internet.

Il protocollo RTP provvede alla distribuzione di servizi punto-punto per i dati che necessitano di trasferimento in tempo reale; cioè stream audio e video, ma sono state definite estensioni anche per altri tipi di dati. Questi servizi includono l'identificazione del formato dei dati trasportato (*payload type)*, la numerazione dei pacchetti (*sequence numbering*), l'assegnazione di un *timestamp*, e il monitoraggio della sessione. Le applicazioni in genere implementano RTP al di sopra di UDP, che fornisce le operazioni di *multiplexing* e *checksum*, anche se, per le sue caratteristiche, RTP può essere usato con altri protocolli di rete e di trasporto. Inoltre RTP supporta il

trasferimento dati verso molteplici destinazioni usando, una distribuzione di tipo multicast. È da notare che questo protocollo non prevede nessun meccanismo che assicuri una corretta trasmissione o garantisca la qualità del servizio, ma per quello che riguarda la gestione della QoS la facilita, dotando il livello applicativo di maggiore visibilità. RTP non si interessa, inoltre, del mancato arrivo a destinazione in ordine dei pacchetti né dell'affidabilità con cui i livelli di rete prevedono il riordino. La numerazione, permette al ricevente di ricostruire la corretta sequenza dei pacchetti inviati dal mittente; inoltre i *sequence number* possono essere usati per determinare la corretta posizione di un pacchetto all'interno di una sequenza senza necessariamente decodificarlo. Sebbene RTP sia stato implementato principalmente per le videoconferenze, esso viene comunemente impiegato anche nella memorizzazione di flussi continui, nelle simulazioni interattive distribuite, e nelle applicazioni di misurazione e controllo.

RTP consiste di due parti principali:

- il Real-Time Protocol (RTP), per trasportare dati che hanno vincoli di real-time
- l'Real-Time Control Protocol (RTCP), per monitorare la qualità del servizio e fornire informazioni sui partecipanti di una sessione in atto. Quest'ultimo aspetto di RTCP può essere sufficiente per applicazioni dove non esiste un esplicito controllo dei partecipanti, ma non è sufficiente per implementare meccanismi di management dei gruppi.

In una trasmissione RTP che preveda l'utilizzo contemporaneo di diversi media (ad esempio audio e video) essi sono trasmessi per mezzo di sessioni RTP separate e i pacchetti RTCP relativi usano due differenti coppie di porte UDP o indirizzi multicast, nel caso di comunicazione multicast. Le informazioni estratte dal protocollo RTCP verranno utilizzate per il monitoring della banda passante. Si rileva però che nell'implementazione del protocollo offerta dal JMF solo il 5% della banda totale viene utilizzata per la realizzazione del protocollo di controllo, come fissato nella prima RFC emessa da IETF. Ne segue che il feedback ricavato attraverso questo strumento, che viene comunque utilizzato per la sua portabilità, essendo un protocollo implementato al livello applicativo, non rispecchia prontamente veloci variazioni nelle condizioni della rete; ci si ripropone in futuro di studiare possibili alternative per la realizzazione di un più sofisticato strumento di monitoraggio della banda trasmissiva, in modo che i feedback ricavati rispecchino più fedelmente le condizioni della rete.

Riteniamo con ciò conclusa la breve panoramica sul protocollo RTP/RTCP utilizzato per la realizzazione dell'applicazione per lo streaming video.

## *7.3 Progettazione delle singole entità*

Per la progettazione di queste tre entità verranno seguite le linee guida proposte in 5.2.2.3, che vengono qui concretizzate nelle architetture di Client, Proxy e Server.

### 7.3.1 Client

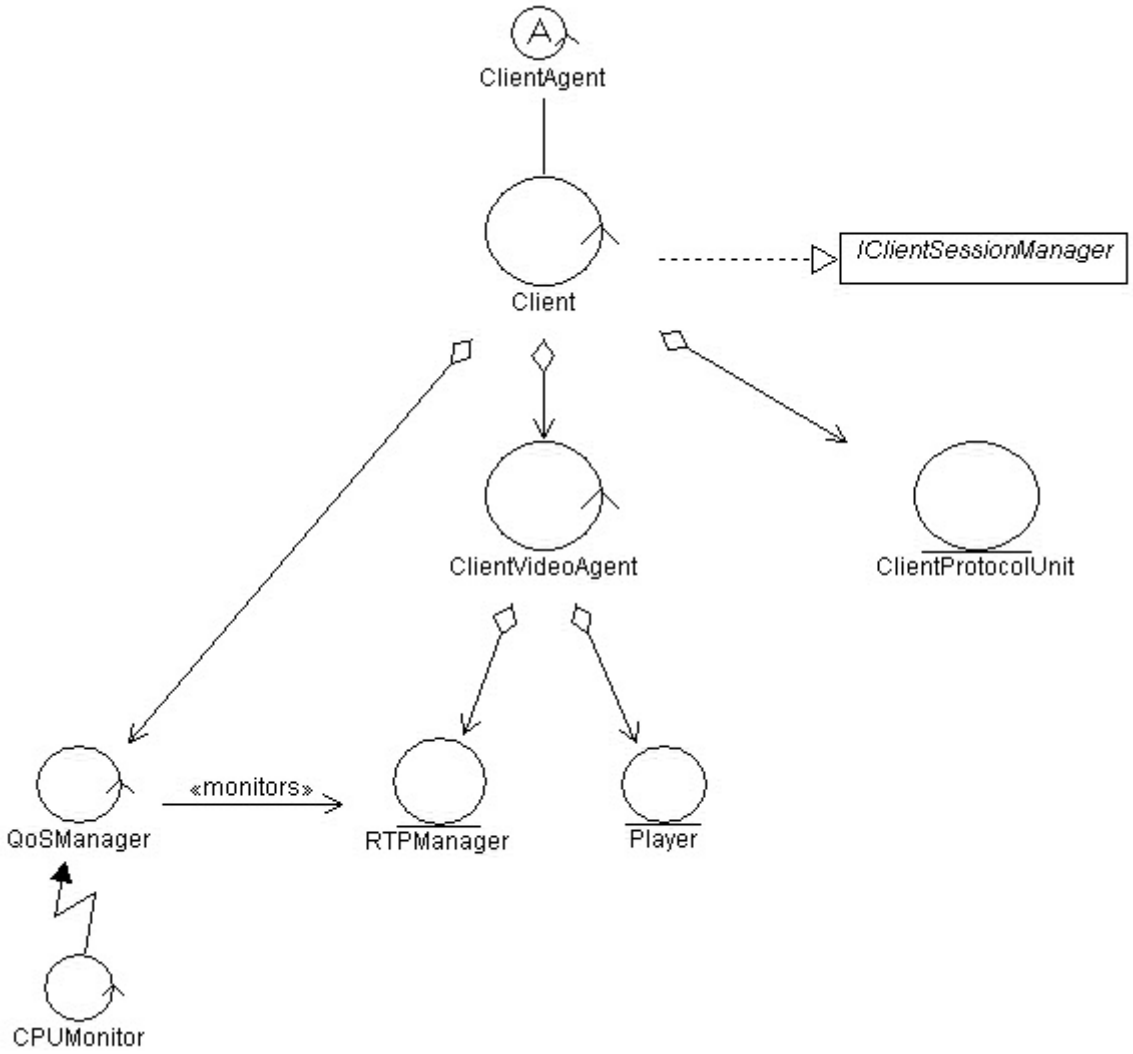


**Figura 7.4**: architettura del client.

In figura 7.4 viene riportata l'architettura del Client. Si nota che ciascuna delle entità proposte, sebbene nella realizzazione concreta di questo prototipo gestisca un solo flusso multimediale, può facilmente essere estesa a gestire più flussi. Come già anticipato in fase di progetto del middleware, tutte le entità che vogliano usufruire dei servizi offerti dal middleware dovranno implementare le interfacce definite in 6.2.2. L'architettura di questo Client è organizzata come segue: il Client svolge il ruolo di coordinatore delle varie entità ed utilizza la

ClientProtocolUnit per inviare in avanti comandi alle altre entità. I comandi sono costituiti dal classico set di comandi per il controllo, (ad esempio di un videoregistratore), tra cui la richiesta di partenza dello streaming, la richiesta di stop e lo spegnimento che produce lo scioglimento del percorso inizializzato e la liberazione delle varie risorse. Il ClientVideoAgent, è invece l'entità preposta alla gestione del flusso multimediale, incapsula infatti il gestore del protocollo RTP e il Player che realizza il rendering del video, e presso il quale ci si potrà procurare l'oggetto visuale presentato all'utente. Si è deciso, anche se il diagramma proposto non mette direttamente in luce questo aspetto, di incapsulare il componente visuale e la sua gestione all'interno del Client. L'altra scelta possibile sarebbe stata dare tali responsabilità al ClientAgent. Si è però optato per la prima soluzione proposta perché in questo modo, nel caso in cui si supporti il movimento di un utente nomadico sarà possibile, anche se il ClientAgent non è ancora giunto nel target place, iniziare da subito il rendering del video.

Particolare importanza riveste la fase di inizializzazione nella quale il Client inizializza la sessione RTP di tipo unicast comunicando con la prossima entità sul percorso. Il Client è stato poi definito come thread per evitare di bloccare l'interfaccia utente, in particolare durante la fase di inizializzazione che potrebbe richiedere tempi piuttosto lunghi.

L'ultimo oggetto che consideriamo è il QoSManager che, utilizzando le API messe a disposizione dal JMF, monitora la sessione RTP e, ricevendo anche notifiche sulla condizione della CPU da parte del CPUMonitor (come stabilito al capitolo precedente), richiede eventuali riconfigurazioni al ClientAgent. In particolare come parametri per ciò che riguarda il monitoraggio della banda vengono considerati il jitter e la percentuale dei pacchetti persi.

Le connessioni aperte dal Client sono tre, due come visto sopra sono connessioni richieste dal protocollo RTP, che tipicamente saranno connessioni di tipo UDP, mentre la terza è una connessione TCP per l'invio dei dati. Si utilizza tale protocollo perché, per quello che riguarda l'invio dei dati, non ci sono particolari urgenze nella consegna; non si ritiene quindi necessaria la realizzazione di alcun protocollo ad-hoc, che potrebbe essere realizzato al di sopra di UDP. Si vorrebbe invece avere una ragionevole certezza che i comandi spediti arrivino al destinatario, utilizzeremo perciò TCP, che ci garantisce una semantica di tipo at-most-once.

### 7.3.2 Server

Prima del Proxy verrà presentato il server, dal momento che il Proxy è, come ci si può ragionevolmente aspettare, un'entità che unisce funzionalità proprie di Client e Server. In figura 7.5 riportiamo il progetto del Server.

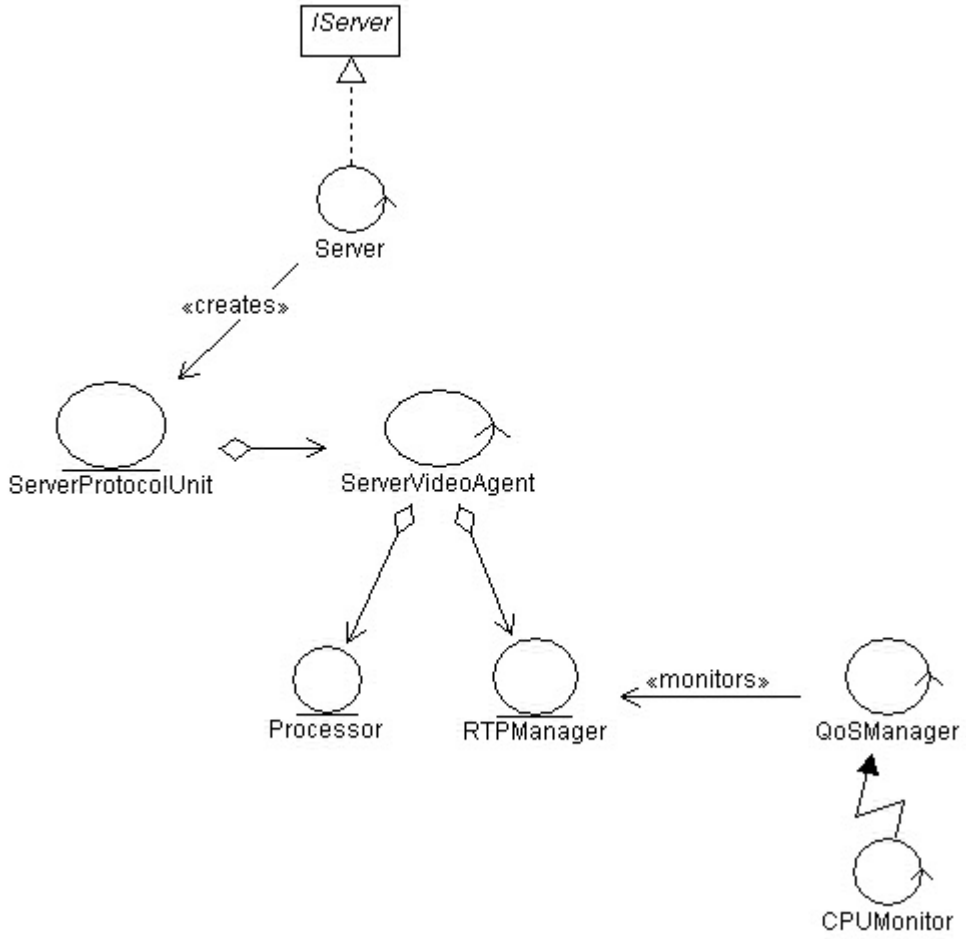


**Figura 7.5**: architettura del server.

Il Server è assai simile al Client analizzato nella sezione precedente, costituendone, naturalmente, il pari nella comunicazione. Come per il Client si è strutturato il lato servitore in modo da dividere la gestione della sessione, cioè l'invio dei comandi dalla gestione dei flussi, che viene incapsulata, in questo caso, dal ServerVideoAgent. Come sopra abbiamo poi il QoSMonitor che monitora la situazione. La differenza maggiore fra Client e Server è però che, mentre il Client aveva ciclo di vita paragonabile a quello del ClientAgent (mobilità a parte), per ciò che riguarda il Server, una volta inizializzato, rimane attivo come demone. Il Server è multithreaded, infatti, ad ogni nuova richiesta ricevuta l'entità Server, crea e fa partire un nuovo thread che gestirà da quel

momento in poi tutta la sessione col cliente, cioè la ServerProtocolUnit evidenziata in figura.

### 7.3.3 Proxy

L'ultima entità che ci rimane da analizzare è il Proxy. Tale entità riceverà dalle entità a valle i comandi, che dovrà inoltrare alle entità a monte, mentre dalle entità a monte i flussi, che dovrà inoltrare a quelle a valle. In figura 7.6 viene riportata l'architettura di questa entità.

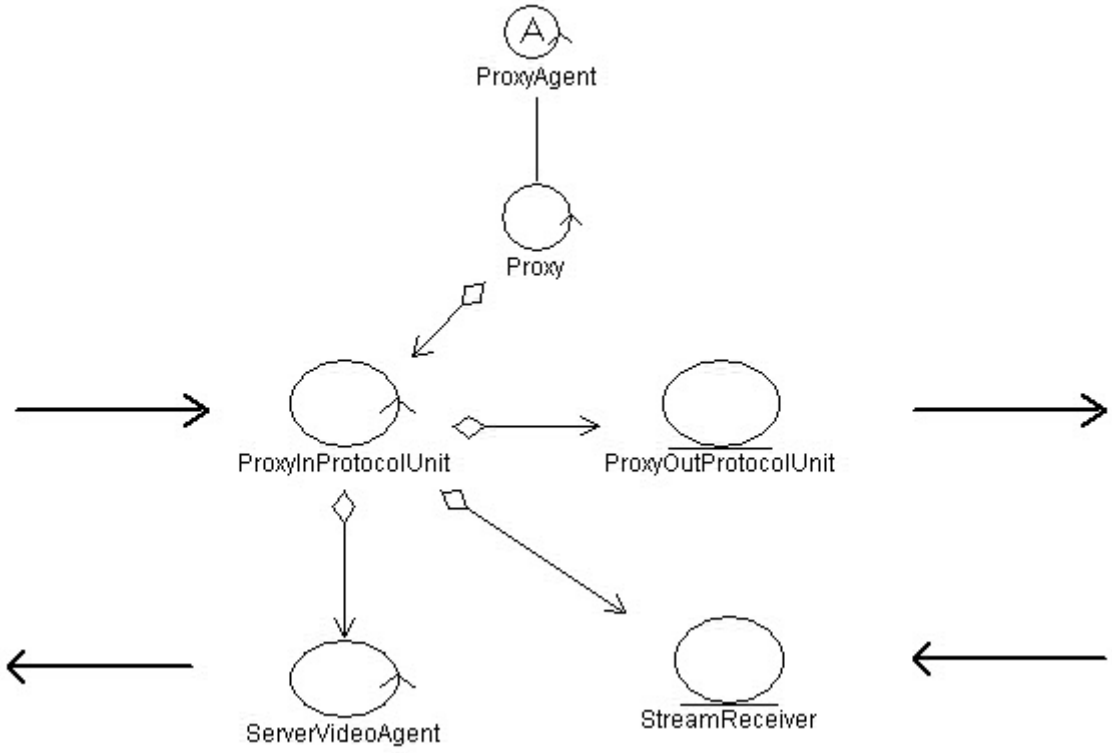


**Figura 7.6**: architettura del proxy.

Le Protocol Unit di ingresso e di uscita sono assai simili a quelle di Server e Client. La differenza più rilevante è che si è scelto di rivestire, un po' come accade anche per l'entità Server, la ProxyInProtocolUnit del ruolo di coordinatore per le altre entità costituenti il Proxy. L'altra scelta possibile sarebbe stata incapsulare il controllo nel Proxy. Abbiamo però optato per la prima scelta perché, durante la gestione dell'hand-off per il supporto alla mobilità, esisterà un transitorio durante il quale verrà istanziata un'altra ProxyInProtocolUnit, in modo da inizializzare una nuova sessione RTP unicast, con l'entità che viene preventivamente istanziata nel Target Place in attesa del movimento agenti. Possiamo perciò dire che, durante tale processo, il Proxy agisce da vero e proprio

Server, potendo accettare più di una connessione. Da queste considerazioni è nata la scelta architetturale sopra esposta.

## *7.4 Test del servizio di inizializzazione del sistema*

Quest'ultimo paragrafo viene dedicato alla descrizione di alcuni primi test effettuati per testare il sistema. In particolare tali test sono rivolti a verificare i tempi di inizializzazione del sistema. Con inizializzazione, per quanto riguarda l'applicazione di video streaming realizzata, si intende la fase che comincia quando l'utente richiede un certo titolo e termina quando tutte le entità sono state inizializzate dal place del cliente fino al place ove si trova la presentazione prescelta. La terminazione della fase di inizializzazione coincide con l'inizio della visione, da parte dell'utilizzatore dell'applicazione, del video.

### 7.4.1 Configurazione dei test

Due computer sono stati coinvolti nell'esecuzione dei test. Essi erano così equipaggiati:

**PC1** portatile Asus

-processore Intel Celeron 1200 MHz, 392 MB RAM

-sistema operativo Windows 2000 Professional

**PC2** fisso

-processore Pentium III 500 MHz, 128 MB RAM

-sistema operativo Windows 2000 Professional

**Rete** i due computer sono stati poi collegati da una rete LAN Ethernet da 10 Mbps

**Codice da scaricare** il codice da scaricare è il codice sviluppato per la realizzazione del Client, del Proxy e del Server. Naturalmente si ipotizza che le librerie multimediali (la JMF), siano già presenti sui vari place coinvolti. Per quello che riguarda il codice da scaricare le dimensioni sono le seguenti:

- *Client*: 47KB;
- *Proxy*: 58KB;
- *Server*: 41KB.

I test presentati si riferiscono al seguente caso. Si considerano due nodi cioè i computer appena descritti, e si considerano i seguenti diversi casi di inizializzazione.

1. Il caso in cui nei nodi coinvolti nell'inizializzazione non sia presente ancora alcun software e anche per il server multimediale il codice debba essere scaricato ed il server inizializzato.
2. Un secondo caso in cui il codice è già presente su tutti i place coinvolti nel test, ma il server deve ancora essere attivato.
3. Un terzo caso in cui il software è già presente sui diversi place ed il server multimediale è già stato avviato.

I test sono stati svolti considerando due, tre e quattro place inizializzando il sistema in modo da avere sempre sul place iniziale il client, sul place finale il server e su tutti i place intermedi i proxy.

### 7.4.2 Risultati raccolti

In fig. 7.7 riportiamo i risultati per le prove svolte (i tempi sono riportati in msec). Come ci si aspetta, all'aumentare dei place, aumenta il tempo richiesto di inizializzazione. Inoltre, passando dal caso 1 al caso 3 si assiste, ugualmente, all'attesa diminuzione dei tempi di inizializzazione.

| | | Prova 1 | Prova 2 | Prova 3 | Media |
|---|---|---|---|---|---|
| NO SW NO SRV | 2 nodi | 14.431 | 13.830 | 14.020 | 14.094 |
| | 3 nodi | 15.334 | 15.454 | 14.872 | 15.220 |
| | 4 nodi | 15.622 | 16.013 | 16.083 | 15.906 |
| SI SW NO SRV | 2 nodi | 12.588 | 12.748 | 12.678 | 12.671 |
| | 3 nodi | 14.662 | 15.222 | 14.861 | 14.915 |
| | 4 nodi | 15.082 | 15.202 | 15.112 | 15.132 |
| SI SW SI SRV | 2 nodi | 2.984 | 2.584 | 2.474 | 2.681 |
| | 3 nodi | 5.308 | 4.096 | 3.816 | 4.407 |
| | 4 nodi | 5.067 | 4.497 | 4.857 | 4.807 |

**Figura 7.7**: test sui tempi di inizializzazione.

Vogliamo poi soffermarci su due punti:

1. il tempo richiesto per lo scaricamento del codice, date anche le piccole dimensioni del codice da scaricare non incide particolarmente sui tempi di inizializzazione del sistema;
2. una volta che il sistema sia giunto a regime, cioè i vari server multimediali siano stati attivati, i tempi di inizializzazione vengono praticamente quasi dimezzati. L'evidenza sperimentale quindi conforta la scelta fatta in fase progettuale di realizzare i server come demoni che possano, una volta attivati su richiesta di una particolare presentazione multimediale, essere poi utilizzati per servire le altre richieste che pervengono allo stesso place, senza dover pagare i lunghi tempi di attesa che, come evidente dai dati sperimentali sono dovuti principalmente all'inizializzazione del server multimediale stesso.

Futuri test saranno svolti per verificare se questi primi confortanti risultati siano estendibili ad un sistema distribuito costituito realmente da più nodi, inoltre lavoro futuro riguarderà anche il testing accurato delle altre parti del sistema.

# Conclusioni

Nel corso dello sviluppo del progetto di tesi sono state approfondite diverse tematiche che hanno spaziato dalla gestione delle risorse allo sviluppo di un supporto middleware per la mobilità, basato su agenti mobili.

Un primo dato rilevato è, come d'altronde ci si aspettava, la carenza di modelli che assistano il processo di sviluppo di tali tipi di piattaforme per il supporto della mobilità e riteniamo che molto debba ancora essere fatto in questo ambito.

La soluzione proposta ha cercato di coniugare il meglio di un consolidato modello, quale il modello cliente/servitore, con l'utilizzo del paradigma ad agenti mobili per la realizzazione degli aspetti più dinamici del sistema. In particolare tale tecnologia viene considerata valida per quello che riguarda l'inizializzazione di applicazioni distribuite ed in generale per la gestione della riconfigurazione delle entità sulla rete fissa, in modo da assistere i movimenti del cliente e del terminale. D'altro canto si ritiene che la progettazione delle diverse entità costituenti l'applicazione distribuita dovrebbe comunque basarsi su modelli più statici. Ad esempio poco senso avrebbe, per lo scenario considerato, realizzare un server come agente mobile dal momento che, una volta istanziato, dovrà per sempre rimanere fisso di un certo nodo; inoltre realizzandolo come agente mobile si esclude a priori la possibilità di riutilizzarlo come componente stand-alone che possa essere eseguito al di sopra di una piattaforma che non supporti necessariamente gli agenti mobili.

Per quanto riguarda la gestione delle risorse, si rileva in particolare come meccanismi per la prenotazione delle stesse possano portare notevoli vantaggi e aprire nuovi orizzonti per la realizzazione di servizi di qualità che si discostino dall'attuale panorama di servizi best-effort. Naturalmente resta aperto il problema dell'accounting per tali servizi, cioè stabilire una modalità per il pagamento degli stessi, ed insieme ad esso il problema della sicurezza. Sempre riguardo alla gestione delle risorse si è poi optato per la realizzazione gli adattamenti del sistema come veri e propri "salti" di qualità, evitando così continui microadattamenti, che appesantiscano inutilmente il sistema stesso.

Sono stati già eseguiti i primi test volti a stabilire i tempi di inizializzazione del sistema, e si ha in programma di procedere ben presto ad un'ulteriore fase testing per stabilire anche i tempi di riconfigurazione dello stesso. I risultati

ottenuti 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 run-time.

Per quanto riguarda la gestione dei flussi multimediali ulteriori approfondimenti richiede lo sviluppo della parte per la gestione dei flussi sulle reti wireless. In particolare l'attuale soluzione prevede la presenza del middlware SOMA sul terminale mobile; tale assunzione è troppo forte per molti degli attuali device portabili. Oggetto di futuri approfondimenti sarà l'estensione delle potenzialità del ProxyAgent, in modo che supporti lo streaming anche verso piattaforme proprietarie, e la realizzazione dei relativi Client, che saranno necessariamente client realizzati ad-hoc. Inoltre sarebbe opportuno testare la validità del middleware proposto per applicazioni multimediali che gestiscano sessioni più complesse formate da diversi flussi multimediali.

Per quanto riguarda infine il supporto alla mobilità riteniamo utile approfondire ulteriormente lo studio di schemi di progettazione "misti" come quello proposto nel nostro lavoro di tesi con lo scopo ultimo di estendere il middleware a supportare altre tipologie di applicazioni, tenendo presenti scenari di Ubiquitous Computing.

# Tabella degli acronimi

| *Acronimo* | *Descrizione* |
|---|---|
| ACE | Adaptive Communication Environment |
| ACK | Acknowledge |
| AP | Access Point |
| API | Application Programming Interface |
| ATM | Asynchronous Transfer Mode |
| BSS | Basic Service Set |
| CE | Computational Environment |
| COD | Code On Demand |
| CORBA | Common Object Request Broker Architecture |
| COS | Core Operationg System |
| CPU | Central Processor Unit |
| CRC | Cyclic Redundant Check |
| CSMA | Carrier Sense Multiple Access |
| CTS | Clear To Send |
| DECT | Digital Enhanced Cordless Telecommunications |
| Diff Serv | Differentiated services |
| DIFS | Distibuted Inter Frame Space |
| DS | Distribution System |
| DSSS | Direct Sequence Spread Spectrum |
| ESS | Extended Service Set |
| EU | Execution Unit |
| FHSS | Frequency Hopping Spread Spectrum |
| GUI | Graphical User Interface |
| IBSS | Graphical User Interface |
| IDL | Interface Definition Language |
| IEEE | Institute of Electrical and Electronics Engineers |
| Int Serv | Integrated services |
| ISO | International Organization for Standardization |
| JVM | Java Virtual Machine |
| LAN | Local Area Network |
| MAC | Medium Access Control |
| MPEG | Moving Picture Experts Group |
| MSDU | MAC Service Data Unit |
| MW | MiddleWare |
| NOS | Network Operating System |
| OMG | Object Management Group |
| ORB | Object Request Broker |
| OSI | Open System Interconnect |
| PDA | Personal Digital Assistant |
| QoS | Quality of Service |
| RMI | Remote Method Invocation |
| RM-ODP | Reference Model for Open Distributed Processing |

| | |
|---|---|
| RPC | Remote Procedure Call |
| RSVP | ReSerVation Protocol |
| RT | RealTime |
| RTP | RealTime Protocol |
| RTS | Request To Send |
| RTSP | RealTime Streaming Protocol |
| SO | Sistema Operativo |
| SOMA | Secure and Open Mobile Agent |
| SRT | Soft-Realtime |
| TCP | Transport Control Protocol |
| IP | Internet Protocol |
| UDP | User Datagram Protocol |
| URL | Uniform Resource Locator |
| VoD | Video on Demand |
| WEP | Wired Equivalence Privacy |
| WLAN | Wireless LAN |
| XML | Extensible Markup Language |

# Bibliografia

[OSI] Basic Reference Model of Open Distributed Processing, Part 1: Overview and guide to use. ISO/IEC JTC1/SC212/WG7 CD 10746-1, International Standard Organization, 1992.
[GEK01] Geihs K., Middleware Challanges Ahead, IEEE Computer, June 2001 (Vol. 34, No. 6).
[CAA98] Aurrecoechea C., Campbell A. T., Hauw L., A Survey of QoS Architectures, Multimedia Systems (1998) 6: 138-151.
[BAF01] Baschieri F., Tesi di Laurea, 2001.
[SHP02] Shenoy P., Hasan S., Kulkarni P., Ramamritham K., Middleware versus Native OS Support: Architectural Considerations for Supporting Multimedia Applications, Proceedings of Real-Time Applications and Systems (RTAS), 2002
[RFC2210] Resource ReSerVation Protocol (RSVP). RFC2210, si veda http://www.ietf.org/rfc/rfc2210.txt
[RFC2205] Resource ReSerVation Protocol (RSVP). RFC2205, si veda http://www.ietf.org/rfc/rfc2205.txt.
[RFC2475] An Architecture for Differentiated Services. RFC2475, si veda http://www.ietf.org/rfc/rfc2475.txt
[RFC1889] RTP: A Transport Protocol for Real-Time Applications. RFC1889, si veda http://www.ietf.org/rfc/rfc1889.txt
[RFC2326] Real Time Streaming Protocol (RTSP). RFC2326, si veda http://www.ietf.org/rfc/rfc2326.txt.
[KOF00] Kon F., Campbell R.H., Mickunas M.D. Nahrstedt K., Ballesteros F.J. (2000), “2K: A Distributed Operating System for Dynamic Heterogeneous Environments”,9th IEEE International Symposium on High Performance Distributed Computing. Pittsburgh. August 1-4, 2000.
[BLG01] Blair G.S., Coulson G., Andersen A., Blair L., Clarke M., Costa F., Duran-Limon H., Fitzpatrick T., Johnston L., Moreira R., Parlavantzas N. and Saikoski K. (2001), "The Design and Implementation of Open ORB 2", IEEE Distributed Systems Online, Vol. 2, No. 6, 2001.
[NAK01] Nahrstedt K., Wichadakul D., Gu X., Xu D. (2001), 2Kq+: An Integrated Approach of QoS Compilation and Reconfigurable, Component-Based Run-Time Middleware for Unified QoS Management Framework, Proc. of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001) , Heidelberg, Germany.
[SCD01] Karr D.A., Rodrigues C., Krishnamurthy Y., Pyarali I., Loyall J.P., Schantz R.E., Schmidt D.C. (2001), Application of the QuO Quality-of-Service

Framework to a Distributed Video Application, Proceedings of the 3rd International Symposium on Distributed Objects and Applications , Rome, Italy.
[BLG99] Blair G.S., Coulson G., Plagemann T. Et alii (1999), Next Generation Middleware: Requirements, Architecture, and Prototypes, Distributed Computing Systems Proceedings. 7th IEEE Workshop on Future Trends of.
[BLG01] Blair G.S., Coulson G., Andersen A., Blair L., Clarke M., Costa F., Duran-Limon H., Fitzpatrick T., Johnston L., Moreira R., Parlavantzas N. and Saikoski K. (2001), "The Design and Implementation of Open ORB 2", IEEE Distributed Systems Online, Vol. 2, No. 6, 2001.
[BLG97] Blair, G.S., Stefani J.B. (1997), "Open Distributed Processing and Multimedia", ISBN 0201177943, Addison-Wesley.
[KAD01] Karr D.A., Rodrigues C., Krishnamurthy Y., Pyarali I., Loyall J.P., Schantz R.E., Schmidt D.C. (2001), Application of the QuO Quality-of-Service Framework to a Distributed Video Application, Proceedings of the 3rd International Symposium on Distributed Objects and Applications, Rome, Italy.
[BBN] BBN Technologies, http://www.bbn.com/
[LOJ98] Loyall J.P., Bakken D.E., Schantz R.E., Zinky J.A., Karr D.A., Vanegas R., Anderson K.R. (1998), QoS Aspect Languages and Their Runtime Integration, Lecture Notes in Computer Science, Vol. 1511, Springer-Verlag. Proceedings of the Fourth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR98), 28-30 May 1998, Pittsburgh, Pennsylvania.
[BLG00] G.S. Blair et al. (2000), "Supporting Dynamic QoS Management Functions in a Reflective Middleware Platform", IEEE Proceedings Software, vol. 147, no. 1, Feb. 2000, pp. 13–21.
[TAK01] Takashio K., Soeda G., Tokuda H. (2001), A mobile agent framework for follow-me applications in ubiquitous computing environment, Distributed Computing Systems Workshop, 2001 International Conference on.
[FUA98] Fuggetta A., Picco G.P., Vigna G., “Understanding Code Mobility”, IEEE Transactions On Software Engineering, Vol.24, No.5, May 1998
[FUA98] Fuggetta A., Picco G.P., Vigna G., “Understanding Code Mobility”, IEEE Transactions On Software Engineering, Vol.24, No.5, May 1998
[TRA98] Tripathi A.R. and Karnik N.M., “Design Issues in Mobile-Agent Programming Systems”, IEEE Concurrency, July-September 1998

[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

[BEM02] Bevilacqua M., “Progetto di un framework location aware”, tesi di Laurea 2002

[80211] IEEE 802.11 Wireless LAN Working Group, http://www.grouper.ieee.org/groups/802/11

[MAC99] LAN MAN Standards di IEEE Computer Society, “Wireless LAN medium access control (MAC) and phisical layer (PHY) specification, IEEE Standards 802.11, 1999 edition”.
[WIRN] Wireless network, http://www.qsl.net/n9zia/wireless.
[FEA02] Ferretti A., “Servizi dipendenti dalla locazione in ambienti wireless”, tesi di Laurea 2002
[MedPat] Mediator Pattern,si veda http://patterndigest.com/patterns/Mediator.html.
[FacPat] Facade Pattern, si veda http://patterndigest.com/patterns/Facade.html.
[DaoPat] DAO Pattern, si veda http://www.cs.unb.ca/profs/wdu/cs4025/notes/note18a.htm.
[VisPat] Pattern Visitor, si veda http://patterndigest.com/patterns/Visitor.html.
[ARL98] Armani Luca, “Metodologie per il monitoraggio di risorse in Java”, tesi di Laurea, 1999
[JMFPG] Java Media Framework, Programmers Guide: http://java.sun.com/products/java-media/jmf/2.1.1/specdownload.html
[JMFH] Java Media Framework home page: http://java.sun.com/products/java-media/jmf/