Università degli studi di Bologna
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica
Reti di Calcolatori

GESTIONE DI PROFILI UTENTE PER SISTEMI DI CACHE

Relatore: Chiar.mo Prof. Ing. Antonio Corradi
Correlatori: Dott. Ing. Paolo Bellavista
Dott. Ing. Luca Foschini

Autore: Antonio Vicciantuoni

Anno Accademico 2003 - 2004

Parole chiave

Internet
Quality of Service
Sistemi Multimediali
Profilazione
Caching

Indice

Introduzione

Internet ha ormai rivoluzionato il modo di concepire e condividere le informazioni; questo si basa sull' astrazione di rete e di connessione tra le più svariate macchine.

Oggigiorno i sistemi multimediali diventano sempre più eterogenei e complessi. Grazie allo stupefacente progresso della tecnologia, i moderni sistemi possono includere una sempre più vasta gamma di dispositivi utente i quali risultano molto diversi, dal punto di vista delle capacità di presentazione e di accesso alle risorse; in un tale scenario un dispositivo, atto all'accesso ad una risorsa, di genere informatico, può essere una workstation, un cellulare WAP, un palmare, un portatile o altro.

Negli ultimi anni, inoltre, hanno preso piede varie metodologie di accesso alla rete e di interconnessione, di tipo wireless che si differenziano per banda, copertura, accesso.

Il metodo di accesso alla rete può variare quindi sensibilmente, possiamo infatti avere connessioni wireless, connessioni classiche di tipo dial-p o anche connessioni fisse attraverso una LAN.

In aggiunta a tutto questo l'utente, spesso anche all'interno dello stesso Sistema Operativo, ha a disposizione diverse possibili applicazioni con cui può accedere al medesimo servizio tutte con alcune piccole differenze di implementazione e di comportamento.

Dal momento che non tutti i "dispositivi" condividono le medesime caratteristiche ed utilizzano il medesimo formato, per lo scambio di informazioni, diventa sempre più importante la possibilità di adattare il contenuto in base alle capacità dei "dispositivi" dell'utente e di mettere quindi a disposizione dell'utente un particolare dato in una forma comprensibile per il dispositivo con cui viene fruito.

La distribuzione del servizio oltre a tenere presente le capacità dei dispositivi utente deve spesso far fronte anche alle "condizioni di ambiente", in primis la banda passante nel caso di risorse a cui si accede attraverso la rete.

Oltre all'adattamento svolto per rendere l' informazione accessibile ad dispositivo vi sono altre funzioni che può avere l'adattamento, vale a dire: adattamento per migliorare l'utilizzo delle risorse, adattamento per venire incontro a particolari preferenze imposte dall'utente, date da quest' ultimo per un suo gusto o per un suo particolare problema.

In questo scenario noi cercheremo di costruire un sistema di descrizione di profili utente adatto a un sistema di distribuzione di flussi multimediali il più possibile generico ed aperto.

Questo dovrà integrarsi nelle attuali architetture di rete delle quali internet è l'incarnazione. Tale sistema dovrà essere inoltre il più possibile indipendente dall'architettura in cui verrà integrato, in modo che possa essere riutilizzato anche in altri contesti, tenendo tuttavia presenti i requisiti in termini di Quality of Service dettati dalle attuali applicazioni multimediali. Altra caratteristica che si prefigge questo lavoro di tesi è l'apertura del sistema, in termini di standard adottati e paradigmi di comunicazione. Il processo di profilazione dovrà essere il più possibile modulare in modo da poterne gestire facilmente le politiche. Infine per quello che riguarda l'efficienza del sistema stesso, in esso dovrà essere facilmente integrabile il concetto di caching.

Nel primo capitolo viene data una breve introduzione al problema della profilazione e poi si passa alla definizione della terminologia adottata e la descrizione delle possibili problematiche.

Nel secondo capitolo presentiamo le attuali soluzioni adottate per la profilazione con particolare accento al campo delle applicazioni multimediali.

Nel terzo capitolo diamo una breve presentazione degli ambienti operativi in cui il presente lavoro si deve integrare.

Nel quarto capitolo diamo la specifica dei requisiti dell'applicazione sviluppata analizzando la struttura generale del sistema.

Nel quinto capitolo diamo una dettagliata descrizione dei componenti necessari allo sviluppo del sistema sviluppato.

Nel sesto capitolo presentiamo brevemente le tecnologie adottate per la successiva fase di sviluppo.

Nel settimo capitolo presentiamo le decisioni implementative ed i risultati di un test iniziale svolto sul sistema sviluppato.

1 Profiling definizioni e problematiche

Questo primo capitolo si occupa di dare una breve introduzione ai concetti che stanno alla base, della gestione dei profili utente. Prima presenteremo uno scenario tipico in cui si rende necessaria la profilazione utente, facendo anche diverse anticipazioni; poi vedremo in rassegna le definizioni e i termini che utilizzeremo all'interno di questo lavoro di tesi.

Poi si passa ad una prima introduzione sui concetti,le problematiche, e le possibili soluzioni al problema della profilazione utente.

1.1 Introduzione alla profilazione utente

In questo paragrafo presenteremo alcuni dei possibili meccanismi per effettuare la profilazione utente dandone una antipazione. Per proflazione si intende il processo di adattamento di una generica risorsa, fruibile in un qualche ambiente, nei confronti di un certo dispositivo. Tale processo necessita la descrizione, in termini di proprietà del dispositivo suddetto; oltre a questo la profilazione può includere la possibilità, per l'utente, di definire delle preferenze sui formati e/o la qualità del servizio desiderato. Contestualmente presenteremo le metodologie adottate e le problematiche associate, con particolare riferimento al mondo del *web*.

1.1.1 Profilazione nel WEB

Il mondo del *web* è stato uno dei primi a porre il problema della profilazione nel senso che i diversi *browser* inizialmente riconoscevano diverse *estensioni HTML*, e per ottenere un medesimo effetto, in una *pagina web* era necessario un adattamento per il *browser* utilizzato dall'utente. . Per questo motivo risulta opportuno vedere in che modo, nel *web*, si è cercato di introdurre la profilazione, e vedremo, poi, come tali metodologie sono tuttora alla base della profilazione utente.

Selezione del contenuto delegata all'utente

Questa tecnica consiste nel preparare la medesima *pagina web* in diverse versioni, adatte ai diversi browser, e una pagina iniziale aggiuntiva di scelta, all'utente viene proposta la pagina iniziale dalla quale sarà lui a scegliere quella più adatta al suo browser. Questa soluzione mostra notevoli inconvenienti:

1. Il processo di preparaione delle pagine risulta assai complesso infatti l'autore delle pagine web deve svilupparle in diverse versioni

2. Problemi di inconsistenza dei dati; questo è la conseguenza del punto precedente, infatti se viene aggiornata la pagina in una versione e non in un'altra possiamo avere descrizioni non congruenti.
3. Introduzione di parti esplicite che non rappresentano il contenuto, ma che sono legate alle modalità di fruizione dello stesso. In un sistema in cui vengono distribuite informazioni sarebbe auspicabile non introdurre, nelle informazioni stesse, dati riguardanti tutt'altra cosa, in quanto da un punto di vista, puramente logico, questo significa, in un certo senso, "sporcarle" e quindi comprometterne il contenuto.
4. Problemi per l'utilizzatore che per poter usufruire del servizio deve scegliere la pagina iniziale, è quindi costretto ad acquisire conoscenze tecniche che non gli competono e quindi potrebbe essere messo in difficoltà nell'uso di tale risorsa.

Selezione del contenuto comandata dal protocollo

Per "selezione del contenuto comandata dal protocollo", in questo caso, si intende il fatto che la scelta della pagina da reperire non deve essere più svolta dall'utente ma viene automaticamente fatta ad opera del protocollo usato per il reperimento della stessa; ovviamente tale scelta viene effettuata in base ad alcuni parametri che vengono usati nel protocollo in modo automatico.

Permangono i problemi relativi alla pubblicazione però in questo modo si evitano i problemi relativi all'utente e alla presentazione; questa tecnica consiste nell'estendere il protocollo usato per lo scambio delle pagine, o utilizzare alcune estensioni dello stesso, al fine di individuare le caratteristiche del sistema cliente intendendo con questo non solo il browser ma anche altre caratteristiche come il sistema operativo in uso, e altro.

Nel mondo del web il protocollo per lo scambio delle pagine ipertestuali è il protocollo *HTTP* che è stato sviluppato appositamente per tale scopo; senza entrare troppo nel dettaglio diamo ora una breve presentazione del protocollo *HTTP* per mostrare come questo venga usato per convogliare informazioni relative all'utente. Tale protocollo prevede, da parte del *client HTTP* (il *browser web*) l'invio di una richiesta relativa alla pagina o a un file da scaricare, e la risposta da parte del *server HTTP* con l'invio di tale risorsa; sia la richiesta che la risposta sono composte da due parti, una parte detta intestazione (*header* della richiesta/risposta) ed una parte detta corpo (*body* della richiesta/risposta), che in genere è assente nella richiesta. Quando, col *browser*, puntiamo un sito web, quest'ultimo invia una richiesta verso il *server http*, identificato da tale indirizzo, che contiene solo l'intestazione

nella quale c'è il nome della risorsa richiesta, la pagina HTML in genere, e il *server http* ci invia una risposta con una intestazione quasi identica a quella della richiesta e con dentro il corpo della risposta la pagina voluta. Ora, nell'intestazione della richiesta, il *browser*, in genere, inserisce informazioni relative a se stesso, alla lingua del sistema in uso ed altro, questi campi sono nella forma: **<campo>: <valore>** dove i vari valori di **<campo>** sono universalmente accettati e riconosciuti, o meno, in base alla versione del protocollo adottato.

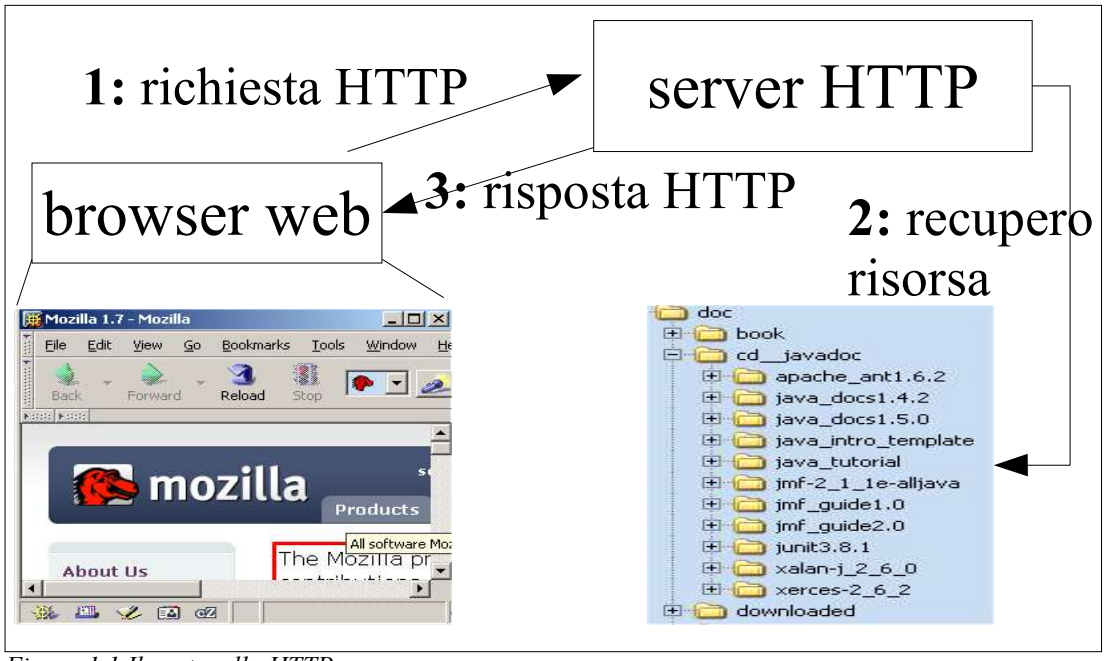

Figura 1.1 Il protocollo HTTP

Fatta questa breve premessa andiamo a vedere quali possono essere i campi di intestazione usati, nel passato per effettuare questa primordiale forma di profilazione utente.

Questi includono campi quali **User-Agent** e **Accept** i quali servono rispettivamente a identificare univocamente il *browser* usato e specificare quali caratteristiche questo supporta come il linguaggio in uso, in questo modo diamo al *server HTTP* la possibilità di scegliere tra diverse pagine in base, ad esempio, alla lingua supportata. Il problema principale di questa tecnica è che spesso questi campi vengono usati in modo improprio e non sono direttamente personalizzabili dall'utente.

Dal punto di vista logico questa tecnica nasconde la fase di selezione della pagina, quindi si evita di dover aggiungere una pagina di scelta per l'utente, comunque un inconveniente è che questo metodo sovraccarica il protocollo *HTTP*, nato con lo scopo di veicolare i contenuti presenti sul *web*, in modo che possa gestire anche informazioni che non hanno nulla a che fare con lo scambio dei dati se non la scelta del dato da inviare, ma che riguardano la forma con cui questi vengono presentati; questa ultima affermazione mette in

evidenza come manchi l'incapsulamento necessario nel passaggio tra protocollo per il trasporto dei dati e protocollo per la profilazione utente che dovrebbe essere invece nascosto dal punto di vista del protocollo HTTP.

1.1.2 Metodologie di profilazione

Nel caso del *web* abbiamo visto come una prima forma di profilazione sia stata realizzata mediante la selezione del contenuto, ma questa non è l'unica tecnica che si può adottare, anzi, in una situazione in cui il contenuto sia di natura multimediale, come filmati video o sequenze audio, tale metodologia risulta spesso inadeguata; in questo ultimo caso infatti la profilazione utente in genere pviene svolta adattando il contenuto, al momento della richiesta, generando, al volo, una versione adattata del filmato o di altro. Tale soluzione, peraltro, è adottata anche nel caso del web, infatti se pensiamo ai meccanismi di inserimento del codice, da interpretare sul *server web*, nelle pagine stesse, vediamo come tale tecnica di adattamento "just in time" viene in realtà già utilizzata, non tanto per adattare il contenuto ma in particolare per selezionarne una parte, in base a certi criteri. Le metodologie di profilazione che si incontrano, quindi, nel mondo del web sono in genere di due tipi:

- Selezione del contenuto guidata da qualche parametro.
- Generazione automatica da parte del server di un contenuto "adattato" alle esigenze sulla base dell'identità dell'utente, più che alle sue caratteristiche.

1.1.3 Gestione dei profili attraverso web

Come abbiamo visto nel passato i profili relativi agli utenti non erano esplicitati, questo significa che da parte del cliente di un servizio non venivano inviati, in chiaro, informazioni relative allo stesso, e il rudimentale meccanismo di matching era eseguito sulla base di poche informazioni, fondamnetalmente l'identità dell'applicazione usata dall'utente. Oggi, visto l'aumento dei servizi fruibili in rete nonchè le tipologie di dispositivi utilizzabili, nasce la nuova necessità di evidenziare una maggiore quantità di informazioni relative al dispositivo da cui proviene una richiesta di erogazione, in modo da migliorare da un lato il servizio offerto al cliente, e dall'altro l'utilizzo delle risorse intermedie occupate nell'erogazione.

In un contesto in cui sia disponibile, in qualche modo, un accesso all' *web* è una scelta opportuna sfruttare questo per reperire, e gestire i profili associati ai dispositivi.

Anzitutto occorre fare una premessa sul luogo in cui vengono mantenuti i profili dei dispositivie su come vengono trasmessi all'erogatore di un servizio. Come primo approccio si

potrebbe pensare di memorizzare il profilo sul dispositivo, e che questo venga inviato, per ogni richiesta verso il fornitore. Questa soluzione risulta assai scomoda per diverse ragioni:

- I profili vengono continuamente trasmessi dal dispositivo al fornitore.
- I profili sono ridondanti e quindi vengono memorizzati in copie uguali su dispositivi uguali.
- Conseguenza della ridondanza, enunciata al punto precedente, è la difficoltà di aggiornamento, nonchè la possibilità di incongruenze.
- Ultima, ma non per questo di minore importanza, la sicurezza, infatti i profili potrebbero essere manomessi e potrebbe essere, con ciò, compromessa l'erogazione del servizio stesso.

Soluzione a tutti questi problemi consiste nel detenere i profili in un contenitore separato, accessibile possibilmante, via *web*, e mantenere nei dispositivi solo un identificatore a tale profilo. L'entità deputata all'erogazione del servizio riceve da parte del dispositivo cliente, un identificatore di profilo, e poi reperisce i dati relativi a questo prendendoli dal contenitore. Eventualmente il dispositivo potrebbe inviare le differenze rispetto al profilo originale, nel caso in cui questo venga modificato, i modo che l'erogatore del servizio potrebbe ottenere, per composizione, il profilo corretto per il dispositivo. Questo ultimo meccansmo riduce il traffico necessario alla conoscenza del profilo da parte dell'erogatore, e al tempo stesso, consente di avere un profilo che può tenere conto dei mutamenti avvenuti sul dispositivo utilizzato per fruire del servizio ad opera dell'utente che ne può avere modificato la configurazione o parte di questa.

Il mondo del *web* fornisce un utile strumento per testare tutto ciò in quanto il protocollo di trasporto delle informazioni è dotato delle estensioni sufficienti per poter trasmettere e reperire profili assieme ovviemente alle comuni richieste utilizzate per *navigare in internet*.

1.2 Terminologia adottata

Prima di passare alla descrizione delle possibili problematiche legate alla gestione di profili utente, vediamo qui di seguito le definizioni dei termini adottati in questo lavoro di tesi. In particolare viene definito cosa si intende per *profili utente* ed anche per *cache*, inoltre vengono brevemente fornite le motivazioni che stanno alla base della necessità di definire tali caratteristiche.

1.2.1 Metadato

Il termine metadato indica un "dato sul dato". Solitamente viene inteso nel significato di "dato strutturato sulle risorse" da utilizzare a supporto di un'ampia gamma di operazioni, tra cui, la descrizione e presentazione delle risorse, la gestione delle risorse informative, e la loro conservazione a lungo termine. Il metadato è l'informazione sull' organizzazione del dato, sui vari domini del dato e sulle relazioni tra questo e il suo campo di applicazione. I metadati non si trovano solo in ambito informatico ma li si trova anche nel mondo reale e vengono quotidianamente utilizzati, si pensi, ad esempio, agli indici dei libri, oppure agli schedari nelle biblioteche. I metadati sono a loro volta dei dati e quindi possono essere soggetti a descrizione tramite altri metadati.

Oggi il termine metadato è stato, forse impropriamente, esteso anche ai dati che descrivono gli *user* di una certa risorsa, gli *user profile,* dove, per *user,* si intendono sia gli utenti finali sia anche dei "dispositivi" hardware o software; questa accezione può essere considerata corretta nel caso vengano considerati *utenti* e *risorse* come entità paritarie, nel senso di "entità di cui deve essere fornita una descrizione" all'interno del sistema.

I metadati possono essere distinti in base al loro utilizzo:

1. metadati per facilitare la ricerca dei dati: vengono utilizzati per dare una descrizione semantica sui dati ed, in genere, sono statici rispetto al singolo dato.
2. metadati per migliorare e/o gestire, l'organizzazione dei dati: sono dati statistici, che quindi vengono usati per meglio organizzare la struttura o lo stoccaggio dei dati, oppure riguardano lo *stato* del dato in uso; possono essere usati in caso di "transizione" oppure nel caso la risorsa abbia vincoli sull'uso o ancora se siamo di fronte a una risorsa numericamente limitata; in questo caso, i metadati, variano nel tempo anche se riferiti allo stesso dato.

Inoltre possono essere distinti, in modo ortogonale, in base alla loro locazione:

1. metadati inseriti all'interno del dato: si trovano come tutt' uno logico o fisico con la risorsa oggetto.
2. metadati separati dal dato che descrivono: si trovano in una locazione fisicamente o logicamente separata dal dato che descrivono.

1.2.2 Profilo utente

Per profilo utente si intende appunto l'insieme di tutti i metadati che descrivono un *utente* dove per *utente*, come peraltro accennato in precedenza, si intende non solo la persona

ma anche il particolare dispositivo, fisico o logico, atto alla fruizione di un certo servizio o all'utilizzo di una certa risorsa. Il *profilo utente* può contenere quindi non solo informazioni relative alle abitudini, alle preferenze di un utilizzatore reale ma anche informazioni sui dispositivi Hardware e Software utilizzati per accedere ad un certo servizio o risorsa. Il punto chiave è che queste informazioni possono essere usate per *adattare il contenuto* che rappresenta la risorsa oggetto dell' utilizzo.

1.2.3 Capability

Per *capability* si intendono le *capacità, caratteristiche, facoltà* di un certo *utente*. Queste spesso sono caratteristiche fisiche del dispositivo, o del sistema software; tutte queste informazioni possono essere più o meno strutturate per facilitarne l'utilizzo in un contesto o per meglio comprenderne il significato, od ancora per distinguere tra omonime caratteristiche.

1.2.4 Adattamento del Contenuto

Adattamento del contenuto proviene dall'espressione inglese c*ontent adaptation* e sta a significare l'atto, in un ambiente in cui sia presente una entità che fornisce un servizio o mette a disposizione una risorsa, di adattare il servizio o la risorsa alle capacità dell' utilizzatore di tale risorsa. Quando un *utente* richiede un servizio questo può essere fornito in diversi modi e in diverse forme e può essere messo a disposizione con diverse caratteristiche qualitative. Tutte queste caratteristiche, che vanno poi specificate, in base alla tipologia di applicazione, non sempre sono caratteristiche tra loro ortogonali, ovvero logicamente distinte, ma spesso sono collegate; a tale proposito possiamo porre il seguente esempio: si supponga di fornire come servizio la fruizione di fotografie, è ovvio che se prendiamo una caratteristica come la "bontà" della foto (intesa come la possibilità di "distinguere gli oggetti" nella foto), questa è innegabilmente correlata alla dimensione della fotografia in quanto, a parità di risoluzione (la *risoluzione* è il rapporto *punti di colore / dimensione*), la foto più piccola avrà una quantità di *punti di colore* inferiore rispetto a quella più grande e che quindi rimane meno "definita", nel senso che se provo a ingrandire l'immagine con una lente, ad esempio, questo ingrandimento risulterà con una risoluzione inferiore e quindi gli oggetti presenti nella stessa risulteranno più sgranati.

1.2.5 Profilazione Utente

Profilazione utente è la traduzione letterale dell'espressione, in lingua inglese,*user profiling,* dove il termine *profilazione* assume il suo significato dal termine inglese che ha il significato letterale di *sagomare*, espressione che può essere interpretata come "adattare il

contenuto" alle esigenze di un utente. A questo punto è necessaria una delucidazione rispetto alla precedente definizione di *adattamento del contenuto* : per *profilazione utente* si intende il processo di adattamento, per un utente, che include, tra l'altro, anche la *negoziazione* come vedremo più avanti, l' *adattamento del contenuto* è il passo finale della *profilazione utente* .

1.2.6 Negoziazione dei Contenuti

Negoziazione dei contenuti dal significato letterale di *content negotiation* si riferisce al fatto che, quando un servizio viene fornito e quando questo servizio può essere fornito in diversi modi e forme , tra *utente* e *fornitore* deve essere stabilita la forma e il modo in cui deve essere rispettivamente ricevuto e fornito il servizio, questa fase decisionale prende appunto il nome di *negoziazione del contenuto* ovvero si decide come e in che maniera fornire il servizio.

1.2.7 Ambiente di esecuzione

Con il termine di *ambiente, contesto, environment* in informatica si intende, spesso, l' *ambiente di esecuzione* in cui una applicazione deve eseguire; tale contesto è rappresentato, non solo dal sistema operativo e dalle primitive messe a disposizione da quest' ultimo, ma anche dai protocolli a cui si appoggia l'applicazione stessa, i quali possono essere, più o meno, ben definiti. Inoltre il contesto può essere visto, quasi sempre, sotto diversi aspetti, in base al *livello di astrazione* su cui poniamo la nostra descrizione. Si noti infine che, se l'applicazione viene ad interagire in un *contesto* di rete, l' *ambiente* viene ad includere, a basso *livello*, anche i protocolli utilizzati per lo scambio di informazione tra le diverse macchine che compongono la rete stessa.

Va infine osservato che il termine ambiente può essere talvolta ambiguo in quanto in una applicazione di rete può essere individuato un *ambiente locale*, detto anche *piattaforma di esecuzione,* che include le parti hardware e software di una macchina su cui esegue un processo, ed un *ambiente globale* o *di rete* che include l'insieme delle macchine interconnesse.

1.2.8 Memoria Cache

Le memorie cache furono introdotte per la prima volta nei processori dei computer per migliorare l'uso della memoria da parte di questi; queste memorie, di limitata capacità ma di eccezionale velocità, per quel che riguarda l'accesso, vengono introdotte in modo che i dati utilizzati dal processore vengano prima caricati in esse e poi utilizzate con velocità superiore a quella che si potrebbe avere se il processore dovesse utilizzarli prelevandoli direttamente dalla memoria convenzionale.

In informatica, per *memoria cache* o più brevemente *cache* si intende un tipo di memoria o un sistema di stoccaggio dati che deve avere le seguenti caratteristiche:

1. si frappone tra il fornitore di una risorsa e colui che ne fruisce
2. deve avere come scopo l'ottimizzazione dell'uso della risorsa
3. deve essere, il più possibile, trasparente all'uso da parte delle entità coinvolte

Passiamo a spiegare il significato delle caratteristiche presentate. Per il primo punto è evidente che, la cache deve fare da tramite nel trasporto di una risorsa, tra chi la richiede, e chi la fornisce; in pratica la risorsa viene reperita presso il fornitore dal *sistema di cache* e poi viene fornita all'utente quando questo ne fa uso. Per il secondo punto si nota che le possibilità sono diverse, ovvero l' introduzione della *memoria cache* viene fatta per migliorare l'accesso alla stessa da parte dell'utente, in particolare per migliorare i tempi di accesso. Il terzo punto è di fondamentale importanza, ovvero dice che nel sistema l'accesso delle risorse da parte dell'utente deve sempre avvenire in modo trasparente all'introduzione della *memoria cache*.

Il fatto che la *memoria cache* contenga una copia dei dati pone delle problematiche per quel che riguarda l'uso concorrente di una risorsa in caso questa venga anche modificata.

Un altro punto importante riguardante le *cache* è la tecnica con cui questa viene gestita, ed in particolare come vengono inseriti e rimossi gli elementi nella stessa. Infatti la *memoria cache* è limitata e può contenere un numero limitato di risorse quindi la tecnica con cui vengono inseriti e rimossi gli elementi diventa una scelta progettuale di grande importanza. Le tecniche con cui questa viene gestita si basano sui principi di:

- Località temporale: un dato richiesto da poco tempo ha una elevata probabilità di essere richiesto nuovamente a breve termine.
- Località spaziale: un dato fisicamente o logicamente vicino ad un dato da poco usato ha grande probabilità di essere richiesto negli istanti immediatamente successivi.
- Frequenza: per definizione un dato che viene richiesto un maggiore numero di volte è quello con maggiore probabilità di essere richiesto nuovamente, ha senso quindi trattenere questi dati che sono letteralmente i "più richiesti".

Altrettanto importante caratteristica che riguarda la gestione della *cache* è la *tecnica di replacement* degli *oggetti* nella stessa, infatti è questa a determinare come vengono scelti gli elementi da inserire e rimuovere dalla cache nel caso venga raggiunto il limite, in termini di spazio o capacità, di occupazione della cache. Le tecniche con cui questa viene gestita sono

fondamentalmente in base alle caratteristiche introdotte.

1. In base alla frequenza delle richieste di un oggetto; in genere il più frequentemente richiesto viene rimosso per ultimo.
2. In base all'istante in cui è stata usata la risorsa; in genere la risorsa richiesta per ultima è l' ultima ad essere scartata secondo una strategia di gestione tipicamente FIFO (First In First Out).
3. In base alle dimensioni o al tempo necessario per il reperimento di una risorsa, da parte del sistema di cache stesso. In questo caso in genere le risorse di dimesioni maggiori vengono scartate per ultime dalla cache.

1.2.9 Proxy

Letteralmente *proxy* significa *delegato*, in informatica identifica una entità intermedia che si pone tra il fornitore di un servizio e il fruitore di questo per migliorare, adattare la fruizione di una risorsa. Il *proxy* non ha solo scopo di accumulatore di risorse, come la *cache*, ma può avere anche altre motivazioni; ad esempio in una architettura *client/server* (vedremo più avanti di cosa si tratta) il *proxy* si interpone tra le due entità e si comporta come il *client* per il *server* e come *server* per il *client*, questo significa che può essere usato per fase da cache ma non solo, anche per fare da adattatore, o per monitorare il traffico, in termini di informazioni scambiate, tra *client* e *server*.

Il *proxy* più spesso viene usato per adattare il contenuto o per gestire le informazioni sui profili ad esempio può essere un gestore introdotto per combinare i profili utente con i profili del dispositivo oppure per aggiungere informazioni sulla connessione tra cliente e fornitore di un servizio nella richiesta che proviene dal cliente.

1.2.10 Authoring

In informatica per *authoring* si intende la preparazione di una *presentazione* ed in genere si usa tale termine per intendere il processo di creazione di una pagina WEB, in questo caso si parla di *web-authoring*. Questo termine risulta particolarmente recente ed è stato introdotto probabilmente per semplificare, sintatticamente, le definizioni che spesso si danno in campo informatico, va tuttavia rilevato che spesso è usato in modo molto generico, anche se sempre in campo informatico.

1.2.11 Flusso di dati

In informatica per flusso di dati, o *stream* per usare il termine inglese, si intende l'astrazione di un dato che transita da una entità ad un'altra, tale concetto include inoltre l'idea

di dato in transito, quindi include, in maniera insita, il concetto di tempo; per tempo si intende qualcosa di quantificato che misura il succedersi degli eventi. Nel caso di applicazioni multimediali per flusso si intende spesso la fruizione di un contenuto multimediale incluse anche le operazioni di rendering e visualizzazione.

1.3 Supporto necessario alla profilazione

In questo paragrafo cerchiamo di puntare l'attenzione su cosa sia necessario per effettuare la profilazione utente.

1.3.1 Metadati scambiati

Ovviamente per poter attuare la profilazione utente bisogna fare uso di metadati. Tali metadati devono descrivono sia le caratteristiche del dispositivo dell'utente, sia la risorsa, dando una descrizione della stessa e fornendo un elenco di possibili manipolazioni attuabili per trasformarla in una forma compatibile. I metadati relativi all'entità client, danno una descrizione dettagliata di quali siano le caratteristiche hardware e software di quest'ultimo. Nel caso di dati multimediali le caratteristiche potrebbero essere dal punto di vista hardware ad esempio la dimensione dello schermo, la frequenza di aggiornamento dello stesso, la risoluzione in termini di pixel per centimetro quadrato, il rapporto tra le dimensioni dello schermo; dal punto di vista software le caratteristiche potrebbero essere la risoluzione logica, o le risoluzioni supportate, i codec utilizzabili per decomprimere il filmato e così via.

Oltre a tali caratteristiche possiamo introdurre le proprietà relative all'ambiente in cui viene fruito il contenuto multimediale, inteso nel senso più ampio del termine ad esempio in un contesto di rete si potrebbero introdurre descrizioni dei nodi interessati ai flussi multimediali che ne riportino le caratteristiche in termini di prestazioni e che partecipano all'erogazione.

1.3.2 Protocollo e negoziazione

Nel successivo paragrafo accenniamo alla scelta finale della risorsa da erogare, tuttavia è necessario presentare prima il meccanismo con cui possono coordinarsi le entità coinvolte cioè il protocollo di negoziazione in cui può prendere parte anche l'utente, inteso come l'utilizzatore umano del dispositivo, che può specificare formati preferiti o altre caratteristiche durante la negoziazione stessa. La necessità di un protocollo di negoziazione si palesa quando da un lato abbiamo un utente che supporta un certo numero di caratteristiche e dall'altro abbiamo un fornitore in grado di presentarci una medesima risorsa in diversi formati tutti compatibili col nostro dispositivo utente. In questa situazione è ovvio come sia necessario che

avvengano scambi di informazione, anche a più turni, tra utente e fornitore, in modo che questi concordino un formato finale che sia il più consono per entrambi e per l'ambiente in cui avviene la distribuzione della risorsa. Ebbene le regole a cui devono sottostare tali scambi di informazione sono appunto il *protocollo di negoziazione* che ha come fine ultimo la scelta della forma di distribuzione della risorsa. Si noti che a seguito di tale fase decisionale si potrebbe anche decidere di non inviare la risorsa all'utente. Ad esempio in [CCPPex] usando richieste di tipo Mandatory (con vincolo di obbligatorietà da parte del fornitore di una risorsa) è possibile, nel caso il fornitore non supporti tale richiesta, una terminazione senza fruizione da parte dell'utente.

Infine osserviamo che quando sul percorso lungo il quale avviene la distribuzione delle risorse sono presenti entità quali i proxy in grado di eseguire anche del caching, in tale caso il protocollo di negoziazione può includere, non solo i formati e le modalità con cui verrà fornita la risorsa richiesta, ma anche la possibilità di reperire la medesima risorsa da un proxy cache, possibilmente più vicino, in senso logico, al richiedente, oppure la possibilità di eseguire una parte dell'adattamento al formato supportato dall'utente, in uno dei proxy intermedi.

1.3.3 Sistema di selezione

Altra importantissima caratteristica è la tecnica con cui viene scelto, il formato finale e le modalità con cui viene distribuita la risorsa verso l'utente finale. L'utente infatti invia una richiesta in cui dichiara anche le proprie caratteristiche mediante l'uso di metadati; poi però è necessaria una fase decisionale, che può essere anche distribuita, in cui viene scelto il formato e la modalità con cui la risorsa deve essere inviata all'utente.

1.3.4 Formato sorgente

Questa caratteristica è di cruciale importanza: con formato sorgente intendiamo quale sia il formato con cui la risorsa viene inizialmente preparata. Infatti in un contesto in cui la risorsa può essere fruibile in diversi formati e da differenti tipologie di dispositivi utente è importante la scelta del formato con cui viene inizialmente preparata tale risorsa. Un esempio di ciò ci viene, ancora una volta, dal mondo del web in cui oggi, spesso, le pagine vengono preparate in formato XHTML (eXtensible HyperText Markup Language) per poi essere sottoposte a una trasformazione XSLT (eXtensible Stylesheet Language Transormation), che le modifica in modo che possano meglio adattarsi al browser in uso e alle caratteristiche da questo supportato. Questa tecnica può essere adattata al mondo della multimedialità considerando un formato sorgente che sia facilmente trasformabile per poter essere adattato ai

formati supportati dalle diverse applicazioni client.

Il fatto di avere un formato il più generico possibile priva il documento iniziale delle estensioni relative ai vari dispositivi *client* e questo facilita il processo di *authoring*; infatti chi prepara la risorsa non deve più preoccuparsi dei dettagli relativi al dispositivo ma può concentrarsi sul contenuto della risorsa. Per ottenere infine il formato con le estensioni relative ai vari dispositivi si introduce un componente separato che effettua l'adattamento arricchendo il formato iniziale. In questo modo separiamo i due compiti, logicamente separati, di preparazione ed adattamento.

Avere un formato in cui mancano estensioni particolari rende minori i costi di manutenzione infatti se prima era necessario preparare la stessa risorsa in diverse versioni, con ovvi problemi di incongruenza e di aggiornamento, ora è possibile gestirne una sola senza preoccuparsi di dovere aggiornare diverse copie.

Infine se abbiamo contenuti composti da diverse parti da assemblare in fase di distribuzione, il fatto che queste siano più generiche possibile ne facilita l'integrazione nella fase finale.

Multimodalità:

Per multimodalità si intende in genere la possibilità di adattare il contenuto a diversi dispositivi, dove, per diversi, non si intende diversi come caratteristiche all'interno dello stesso ambito applicativo, ma si intende diversi anche come ambito applicativo, un esempio di ciò può essere nel web una pagina che sia accessibile in modo *normale* attraverso schermo e Personal Computer, ma anche in modalità adatta ai non vedenti, quindi tramite un dispositivo in grado di dare una rappresentazione equivalente dello stesso ad esempio in formato braille, o ancora un sistema in grado di trasformare la generica pagina in "parlato"; in ambito multimediale la forma più semplice che si possa pensare di multimodalità potrebbe essere un filmato e la sua "versione audio" raccontata. Ovviamente per supportare questo tipo di trasformazioni dovrà essere scelto con estrema cura il formato sorgente per la risorsa messa a disposizione, e lo stesso processo di *authoring* risulta essere, da un lato, più complesso, in quanto richiede, a colui che pubblica la risorsa maggiori conoscenze e competenze, dall'altro più ricco, in quanto consente di inserire una quantità di informazioni sul dato superiore a quelle necessarie ad una visualizzazione nell'ambito applicativo per cui era stata inizialmente pensata la risorsa stessa.

1.4 Caratterizzazione della profilazione utente

Nel sottoparagrafo immediatamente successivo enunceremo le fasi della profilazione utente, ovvero come questo processo abbia luogo e quali siano le fasi principali.

Successivamente passeremo ad osservare come la profilazione può essere vista sotto diversi punti di vista, ad esempio uno di questi può essere la finalità per cui viene attuata, un altro può essere quale sia l'entità che realizza la profilazione, e così via; possiamo quindi caratterizzare la profilazione utente in base a diversi aspetti che andremo ad esporre. Le diverse caratterizzazioni da noi enunciate non vogliono essere né esaustive né tra tra loro esclusive, si vuole solo proporre una schematizzazione utile per esaminare le solzioni che saranno proposte nel seguito.

1.4.1 Fasi della profilazione utente

Le fasi di un generico sistema di profilazione utente sono temporalmente illustrate dal seguente diagramma preso da [NEG].

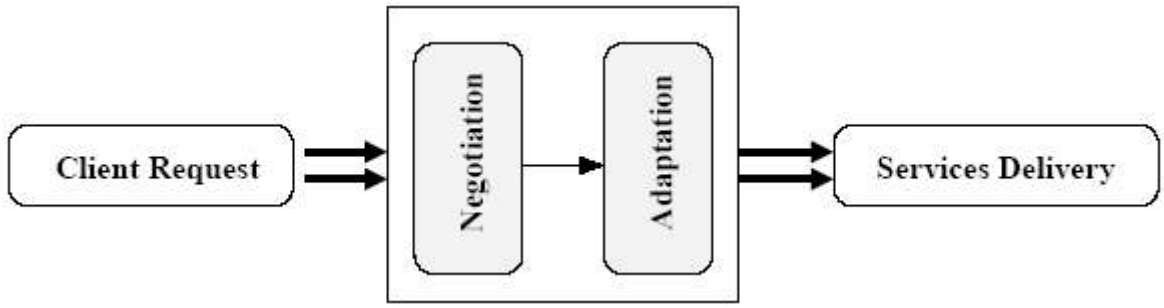

Figura 1.2 Adattamernto del contenuto

In realtà il sistema mostrato in Figura 1.2 illustra la tecnica di adattamento del contenuto che è lo scopo della profilazione utente, inizialmente l'entità cliente invia la richiesta con all'interno la specifica del proprio profilo, viene svolta una fase di negoziazione, si passa all'adattamento sulla base dei risultati della negoziazione ed infine viene svolta la fase di *Services Delivery* che consiste nella distribuzione effettiva del servizio o della risorsa adattata.

1.4.2 Scopo e caratteristiche su cui viene basata la profilazione utente

La profilazione utente nasce con l'intento di migliorare l'utilizzo di una risorsa da parte del particolare dispositivo. Inizialmente la profilazione utente, come è insito nel nome, veniva utilizzata per lo specifico richiedente; un bell'esempio di profilazione fatta per l'utente si vede

oggi nei servizi web e di posta elettronica in cui l'utente può personalizzare la pagina di ingresso e/o l' interfaccia della stessa. Con l'evolversi della tecnologia la profilazione utente ha incluso anche la possibilità di dare una descrizione dell'ambiente in cui avviene l'erogazione del servizio,ad esempio del dispositivo verso cui il servizio viene erogato, allo scopo di migliorare la fruizione ma anche l'utilizzazione delle risorse.Un esempio può essere il caso riportato in Figura 1.3 in cui nel passaggio dal server verso il dispositivo finale attraversiamo dei dispositivi con una capacità, in termini di banda passante, sempre minore.

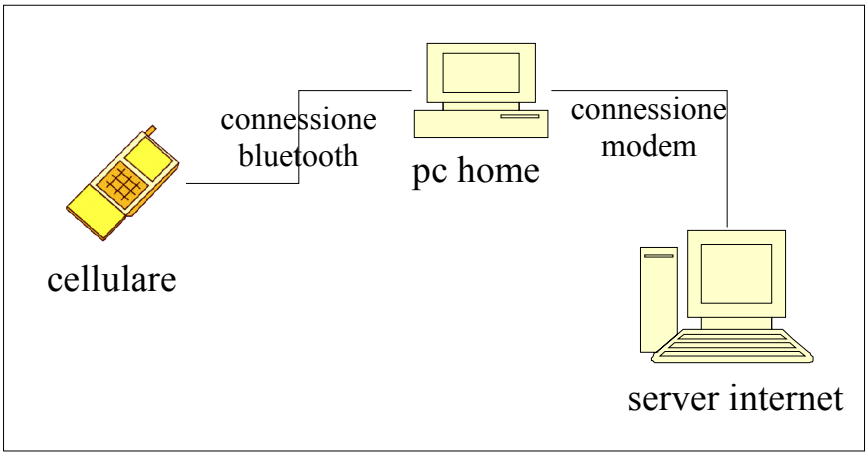

Figura 1.3 Caratteristiche del percorso di connessione

Altro aspetto importante sono le caratteristiche in base alle quali viene fatta la profilazione utente. Come abbiamo visto, nel caso del web, un primo approccio consiste nello scegliere un profilo sulla base solo dell'applicazione utente, il browser nel caso suddetto, e una volta individuato questo può essere attuata la profilazione. Questa tipologia di profilazione poco si presta a evoluzioni future. Se ad esempio di presenta una applicazione client per cui non è stato studiato un profilo il servizio non può essere erogato per sopperire a questa carenza si può pensare di utilizzare un *profilo di default*. Si noti però che questo approccio, sebbene attuabile, presenta alcuni svantaggi:

1. l'impossibilità da parte dell'utente di modificare le preferenze; infatti queste vengono decise , una volta per tutte, da chi eroga il servizio.
2. l'impossibilità di includere informazioni ausiliarie quali quelle sull'ambiente in cui avviene la distribuzione del servizio (il *delivery*), o sulla piattaforma su cui è i esecuzione l'applicazione cliente quali notizia sul software e sull' hardware disponibile
3. la staticità del sistema; con questo si intende che un profilo riguarda una e una sola applicazione cliente ed una ed una sola sua versione; in pratica se il browser subisce una

modifica, più o meno, radicale tra una versione ed un'altra è necessario studiare un nuovo profilo.

Un altro modo per attuare la profilazione utente consiste nell'estendere il protocollo utilizzato per il trasporto delle informazioni aggiuntive necessarie alla profilazione. Sebbene piccole estensioni possano giovare sicuramente al protocollo, una estensione dello stesso per veicolare informazioni relative alla profilazione è una pratica ancora sconsigliabile, infatti un protocollo "pensato" per il trasporto delle informazioni spesso non si presta alla negoziazione e all'introduzione di informazioni aggiuntive, infatti la negoziazione spesso necessita di diversi scambi di informazione tra fornitore e utente mentre nel caso di erogazione il processo è unilaterale. Tuttavia la scelta di inserire, nel protocollo, informazioni aggiuntive sul dispositivo cliente e sull'ambiente di erogazione è sicuramente una buona scelta. Utilizzando un insieme di caratteristiche, più o meno strutturate, con i relativi valori consente di avere un quadro completo durante l'erogazione del servizio verso l'utente, inoltre possiamo pensare di aggiungere un numero qualsivoglia di campi di cui l'uso è prerogativa del sistema stesso, vale a dire che il sistema può scegliere se ignorare o utilizzare tali dati per effettuare la profilazione utente.

La soluzione di gran lunga migliore consiste nell'incapsulare le informazioni relative alla profilazione utente all'interno del protocollo di trasporto, in modo tale può essere possibile, veicolare la profilazione, in modo indipendente dal protocollo utilizzato per il trasporto delle informazioni, oltre al fatto che tale protocollo non risulta più "sporcato" dalle informazioni relative alla profilazione. Ovviamente per poter incapsulare il le informazioni, ovvero le *capability* relative all'utente e all'ambiente , il protocollo deve essere pensato anche per tale scopo.

1.4.3 Profilazione per selezione e per generazione

La profilazione utente può essere realizzata con diverse tecniche tuttavia possiamo individuare due situazioni estreme

1. Profilazione per selezione
2. Profilazione per generazione o modifica

Per *profilazione per selezione* si intende il fatto che uno stesso documento è presente in diverse versioni, e una volta che l'utente ne fa richiesta viene scelta la versione a lui più

consona.

Per *profilazione per generazione o modifica* si intende la possibilità, a partire da un formato, generico o *raw* (per *raw* si intende un dato *grezzo*, ad esempio nel caso di un filmato può essere considerato *raw* la sua forma non compressa) applicando delle funzioni di trasformazione, di un formato più adatto all'utente finale. Un'altra tecnica può consistere nel generare un dato a partire dalle sue parti e assemblate per l'utente finale; un esempio di tale tecnica, nel caso di presentazioni multimediali, può essere nell'assemblaggio di video e audio scelti singolarmente tra diverse versioni degli stessi. Da un punto di vista puramente concettuale le due tecniche, selezione o modifica e generazione, sono assolutamente equivalenti, infatti si può passare da un sistema all'altro, senza perdita, se pensiamo il frutto delle trasformazioni come le diverse versioni della stessa risorsa da scegliere e viceversa; tuttavia va osservato come la tecnica di modifica/generazione sia molto più flessibile, è infatti assai difficile, per non dire impossibile, pensare di memorizzare e mantenere consistenti tante versioni quante possono essere generate a tempo di esecuzione, nei sistemi con generazione automatica, nell'equivalente sistema funzionante per selezione.

Vi sono infine soluzioni miste, soprattutto in ambito distribuito, in cui le due tecniche si confondono a tal punto da dare luogo a tecniche considerabili come assolutamente indipendenti. Questo può essere in caso di un sistema di erogazione di contenuti multimediali distribuito, dotato, sui vari nodi, di sistemi di cache in cui vengono memorizzate certe versioni del filmato; in questa situazione ad esempio possiamo avere un sistema che detiene la risorsa in tutte le sue versioni, o che è in grado di trasformarla nelle diverse versioni. Le diverse cache che posseggono tali versioni e l'applicazione cliente può richiedere la risorsa dal sistema possessore della stessa o richiederla alla sua cache che detiene la risorsa nel formato che più si addice al profilo del client. In Figura 1.4 viene illustrata la situazione in cui abbiamo una copia della risorsa sul percorso di erogazione del servizio più vicina al sistema.

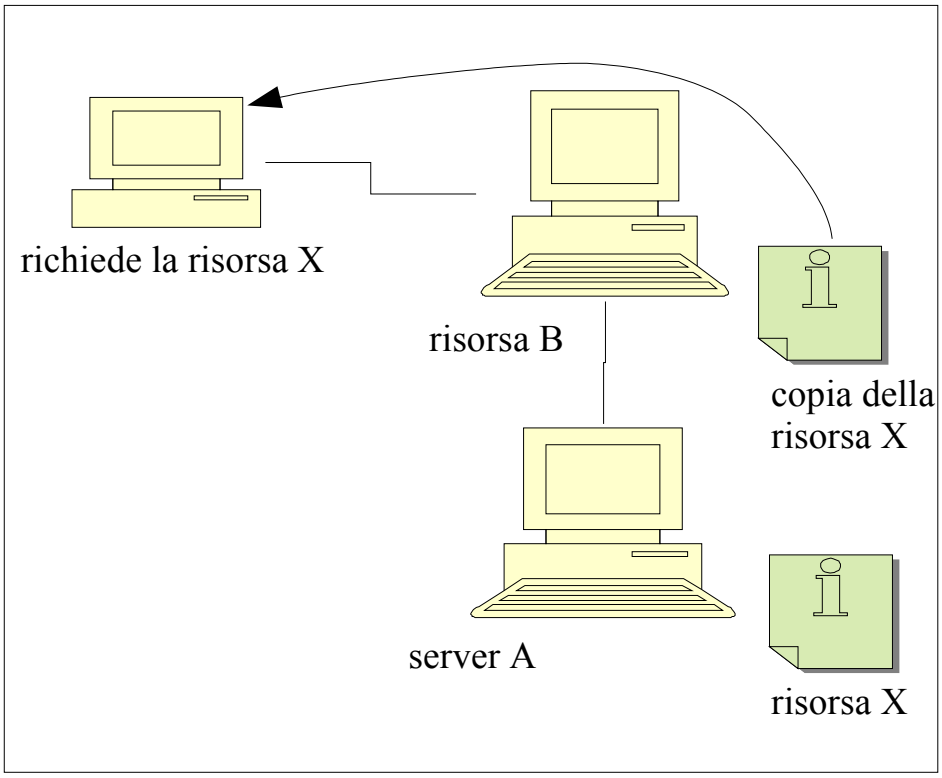

Figura 1.4 Copie sul percorso di richiesta

Un altro esempio di sistema misto può essere il seguente, esposto in Figura 1.5 in cui abbiamo una entità che possiede la risorsa collegata ad un *proxy* in grado di effettuare la trasformazione del formato da HTTP a WAP.

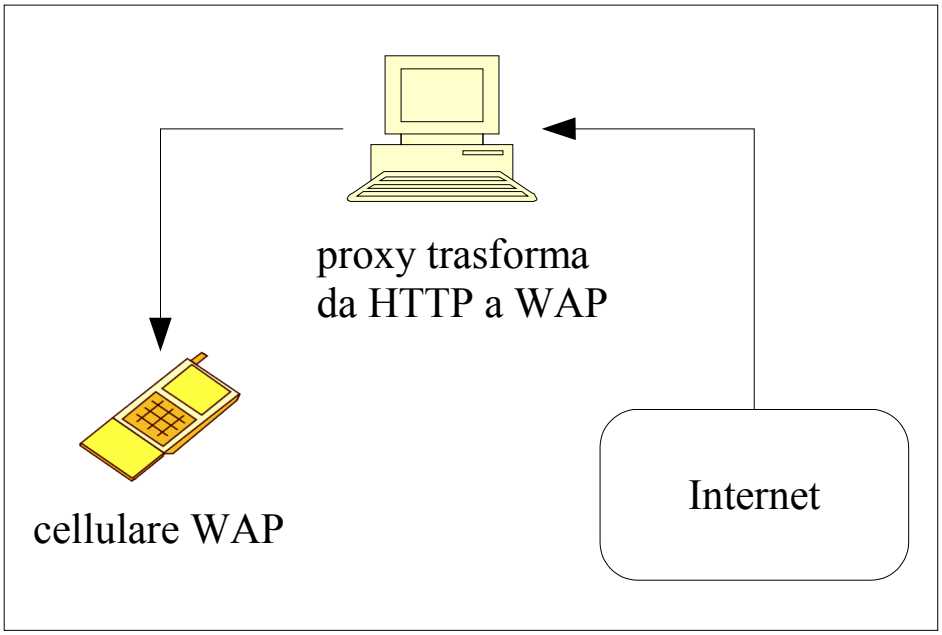

Figura 1.5 Proxy adattatore

Per concludere un ultimo caso in cui si può attuare una tecnica mista può essere il caso in cui abbiamo un *proxy* che smista un singolo flusso per diversi utilizzatori localizzati presso di lui; questo viene illustarto in figura 1.6

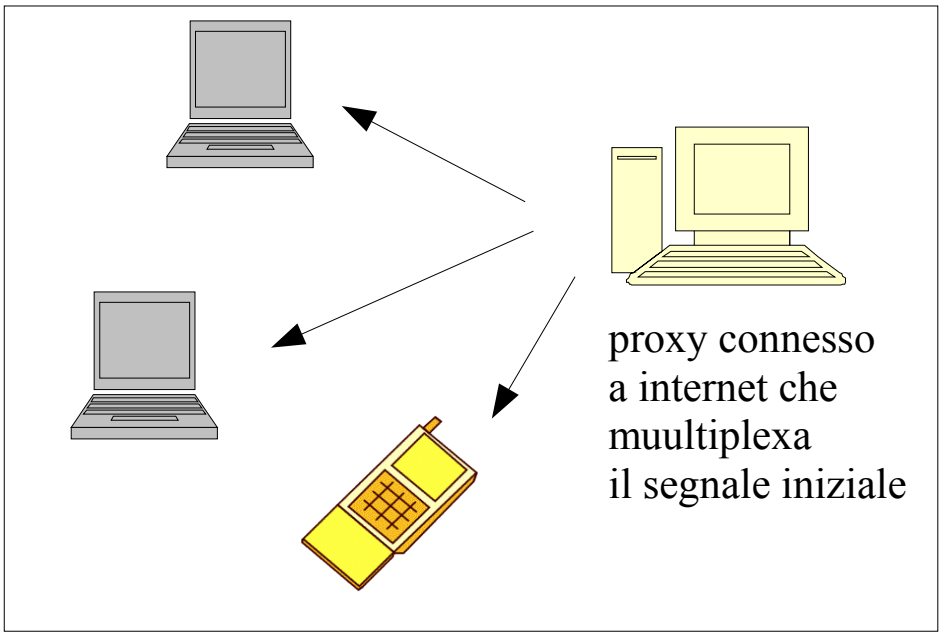

Figura 1.6 Multiplexing del segnale

1.4.4 Politiche per la realizzazione della profilazione utente: file separati, script, moduli per la trasformazione

Le metodologie atte a realizzare la profilazione utente sono le più svariate, e possono essere suddivise a seconda della tecnica con cui vengono specificate le politiche di gestione della profilazione. Ad esempio si può avere un file separato di specifica delle politiche di trasformazione: questo è il caso dell'uso di contenuti espressi in un file XML associati a un qualche tipo di trasformazione specificata in un file separato. Questo metodo consiste nell'avere il file, la risorsa, in un formato ricco di informazione e un file che separato che specifica quali trasformazioni debbano essere attuate per avere il formato voluto. In questo caso abbiamo quindi un file in formato XML a cui viene successivamente associato un file di trasformazione per ottenere una informazione compatibile col browser utilizzato; i file di trasformazione possono essere anche più di uno ciascuno associato ad un particolare browser.

Un altra tecnica può consistere nell'utilizzare un linguaggio di scripting con cui specificare la trasformazione; lo script può essere incluso o meno nella risorsa da personalizzare Consideriamo ad esempio la visualizzazione: nel caso del web assistiamo all'uso di linguaggi di scripting, sia lato client come javascript, sia lato server come ASP o

PHP con i quali la pagina HTML viene formattata al volo per il browser selezionato. Si noti però che in un ambito multimediale tale tecnica risulta difficilmente attuabile non esistono infatti standard per l'inserimento di parti interpretabili in file contenenti materiale multimediale.

Infine è possibilite avere componenti software, generalmente sul lato del fornitore, in grado di effettuare le trasformazioni necessarie alla profilazione utente. Tali componenti saranno attivati dai moduli che realizzano la politica di profilazione. Ad esempio possiamo avere un componente che cambia la risoluzione e/o la dimensione delle immagini inviate in base alle caratteristiche dichiarate dal dispositivo che le ha richieste.

1.4.5 Entità che realizzano la profilazione utente

All'interno di un sistema di profilazione utente le entità che realizzano l'adattamento possono essere molteplici, come molteplici possono essere quelle che prendono parte alla negoziazione del contenuto. Queste entità sono, come abbiamo già visto:

- l'entità che fornisce il servizio: il fornitore
- l'entità che riceve il servizio: l'applicazione cliente
- eventuali entità intermediarie presenti sul percorso di servizio dal fornitore al cliente: i proxy

Realizzare la profilazione utente significa prendere parte non solo all'adattamento del contenuto ma anche alla negoziazione dello stesso.

Per quel che riguarda la negoziazione le entità che prendono sempre parte alla negoziazione sono il fornitore e l'applicazione cliente, mentre gli intermediari a volte possono astenersi durante la fase di negoziazione; per quel che riguarda la fase di adattamento, se sono presenti degli intermediari il loro scopo è proprio quello di fornire un supporto all'adattamento magari per sgravare dal carico computazionale il fornitore, oppure per migliorare lo sfruttamento della rete tramite l'uso di caching, o ancora per arricchire di funzionalità le possibilità del fornitore nei confronti del cliente.

Seguendo sempre il parallelo con il mondo del web non ci sarebbe da stupirsi se l'adattamento, o parte di esso, venisse fatta lato client, infatti è ciò che avviene tuttora mediante l'uso, ad esempio, di linguaggi di scripting client come javascript o vbscript. Quello di usare un linguaggio di scripting non è la unica soluzione infatti in ambiente unix-like possiamo trovare anche browser quali *lynx* che adatta le pagine web per una visualizzazione su schermo a caratteri proponendo un testo alternativo. Ora nel caso di applicazioni

multimediali se si rende necessario una funzione di abbassamento della risoluzione di un filmato, è ovvio che tale operazione è preferibile che sia svolta sul sistema fornitore o su qualche intermediario, in quanto risulta essere una operazione assai pesante computazionalmente; però se invece dobbiamo passare da un rapporto di forma ad un altro senza che questo comporti variazione dei dati trasmessi, ad esempio mediante una semplice aggiunta di bande nere sopra e sotto l'immagine, questo potrebbe essere fatto lato cliente con il minimo sforzo.

1.5 Preferenze dell'utilizzatore e forzate dalle condizioni

Durante tutti i discorsi precedenti si è presa in considerazione l'idea di profilo utente come associata al dispositivo, i questo paragrafo distingueremo il *profilo utente* come un insieme di preferenze imposte dall'utente umano, da quello di *profilo del dispositivo* considerando questo come l'insieme delle caratteristiche, hardware e software del dispositivo utilizzato per la fruizione del servizio, e le caratteristiche che eventualmente caratterizzano le condizioni di contorno all'applicazione. Spesso durante un processo di *tailoring* ovvero di adattamento, il sistema si trova ad avere a che fare sia con profili imposti, e dai dispositivi, e dalle condizioni di contorno, ma a volte è possibile che in tale processo vengano presi in considerazione anche delle caratteristiche volute ad esempio dall'utente, in tale caso queste caratteristiche prendono il nome di *profilo utente*. Ovviamente il profilo utente potrebbe includere altresì informazioni non necessariamente associate al processo di profilazione, caratteristiche non funzionali quali ad esempio una rubrica associata all'utente, ma noi non ci interessiamo di queste e considereremo come *profilo utente* solo le caratteristiche che specificano caratteristiche volute dall'utente durante l'erogazione di un servizio. L'approccio usato in una tale situazione potrebbe essere il seguente:

In una prima fase si prendono in considerazione solo il *profilo del dispositivo*, la fase finale di questo processo ci porterà ad un insieme di possibili modalità di fornitura del servizio, compatibile con il dispositivo e con le condizioni.

In una seconda fase prendo infine in considerazione il *profilo utente* ed uso le preferenze che questo esprime per mettere in graduatoria le possibilità ottenute alla fase precedente, scegliendo infine quella più consona che in genere è la più alta in graduatoria, oppure, potrei dare all'utente la possibilità di scegliere ad esempio tra una soluzione ad alto costo compatibile con il *profilo utente*, ed una a costo magari bassissimo ma che risulta leggermente incompatibile col *profilo utente*.

Tuttavia questa non è l'unica soluzione: potremmo infatti pensare di utilizzare le *preferenze utente* per ridurre drasticamente le possibilità prese in considerazione durante la fase di negoziazione; in questo modo troviamo una soluzione "non ottima" ma il tempo utilizzato per reperirla è sicuramente inferiore.

Abbiamo visto quindi due possibili estremi nel trattamento dei *profili utente*, che si riflette nel considerarli come accessori, o come prioritari.

Un esempio di scenario può essere in seguente: consideriamo di avere un utente che utilizza due dispositivi per accedere a un servizio in dispositivo D1 ed il dispositivo D2 ora il servizio può essere fornito nel formato F1,F2 o F3 e la preferenza dell'utente è, in ordine, F1,F3 supponiamo inoltre che il dispositivo D1 supporti F1,F2,F3 mentre D2 supporti solo i formati F2,F3; è ovvio che quando l'utente utilizza il dispositivo D1 può usare il formato F1, mentre quando l'utente utilizza il dispositivo D2 il formato F1 non può essere usato mentre potremmo usare sui F2 che F3, in questo caso si usa la preferenza utente cioè F1,F3 per mettere in ordine di preferenza i formati accettabili, cioè vengono scelti, dai formati accettabili, quelli presi in ordine tra quelli preferiti, nel nostro esempio F1 non è presente tra quelli accettabili e quindi non viene scelto, F3 è presente e viene scelto come accettabile in prima posizione, in coda i formati compatibili ma non preferiti, nell'esempio F2, al termine prendiamo il formato F3 che è il primo tra quelli selezionati.

1.6 Conclusioni

In questo capitolo si è cercato di dare un quadro geneale di cosa sia la profilazione utente e di quali problematiche induca, oltre ovviamente a dare un insieme di definizioni dei termini utilizzati nel caso di profilazione. Inoltre abbiamo dato alcuni esempi in cui viene utilizzata, anche se in maniera lasca, la profilazione utente. Infine abbiamo mostrato come la profilazione possa includere preferenze imposte da un utente.

2 Stato dell'arte e standard correlati alla profilazione utente

In questo capitolo presenteremo anzitutto la correlazione esistente tra profilazione utente e QoS per poi passare in rassegna quattro sistemi per gestire la profilazione utene sviluppati nel mondo reale cioè le Media Feature, MPEG-21, CC/PP e UAProf che viene presentato insieme a CC/PP in quanto ne condivide in parte la sintassi. Di questi sistemi verrà presentata l'architettura utilizzando di volta in volta i termini da questi introdotti.Si fa inoltre un accenno a XML in quanto rappresenta il linguaggio di base per due di questi, MPEG-21, CC/PP e UAprof, ed in oltre perchè di per sè otrebbe essere usato anch'esso come linguaggio per descrivere i profili.

2.1 Livelli nel trattamento della profilazione utente

Quando si parla di profilazione utente è opportuno stabilire diversi livelli di astrazione per il trattamento dei dati. Ovviamente questa suddivisione presuppone anche protocolli di incapsulamento, o, quantomeno, di trasformazione da un livello di astrazione ad un altro.

Le rappresentazioni ai vari livelli sono fortemente collegati alla tipologia di applicazione che vanno a rappresentare. Nel caso di applicazioni multimediali in ambito distribuito, una possibile suddivisione proposta da [QOSSL] è la seguente:

1. Livello di utente
2. Livello di applicazione
3. Livello di risorse

Per quanto vedremo tale suddivisione, in realtà, può anche essere visto come una sorta di *hiding* (mascheramento occultamento) delle informazioni controllato, in modo da non potere inserire informazioni contraddittorie. Quello che si vuole dire è che le informazioni ai vari livelli possono essere detenute anche tutte in una unica struttura dati, a patto che tali informazioni non siano contraddittorie tra loro, e che poi ai vari livelli vengono presentate in maniera più o meno esplicita.

2.1.1 Livello di utente

Il livello di astrazione utente è quello sul quale l'utilizzatore deve poter agire direttamente, deve essere quindi di estrema semplicità e non deve consentire all'utente di inserire richieste contraddittorie. All'utente viene in genere presentata un'interfaccia in cui lui può scegliere, solo entro certi intervalli, la qualità del filmato, c'è poi un sistema di

trasformazione che traduce letteralmente, tale richiesta in una richiesta con certi parametri di negoziazione, al livello immediatamente sottostante in modo che tale richiesta risulti essere non contraddittoria.

2.1.2 Livello di applicazione

Questo è il livello dipendente dalla tipologia di applicazione, in questo punto possiamo specificare ad esempio il frame-rate desiderato e la risoluzione, voluta in un filmato. Ovviamente tali caratteristiche in parte sono dettate dalle richieste dell'utente, in parte dalla piattaforma sottostante; ad esempio questo significa che una richiesta di risoluzione massima a livello utente si traduce in una specifica risoluzione a livello di applicazione, riferita alla sottostante piattaforma.

2.1.3 Livello di risorse

In questo livello diventa necessario tradurre in termini di risorse macchina le risorse specificate a livello applicazione ad esempio fissato un certo frame-rate ed una certa risoluzione, e la profondità di colore, nonché la codifica usata per le immagini, dobbiamo "tradurre" tali informazioni in termini di cicli di CPU necessari, o di banda passante necessaria.

Quest'ultimo risulta essere quindi il livello più legato all'architettura hardware e software su cui gira l'applicazione. Inoltre il passaggio tra livello applicazione e livello utente non sempre è lineare e quantificabile in termini esatti. Il *mapping* dipende infatti, sia dall'architettura hardware, sia dai protocolli adottati per il trasferimento dei dati, oppure dall'implementazione di una data procedura di rendering. Per i motivi succitati spesso risulta impossibile a priori tale *mapping*. In tali casi vengono utilizzate tecniche di monitoraggio sia per adottare politiche di adattamento dinamico, sia per reperire informazioni sui dispositivi fisici, e sulle possibili implementazioni delle procedure che possono poi essere utilizzate come upper bound (cioè come massimi più o meno raggiungibili) per limitare le prestazioni attese al livello di applicazione.

2.2 Relazione tra Qualità del Servizio e Profilazione Utente

Anzitutto va specificato cosa di intende per Qualità del Servizio, o, più brevemente QoS, acronimo dell'espressione inglese Quality of Service: per QoS si intende una serie di specifiche per garantire l'erogazione di un servizio con certe caratteristiche e il suo mantenimento. Risulta evidente come il trattamento della QoS sia strettamente legato alla profilazione utente, infatti la profilazione utente risulta essere uno dei meccanismi attraverso i

quali la QoS può essere applicata.

Si osserva inoltre come, nella specifica della QoS, sia necessario stabilire anche il comportamento da adottare nel caso in cui le prestazioni degenerino. Infatti spesso, quando vengono negoziate le caratteristiche della QoS, viene anche stabilito un limite sul livello del calo delle prestazioni oltre il quale deve essere rinegoziata la specifica della QoS. Questo calo spesso può essere determinato da congestioni sul percorso di servizio in un ambiente in cui non sia disponibile un meccanismo per riservare risorse quali ad esempio la banda passante.

2.2.1 Linguaggi di Specifica per la QoS per applicazioni multimediali distribuite

La profilazione utente e la specifica della QoS sono due aspetti strettamente correlati in quanto in genere la profilazione è un processo che rappresenta una parte attiva nell'erogazione di un servizio con una fissata QoS. Quando un utente esprime delle preferenze circa il formato in cui vuole ricevere un filmato non fa' altro che esprimere delle specifiche sulla QoS desiderata, in pratica l'utente come specifica di QoS richiede che il servizio venga erogato in un certo formato. Analogamente, nel caso di profilazione sul dispositivo abbiamo delle regole per fissare le caratteristiche che deve avere il servizio erogato per venire incontro alle esigenze/capability del dispositivo, queste regole sono le specifiche di QoS, infatti in base alle caratteristiche che vogliamo ottenere, in termini di QoS, cambia la regola di selezione nell'insieme dei possibili profili compatibili all'erogazione. Detto questo è evidente come dare una specifica di QoS sia equivalente a dare una regola di trattamento della profilazione e le due cose possano essere descritte in modo equivalente.

2.2.2 Livelli nel trattamento della Quality of Service

Come abbiamo detto nel precedente paragrafo, nel caso di applicazioni multimediali, il trattamento dei parametri relativi alla profilazione utente, possono essere modellati su tre livelli; lo stesso accade per quello che riguarda la specifica della QoS per applicazioni multimediali in cui analogamente ancora distinguere tre livelli.

1. Livello utente: quando un utente richiede un video vuole poter specificare quale sia la "bontà" dell'immagine, a livello di percezione sensoriale.
2. Livello di applicazione: l'applicazione deve traslare le caratteristiche astratte, specificate a livello Utente, in caratteristiche concrete, a livello di applicazione. Questa traslazione, si assume fatta senza conoscenza del livello sottostante, in modo che risulti indipendente ad esempio dal sistema operativo, o della tipologia di rete usata.

3. Livello di risorse: infine è necessario mappare la richiesta, associata all'applicazione, in termini di risorse all'interno dell'ambiente in cui si svolge il servizio.

Come prima cosa va osservato come la struttura, a livelli consenta di avere una connessione di tipo ent-to-end tra entità allo stesso livello, cioè tra due entità che operano allo stesso livello di astrazione non c'è visibilità di quelle che sono le problematiche nei livelli sottostanti.

La QoS è stata, in passato, associata ad ambienti di rete, oggi si osserva una nuova necessita: specificare la QoS anche per quello che riguarda il livello di applicazione. Un evidente esempio di questa nuova necessità è la nascita di nuove applicazioni dette di tipo *Real-time* in cui viene richiesto un supporto, a livello di ambiente, per la specifica del tempo massimo di esecuzione di una procedura, o di un processo.

Spesso il livellamento induce una forma di mappaggio per cui una caratteristica può essere vista ad un livello di risorse ed essere mappata, magari 1-1, a un livello di applicazione; tuttavia come osservato precedentemente, non tutte le caratteristiche sono frutto di una traslazione diretta di caratteristiche, ma a volte sono caratteristiche specifiche del livello in esame.

2.2.3 Caratteristiche quantitative e qualitative e valutazione dei linguaggi per la specifica della QoS

Si noti che la QoS può essere specificata anche in base a caratteristiche *qualitative*, vale a dire legate alla percezione soggettiva di un generico utilizzatore del servizio. Vi sono cioè alcune caratteristiche che un utente percepisce come qualcosa di positivo ed un altro come negativo, ad esempio un utente che vuole ricevere un filmato attraverso una connessione potrebbe essere soddisfatto se il filmato viene ricevuto in fretta anche a scapito della qualità delle immagini, invece un altro potrebbe preferire avere un filmato con maggiore definizione anche se ciò richiede un tempo maggiore per ottenerlo. Questo piccolo esempio ci mostra come sia necessario utilizzare caratteristiche qualitative il più possibile oggettive, pur essendo, tali caratteristiche, per loro natura soggettive.

Le caratteristiche quantitative sono il risultato di una misura e quindi non possono essere assoggettate al giudizio dell'utente, esprimono *esattamente* la quantità di una qualche caratteristica misurabile.

A livello utente in genere vengono specificate solo caratteristiche qualitative, a livello applicazione e risorse in genere le caratteristiche che vengono specificate sono quantitative,

per evidenziare i livelli entro cui deve operare il sistema, mentre vengono usate caratteristiche qualitative per coordinare le entità coinvolte, come ad esempio per specificare come passare da un modo di operare ad un altro.

La seguente tabella (Figura 1.7) presa da [QOSSL] mostra quali siano le caratteristiche da prendere in considerazione ai tre livelli Utente, Applicazione, Risorse; mostra altresì come il livello applicazione debba considerarsi un livello indipendente dalla piattaforma di esecuzione, mentre il livello di risorsa è strettamente collegato ad esso.

Table 1. Summary of QoS issues for the three layers: user, application, and resource.

QoS layers	QoS issues
User (subjective criteria)	Perceptive media quality (excellent, good, fair, bad) Window size (big, medium, small) Pricing model (flat rate, per transmitted byte charge) Range of price (high, medium, low)
Application (hardware- and platform independent)	Quantitative issues (video frame rate, image/audio resolution) Qualitative issues (inter/intrastream synchronization schemes) Adaptation rules (if video quality is good, then drop all B frames)
Resource (hardware- and platform-dependent)	Quantitative issues (throughput, delay, delay jitter, memory size, timing of resource requirements) Qualitative issues (OS scheduling, reservation style, loss detection/recovery mechanisms) Adaptation rules (for example, if 80ms < skew < 160ms, then drop *x* video frames)

Figura 2.1 Tabella delle caratteristiche per i tre livelli della QoS.

Le proprietà esposte ai vari livelli di specifica sono differenti quindi per una valutazione della qualità dei vari linguaggi di specifica della QoS ai vari livelli vanno utilizzati criteri differenti:

- Valutazione del Livello Utente:

  A livello Utente ci si aspetta di avere a che fare con un utilizzatore non esperto, quindi è necessario nascondere molte caratteristiche implementative, e rendere tuttavia disponibile una interfaccia, per la specifica della QoS, che consenta all'utente, in modo semplice, la scelta quelle che lui ritiene le caratteristiche desiderate, ma che metta altresì a disposizione una interfaccia espressiva dove, per espressiva si intende il fatto che deve evidenziare il costo delle scelte effettuate. In definitiva l'interfaccia per l'utente deve essere, per usare una espressione inglese"user

friedly", vale a dire di facile uso per l'utente, e ciò si ottiene fornendo all'utilizzatore un insieme con un numero ridotto di proprietà configurabili ma che coprano tutte le caratteristiche qualitative desiderabili sull'applicazione.

- Valutazione del Livello Applicazione:

  A livello Applicazione la specifica delle caratteristiche legate alla QoS viene fatta mediante linguaggi di specifica generici, quindi la valutazione del linguaggio di specifica può essere basata sulle seguenti caratteristiche:

  - Espressività: un buon linguaggio di specifica della QoS deve essere in grado di specificare un ampia gamma di caratteristiche, regole di trattamento delle stesse, e relative risorse da occupare; in pratica il linguaggio usato deve essere ricco di costrutti e deve dare un accesso completo alle proprietà relative all'applicazione.
  - Dichiarativitá: la dichiarazione di una specifica di QoS può essere, per sua natura, dichiarativa, vale a dire che non si vuole specificare come ottenere le caratteristiche volute, ma semplicemente quali caratteristiche si vogliono ottenere.
  - Indipendenza: la specifica deve essere indipendente dagli aspetti funzionali del codice sottostante e deve essere facilmente leggibile e modificabile per lo sviluppatore.
  - Estensibilità: questa qualità consente di valutare quanto facilmente il linguaggio possa essere esteso per esprimere nuove caratteristiche, dimensioni nella specifica della QoS .
  - Riusabilità: questa caratteristica è particolarmente utile quando la specifica può diventare molto ampia, in tale caso parte di alcune specifiche possono essere riutilizzate in altre. Parimenti a qualunque linguaggio di programmazione la riusabilità si ottiene mediante l' indipendenza e la separazione di quelle che sono le caratteristiche funzionali da quelle non funzionali, del linguaggio di specifica.

- Valutazione del Livello Risorse: a questo livello si specificano le risorse da allocare sul sistema, quindi il criterio più consono per valutare un linguaggio di specifica della QoS è l'*espressività,* vale a dire la quantità di risorse specificabili e il dettaglio con cui possono essere specificate.

2.2.4 Specifica a livello utente

A livello utente è preferibile avere una interfaccia grafica dal design semplice, per la specifica della QoS, che metta a disposizione dell'utente, non una miriade di caratteristiche,

ma un numero limitato, concentrato sulle caratteristiche qualitative che vengono percepite come importanti per l'utente.

La specifica a questo livello deve soddisfare due condizioni:

1. deve fornire una descrizione qualitativa delle proprietà multimediali
2. deve fornire informazioni sul costo, in termini tipologia di risorse occupate, necessarie per l'erogazione del servizio. Questa è una importante caratteristica spesso trascurata, infatti se non mettessimo in evidenza il costo relativo ad una certa qualità fornita, risulta ovvio che qualunque utente sceglierebbe le massime caratteristiche per l'applicazione multimediale. Un esempio di costo, relativo a un filmato, da associare, più o meno direttamente, alla dimensione dell'immagine e alla banda disponibile, potrebbe essere il tempo necessario per lo scaricamento, e di conseguenza per la visualizzazione dello stesso.

2.2.5 Specifica a livello applicazione

A livello di applicazione esistono due tipi di caratteristiche specificabili: relative alle performance e relative al comportamento. Le prime consentono di specificare le caratteristiche quantitative relative alla QoS come ad esempio il frame-rate, mentre le seconde quelle qualitative come ad esempio il comportamento da adottare nel caso in cui scarseggi la banda passante. Il linguaggio di specifica può mettere a disposizione certe astrazioni, per cui lo sviluppatore non deve preoccuparsi di come queste vengono trattate a basso livelli. Queste astrazioni possono essere messe a disposizione nel linguaggio sotto forma di API (interfacce di accesso a delle librerie aggiuntive) o sotto forma di estensioni o strutture del linguaggio stesso; ad esempio in linguaggio Java viene messo a disposizione il concetto di *Thread* come API mentre per quel che riguarda l'accesso esclusivo alle risorse, viene fornito il costrutto sintattico *synchronized* a livello di linguaggio.

Il linguaggio di specifica della QoS a livello di applicazione può essere discusso sulla base del paradigma adottato dal linguaggio stesso. Di seguito vengono presentati sette possibili paradigmi di programmazione utilizzabili per la specifica della QoS in ambito multimediale ripresi da [QOSSL].

1. Paradigma basato sugli Script: tale paradigma consente la specifica delle caratteristiche mediante una entità detta genericamente ScriptManager che si interpone sul flusso dei multimediale e, istruita tramite un linguaggio di scripting,in genere imperativo, agisce direttamente sul flusso, ad esempio marcandone i pacchetti per influenzarne la priorità.

Questo approccio in genere è poco espressivo, molto legato all'architettura, tuttavia consente di avere completa separazione tra applicazione e politiche.

2. Paradigma basato sui Parametri: questa tecnica, molto usata, consente la specifica mediante la specifica di parametri in una sorta di tabella. I parametri possono essere raggruppati in diverse categorie, inerenti alle diverse caratteristiche macroscopiche che vanno a definire. Possiamo avere gruppi di parametri per la specifica: del flusso, delle caratteristiche statistiche volute, delle politiche di adattamento, di come deve essere mantenuto un certo servizio,di come riservare risorse, di quale costo siamo disposti a pagare, in termini di allocazione di risorse, per aver un certo servizio.
3. Paradigma orientato ai Processi: in questa ottica i processi, intesi come unità di esecuzione, comunicano tra loro tramite scambio di messaggi, o canali di comunicazione, e la specifica della QoS consente di frapporci su questi canali e specificare i vincoli, negoziare la qualità voluta, monitorare l'esecuzione. Questa tecnica tuttavia risulta difficile da manutenere in quanto la politica è sparsa tra i blocchi funzionali, i processi.
4. Paradigma logico: i sistemi adattativi in genere adottano approcci basati sul controllo logico delle politiche di adattamento e di specifica del flusso. Alcuni sistemi usano una tecnica proporzionale-integro-derivativa per specificare con estrema precisione il comportamento quando si controllano flussi o processi. Altri sistemi utilizzano logiche di tipo fuzzy; questa tecnica comprende due componenti: l'adattatore e il configuratore. L'adattatore genera le decisioni di controllo con conoscenza globale della situazione, il configuratore usa la specifica fuzzy e traduce le decisioni prese dall'adattatore in valori concreti di parametri da usare per le azioni da intraprendere durante l'erogazione del servizio. Questo approccio consente di scrivere regole in cui si ha conoscenza della QoS attuale; le regole sono specificate nel formato *if-then* ("*se-allora*"). Un esempio di regola può essere: "“if cpu_availability is very_high and available_bandwidth is very_low, then rate_demand is compress"; un limite della logica fuzzy qui prefigurata è che si possono specificare solo azioni e non altri parametri.
5. Paradigma basato sui linguaggi di markup: il linguaggio di markup per eccellenza è XML e può essere usato per contenere sia i dati che le informazioni sugli stessi. I tag introdotti possono essere interpretati e tradotti in regole, in modo molto simile a quanto avviene in un linguaggio dichiarativo. Possiamo avere una parte della struttura, che

rappresenta ad esempio la condizione, la QoS richiesta, ed una parte che dichiara le azioni da svolgere in tale caso. Un linguaggio siffatto risulta essere molto buono in termini di dichiaratività, espressività ed indipendenza, tuttavia poco si presta al riuso, ed all'estensione.

6. Paradigma basato sull'aspetto: in tale paradigma gli "aspetti" sono le caratteristiche che influenzano sistematicamente le performance o rappresentano la semantica dei componenti. Questa tecnica è basata su un linguaggio di specifica, come può essere il linguaggio IDL di specifica delle interfaccia in CORBA. L'interfaccia viene implementata anche in differenti linguaggi. Il vantaggio è nel fatto che posso decomporre il problema in "aspetti" che vengono poi realizzati nel più appropriato linguaggio e miscelati e riuniti i una unica applicazione da uno speciale componente che usando le interfacce, unisce le diverse parti. Così facendo si evita di mescolare, caratteristiche funzionali, legate al linguaggio, con quelle relative all'applicazione, e si evita di avere le politiche sparpagliate su diversi linguaggi di programmazione in modo disordinato, inoltre consente di usare il linguaggio, a basso livello, più appropriato per realizzare ciascuno degli "aspetti" necessari.
7. Paradigma orientato agli oggetti: questo paradigma prende il nome dalla analoga tipologia di linguaggi di programmazione, perché il processo di raffinamento nella specifica della QoS può essere trattata in modo simile all'ereditarietà nei linguaggi orientati agli oggetti. Un approccio del genere può essere visto in QML (QoS Modeling Language sviluppato da HP Laboratories) in cui viene specificando un "Tipo di contratto" in cui si specificano le caratteristiche in gioco, i parametri da tenere sotto controllo poi sono previsti due tipi di raffinamento: il "contratto" e il "profilo"; col "contratto" si impongono vincoli entro cui devono restare i parametri dichiarati nel "tipo di contratto", mentre nel "profilo" specifica le azioni da compiere quando avvengono determinate condizioni sui parametri relativi al "tipo di contratto". Tra i linguaggi di specifica a livello applicazione QML risulta riusabile, attraverso "tipo di contratto" e "profilo", estendibile, indipendente, tuttavia il limite di QML è che per ogni "profilo" può essere usato un solo "tipo di contratto" e non due istanza dello stesso "tipo di contratto" nello stesso "profilo".

2.2.6 Specifica a livello risorse

A livello di risorse la caratteristica essenziale mediante la quale possiamo dare un

giudizio è l'espressività del linguaggio. Possiamo distinguere due categorie di linguaggi:

1. linguaggi con granularità grossa : in questo caso ci si aspetta una descrizione di metalivello; questo tipo di descrizione non permette la precisa definizione delle risorse da allocare. Ad esempio può specificare quante risorse siano necessarie, ma non quando allocarle.
2. linguaggi con granularità fine: consente una precisa descrizione delle risorse da allocare; permette una definizione qualitativa e quantitativa delle risorse necessarie per realizzare la QoS nonché la specifica dei tempi e delle regole di adattamento.

2.3 Media Features Tag

Il sistema dei *media features tag* è stato forse il primo serio approccio alla profilazione utente, in particolare per l'adattamento del contenuto. Questo sistema è stato sviluppato dal CONNEG (Content Negotiation Working Group) un gruppo di lavoro all'interno dell' IETF (Internet Engineering Task Force) che per primo si è seriamente posto il problema della standardizzazione di una sintassi per esprimere le caratteristiche dei dispositivi, articolata in diverse parti che andremo ad enunciare.

2.3.1 Procedura di registrazione

La procedura di registrazione descritta in [RFC2506] dà la specifica del processo di registrazione di quelli che vengono detti *media features tag* in un sistema di nomi riconosciuto a livello mondiale; questi ultimi altro non sono che i nomi delle caratteristiche descrittive. Inoltre in [RFC2506] si introduce la sintassi utilizzata per la definizione.

Il sistema presentato dal CONNEG è fondamentalmente composto da insiemi di nomi di caratteristiche relative associate ai contenuti che i dispositivi multimediali trattano, in pratica un dispositivo viene descritto in base ai formati video supportati, o ai formati di immagini supportate. Tutto il sistema è composto da nomi, valori, e relazioni introdotte tra questi.

I nomi degli attributi sono detti *feature-tag*, i valori possibili specificati in [RFC2506] sono istanze dei seguenti tipi: interi senza segno, razionali (una sorta di rapporto), token (sequenza di caratteri) con relazione di ordine oppure con relazione di uguaglianza, stringhe (sequenza di caratteri tra doppi apici) con relazione di ordine oppure di uguaglianza.

Al momento della registrazione ogni *feature-tag* va associato a un singolo tipo. Non sono previsti insiemi di valori, vale a dire che non posso avere un *feature-tag* associato a un insieme di valori o a una sequenza.

Durante la registrazione è possibile, contestualmente, richiedere l'associazione dei *feature-tag* a un insieme di elementi nel sistema ASN (vedi sigla ASN). La registrazione avviene all'interno di quello che viene definito un albero di registrazione (*registration tree*) in modo che i *feature-tag* abbiano nomi completi a cui riferirsi globalmente nella forma "**tree.feature-name**". Infine viene proposto un modulo standard per richiedere la registrazione di un insieme di *feature-tag* in cui vengono presentate parti da compilare e istruzioni su come compilarlo. L'attenzione è posta sui nomi dei *feature-tag* il tipo e le relazioni di ordine tra i valori di questi ultimi.

2.3.2 Sintassi per la descrizione dei Media Features Tag

Nella sintassi descritta in [RFC2533] emerge subito che le descrizioni dei dispositivi vengono fatte in termini di coppie *feature-tag* e valore associato, inoltre tali valori devono essere atomici e confrontabili. Quindi i *feature-tag* descrivono le capability del dispositivo in termini dei valori delle caratteristiche che devono avere i documenti che questo deve manipolare.

Nella sintassi che viene presentata all'interno di [RFC2533] in particolare vengono definiti i seguenti termini:

- *feature-collection*: un insieme di diversi nomi di attributi con i valori associati; può essere visto come la descrizione di uno specifico documento.
- *feature-set*: un insieme di zero o più *feature-collection*.
- *feature-set-predicate*: una funzione su una *feature-collection* che restituisce un valore booleano *vero* se l'insieme soddisfa le caratteristiche espresse nel *predicato*.

Si assume che i valori dei *feature-tag* siano semplici valori atomici di tipo booleano, enumerativo, stringhe di testo, oppure numerici. Tutti questi tipi di valore hanno la caratteristica di poter essere messi in relazione di ordine (relazioni maggiore, minore, uguale) oppure possono essere comparati (relazione uguale o diverso) ed è questo che viene fatto nei *predicati*.

Ogni singolo *feature-tag* può essere pensato come un componente di una *feature-collection* che descrive una certa caratteristica di una certa istanza di una risorsa. Un *feature-set* descrive un insieme di possibili istanze per le caratteristiche evidenziate nelle *feature-collection* che lo compongono.

L'approccio da cui si parte è il seguente, si assume che un *feature-set* associato a una categoria di documenti sia inizialmente non vincolato, nel senso che consenta di descrivere

tutte le possibili istanze di un certo documento, cioè le *feature-collection*; poi entra i gioco il *feature-set-predicate* questo rappresenta un insieme di vincoli che applicati al *feature-set* consente la rimozione delle istanze non valide. Il meccanismo usato per rimuovere queste istanze è l'idea matematica di *relazione* ad esempio una funzione booleana che riceve in ingresso le istanze e applicata a queste restituisce un valore *vero* o *falso* per cui se il risultato è *vero* l'istanza del documento appartiene all'insieme delle *feature-collection* valide altrimenti no e viene scartata. Il risultato dell'applicazione di questo predicato è un piccolo insieme di *feature-collection* che rappresenta i documenti manipolabili dal dispositivo descritto dal *predicato*. In sostanza le *feature-collection* del *feature-set* filtrato dal *feature-set-predicate* rappresentano i formati di documenti validi inviabili al dispositivo.

Ogni vincolo espresso nel predicato è rappresentato da un insieme di vincoli di tipo relazionale o di confronto sui singoli *feature-tag* messi insieme con predicati di *and*, *or* e *not*.

Presentiamo di seguito un esempio in cui abbiamo una risorsa che può essere fornita in uno dei seguenti formati:

- 750x500 pixel con profondità di colore pari a 15 colori (4bit)
- 150dpi su foglio A4

che deve essere renderizzata su un dispositivo con le seguenti caratteristiche

- su schermo
  - risoluzione 640x480 pixel e 16 milioni di colori (24bit per pixel)
  - risoluzione 800x600 pixel e 65 mila di colori (16bit per pixel)
  - risoluzione 1024x768 pixel e 256 colori (8bit per pixel)
- su stampante
  - con risoluzione pari a 300dpi e foglio A4

Il risultato dell'applicazione dei predicati associati al dispositivo portano a un *feature-set* finale pari a:

renderizzazione su schermo a 750x500 pixel e 15 colori oppure

stampa con risoluzione pari a 300dpi e formato di foglio A4

Questo viene mostrato nella seguente tabella dai predicati espressi nella forma indicata dal CONNEG:

Opzioni risorsa	`(| (& (pix-x=750) (pix-y=500) (color=15) )` `(& (dpi>=150) (papersize=iso-A4) ) )`

Capacità del dispositivo ricevente	`(| (& (| (& (pix-x<=640) (pix-y<=480)(color<=16777216) )`
	`(& (pix-x<=800) (pix-y<=600) (color<=65535) )`
	`(& (pix-x<=1024) (pix-y<=768) (color<=256) ) )`
	`(ua-media=screen) )`
	`(& (dpi=300)`
	`(ua-media=stationery)`
	`(papersize=iso-A4) ) )`

formati finali selezionati	`(| (& (pix-x=750) (pix-y=500) (color=15) )` `(& (dpi=300) (ua-media=stationery) (papersize=iso-A4)))`

Come si può notare la sintassi è molto simile al linguaggio *lisp* in cui abbiamo i predicati di *and* e *or* rappresentati rispettivamente da "&" e da "|" inoltre notiamo come tutto viene espresso in modo omogeneo cioè le caratteristiche, le capacità, i formati selezionati, da osservare come i *feature-set-predicate* svolgono il duplice ruolo di profilo e di filtro.

Oltre a questo nella sintassi presentata viene aggiunta la possibilità di esprimere una caratteristica aggiuntiva *quality* associabile a ciascun predicato con un valore reale compreso tra 0 e 1 che esprima la preferenza verso un certo profilo anche se poi no viene realmente impiegata nell'algoritmo proposto; viene tuttavia evidenziato come questa caratteristica possa essere utilizzata per ordinare le possibili uscite dell'algoritmo di selezione usato per arrivare all'insieme delle *feature-set* associate al dispositivo.

Al termine del documento [RFC2533] viene mostrata la possibilità di introdurre dei *named-predicate* vale a dire dei predicati dotati di nome ed utilizzabili a tutti gli effetti come se fossero delle *feature-tag* con parametri; questa astrazione è molto simile al concetto di funzione nei linguaggi di programmazione, tuttavia il meccanismo per la risoluzione di questi predicati è di "sostituzione in linea prima di attuare l'algoritmo di selezione" (molto simile a quanto avviene per le macro in linguaggio C), cosa che vieta l'uso della ricorsione nella definizione di questi predicati. Infine vengono poste alcune questioni riguardo la sicurezza di questo sistema.

2.3.3 Algoritmo proposto per i Media Features Tag

Sempre in [RFC2533] viene proposto un algoritmo per la selezione del profilo da adottare, dato un insieme di caratteristiche possibili e un insieme di caratteristiche accettate dal dispositivo client.

Anzitutto va fatta una precisazione: in genere la descrizione del client avviene in termini

di *feature-set-predicate* mentre quella dei contenuti avviene in termini di *feature-set* oppure in termini di *feature-set-predicate*; nel primo caso è sufficiente inserire i vari *feature-collection* che compongono il *feature-set* nei vari predicati e vedere se vengono verificati. Nel secondo caso è necessario attuare un algoritmo di matching in modo da ottenere un *feature-set* finale che rappresenti le soluzioni accettabili.

Si introduce ufficialmente la notazione per i predicati come funzioni logiche *and* e *or* applicate a relazioni (minore,maggiore eccetera) che coinvolgono i *feature-tag* come se fossero variabili e valori. Vengono introdotte alcune notazioni *zucchero-sintattiche*. Infine va osservato come la forma normale disgiuntiva (*or* di *and*) raggiungibile da un *feature-set-predicate* iniziale, detta *espressione canonica, sia equivalente* con la forma clausole nei sistemi con logica del secondo ordine, utilizzata ad esempio nel linguaggio prolog per la risoluzione dei predicati. L'algoritmo finale si compone dei seguenti passi:

1. Formulazione dell'obbiettivo. In genere consiste nello scrivere una espressione come la seguente: "(& P Q)" dove P e Q sono i *feature-set-predicate* associati alla sorgente del documento e alle caratteristiche supportate dal ricevente.
2. Rimpiazzare le espressioni di *zucchero-sintattiche* di cui accennato con le relative relazioni. Ottengo quindi una forma che contiene solo relazioni (uguale, maggiore, minore etc.) ed espressioni booleane di tipo *and, or* e *not*.
3. Muovere le negazioni verso l'interno mediante le seguenti regole di trasformazione generalizzazione dalle leggi di De Morgan e la regola della doppia negazione:

```
(! (& A1 A2 ... Am ) ) --> (| (! A1 ) (! A2 ) ... (! Am ) )
(! (| A1 A2 ... Am ) ) --> (& (! A1 ) (! A2 ) ... (! Am ) )
(! (! A ) ) --> A
```

4. Rimpiazzare le negazioni con le equivalenti comparazioni come nel seguente esempio:

```
(A = B) & (B = C) => (A = C)
(A <= B) & (B <= C) => (A <= C)
```

   In questa fase vogliamo giungere ad una forma finale in cui vi siano un insieme di relazioni sufficientemente ridotto da poter essere trattato sistematicamente; nel sistema descritto si arriva ad una forma finale in cui le uniche relazioni presenti sono *and* e *or* logico e quattro tipi di relazioni indicate con LE,GE,NL,NG che rappresentano rispettivamente le relazioni *minore o uguale*,*maggiore o uguale* e le relative negazioni.
5. Passaggio alla forma canonica; da osservare che le negazioni sono state eliminate al passo precedente. In questo passaggio vengono espanse le disgiunzioni e raggruppate le congiunzioni come nei seguenti esempi in cui abbiamo indicato con i termini

A1,A2,...An ... le relazioni ottenute al passo precedente

```
(& (| A1 A2 ... Am ) B1 B2 ... Bn )
--> (| (& A1 B1 B2 ... Bn )
       (& A2 B1 B2 ... Bn )
        :
       (& Am B1 B2 ... Bn ) )
(& (& A1 A2 ... Am ) B1 B2 ... Bn )
  --> (& A1 A2 ... Am B1 B2 ... Bn )
(| (| A1 A2 ... Am ) B1 B2 ... Bn )
  --> (| A1 A2 ... Am B1 B2 ... Bn )
```

6. Vengono poi raggruppati le congiunzioni in blocchi comuni cioè quelli che coinvolgono *feature-tag* comuni vengono messe vicine, un esempio può essere il seguente in cui per brevità sono state mantenute le relazioni di uguaglianza

```
(| (& (dpi=200) (dpi=300) (grey=2) (grey=2) (color=0)
      (image-coding=MH) (image-coding=MR) )
   (& (dpi=200) (dpi=300) (grey=2) (grey=2) (color=0)
      (image-coding=MR) (image-coding=MR) )
)
```

Infine vengono rimosse le congiunzioni che risultano in contraddizione, dell'esempio precedente rimane il seguente essendo in contraddizione il *feture-tag* "image-coding"

```
(| (& (dpi=300) (grey=2) (color=0) (image-coding=MR) ) )
```

Quindi al passo finale abbiamo un *feature-set* che rappresenta le possibili istanze di documenti validi per l'erogazione del servizio come voci della disgiunzione, oppure un insieme vuoto se non è possibile fornire un servizio adattato.

2.4 Conclusioni sull'uso dei Media Features Tag

Il sistema proposto dal CONNEG ha il vantaggio dare un insieme di nomi, vocabolario, riconosciuto globalmente di cui un esempio è quello definito in [RFC2531], inoltre propone un meccanismo sistematico e quindi automatizzabile per la gestione della risoluzione dei profili. Infine la descrizione data di un dispositivo in termini dei documenti accettati rende più semplice la definizione del meccanismo di risoluzione e più omogenea la descrizione complessiva del sistema composto dal dispositivo e dalla risorsa.

L'ultimo punto enunciato è tuttavia un punto di forza ma anche di debolezza in quanto non viene esplicitamente descritto un dispositivo quanto piuttosto un insieme di documenti. Altro punto di debolezza sta nel trattare i *feature-tag* come entità atomiche, questo implica che una caratteristica è vincolata ad assumere un valore piuttosto che un insieme, manca in pratica un sistema per esprimere caratteristiche di tipo insiemistico. Inoltre le relazioni introdotte per trattare i *feature-tag* da un lato facilitano la definizione dell'algoritmo di risoluzione ma dall'altro vincolano i valori dei *feature-tag* ad essere sempre entità

confrontabili, cosa non sempre realizzabile, pensiamo se ad esempio volessimo esprimere delle percentuali di un certo valore, queste percentuali non possono essere confrontate, ma vanno comparai i valori risultanti dell'applicazione delle stesse.

In definitiva il meccanismo dei *feature-tag* risulta formalmente ben specificato tuttavia in alcune situazioni può risultare inadeguato o troppo vincolato per esprimere caratteristiche e relazioni delle entità coinvolte.

2.5 MPEG

Oggi i dati di tipo multimediale vengono trasmessi sempre più in forma digitale, questo significa da un lato avere informazioni maggiormente tolleranti agli errori, dall'altro un problema dovuto alla elevata bit-rate necessaria per trasmettere/memorizzare questo tipo di segnali rispetto ai canali/supporti utilizzati nella trasmissione/memorizzazione. Questo ha fatto sorgere la necessità di algoritmi per la compressione da utilizzare per le applicazioni di tipo audiovisivo con e senza perdita di informazione. Il recente sviluppo tecnologico ha reso poi possibile l'utilizzo di tecniche di compressione audio e video, così è nata la nuova necessità di standardizzare queste ultime; infatti solo attraverso la standardizzazione si possono ridurre i costi dei dispositivi di compressione, e quelli relativi all'interfacciamento tra questi in modo da favorire la diffusione di applicazioni di questo genere. Inoltre la definizione di un efficace algoritmo di compressione riduce notevolmente la mole di dati scambiati rendendo possibile una migliore gestione dei mezzi trasmissivi e di memorizzazione.

Per queste motivazioni l' ISO (International Organization for Standardization) ha creato il Moving Pictures Experts Group in breve MPEG.

Le attività iniziali dell'MPEG furono la definizione di sistemi di compressione video e audio nonché la sincronizzazione tra questi.

L' MPEG ha definito o lavora sui seguenti standard:

- MPEG-1: progettato per la codifica di immagini in movimento e per l'audio ad esse associato, in forma digitale, con un bit-rate che arriva fino a circa 1.5 Mbit/s.
- MPEG-2: progettato per la codifica generica di immagini in movimento e audio ad esse associato.
- MPEG-3: confluito in MPEG-2
- MPEG-4: progettato per la codifica audiovisiva a bassissimo bit-rate.
- MPEG-7: definisce un sistema per la descrizione e interpretazione delle informazioni compresse all'interno dei filmati.

- MPEG-21: definisce una generica architettura per la descrizione dei dispositivi e dei filmati con lo scopo dell'adattamento del contenuto,questo verrà trattato più dettagliatamente nei prossimi paragrafi.

2.5.1 MPEG-7

Si fa' accenno a questo standard in quanto costituisce una forma di descrizione per i contenuti multimediali e quindi rappresenta la parte di descrizione necessaria dal punto di vista di chi fornisce un servizio di tipo multimediale per poi eseguire l'adattamento, tanto più che in parte viene ripreso in MPEG-21.

MPEG-7 detto "Multimedia Content Description Interface" fornisce un sistema standardizzato di tecnologie di base per consentire la descrizione di contenuti audiovisivi in ambito multimediale. Estende le limitate soluzioni proprietarie nell'identificare il contenuto con l'introduzione di una più vasta gamma di strutture e tipi di dato.

Le parti di cui si compone lo standard MPEG7 sono le seguenti:

- MPEG-7 Systems: sono gli strumenti necessari per preparare le descrizioni MPEG7 per un efficiente trasporto e memorizzazione e l'architettura del terminale.
- MPEG-7 Description Definition Language: il linguaggio per definire la sintassi del successivo Description Tool, usato per definire i Description Schema, strutture dati per annotare le descrizioni dei contenuti multimediali.
- MPEG-7 Visual : il Description Tool cioè gli strumenti di descrizione per le descrizioni (solo) delle parti visive.
- MPEG-7 Audio: il Description Tool per le parti audio.
- MPEG-7 Multimedia Description Schemes: il Description Tool utilizzato per descrivere le caratteristiche generiche dei contenuti multimediali.
- MPEG-7 Reference Software: software che implementa rilevanti parti di MPEG7 in stato normativo.
- MPEG-7 Conformance Testing: linee guida e procedure per testare la conformità dell'implementazione di MPEG7.
- MPEG-7 Extraction and use of descriptions: materiale informativo sotto forma di rapporto tecnico sull'estrazione e uso dei Description Tool.

2.5.2 MPEG-21

Basato su precedenti osservazioni MPEG-21 mira a definire una architettura aperta per la distribuzione ed il consumo di contenuti multimediali. MPEG-21 è basato su due concetti

essenziali: la definizione di una fondamentale unità per la distribuzione, il *digital item*, e il concetto di *utenti* che interagiscono con i *digital item*. Il *digital item* rappresenta il "cosa" della struttura per la distribuzione multimediale mentre gli *utenti* rappresentano il "chi" della struttura. L'obbiettivo di MPEG-21 può così essere riscritto come "definire una tecnologia necessaria a consentire agli *utenti* l'accesso, la manipolazione, e la fruizione dei *digital item* in modo efficiente, trasparente ed interoperabile". MPEG-21 identifica e definisce i meccanismi e gli elementi necessari al supporto nella catena di distribuzione di contenuti multimediali.

Modello Utente

Per modello utente in MPEG-21 si intende il fatto che chiunque manipola i *digital item,* dal fornitore del servizio, al fruitore dello stesso ed anche lo stesso autore, vengono considerati come utenti del suddetto oggetto.

Presentazione dei *digital item*

Vi sono molti tipi di contenuti multimediali e molti modi per rappresentarli, è quindi necessaria la definizione di un modo formale per descrivere questi ultimi in modo oggettivo.

Le parti di MPEG-21

MPEG-21 si articola in undici parti:

1. Vision, technologies, and strategy: descrive, in modo generale, la struttura del sistema multimediale, i suoi elementi architetturali e i requisiti funzionali.
2. Digital Item Declaration (DID): fornisce uno schema astratto, flessibile ed interoperabile per la dichiarazione dei *digital item*.
3. Digital Item Identification (DII): definisce il sistema per identificare ogni entità in base alla sua natura, al sua tipologia o granularità.
4. Intellectual property management and protection (IPMP): fornisce un modo sicuro per gestire e proteggere i contenuti attraverso la rete.
5. Rights Expression Language (REL): specifica un linguaggio comprensibile alle macchine per dichiarare i permessi e diritti usando i termini definiti in Rights Data Dictionary.
6. Rights Data Dictionary (RDD): specifica un dizionario di parole chiave necessarie a descrivere i diritti utente.
7. Digital Item Adaptation (DIA): definisce gli strumenti di descrizione per l'uso di un ambiente e di caratteristiche del formato del contenuto che influenzano l'accesso

trasparente alle risorse; in genere definisce gli strumenti per la descrizione di terminali, reti,utenti e l'ambiente naturale in cui questi sono collocati e uno modello generale per il trattamento dell'adattamento.

8. Reference software: include i software che implementano gli strumenti specificati in MPEG-21.
9. File format: definisce il formato per i file usati per distribuire i *digital item*.
10. Digital Item Processing (DIP): definisce meccanismi per processare i *digital item* in modo interoperabile e standard.
11. Evaluation methods for persistent association technologies: documenta i migliori modi per valutare le tecnologie attuali usando una metodologia comune piuttosto che standardizzare le stesse.

Andiamo di seguito a descrivere quelle che sono le parti fondamentali di MPEG-21

2.5.2.1 Vision, technologies, and strategy

Il titolo di questa parte è stato scelto per riflettere il fondamentale proposito che questo rappresenta. Questo viene fatto per:

- Definire una "visione" del sistema multimediale per consentire un modo trasparente e "più ricco" di possibilità nell'uso delle risorse attraverso la rete.
- Puntare all'integrazione attraverso le reti per facilitare l'armonizzazione delle "tecnologie" usate per creare, manipolare, trasportare, distribuire e utilizzare i contenuti multimediali.
- Definire una "strategia" per ottenere una infrastruttura dallo sviluppo delle specifiche di standard basati su "ben definite" necessità funzionali, tutto attraverso la collaborazione di diverse entità.

2.5.2.2 Digital Item Declaration (DID)

Il proposito di questa sezione è di descrivere un insieme di termini astratti e di concetti per creare un modello base per definire i *digital item*.

Questa sezione si articola in tre parti:

1. Modello: descrive un insieme di termini astratti e concetti di base per definire i *digital item.*
2. Rappresentazione: descrizione della sintassi e della semantica di ognuno degli elementi introdotti e rappresentati in XML.
3. Schema: schema XML che comprende l'intera grammatica introdotta in questa sezione

per descrivere i *digital item*.

Gli elementi introdotti in DID

Di seguito vengono presentati in rassegna gli elementi introdotti in DID e una descrizione degli stessi, per correttezza come peraltro gi fatto in precedenza i termini vengono presentati in lingua inglese.

1. *Container*: è una struttura che consente di raggruppare *digital item* e/o *container* stessi. Da osservare che un *container* non è un *item*.
2. *Item*: è un raggruppamento di sub-*item* e/o *componenti* che viene identificato da un *descriptor*.
3. *Component*: è il legame tra una *resource* e il suo *descriptor*.
4. *Anchor*: collega una *descriptor* a un *fragment* che corrisponde allo specifico intervallo in una risorsa.
5. *Descriptor*: associa informazioni all'elemento che lo include
6. *Condition*: descrive l'elemento che racchiude come opzionale e lo collega alla *selection* che coinvolge quest'ultimo. Più *predicate* dentro una *condition* sono combinati come congiunzione logica (*and*). Un *predicate* può essere negato in una *condition*. Più *condition* associate a un certo elemento sono combinate come disgiunzione logica (*or*).
7. *Choice*: racchiude un insieme di *selection* che influenzano la configurazione di un *item*.
8. *Selection*: descrive una decisione che ha effetto su una o più *condition* definita in qualche *item.*
9. *Annotation*: mostra un insieme di informazioni circa altri elementi del modello senza alterare la descrizione dell'elemento in cui si trova. Prende la forma di *assertion*, *descriptor* o *anchor*.
10. *Assertion*: definisce in modo completo o parziale lo stato di una *choice* asserendo *vero*, *falso* o *indecidibile* per un certo numero di *predicate* associati alla *selection* per la *choice*.
11. *Resource*: è una risorsa ben identificata come un filmato o una traccia audio.
12. *Fragment*: designa, in modo non ambiguo, una parte di una *resource*.
13. *Statement*: un valore testuale che contiene informazioni ma non *resource*.
14. *Predicate*: è una dichiarazione identificabile e non ambigua che può essere *vera*, *falsa* o *indecidibile*.

Relazione tra i principali elementi definiti in DID

La figura seguente mostra, come empio, i principali elementi presentati in Digital Item Declaration e le relazioni tra questi in termini di inclusione.

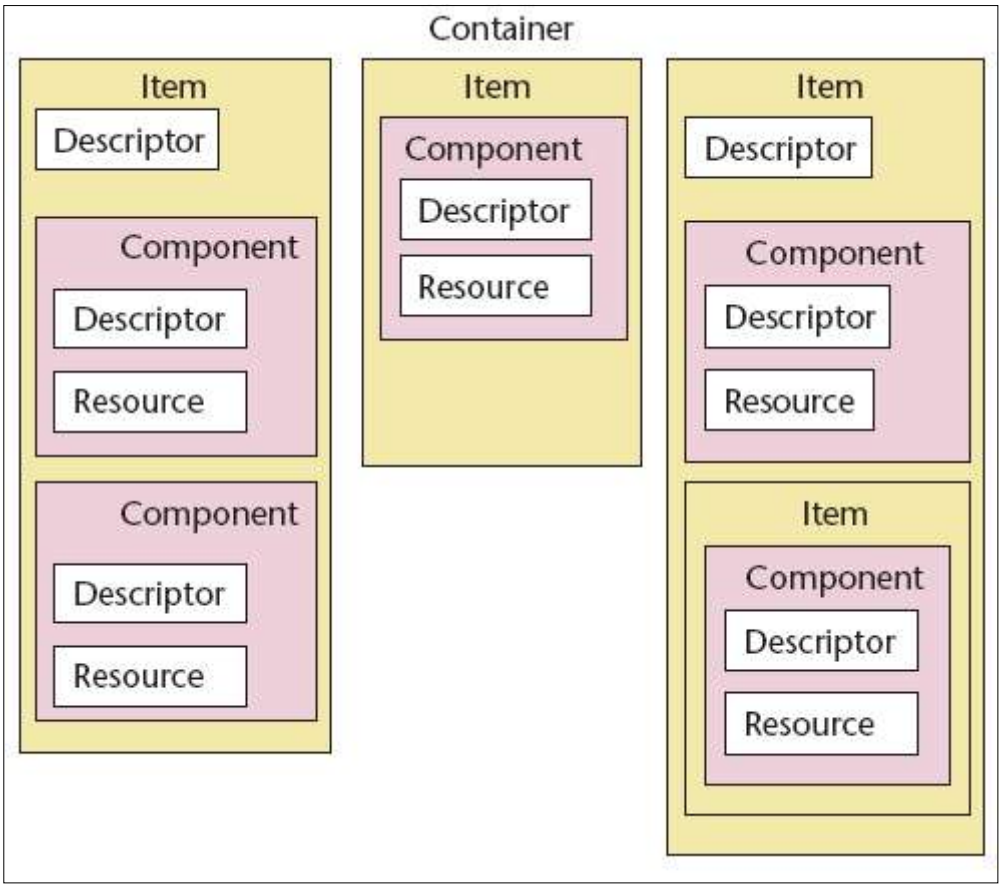

Figura 2.2 Elementi principali di DID e relazioni

La figura seguente mostra un esempio di *digital item declaration*

```
<?xml version="1. 0" encoding="UTF -8"?>
<DIDL
xmlns="urn:mpeg:mpeg21:2002:01 -DIDL-NS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema -instance"
xsi:schemaLocation ="urn:mpeg:mpeg21:2002:01 -DIDL-NS E:\Users\RVdW\Temp\DIDL.xsd">
    <Container >
        <Descriptor >
            <Statement mimeType ="text/plain">
                This information package was developed by University Records Unlimited
            </Statement >
        </Descriptor >
        <Item>
            <Descriptor >
                <Statement mimeType ="text/plain">
                    Copyright owner: University Records Unlimited
                    Permission: Read  Only
                </Statement >
            </Descriptor >
            <Descriptor >
                <Component >
                    <Resource ref="http://www.uruweb.org/logos/uow.jpg  " mimeType ="image/jpeg "/>
                </Component >
            </Descriptor >
            <Choice choice_id ="INFO_PICKER ">
                <Descriptor >
                    <Statement mimeType ="text/plain">
                        Choose the information you want to receive:
                    </Statement >
                </Descriptor >
                <Selection select_id ="VIDEO">
                    <Descriptor >
                        <Reference target="#VIDEO_TITLE "/>
                    </Descriptor >
                </Selection >
                <Selection select_id ="TEXT ">
                    <Descriptor >
                        <Reference target="#TEXT_TITLE "/>
                    </Descriptor >
                </Selection >
            </Choice >
            <Item>
                <Condition require="VIDEO"/>
                <Descriptor id="VIDEO_TITLE ">
                    <Statement mimeType ="text/plain">
                        Research over view video
                    </Statement >
                </Descriptor >
                <Component >
                    <Condition require ="VIDEO"/>
                    <Resource ref="http://www.uruweb.org/video/research.mp4  " mimeType ='video/mp4 "/>
                </Component >
            </Item>
            <Item>
                <Condition require="TEXT "/>
                <Descriptor id="TEXT_TITLE ">
                    <Statement mimeType ="text/plain">
                        Lecture notes
                    </Statement >
                </Descriptor >
                <Component >
                    <Condition require ="TEXT "/>
                    <Resource ref="http://www.uruweb.org/text/lecturenotes.txt  " mimeType="text/plain"/>
                </Component >
            </Item>
        </ Item>
    </Container >
</DIDL >
```

Figura 2.3Esempio di Digital Item Declaration

2.5.2.3 Digital Item Identification (DII)

Lo scopo di questa sezione include:

- come identificare univocamente i *digital item* e parte di questi.
- come identificare univocamente le proprietà intellettuali associate ai *digital item*.
- come identificare univocamente gli schema di descrizione
- come collegare i *digital item* a informazioni correlate quali i metadati
- come identificare diversi tipi di *digital item*

Da notare che in questa sezione non vengono introdotti sistemi di identificazione per elementi per cui sono stati già definiti da altri standard e nemmeno schemi di descrizione per il contenuto.

Le ultime due sezioni (DID e DII) sono strettamente correlate; possono essere associati identificatori definiti in DII a *digita item* includendo questi descrittori in *statement* definiti dentro DID. Questo viene mostrato nella seguente figura (Figura 2.4) che mostra come vengano aggiunte informazioni relative alla identificazione per la revisione della risorsa.

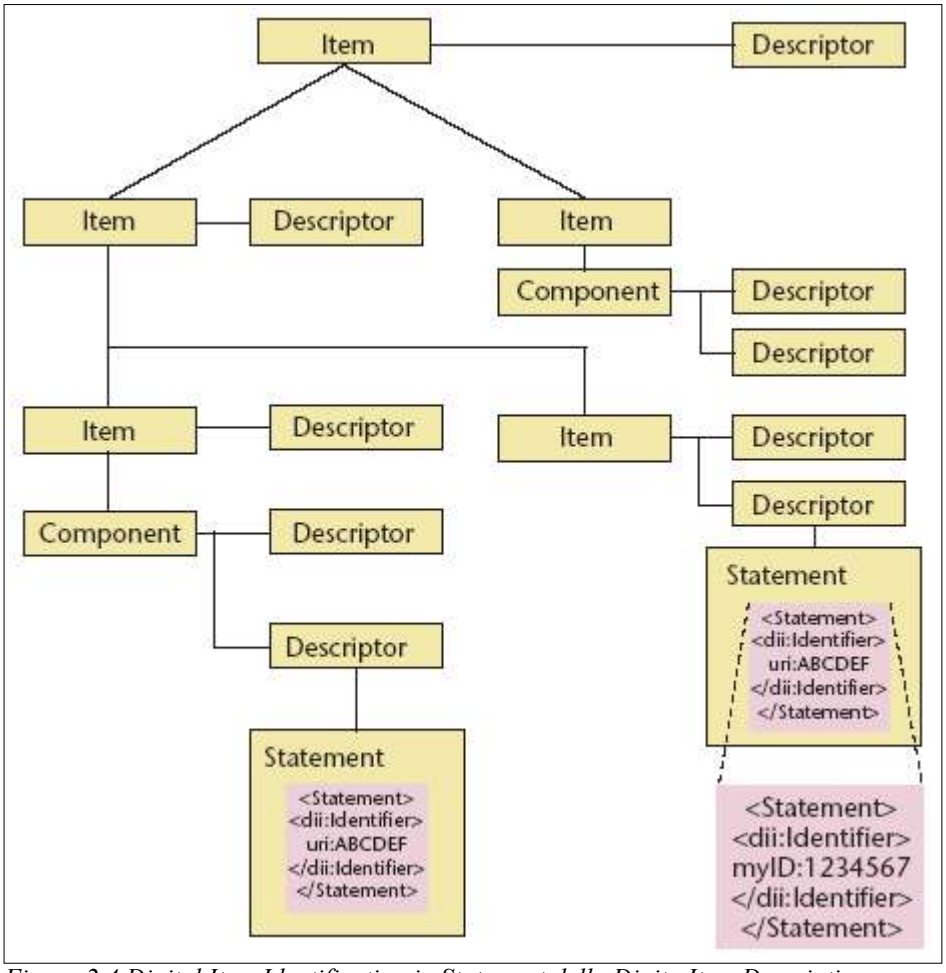

Figura 2.4 Digital Item Identification in Statement della Digita Item Description

2.5.2.4 Intellectual property management and protection (IPMP)

Questa sezione definisce una struttura interoperabile per la gestione della proprietà intellettuale. Questo standard viene dopo MPEG-4 e si aggancia al suo sistema di gestione delle proprietà intellettuali; tuttavia il sistema usato in MPEG-4 trova diverse implementazioni nei diversi dispositivi e spesso questi ultimi non riescono a collaborare tra loro. Lo standard qui proposto include un meccanismo di recupero da locazioni remote di strumenti per gestire IPMP nonché un sistema di scambio di messaggi tra i terminali coinvolti per ottenere maggiore interoperabilità.

Fornisce infine un sistema di integrazione per la gestione delle Right Expression, espressioni che esprimono i diritti sui *digital item,* in accordo con REL e RDD descritti di seguito.

2.5.2.5 Rights Expression Language (REL)

In questa sezione viene presentato un linguaggio, utilizzabile dalle macchine, per esprimere diritti e permessi sui "termini di uso" di una generico *digital item*. Questo linguaggio usa i termini che vengono definiti nel RDD descritto successivamente.

L'utilizzo della risorsa viene ad essere così "aumentato" grazie alla possibilità di utilizzare informazioni sui diritti sulla stessa contestualmente al suo utilizzo.

Inoltre fornisce un flessibile meccanismo per garantire che i dati personali vengano processati in accordo con i diritti stabiliti.

Modello dei dati per REL

Il modello dei dati per REL è costituito da quattro entità che cooperano:

1. *Principal*: incapsula l'entità per la quale i diritti e i permessi vengono utilizzati.
2. *Right* : rappresenta i diritti di accesso che devono essere garantiti per certe risorse sotto certe condizioni
3. *Resource*: rappresenta l'oggetto sul quale vengono garantiti i diritti
4. *Condition* : specifica i termini che devono verificarsi perché vengano forniti i diritti.

La figura seguente presa da [MPEG21O] mostra le relazioni che sussistono tra le succitate entità.

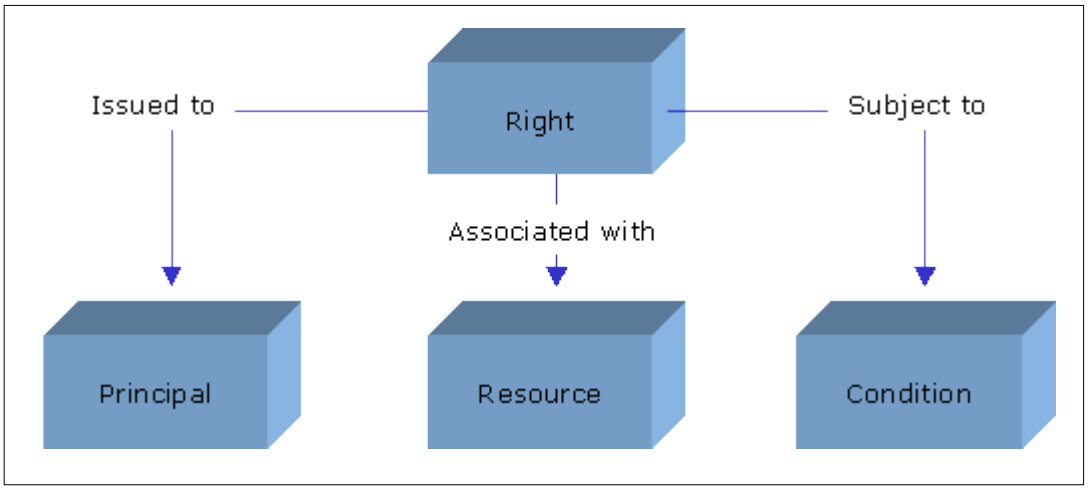

Figura 2.5 MPEG-21 entità in Right Expression Language Model

2.5.2.6 Rights Data Dictionary (RDD)

Questa sesta parte specifica un dizionario da utilizzare nello standard MPEG-21

Questo include un insieme di termini chiari, consistenti, strutturati, integrati ed univocamente identificati utilizzati poi in REL.

In RDD vengono riconosciute definizioni legali introdotte da altre autorità o che possono essere, al suo interno, mappate. Introduce quindi, oltre a un sistema di termini, un sistema per mappare quelli esistenti al suo interno mediante un meccanismo di *namespace*.

2.5.2.7 Digital Item Adaptation (DIA)

Questa settima parte di MPEG-21 specifica quali siano gli strumenti per l'adattamento dei *digital item*. Uno degli obiettivi principali di MPEG-21 è di consentire un accesso trasparente, rispetto all'ambiente di rete, alle risorse multimediali, nascondendo all'utente i problemi dovuti ai dispositivi di interconnessione e le questioni relative alla installazione. Per ottenere questo MPEG-21 definisce la struttura necessaria alla DIA (Figura 2.6). In questa struttura il *digital item* deve essere soggetto a un processo di adattamento della risorsa, ma non solo, anche di adattamento della descrizione nonché di adattamento del sistema DID, come mostrato nella Figura 2.6 .

Per l'adattamento è necessario che sia disponibile non solo la descrizione del contenuto ma anche una descrizione del formato e dell'uso nell'ambiente di questo, in modo da dare all'utente la migliore esperienza possibile nell'uso della risorsa.

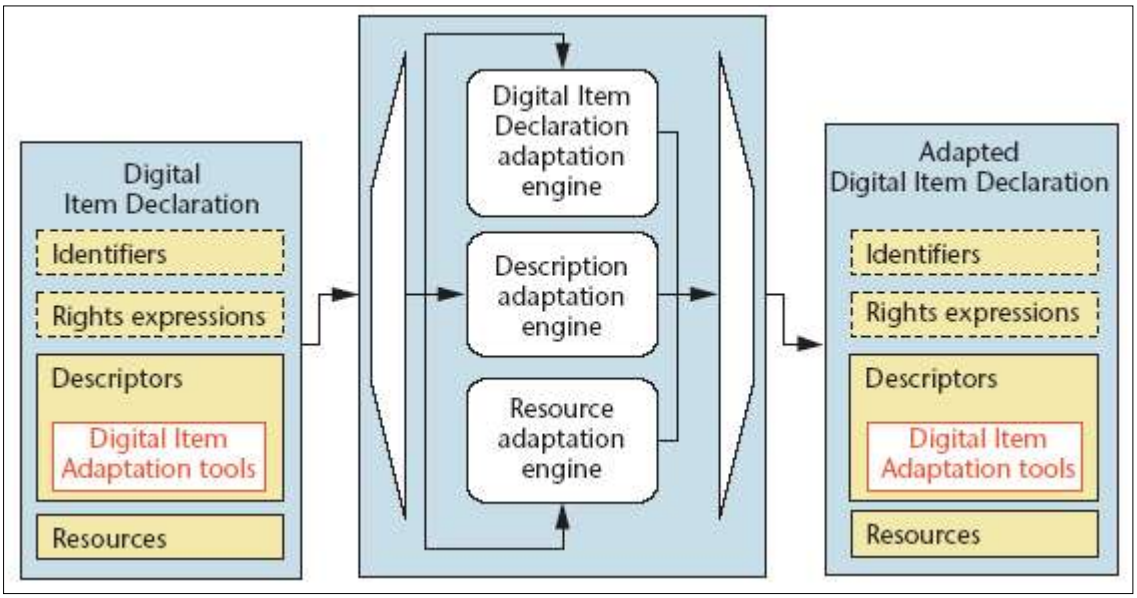

Figura 2.6 Digital Item Adaptation in MPEG-21

MPEG-7 viene usato per la descrizione del contenuto come DIA in MPEG-21 viene usato per la descrizione dell'ambiente e del formato. Questo significa che questa sezione di MPEG-21 non definisce, in modo assoluto, un meccanismo per l'adattamento del contenuto ma solo la struttura necessaria a realizzarlo; viene altresì evidenziato come le descrizioni introdotte da MPEG-21, e la struttura indipendente dal formato delle stesse, fornisce un valido supporto per la realizzazione dell'adattamento del contenuto, e per l'eventuale specifica della QoS.

Infine poniamo nuovamente l'accento sul fatto che nel modello proposto da MPEG-21 il processo di adattamento non coinvolge solo la risorsa ma anche la descrizione della stessa e del sistema usato per fare tale descrizione, questo significa che in una prima fase detta di *armonizzazione*, vengono trasformate le descrizioni, presenti in diversi formati, in un formato comune, queste poi vengono combinate, anche con informazioni inerenti all'ambiente, per ottenere una descrizione dettagliata in un formato finale compatibile con l'applicazione. Di questo modello non vi è riscontro in nessun altro standard in cui in genere le caratteristiche vengono *armonizzate* staticamente nella fase di progetto del sistema. Tuttavia il modello presentato risulta astratto in quanto non specifica come attuare queste operazioni.

2.5.2.8 Reference software

Contiene il software di riferimento o perlomeno quello candidato ad esserlo.

2.5.2.9 File format

Un *digital item* in MPEG-21 può contenere una complessa collezione di informazioni.

Può includere dati statici, come immagini, o dinamici, come video e ancora informazioni come metadati che li descrivono e così via. Può contenere dati testuali come XML e binari, come video codificati in MPEG-4. Per queste ragioni MPEG-21 riprende molto del lavoro fato in MPEG-4 per ottenere un formato in grado di supportare i più svariati propositi.

2.5.2.10 Digital Item Processing (DIP)

Uno degli aspetti essenziali in MPEG-21 è che il *digital item* è una dichiarazione statica solo di informazioni; nella dichiaratività stessa delle informazioni è insito il fatto che non vi siano informazioni riguardo a come debbano essere processate. Questo è importante in quanto è in contrasto con altri tipi di linguaggi, come ad esempio HTML in cui la dichiarazione del formato è combinata con l'informazione stessa. Se da un lato questo ha il vantaggio di avere informazioni separate dal contenuto, utilizzabili indipendentemente, dall'altro ha lo svantaggio di non dire nulla su come vadano trattate queste informazioni. Così questa sezione (DIP) ha il compito di indicare come vadano trattate le informazioni dal punto di vista dell'utente (l'applicazione che li riceve). Questa sezione copre gli aspetti procedurali associati alle informazioni come lo scaricamento delle descrizioni dei profili ad esempio per la gestione dei diritti, oppure l'esecuzione procedure (*method*) atte alla modifica dei dati stessi. Il concetto di *digital item method* sta nel fatto che l'utente ha a disposizione una serie di *method* applicabili al *digital item* per manipolarlo.

2.5.2.11 Evaluation methods for persistent association technologies

Questa sezione si occupa della valutazione delle implementazioni e delle tecnologie correlate a MPEG-21 cercando di armonizzare il tutto usando una metodologia comune.

2.6 XML

Diamo di seguito una breve introduzione a XML al fine di facilitare la comprensione degli argomenti successivi.

XML sta per eXtensible Markup Language; questo standard definisce una metodologia di presentazione dei dati di tipo *tag* assolutamente generica, in cui cioè gli elementi di marcatura (i *tag* appunto) assumono un significato in base al contesto in cui si trovano. Una caratteristica interessante di XML è il fatto che può definire dei *vocabolari* vale a dire dei contesti in cui i *tag* assumono particolari significati. Le tecnologie correlate a questo sono diverse; ne elenchiamo le più importanti:

- XML-namespace: consentono la possibilità di definire dei prefissi associati a degli identificatori univoci detti URI da usare nella definizione dei *tag* personalizzati. Questo

consente di avere TAG col medesimo nome che assumono significati diversi in base al XML-namespace selezionato. Questo è il meccanismo di base per definire quelli che precedentemente abbiamo chiamato vocabolari XML.

- URI, URL, URN: sono tre forme di identificatori molto simili tra loro:
  - URI Uniform Resource Identifier ([RFC2396]): un sistema più generale la cui sintassi include tutti gli altri.
  - URL Uniform Resource Locator: un sistema di nomi che attribuisce a una risorsa un nome dipendente dalla sua locazione; tutti gli indirizzi internet sono URL.
  - URN Uniform Resource Name: un sistema di attribuzione dei nomi che può dipendere dalla locazione o può essere costruito in altro modo.

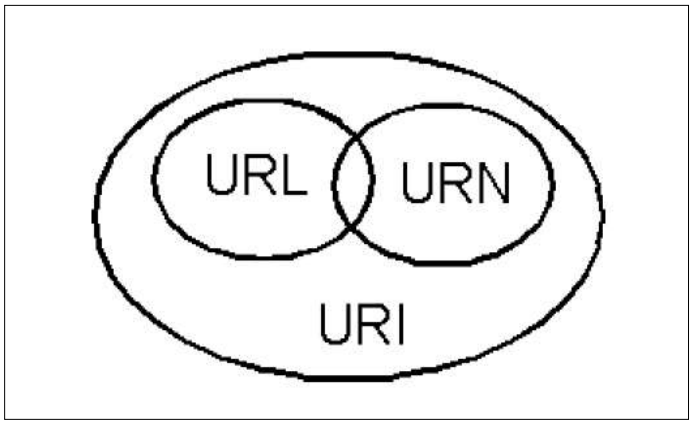

Figura 2.7 Sistemi di identificazione delle risorse

- Data Type Definition (DTD): consente di definire la "sintassi" che deve assumere un documento XML in termini di quali *tag* deve contenere, come devono essere innestati.
- XMLSchema: equivalente a DTD come proposito ma più completo e omogeneo. Più completo perché ha una varietà di tipi di dato maggiore e consente la definizione di strutture più complesse; omogeneo perché è scritto in XML stesso.
- eXtensible Stylesheet Language (XSL): Standard che consente di definire in XML le trasformazioni da associare ai *tag* XML per ottenere altri formati di documenti. Si divide in XSLT (XSL Tansformation) che descrive le trasformazioni, in termini quasi dichiarativi, da applicare ai singoli *tag*; ed XFO (XSL Formatting Object) atto a descrivere le caratteristiche visive dei *tag* descritti.

Una possibile idea per trattare i profili utente potrebbe essere la definizione di un *namespace* XML cioè di un vocabolario XML, e tramite XML Schema la sintassi dello stesso, in modo da definire con questo i profili, tuttavia questa idea in genere è da scartare perché,

fortemente chiusa, ed in oltre non prende in considerazione la possibilità di usare standard già realizzati e utilizzati.

2.7 CC/PP e UAProf

Recentemente due nuovi standard sono stati creati per la gestione dei profili utente, questi sono Composite Capabilities / Preferences Profile (CC/PP) creato da W3C [W3C] e User Agent Profile (UAProf) creato da WAP Forum oggi confluito in OMA [OMA]. Questi sono interoperabili e tra loro compatibili, e vengono utilizzati per la gestione dei profili dei cellulari WAP.

Essendo CC/PP del tutto compatibile con UAProf passeremo ora alla descrizione di questo standard evidenziando le eventuali differenze/estensioni di UAProf.

2.7.1 RDF

CC/PP è basato su Resource Description Framework [RDF] uno standard proposto inizialmente dalla W3C allo scopo di descrivere insiemi di oggetti in termini di proprietà ad essi associate. RDF si è poi evoluto ed è diventato un vero e proprio linguaggio semantico, cioè in grado di rappresentare gli oggetti e le relazioni tra queste.

RDF rappresenta a livello concettuale ciò che XML rappresenta nei sistemi di rappresentazione dei dati, in quanto è un vero e proprio linguaggio per esprimere concetti in modo formale; tali concetti vanno da relazioni quali *sopo-sottotipo* a relazioni quali *entità-istanza*; RDF non definisce una rappresentazione definitiva di sé stesso ma propone ed utilizza attualmente XML come linguaggio di scrittura formale oltre ad una notazione grafica ad albero di cui la Figura 2.8 rappresenta un esempio.

RDF si basa sull'idea che tutte le cose possono essere descritte come oggetti dotati di proprietà.

Anzitutto le *affermazioni* in RDF assumono una forma come, ad esempio, la seguente:

"http://exemple.com/index.html ha come creatore Marco"

in cui abbiamo una risorsa *soggetto* dell'*affermazione,* nell'esempio "http://exemple.com/index.html"; poi abbiamo il *predicato* indicato dalla *proprietà* "creatore" nell'esempio, ed infine abbiamo l'*oggetto* che nell'esempio è dato dal nome "Marco". Questo tipo di rappresentazione può essere presa come modello e rappresentata in modo grafico nel modo seguente.

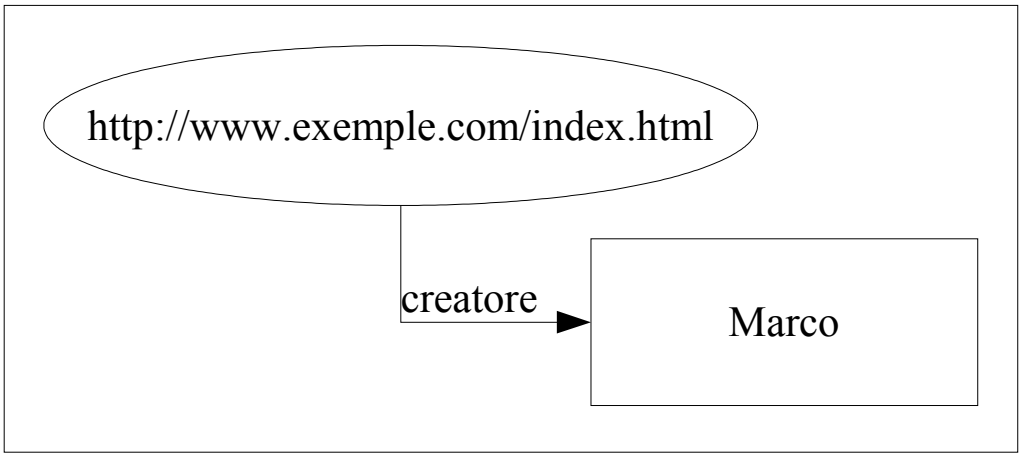

Figura 2.8 Asserzioni in RDF

RDF quindi definisce tre entità fondamentali:

- *risorse* che vengono associate a un URI ed indicano una entità esternamente definita
- i *predicati* che indicano una generica proprietà che viene associata al soggetto e punta sul valore della stessa
- le *asserzioni* che altro non sono che espressioni che coinvolgono un *soggetto* a cui viene applicato un *predicato* il cui valore rappresenta l'*oggetto* dell'asserzione

Va osservato come nella Figura 2.8 il *predicato* è rappresentato da una freccia, inoltre viene presentato l'oggetto come un rettangolo, questo perché rappresenta un *termine letterale* cioè un valore. In RDF le entità possono essere *risorse* o *letterali* con il vincolo che il *soggetto* di una *asserzione* deve essere una *risorsa* mentre l'*oggetto* può essere un *letterale* o una *risorsa*. Questo significa che posso avere predicati con all'interno altri predicati come nel seguente esempio (Figura 2.9) in cui anche i *predicati* assumono un nome uguale ad un URI.

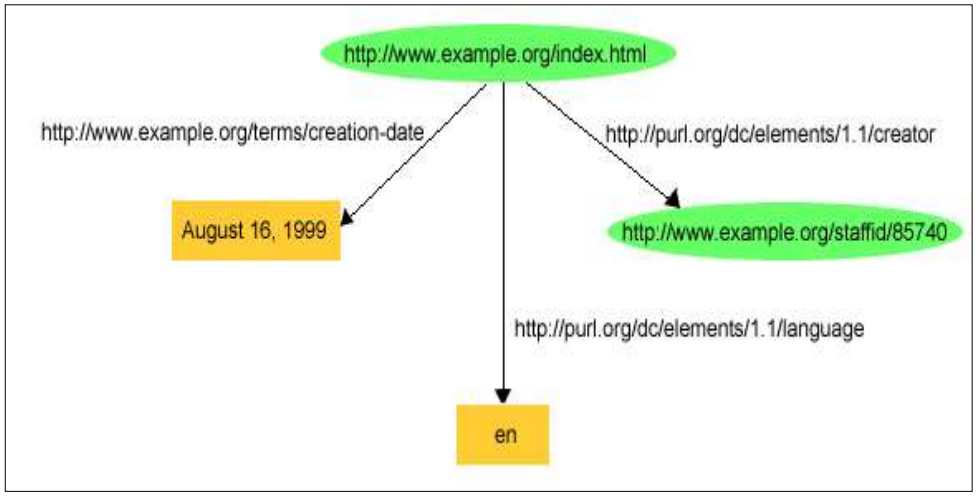

Figura 2.9 Esempio di espressione RDF

RDF è quindi uno standard che non fissa inizialmente le regole di un linguaggio per essere rappresentato, ma rappresenta un sistema concettuale la cui rappresentazione attualmente attualmente può essere realizzata con la tecnologia XML.

Infine va osservato che nella serializzazione in XML, RDF si avvale dei namespace XML per definire e distinguere tra diversi vocabolari. Inoltre, come avviene analogamente per lo stesso XML, in RDF i vocabolari vengono definiti in RDF.

Un documento RDF serializzato in XML deve avere una struttura ben precisa di cui di seguito ne viene dato un esempio.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:exterms="http://www.example.org/terms/">
 <rdf:Description rdf:about="http://www.example.org/index.html">
    <exterms:creation-date>August 16, 1999</exterms:creation-date>
 </rdf:Description>
</rdf:RDF>
```

La radice di un documento RDF è sempre il *tag* XML <rdf:RDF> al cui interno vengono inseriti le dichiarazioni dei namespace che rappresentano i vocabolari RDF selezionati.

All'interno abbiamo *tag* *RDF:Description* in cui viene inserito, come parametro XML in *rdf:about,* il valore del soggetto dell'affermazione, poi dentro questo *tag* abbiamo il *predicato* rappresentato da un *tag* XML con dentro il valore. I tipi di valori in RDF possono essere valori singoli oppure di tipo *contenitori*, a questa categoria appartengono gli oggetti di tipo *Bag*, *Sequence* e *Alternative*. Per tutti questi tipi di valore nella rappresentazione del letterale contenuto come valore possiamo avere una notazione di base estesa in cui i valori vengono inseriti come *tag* XML, ma abbiamo anche una versione semplificata in cui i TAG sono inseriti come *attributi* XML del *tag* in cui vanno inseriti. Un documento RDF-XML viene considerato corretto se rispetta la sintassi introdotta per RDF oltre ad essere un documento XML "ben formato".

2.7.2 CC/PP struttura dei documenti ed equivalente grafo RDF

CC/PP è basato sulla rappresentazione serializzata di RDF però impone che l'altezza dell'albero RDF, rappresentazione grafica del sistema di descrizione, sia pari a due cioè avremo una *risorsa RDF* con al suo interno, legata da un predicato, un'altra *risorsa RDF* che è vincolata a contenere *letterali RDF* ovvero no può contenere altre *risorse RDF*; ; questo, i definitiva, significa che in CC/PP abbiamo un *profilo* che contiene uno o più *componenti* al cui interno abbiamo uno o più *attributi* che devono essere dei *letterali* oppure di tipo *rdf:Sequence* o *rdf:Bag* vale a dire collezioni di *letterali*.

Il seguente esempio di profilo CC/PP viene presentato come grafo RDF nella successiva Figura 2.10

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:ccpp="http://www.w3.org/2002/11/08-ccpp-schema#"
       xmlns:ex="http://www.example.com/schema#">

 <rdf:Description
    rdf:about="http://www.example.com/profile#MyProfile">

  <ccpp:component>
   <rdf:Description
      rdf:about="http://www.example.com/profile#TerminalHardware">
    <rdf:type
       rdf:resource="http://www.example.com/schema#HardwarePlatform" />
    <ex:displayWidth>320</ex:displayWidth>
    <ex:displayHeight>200</ex:displayHeight>
   </rdf:Description>
  </ccpp:component>

 </rdf:Description>
</rdf:RDF>
```

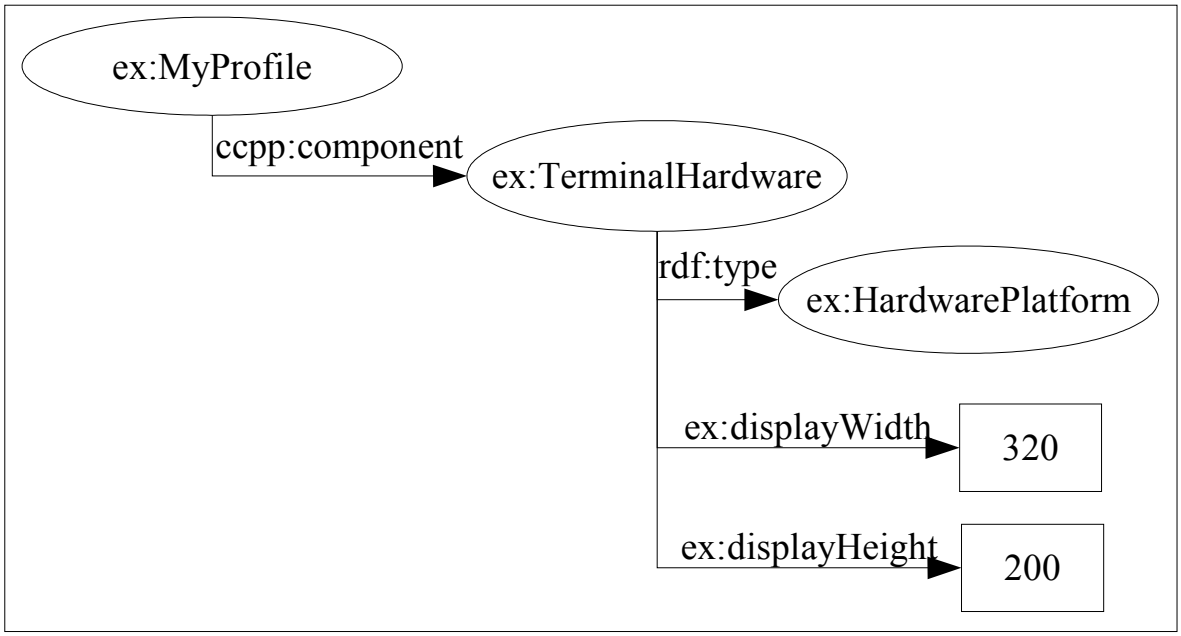

Figura 2.10 Grafo RDF associato a profilo CC/PP

2.7.3 Uso di Default in CC/PP

In CC/PP gli attributi di un componente possono essere specificati direttamente nel componente oppure, all'interno dello stesso, può essere presentato un riferimento (URI) a un componente di default da cui prelevare gli attributi.

Un esempio di questo viene presentato qui di seguito

Esempio di documento con riferimento a componenti di default :

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:ccpp="http://www.w3.org/2002/11/08-ccpp-schema#"
       xmlns:ex="http://www.example.com/schema#">
 <rdf:Description
    rdf:about="http://www.example.com/profile#MyProfile">
  <ccpp:component>
   <rdf:Description
      rdf:about="http://www.example.com/profile#TerminalHardware">
    <rdf:type
       rdf:resource="http://www.example.com/schema#HardwarePlatform" />
    <ccpp:defaults
       rdf:resource="http://www.example.com/hardwareProfile#HWDefault" />
   </rdf:Description>
  </ccpp:component>
 </rdf:Description>
</rdf:RDF>
```

Documento RDF contenente il componente di default "HWDefault" referenziato dal precedente esempio

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:ex="http://www.example.com/schema#">
 <rdf:Description
    rdf:about="http://www.example.com/hardwareProfile#HWDefault">
  <rdf:type
     rdf:resource="http://www.example.com/schema#HardwarePlatform" />
  <ex:displayWidth>320</ex:displayWidth>
  <ex:displayHeight>200</ex:displayHeight>
 </rdf:Description>
</rdf:RDF>
```

I default vengono introdotti per ridurre il traffico di rete, in pratica possiamo fare in modo che quando sia necessario inviare un profilo a un gestore dello stesso, anziché il profilo, venga inviato una struttura vuota con solo il riferimento al profilo di default il quale è già a disposizione del gestore, ed eventualmente alcuni attributi aggiuntivi. L'eventuale conflitto tra le caratteristiche inserite "in linea" e quelle prese dal default viene risolto dando la priorità ai componenti inseriti nel profilo "in linea".

UAProf e Default

In UAProf il *tag* per esprimere i default ha una sintassi leggermente differente che quindi viene accettata in CC/PP per compatibilità. Inoltre mentre in CC/PP i default fanno sempre riferimento a elementi esterni un UAProf questi possono essere inseriti direttamente profilo utilizzato. Questo, sebbene possa sembrare una pratica non utile, può servite per definire dei componenti utilizzabili interamente come dei default da usare in un sistema di composizione dei profili.

2.7.4 Estensibilità dei vocabolari e uso dei namespace XML

Un *vocabolario* CC/PP è costituito dall'insieme dei tipi di componenti e degli attributi ad esso associati e viene identificato tramite un URI. Si osserva inoltre che un vocabolario CC/PP può essere definito mediante un vocabolario RDF (detto **RDF Schema**), pratica peraltro consigliata nella definizione di CC/PP anche se non in maniera vincolante.

Il meccanismo per accedere a questi vocabolari nel documento CC/PP è dato dell'introduzione di un *namespace* XML (si veda [XMLNAMESPACE]) con un *namespace prefix* ad esso associato da usare nella definizione dei componenti. Mostriamo di seguito l'inizio di un documento CC/PP in cui vengono selezionati i vari vocabolari mediante l'uso dei *namespace* XML.

```
<?xml version="1.0"?>
<RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ccpp="http://www.w3.org/2002/11/08-ccpp-schema#"
    xmlns:prf="http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#">
.....
```

Ricordo che i namespace prefix che abbiam usato fino adesso (ccpp e rdf) non sono obbligatori nel senso che possono cambiare da documento a documento, l'importante è che siano associati agli URI corretti.

Osserviamo come vengono costruiti i nomi completi dei tipi di componenti e dei nomi di attributi: questi nella loro forma estesa sono rappresentati appendendo il proprio nome locale all'URI che identifica il loro vocabolario, si osserva quindi che tale URI non è un URI generico ma un fragment-URI (in prima battuta URI seguito dal carattere '#').

Abbiamo precedentemente fatto riferimento a un nome locale con questo viene indicato il nome completo privato del fragment-URI iniziale.

Inolte si osserva che in elementi RDF, quali rdf:type, in cui viene usato rdf:resource che associa un URI all'elemento, è necessario sempre usare l'URI completo.

Identificatori in CC/PP e UAProf

In CC/PP le entità vengono in genere identificate tramite l'attributo XML *about* in questo è riportato il nome completo del componente, in UAProf invece si usa *ID* in cui viene inserito il nome locale del componente e che viene ammesso in CC/PP per compatibilità. Segue ora un esempio di *ID* e di *about* usati come nome per un componente:

Uso di rdf:about in

```
<rdf:Description rdf:about="http://www.example.com/profile#TerminalHardware">
```

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ccpp="http://www.w3.org/2002/11/08-ccpp-schema#"
         xmlns:ex="http://www.example.com/schema#">

  <rdf:Description
     rdf:about="http://www.example.com/profile#MyProfile">

    <ccpp:component>
      <rdf:Description rdf:about="http://www.example.com/profile#TerminalHardware">
        <rdf:type rdf:resource="http://www.example.com/schema#HardwarePlatform" />
        <ex:displayWidth>320</ex:displayWidth>
        <ex:displayHeight>200</ex:displayHeight>
      </rdf:Description>
    </ccpp:component>

  </rdf:Description>
</rdf:RDF>
```

Uso dell'equivalente elemento rdf:ID in:

```
<rdf:Description rdf:ID="TerminalHardware">
```

da osservere che l'URI di base è comunque individuato dal *tag* XML rdf:type.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:ccpp="http://www.w3.org/2002/11/08-ccpp-schema#"
       xmlns:ex="http://www.example.com/schema#">

 <rdf:Description
    rdf:about="http://www.example.com/profile#MyProfile">

  <ccpp:component>
   <rdf:Description rdf:ID="TerminalHardware">
    <rdf:type rdf:resource="http://www.example.com/schema#HardwarePlatform" />
    <ex:displayWidth>320</ex:displayWidth>
    <ex:displayHeight>200</ex:displayHeight>
   </rdf:Description>
  </ccpp:component>

 </rdf:Description>
</rdf:RDF>
```

Infine facciamo notare come per i nomi degli attributi venga usato il *namespace prefix* per identificare l'URI di base da associare al nome dell'attributo, questo nel precedente esempio si riflette nell'uso del namespace prefix "ex:" in:

```
<ex:displayWidth>320</ex:displayWidth>
```

2.7.5 Componenti CC/PP e rappresentazioni in XML

I componenti in CC/PP sono entità dotate di un nome ed un tipo e possono essere rappresentati in due modi equivalenti:

1. rappresentazione di base di cui forniamo un esempio

```
<rdf:Description rdf:ID="MyHWPlatform">
<rdf:type rdf:resource="http://www.wapforum.org/profiles/UAPROF/ccppschema-
20010430#HardwarePlatform"/>
...
</rdf:Description>
```

2. rappresentazione abbreviata di cui forniamo un esempio

```
<prf:HardwarePlatform rdf:ID="MyHWPlatform">
...
</prf:HardwarePlatform>
```

Osserviamo come nella rappresentazione abbreviata venga usato il *namespace prefix* per individuare l'URI di base e il nome del tipo compaia in chiaro come nome del *tag* XML.

2.7.6 Attributi CC/PP e rappresentazioni in XML

Gli attributi in CC/PP appartengono a tipi di base: *stringhe*, *numeri*, e *razionali*.

Le *stringhe* sono le solite sequenze di caratteri, i *numeri* sono l'insieme degli interi con segno, i *razionali* hanno una forma del tipo <intero>"/"<intero senza segno> e rappresentano un rapporto; in UAProf questi nomi diventano un attributo RDF detto base-type che assume valore rispettivamente: Literal, Number e Rational per le *stringhe*, i *numeri*, i *razionali*; inoltre UAProf introduce altri due tipi di base Boolean che può assumere valore vero o falso e Dimension del tutto simile a Rational ma nella forma <intero senza segno>"x"<intero senza segno> con un ovvio significato semantico.

Inoltre gli attributi possono essere semplici o complessi; se complessi vengono rappresentati come Bag o Sequence cioè assumono come valore un insieme di valori; il tipo Bag rappresenta un insieme senza ordine mentre Sequence rappresenta una tupla di valori in cui è rilevante l'ordine degli elementi.

In UAProf questa caratteristica assume il nome dell'attributo RDF composition, nel vocabolario, che può avere valore simple, set o sequence rispettivamente per elementi di tipo semplice, o complesso: rdf:Bag o rdf:Sequence.

Notazione per gli attributi semplici: possono essere rappresentati secondo una notazione base o abbreviata cioè come *tag* XML o come *attributi* XML come presentato nel seguente esempio

notazione di base	notazione abbreviata
<rdf:Description rdf:ID="MySWPlatform" > <prf:ScreenSize>121x87</prf:ScreenSize> <prf:Model>R999</prf:Model> .... </rdf:Description>	<rdf:Description rdf:ID="MySWPlatform" prf:ScreenSize="121x87" prf:Model="R999"> ... </rdf:Description>

Le notazioni per gli attributi con valori complessi possono essere anche in questo caso semplici o abbreviate come mostrato nel seguente esempio:

notazione di base	notazione abbreviata
<prf:InputCharSet> <rdf:Bag> <rdf:li>ISO-8859-1</rdf:li> <rdf:li>US-ASCII</rdf:li> <rdf:li>UTF-8</rdf:li> <rdf:li>ISO-10646-UCS-2</rdf:li> </rdf:Bag> </prf:InputCharSet>	<prf:CcppAccept-Charset> <rdf:Bag rdf:_1="US-ASCII" rdf:_2="ISO-8859-1" rdf:_3="UTF-8" rdf:_4="ISO-10646-UCS-2"/> </prf:CcppAccept-Charset>

Da notare come le notazioni abbreviate consentano di avere come nomi dei valori più

complessi in quanto gli attributi XML possono contenere caratteri come ad esempio "#" che non possono essere inseriti come elementi.

Osserviamo infine che per le notazioni di attributi complessi la forma abbreviata induce un ordine con l'uso di numeri come nomi degli attributi XML mentre nella forma base abbiamo che l'ordine è indotto dalla struttura; si ricordi che l'ordine è rilevante solo per il tipo rdf:Sequence.

2.7.7 UAProf regola di composizione dei profili

Spesso è necessario fare il *merge* o la *risoluzione* di diversi profili; questa procedura è spesso legata al fatto che nel passaggio, che fa il profilo attraverso diversi nodi tra il cliente e il fornitore, i nodi intermedi possono introdurre frammenti per rappresentare il proprio stato e per mettere a conoscenza il fornitore della situazione dell'ambiente.

CC/PP non specifica come comporre diversi profili, invece in UAProf viene specificato un algoritmo.

Gli attributi appartenenti allo stesso tip di componente vengono inseriti in un unico componente; l'inserimento è regolato l'uso di un parametro associato ai vari attributi che indica la regola di risoluzione (attributo *resolution-rule*) questo può assumere uno dei tre seguenti valori:

locked: utilizzabile con attributi semplici e complessi indica che il primo profilo in cui si trova tale attributo fissa il valore per tale attributo

override: utilizzabile con attributi semplici e complessi indica che l'ultimo profilo in cui si trova tale attributo fissa il valore per tale attributo

append: utilizzabile con attributi complessi indica che i valori vanno uniti in un unico insieme.

La procedura completa, a dire il vero, include anche una fase di inizializzazione con i default.

2.7.8 CC/PP architettura di supporto e protocolli CCPP-ex e W-HTTP

In [CCPPreq] viene ipotizzata una struttura di supporto per l'uso dei profili nel protocollo HTTP e WAP. L'architettura iniziale presentata in Figura 2.11 prevede che dal dispositivo client, in questo caso un PC, venga inviato direttamente il profilo verso il fornitore e che questo risponda adattando il contenuto

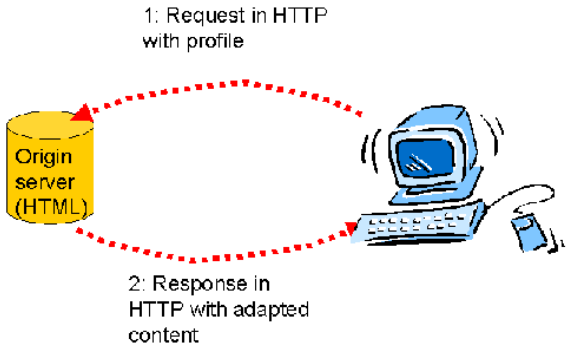

Figura 2.11 Architettura Iniziale

In una successiva presentazione la struttura è tale per cui il cliente invia una richiesta con all'interno un riferimento al profilo ed è il fornitore a reperirlo da un contenitore separato, questo viene illustrato in Figura 2.12; questa architettura può essere estesa con l'invio da parte del cliente di un riferimento e altri valori di profili aggiuntivi. Questo ultimo tipo di architettura può essere ulteriormente estesa per l'inserimento, da parte di entità intermedie, di informazioni circa lo stato della rete.

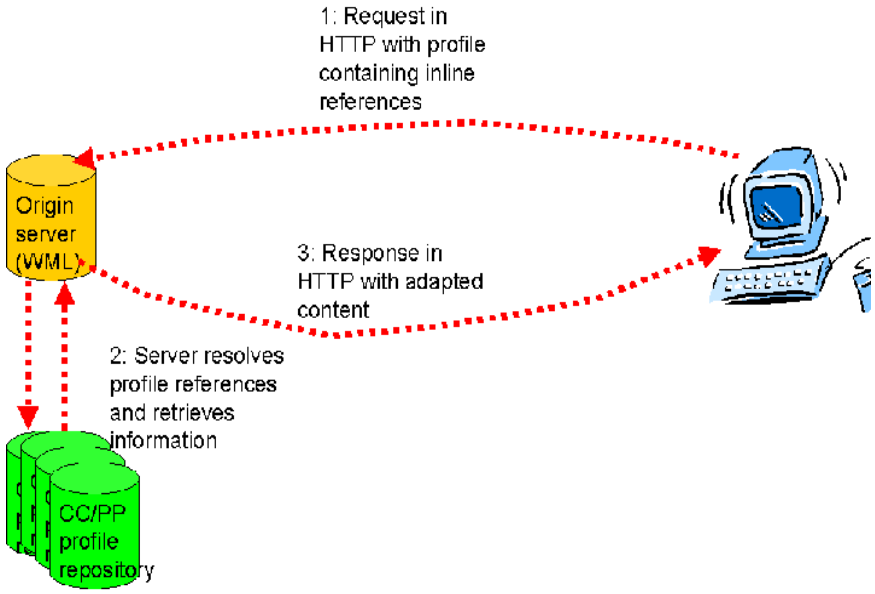

Figura 2.12 Architettura con server dei profili

Poi vengono presentate altre architetture del tutto simili alle precedenti in cui vengono eventualmente aggiunte entità che gestiscono la profilazione del documento sulla base di un profilo associato al documento stesso da fornire, oppure in cui abbiamo dei proxy che nei passi intermedi ricevono un contenuto non adattato e un profilo ed eseguono la profilazione al passo finale compiendo degli adattamenti; da notare che i proxy possono agire da adattatori in entrambe le direzioni, se ad esempio il cliente non è in grado di inviare un profilo il proxy

potrebbe selezionare il profilo in base ad altri parametri, ad esempio l'identità del dispositivo cliente, e passare un profilo scelto adeguatamente al fornitore del servizio.

In [CCPPex] invece viene proposta una estensione del protocollo HTTP basata a sua volta su una estensione del protocollo HTTP cioè HTTP-ex che consente l'introduzione di parametri identificati numericamente all'interno dell'intestazione della richiesta HTTP nonchè l'estensione del set di comandi utilizzabili durante il protocollo HTTP, questa estensione è tuttora sperimentale e quindi sconsigliata nell'uso tuttavia è stata ripresa e introdotta in modo compatibile in HTML i W-HTTP (Wireless Profiled Protocol) un sistema definito in [UAProf] dalla OMA.

CCPP-ex consente il passaggio di *profile* e *profile-diff* dove gli oggetti *profile* altro non sono che un riferimento (URL) al profilo da usare come base e i *profile-diff* sono altri frammenti di profili da comporre con quello iniziale; da osservare che il meccanismo qui presentato non ha nulla a che vedere con i default introdotti nei componenti CC/PP anche se può essere usato con medesima finalità. L'introduzione dei *profile-diff* consente a eventuali nodi intermedi di aggiungere informazioni circa lo stato dei nodi, in effetti oltre a questa estensione in CCPP-ex si parla di risoluzione *end-to-end* oppure *hop-by-hop*; questa si riferisce appunto alla possibilità rispettivamente di usare il profilo direttamente tra cliente e fornitore oppure avere dei nodi intermedi che aumentano di frammenti i profili e, ad esempio possono compiere adattamento; un esempio di ciò può essere un proxy che riceve una richiesta da un browser che tratta solo immagini in formato gif mentre lui è in grado di convertirle da jpeg a gif, ora passerà un richiesta al server HTTP in nell'elenco dei formati supportati c'è anche jpeg (questo in CCPP-ex significa aggiungere un frammento CC/PP) nell'intestazione della richiesta, e se l'immagine ricevuta è in formato jpeg sarà il proxy a convertirla per il browser.

Nell'intestazione HTTP i campi aggiuntivi introdotti sono due: "*man*:" indica obbligatorietà della richiesta "*opt*:" indica opzionalità. Inoltre sono previsti comandi aggiuntivi M-GET e M-POST; tuttavia questi ultimi sono da sconsigliare in quanto non standard.

Di seguito viene presentato un esempio di richiesta HTTP in formato CCPP-ex

```
GET /a-resource HTTP/1.1
Host: www.w3.org
Man: "http://www.w3.org/1999/06/24-CCPPexchange" ; ns=25
25-Profile: "http://www.ex.com/hw","1-CWccARHXxtYJE+rKkoD8ng=="
25-Profile-Diff-1: <?xml version="1.0"?>
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:prf="http://www.wapforum.org/profiles/UAPROF/ccppschema-
20010430#">
  <rdf:Description rdf:ID="MyDeviceProfile">
  <prf:component>
     <rdf:Description rdf:ID="HardwarePlatform">
        <rdf:type
         rdf:resource="http://www.wapforum.org/esempio#HardwarePlatform"/>
        <prf:SoundOutputCapable>No</prf:SoundOutputCapable>
     </rdf:Description>
  </prf:component>
  </rdf:Description>
  </rdf:RDF>
```

Da notare la sintassi è la seguente: il campo *Man:* deve contenere sempre "http://www.w3.org/1999/06/24-CCPPexchange", di seguito una dichiarazione di namespace "ns=25" dove "25" è scelto a caso; poi i vari campi appartenenti al namespace 25 che iniziano con "25-" nel nome che sono (privati del prefisso):

- un elemento "Profile" che contiene l'url associata al profilo di base e una serie opzionale di identificatori di *profile-diff* nella forma "<NUM>-<MD5DIGEST>" dove <NUM> è un numero progressivo che indica in che ordine processare i successivi *profile-diff* ed <MD5DIGEST> altro non è che un codice calcolato sull'intero *profile-diff*
- i successivi elementi "Profile-Diff-<NUM>" dove <NUM> fa riferimento al numero dichiarato nel campo "Profile".

Da notare che <MD5DIGEST> viene usato a scopo di caching per i *profile-diff* o a scopo di verifica.

I problemi di questa codifica sono i namespace HTTP numerici, il <MD5DIGEST> che non è affidabile in quanto HTTP non considera i caratteri bianchi, nonchè l'uso eventuale di M-GET al posto di GET.

Poniamo di seguito lo stesso esempio di richiesta però in W-HTTP; da osservare come i campi usati hanno un nome fisso: "x-wap-profile" e "x-wap-profile-diff" che si sostituiscono a "25-Profile" e "25-Profile-Diff-1" dell'esempio precedente, e i parametri numerici relativi al numero di *profile-diff* che sono inseriti come valore del campo *x-wap-profile-diff*; questo facilita il trattamento dei campi, in quanto elimina i namespace numerici, inoltre non serve più il campo *man* o *opt* lasciando libertà, a chi riceve, se trattare o mento il profilo.

```
GET /a-resource HTTP/1.1
Host: www.w3.org
x-wap-profile: "http://www.ex.com/hw","1-CWccARHXxtYJE+rKkoD8ng=="
x-wap-profile-diff: 1;<?xml version="1.0"?>
  <rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:prf="http://www.wapforum.org/profiles/UAPROF/ccppschema-
20010430#">
  <rdf:Description rdf:ID="MyDeviceProfile">
  <prf:component>
     <rdf:Description rdf:ID="HardwarePlatform">
        <rdf:type
         rdf:resource="http://www.wapforum.org/esempio#HardwarePlatform"/>
        <prf:SoundOutputCapable>No</prf:SoundOutputCapable>
     </rdf:Description>
  </prf:component>
  </rdf:Description>
  </rdf:RDF>
```

In definitiva i due sistemi proposti sono quasi equivalenti ed è da consigliare il secondo in quanto i campi introdotti hanno nome fisso e non introduce comandi aggiuntivi quali "M-GET" proposto in CCPP-ex.

2.8 Conclusioni

In questo paragrafo abbiamo inizialmente presentato la relazione tra la QoS e la profilazione, di seguito abbiamo mostrato tre sistemi possibili per la gestione dei profili utente collegati con i sistemi di rete e con le applicazioni di tipo multimediale evidenziando come in queste architetture vengano presentati e gestiti i metadati e come sia possibile usarli per gestire l'adattamento dl contenuto ed in definitiva la profilazione.

3 Sistema preesistente allo sviluppo

In questo capitolo viene inizialmente fatta una breve introduzione delle tipologie di architetture computazionali nonché i meccanismi per la comunicazione in ambiente condiviso, poi vengono brevemente presentati gli ambienti SOMA e l'architettura MUM a cui ci si appoggia.

3.1 Paradigmi per la comunicazione in ambiente condiviso

Per comunicazione in ambiente condiviso si intende la possibilità, da parte di processi, o di architetture, di comunicare tra loro, ovvero di scambiarsi informazioni. Questo si rende necessario per varie ragioni quali ad esempio la semplice sincronizzazione tra entità che devono collaborare per portare a termine un lavoro comune. La comunicazione può avvenire sulla stessa macchina oppure tra macchine diverse. Le soluzioni per la comunicazione tra diversi processi sono essenzialmente due:

- memoria condivisa
- scambio di messaggi

Nel caso di memoria condivisa le entità che devono comunicare hanno libero accesso ad una zona di memoria comune in cui depositano i messaggi e da cui li prelevano. Questo tipo di soluzione benché molto libera è, nei casi reali, assai poco applicabile in quanto risulta essere intrinsecamente poco scalabile, infatti per avere una efficiente gestione di un tale tipo di memoria è necessario prevedere un accesso diretto, da parte di ogni entità, a questa risorsa che risulta quindi una sorta di "collo di bottiglia" in cui tutti depositano, e prelevano dati. La soluzione a memoria condivisa è attuata in genere all'interno di una stessa macchina in cui i vari processi o thread in cui viene scomposto internamente un processo possono accedere a questa zona comune in modo coordinato.

Il caso dello scambio dei messaggi sebbene abbia meno potenzialità rispetto a quello della memoria condivisa è di gran lunga la soluzione migliore in un ambiente in cui si necessaria la scalabilità; il fatto che non sia necessario occupare completamente la comune risorsa per potere eseguire l'invio di messaggi, rende il sistema dello scambio dei messaggi molto più efficiente e scalabile. Questa è la soluzione adottata in ambiente di rete, in cui è necessario far comunicare macchine tra loro tramite connessione.

Oltre alla comunicazione tra processi si deve tenere conto della semantica della comunicazione vale a dire come questa avviene: come i processi vengono avvisati e come

inviano i messaggi.

In base a chi scatena l'avvenuta ricezione del messaggio abbiamo il modello *pull* in cui chi è interessato al messaggio lo deve prelevare verificando se questo è presente; oppure il modello *push* in cui l'interessato riceve l'informazione della presenza del messaggio contestualmente alla ricezione.

In base al fatto se ci si aspetta risposta o meno dal messaggio abbiamo messaggi *sincroni* in cui è attesa una risposta e *asincroni* in cui non ci aspettiamo alcuna risposta.

Si potrebbe catalogare il sistema dello scambio dei messaggi anche in base ad altre caratteristiche che tuttavia non interessano l'aspetto logico della comunicazione.

3.2 Paradigmi per l'esecuzione in ambiente condiviso

In ambiente condiviso i modelli computazionali sono diversi e classificabili i base a dove viene eseguito il codice e chi possiede l'entità computazionale ovvero l'algoritmo di esecuzione. Oggi assistiamo all'emergere di un nuovo modello computazionale basato sulla mobilità del codice, questo significa che un processo è in grado di interrompere la propria esecuzione, spostarsi su un altro sito, e riprendere l'esecuzione. I modelli basati sulla mobilità del codice sono due:

- mobilità forte del codice: in questo caso abbiamo che il processo viene congelato e con esso il suo stato di esecuzione, viene spostato, e viene riavviato in un altro sito. Questo tipo di mobilità presuppone un sistema di esecuzione del codice che dia accesso a quello che viene definito lo *stato di esecuzione* del codice, che in genere è cablato nel processo o nell'interprete che lo esegue ed è costituito dalle strutture dati atte al progredire dell'esecuzione (stack, heap, eventuali file descriptor eccetera). Ovviamente questo modello è difficile da realizzare soprattutto perché per realizzarlo è necessario uno strato di astrazione che mi consenta di localizzare le risorse in rete in modo trasparente alla allocazione.
- mobilità debole del codice: in questo caso il processo interrompe volontariamente la propria esecuzione, rilasciando le eventuali risorse occupate sul sistema su cui stava eseguendo, viene serializzato, e trasferito nel nuovo sito in cui riprenderà l'esecuzione da un punto prefissato. Questo modello risulta di più facile realizzazione sia per il fatto che non è necessario trasferire lo stato di esecuzione sia perché non si rende necessario ripristinare i legami che questo aveva con risorse che ora si trovano al di fuori del sistema su cui esegue.

Volendo commentare questi due modelli per la mobilità potremmo dire che nel primo caso trasferiamo il processo mentre nel secondo caso trasferiamo l'algoritmo, ove per processo intendiamo l' "algoritmo in esecuzione".

In ambiente di rete assistiamo ai seguenti modelli per l'esecuzione in ambiente di rete:

- Client/Server (C/S): l'entità *client* richiede l'esecuzione a una entità *server* su cui risiede sia la risorsa oggetto dell'esecuzione sia la procedura necessaria all'esecuzione.
- Remote EValuation (REV): l'entità richiede l'esecuzione a un sistema remoto che possiede la risorsa oggetto della valutazione mentre la procedura necessaria all'esecuzione è detenuta dall'entità richiedente che deve quindi trasferirla al sistema a cui fa la richiesta.
- Code On Demand (COD): l'entità che richiede l'esecuzione possiede la risorsa oggetto dell'elaborazione e richiede al servizio la procedura da eseguire, che viene poi trasferita trasferita dal servizio al richiedente.
- Mobile Agent (MA) : l'entità che deve eseguire si muove sul sistema su cui si trova la risorsa oggetto della computazione.

3.3 Ambiente SOMA

SOMA (Secure and Open Mobile Agent) è un sistema che viene sviluppato presso il Dipartimento di Elettronica, Informatica e Sistemistica (DEIS) della facoltà di Ingegneria Informatica dell'Università di Bologna. Essenzialmente è un sistema ad agenti mobili, sviluppato in Java, che realizza la *mobilità debole* del codice.

Il sistema costituisce una infrastruttura per il movimento e per l'esecuzione degli agenti.

Le principali caratteristiche di SOMA sono:

- Portabilità: questo viene realizzato grazie all'ambiente Java che può eseguire su qualsiasi piattaforma, dalle potenti Workstation ai ridottissimi cellulari.
- Scalabilità: tramite opportune ipotesi di località il sistema risulta ampliabile con minimo degrado prestazionale.
- Sicurezza: il sistema prevede un modello di esecuzione con credenziali vale a dire che è possibile associare un proprietario all'agente e delle politiche sull'accesso che questo può avere nel sistema.
- Apertura: questa è ottenuta tramite la compatibilità con CORBA una infrastruttura standard internazionale per l'esecuzione nel distribuito, secondo un paradigma orientato agli oggetti.

- Dinamicità: attualmente in SOMA è possibile aggiungere ambienti di esecuzione dinamicamente e la modifica viene propagata ai nodi interessati.

Ipotesi di località e schema dei nomi:

In SOMA gli ambienti di esecuzione in cui si trova a lavorare un agente viene detto *place* ed è l'astrazione di un *nodo in un grafo*. Affinché il sistema risulti scalabile, il sistema dei nodi viene organizzato in *domini* e gerarchie di *domini*.

Il *place* rappresenta l'ambiente di esecuzione in cui un *agente SOMA* va ad eseguire come mostrato nella seguente figura

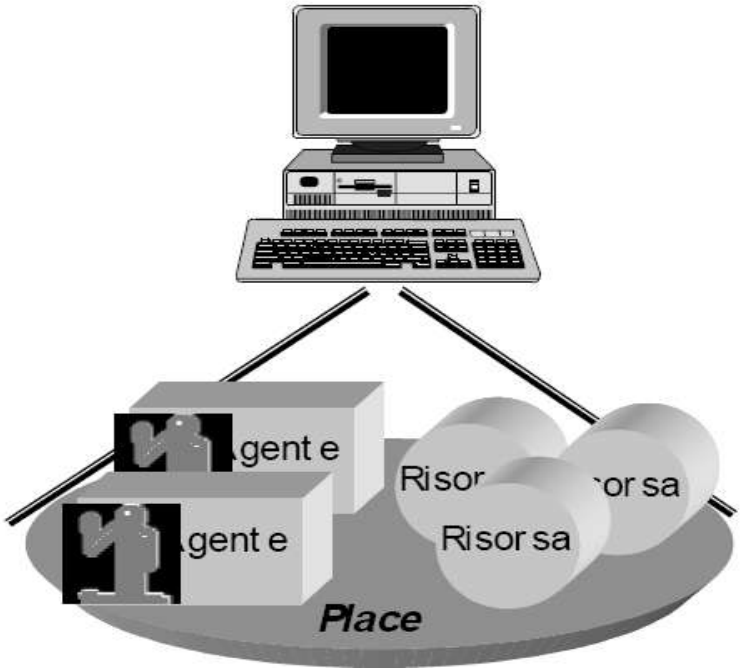

Figura 3.1 Astrazione di Place

Un *place*, come abbiamo detto, è l'astrazione del nodo in cui l'agente in SOMA si trova ad eseguire, tuttavia deve essere anche localizzabile nel sistema ed è per questo che i *place* sono dotati di nomi e di un sistema di nomi. I *place* si distinguono in *place comuni* e *place di default*, un *place di default* è responsabile per un *dominio SOMA* e ne gestisce il *sistema di nomi* (Place Name Service).Ogni *place comune* appartiene ad un *dominio*, cioè viene registrato nella tabella dei nomi del suo p*lace di default*, questa tabella è detta PNS (Place Name Service) contiene l'associazione tra i nomi dei *place* del dominio e il loro indirizzo ed è replicata su tutti i *place* del dominio, la gestione di questa viene eseguita dal *place di default* del *domino*.

I *domini SOMA* vengono organizzati in gerarchia, cioè ogni p*lace di default* ha un

riferimento al suo p*lace di default Padre*. Oltre a questo abbiamo che nei p*lace di default* viene gestita un'altra tabella detta DNS (Domain Name Service), questa associa ai vari *default place* appartenenti alla gerarchia gli indirizzi.

Il sistema dei nomi è tale che all'interno del sistema i *place comuni* vengono subito individuati come appartenenti a un certo *dominio* in quanto:

il nome del *place di default* è una stringa alfanumerica senza spazi e dà il nome al *dominio* che amministra, il nome dei *place comuni* è costituito da due stringhe alfanumeriche senza spazi separate da uno o più caratteri di spazio in cui la prima stringa è il nome del *dominio* di appartenenza, vale a dire il nome del *default place* a cui fa' riferimento.

Di seguito in Figura 3.2 viene presentata una gerarchia di *domini SOMA* ciascuno dei quali, al suo interno, possiede vari *place comuni*, oltre naturalmente ad un d*efault place* che rappresenta il gestore del dominio

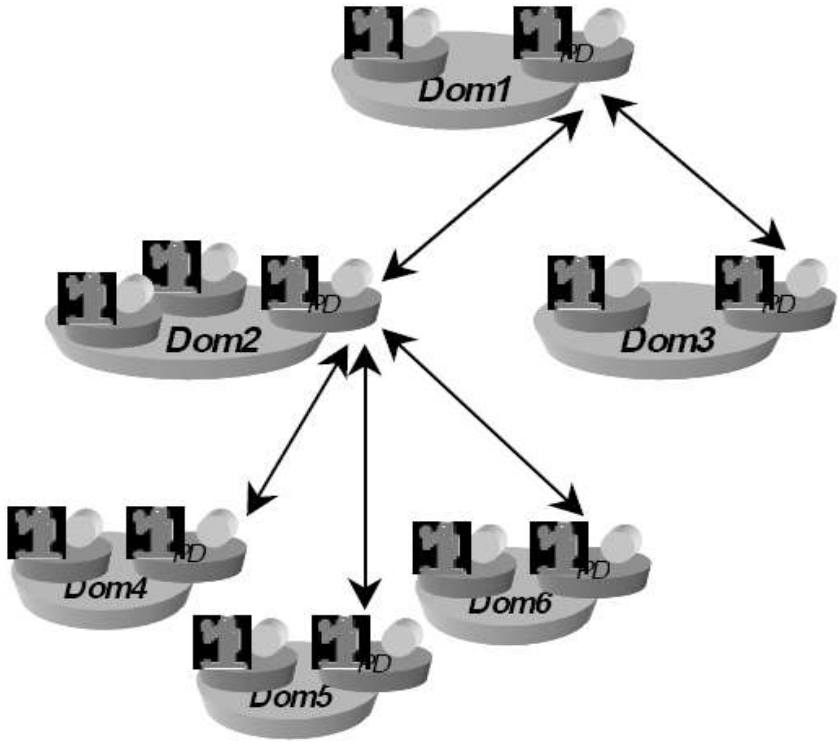

Figura 3.2 Una gerarchia di Domini SOMA

Per quel che riguarda il sistema ad agenti questi hanno un accesso all'ambiente di esecuzione tramite un attributo associato all'agente stesso che consente, tra l'altro, di reperire un riferimento all'ambiente completo. Per il movimento questo viene realizzato tramite un meccanismo di spostamento in cui è possibile decidere: dove migrare e quale procedura

lanciare al momento dell'arrivo nel place di destinazione. Inoltre è da osservare come in SOMA gli agenti vengano realizzati come *oggetti passivi* vale a dire che il flusso computazionale viene realizzato da un'entità esterna (contestualmente un thread Java) che utilizza la *struttura agente* eseguendola. A tale tipo di oggetti si contrappongono gli *oggetti attivi* in cui la politica di esecuzione è inglobata, assieme ai dati, nell'oggetto stesso.

Oltre a questo in SOMA abbiamo un meccanismo per lo scambio di messaggi tra agenti, ed un meccanismo a memoria condivisa basato su tuple che però è utilizzabile solo da *place* posti nello stesso *dominio*.

Abbiamo, in SOMA, supporto per la sicurezza in quanto è possibile associare agli *agenti* una sorta di firma digitale per autenticarli, e nei *place* è possibile stabilire le politiche di accesso degli *agenti* all'ambiente in base alle credenziali da questo esposte.

SOMA a basso livello viene realizzato mediante il paradigma REV infatti il trasporto dell'*agente*, i messaggi scambiati tra gli agenti, nonché i messaggi per l'aggiornamento dei sistemi dei nomi, e per la gestione quindi della topologia dell'ambiente è gestito mediante l'astrazione di *commmand* che portano con sé la procedura da eseguire; vale a dire che abbiamo degli oggetti di tipo *command* che vengono inviati tramite socket sui *place* in cui vengono poi eseguiti.

3.4 Architettura di MUM

MUM (Mobile agent based Ubiquitous multimedia Middleware) è un sistema che viene sviluppato presso il Dipartimento di Elettronica, Informatica e Sistemistica (DEIS) della facoltà di Ingegneria.

Fondamentalmente MUM è un *middleware,* basato su SOMA, per la gestione dei flussi multimediali nei nuovi ambienti di rete wireless in cui abbiamo utenti che possono spostarsi all'interno della rete.

Gli obbiettivi di MUM sono:

- Service configuration: fondamentalmente questo servizio consiste nella possibilità di scaricare il software necessario all'esecuzione in modo trasparente.
- Session continuity: consente di realizzare la mobilità utente di tipo *nomadic* o *roaming*, vale a dire movimento dell'utente in cui l'utente chiude la sessione e la riattiva, oppure l'utente si sposta da un nodo a un altro durante l'erogazione del servizio senza interrompere la sessione; ovviamente in quest'ultimo caso l' "utente" è un terminale mobile mentre nel caso *nomadic* l'utente è l'utilizzatore effettivo che si sposta da un

terminale a un altro.

- Quality of Service: questo viene ottenuto in modo statico, riservando e tenendo conto delle risorse occupate sui vari nodi, e in modo dinamico monitorando lo stato delle risorse sui nodi che si trovano sul percorso lungo cui viene erogato il servizio, in breve il service path.

Successivamente è stato integrato in MUM un sistema di caching per dati e metadati detto MUM Object Caching, detto MUMOC, che ha introdotto le seguenti caratteristiche:

- Content caching: caching dei contenuti e dei metadati
- Distributed metadata browsing: che consente l'accesso distribuito ai metadati associati alle risorse messe a disposizione da MUM.

MUM è strutturato a livelli, come mostrato nella tabella presentata più avanti, in cui sono state evidenziate le caratteristiche di MUM ai vari livelli e come sono organizzate all'interno dei servizi realizzati.

In particolare abbiamo i seguenti servizi:

- a livello Middleware Mechanism Layer:

  Si occupa della gestione a basso livello delle risorse quali la memorizzazione di profili, i servizi di localizzazione, la memorizzazione del software scaricabile, e il monitoraggio e allocazione di risorse di basso livello di sistema quali CPU e banda passante.

- a livello Middleware Facility Layer

  Si occupa ad alto livello della dei servizi i scaricamento del software, configurazione automatica, gestione del cambio di sessione, gestione delle risorse accessibili nel sistema, i filmati, con specifica di QoS.

Tipologia applicativa	Livelli	Paradigma Adottato	Service Configuration Entità e servizi	Session Continuity Entità e servizi	Quality of Service Entità e servizi
	Application Layer				
dinamica	Middleware Facility Layer	Mobile Agent	Configuration Service, Download Service	Session Continuity Service, Download Service	Resource Manager
statica	Middleware Mechanisms Layer	Clinet/Server	Profile Metadata Manager, Location Service, Resource Manager, Software Manager	Location Service, Software Manager	Resource Monitor, Resource Brokers

Presentiamo di seguito una analoga rappresentazione di MUMOC e dei livelli ad esso associati.

Anche qui abbiamo le seguenti entità interessate:

- a livello Middleware Mechanism Layer

Si occupa a basso livello della gestione della cache relativa ai contenuti multimediali, nonchè le tracce iniziali degli stessi (prefix), messi in cache per avere una risposta più veloce in caso di richiesta; mentre per quel che riguarda la memorizzazione dei metadati si occupa della cache in formato XML dei metadati associati alle presentazioni, e della definizione di un protocollo di comunicazione tra le cache.

- a livello Middleware Facility Layer

Si occupa ad alto livello dell'accesso alla cache dei filmati e dell'accesso ai metadati.

Tipologia applicativa	Livelli	Paradigma Adottato	Content Cache Entità e servizi	Metadata Browser Entità e servizi
	Application Layer			
dinamica	Middleware Facility Layer (Middleware Component)	Mobile Agent	Content Cache	Metadata Browser
statica	(Application Component) Middleware Mechanisms Layer	Clinet/Server	Presentation Store, Prefix Store	Metadata-XML repository, Protocol Module

Per una descrizione dettagliata dei componenti presentati nelle tabelle si veda [MUM].

Come viene evidenziato nelle stesse tabelle MUM rappresenta un middleware ovvero una struttura che realizza dei meccanismi a livello base su cui poi va costruita l'applicazione vera e propria. Al suo interno dunque, le varie entità, cooperano allo scopo di fornire supporto al livello applicativo sul quale si aggancia l'applicazione di interfaccia con l'utente.

Una nota va fatta sull'attuale supporto in MUM per la gestione dei profili utente, questi infatti vengono gestiti con una struttura a due livelli, da un lato abbiamo i profili degli utenti fisici con le loro caratteristiche di base: nome, identificatore dell'utente; all'interno del profilo dell'utente abbiamo un elenco di identificatori relativi ai dispositivi utente da lui usati e associati a questi il percorso, nella gerarchia SOMA, delle località in cui utilizza questi dispositivi. La struttura a due livelli consente di riutilizzare i profili dei dispositivi per diversi utenti. I profili degli utenti e dei dispositivi, in MUM, vengono memorizzati indipendentemente su un unico *repository* globale e questo può essere un limite, potrebbe essere una soluzione migliore memorizzarli separatamente e/o con tecniche diverse. Inoltre non è preente in MUM nessun meccanismo per la composizione di profili o per l'espressione di preferenze utente. Infine i profili utente non vengono sottoposti ad alcun meccanismo di cache. Questi ultimi infine sono gli obbiettivi di studio dell'attuale progetto di tesi.

3.5 Conclusioni

Nel capitolo qui illustrato abbiamo messo in evidenza anzitutto i ossibili modelli computazionali per poi passare a una introduzione degli ambienti su cui si inserisce il progetto sviluppato.

4 Analisi dei Requisiti

Nel capitolo che andiamo a presentare andremo a evidenziare la struttura necessaria allo sviluppo del sistema progettato in questo lavoro di tesi, inoltre metteremo in risalto le problematiche che andremo ad affrontare e il metodo di integrazione nell'architettura MUM.

4.1 Descrizione generale del sistema

Il sistema di gestione dei profili considerato ha i seguenti obbiettivi: gestione separata dei profili relativi agli utenti rispetto ai profili dei dispositivi, possibilità nel profilo dell'utente di introdurre preferenze relative al comportamento del dispositivo da comporre in un profilo finale da usare per fruire dei servizi offerti nell'infrastruttura MUM; infine ci si propone come obbiettivo l'interoperabilità, ottenuta mediante un sistema standard di rappresentazione dei dati [CCPP], e il minimo disturbo sull'architettura MUM ottenuta tenendo separato il sistema di gestione rispetto a quello che è la suddetta architettura.

Il sistema di gestione dei profili che andiamo a sviluppare in questo progetto di tesi assume come fondamentali alcune caratteristiche del sistema su cui ci appoggiamo, in particolare si considera come caratteristica fondamentale la struttura gerarchica dei *domini SOMA*. Questo perché, come vedremo, utilizzeremo l'astrazione di *path SOMA* per ottenere identificatori associati ai *domini SOMA* e questi verranno composti con i *nomi utente* per ottenere un sistema di nomi univoci in tutta l'architettura MUM; inoltre il fatto di avere una gerarchia ci consente di sviluppare un sistema di gestione, per i profili degli utenti, distribuito gerarchicamente.

4.2 Sistema di gestione dei profili separato

Il meccanismo di gestione dei profili deve essere separato dall'architettura MUM per perturbare il meno possibile l'erogazione dei servizi in questo introdotti. Per questo motivo si è pensato di introdurre una entità esterna ad ogni dominio SOMA ad esso associata e a cui viene delegata la risoluzione e la gestione, in termini di memorizzazione, dei profili.

La figura seguente (Figura 4.1) illustra il concetto sopra enunciato; come possiamo notare in prima istanza, per ogni *dominio SOMA* abbiamo una entità deputata alla gestione dei profili; ricordiamo che un *dominio SOMA* è identificabile col *default place SOMA* che ne gestisce il sistema di nomi.

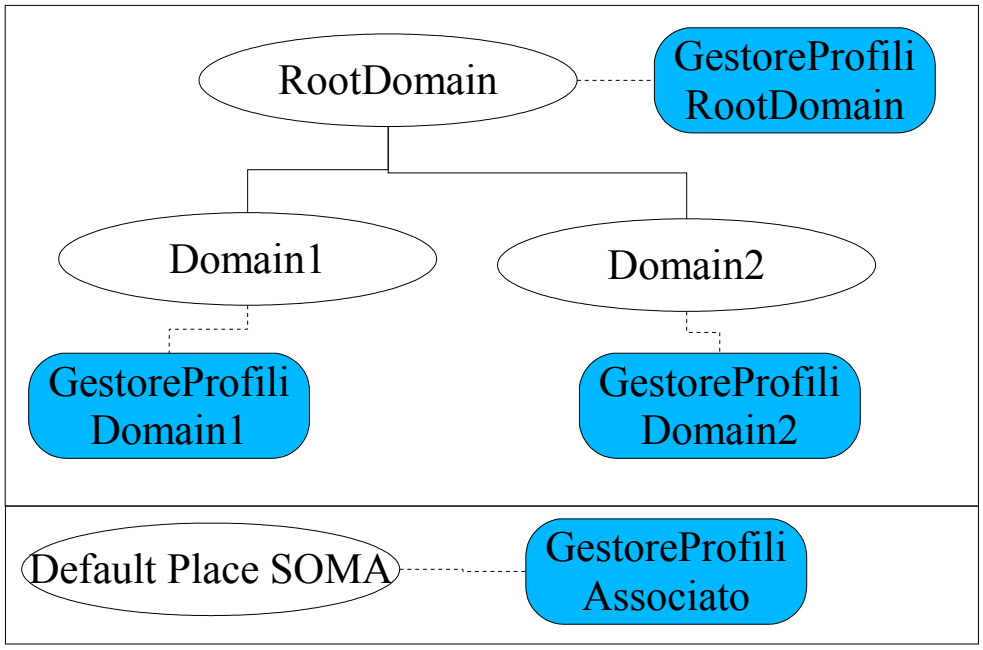

Figura 4.1 Domini SOMA e gestori dei profili associati

Successivamente possiamo pensare di distribuire meglio il carico computazionale dovuto alla gestione di profili e, ad esempio potremmo avere un unico gestore per diversi domini ad come illustrato di seguito in Figura 4.2.

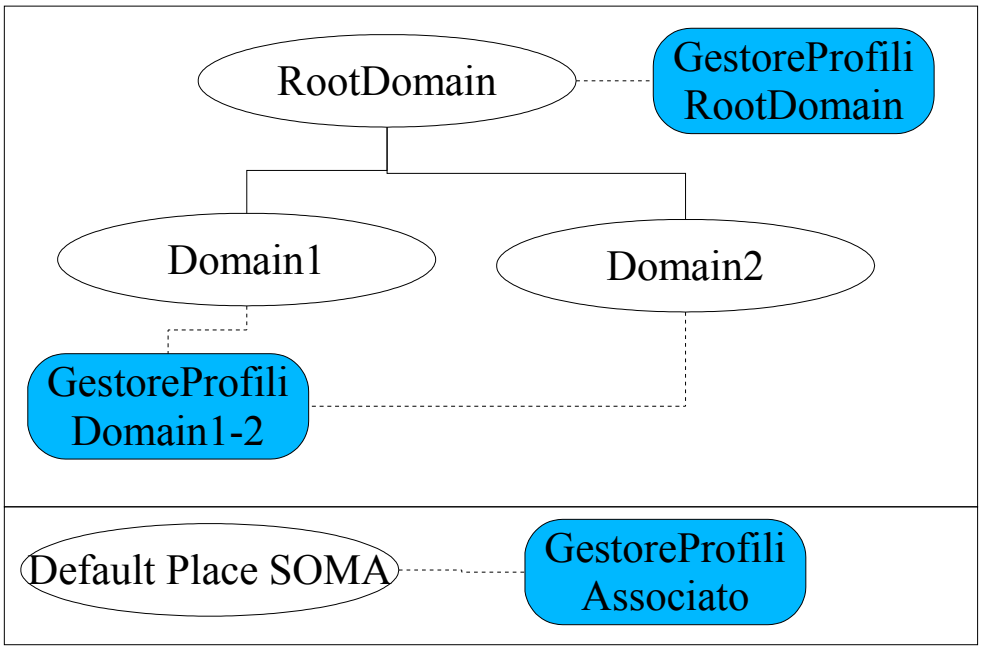

Figura 4.2 Due Domini SOMA associati allo stesso gestore

Ma ovviamente è possibile anche la situazione opposta in cui possiamo avere un gestore che ripartisce il carico computazionale su altri sub-gestori come mostrato nella Figura 4.3

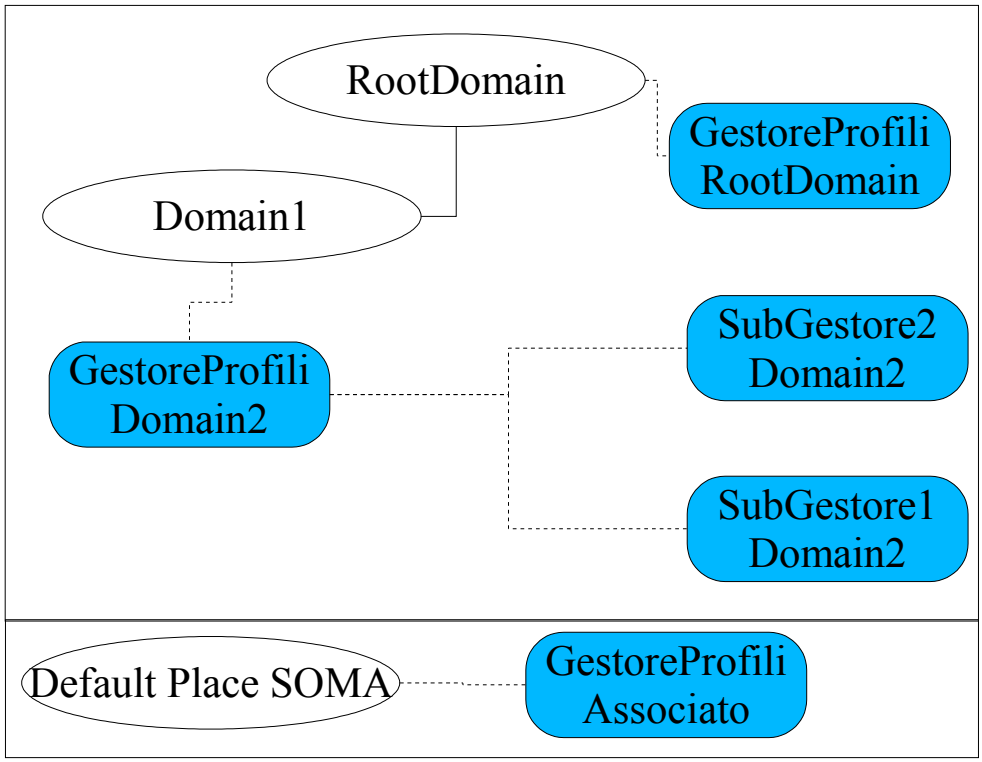

Figura 4.3 Dominio SOMA con due sub-gestori di profili

La convenienza o meno di una delle situazioni presentate in precedenza deve essere ovviamente studiata staticamente e decisa in base alle possibili attese nel sistema, infatti è da ricordare che i *domini SOMA* in MUM vengono considerati in gerarchia fissata all'avvio del sistema e non modificabile.

Prima di proseguire è doveroso specificare il modo in cui i profili, degli utenti e dei dispositivi, saranno trattati dall'interno di MUM verso i gestori. Il paradigma considerato è il modello *Client/Server* in modo *pull* per quel che riguarda l'accesso ai profilo, cioè quando un utilizzatore accede in MUM in genere ha accesso in un *dominio* e da questo le richieste di accedere al profilo vengono *ridirette* verso il gestore a questo associato il quale si occupa poi di reperire questi ultimi.

La Figura 4.4 seguente mostra come da un *dominio SOMA* ci di interfaccia verso il gestore associato. Il protocollo è un tipico protocollo *request-response*; in essa abbiamo evidenziato il susseguirsi delle operazioni etichettandole con un numero.

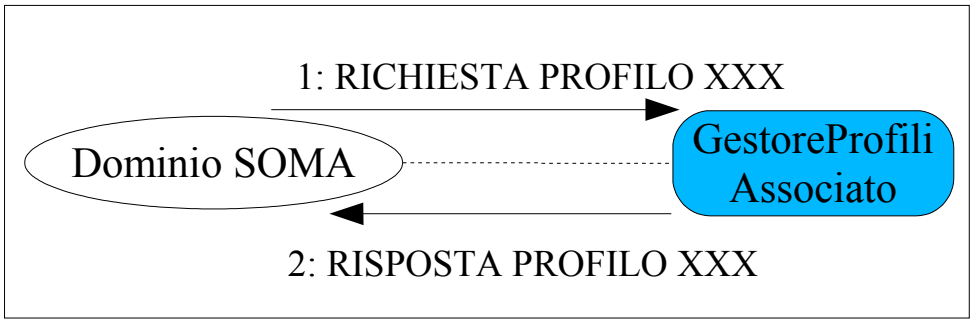

Figura 4.4 Protocollo request response tra dominio e gestore associato

Come precedentemente accennato il sistema deve perturbare il meno possibile le attività all'interno di MUM e per questo l'idea di base è che da ogni *dominio SOMA* le richieste vengano fatte sempre allo stesso gestore dei profili, poi deve essere lui a contattare le eventuali altre entità per reperire il profilo, queste ultime altro non sono che altri gestori di profili con competenza sui profili richiesti; questo è quanto illustrato nella seguente figura.

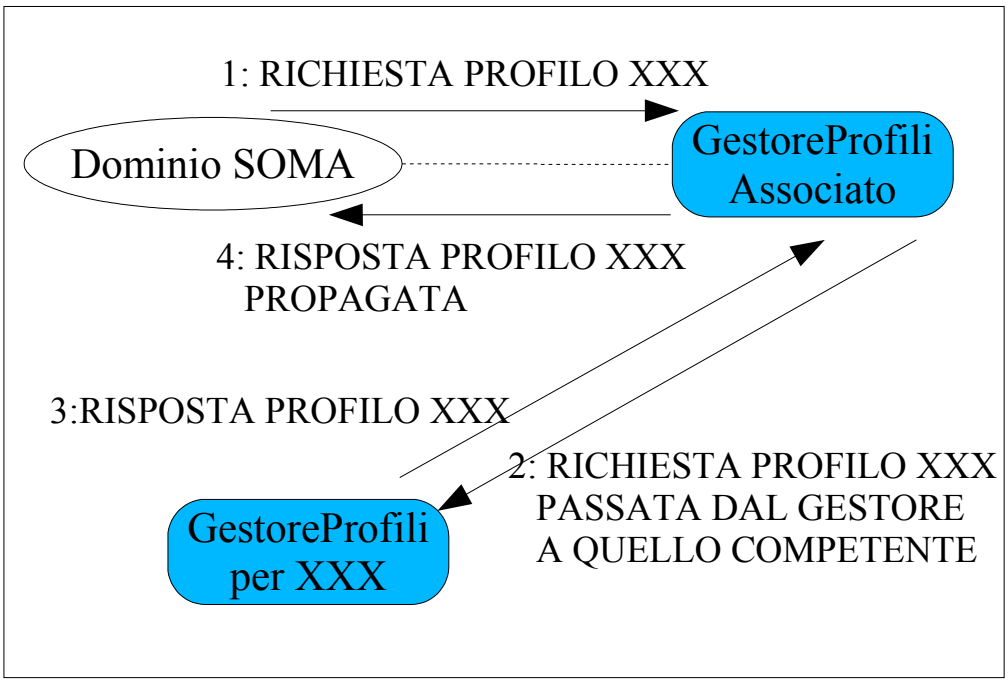

Figura 4.5 Protocollo di recupero dei profili

Uno schema come il precedente, in cui viene delegata l'entità gestore a reperire il profilo esternamente, come vedremo, necessita una fase di istruzione del gestore stesso da fare in fase di configurazione del sistema e durante la richiesta. In particolare durante la fase di configurazione è necessario informare il gestore su quali siano i profili sui quali ha competenza, cioè a quale dominio esso è associato, e poi, durante l'esecuzione, contestualmente alla richiesta deve essere passato al gestore anche l'indirizzo a cui rivolgere la

richiesta.

4.3 Sistema di gestione dei profili dei dispositivi

Andiamo ora a presentare il sistema di gestione dei profili dei dispositivi. Come si è già detto questi devono essere gestiti in maniera centralizzata in quanto utenti diversi, ma soprattutto, di *domini* diversi potrebbero utilizzare i medesimi dispositivi, questo rende utile avere un sistema per accedere a questi in tutto il sistema in modo uniforme.

Per ottenere ciò si rende necessaria la presenza di un gestore centralizzato per la memorizzazione e l'accesso dei profili; nel nostro caso associeremo il gestore centralizzato al dominio radice della gerarchia SOMA e aggiungeremo, nella configurazione relativa ai vari gestori un riferimento a questo. Quanto detto viene illustrato nella seguente figura in cui abbiamo messo in evidenza con una freccia l'associazione tra il generico gestore e il gestore del dominio radice.

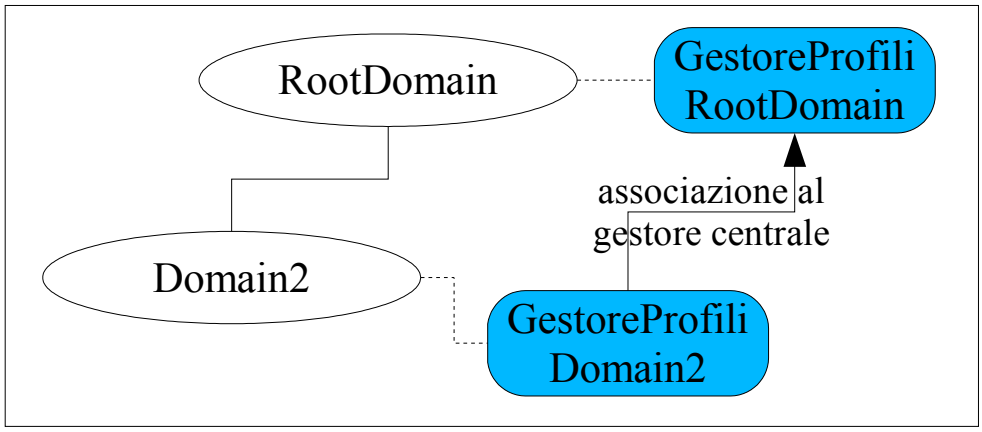

Figura 4.6 Associazione del gestore dei profili dei dispositivi al gestore del nodo radice

Ovviamente l'associazione qui presentata deve essere impostata in fase di configurazione del gestore dei profili dei dispositivi per il generico nodo.

Da notare che abbiamo scelto di associare il gestore centrale al nodo radice di SOMA ma questa non è una scelta obbligata, anzi potremmo addirittura pensare di considerarlo una entità distaccata dal sistema dei domini SOMA.

Si osserva inoltre come questa impostazione includa implicitamente il fatto che un generico gestore deve avere due comportamenti differenziati in base al fatto se sia o no il gestore principale:

1. nel caso sia il gestore principale deve risolvere immediatamente le richieste
2. nel caso sia uno dei gestori secondari deve passare attraverso quello centralizzato

La scelta di avere un gestore centrale a cui tutti si rivolgono è sicuramente la più

efficiente per quello che riguarda i tempi di risposta, tuttavia potrebbe congestionare il suddetto gestore, per evitare questo potremmo pensare a soluzioni alternative ad esempio con caching gerarchico tra i nodi gestori che implicitamente riproducono la gerarchia SOMA.

4.4 Sistema di gestione dei profili degli utenti

Per quello che riguarda la gestione dei profili degli utenti è sicuramente preferibile una scelta distribuita in cui il profilo dell'utente viene memorizzato sul gestore associato al nodo in cui localmente si trova ad operare l'utente. Questo consente il rispetto dei principi di località secondo cui una risorsa risulta localizzata nelle vicinanze dell'utente che ne fa uso.

Il problema maggiore associato a questa soluzione è il fatto che la gestione dei profili degli utenti è distribuita su vari nodi ed è quindi necessario una qualche forma di coordinamento per evitare di avere identificativi doppi nel sistema. Per tale forma di coordinamento ci viene incontro la struttura gerarchica dei *domini SOMA* infatti possiamo costruire un identificativo per l'utente componendo insieme l'identificatore locale, univoco nel dominio SOMA, con il *path*: il percorso dal dominio radice a quello considerato.

In SOMA un esempio di path potrebbe essere il seguente, ripreso dalla Figura 4.7:

/Root/Europa/Italia in su vengono presi in considerazione i nodi che portano dal nodo radice *Root* al nodo finale *Italia* in cui si registra il generico utente.

Questo consente di suddividere la gestione dell'univocità dell'identificatore tra i vari nodi: nel generico dominio viene verificata l'unicità del nome locale e come *nome globale* viene considerato il *nome locale* assieme al *path* del dominio di appartenenza.

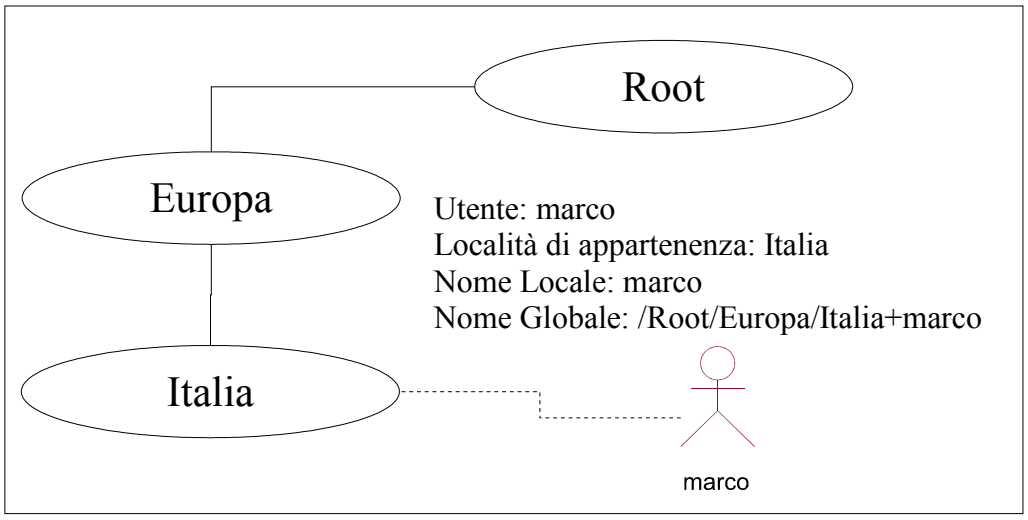

Figura 4.7 Nome Locale e Nome Globale del generico utente

Un sistema siffatto pone un altro problema ed è quello del reperimento del profilo utente

nel caso in cui costui non si trovi nella propria località.

Per risolvere questa problematica è necessario che l'utilizzatore conosca, oltre al suo nome globale, anche l'indirizzo dell'entità associata al proprio dominio di appartenenza.

Con queste informazioni l'utente effettua la richiesta all'interno del dominio generico, questo passa la richiesta al proprio gestore e insieme a questa informazione gli passa l'indirizzo dell'entità a cui rivolgere la richiesta. Questo viene mostrato in Figura 4.8

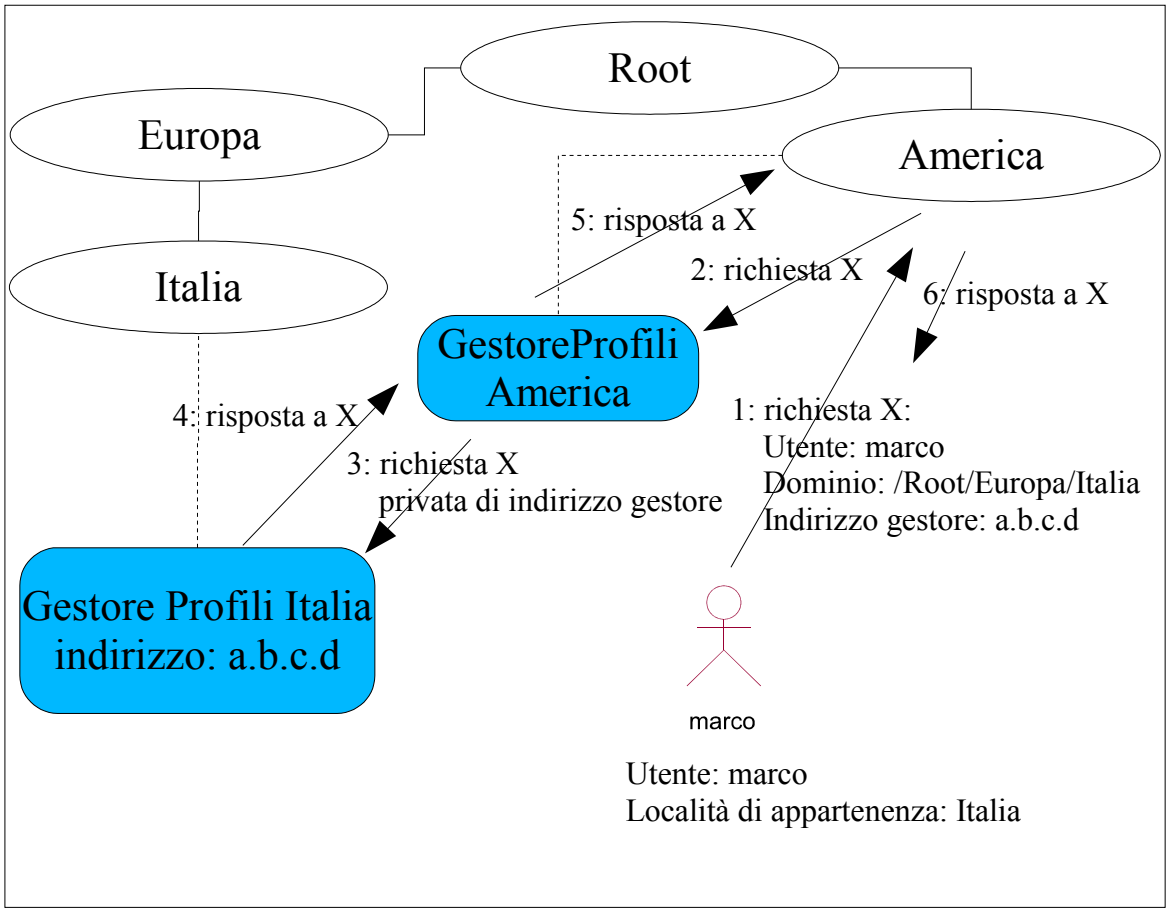

Figura 4.8 Risoluzione di profilo da località esterna

E' doveroso fare una ultima precisazione, il *path* dei domini SOMA di cui si parlato servono non solo a livello di gestione del sistema per avere identificatori globali unici ma anche all'interno del generico gestore di profili per distinguere i profili appartenenti a diversi domini, nel caso lo stesso sia associato a due o più domini SOMA.

Questo inoltre consente di trattare alcune caratteristiche interne al gestore dei profili in modo uniforme: infatti potremmo pensare di usare, internamente al gestore, una sorta di estensione del *path SOMA* ad esempio per i profili utente appendendo la stringa "/users" e per

i profili dei dispositivi e la stringa "/devices" in modo da usare lo stesso sottosistema di memorizzazione gestire sia i profili utente che quelli dei dispositivi.

4.5 Sistema per la composizione dei profili

Il sistema per la composizione dei profili è anch'esso delegato al gestore esterno il quale si fa carico di eseguire le seguenti operazioni:

1. reperimento del profilo del dispositivo indicato
2. reperimento del profilo dell'utente indicato
3. composizione dei profili recuperati

Le prime due fasi sono già state ampiamente dettagliate, per quello che riguarda la fase di composizione, questa deve essere fatta sul gestore e secondo certe politiche, e coinvolge le informazioni reperite nel profilo utente e quelle reperite nel profilo del dispositivo.

La gestione della composizione dei profili è un processo assai delicato in quanto non coinvolge solo aspetti quantitativi ma anche semantici, nel senso che le regole di composizione possono variare in base al significato degli attributi rappresentati.

Nel profilo utente potrebbe essere specificata la percentuale di banda da occupare, mentre nel profilo del dispositivo la banda massima supportata, in questo caso la composizione è a percentuale applicata alla banda del dispositivo.

In un altro esempio l'utente potrebbe specificare la lingua o l' *encoder* (identifica l'algoritmo di codifica usato nei filmati) preferiti, oppure metterli in ordine di preferenza, mentre nel dispositivo ne abbiamo un elenco, in questo caso la scelta potrebbe ricadere su quello preferito se presente.

Quindi essendo la composizione un processo la cui specifica è tipicamente algoritmica e codificata con qualche linguaggio, lasceremo questo compito a un modulo codificato contenuto all'interno del gestore dei profili.

4.6 Conclusioni

In questo capitolo abbiamo introdotto la struttura, in termini di protocolli ed entità, che andremo a sviluppare in dettaglio nel prossimo capitolo, necessaria all'introduzione di un sistema di gestione di profili nell'architettura MUM con particolare attenzione alla struttura gerarchica dei *domini SOMA* su cui abbiamo basato alcune caratteristiche del sistema. Resta comunque da osservare come il sistema sia quasi indipendente dall'architettura MUM in cui viene ad integrarsi.

5 Progetto

Nel capitolo che andiamo a delineare descriviamo nel dettaglio le entità necessarie allo sviluppo del sistema in termini di comunicazione tra queste e dati di supporto necessari. Inoltre daremo rapporto astratto sulle proprietà di queste strutture e sulle funzionalità che devono mettere a disposizione. Concluderemo presentando come nel sistema possa essere introdotta una qualche forma di caching.

5.1 Progetto dell'architettura del sistema

Presenteremo le entità che operano nel sistema raggruppandole mediante l'astrazione e package. Le entità di cui parleremo sono oggetti che nel sistema accorpano operazioni e dati su cui operano; i package rappresentano una sorta di contenitori per tali componenti, per cui è possibile mettere insieme quelli che risultano affini, da un punto di vista concettuale.

La seguente figura mostra questi package, evidenziando la struttura degli stessi in termini di inclusione.

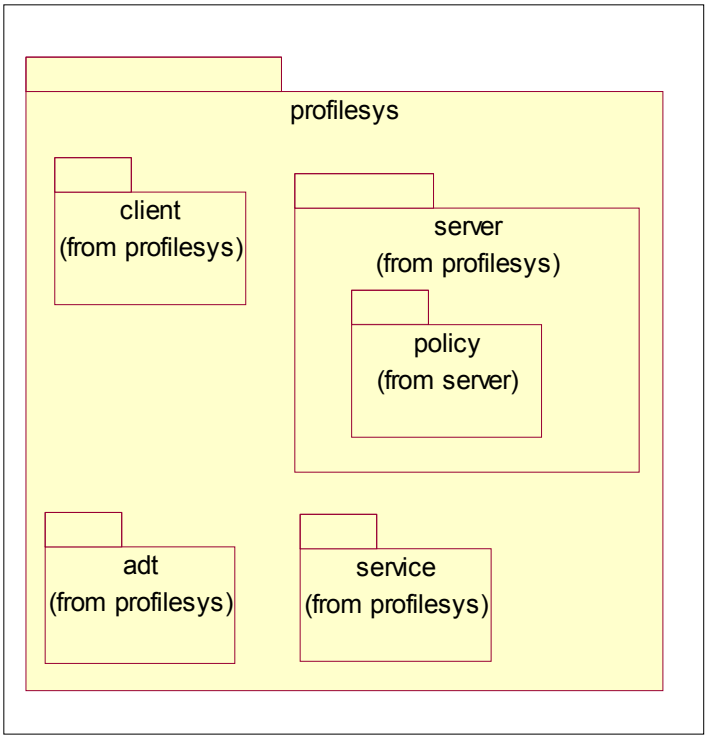

Figura 5.1 Architettura generale del sistema

I package da noi individuati sono i seguenti:

- il package profilesys che rappresenta il contenitore entro cui è sviluppato il progetto.
- il package profilesys.server il quale contiene i componenti deputati alle azioni da

compiere sull'entità esterna, che da ora in poi chiameremo server; qui troveremo tutte gli oggetti necessari alla gestione, in termini di operazioni interne, che il server deve esporre verso l'esterno.

- il package profilesys.server.policy che contiene le entità che rappresentano le regole per eseguire la risoluzione del profilo dato un profilo utente e un profilo di dispositivo.
- il package profilesys.client il quale contiene tutti i componenti utili per l'interfacciamento col server e la risoluzione delle richieste che dovranno andare dal sistema interno a quello esterno e il reperimento delle risposte.
- il package profilesys.adt che contiene le entità utilizzate per trattare i dati all'interno del nostro sistema, cioè i profili dell'utente e quelli dei dispositivi.

5.1.1 Visione generale delle entità per la comunicazione

Nel sistema sviluppato vi sono due punti di vista:

- punto di vista del client: colui che è interessato a eseguire operazioni sul sistema senza preoccuparsi del protocollo di comunicazione; questo è la parte che dall'interno di MUM ci porterà a comunicare col servizio di profili.
- punto di vista del server: rappresenta l'insieme delle operazioni disponibili sul server e che costui deve eseguire. Si rende necessario a questo un meccanismo per cui i messaggi inviati verso di lui dal client, dovranno essere tramutati in azioni e relative risposte verso il client.

Come riflesso di questa architettura abbiamo nei package porfilesys.server e pofilesys.client delle entità che espongono la medesima *interfaccia logica* in termini di operazioni accessibili. Nel primo ci sono i componenti deputati all'esecuzione dei servizi offerti, nel secondo abbiamo gli oggetti di interfacciamento col protocollo e risoluzione dei servizi. Questa dualità è espressa anche nel nome dato a tali entità, abbiamo ad esempio un oggetto ClientServerManager lato client per eseguire le operazioni di amministrazione sul server, e l'omologo ServerManager che contiene le operazioni da eseguire sul server.

La seguente figura mostra quanto detto, vale a dire quale sia il punto di vista lato client che lato server, inoltre mette in evidenza l'associazione tra i componenti che implementano le funzionalità rese disponibili dal sistema; infine viene presentata, mediante delle frecce, la visualizzazione del cammino logico che attraversa una generica richiesta a partire dal client fino al server dove questa richiesta viene eseguita.

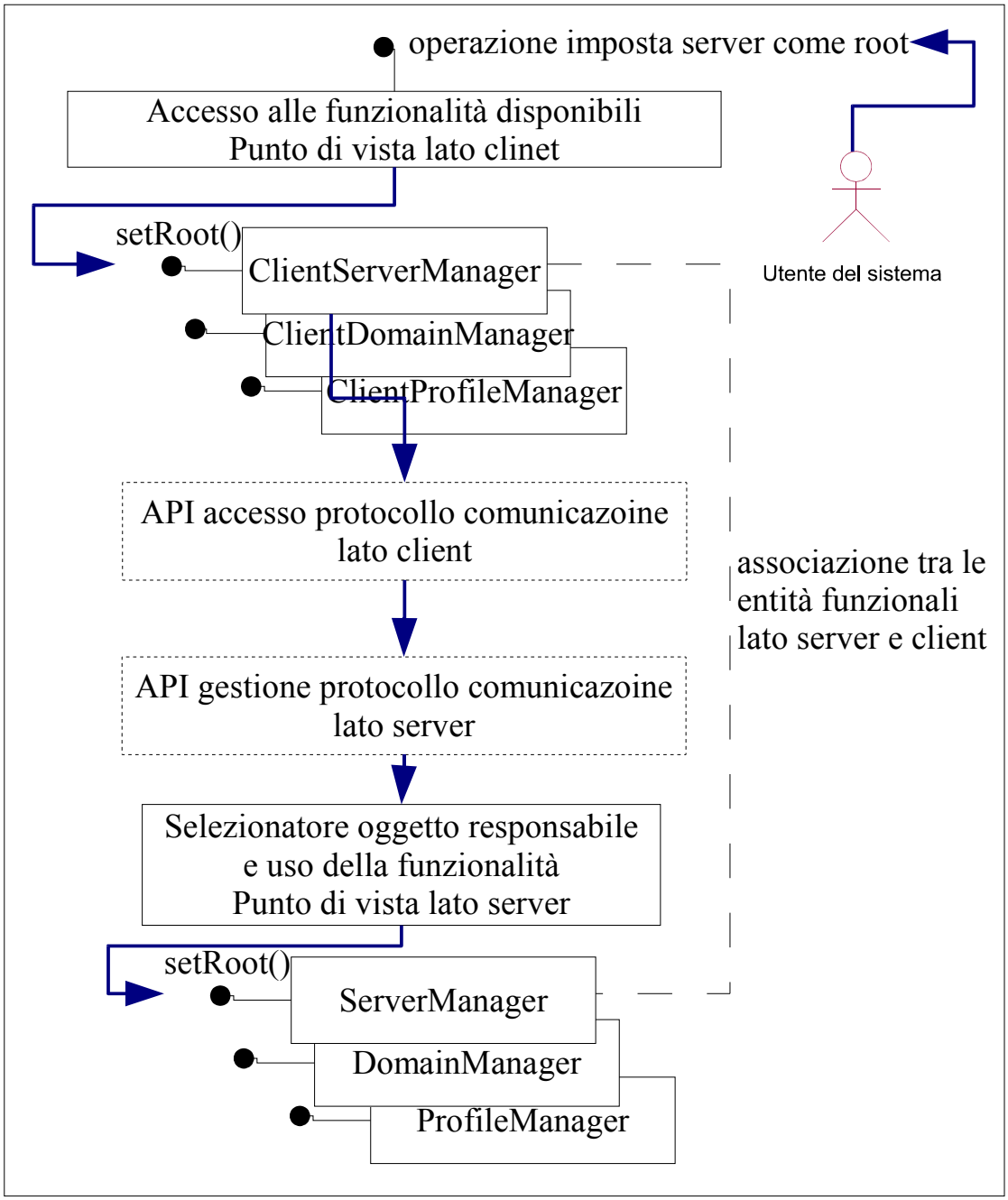

Figura 5.2 Catena delle chiamate, associazione tra gli oggetti e punti di vista client e server

5.1.2 Servizi offerti dal sistema di gestione dei profili

Il sistema di gestione dei profili deve mettere a disposizione le funzionalità esposte qui di seguito:

deve essere possibile, per un gestore dei profili essere impostato come gestore centrale per i profili dei dispositivi oppure come gestore che faccia da proxy che reperisca dal gestore centrale i profili; quest'ultimo come visto in precedenza, è associato al nodo radice di SOMA.

Questa funzionalità viene realizzata dal componente ServerManager che deve esporre le funzionalità mostrate nella seguente tabella come se fossero delle definizioni di metodi.

funzionalità	**descrizione**
isRoot():Boolean	verifica che sia il gestore centrale
setRoo()	imposta come gestore centrale
setRootAddress(RootAddress)	imposta il riferimento al gestore centrale
getRootAddress():RootAddress	accede al riferimento al gestore centrale

Deve essere in grado di gestire gli indirizzi utente per diversi domini SOMA e distinguere i profili in base al dominio e al nome dell'utente, per questo è necessario realizzare un sistema per la gestione dei nomi dei domini SOMA che verranno identificati con il relativo path.

La tabella seguente mostra le funzionalità offerte.

funzionalità	**descrizione**
existDomain(Domain):Bolean	verifica se il dominio viene gestito dal server
listDomain():Domain[]	elenca i domini gestiti dal server
addDomain(Domain)	aggiunge un dominio a quelli gestiti dal server
delDomain(Domain)	rimuove un dominio da quelli gestiti dal server

Deve essere possibile inserire, rimuovere, ed accedere ai profili, per questo vengono presentate le funzionalità di gestione dei profili utente e dei profili dei dispositivi insieme con l'accorgimento che nel caso di profili utente l'identificatore del profilo deve contenere anche il dominio SOMA di appartenenza e nella richiesta deve essere aggiunto l'indirizzo della macchina a cui fare la richiesta nel caso il dominio non sia tra quelli gestiti dal server a cui viene fatta la richiesta.

Presentiamo le funzionalità di base necessarie nella seguente tabella.

funzionalità	descrizione
getProfile(ProfileID):Profile	accesso al profilo
addProfile(ProfileID,Profile)	aggiunta di un profilo
listProfile():ProfileID[]	accesso all'elenco dei profili
delProfile(ProfileID)	rimozione di un profilo

Infine l'ultima funzionalità che viene esposta dal sistema deve essere quella per la risoluzione dei profili. Questa viene mostrata nella seguente tabella

funzionalità	descrizione
resolveProfile(UserProfileID,DeviceProfileID): DeviceProfileID	crea un profilo per dispositivo riorganizzato in base alle preferenze esposte nel profilo utente

Ciascuna delle funzionalità esposte fino ad ora viene quindi realizzata nel sistema da una coppia di oggetti: lato server avremo il componente che risponde compiendo le azioni necessarie a soddisfare la richiesta del client mentre lato client avremo un componente che è responsabile dell'interfacciamento con il protocollo di comunicazione. Questo è quanto mostrato dalla seguente figura.

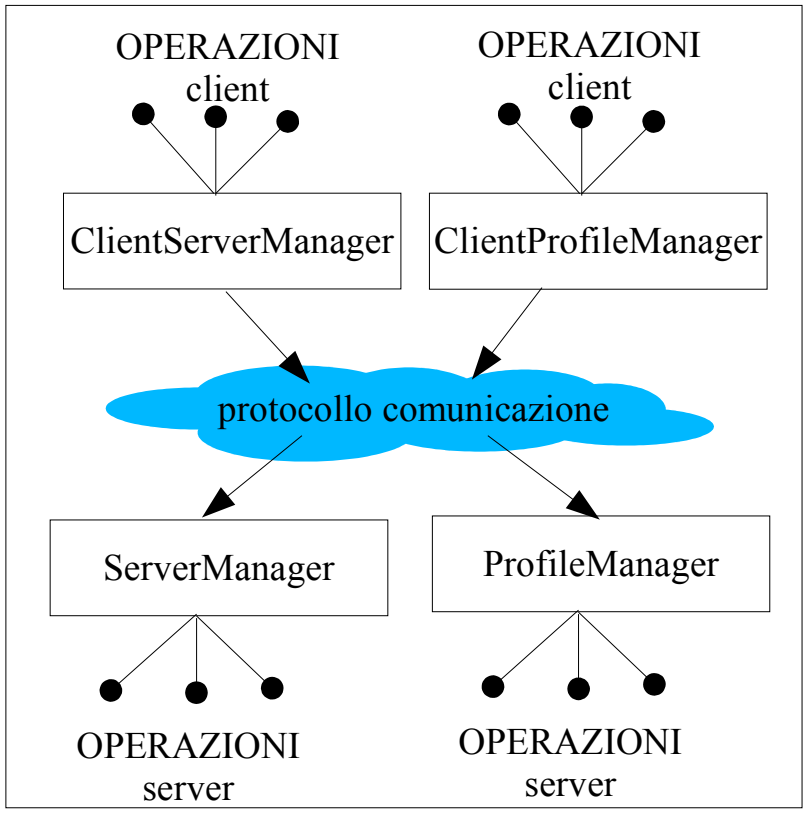

Figura 5.3 Oggetti di supporto lato client e lato server

5.1.3 Supporto alla permanenza dei dati e della configurazione

Per permanenza dei dati si intende il fatto che, non solo i profili devono essere memorizzati in modo permanente, ma anche le impostazioni effettuate sul sistema devono permanere anche a un successivo riavvio dello stesso; questo significa che, se sul server ho impostato ad esempio l'indirizzo del dominio root, responsabile dei profili dei dispositivi, al successivo riavvio non deve essere necessario reimpostarlo. Per realizzare questo è necessario un sistema di salvataggio dei dati che riguardano il controllo del sistema e deve essere inoltre svolto il salvataggio delle modifiche contestualmente ad ogni modifica. Il fatto di salvare immediatamente le modifiche rappresenta quella che viene definita una politica di tipo *write-through* che consente di evitare problemi relativi all'inconsistenza dei dati.

Un'altra cosa da osservare è che la modifica di tali dati di controllo va fatta agendo direttamente sul server associato al nodo in cui facciamo l'accesso. In pratica viene consentito il controllo dei server solo accedendo a questi direttamente per quello che riguarda la gestione del server stesso e la gestione dei domini; la gestione dei profili invece consente di accedere al profilo remoto anche da altro server, ma non la modifica. Questo si sposa con il concetto che

l'utente deve compiere le operazioni importanti, cioè la modifica del proprio profilo, o la gestione del server, solo se ha accesso diretto a questo in modo da evitare problemi dovuti a malintenzionati. Una soluzione alternativa poteva essere quella di studiare un protocollo di autenticazione, ma questo esula dal presente progetto di tesi.

Infine è necessario evidenziare che per i propositi suddetti di memorizzazione della configurazione e dei dati nel sistema che andremo a costruire saranno necessarie delle entità aggiuntive di cui non discutiamo in sede di progetto, poiché legate alla tecnologia implementativa. Tuttavia possiamo cominciare col dire che serviranno comunque delle funzionalità di memorizzazione per quelle parti relative a servizi che necessitano sia mantenuto uno stato sulla scelta fatta, queste parti, al di là dei profili, di cui comunque deve essere gestita la memorizzazione, sono quelle relative all'impostazione del server per quello che riguarda i profili dei dispositivi, come server centrale o come proxy, e relative alla gestione dei domini, infatti è necessario mantenere memoria di quali domini sono quelli gestiti da ciascun server.

5.2 Entità e ruoli nel sistema di gestione dei profili

Nel sistema presentato abbiamo diversi oggetti che rappresentano l'interfaccia per la comunicazione, il comportamento o la rappresentazione dei dati. Di seguito metteremo in evidenza questi componenti raggruppandoli in modo congruente alla struttura a package presentata all'inizio del capitolo. Nella seguente figura mettiamo in evidenza le parti relative al sistema di gestione dei profili e presentiamo i componenti e la loro ubicazione in termini di blocchi funzionali. Nella figura viene altresì evidenziata la corrispondenza tra gli oggetti usate dal client e quelle usate lato server.

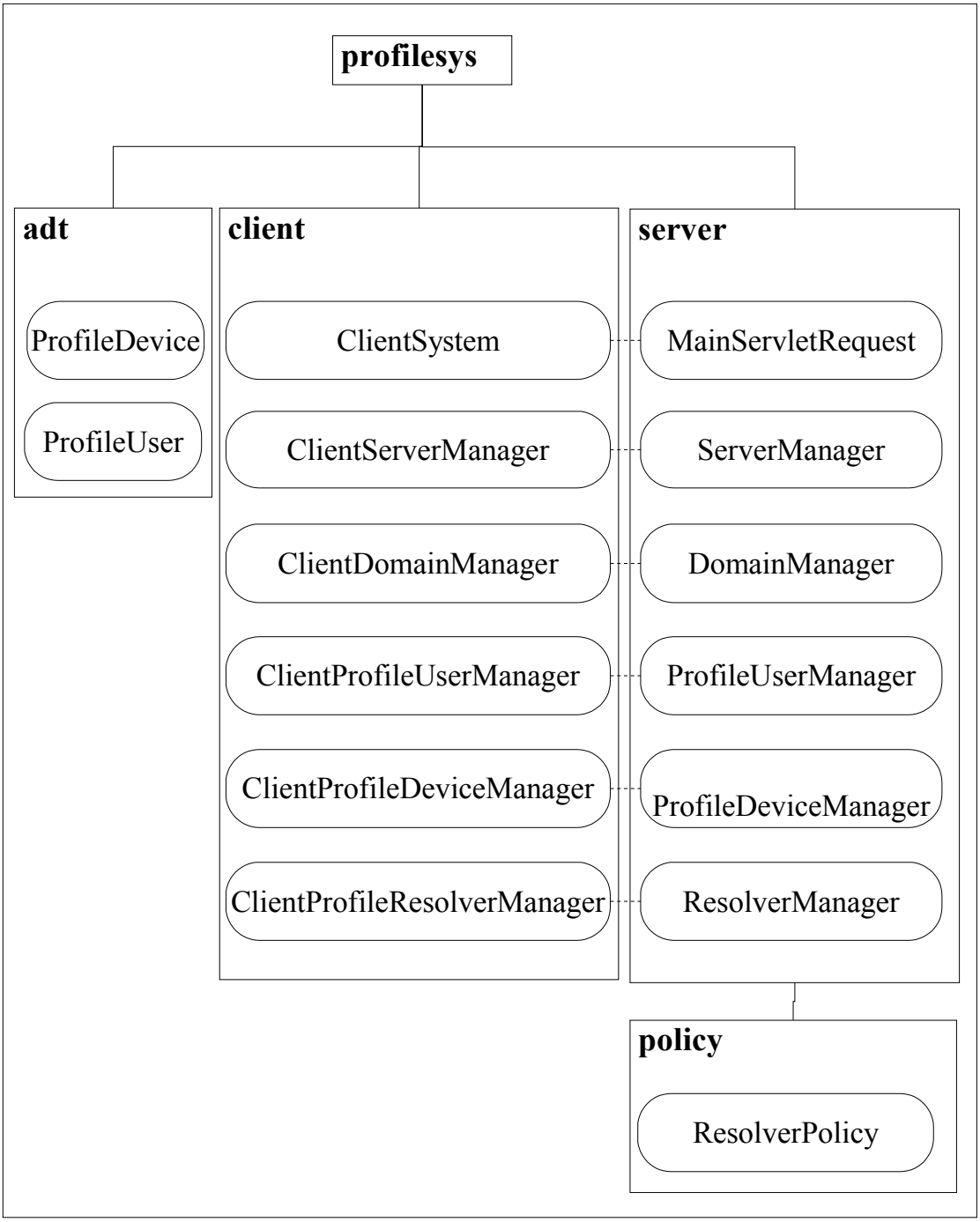

Figura 5.4 Presentazione degli oggetti e struttura del sistema di gestione dei profili

Presentiamo di seguito l'elenco dei componenti necessari e la descrizione degli stessi raggruppati secondo i blocchi funzionali che nel sistema vengono rappresentati come package.

5.2.1 Componenti di supporto per il client

Andiamo adesso a descrivere gli oggetti di supporto per le richieste che il client può compiere:

- ClientSystem: rappresenta l'oggetto di interfaccia, a basso livello, al protocollo usato per l'invio delle richieste e la ricezione delle risposte, per questo motivo verrà dettagliato nella fase di implementazione; per ora è sufficiente dire che tutti i componenti presenti nel package profilesys.client ereditano il proprio comportamento da questo e ne utilizzano le funzionalità esposte per accedere al protocollo sottostante.
- ClientServerManager: consente l'impostazione del server locale per usare un server remoto come server dei profili di dispositivo, oppure per la gestione esso stesso come server dei dispositivi.
- ClientDomainManager: consente l'uso solo del server associato al dominio SOMA da cui avviene l'accesso al sistema e permette di informare il server su quali domini abbia competenza.
- ClientProfileUserManager: questo componente espone i metodi di accesso al servizio di gestione dei profili utente quindi presenterà i metodi di inserimento, rimozione, accesso ai profili. Ricordiamo che l'inserimento e la rimozione devono essere consentiti solo se l'accesso al server è diretto. Inoltre si evidenzia il fatto che per effettuare una tale richiesta l'utente deve conoscere: il proprio dominio SOMA, l'indirizzo del server competente oltre ovviamente al proprio username locale.
- ClientProfileDeviceManager: questo si comporta assolutamente come il precedente componente esposto, però il sistema non richiede che venga usato, come parametro, né il dominio né l'indirizzo del server competente, in quanto questo indirizzo viene impostato sul server nella fase di configurazione.
- ClientResolverManager: consente l'accesso alla risoluzione dei profili; in sostanza si occupa di preparare una richiesta del tutto uguale a quella per l'accesso al profilo utente con in più un identificatore di profilo di dispositivo, il risultato dell'unica operazione associata a questo componente è il profilo del dispositivo personalizzato per l'utente.

5.2.2 Componenti di supporto per il server

I componenti predisposti per il server sono a livello di interfaccia concettualmente identici a quelli a livello client, tuttavia realizzano le operazioni direttamente, mentre quelli a livello client rappresentano solo un *front-end* al servizio per la gestione del protocollo di comunicazione.

- MainServletRequest questo componente anticipa nel nome il sistema di gestione

utilizzato che sarà quello delle *servlet* e per tale motivo verrà specificato in seguito; per ora è sufficiente dire che il meccanismo usato per la gestione dei comandi è molto simile al modello *event-driven* in cui questo oggetto fa' il ruolo di gestore degli eventi, i quali gli vengono automaticamente recapitati dall'infrastruttura; tali eventi altro non sono che richieste HTTP.

- ServerManager: fornisce il sistema di impostazione per il funzionamento come server centrale per la gestione dei profili dei dispositivi; ricordiamo che tale impostazione dovrà essere permanente e quindi deve essere prevista una fase di memorizzazione dell'impostazione contestualmente all'applicazione della stessa.
- DomainManager: questo componente consente di impostare il server corrente per la gestione di più domini, per tale impostazione è necessario accedere direttamente al server di competenza, e quindi se accedo tramite MUM devo accedere dal dominio SOMA associato a questo server.
- ProfileUserManager: questo componente si occupa di reperire, e memorizzare i profili utente; per la memorizzazione deve essere necessario accedere direttamente al server, per il reperimento deve comportarsi con accesso diretto se il dominio specificato nella richiesta è di competenza di tale server, oppure deve avere un comportamento da proxy nel caso di accesso a un profilo utente il cui dominio non è gestito da questo server. Da osservare che, per il comportamento da proxy, viene usato il componente ClientProfileUserManager.
- ProfileDeviceManager: questo componente si occupa di reperire il profilo del dispositivo, come nel caso precedente se vogliamo inserire i profili dobbiamo accedere direttamente dal nodo radice; per il reperimento delle informazioni il server deve: fornire direttamente il profilo, se è il server radice, altrimenti deve agire da proxy. Da osservare che per il comportamento come proxy viene riutilizzata il componente ClientProfileDeviceManager.
- ResolverManager: questo componente si occupa di reperire il profilo utente, quello del dispositivo e con questi, usando l'oggetto ResolverPolicy, ottenere il profilo del dispositivo personalizzato per l'utente.

5.2.3 Componenti di supporto alla tecnica di profilazione

Esiste una unico componente di supporto a tale servizio ed è il seguente:

- ResolverPolicy: si occupa semplicemente di trasformare il profilo del dispositivo in un

altro profilo di dispositivo compatibile con le preferenze espresse dall'utente secondo le regole imposte. Questo oggetto espone una sola funzionalità per realizzare la risoluzione del profilo del dispositivo col profilo utente; è stata pensata per poter personalizzare la profilazione in quanto potremmo prevedere una entità che risolve il profilo sulla base si una politica specificata esternamente e interpretata da questo componente.

5.2.4 Oggetti di supporto alla rappresentazione dei dati

Come oggetti di supporto alla rappresentazione dei dati sono previsti fondamentalmente due componenti:

- ProfileUser: questo rappresenta il profilo utente. Oltre alle informazioni circa le preferenze sul dispositivo è previsto una proprietà che contiene l'elenco dei dispositivi attualmente usati dall'utente. Questo potrebbe servire eventualmente per associare una posizione al dispositivo nell'ambiente.
- ProfileDevice: questo rappresenta il profilo del dispositivo, al suo interno troveremo come attributi i parametri che rappresentano le capability del dispositivo.

Questi oggetti oltre ai profili in un formato interno contengono anche le procedure per la trasformazione degli stessi in formato CC/PP un formato compatibile con i moderni standard di rappresentazione.

Oltre alle normali informazioni abbiamo inserito all'interno del profili stessi alcune meta-informazioni sullo stesso quali un identificatore del profilo e la data di inserimento. Questi dati potrebbero essere usati successivamente a scopo di caching.

5.3 Aspetti comportamentali del sistema di gestione dei profili

In questo paragrafo illustreremo in breve il comportamento che ci aspettiamo dal sistema di profili nel caso di accesso alle varie funzionalità e presentiamo dei piccoli diagrammi che ne consentano una migliore comprensione.

5.3.1 Accesso alle funzionalità di servizio

Per funzionalità di servizio intendiamo tutte quelle operazioni quali l'impostazione dei domini da gestire, o dell'indirizzo del server centrale per i dispositivi, che rappresentano la fase di configurazione del sistema.

Come abbiamo già detto per poter usare tali funzionalità è richiesto che l'utente acceda direttamente al server competente, questo sia per problemi di concorrenza nell'accesso ai dati, sia per una primordiale forma di sicurezza.

Lo schema seguente mostra come vengono gestite le modifiche delle impostazioni del

sistema che, come precedente accennato seguono una politica di tipo *write-through*, in pratica si esegue la modifica e in modo atomico di compie il salvataggio della nuova configurazione.

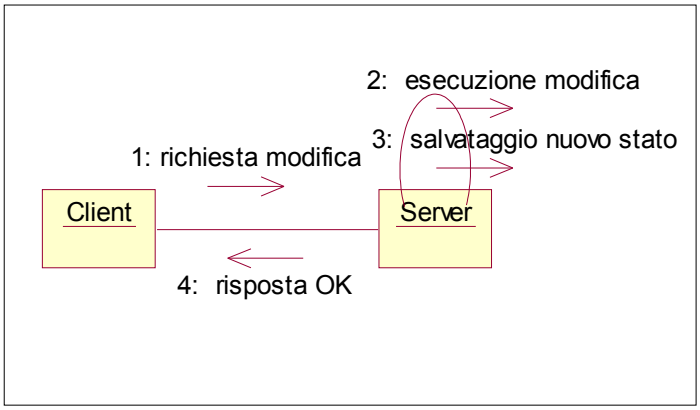

Figura 5.5 Protocollo di esecuzione comandi di controllo

Un altro processo importante è la fase di ripristino della configurazione all'avvio del sistema, infatti deve essere previsto, all'avvio del sistema, una fase di ripristino in modo che il sistema mantenga la configurazione impostata una volta per tutte. Questo è mostrato nel seguente diagramma dove, per completezza, abbiamo anche rappresentato il possibile stato *non configurato* il cui passaggio, allo stato configurato, avviene ad opera di una interazione con un utente esterno.

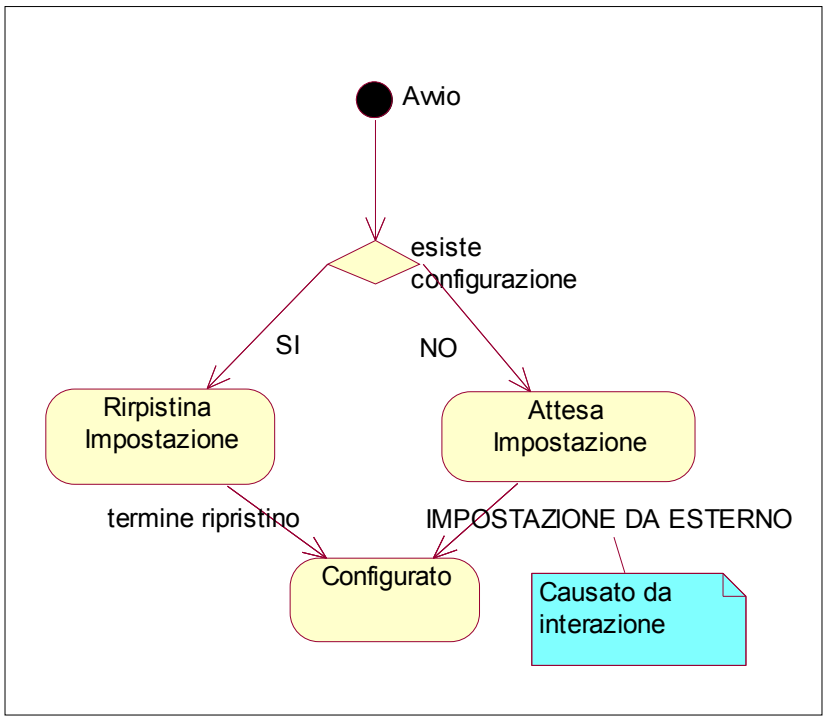

Figura 5.6 Ripristino impostazioni all'avvio

5.3.2 Accesso alle funzionalità di reperimento e risoluzione dei profili

Per quello che riguarda le funzionalità di accesso ai profili ed alla risoluzione degli stessi il sistema, in entrambi i casi, sia che si tratti di profili utente, sia che si tratti di profili di dispositivo, deve assumere *automaticamente* un comportamento da proxy o da gestore in base al fatto che sia o meno il server centrale, nel caso di dispositivi, o quello competente per il dominio, nel caso di profili utente.

Per il discorso della memorizzazione dei profili viene proposto lo stesso vincolo adottato per le istruzioni di controllo, e cioè l'utente può aggiungere profili a patto che acceda direttamente al server competente. Anche qui la politica seguita nel salvataggio dei dati è di tipo *write-through*.

Per la risoluzione dei profili, come precedentemente accennato, al server vengono passati solo degli indirizzi e degli identificatori di profilo, è poi il server a risolverli prelevando, se necessario, i profili dai nodi di competenza.

Nella figura che segue mostriamo un esempio di richiesta in cui il profilo utente e quello del dispositivo vengono prelevati e combinati sul server di competenza del nodo SOMA da cui parte la richiesta.

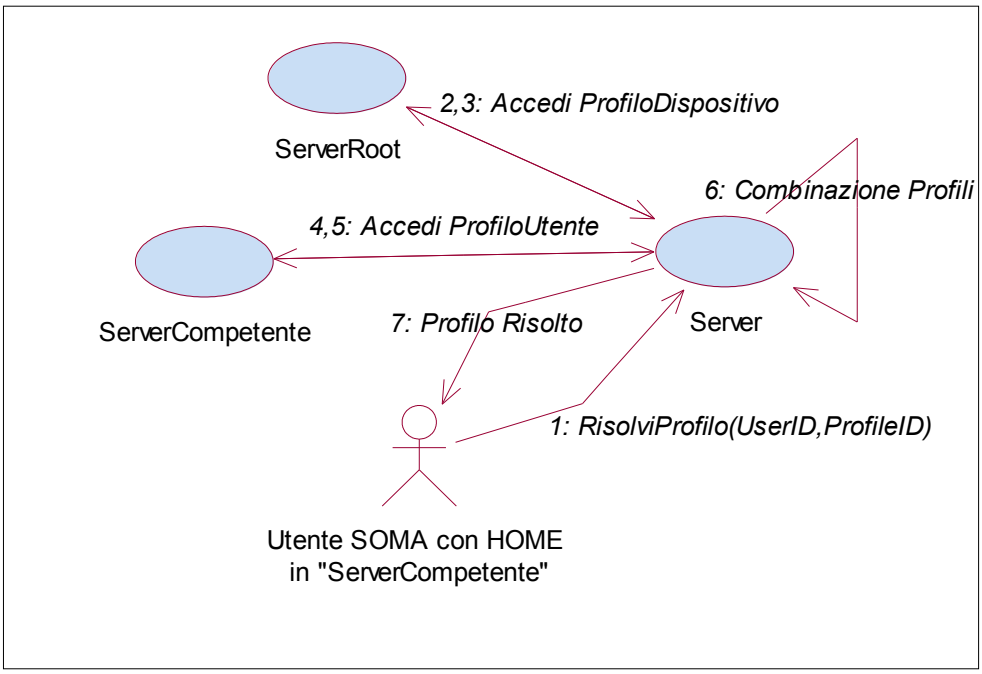

Figura 5.7 Risoluzione Profilo

5.4 Introduzione del caching

Uno degli aspetti salienti del sistema sviluppato è la possibilità di introdurre del

caching, per quello che riguarda il reperimento dei profili, senza minimamente modificare l'architettura e senza nemmeno modificare il sistema di accesso al servizio realizzato. Scopo dell'introduzione del caching all'interno del sistema è soprattutto migliorare le prestazioni circa il reperimento dei profili, ma non solo, se prendiamo infatti in esame il sistema adottato per il reperimento dei profili dei dispositivi ci si accorge subito come questo vada a sovraccaricare eccessivamente il server centrale, al quale arrivano tutte le richieste relative ai profili dei dispositivi, e come quindi un sistema di caching ridurrebbe il carico su quest'ultimo.

Presentiamo di seguito le linee guida per l'introduzione del caching nel sistema realizzato.

Per quello che riguarda la gestione dei profili il caching potrebbe essere basato sui domini, ad esempio quando chiedo un profilo utente ad un server che per tale dominio non è responsabile, questo potrebbe eseguire il caching del profilo stesso sulla base del dominio considerato. In questo modo potrei eseguire caching basato sul dominio, anzi potrei addirittura pre-caricare tutti i profili per tale dominio.

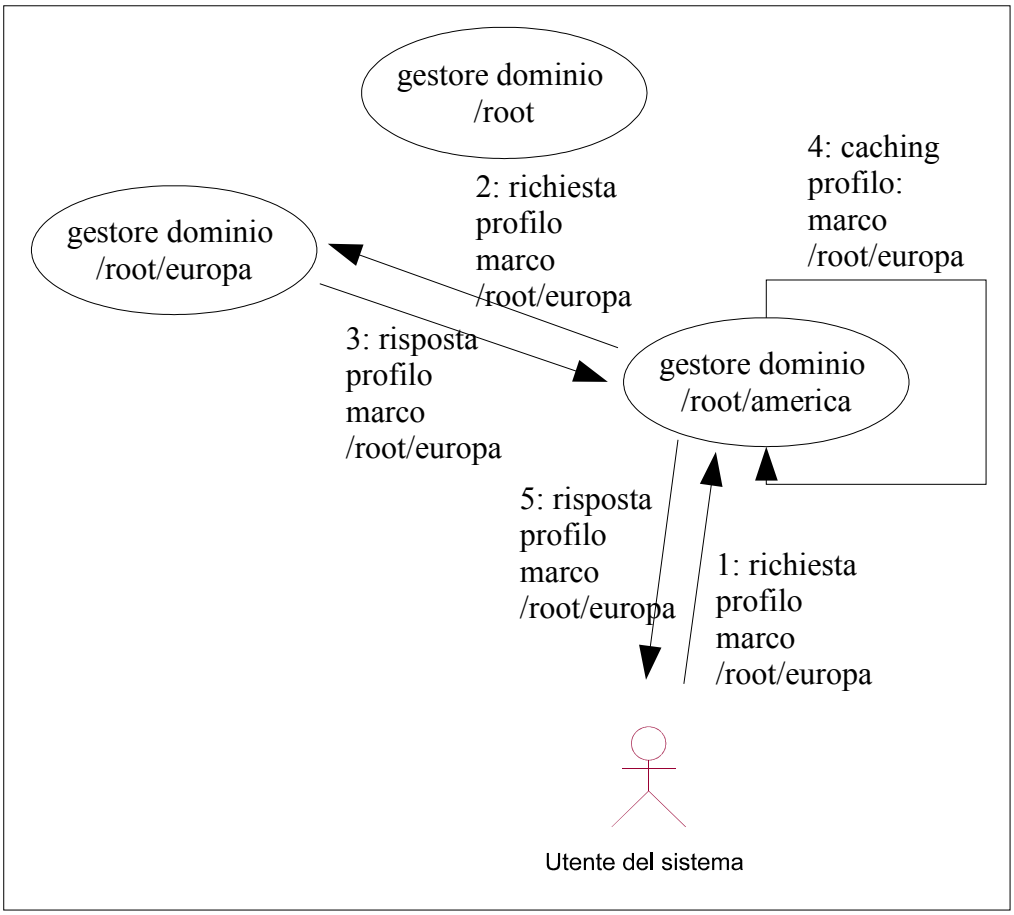

Figura 5.8 Gestione profili utente con caching

Per quello che riguarda il caching dei profili dei dispositivi le alternative più interessanti sono due:

- caching gerarchico: tra i server sussiste una gerarchia che ricalca quella dei domini SOMA e ad una richiesta di un profilo di dispositivo questo potrebbe essere richiesto al nodo parent il quale a sua volta lo richiede al parent e così via; poi il profilo richiesto verrebbe memorizzato su tutti i domini attraversati dalla richiesta. In questo modo a richieste successive i nodi intermedi avrebbero già pronto il profilo richiesto. La seguente figura mostra quanto enunciato circa il comportamento del caching gerarchico.

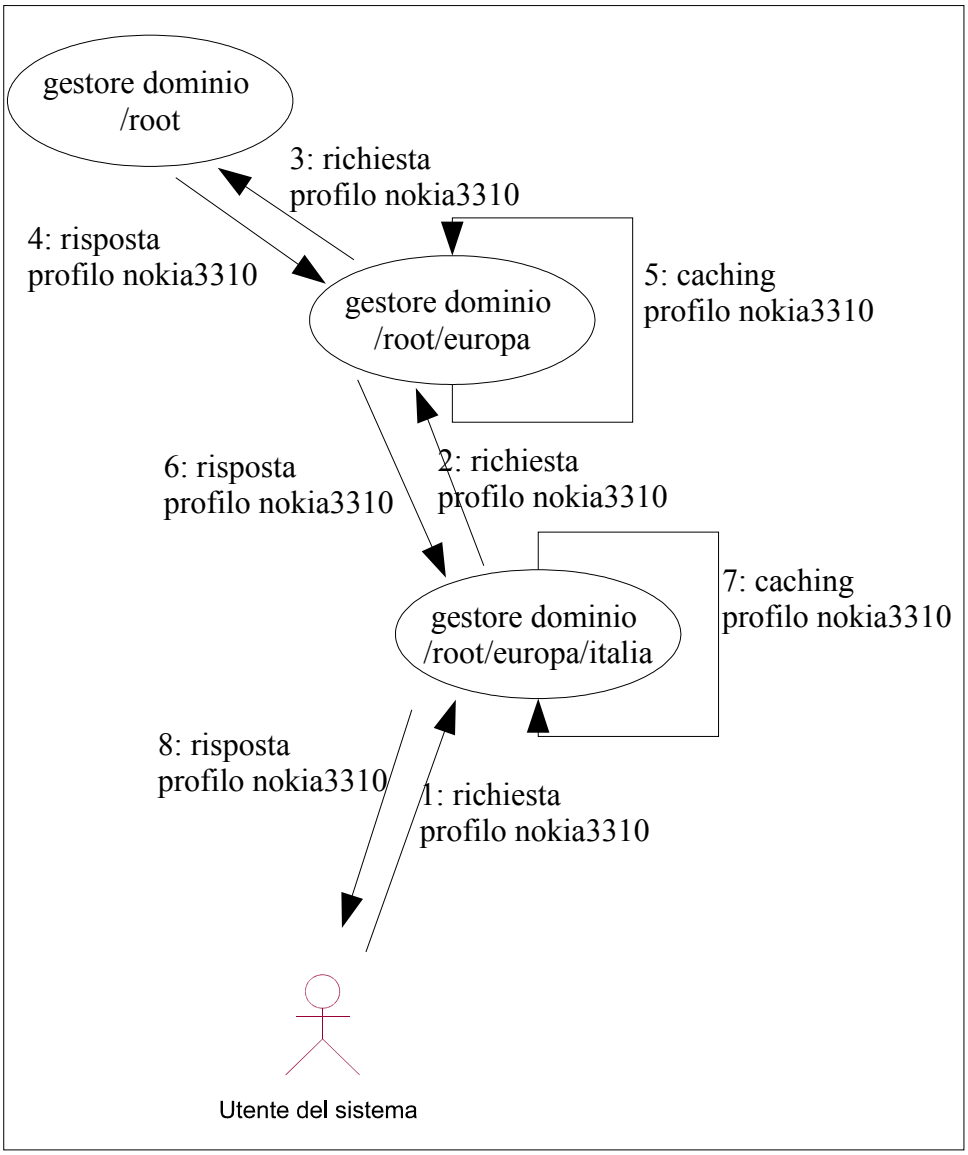

Figura 5.9 Caching gerarchico per i profili dei dispositivi

- caching centralizzato: alla prima richiesta il nodo si rivolge al nodo root e poi memorizza il profilo per usi successivi. Questa soluzione ha lo svantaggio di appesantire ancora una volta il server centrale, anche se solo per una richiesta, tuttavia risulta più veloce nel fornire la risposta per la prima richiesta in quanto non introduce una lunga catena di chiamate.

  La figura seguente mostra come si comporta il sistema per quello che riguarda un sistema di caching centralizzato nel caso della prima richiesta.

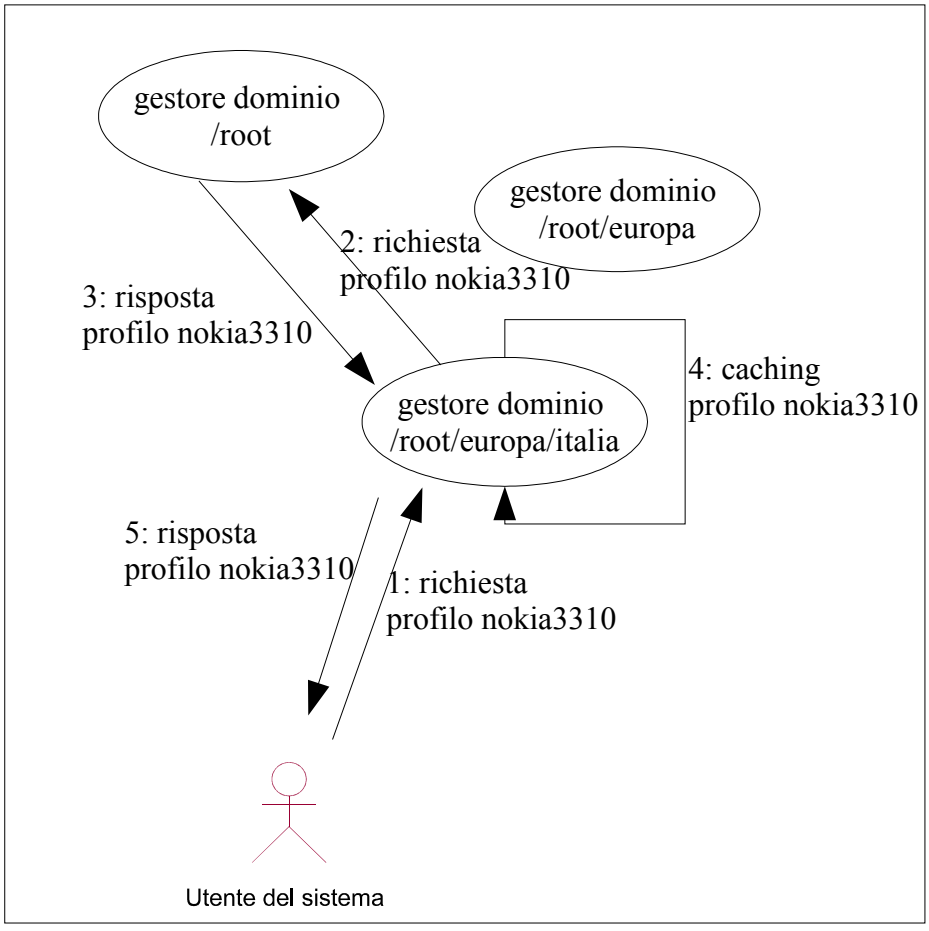

Figura 5.10 Caching centralizzato per i profili dei dispositivi

Una tecnica di caching introdurrebbe inoltre altre problematiche relative all'aggiornamento della cache. Questo problema potrebbe essere affrontato in modo attivo da parte del server o da parte dei client: se da parte del server è lui che si deve preoccupare di informare i cache server delle modifiche; se da parte del client è lui a doversi preoccupare, ad ogni richiesta, di verificare la consistenza del profilo di cui detiene una copia.

Le due tecniche enunciate, per l'aggiornamento della cache, hanno in entrambi i casi lo

svantaggio di sovraccaricare il mezzo di comunicazione, mentre per quello che riguarda il carico computazionale viene addossato nei due casi una volta al server, ed un'altra al client. Una possibile soluzione alternativa potrebbe essere quella di dotare i profili di una *data di scadenza* (*Time To Live*) trascorsa la quale l'elemento in cache è da reputare non valido e va aggiornato. Questa soluzione potrebbe essere un buon compromesso per quello che riguarda la gestione dell'aggiornamento della cache.

Per quello che riguarda la struttura del sistema in cui venga introdotto un sistema di cache è necessario sdoppiare le funzionalità atte al reperimento dei profili su due livelli:

- a basso livello avrei le funzionalità di base con visibilità e gestione della cache, in cui le tipiche funzionalità potrebbero essere: reperimento del profilo dall'esterno, memorizzazione in cache del profilo, aggiornamento e verifica dello stato del profilo.
- ad alto livello il sistema rimarrebbe praticamente identico a livello di interfaccia; cambia solo la modalità di gestione delle richieste dei profili quando questi devono essere reperiti esternamente, per l'utilizzo del sottostante livello di cache.

5.5 Conclusioni

Nel capitolo qui concluso sono state evidenziate le caratteristiche architetturali del sistema in modo il più astratto possibile rispetto a quello che è la piattaforma di appoggio e il linguaggio di sviluppo. In particolare abbiamo delineato un raggruppamento delle entità in gioco in termini di package. Sono state presentate le entità come componenti e funzionalità messe a disposizione dagli stessi. Si sono date le tracce per quello che riguarda gli aspetti comportamentali, e di protocollo pur senza entrare in una specifica precisa. Infine abbiamo posto le basi per l'introduzione, nel sistema presentato, di un meccanismo di caching dei profili.

6 Tecnologie implementative utilizzate

In questo capitolo descriveremo le tecnologie utilizzate nella fase di sviluppo con particolare attenzione per l' API [JSR188] utilizzata per il supporto a CC/PP.

6.1 Introduzione ad HTTP

Facciamo in questo paragrafo un brave introduzione al protocollo HTTP in quanto serve per meglio comprendere le discussioni successive.

HTTP (HiperText Transfer Protocol) definito definito negli [RFC1495] e [RFC2068] ed è fondamentalmente un protocollo di tipo *request-response* codificato in ASCII; inizialmente pensato per la fruizione delle pagine HTML, oggi estesosi per gestire il trasferimento di qualsivoglia tipo di informazione.

Il fatto che sia di tipo *request-response* significa che abbiamo due entità in gioco, il *client* che invia la richiesta,cioè il browser, e il *server* che la riceve e risponde.

Questo viene mostrato nella seguente figura in cui peraltro mostriamo il formato della richiesta e della risposta. In entrambi i casi abbiamo una intestazione in cui la prima riga rappresenta il comando richiesto al server, la risorsa da reperire, il protocollo associato, nel caso della richiesta e, nel caso della risposta, lo stato di esecuzione codificato numericamente seguito da una stringa descrittiva e preceduto dalla versione del protocollo, seguono una serie di campi del tipo "**campo:valore**", uno per ogni riga, che possono essere anche ripetuti, e infine, dopo l'ultimo campo, abbiamo una riga vuota che delimita l'inizio del corpo della richiesta/risposta.

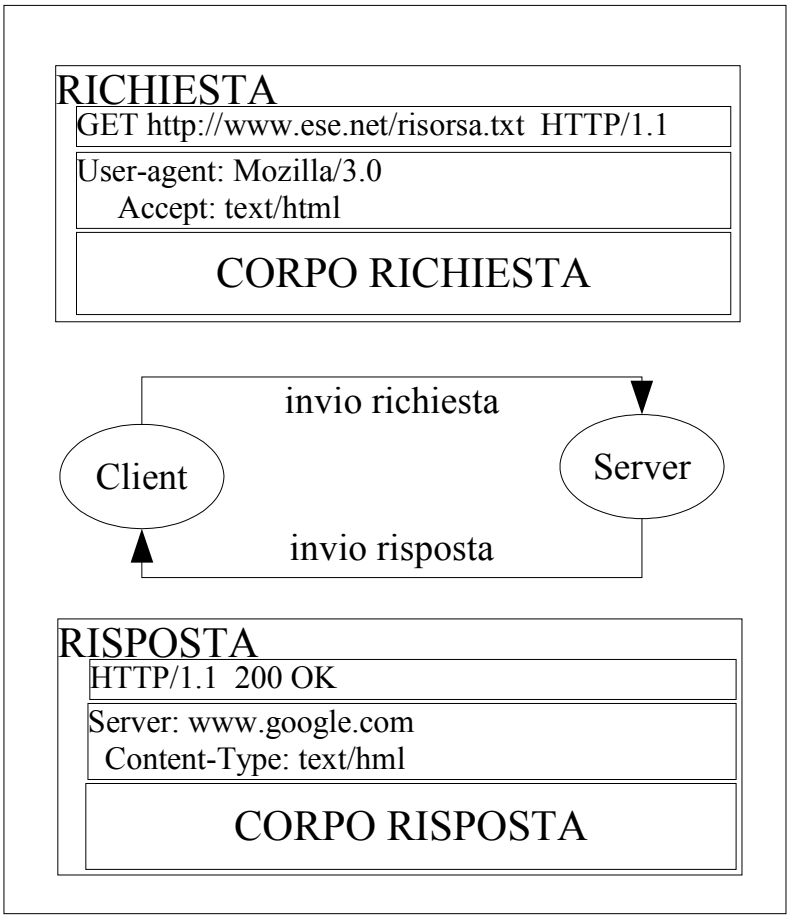

Figura 6.1 Protocollo HTTP

I comandi principalmente usati in HTML sono due:

- comando GET: usato per reperire una pagina HTML ha come argomento l'URL della pagina e corpo vuoto; in questo comando è possibile passare dei parametri nella riga di richiesta inserendo parametri e risorsa come nel seguente esempio

  "http://www.ese.net/risorsa?nome1=valore2&nome2=valore2... "

  in cui i parametri e i valori sono eventualmente codificati. Usare il comando GET per inviare parametri e valori ha lo svantaggio di non potere inviare un numero eccessivo di parametri, in quanto esiste una limitazione alla lunghezza della riga di intestazione.
- comando POST: usato per richiedere una pagina e contestualmente, inviare nel corpo una serie di parametri nella forma "nome1=valore2&nome2=valore2..." eventualmente codificati, come avviene nel caso di GET nell'intestazione, però in questo caso sparisce il limite sul numero di parametri. Questo comando inizialmente era stato pensato per "inviare risposte nei gruppi di discussione", "postare" in gergo; oggi viene usato per dialogare attivamente con i server web.

I comandi in realtà sarebbero molti di più tuttavia questi risultano quelli essenziali per "dialogare" con un ipotetico server web.

6.2 CC/PP e implementazione in JSR188

[JSR188] è una API sviluppata da Sun Microsystems definita per processare i profili dei dispositivi utente, il suo obbiettivo è di facilitare lo sviluppo di applicazioni scritte in Java fornendo una serie di interfacce standard per processare i profili utente scritti in CC/PP.

La versione di base dell'API fornisce solo la struttura per accedere alle informazioni tuttavia è disponibile anche una versione JRS188-RI (Reference Implementation) che realizza al suo interno tutte le funzionalità esposte dall'API.

6.2.1 Vocabolari CC/PP in JSR188

D'ora in poi si farà riferimento spesso al termine *caricare* con tale termine si intende il processo di trasformazione da formato di tipo testo a una rappresentazione interna in termini di classi Java.

Ricordiamo che un vocabolario in CC/PP rappresenta un contratto su quali debbano essere i nomi dei tipi di componenti e i nomi degli attributi nonché la forma di aggregazione e il tipo di base dei valori associati ai componenti. Infine si ricordi altresì che si consiglia la definizione dei vocabolari in file di tipo RDF-Schema, e per i vocabolari di base, vale a dire quelli relativi a RDF e CC/PP, non è necessario che siano caricabili all'interno del sistema ma possono essere realizzati in modo embedded.

In JSR188 i vocabolari di base sono embedded vale a dire che non è necessario caricarli da una definizione esterna, inoltre la definizione dei vocabolari viene fatta in file XML semplificati rispetto a quella che sarebbe la definizione di un vocabolario in RDF.

Diamo uno spezzone di vocabolario definito per JSR188 qui di seguito.

```
<?xml version="1.0"?>
<profile-desc
id="http://www.ese.org/tech/profiles/ccppschema-20030226#">
 <component-desc id="BrowserUA">
  <attribute-desc
     name="BrowserName"
     base-type="literal"
     composition="simple"
     resolution-policy="locked"/>
  <attribute-desc
     name="BrowserVersion"
     base-type="literal"
     composition="simple"
     resolution-policy="locked"/>
.......
```

Come possiamo notare tra le caratteristiche associate agli attributi abbiamo anche *resolution-policy* questo perché in JSR188-RI viene realizzato il meccanismo di *risoluzione* dei profili specificato dalle regole di UAProf.

Ovviamente al file XML di definizione, del vocabolario, è associato uno *schema XML* per la validazione che mostriamo qui di seguito. Ricordiamo che lo *schema XML* definisce come debba essere costruito un file XML per essere valido.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
 <xs:element name="profile-desc">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="extends" minOccurs="0"/>
    <xs:element ref="component-desc" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:string" use="required"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="extends">
  <xs:complexType>
   <xs:attribute name="ref" type="xs:string" use="required"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="component-desc">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="attribute-desc" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:ID" use="required"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="attribute-desc">
  <xs:complexType>
   <xs:attribute name="name" type="xs:string" use="required"/>
   <xs:attribute name="base-type" use="required">
    <xs:simpleType>
     <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="integer"/>
      <xs:enumeration value="literal"/>
      <xs:enumeration value="rational"/>
      <xs:enumeration value="boolean"/>
      <xs:enumeration value="dimension"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="composition" use="required">
    <xs:simpleType>
     <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="simple"/>
      <xs:enumeration value="set"/>
      <xs:enumeration value="sequence"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="resolution-policy" use="required">
    <xs:simpleType>
     <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="locked"/>
      <xs:enumeration value="override"/>
      <xs:enumeration value="append"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

In JSR188 questi file vanno caricati, una volta sola, all'avvio del sistema e poi sono disponibili per tutta la durata d'esecuzione e in tutta l'applicazione, in pratica il vocabolario è sviluppato seguendo il *pattern singleton.* Il meccanismo per il caricamento dello schema è dato da un metodo nella classe *DescriptionManager*, mentre per caricare i vocabolari viene seguito il *pattern factory*, abbiamo un metodo associato alla classe *DescriptionManager* per ottenere una istanza di essa stessa.

Per quel che riguarda la struttura dell' API associata al vocabolario, questa è composta come mostrato dalla figura seguente. La notazione usata è UML di cui parleremo più avanti.

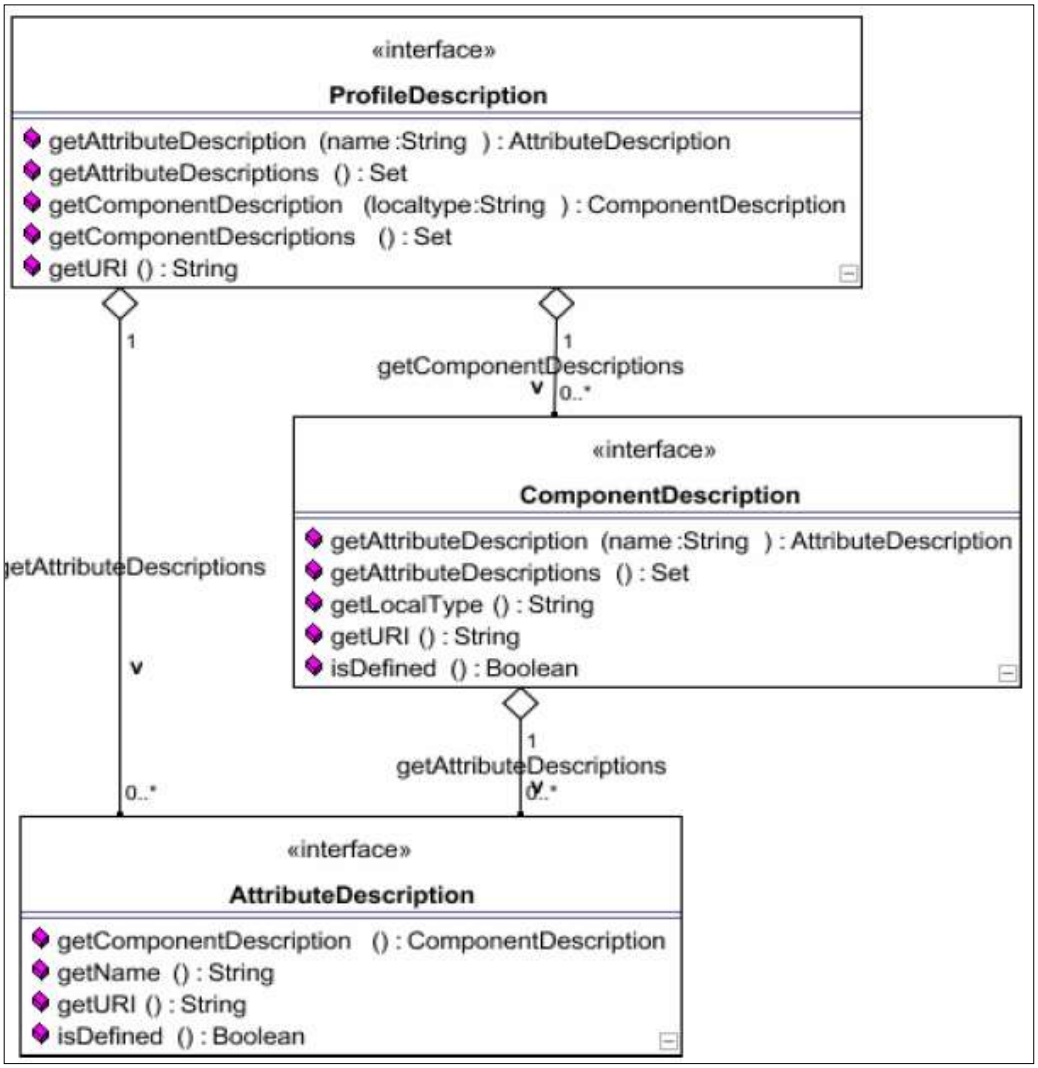

Figura 6.2 API associata ai vocabolari CC/PP

Come si può notare il generico vocabolario è costituito da un oggetto *ProfileDescription* all'interno del quale abbiamo oggetti di tipo *ComponentDescription* che a loro volta contengono *AttributeDescription*; l'accesso alle varie proprietà del vocabolario avviene

tramite nome in formato *String*. Come vedremo più avanti le entità di classe *Description* vengono poi associate a omonime entità che rappresentano l'istanza del profilo.

6.2.2 Profili CC/PP in JSR188

I profili in CC/PP possono essere caricati in diversi modi in quanto l'API consente di processare sia a partire da file che a partire da richieste HTTP, il meccanismo segue sempre il *pattern factory* tuttavia viene svolto su due livelli: un livello di base in cui si ottiene un profilo direttamente, ed un livello innestato per ottenere profili dal *merge* di più oggetti di tipo *ProfileFragment* anch'essi ottenuti tramite il *pattern factory;* la situazione è la seguente:

- caricamento tramite diretto da richiesta HTTP. JSR188 è in grado di processare richieste di tipo W-HTTP, HTTP-ex e WSP (Protocollo WAP).
- caricamento mediante processing di oggetti di classe *ProfileFragment* istanziati a partire da:
  - Stringhe che contengono i profili in CC/PP
  - File il cui contenuto sono profili CC/PP
  - URL che fanno riferimento a pagine contenenti profili CC/PP

L'API consente l'accesso ai profili caricati mediante gli oggetti presentati nella seguente figura (Figura 3.5); da osservare come, per ogni oggetto, abbiamo l'accesso al corrispondente oggetto *Descriptor*.

Fondamentalmente un *Profile* è un insieme di *Component* che a sua volta è un insieme di *Attribute*.

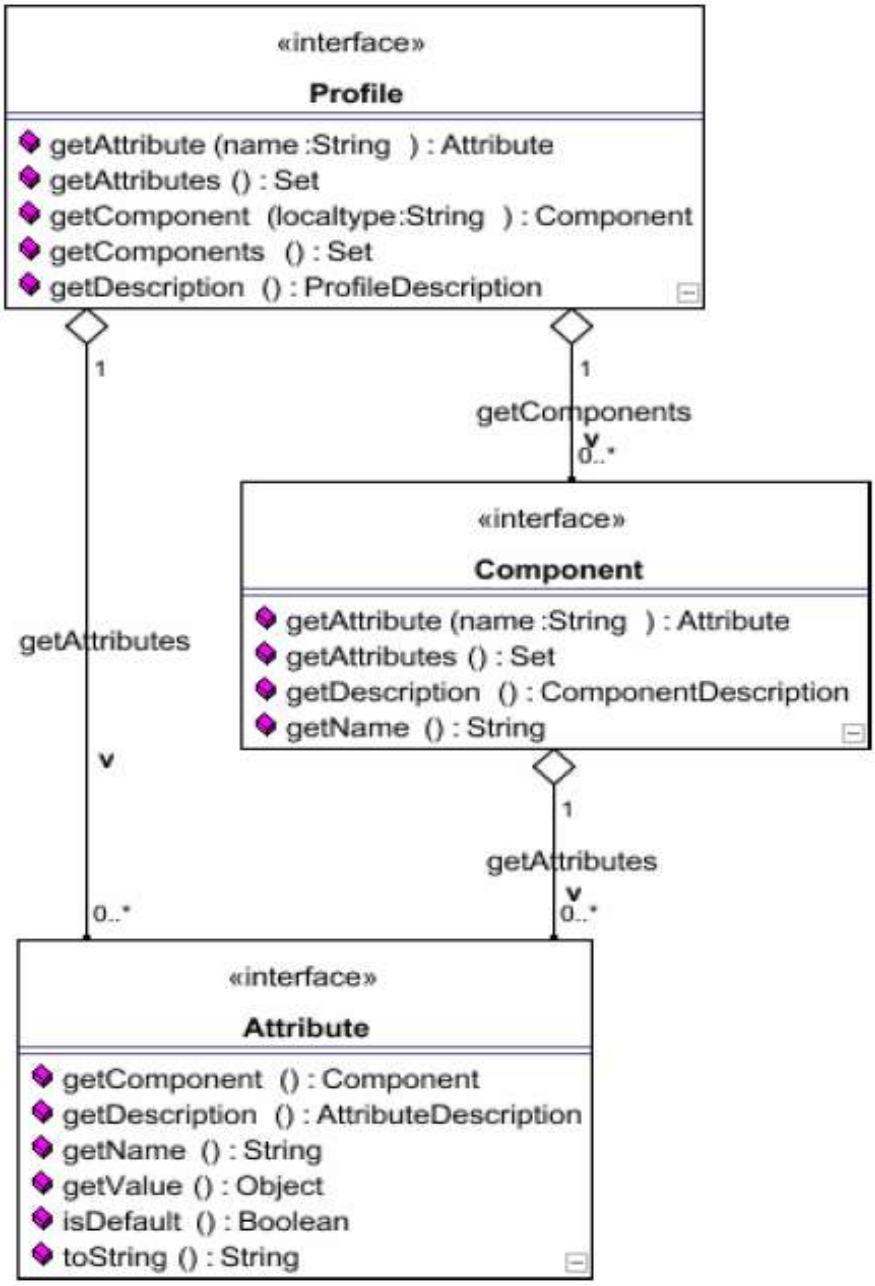

Figura 6.3 API di accesso ai Profili CC/PP

6.2.3 Processing dei profili CC/PP in JSR188

JSR188 è in grado di processare una serie di profili CC/PP e *mixarli* secondo le regole date da UAProf; in particolare questi profili in JSR188 assumono la forma di *ProfileFragment* che poi vengono *mixati* per ottenere un oggetto di tipo *Profile* definitivo.

Durante la spiegazione dell'algoritmo di risoluzione, che verrà illustrato di seguito, si usano i termini *profilo*,*componente* e *attributo* riferendosi agli omonimi elementi introdotti in

CC/PP; viene di seguito riportato l'algoritmo di risoluzione adottato in JSR188:

1. Si risolvono i *default* all'interno dei profili CC/PP ottenendo dei *profili-risolti* se abbiamo attributi in linea con lo stesso nome dei *default* questi li sovrascrivono.
2. Si considerano i *profili-risolti* in ordine, l'ordine è rilevante, e si applica la seguente procedura ricorsiva ai *frammenti* **f1**,**f2**,...**fn** i quali, in sostanza, sono dei profili parziali; la ricorsione consiste nel fatto che la procedura viene applicata ai frammenti **f1** e **f2** e il risultato a **f3**, il risultato a **f4** e così via.

   La procedura è la seguente riferita ai *frammenti* **f1** e **f2**:

- Se un *componente* di un certo tipo presente in **f2** non compare in **f1** viene aggiunto a questo;
- altrimenti si passa a processare gli *attributi* dei *componenti* in **f1** ed **f2** secondo la seguente regola:
  - se un particolare *attributo* di un *componente* di **f2** non è presente nel corrispondente *componente* di **f1** viene copiato in **f1**
  - altrimenti si aggregano gli elementi, gli *attributi*, secondo la seguenti regole:
    - se la regola di aggregazione è *OVERRIDE* il valore dell'*attributo* di **f1** diventa quello di **f2**
    - se la regola di aggregazione è *LOCKED* il valore dell'*attributo* di **f1** rimane e si scarta quello di **f2**
    - se la regola di aggregazione è *APPEND* il valore dell'*attributo* di **f1** ed **f2** devono essere con composizione di tipo insiemistica e nell'*attributo* in **f1** vanno appesi di seguito gli elementi dell'attributo in **f2**

6.2.4 Altre questioni relative a JSR188

In JSR188 vengono inoltre affrontate altre questioni quali:

- la possibilità di avere vocabolari che ne estendono altri
- la validazione dei vocabolari scritti in XML, mediante lo schema presentato in precedenza
- la validazione dei profili in base ai vocabolari assegnati
- come conseguenza del punto precedente, vengono risolte problematiche circa il riconoscimento di componenti sconosciuti e di assegnare un *tipo base* e la *composizione* ai valori degli attributi CC/PP sconosciuti.

6.3 Tomcat un contenitore di servlet

Tomcat è un generico contenitore per *servlet* pagine *JSP*.

Le *servlet* altro non sono che la reinterpretazione, in chiave Java, delle CGI ovvero applicazioni che eseguono servizi su un *server HTTP* e restituiscono una risposta tramite lo stesso protocollo.

Le pagine *JSP* (Java Server Page) rappresentano invece la risposta Java ai linguaggi di *scripting web* lato server; in pratica una pagina *JSP* contiene sia testo HTML sia tratti di codice Java che viene interpretato al momento della richiesta costruendo la risposta da inviare al volo al browser.

Siccome siamo interessati solo alle *servlet* andremo a descrivere, brevemente, l'API messa a disposizione in Tomcat che altro non è che una implementazione di una serie di interfacce definite in Java.

Il comportamento voluto viene ottenuto implementando i giusti metodi nella classe **javax.servlet.http.HttpServlet** che verranno richiamati al momento della richiesta HTTP con i parametri implementazione delle seguenti interfacce: **javax.servlet.http.HttpServletRequest** e **javax.servlet.http.HttpServletResponse**

Il meccanismo è quindi un modello *pull*, o *ad eventi* incui il server richiama la procedura adeguata passando i due parametri necessari per avere accesso alla richiesta (**HttpServletRequest**) e per poter inviare la risposta (**HttpServletResponse**).

Le due interfacce usate per l'interazione vengono implementate in diverse classi in base alla tipologia di richiesta e di risposta, in particolare va osservato come l'oggetto di accesso alla richiesta consenta di accedere sia ai campi delle intestazioni della richiesta HTTP sia ai parametri della richiesta nel caso sia una richiesta di tipo *GET* sia che sia una richiesta di tipo *POST* in modo trasparente, non solo al tipo di richiesta, ma anche alla codifica usata per i parametri.

Oltre a queste caratteristiche è possibile specificare del codice da eseguire al momento dell'avvio del server e della chiusura, e infine va osservato come possono porsi problemi di concorrenza in quanto una servlet viene eseguita da un *thread Java* separato per ogni richiesta HTTP.

6.4 HTTP Client: una API per la comunicazione HTTP

Questa API, sviluppata all'interno del progetto Jakarta Apache, consente di creare e inviare una richiesta HTTP e accedere alla risposta.

In questo caso il meccanismo di creazione della richiesta passa attraverso l'istanziazione di una classe che implementa l'interfaccia **org.apache.commons.httpclient.HttpMethod** che trova implementazione in due classi in particolare **org.apache.commons.httpclient.methods.PostMethod** e **org.apache.commons.httpclient.methods.GetMethod** che consentono la preparazione di oggetti che incapsulano il concetto di richiesta HTTP, incluso l'URL a cui effettuare la richiesta nonché la codifica di eventuali parametri passati contestualmente alla riciesta.

Il passo successivo consiste nell'istanziazione e uso della classe **org.apache.commons.httpclient.HttpClient** che consente di inviare la richiesta, gestire eventuali errori e manipolare la richiesta inserendo in essa la risposta.

Come passo finale è possibile accedere al corpo della risposta mediante lo stesso oggetto usato come richiesta e, un volta terminato, chiudere la connessione.

Il meccanismo è quindi molto semplice e consente una facile manipolazione delle connessioni via HTTP, inoltre rappresenta il lato client rispetto a quello che viene realizzato in Tomcat con le *servlet*.

6.5 Conclusioni

In questo capitolo abbiamo illustrato le tecnologie a cui ci si appoggia nella successiva fase di implementazione.

7 Implementazione e testing

In questo settimo capitolo descriveremo come è stato implementato il sistema nel suo complesso e daremo, per quanto possibile, una motivazione alle scelte implementative svolte. Inoltre mostreremo nel dettaglio alcune delle classi che a livello di progetto erano rimaste descritte in modo superficiale in quanto legate alle tecnologie adottate. Infine presenteremo i risultati di alcuni test svolti sul sistema.

7.1 Scelte implementative

7.1.1 Il linguaggio di programmazione

Come linguaggio per l'implementazione si sceglie di usare Java. Questa scelta è dettata non solo dalla necessità dovuta al fatto che il sistema viene integrato in MUM che è interamente scritto in Java, ma anche per la facilità di uso di questo linguaggio, per la disponibilità di parecchie librerie, e per la portabilità del sistema stesso.

7.1.2 Il protocollo di comunicazione

Come protocollo per la comunicazione tra le entità abbiamo scelto il protocollo HTTP data la diffusione e le crescenti estensioni dello stesso proprio in termini della possibilità di eseguire computazioni remote. La scelta adottata, inoltre ci consente di rendere disponibile in servizio anche per un accesso tramite browser, nonché una comunicazione più libera all'interno della rete internet, con meno vincoli relativi a proxy, firewall, o quanto altro.

7.1.3 La rappresentazione dei dati

Come rappresentazione da usare per i dati relativi ai profili, nello scambio dei messaggi, si è scelto di utilizzare CC/PP in quanto questo risulta assai compatibile, ma meno vincolato, con uno dei più affermati standard usati oggi per la profilazione: UAProf. CC/PP mette insieme le capacità descrittive di UAProf con la possibilità di eseguire una fase di processing molto più libera rispetto quanto imposto in UAProf. Inoltre sono disponibili vocabolari CC/PP che coprono quasi interamente le necessità di questo progetto.

Una alternativa sarebbe stato utilizzare MPEG21 in quanto tale standard risulta molto più completo, tuttavia questo non è ancora largamente affermato, in quanto solo recentemente è si è passati dalla fase di revisione a quella di normativa. Questo lo rende di difficile uso mancando i tool e gli strumenti per il trattamento dello stesso. Anche per i *media-feature* vale un ragionamento analogo in quanto questo standard non si è mai completamente affermato,

inoltre è da osservare che il processo di risoluzione riportato da questo risulta molto pesante a livello computazionale.

7.1.4 Le API adottate

Come librerie abbiamo usato [JSR188] che consente di eseguire il parsing dei profili scritti in CC/PP, tuttavia le entità che vengono generate a seguito della scansione non possono essere usate per creare un oggetto in memoria e quindi la si è usata solo per eseguire le operazioni di parsing per poi ricondurre le entità ad un formato interno. Questo sposa l'interoperabilità di CC/PP con la possibilità di un più facile uso delle entità entro l'ambiente di programmazione. Altro problema di JSR188 è l'impossibilità di rigenerare, dalle entità che questa crea in memoria, il formato XML iniziale, per questo nelle entità usate le classi che implementano i profili prevedono un metodo proprio per tale scopo.

Altra libreria usata è HTTP Client (vedere [HTTPCLI]) la quale consente in modo facile l'accesso alle chiamate HTTP; queste sono necessarie per eseguire le richieste verso il server.

L'ambiente di sviluppo del sistema finale è infine Tomcat (vedere [TOMCAT]) in cui è stato possibile realizzare una servlet che implementa, su HTTP, il semplice protocollo di comunicazione che andremo ad enunciare e che ci consente l'accesso sia tramite applicazione che tramite browser.

7.2 Protocollo di comunicazione in generale

Presentiamo il protocollo con cui implementiamo il meccanismo delle richieste e delle risposte. La scelta per la comunicazione ricade sul modello a scambio di messaggi Client/Server. In particolare andremo ad utilizzare come supporto per la comunicazione la possibilità di effettuare richieste di tipo POST sul protocollo HTTP. Ricordo che questo tipo di richieste consentono di trasferire, oltre ai normali campi della richiesta, nel corpo della stessa delle coppie di parametri e valori associati codificati; questo consente di usare un parametro come selettore per il comando ed altri come parametri per il funzionamento del comando. Oltre a quanto detto il fatto di usare il meccanismo delle richieste POST consente la possibilità di associare alle varie tipologie di comandi diverse pagine HTML statiche che ci consentono di eseguire i medesimi comandi, che svolgiamo tramite l'applicazione con HTTP Client ([HTTPCLI]) anche tramite browser.

Il protocollo studiato prevede, come detto, la presenza di un parametro selettore del comando e i successivi parametri eventualmente necessari. Per quello che riguarda le risposte

che il server ci deve fornire ci si è ispirati allo stesso protocollo HTTP stabilendo che una risposta deve essere composta da una riga di stato terminata con un carattere di *fine-linea* ed un eventuale corpo della risposa. Ovviamente dovremo codificare le tre categorie di entità che vengono qui di seguito evidenziate con oggetti di tipo String fissati. Queste tre categorie sono i messaggi di risposta, i nomi dei parametri passati tramite richiesta POST che identificano oltre al comando gli eventuali parametri, e ovviamente i nomi dei possibili comandi.

Il protocollo così costruito consente l'invio di richieste da parte del client verso il server e le risposte del server al client con associato uno stato che indica il buon fine della richiesta stessa.

7.3 Memorizzazione impostazioni e profili

Per realizzare un sistema di memorizzazione permanente dei profili degli utenti è stato realizzato un sistema che salva i profili su file e li raggruppa in directory in base al dominio di appartenenza; questa scelta serve solo per organizzare logicamente i file; infatti si potrebbe pensare di salvare i profili usando dei nomi che includono il nome del dominio, tuttavia non optiamo per tale scelta perché così l'insieme dei file gestiti risulta più strutturato. Per quello che riguarda il salvataggio dei profili dei dispositivi anche in questo caso si è optato per una directory separata. Da osservare che avremmo potuto tenere tutto insieme ad esempio se consideriamo un path SOMA del tipo "/root/europa/italia" associato a un utente "marco" potremmo pensare di codificare un nome associato al file del tipo: "/profiles/root/europa/italia/marco" così potremmo anche associare dei nomi del tipo "/devices/nomedispositivo" per sfruttare un unico meccanismo per memorizzare tutto nella stessa directory, tuttavia si è optato per un sistema più strutturato in quanto consente di avere, anzitutto il sistema più ordinato e poi rende più facili richieste quali il listato degli utenti di un dominio, o dei dispositivi disponibili.

Andiamo ora a dare il dettaglio di quelle che sono le classi atte alla memorizzazione delle impostazioni:

- classe FileNameListManager: contiene una tabella che mappa un nome in un path nel filesystem e due metodi per la memorizzazione e il ripristino di tale mappa, poi contiene i metodi per l'inserimento, la rimozione, la lettura di tutti gli elementi nella mappa.
- classe DirListManager: deriva dalla precedente e la estende aggiungendo i metodi per la creazione e la rimozione delle directory in concomitanza all'inserimento di elementi nella mappa. In corrispondenza a tali operazioni richiama metodi per il salvataggio

della mappa ereditato.

- classe FileListManager: deriva da FileNameListManager e si comporta come DirListManager solo che aggiunge le funzionalità di creazione e salvataggio di file, anzichè directory con il relativo contenuto.

7.3.1 Formato dei file salvati e impostazione del sistema

I file, come le directory usate per selezionare i domini vengono memorizzate come associazione, in pratica viene generato un nome di file o directory in modo quasi casuale e poi si usa una entità di associazione che associa il nome alla directory o file. Questa associazione viene memorizzata su un file XML nella directory specificata in fase di impostazione dell'oggetto; Questo consente di ripristinare l'oggetto in fase di avvio del sistema. Nonostante il sistema sia forse troppo complesso ci consente di astrarre dal file-system considerato, che nel nostro progetto rappresenta l'astrazione di *repository*. Inoltre è da osservare che questo modo di gestire le cose ha il duplice ruolo di associare il *repository* giusto all'insieme/dominio considerato ma anche di indicarci quali siano ad esempio i domini gestiti dal sistema attuale. Inoltre per efficienza si potrebbe pensare d memorizzare questa associazione anche serializzando la classe che la rappresenta sfruttando la serializzazione presenta nel linguaggio Java.

Per quello che riguarda i file contenenti il profilo questi vengono memorizzati in CC/PP che è anche il formato che ci si aspetta di reperire a fronte di una richiesta di accesso.

7.3.2 Salvataggio e ripristino della configurazione del sistema

Come accennato in precedenza si è scelto di sfruttare il linguaggio XML per salvare le impostazioni sul server, e le informazioni che riguardano i domini gestiti. Questo infatti ci consente, a seguito di ogni operazione di configurazione, di memorizzare il nuovo stato del sistema e in seguito, a un successivo riavvio, di ripristinare il sistema.

Il processo di ripristino per quello che riguarda la gestione dei profili dei dispositivi avviene così:

1. Si verfica se si è il server centrale e se si procedo altrimenti ho terminato
2. Si genera un oggetto FileListManager e lo imposto prelevando le coppie *nomeprofilo-nomefile* da una directory specificata.
3. A questo punto ho terminato, a una successiva richiesta l'oggetto FileListManager espone i metodi di prelievo del contenuto del file *nomefile* senza dovere specificare il nome dello stesso, a partire da *nomeprofilo*.

Processo di ripristino relativo alla gestione dei domini.

1. Si genera un oggetto DirListManager a partire da una directory specificata che consente poi l'accesso alle coppie *nomedominio-nomedir*.
2. A quest punto a una richiesta di profilo relativa a un dominio: si verifica se il dominio è tra quelli gestiti e se lo è si preleva il *nomedir* a partire da *nomeprofilo* da DirListManager.
3. Poi proseguo caricando dalla directory *nomedir* un oggetto FileListManager che contiene l'elenco di associazioni *nomeprofilo-nomefile*.
4. Infine posso accedere al contenuto del file con *nomeprofilo*.

Un unica osservazione: i nomi dei profili considerati in FileListManager per quello che riguarda i profili utente può essere un nome locale che quindi non include il domino nel proprio nome.

7.4 Protocollo di comunicazione dettagli implementativi

Di seguito vengono esposte le possibili richieste a cui il sistema deve rispondere in base alle funzionalità evidenziate cercando di mettere in evidenza, per ogni comando, i parametri necessari e le possibili risposte. Ovviamente questa rappresentazione riprende quella fatta nel capitolo di progetto e dettagliano meglio il sistema aggiungendo l'informazione circa l'esito della richiesta.

7.4.1 Servizio di gestione del server

Questo svolge le funzioni per l'impostazione come gestore dei profili dei dispositivi, oppure per impostare l'indirizzo verso cui girare le richieste. La seguente tabella riassuntiva mostra i parametri necessari per l'esecuzione dei comandi associati e i possibili esiti.

comando	parametri	risposta	stato
isRoot		si/no	ok
setRoot			ok
setRootURL	URL remoto		ok/errore se url non valido
getRootURL		URL se presente o nulla	ok/errore se è root

7.4.2 Servizio di gestione dei domini

Consente di gestire l'associazione del sistema ai vari domini.

comando	parametri	risposta	stato
listDomain		lista o nulla	ok /errore se manca domini
existDomain	dominio	si/no	ok
addDomain	dominio		ok/errore nome non valido
delDomain	dominio		ok/errore nome non valido

7.4.3 Servizio di gestione dei profili utente

Consente di memorizzare o reperire i profili utente, ricordo che è necessario passare sempre i parametri circa il dominio e l'URL associato al server competente nel caso di richiesta

comando	parametri	risposta	stato
getProfile	dominio url destinazione utente	profilo o nulla	ok /errore
addProfile	dominio utente profilo		ok /errore
listProfile	dominio	elenco profili o nulla	ok /errore
delProfile	dominio utente		ok/errore

7.4.4 Servizio di gestione dei profili dei dispositivi

Consente accesso al sistema di profili dei dispositivi, va ricordato che per questi non serve conoscere né il dominio né l' URL del sito competente

comando	parametri	risposta	stato
getProfile	dispositivo	profilo o nulla	ok /errore
addProfile	dispositivo profilo		ok /errore
listProfile		elenco profili	ok /errore
delProfile	dispositivo		ok/errore

7.4.5 Servizio di risoluzione dei profili

Si occupa, dato un profilo utente ed uno di un dispositivo di personalizzare, secondo certe politiche, il profilo del dispositivo con le preferenze dell'utente.

comando	parametri	risposta	stato
resolveprofile	dispositivo dominio url destinazione utente	profilo o nulla	ok /errore

7.4.6 Sistema di gestione degli errori

La gestione degli errori è un problema che può essere rilavante in un contesto Client/Server nel senso che è necessario decidere dove gestire gli errori. La scelta è stata di gestirli dal punto di vista del server, come peraltro evidenziato nella presentazione delle funzionalità offerte. Questo consente di sgravare il client dal controllo dei parametri usati nelle richieste. Oltre a ciò e da evidenziare che comunque il server deve controllare i parametri e quindi sarebbe inutile sottoporli a controllo da parte del client, tuttavia potrebbe essere una pratica utile controllare, lato client, i parametri per evitare al server un sovraccarico dovuto a richieste errate, il controllo lato client risulta quindi una sorta di ottimizzazione.

7.5 Classi nel sistema

Le classi presenti nel sistema ricalcano esattamente le entità presentate nella fase di progetto. In questo contesto andiamo quindi a presentare dettagliatamente gli oggetti che in fase di progetto non avevamo specificato perché troppo legate al contesto applicativo, poi passeremo brevemente in rassegna le classi implementazione delle entità precedentemente descritte.

- classe ClientSystem: questa classe viene usata come classe base per tutte le entità client ed incorpora la logica dei comandi da inviare verso il server remoto. La sua logica comprende un sistema di preparazione della richiesta ed uno per accedere alla risposta. La logica prevede tre fasi per eseguire una richiesta che sono: impostazione dell'indirizzo di destinazione, il server verso cui invieremo la richiesta, preparazione della richiesta mediante un metodo che prepara i parametri della richiesta in termini di nome e valore; la terza fase consiste nell'esecuzione della richiesta; al termine di questa fase viene terminata la connessione col server e sono accessibili i parametri della risposta che sono suddivisi in intestazione, che contiene una linea di testo che indica lo stato del server, sempre presente, ed il corpo della risposta se atteso. E' importante osservare, in questa sede, come sia necessaria l'impostazione di un indirizzo di

destinazione che nell'integrazione in MUM viene rimossa a causa dell'assunzione che ad ogni server sia associato un dominio o che comunque sia possibile impostare un indirizzo o URL a cui faranno riferimento tutti i nodi nel dominio, come impostazione in SOMA. Il protocollo scelto inoltre consente l'interfacciamento tramite browser in cui compare la risposta per intero, a tale scopo sono state sviluppate alcune pagine statiche per testare le funzionalità offerte.

implementazione e uso

Questa classe è dotata dei seguenti metodi che vanno usati uno di seguito all'altro per preparare la richiesta:

- setRemoteUrl: imposta l'URL verso cui fare la richiesta, va impostato una sola volta (politca *at least one*).
- setCommand: imposta il parametro per il comando POST da inviare
- addParameter: aggiunge eventuali altri parametri nella richiesta, ricordiamo che i parametri devono essere specificati con nome e valore; setCommand esegue semplicemente la stessa operazione con il nome di parametro fissato.
- exeCommand: esegue il comando preparato con i metodi precedentemente enunciati, in pratica viene preparata una richiesta di tipo POST mediante l'API *HTTP Client* e sempre con questa viene inviato il comando. L'API menzionata ripete le stesse operazioni da noi enunciate, vale a dire: prepara la richiesta e la invia, e infine riceve la risposta e la rende disponibile come *array di byte*. Tutto questo nasconde l'apertura di una socket verso il server a cui vengono inviati i dati qui preparati e poi la lettura dalla stessa della risposta.

  exeCommand termina rilasciando la connessione col server e formattando in due campi al suo interno le parti attese della risposta, vale a dire l'intestazione e la risposta vera e propria.
- getResponseH: consente l'accesso all'intestazione della risposta che contiene lo stato di esecuzione del comando
- getResponseB: contiene l'eventuale corpo della risposta.

Ricordo che le intestazioni e corpo di cui qui si parla non sono quelli relativi al protocollo HTTP, ma sono quelli relativi al nostro protocollo e sono quindi contenuti tutti e due nel corpo della risposta HTTP.

- classe MainServletRequest: è l'estensione di una classe *HttpServlet* definita in Tomcat, la logica è la seguente: si deriva da questa classe e se ne implementano i metodi *doGet*() e/o *doPost*() che ricevono come parametri, all'arrivo di una richiesta HTML di tipo GET o POST , due oggetti di classe: *HttpServletRequest* e *HttpServletResponse* di cui il primo rappresenta la richiesta e il secondo l'oggetto che incapsula la risposta usata per rispondere al client. All'interno di questo metodo eseguiamo un ciclo sui vari metodi che riconoscono le varie categorie di richieste, eseguono i comandi associati con i metodi definiti nelle opportune classi, ed inviano la risposta. Un'ultima cosa da osservare è l'uso del metodo *init()* definito nella classe *HttpServlet* e che viene richiamato all'avvio del sistema, questo è il modo in cui impostiamo la configurazione iniziale del server, istanziando laddove serve le varie classi e/o caricandole dal filesystem.

  implementazione e uso

  Questa classe contiene quindi i seguenti metodi che vengono automaticamente invocati dall'architettura introdotta da Tomcat all'avvio di una connessione verso il server nascondendo quindi il processo di preparazione di una socket di ascolto e il relativo processo di generazione di un *Thread Java* per gestire il processo di risposta. Resta a carico dell'utente la preparazione e l'invio della risposta e la chiusura della connessione, anche se questa viene automaticamente chiusa al termine del *Thread*.

  - metodo doGet(): richiama il metodo executeCommand() passandogli i parametri ricevuti.
  - metodo doPost():richiama il metodo executeCommand() passandogli i parametri ricevuti.
  - metodo executeCommand(): riceve gli oggetti *HttpServletRequest* e *HttpServletResponse* poi richiama un elenco di metodi implementati nella stessa classe che restituiscono un valore *di successo* se riconoscono il comando come quello da loro gestito. Se nessuno ha successo risponde che non gestisce il comando.
  - metodo serverManagerCommand(): realizza i comandi relativi alla gestione del server utilizzando i metodi offerti dalla classe ServerManager di cui una istanza è un attributo della classe.
  - metodo domainManagerCommand(): realizza i comandi relativi alla gestione del server utilizzando i metodi offerti dalla classe DomainManager di cui una istanza è

un attributo della classe.

- metodo profileUserManagerCommand():realizza i comandi relativi alla gestione del server utilizzando i metodi offerti dalla classe ProfileUserManager di cui una istanza è un attributo della classe.
- metodo profileDeviceManagerCommand():realizza i comandi relativi alla gestione del server utilizzando i metodi offerti dalla classe ProfileDomainManager di cui una istanza è un attributo della classe.
- metodo resolverManagerCommand():realizza i comandi relativi alla gestione del server utilizzando i metodi offerti dalla classe ResolverManager di cui una istanza è un attributo della classe.

Le altre classi a livello server sono le seguenti:

- ServerManager: espone i metodi per verificare se il server corrente è o meno in server centrale e per l'impostazione dello stesso per tale comportamento nonché quelli per l'impostazione dell'URL del server centrale. Queste impostazioni saranno salvate con l'ausilio di un'altra classe in formato XML in un file specificato.
- DomainManager: espone i metodi per la gestione dei domini associati al server; usa un oggetto di classe DirListManager precedentemente descritta, per la gestione dei domini. Espone i metodi per l'aggiunta, la rimozione l'accesso all'elenco degli stessi.
- ProfileUserManager: espone i metodi per la gestione dei profili utente quali inserimento, rimozione e accesso; nel caso di accesso usa una oggetto di classe DirListManager per decidere se gestire direttamente il domini o se utilizzare un oggetto di classe ClientProfileUserManager per fare da proxy. Se non deve fare da proxy con l'oggetto DirListManager sopra menzionato reperisce un oggetto FileListManager con cui gestisce il profilo.
- ProfileDeviceManager: espone i metodi per la gestione dei profili dei dispositivi quali accesso, rimozione, inserimento. Nel caso di accesso utilizza un l'oggetto ServerManager per decidere se fare da proxy passare la richiesta al server centrale tramite un oggetto di classe ClientProfileDeviceManager, oppure se gestirlo direttamente usando un appropriato oggetto di classe FileListaManager.
- ResolverManager: utilizza oggetti di classe ProfileDeviceManager e ProfileUserManager per reperire i profili e poi li combina usando la classe

ResolverPolicy che espone un unico metodo che accetta un profilo utente, uno di dispositivo e restituisce un profilo di dispositivo personalizzato secondo certe regole.

Le altre classi a livello client sono le seguenti e tutte derivano da ClientSystem:

- ClientServerManager: gestisce le richieste di impostazione del server come server centrale o come delegato, usa i metodi di ClientSystem per preparare la richiesta e estrarre la risposta.
- ClientDomainManager: gestisce le richieste per impostare i domini a cui è associato il server remoto e usa i metodi di ClientSystem per preparare la richiesta e estrarre la risposta.
- ClientProfileUserManager: gestisce le richieste la manipolazione dei profili utente e usa i metodi di ClientSystem per preparare la richiesta e estrarre la risposta.
- ClientProfileDeviceManager: gestisce le richieste la manipolazione dei profili di dispositivo e usa i metodi di ClientSystem per preparare la richiesta e estrarre la risposta.
- ClientServerManager: gestisce le richieste la risoluzione dei profili di dispositivo con profili utente e usa i metodi di ClientSystem per preparare la richiesta e estrarre la risposta.

7.6 Rappresentazione dei dati interni: i profili

La rappresentazione generale che viene data dei profili, internamente al sistema, è associata a delle classi che espongono i metodi per l'accesso agli stessi e dispongono di due metodi, uno per il recupero del profilo dalla forma generata da JSR188 e l'altro per la serializzazione in CC/PP. Per quello che riguarda i parametri ci siamo in parte adeguati agli standard ed in parte abbiamo esteso i vocabolari standard utilizzati per aggiungere alcuni parametri da trattare contestualmente al sistema. Di seguito presentiamo un elenco dei vocabolari da cui abbiamo prelevato i vari attributi e componenti, infatti una caratteristica saliente di CC/PP è la possibilità di usare e miscelare più vocabolari mediante l'uso dei namespace che li identificano.

Qui di seguito i vocabolari usati:

prefisso usato	URI associato
prf	http://www.openmobilealliance.org/tech/profiles/UAPROF/ccppschema-20030226#
pca	http://developer.intel.com/pca/developernetwork/devsupport/intel-pca-schema-200201#
tay	http://www.inrialpes.fr/opera/people/Tayeb.Lemlouma/NegotiationSchema/ClientProfileSchema-03012002#
gen	http://www.lia.deis.unibo.it/ccpp/generic#

L'ultimo è un vocabolario definito in sede di progetto per l'introduzione di alcuni parametri utili al sistema che presento nella sua forma compatibile con JSR188, qui di seguito.

```
<?xml version="1.0"?>
<profile-desc id="http://www.lia.deis.unibo.it/ccpp/generic#" >

  <component-desc id="MetaInfo">
    <attribute-desc name="ProfileID"
       base-type="literal"
       composition="simple"
       resolution-policy="locked"/>
    <attribute-desc name="ProfileLastUpdate"
       base-type="literal"
       composition="simple"
       resolution-policy="locked"/>
  </component-desc>

  <component-desc id="UserInfo">
    <attribute-desc name="UserName"
       base-type="literal"
       composition="simple"
       resolution-policy="locked"/>
    <attribute-desc
       name="UserDevices"
       base-type="literal"
       composition="set"
       resolution-policy="locked"/>
    <attribute-desc name="PreferredBearerBitRateOccupation"
       base-type="integer"
       composition="simple"
       resolution-policy="locked"/>
    <attribute-desc name="PreferredVideoInputEncoder"
       base-type="literal"
       composition="sequence"
       resolution-policy="locked"/>
    <attribute-desc name="PreferredLanguage"
       base-type="literal"
       composition="sequence"
       resolution-policy="locked"/>
  </component-desc>

</profile-desc>
```

Gli elementi definiti sono quattro e sono: ProfileID e ProfileLastUpdate nel componente CC/PP MetaInfo e UserName, UserDevices, PreferredBitRateOccupation, PreferredVideoInputEncoder, PreferredLanguage nel componente UserInfo che ovviamente rappresenta il profilo dell'utente. Per il profilo dei dispositivi abbiamo introdotto altri parametri presi dagli altri vocabolari.

Infine la composizione degli elementi si concretizza nelle due classi UserProfile e DeviceProfile di cui qui di seguito elenchiamo gli attributi a cui avremo accesso. Si è preferito dare una rappresentazione piatta dei profili, senza componenti, in quanto risulta di assai più semplice utilizzo all'interno del sistema piuttosto che dovere reperire gli attributi dai vari componenti.

Classe GenericProfile da cui derivano le altre; questa contiene i seguenti attributi:

- ProfileID usato come identificatore per il profilo
- ProfileLastUpdate usato come data di riferimento.

Questi attributi possono essere usati in una successiva evoluzione a scopo di caching.

Classe UserProfile: da accesso alle seguenti caratteristiche aggiuntive:

- UserName: nome dell'utente usato per identificare il profilo.
- UserDevices: elenco dei dispositivi dell'utente.
- PreferredBearerBandOccupation: preferenza sulla percentuale di banda occupata.
- PreferredLanguage: lingue preferite, in ordine.
- PreferredVideoInputEncoder: video-encoder preferiti, in ordine.

Classe DeviceProfile:da accesso alle seguenti caratteristiche:

- DeviceName: usato come identificatore
- DeviceType: indica il tipo di dispositivo ad esempio PDA o LAPTOP
- ColorCapable: indica se ha il display a colori o in bianco e nero
- BitsPerPixel: profondità di colore o di grigi come numero di bit per pixel
- CPU: classe del processore
- PixelAspectRatio: rapporto larghezza su altezza dei pixel
- ScreenSize: dimensioni dello schermo in pixel
- screen: dimensioni dello schermo in millimetri
- CurrentBearerService: protocollo attualmente usato per la trasmissione
- SupportedBearers: protocolli di comunicazione supportati
- CurrentBearerMaximumBitRate: velocità massima sulla connessione attuale in b/s
- SupportedBearersMaximumBitRates: elenco delle velocità delle connessioni supportate
- VideoInputEncoder: elenco dei video-encoder supportati
- CcppAccept-Language: elenco in ordine di linguaggi supportati, in ordine.

7.7 Sistema di risoluzione dei profili

Il sistema di gestione per la profilazione, come già detto, è un servizio che prevede un comando che sia in grado di indicare un profilo utente, un profilo di dispositivo e che restituisce un profilo di dispositivo personalizzato col le opzioni date dall'utente. Per fare questo il corpo principale preleva i profili necessari, e poi passa il tutto a una classe che implementa la logica di risoluzione. Nel nostro caso abbiamo preso in considerazione tre parametri di personalizzazione: l'occupazione di banda, che esprime la percentuale di banda da occupare e che modifica la banda massima attuale di conseguenza, poi abbiamo considerato i linguaggi e i video-encoder preferiti, per questi che sono elenchi di valori la scelta è di riordinarli in base alle preferenze espresse nel profilo dell'utente.

7.8 Testing del sistema

Come test del sistema si è verificato che la politica di risoluzione introdotta si comportasse come desiderato di seguito i risultati raccolti relativamente alle caratteristiche a cui si è interessati.

<table>
<tr><th>Profili usati</th><th>Profilo di dispositivo generato</th></tr>
<tr><td>utente:
• PreferredBearerBandOccupation: 50
• PreferredLanguage:[it, en, fr]
• PreferredVideoInputEncoder:
[MPEG-1, MPEG-2, H.261]</td><td rowspan="2">dispositivo:
• CurrentBearerMaximumBitRate:
100
• CcppAccept-Language:
[en, ja]
• VideoInputEncoder:
[MPEG-1, H.261]</td></tr>
<tr><td>dispositivo:
• CurrentBearerMaximumBitRate: 200
• CcppAccept-Language:[ja,en]
• VideoInputEncoder:
[H.261, MPEG-1]</td></tr>
</table>

Per quello che riguarda la performance abbiamo preparato un topologia con tre nodi:

1. macchina Pentium 3 733MHz, 512MB di ram SDRAM. impostata con dominio "/root" e radice della gerarchia, su questa abbiamo inserito il profilo di un utente "luca", e un dispositivo con profilo "nokia3310".
2. macchina Centrino1.5 GHz, 512MB di ram DDR. Impostata con dominio "/root/italia", su questa abbiamo inserito un utente "marco"
3. macchina Pentium 4 2.6GHz, 512MB di ram DDR. impostata con dominio "/root/america" su questa abbiamo inserito un utente "manu"

Tutte le macchine sono state dotate di schede a 100Mbit collegate tramite switch, e tutte montano WindowsXP come sistema operativo.

La seguente figura mostra la topologia su cui è stato effettuato il test.

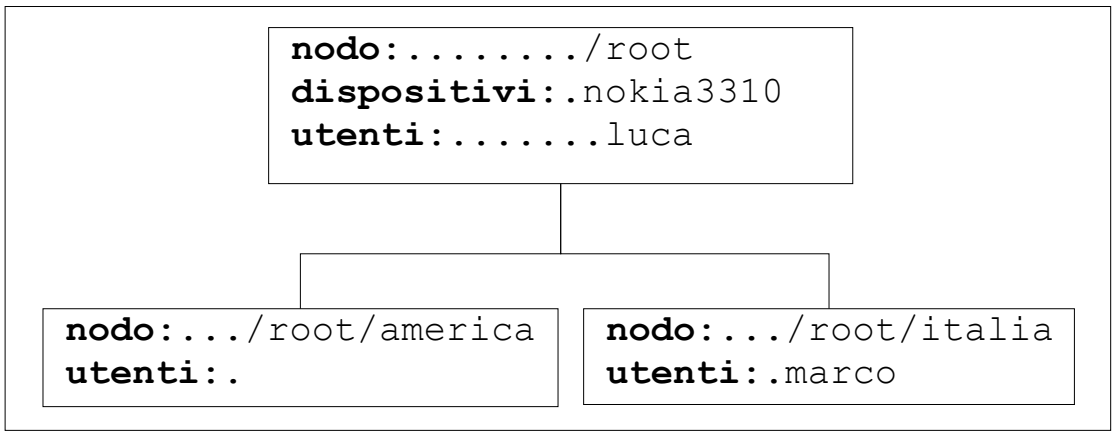

Figura 7.1 Topologia di test

I test seguenti sono stati svolti su tutte e tre le macchine. Come test abbiamo misurato il tempo medio necessario per ottenere un profilo di dispositivo personalizzato a partire da quello utente e quello del dispositivo iniziale. I tre profili utente sono identici, quello che interessa è il tempo di reperimento delle parti dalla rete.

I test vengono svolti facendo eseguire l'applicazione di testing direttamente sulla macchina a cui si collega la stessa per richiedere i risultati; in questo modo è il nodo del sistema che deve occuparsi di reperire se necessario dagli altri nodi i profili utili a effettuare la risoluzione. Questo significa che in alcuni casi i profili da reperire si trovano già sul nodo contattato mentre in altri casi il profilo del dispositivo e/o quello dell'utente si trova in un nodo esterno. Questo noi intendiamo nel seguito per profili esterni al nodo.

Richieste effettuate nei test:

1. combinazione di utente "luca" con dispositivo "nokia3310"; questa richiesta quando

eseguita sul nodo "/root" non richiede alcun profilo esterno, sui nodi "/root/italia" e "/root/america" richiede due profili esterni.

2. combinazione di utente "marco" con dispositivo "nokia3310"; questo test quando eseguito sul nodo "/root" e sul nodo "/root/italia" richiede una profilo esterno, se eseguito su "/root/america" richiede due profili esterni
3. combinazione di utente "manu" con dispositivo "nokia3310"; in questo caso se eseguito sul nodo "/root" o sul nodo "/root/america" richiede un profilo esterno, mentre nel caso di "/root/italia" i profili esterni sono due.

Qui di seguito vengono presentati i risultati medi raccolti in millisecondi, in base al numero di profili esterni richiesti dal test, per le varie macchine considerate.

n° profili esterni	**macchina /root**	**macchina /root/italia**	**macchina:/root/america**
0	44		
1	50	30	48
2		42	57

Ovviamente nel caso del server centrale vi sarà al massimo solo un profilo esterno da reperire al massimo essendo questo a detenere i profili dei dispositivi; il caso duale è rappresentato dalle macchine che fanno riferimento al server centrale per le quali ci sarà sempre un profilo esterno, quello dei dispositivi. E' interessante notare come in tutte e tre le macchine considerate il tempo necessario aumenta con il numero di profili esterni da reperire; tuttavia i tempi non sono uguali a causa della diversità, in termini di prestazioni, delle architetture considerate. Questo mette chiaramente in evidenza la necessità di fare caching dei profili. Infatti se i profili sono già presenti sul nodo a cui perviene la richiesta di risoluzione è chiaro che i tempi diventerebbero paragonabili a quelli in cui non abbiamo alcun profilo esterno e quindi minimi. Oltre a questo la possibilità di fare caching ridurrebbe sicuramente il traffico sulla rete, e il carico computazionale a cui è sottoposto soprattutto il server centrale a causa delle continue richieste, per reperire sempre i medesimi profili che, nel caso dei dispositivi, vanno considerati pressoché immutabili. Una interessante proposta di sperimentazione potrebbe essere quella di eseguire il test su macchine uguali collegate in modo uniforme, oppure in un contesto wireless. Nonostante non sia stato possibile svolgere tali prove i risultati raccolti indicano comunque che l'introduzione del caching gioverebbe sicuramente al sistema considerato.

7.9 Conclusioni

Questo settimo capitolo rappresenta il frutto concreto degli studi svolti inerentemente al progetto di tesi ed ha mostrato come sia possibile introdurre un semplice sistema di gestione accessibile anche tramite web, per la gestione dei profili utente. Infine la fase di testing ha mostrato come il sistema combini i profili mediante una semplice politica che in questo caso è stata cablata nel codice ma il cui supporto può facilmente essere esteso per gestire qualunque forma di script separato, per la definizione delle politiche. Infine i test svolti sulle performance hanno evidenziato una leggera carenza nel sistema dovuta all'assenza di qualunque forma di caching.

Conclusioni

Questo lavoro di tesi si è rivelato un interessante mezzo per approfondire gli studi sui sistemi di profilazione attualmente disponibili, utilizzabili in ambienti multimediale distribuiti. In particolare abbiamo inizialmente posto l'attenzione sui metodi di presentazione dei sistemi di profilazione in ambito multimediale per poi passare alla definizione degli standard oggi presenti per tali finalità.

Dopo questo abbiamo preso in considerazione tre standard per la rappresentazione delle caratteristiche utente, e il relativo processo di profilazione: *media-feature*, MPEG21 e CC/PP. Il sistema preso in considerazione, in questo lavoro di tesi, è stato lo standard CC/PP il quale si è consolidato da poco e che tuttavia risulta essere molto affermato perché del tutto compatibile con UAProf , lo standard *de facto* per la profilazione dei cellulari WAP. La diffusione e l'affermazione di CC/PP è dimostrata anche dalla presenza di parecchi strumenti per il trattamento dello stesso oggi disponibili gratuitamente. Lo standard utilizzato tuttavia non è esente da difetti il che dimostra che ancora molto lavoro deve essere fatto per ottenere sistemi universalmente adattabili.

Per quanto rilevato quindi possiamo dire che oggi siamo solo all'inizio per quello che riguarda l'affermazione di un sistema veramente usabile per il trattamento della profilazione. Oltre a ciò si rilevano anche carenze a livello di protocolli e sistemi operativi. Tuttavia nel contesto generale il fatto stesso che ci si è posti il problema indica che si è sulla giusta via per la risoluzione di tali problematiche.

L'ambiente SOMA e l'architettura MUM in cui ci si è trovati ad operare rappresentano una valida alternativa ai classici modelli di interazione e mostra la sua forza nell'uso congiunto di tecniche di caching, agenti mobili e architetture Client/Server, per quello che riguarda le problematiche relative alla distribuzione di materiale multimediale.

Nello sviluppo del progetto si è avuto modo di sperimentare in prima persona quelle che sono le tecnologie attualmente disponibili in ambiente web per la gestione di servizi. In particolare l'uso di tecnologie consolidate e aperte ha consentito lo sviluppo di una architettura effettivamente aperta e integrabile con gli attuali sistemi. Oltre a questo è da osservare come il linguaggio Java faccia da *collante* nei moderni sistemi di sviluppo; non a caso per Java si trovano moltissime applicazioni e librerie utilizzabili gratuitamente cosa che consente

l'alimentarsi di quel circolo virtuoso che aumenta la diffusione di questo straordinario linguaggio.

Infine l'utilizzo del sistema ha dimostrato come un paradigma di tipo *Request-Response* possa essere utilizzato per costruire una infrastruttura per la fruizione, da remoto, di profili utente facilmente integrabile con i moderni ambienti di rete. Questo modello computazionale d'altra parte è particolarmente adatto allo sviluppo di oggetti quali i *proxy* che fanno da intermediario tra le diverse entità in comunicazione. L'introduzione dei suddetti *proxy* dà inoltre la possibilità di compiere operazioni quali l'adattamento dei contenuti e il monitoraggio dell'ambiente in cui questi sono posti; anche in questo frangente è necessario un sistema di descrizione che dia i vari profili delle entità in gioco in termini di capacità e disponibilità, che devono essere poi sfruttati secondo le preferenze date dall'utente in modo da consentirgli quella che egli ritiene la migliore esperienza possibile nella fruizione del servizio.

Il meccanismo proposto nel presente lavoro tesi consente all'utente di dichiarare quelle che sono le sue preferenze in termini qualitativi, è poi il sistema ad adattare queste sue scelte dinamicamente ed automaticamente ai dispositivi da lui adottati.

Come sviluppi futuri al progetto di tesi qui presentato una proposta potrebbe essere l'introduzione di una forma di cache, di cui abbiamo tracciato le linee guida; questo potrebbe migliorare sensibilmente l'efficenza del sistema, come dimostrato dai primi test svolti. Un'altra proposta potrebbe essere l'estensione del sistema in modo che possa trattare oltre alle informazioni dell'utente anche quelle relative all'ambiente operativo in termini di capacità dei nodi coinvolti nell'erogazione del servizio.

Sigle Abbreviazioni e Acronimi

- MUM: Mobile agent based Ubiquitous multimedia Middleware; progetto sviluppato all'interno del DEIS (Dipartimento di Elettronica, Informatica e Sistemistica) dell'Università di Bologna per la distribuzione di materiale multimediale all'interno della nuova generazione di protocolli di rete wireless, realizzato seguendo il paradigma degli Agenti Mobili appoggiandosi a SOMA. http://lia.deis.unibo.it/Research/MUM/
- SOMA: Secure and Open Mobile Agent; progetto sviluppato all'interno del DEIS (Dipartimento di Elettronica, Informatica e Sistemistica) dell'Università di Bologna che realizza una infrastruttura per la gestione di un sistema ad agenti mobili in rete. http://www.lia.deis.unibo.it/research/SOMA/
- CC/PP: Composite Capabilites/Preferences Profile.
- FIFO: First In First Out; tecnica di gestione delle strutture dati di tipo aggreagato, con ordinamento e con numero di elementi limitato, in cui il primo elemento che entra nell'insieme è anche il primo ad uscirne. Gestione usata, tipicamente nelle strutture dati dette, in informatica, *code*. Si contrappone a LIFO.
- LIFO: Last In First Out; tecnica di gestione delle strutture dati di tipo aggreagato, con ordinamento e con numero di elementi limitato, in cui il l'ultimo elemento che entra nell'insieme è anche il primo ad uscirne. Gestione usata, tipicamente nelle strutture dati dette, in informatica, *stack*. Si contrappone a FIFO.
- Internet: per internet si intende la rete delle reti ovvero l'insieme delle reti interconnesse nel mondo.
- WEB: per Web si intende spesso il mondo di internet e delle reti interconnesse.
- WWW: World Wide Web si intende l'insieme delle reti con particolare accento sull'insieme delle pagine HTML messe a disposizione e tra loro interconnesse.
- HTML: HyperText Markup Language, linguaggio simbolico di marcatura usato per scrivere pagine contenenti ipertesti, ovvero testi con riferimenti ad altri testi.Utilizzato nel WEB per pubblicare documenti il cui contenuto può contenere riferimenti ad altri documenti.
- pagine HTML: pagine scritte in HTML.
- HTTP: HyperText Transfer Protocol; protocollo di trasferimento utilizzato per trasferire le pagine HTML.

- Browser: indica l'applicazione client usata per accedere a internet.
- MIME: Multi purpose Internet Mail Extension; standard utilizzato in internet per definire il "tipo" associato a un documento scambiato.
- XML : eXtensible Markup Language; linguaggio simbolico di marcatura generico, simile all'HTML ma con regole di scrittura più rigide.
- XHTML:eXtensible HyperText Markup Language; versione XML di HTML usato per definire pagine HTML che rispettano la più rigida sintassi XML.
- XSL: eXtensible Stylesheet Language; linguaggio che descrive lo stile da applicare a un documento XML.
- XSLT: eXtensible Stylesheet Language Transormation; specifica di XSL che indica gli stili in termini di trasformazioni (sostituzione di TAG con altro) da applicare a un documento XML per ottenere la versione trasformata.Un'altro standard all'interno di XSL è Formatting Object che è un linguaggio usato per descrivere gli stili (visivi) da applicare ai vari TAG del documento XML simile a CSS.
- CSS: Cascading Style Sheets standard inrodotto in HTML per specificare le caratteristiche visive da applicare ai TAG HTML.
- QoS: Quality of Service. Indica la possibilità di stabilire le caratteristiche che devono essere rispettate durante l'erogazione di un servizio.
- ISO: International Organization for Standardization; ente internazionale per la standardizzazione. http://www.iso.org/
- OSI: Open System Interconnection; modello per il sistema di comunicazione tra sistemi aperti sviluppato da ISO.
- MPEG : Moving Pictures Experts Group. Gruppo costituito in OSI per la standardizzazione delle codifiche per applicazioni multimediali.
- API: Application Program Interface. Interfaccie di accesso alle librerie in termini di funzioni o metodi ed oggetti.
- CORBA: Common Object Rquest Broker; è uno standard definito per garantire interoperabilità e portabilità di sistemi object-oriented nel distribuito. Definisce una architettura object-oriented nel distribuito senza specificare linguaggio e piattaforma di appoggio.
- IDL: Interface Data Language; indica un generico metalinguaggio usato in genere per descrivere l'interfaccia co cui si presenta un oggetto software che viene poi

implementato in un linguaggio non specificato.

- TAG : letteralmente *etichetta* viene usato per indicare gli elementi nei linguaggi di marcatura.
- UML: Unified Modelling Languege http://www.uml.org/
- CONNEG: Content Negotiation Working Group, gruppo di lavoro all'interno dell'IETF che si è occupato della stesura degli RFC relativi al sistema delle Media Feature Tag, un sistema di descrizione dei dispositivi multimediali.
- IETF: Internet Engineering Task Force; gruppo che si occupa della standardizzazione dei vari protocolli che nascono per internet tramite gli RFC. http://www.ietf.org/
- IANA: Internet Assigned Names Authority; l'ente ufficiale responsabile dell'assegnazione dei nomi, settori, indirizzi in rete e protocolli in Internet. http://www.iana.org/
- RFC: Request For Comments; presi quasi come standard descrivono i protocolli relativi al mondo internet, e presentano nuove proposte per la standardizzazione. http://www.ietf.org/rfc.html
- ASN Abstract Syntax Notation; sintassi e standard atta alla descrizione di strutture dati complesse in forma numerica codificata.
- WAP: Wireless Application Protocol; protocollo relativo alle trasmissioni tramite cellulare.http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html
- CGI Common Gateway Interface; standard per interfacciare applicazioni esterne con una applicazione server.
- JSP Java Server Pages; standard per ottenere pagine dinamicamente usando linguagio java all'interno di pagine HTML. Associato alla tecnologia delle Java Servlet.
- Java Servlet; standard che realizza le CGI mediante linguaggio Java.

Bibliografia

[MPROF] "Modelli di Profilazione e Sistemi di Gestione di Profili per Terminali Mobili" Alessandra Agostini, Claudio Bettini, Nicolò Cesa-Bianchi,Dario Maggiorini,Daniele Riboni; Università di Milano; Napoli 17/18 giugno 2003

[EXPCCPP] "Experience in Using CC/PP in Context-Aware Systems" Jadwiga Indulska, Ricky Robinson, Andry Rakotonirainy, Karen Hericksen; School of Information Technology and Electrical Engineering The University of Queensland Australia

[NEG] "The Negotiation of Multimedia Content Services in Heterogeneous environments" Tayeb Lemlouma and Nabil Layaïda; OPERA Project, INRIA Rhône-Alpes Montbonnot Saint Martin France

[SQOSA] "A Survey of QoS Architectures" Cristina Aurrecoechea, Andrew T. Campbell,Linda Hauw ;Center for Telecommunication Research, Columbia University, NewYor k, USA http://www.etr.columbia.edu/comet/members.html ;Multimedia Systems (1998) 6:138-151; Multimedia Systems ©Springer-Verlag 1998

[SQOSMCE] "A Survey of Quality of Service in Mobile Computing Environments" Dan Chalmers and Morris Sloman;Imperial College London; IEEE Communications Surveys, Second Quarter 1999 http://www.comsoc.org/pubs/surveys

[USRC] "Storing and Accessing User Context" Stéphanie Riché,Gavin Brebner; HP Laboratories Grenoble France ;M.-S. Chen et al. (Eds.): MDM 2003, LNCS 2574, pp. 1-12, 2003; ©Springer-Verlag Berlin Heidelberg 2003

[CAAPP] "An Architecture for the Effective Support of Adaptive Context-Aware Applications" Christos Efstratiou,Keith Cheverst,Nigel Davies and Adrian Friday; Distributed Multimedia Research Group, Department of Computing, Lancaster University Bailrigg, Lancaster; 2001

[HTMXML] "HTML e XML per principianti" Michael Morrison; © 2001 Mondadori Informatica cap 17-19

[QOSSL] "QoS Specification Language for Distributed Multimedia Applications: A Survey and Taxonomy" Jingwen Jin and Klara Nahrstedt ; University of Illinois at Urbana-Champaign, IEEE Annals of the History of Computing, 2004;July-September 2004 ; http://www.computer.org/annals

[UPCN] "Universal Profiling for Content Negotiation and Adaptation in Heterogeneous

Environments" Tayeb Lemlouma and Nabil Layaïda; OPERA Project, INRIA Rhône-Alpes Montbonnot Saint Martin France

[CAAMD] "Context-Aware Adaptation for Mobile Device" Tayeb Lemlouma and Nabil Layaïda; WAM Project,INRIA,Zirst 655 Avenue del l'Europe ,Montbonnot,Saint Martin France

[CCPPUAP] "CC/PP and UAProf: Issues,Improvements and Future Directions. Technical Report HPL-2002-35" Dr. Mark H.Butler; Hewlett Packard Laboratories,Bristol UK; 1 February 2002

[CCPPPS] "Composite Capability/Preference Profiles (CC/PP) Processing Specification" Luu Tran, Mark Butler, Steve Geach et al. ;Version 1.0 September 15, 2003

[MUMT] "Gestione di flussi multimediali e reti integrate fisse e mobili" Luca Foschini; Università di Bologna, Tesi di laurea; Anno accademico 2002-2003

[CACHED] "Caching di presentazioni multimediali e studio delle politiche di replacement" Simone Bertorelli; Università di Bologna, Tesi di laurea

[CACHEM] "Definizione e caching di metadati per presentazioni multimediali" Vincenzo Borelli; Università di Bologna, Tesi di laurea

[MPEG21O] "MPEG-21 Overview v.5" Jan Bormans, Keith Hill; Requirement Group; Shanghai, October 2002

[MPEG21G] "MPEG-21 Goals and Achievements" Ian Burnett,Rick Van de Walle, Keith Hill, Jan Bormans, Fernando Pereira; IEEE Multimedia; October-December 2003

[MPEG7O] "MPEG-7 Overview (version 9)" Josè M. Martinez; Requirements Group; Pattaya,March 2003

[MUM] "Mobile agent based Ubiquitous multimedia Middleware" ; http://lia.deis.unibo.it/Research/MUM/

[XMLNS] "Namespaces in XML" http://www.w3.org/TR/REC-xml-names/

[RFC2119] "Key words for use in RFCs to Indicate Requirement Levels"; http://www.ietf.org/rfc/rfc2119.txt?number=2119

[RFC2506] "Media Feature Tag Registration Procedure" http://www.ietf.org/rfc/rfc2506.txt?number=2506

[RFC2533] "A Syntax for Describing Media Feature Sets" http://www.ietf.org/rfc/rfc2533.txt?number=2533

[RFC2534] "Media Features for Display, Print, and Fax"

http://www.ietf.org/rfc/rfc2534.txt?number=2534

[RFC2531] "Content Feature Schema for Internet Fax"

http://www.ietf.org/rfc/rfc2531.txt?number=2531

[RFC2396] "Uniform Resource Identifiers (URI): Generic Syntax"

http://www.ietf.org/rfc/rfc2396.txt?number=2396

[RFC2068] "Hypertext Transfer Protocol – HTTP/1.1";

http://www.ietf.org/rfc/rfc2068.txt?number=2068

[RFC] "Hypertext Transfer Protocol – HTTP/1.0"

http://www.ietf.org/rfc/rfc1945.txt?number=1945

[W3C] "W3C World Wide Web Consortium"; http://www.w3.org/

[CCPP] "CC/PP Composite Capabilities / Preferences Profile";

http://www.w3.org/Mobile/CCPP/

[CCPPex] CC/PP exchange protocol based on HTTP Extension Framework

; http://www.w3.org/1999/06/NOTE-CCPPexchange-19990624

[UAProf] OMA/WAP Forum UAProf Specification;

http://www1.wapforum.org/tech/documents/WAP-248-UAProf-20011020-a.pdf

[CCPPreq] Composite Capabilities/Preference Profiles: Requirements and Architecture;

http://www.w3.org/TR/2000/WD-CCPP-ra-20000721/

[OMA] "Open Mobile Alliance";http://www.openmobilealliance.org/index.html

[WAP] "Wireless Applicazion Protocol";

http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[RDF] "RDF Resource Description Framework"; http://www.w3.org/RDF/

[IETF] "IETF The Internet Engineering Task Force

" http://www.ietf.org/

[RFC] "RFC IETF Request for Comments" http://www.ietf.org/rfc.html

[JAVA] http://java.sun.com/

[MPEG] http://www.chiariglione.org/mpeg/ http://www.mpeg.org/MPEG/

[JSR188] "Java Specification Rquest 188";http://www.jcp.org/en/jsr/detail?id=188

[SUN] "Sun Microsystems, Inc.";http://www.sun.com/

[TOMCAT] "Apache Tomcat ";http://jakarta.apache.org/tomcat/index.html

[HTTPCLI] "Jakarta Commons: HTTP Client";

http://jakarta.apache.org/commons/httpclient/