INDICE

Introduzione……...8

1. Adattamento di servizi multimediali sulla base di un sistema di località…….……...…….10

1.1 Definizioni fondamentali…..……..11
1.1.1 Flusso……..…….…...……..…..….11
1.1.2 Larghezza di banda…….12
1.1.3 Parametri caratterizzanti i flussi: latenza, jitter, loss rate e skew…….....…….13
1.2 Applicazioni multimediali: problematiche e linee evolutive……15
1.2.1 Risorse fisiche per le applicazioni multimediali……..15
1.2.2 Estensioni dei sistemi distribuiti per la gestione di applicazioni multimediali…….…...……17
1.2.3 Mobilità di utente e terminale…...……...……..20
1.3 Modelli di Comunicazione e Programmazione nei Sistemi Distribuiti.....21
1.3.1 Modelli di comunicazione…....…….23
1.3.1.1 Modello cliente/servitore……...…….…..…….23
1.3.1.2 Modello a memoria condivisa……..24
1.3.1.3 Modelli basati sulla mobilità di codice……25
1.3.2 Mobilità di codice…...…….…..……26
1.3.2.1 Mobilità dello stato di esecuzione……27
1.3.2.2 Mobilità del codice……..……29
1.3.2.3 Gestione dello spazio dei dati……..…....30
1.3.2.4 Paradigmi di mobilità di codice……...30
1.3.2.4.1 Remote Evalutation (REV)…….…….....31
1.3.2.4.2 Code On Demand (COD)…….32
1.3.2.4.3 Mobile Agent (MA)…...……..32
1.3.2.5 Sicurezza……..33
1.4 Conclusione…….....34

2. Definizione di un modello per un sistema di località e QoS based routing….…….……35

2.1 Modelli per un Overlay Network……...…….35
2.1.1 Struttura ad Anello……..…..…...36
2.1.2 Struttura a Stella……..37
2.1.3 Struttura a Bus unico condiviso……..37
2.1.4 Struttura a Bus Multipli…….....38
2.1.5 Struttura ad albero…….....39
2.1.6 Struttura a mesh e ipercubi……..40
2.1.7 Struttra a grafo.…….…..….....40
2.1.8 Topologie dimaniche e statiche…….....42
2.2 Modello per un Sistema di Località……42
2.2.1 Concetto di Vicinanza……..43
2.2.2 I parametri di una Località……...45
2.2.2.1 Sovrapposizione e centratura di una località…...46
2.2.2.2 Dimensione di una località……..48
2.3 QoS-based routing……..…….49
2.3.1 Metriche, euristiche e definizione dei percorsi…….....49
2.3.2 Approccio al problema della QoS nella pratica effettiva……..…….55
2.3.2.1 Il protocollo Private Network-Network Interface……..……56
2.3.2.2 Ticket based probing protocol……..…...57
2.3.2.3 Logical Application Stream Model…….58
2.3.2.4 Il middleware mobiware…….….59
2.4 Conclusione…….....62

3. SOMA e MUM…….……..…...…63

3.1 La Piattaforma SOMA…..……..63
3.1.1 Astrazioni di località e topologia in SOMA……...65

3.1.2 Introduzione all'architettura di SOMA…..…….....68
3.1.2.1 L' Ambiente di Esecuzione…..……68
3.1.2.2 Identificazione di Agenti e Place…….....69
3.1.2.3 Comunicazione in SOMA……..…….....69
3.1.2.4 Sicurezza in SOMA…..…….....70
3.1.3 Dettagli della Piattaforma SOMA……...…71
3.1.3.1 Gli Agenti…….71
3.1.3.2 L'accesso ai Servizi : l' Environment……..72
3.1.3.2 Gestore degli Agenti……73
3.1.3.3 Informazioni su Place e Domini : l'Information Service……..…73
3.1.3.4 Interazione tra Place : gli oggetti Command.....76
3.1.3.5 Gestore di Rete…….....77
3.2 Il middleware MUM……77
3.2.1 Scenario per lo sviluppo del MiddleWare……...78
3.2.2 Entità per la fruizione del materiale multimediale…...79
3.2.3 Le caratteristiche del Middleware MUM…….80
3.2.3.1 Movimento degli utenti……80
3.2.3.2 Movimento dei terminali…...…….....80
3.2.3.3 Inizializzazione e riorganizzazione dinamica del sistema….……..……81
3.2.3.4 Gestione delle risorse……...81
3.2.3.5 Struttura a livelli…...…….....82
3.2.4 La realizzazione del Middleware…….83
3.2.4.1 Servizio per l'istanziazione di entità…...….....83
3.2.4.2 Servizio di downloading del software…...…….83
3.2.4.3 Servizio di attivazione delle entità multimediali…….....84
3.2.4.4 Schema per la gestione delle risorse…...……85
3.2.4.5 Architettura delle entità che effettuano lo streaming…….……86
3.2.4.6 Servizio di inizializzazione e riconfigurazione dinamica del sistema……..…….....87
3.2.4.7 Sottosistema di gestione della qualità di servizio…….89

3.2.4.8 Schema di supporto per la mobilità utente……..91
3.2.4.9 Servizi di supporto alla mobilità del terminale…….....92
3.2.4.10 Schema per la mobilità del terminale……93
3.3 Conclusione……...94

4. Realizzazione del sistema di località in SOMA……..95

4.1 Motivazioni per una una nuova entità come gestore delle località…….....95
4.2 Modifiche alla piattaforma SOMA……96
4.2.1 La classe Environment…….97
4.2.2 Modifiche per il DomaniNameService……97
4.2.2.1 La registrazione di un DefaultPlace nel DNS…..98
4.2.2.2 La fase di Recovery e le modifiche al DNS…….99
4.2.3 Il NetworkManager……....100
4.3 Implementazione del sistema di località…….....100
4.3.1 Centratura e raggio delle località...……....100
4.3.2 I servizi di una località…..……....103
4.3.3 Le entità che compongono una località……..…..104
4.3.4 I servizi dell' interfaccia LocalityService…..….....105
4.3.5 I servizi di località non accessibili in LocalityService.....106
4.3.6 Le strutture dati delle località……..108
4.3.7 I componenti del manager di località……....112
4.3.8 I meccanismi di Inizializzazione e Registrazione….114
4.3.8.1 Fase di inizializzazione…….114
4.3.8.2 Fase di registrazione……..115
4.3.8.3 Creazione di una nuova località……...…….116
4.3.8.4 Motivi della sequenzializzazione…….…….121
4.3.8.5 Creazione di un Place normale…...……..123
4.3.9 Aggiornamento delle località.……...124
4.3.9.1 Il probing……...125

4.3.9.2 La politica di notifica degli eventi……....126
4.3.9.3 Politica di probing globale…….....127
4.3.9.4 Politica di Probing locale…..……129
4.3.9.5 Politica di notifica di eventi…...……..131
4.3.10 Il servizio di recovery…..…….....134
4.3.10.1 Il protocollo di recovery del sistema….....135
4.4 Conclusione..……...……....140

5 Modello per l' Adattamento dei Flussi Multimediali alle Condizioni del Sistema……...……141

5.1 Modello per la riconfigurazione del sistema e l'adattamento dei flussi……..141
5.1.1 Riconfigurazione del sistema…….....141
5.1.2 Avvio del processo di riconfigurazione del sistema.....142
5.1.2.1 Riconfigurazione su richiesta del client.....143
5.1.2.2 Riconfigurazione del singolo flusso……...143
5.1.2.3 Riconfigurazione di un insieme di flussi…..….144
5.1.2.4 Concorrenza tra gli eventi di riconfigurazione…….145
5.1.3 Fasi della Riconfigurazione…….....146
5.2 Modello di descrizione dei flussi…….....149
5.2.1 Il Multimedia Active Component……..…154
5.2.2 Active Client…….....155
5.2.3 Active Proxy.…….155
5.2.4 Active Server.……156
5.2.5 Stand Alone Component…….……...157
5.3 I Protocolli di Riconfigurazone…….157
5.3.1 Reconfiguration Action…….157
5.3.2 Remote Reconfiguration Action…….……...158
5.3.3 Possibili Azioni di Riconfigurazione locali e remote……..160

5.3.4 Protocollo per la scelta del percorso di un flusso nell'ambito di una località……..…….....161
5.4 Architettura del modello…….....171
5.4.1 Il sottosistema per la generazione dei segnali……172
5.4.2 Reconfiguration Planner……172
5.4.3 ReconfigurationManager……...172
5.4.4 I Reconfiguration Listener…….....173
5.5 Conclusione……..174

6. Realizzazione del modello di Riconfigurazione su MUM……...175

6.1 L'implementazione del modello di riconfigurazione…...175
6.1.1 Il sottosistema per la generazione dei segnali…….....176
6.1.2 Il Multimedia Services Manager…….178
6.1.3 Il Reconfiguration Planner……..……181
6.1.4 Il Reconfiguration Manager……182
6.1.5 I Reconfiguration Listeners...……..……183
6.1.6 Gli Active Component…...…….184
6.1.6.1 I Component Identifier…….……185
6.1.6.2 Active Client, Proxy, Server e StandAloneComponent…….……...187
6.1.7 Le Azioni di Riconfigurazione……..….192
6.1.8 I Reconfiguration Command……..194
6.2 Un esempio pratico : la New Path Action…….195
6.2.1 Presentazione dell'applicazione utente……..195
6.2.1.1 Java Media Framework……..195
6.2.1.2 Il Protocollo RTP……..……198
6.2.1.3 Le unità Client, Proxy, Server……...199
6.2.2 La realizzazione della New Path Action…….....202
6.2.3 Le modifiche ai componenti Client, Proxy e Server…….....206
6.2.4 Test dell'azione NewPathAction……...……208
6.2.5.1 Configurazione del test e risultati…….…….209

6.3 Lo schema di integrazione dei QoS Manager…...……211

Conclusioni…….213

AppendiceA……216

Bibliografia…….....227

Introduzione.

Grazie alle nuove capacità tecnologiche, possibilità sconosciute fino a pochi anni or sono, quali la fruizione su di un dispositivo mobile di flussi multimediali scaricati run-time dalla rete, o la realizzazione di video chiamate, hanno concentrato l'attenzione al problema della distribuzione di flussi multimediali.
Conseguentemente sono cresciute e si sono sviluppate tematiche connesse alla distribuzione di materiale multimediale la principale delle quali è certamente quella di garantire la qualità del servizio (Quality of Service, QoS) fruito dagli utenti. Oltre a ciò, sempre più utenti vogliono oggi accedere ai proprio servizi indipendentemente dal terminale da loro utilizzato e mantenere aperte le proprie sessioni mentre si muovono fra terminali diversi, si pensi ad esempio ad un utente che voglia spostare una telefonata dal proprio telefono cellulare al proprio PC. L'insieme di queste nuove esigenze, ha portato gli sviluppatori di software a esplorare soluzioni innovative: ad adottare paradigmi di programmazione avanzati, a definire nuovi protocolli e a creare infrastrutture distribuite che si interpongono tra il sistema sottostante e le applicazioni utente, fornendo a queste ultime servizi in maniera trasparente.
In particolare, l'imprevedibilità del comportamento del sistema nel suo complesso e la continua fluttuazione delle risorse di sistema, richiedono, per mantenere il livello di qualità desiderato, un continuo monitoraggio del sistema ed eventualmente, qualora le condizioni peggiorino, l'adozione di opportuni aggiustamenti dell'infrastruttura. Partendo da questi presupposti, la tematica fondamentale sulla quale si concentrerà questa tesi è quella della riconfigurazione di una infrastruttura per la distribuzione di flussi multimediali e dei percorsi di distribuzione di tali flussi.
L'intento che ci si propone è poi quello di realizzare una soluzione che mantenga limitato sia l'overhead che l'impegno delle risorse del sistema, in coerenza con il principio di minima intrusione. In particolare, un tema molto importante sarà relativo alle strategie

adottate per raccogliere e mantenere informazioni relative allo stato delle risorse del sistema distribuito. A tale proposito vale la pena notare come sia auspicabile che un deterioramento nelle condizioni delle risorse abbia un impatto locale e non sulla globalità del sistema stesso. Grazie a questa proprietà è possibile allora pensare all'adozione di strategie informate a basso overhead basate sul reperimento di informazioni sullo stato delle risorse solo in un intorno della zona nella quale si sia verificato il problema.
In questo contesto si inserisce l'attuale progetto di tesi. Esso consiste nella realizzazione di un sistema di località, in un sistema distribuito, che permetta la definizione di efficaci protocolli per il QoS-based routing, e nella realizzazione delle strutture che consentano un servizio di riconfigurazione degli stream multimediali.
La tesi sarà quindi organizzata come segue. Nel capitolo 1 verranno introdotte le esigenze delle applicazioni multimediali; il problema della QoS, la mobilità di utente e terminale e i possibili modelli di comunicazione nei sistemi distribuiti. Nel capitolo 2, vengono presentati alcuni modelli di topologia per le reti, il concetto di località, e alcuni esempi dei diversi approcci al problema della QoS. Nel capitolo 3 introdurremo la piattaforma SOMA e il middleware MUM. La piattaforma SOMA è infatti la base su cui è stato realizzato il sistema di località che verrà presentato nel capitolo 4 mentre MUM è un middleware, per la distribuzione di flussi multimediali in ambiente distribuito, che verrà integrato con un sistema per la riconfigurazione dei flussi, il cui modello sarà presentato nel capitolo 5. Nel capitolo 6, infine, verrà descritta l'implementazione del modello di riconfigurazione visto, insieme alla presentazione di un caso di studio.

CAPITOLO 1.

1 Adattamento di servizi multimediali sulla base di un sistema di località.

Negli ultimi anni, molto lavoro è stato fatto per lo sviluppo di infrastrutture che supportino applicazioni multimediali.
Nell'ambito delle applicazioni multimediali, il problema principale legato alle strutture storicamente presenti, è dovuto alla natura di tipo best-effort delle reti. Le applicazioni multimediali, infatti, sono soggette a vincoli di tempo più o meno stretti e richiedono delle garanzie per il recupero delle informazioni dalla rete: garanzie che le reti best-effort non sono in grado di fornire. Il superamento dei limiti di qualità imposti da questo tipo di reti diviene quindi l'obiettivo principale di tali infrastrutture. Oltre a questo naturalmente dovranno anche fornire servizi di supporto; il tutto, nell'ottica di garantire livelli di prestazione e servizi predefiniti.
Nell'ambito di un sistema le cui condizioni possono però variare in maniera impredicibile e in cui temi come la mobilità di utente e di terminale divengono sempre più importanti, questo problema non può essere risolto a priori. Nasce cioè l'esigenza di operare dei cambiamenti durante lo svolgimento stesso delle applicazioni; di adattare, in sostanza, le applicazioni stesse alle condizioni del sistema e alle esigenze e richieste degli utenti. Tale obiettivo è però in modo imprescindibile, legato al problema della raccolta delle informazioni e del monitoraggio del sistema stesso.
In tale ottica, nasce questo lavoro di tesi, che si propone come obiettivo quello della definizione di un modello per la raccolta delle informazioni e del monitoraggio del sistema basato sul concetto di località e nella realizzazione di un servizio di riconfigurazione e adattamento delle applicazioni multimediali.

Questo capitolo intende creare il background necessario per la comprensione delle tematiche trattate ed è quindi un capitolo introduttivo, in cui intendiamo prendere in considerazione i concetti fondamentali per la realizzazione di applicazioni multimediali in ambito distribuito e i temi ad esso connessi.

1.1 Definizioni fondamentali.

In questa sezione intendiamo fornire alcune definizioni fondamentali per la comprensione degli argomenti che saranno trattati in seguito. In particolare vengono definiti il concetto di flusso e alcuni parametri che saranno richiamati nel corso della trattazione.

1.1.1 Flusso

Il flusso è un'astrazione che viene introdotta per specificare la fruizione di un singolo oggetto multimediale. Con fruizione si intende il recapito e il rendering dell'oggetto multimediale richiesto.
Per singolo oggetto multimediale si intende un particolare tipo di dato, caratterizzato dal fatto di variare con continuità e di essere soggetto a vincoli di tempo; tale oggetto multimediale potrebbe essere per esempio un video, un audio, o un generico flusso di dati.
Questa definizione, come vedremo, è di importanza fondamentale, nel momento in cui si inizi a esigere qualità di servizio (Quality of Service, in seguito indicata con l'acronimo QoS). Il singolo flusso può infatti essere considerato come unità fondamentale cui riferire la QoS.
I flussi sono l'entità minima identificabile per la trasmissione di oggetti multimediali, e possono essere sia di tipo unicast (da 1 sorgente a 1 destinatario) che di tipo multicast (da 1 sorgente a N destinatari). Il flusso attraversa diverse entità (transitando dalla sorgente al destinatario/destinatari) ognuna delle quali, in un'ottica di QoS, dovrà riservare una parte delle risorse disponibili per processare il flusso stesso.

Vediamo ora quale sia la semantica che regola la trasmissione dei flussi. La definizione di un oggetto multimediale data sopra sottolinea il fatto che esso è, per sua natura, continuo; questo aggettivo si riferisce alla percezione che l'utente finale ha dello streaming multimediale. In realtà, però, è noto che, per poter essere processati da un elaboratore, gli oggetti multimediali devono essere discretizzati e quantizzati.
Ne segue che, se consideriamo il flusso ad un livello di astrazione più basso, lo possiamo pensare come una sequenza discreta di dati soggetti a vincoli di tempo: l'istante di ricezione degli stessi da parte del ricevente ne determina la validità. In altri termini, se un dato viene ricevuto in tempo utile per il suo utilizzo da parte del ricevente, verrà consumato, altrimenti sarà scartato e non sarà ritenuto valido.
Una semantica di questo tipo viene detta isocrona. Perciò diciamo che i flussi multimediali sono per loro natura isocroni.

1.1.2 Larghezza di banda.

La larghezza di banda è il parametro che misura la portata di un canale trasmissivo digitale, poiché definisce il numero massimo di dati che possono transitare nella sezione nell'unità di tempo. Solitamente l'unità di misura è data da bit/secondo (bps), parametro importante perché rappresenta un vincolo assai stringente per la trasmissione di oggetti multimediali.
Un flusso multimediale, infatti, richiede solitamente una banda molto ampia e ormai da diversi anni sono allo studio algoritmi di compressione che hanno come scopo quello di diminuire i requisiti di banda per la trasmissione di oggetti multimediali, soprattutto di tipo video.
L'aspetto negativo dell'uso di tali algoritmi è che richiedono (solitamente dal lato della sorgente) una computazione ulteriore piuttosto consistente, occupando la risorsa più costosa, la CPU.

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

La **latenza** è il parametro che misura il ritardo introdotto dalla linea di trasmissione, cioè il tempo impiegato dai dati discretizzati e impacchettati per giungere dalla sorgente al ricevente.
Tale parametro è rilevante soprattutto nelle applicazioni che richiedono l'interazione con un sistema remoto, o con altri utenti a distanza. A tale proposito è stato valutato che il ritardo massimo tollerabile per una conversazione sia di 150 ms, mentre per una applicazione di tipo Video on Demand (VoD), nella quale si interagisce con un "videoregistratore remoto", sia al massimo di 500 ms. Ovviamente la latenza non è costante, ma soggetta a variazioni, sarà quindi rappresentata da una variabile aleatoria.
Un parametro ancora più interessante per la caratterizzazione dei flussi è il **jitter**. Tale parametro misura la varianza della latenza, cioè lo scostamento rispetto al valor medio della stessa.

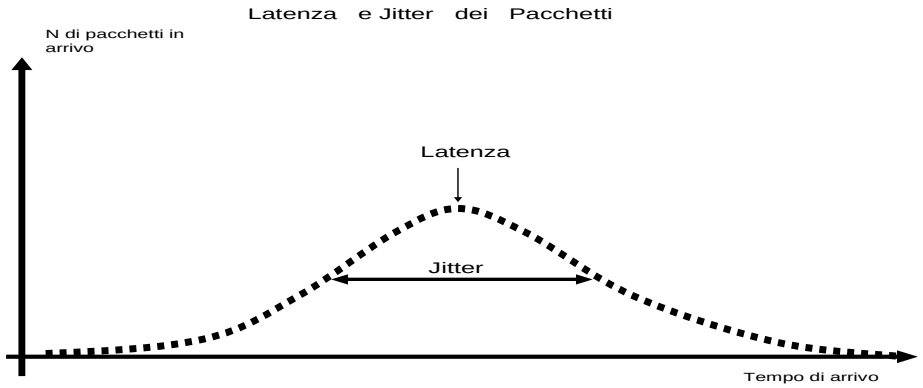


Fig. 1.1

In figura 1.1, tratta da [BAF01], si può osservare la rappresentazione analitica di quanto espresso fin qui a parole. Viene rappresentata la latenza dei pacchetti dati come densità di probabilità, e il jitter.
La freccia contrassegnata con "Latenza" rappresenta il valor medio della latenza. La situazione ideale si verificherebbe se tutti i

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

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

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

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

L'ultimo parametro che consideriamo è lo **skew**. Fino ad ora abbiamo sempre considerato un singolo flusso, con i relativi problemi di sincronizzazione. Esiste però la possibilità che più oggetti multimediali costituiscano insieme un unicum, che verrà definito *presentazione multimediale*. In tal caso parliamo di sincronizzazione inter-oggetto, contrapponendola alla sincronizzazione intra-oggetto che si riferisce al singolo flusso. Lo skew quantifica (come intervallo di tempo) lo sfasamento nell'arrivo di flussi diversi facenti parte della stessa presentazione, ad esempio un flusso audio e video per un film.

Con quest'ultima definizione è conclusa questa sezione nella quale sono state date alcune definizioni di base sul concetto di flusso,

oggetto multimediale, presentazione multimediale e la loro caratterizzazione.

1.2 Applicazioni multimediali: problematiche e linee evolutive

Denominatore comune di tutti i tipi di applicazioni che lavorano nell'ambito del multimediale, è l'elevata richiesta di risorse e di garanzie di qualità di servizio che aumentano all'aumentare della criticità e dei vincoli di tempo dell'applicazione.
Se ci si chiede per quale motivo questo tipo di applicazioni non siano ancora disponibili dappertutto e con accesso da un qualsiasi device si capisce come i motivi siano da ricercare principalmente nell'architettura del software di base al di sopra del quale è sviluppata la maggior parte dei sistemi distribuiti, e nel modo in cui Internet e i suoi protocolli siano stati pensati sin dall'inizio della loro esistenza. Come vedremo, i limiti con i quali ci si confronta sono sia di tipo fisico (carenza di risorse), sia dovuti al software.
Per esempio, l'odierna comunicazione di rete si appoggia largamente sulla suite di protocolli TCP/IP, che non dà alcuna garanzia di qualità di servizio. Nasce dunque la necessità di introdurre nell'architettura software dei sistemi distribuiti, un nuovo livello, occupato dai cosiddetti middleware. Queste entità sono, infatti, dedicate a sopperire a molte delle mancanze delle attuali architetture , fornendo alle applicazioni multimediali che vengono sviluppate al di sopra di esso garanzie in termini di QoS e utili servizi per portare a termine i loro obiettivi.

1.2.1 Risorse fisiche per le applicazioni multimediali.

Uno dei fattori che maggiormente limitano la diffusione delle applicazioni multimediali è la grande richiesta di risorse. In questa sezione vogliamo indicare quali sono le risorse fisiche più sollecitate nel recapito e nell'utilizzo di un flusso multimediale

Le risorse che prendiamo in considerazione sono quattro; le prime tre sono strettamente legate al recapito e al consumo di un flusso multimediale; l'ultima viene invece presa in considerazione pur non essendo direttamente coinvolta in tali meccanismi.
Le risorse che prendiamo in considerazione sono: la CPU, la banda trasmissiva, la memoria primaria e la memoria secondaria.

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

   Ciò solo per citare alcune delle operazioni più costose operate durante la trasmissione di un flusso multimediale. Solitamente i requisiti, per un dato tipo di CPU, vengono espressi come frazioni di tempo di un periodo. Per esempio un'applicazione potrebbe dichiarare che necessita di 10 ms di CPU, ogni 100 ms; oppure, sotto forma percentuale che necessita del 10% dei cicli del microprocessore.
2) La **banda trasmissiva** è un'altra risorsa che solitamente viene fortemente richiesta. Basti pensare che un filmato non compresso con qualità video TV standard richiede approssimativamente una banda pari a 120 Mbps (Mega bit per second), che eccede la capacità di molte delle Ethernet oggi utilizzate, pari a 100 Mbps. Come già accennato sopra (al punto 1.1.2), l'utilizzo degli algoritmi di compressione riduce notevolmente la richiesta di banda. Per esempio, utilizzando la prima delle codifiche studiate dal Moving Picture Experts Group (MPEG), cioè MPEG-1, possiamo abbassare questo valore a 1,5 Mbps. Naturalmente queste trasformazioni richiedono un costo, cioè pesano maggiormente sulla risorsa CPU.
3) La terza risorsa che consideriamo è la **memoria primaria**. La trasmissione e visualizzazione dei flussi multimediali richiede un'elevata quantità di memoria. Come già visto, il ricevente

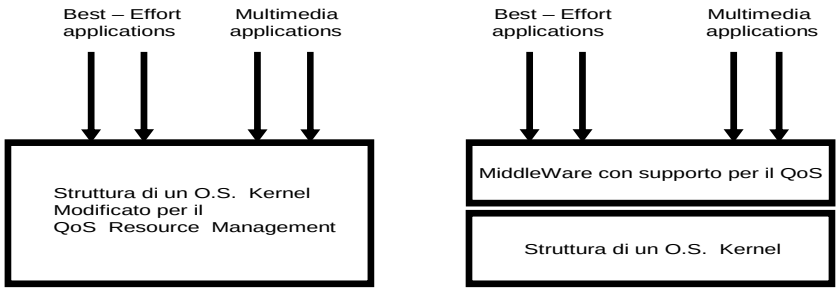
dovrà bufferizzare i dati in arrivo per poter continuare la visualizzazione dell'oggetto multimediale, anche in presenza di momentanei problemi di trasmissione. Inoltre ci sarà bisogno di altri buffer per salvare i dati durante la loro trasformazione ad opera dei CODEC (cioè i moduli di codifica/decodifica, che operano le compressioni/decompressioni dei dati), e infine, per il rendering degli oggetti multimediali in arrivo.

4) La quarta risorse che prendiamo in considerazione è infine la **memoria secondaria**. Come abbiamo già anticipato, mentre le tre precedenti risorse sono direttamente legate ai meccanismi di consumo e recapito dei flussi multimediali, non è così per questa ultima risorsa. Essa è però legata ad un meccanismo che negli ultimi anni sta assumendo importanza sempre maggiore per quanto riguarda la creazione di strutture per il supporto di applicazioni multimediali; cioè il meccanismo di caching. In sostanza, nel caso di presentazioni multimediali particolarmente richieste, vengono sviluppati meccanismi per la replicazione delle stesse in prossimità del luogo di origine delle risorse consentendo un utilizzo di risorse del sistema distribuito molto inferiore nelle fasi di recapito dell'oggetto multimediale. Nonostante la discussione di tale meccanismo esuli dagli obiettivi di questo progetto, è comunque necessario tenerlo in considerazione nell' ambito di un modello di raccolta delle informazioni sulle risorse del Sistema.

1.2.2 Estensioni dei Sistemi Distribuiti per la gestione di applicazioni multimediali.

Nell'ambito degli attuali sistemi distribuiti, in realtà, non vi è ancora molta chiarezza su quale sia la strada migliore per la creazione di nuovi servizi in appoggio allo sviluppo di Applicazioni Multimediali. Né, tanto meno, stanno emergendo standard de facto. In generale, però, lo studio, nell'ambito dei sistemi distribuiti, di nuove possibilità per applicazioni Real Time e soft Real Time, confluisce verso due settori di ricerca distinti e complementari; il primo consiste nell'estensione delle architetture

software delle macchine per il supporto di questo nuovo genere di applicazioni. Il secondo, nella sviluppo di nuovi protocolli che sostituiscano o complementino quelli attualmente impiegati e ne sopperiscano le carenze.
Vediamo brevemente in cosa consistono questi due rami di ricerca. Nell'ambito dell'estensione delle architetture software delle piattaforme di rete si possono seguire due strade diverse; la prima consiste nell'estendere i sistemi operativi per supportare anche questo nuovo tipo di applicazioni, la seconda consiste nello sviluppare uno strato di middleware che realizzi il supporto necessario per la gestione delle risorse. Si può notare da subito come le due soluzioni proposte siano fra loro ortogonali e possano coesistere all'interno dello stesso sistema;
Nella prima ipotesi si va ad agire a livello di sistema operativo (SO) mettendo a disposizione dello sviluppatore opportune API (Application Programming Interface), che estendano il set di API usato solitamente. In questo modo i processi con vincoli di tipo Realtime (RT) o Soft-Realtime (SRT) possono usare il set di API esteso, mentre gli altri processi di tipo best-effort possono continuare ad usare il vecchio set di API. Nella seconda ipotesi si inserisce uno strato di middleware, che si occuperà di gestire in modo opportuno le varie richieste fra il sistema operativo e le applicazioni. La figura 1.2, rappresenta graficamente le due alternative.



(a) Multimedia Operating System (b) Multimedia Middleware

Fig. 1.2

Naturalmente per ognuna di queste due soluzioni esistono lati positivi e lati negativi. La prima soluzione richiede di intervenire sul kernel, modificando, o meglio estendendo lo scheduler dei processi e quello di input/output (I/O), mentre la seconda no.
Ragionando in un'ottica di accrescimento dei sistemi preesistenti, l'adozione della prima via non richiede alcuna modifica dell' applicativo preesistente potendo esso continuare ad appoggiarsi sulle vecchie API; la seconda, invece, a garanzia che i contratti fatti dalle applicazioni multimediali vengano onorati, presuppone che tutte le richieste passino attraverso il middleware, e ciò potrebbe portare alla modifica di molte delle applicazioni scritte in precedenza. Infatti, invece di utilizzare le API preesistenti, offerte dal S.O. , tali applicazioni dovrebbero essere nuovamente scritte in modo da utilizzare il set di API fornito dal middleware. Un'ultima differenza è che l'utilizzo della seconda alternativa consente di cambiare il sistema operativo sottostante senza che ciò influisca in alcun modo sulle applicazioni (basterà infatti che esistano delle implementazioni del middleware multipiattaforma). Un interessante confronto fra i pro e i contro di queste scelte architetturali si può trovare in [SHP02].
Per quanto invece riguarda l'altro ambito di ricerca ossia lo sviluppo di nuovi protocolli, nonostante lo stack OSI (si veda [OSI]) preveda una fase di negoziazione della QoS ai vari livelli, l'attuale stack TCP/IP ossia lo standard de facto, non prevede alcun controllo particolare per assicurare qualità. Al fine di colmare questa mancanza sono state avanzate varie proposte. Per esempio i Servizi Integrati (IntServ, [RFC2210], si veda inoltre il ReSerVation Protocol, RSVP, [RFC2205]) e i Servizi Differenziati (DiffServ, si veda [RFC2475]) proposti da IETF (Internet Engineering Task Force), sono stati pensati con l'intento di assicurare qualità di servizio per il singolo flusso (i primi) e per classi di flussi appartenenti alle stesse classi di servizio (i secondi). Inoltre, per quello che riguarda più specificatamente il dominio multimediale, sono stati proposti due protocolli, e cioè il RealTime Protocol (RTP, si veda [RFC1889]) e il RealTime Streaming

Protocol (RTSP, si veda [RFC2326]). Le ultime soluzioni sono quelle oggi utilizzate per assicurare QoS nella trasmissione di flussi multimediali, vengono sviluppate a livello applicativo.

1.2.3 Mobilità di utente e terminale.

Come abbiamo anticipato, lo sviluppo delle nuove applicazioni nell'ambito del Multimediale, deve tenere in considerazione due nuove tematiche oltre a quelle introdotte fino ad ora. Tali tematiche sono quelle della mobilità di utente e della mobilità di terminale. Proprio negli ultimi anni, tali tematiche sono divenute da comprimarie a fondamentali soprattutto per quanto riguarda lo sviluppo dei nuovi servizi sulle reti wireless. Non solo, per esempio, deve essere possibile fruire su un dispositivo portatile di una presentazione multimediale, ma deve anche essere possibile spostare la sessione su altri terminali (per esempio "arrivato a casa" un utente può decidere di utilizzare un terminale più adatto, come il proprio computer o televisore).
I middleware che prevedano entrambe queste possibilità devono anche prevedere che parte dei servizi di riconfigurazione deve essere accessibili alle entità in maniera diretta alle applicazioni. Nel caso della mobilità di terminale, in genere, sono i livelli inferiori dell'architettura ad occuparsi dell' identificazione della stazione trasmissiva migliore e della gestione dell'hand-off ed è quindi possibile integrare direttamente la fase di riconfigurazione del middleware con quella "fisica" e in maniera trasparente all'utente.
Nel caso invece di mobilità di utente, ciò non è possibile: è l'utente, infatti, che decide quando abbandonare un terminale e passare ad un altro. Tale evento, è assolutamente asincrono e imprevedibile per il sistema ed è l'utente pertanto a richiedere direttamente all'applicazione la riconfigurazione.
Generalizzando, la mobilità da un terminale a un altro può essere solo una di molte possibili richieste (a run–time) di un utente. E' quindi necessario che un middleware offra la possibilità alla

applicazioni costruite sopra di effettuare richieste dirette in termini di Riconfigurazione e Adattamento del proprio flusso.
Oltre a ciò, questi modelli di mobilità, rendono di fondamentale importanza la costituzione di servizi che rispondano ai principi di **Context** e **Location Awareness**. Con servizi Context Aware intendiamo il principio per cui l'architettura software che fornisce il servizio sia consapevole di quelle che sono le caratteristiche delle piattaforme utilizzate per la realizzazione del servizio stesso; conoscendo, quindi, le configurazioni dei dispositivi utilizzati dall'utente potrà ottimizzare il servizio adeguandolo a questi. Oltre a ciò, sempre nell'ambito dei servizi Context Aware è fondamentale importanza per la mobilità di terminale predisporre servizi di discovery in modo tale che i terminali appena giunti in una nuova località siano in grado di sapere quali servizi sono disponibili e quali no.
Il principio di Location Awareness, invece, consiste nel fornire informazioni di località fisica all'applicativo sovrastante.
Nell'ambito dello sviluppo di applicazioni multimediali, questo genere di informazioni, consente l'individuazione dei siti più vicini con la presentazione multimediale richiesta e di ottimizzare quindi la creazione dei percorsi per i flussi stessi.

1.3 Modelli di Comunicazione e Programmazione nei Sistemi Distribuiti .

Concludiamo questo capitolo introduttivo, con la presentazione dei modelli di Comunicazione e di Programmazione per sistemi distribuiti. Nell'ambito della programmazione di applicazioni e infrastrutture per la gestione e il delivery di flussi multimediali, i nuovi modelli di programmazione e comunicazione, possono svolgere un ruolo importante nella semplificazione delle architetture del sistema. In particolare, nello scenario che prendiamo in considerazione, gli stream multimediali sono consegnati a utenti che possono essere mobili, sia perché dotati di terminali mobili, che perché nomadici. In questo caso, per esempio, gli oggetti che effettuano il

delivery, possono essere realizzate come entità mobili consentendo un approccio più semplice al problema della riconfigurazione dei percorsi; in particolare, il modello ad agenti, risulta molto utile, in caso di mobilità, in quanto consente di "seguire" fisicamente lo spostamento dell'utente, lasciando alla piattaforma sottostante il compito di gestire il movimento dell'applicazione. Allo stesso modo, i modelli di mobilità di codice, consentono di spostare la conoscenza necessaria all'applicazione, da un luogo ad un altro (seguendo, ad esempio, il percorso del flusso multimediale); la loro utilità è però ancora maggiore quando si tratta di segnalare la presenza di particolari eventi ad altri nodi del sistema (per esempio quando si vuole segnalare la necessità di riconfigurare il percorso di uno stream); in questo caso è, infatti, possibile contemporaneamente segnalare la presenza dell'evento e fornire le istruzioni per la gestione dell'evento stesso.
In questo paragrafo esamineremo i modelli di comunicazione, dal più semplice e consolidato, cioè il tipico modello cliente e servitore, alle sue evoluzioni, fino a considerare i modelli di comunicazione che prevedono la mobilità di codice. Vedremo quindi come, il concetto di mobilità possa essere esteso, dalla mobilità di codice alla mobilità di un processo, e introdurremo quindi il concetto di mobilità debole e mobilità forte; infine considereremo un nuovo paradigma di programmazione che consiste nella programmazione ad agenti. Naturalmente, l'insieme di questi argomenti non può certamente essere trattato in maniera approfondita nell'ambito di poche pagine; il nostro attuale intento è infatti quello di soffermarci maggiormente sugli strumenti utilizzati nell'ambito del progetto svolto.
Anche se in misura diversa, i modelli di comunicazione, descritti nelle prossime sezioni, sono stati tutti utilizzati (a parte il modello a memoria condivisa che invece viene presentato per completezza). Tra questi, quelli sicuramente maggiormente utilizzati nell'ambito del progetto sono quelli relativi alla mobilità di codice che ci hanno consentito una notevole flessibilità e capacità espressiva.

In particolare, ci soffermeremo sulla presentazione del modello di programmazione ad agenti, che, come vedremo, riveste un ruolo particolarmente importante sia relativamente alla piattaforma di sviluppo che per il middleware di supporto alle applicazioni.
Cercheremo, anche in questo caso, di realizzare una panoramica completa, evitando gli aspetti legati alle singole piattaforme e implementazioni (aspetti che, nel nostro caso, verranno più avanti discussi quando si introdurranno, nel terzo capitolo, gli strumenti di sviluppo utilizzati).

1.3.1 Modelli di comunicazione

In questa sezione si considerano tre diversi modelli di comunicazione, cercando di evidenziarne pregi e difetti.

1.3.1.1 Modello cliente/servitore

Il primo modello che consideriamo è il modello cliente/servitore classico. Questo modello prevede l'interazione di due entità: il cliente, che richiede un servizio e il servitore che offre il servizio stesso. In figura 1.3 viene rappresentata questa situazione.

Fig. 1.3

Nella sua accezione più semplice il modello prevede che il cliente conosca direttamente il servitore, mentre il servitore non conosce direttamente il cliente che sta richiedendo il servizio e l'interazione è *sincrona* e *bloccante*.

Si utilizza, in questo caso, un modello **pull** in cui il cliente è caricato della responsabilità dell'ottenimento delle informazioni a cui è interessato. A livello logico, esiste un unico thread che viene eseguito, in parte sul lato client, e in parte sul lato server. L'esempio forse più noto di questo modello sono le chiamate di procedura remota (Remote Procedure Call) e il corrispettivo per la programmazione Object Oriented, cioè le chiamate di metodo remoto (Remote Method Invocation). Tali meccanismi permettono la comunicazione tra processi (le prime) ed oggetti (le seconde) che si trovano su macchine diverse, gestendo la conversione dei dati fra le diverse piattaforme e i dettagli di comunicazione in modo trasparente al programmatore, che si può perciò disinteressare di questi dettagli.
Il problema di questi modelli di programmazione distribuita è che lo stile di interazione è solitamente sincrono, richiede perciò al cliente di attendere la terminazione del servizio prima di continuare la computazione e non è possibile stabilire a priori il tempo che occorrerà per il completamento del servizio stesso. Per rispondere a questi problemi, oggi viene spesso utilizzato un modello **push**, in cui il cliente richiede un servizio, ma è il servitore ad avere la responsabilità del recapito del risultato.
Esempi dell'applicazione di questo modello sono i servizi di notifica degli eventi, che gestiscono l'invio messaggi, disaccoppiando gli interessati. Tali servizi vengono spesso utilizzati nei multimedia middleware per l'implementazione del resource monitoring distribuito.

1.3.1.2 Modello a memoria condivisa

Questo secondo modello di comunicazione che consideriamo unisce l'astrazione di memoria condivisa e quella di comunicazione. L'interazione tra i processi che risiedono sulle diverse macchine avviene accedendo all'informazione condivisa attraverso uno **spazio delle tuple**, cioè un insieme strutturato di relazioni intese come attributi e valori. Le piattaforme che intendono sfruttare questo modello di comunicazione, necessariamente

dovranno implementare un middleware strutturato in modo tale da dare ai processi l'illusione di memoria condivisa. A tale memoria i processi potranno poi accedere attraverso le primitive di lettura e scrittura delle tuple, come avverrebbe nel concentrato.
Vale la pena di evidenziare come, in scenari di mobile computing, questa astrazione consenta alcuni vantaggi come ad esempio durante le momentanee disconnessioni che si possono verificare.

1.3.1.3 Modelli basati sulla mobilità di codice

I modelli basati sulla mobilità di codice aggiungono flessibilità e adattabilità alle applicazioni distribuite e possono evitare la trasmissione di ingenti moli di dati, nei casi in cui si sia interessati solo ad un piccolo sottoinsieme di essi [GEK01]. Questo nuovo paradigma prevede infatti la spedizione non più dei soli dati, ma anche di codice in modo da "spostare l'intelligenza" là dove sia necessario eseguire una certa computazione. Esempi di applicazione di questo modello sono: il movimento di processi per operare bilanciamento di carico, la distribuzione di aggiornamenti software per quegli apparati che debbano sempre rimanere in funzione, senza dover mai essere spenti, l'information retrieval, l'esecuzione di task di controllo e monitoraggio su reti remote, ecc. A tutt'oggi non è però ancora stata trovata una cosiddetta "killer application" che abbia definitivamente stabilito l'adozione generalizzata di questo modello.
Infatti, anche se il modello offre allo sviluppatore notevoli potenzialità, e la possibilità di gestire interazioni che vanno ben al di là sia di quelle del modello client/server che di quello a memoria condivisa, esistono diverse problematiche che ne hanno finora limitato l'utilizzo generalizzato. Prima fra tutte è la richiesta di un ambiente di esecuzione uniforme al di sopra del quale fare eseguire il codice mobile. Tale richiesta rappresenta oggi una assunzione molto forte, vista l'enorme eterogeneità di piattaforme computazionali (è tuttavia da notare che nel nostro caso, tale assunzione è garantita dalla presenza del middleware a cui si fa riferimento; è in

questo caso, il middleware stesso che si occupa del superamento delle eterogeneità).
Oltre a problemi di natura tecnica è da evidenziare come ci siano forti resistenze all'utilizzo generalizzato di questo modello computazionale in quanto l'utilizzo di codice mobile, che in molti casi dovrà accedere alle risorse di sistema e in generale dovrà comunque utilizzare la risorsa CPU, pone non pochi problemi di sicurezza, monitoraggio e accounting.
In particolare, nell'ambito della sicurezza, si desidera consentire al codice "straniero" solo l'esecuzione di alcune azioni, accedendo solo alle risorse per cui è autorizzato. Oltre a ciò, il codice mobile, che verrà eseguito dovrà essere soggetto a monitoraggio; se la macchina non è di proprietà dell'utente che ha creato per cui tale codice opera, si deve poter quantificare l'ammontare delle risorse utilizzate in modo da addebitare tale costo a carico dell'utente stesso.
Il problema è che non esistono ancora dei paradigmi di programmazione atti a guidare lo sviluppo di applicazioni "sicure" che facciano uso di questo nuovo modello. E' tuttavia innegabile che tale modello, per capacità espressive e flessibilità, rappresenta uno strumento molto interessante nell'ambito della programmazione distribuita.

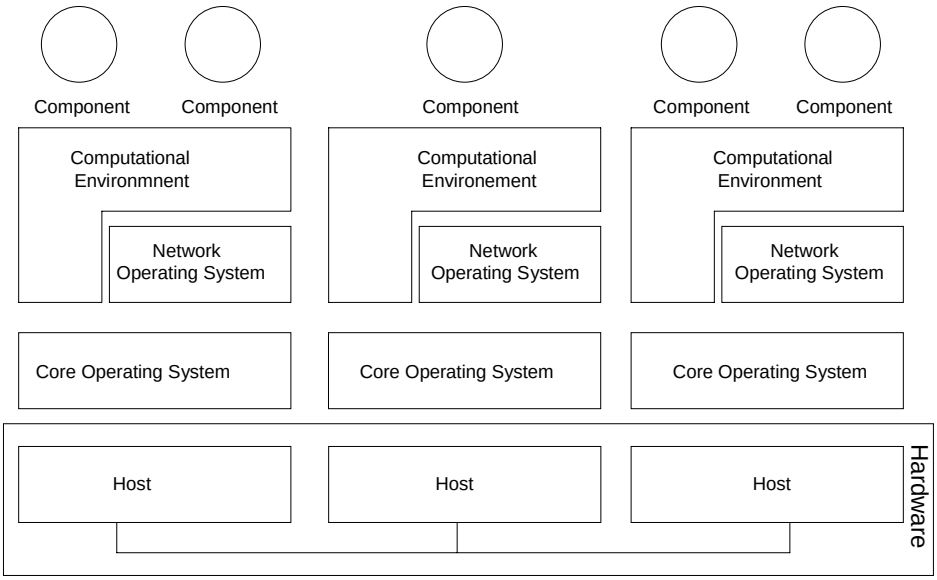
1.3.2 Mobilità di codice

In questa sezione vengono considerate più in dettaglio la mobilità di codice e i diversi paradigmi di tale mobilità. Un ottimo articolo sull'argomento è [FUA98], dal quale sono anche tratte alcune delle immagini seguenti.
L'architettura di riferimento che verrà considerata per la mobilità di codice è quella mostrata in figura 1.4. Al di sopra dell'hardware si hanno i diversi sistemi operativi, qui chiamati Core Operating System (COS), e il Network Operating System (NOS), che offre servizi per una comunicazione non trasparente.
Le applicazioni sviluppate direttamente al di sopra dei NOS devono nominare direttamente l'host col quale vogliono comuni-

care (l'astrazione di socket appartiene ai NOS). Infine, al di sopra di questi due strati, risiede il Computational Environment che ospita l'esecuzione dei Component, che rappresentano il codice mobile. Questa figura mette in rilievo una prima caratteristica fondamentale degli ambienti di computazione che ospitano il codice mobile (Component). I Computational Environment (CE) possono ospitare i component, offrendo loro un ambiente di esecuzione uniforme, ma sono logicamente tra loro divisi. Permane cioè una nozione di località anche se il livello applicativo viene sollevato dalla gestione dei canali di comunicazione, gestita più a basso livello dai CE interagendo coi NOS.

Architettura di Riferimento per la mobilità di Codice

Fig. 1.4



Quando vengono utilizzati middleware che offrono trasparenza alla locazione, invece, tale informazione non è disponibile al livello applicativo, dato che tutti i componenti vengono percepiti come locali. Esempi di queste piattaforme sono i riferimenti ad oggetti remoti. In questi casi il programmatore si può disinteressare della fisica locazione dei componenti e dell'organizzazione della rete sottostante perdendo però in capacità espressiva.

1.3.2.1 Mobilità dello stato di esecuzione

I Component introdotti nella sezione precedente possono essere distinti in: risorse e execution unit (EU). Una EU rappresenta un singolo thread con un proprio stato di esecuzione e uno spazio dati e può condividere delle risorse con altre EU (la struttura interna di una EU è rappresentata in figura 1.5).
Relativamente al fatto che lo stato di esecuzione di una EU venga mosso o meno vengono distinti due tipi di mobilità: **mobilità forte** e **mobilità debole**.

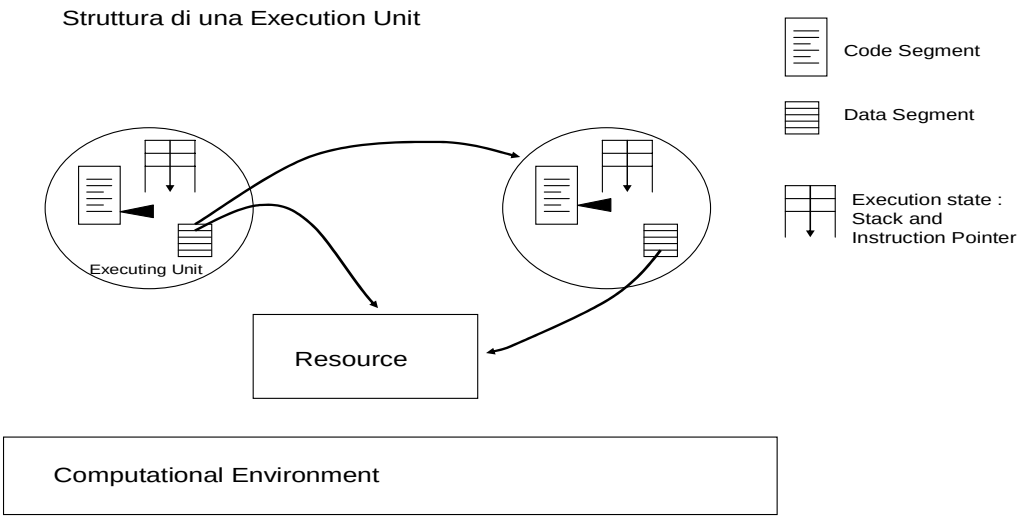


Fig. 1.5

La **mobilità forte** supporta la migrazione sia del codice della EU che dello stato di esecuzione. Affinché tale tipo di mobilità sia attuabile deve essere possibile salvare lo stato di esecuzione cioè, come mostrato in figura, lo stack e l'instruction pointer di una certa EU. All'arrivo nel nuovo CE lo stato di esecuzione verrà ristabilito e la EU potrà continuare la propria esecuzione da dove era stata sospesa. Naturalmente rimane il problema di come gestire i riferimenti alle risorse, aspetto che affronteremo nei prossimi punti.

La **mobilità debole** invece supporta la migrazione del codice ed eventualmente di uno stato. Non un'immagine dello stato di esecuzione, bensì uno stato costruito al livello applicativo, in modo che, quando la computazione riprenda all'arrivo nel nuovo CE, tale stato possa essere riutilizzato per inizializzare la EU e continuare la computazione.

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

1.3.2.2 Mobilità del codice

Poiché nell'ambito di questo progetto si utilizzerà esclusivamente il modello di mobilità debole, vediamo ora quelli che sono gli aspetti fondamentali da tenere in considerazione in questo caso; in particolare, a seconda di come avviene la mobilità del codice, vengono distinti vari casi, considerando le dimensioni sotto presentate :

1) Direzione nella quale avviene il trasferimento di codice: una EU può richiedere il trasferimento di codice (**fetch the code**) da remoto per poi eseguirlo localmente, oppure può spedire (**ship the code**) essa stessa del codice ad un'altra CE.
2) Natura del codice che deve essere mosso: il codice può essere **stand-alone code**, cioè codice che sia autocontenuto e, una volta arrivato nella nuova località, venga utilizzato per istanziare una nuova, indipendente EU, oppure può rappresentare un frammento di codice (**code fragment**) che verrà utilizzato da

una EU già in esecuzione per essere collegato (linked) ed eseguito insieme ad altro codice.

3) Sincronizzazione: dipendentemente dal comportamento della EU che richiede il trasferimento del codice, vengono distinti due casi: il caso sincrono quando l'EU si sospende in attesa che il codice trasferito sia stato completamente eseguito e il caso asincrono, se non c'è sospensione.

1.3.2.3 Gestione dello spazio dei dati

Ritornando a quanto detto sopra, rimane da affrontare un'ultima problematica, cioè come venga gestito lo spazio dei dati (data space, si veda la figura 1.5), durante la migrazione di una EU. Ogni risorsa può venire referenziata con tre modalità principali: con un'identificatore unico all'interno del sistema, o per valore o per tipo. A seconda di quale, o quali modalità vengano utilizzate, esistono diverse possibilità per la riallocazione delle risorse e la riconfigurazione dei binding alle risorse, dopo il salto di una EU. In [FUA98], cui si rimanda per ulteriori approfondimenti, viene riportato uno studio di questa problematica. Si vuole solo sottolineare che le applicazioni che verranno presentate, utilizzeranno principalmente il meccanismo **re-binding**; le risorse sono cioè referenziate per tipo e, dopo una migrazione, durante la fase di inizializzazione, si ricostruisce lo spazio dei dati, procedendo al re-bind delle varie risorse di cui una certa EU necessita nel nuovo CE.

1.3.2.4 Paradigmi di mobilità di codice

In relazione a come avviene l'esecuzione del servizio e alla posizione delle EU e delle risorse dopo la terminazione del servizio si distinguono diversi paradigmi di mobilità del codice.
In figura 6 viene riportata una tabella che riassume le diverse possibilità. Con A viene indicata una generica entità computazionale, collocata nel sito S_A, che è interessata alla ricezione del risultato di un certo servizio.

MOBILE CODE PARADIGMS

Paradigma	PRIMA		DOPO	
	S_A	S_B	S_A	S_B
Cliente / Servitore	A	Know - how Resource B	A	Know - how Resource B
Remote Evalutation	Know – how A	Resource B	A	Know - how Resource B
Code on Demand	Resource A	Know - how B	Resource Know – how A	B
Mobile Agent	Know – how A	Resource		Know – how Resource A

Fig. 1.6

Si assume poi l'esistenza di un sito S_B che sarà coinvolto nella esecuzione di tale servizio. A e B sono due entità computazionali e in grassetto viene indicata l'entità che esegue il codice. Nel classico modello client/server, ovviamente, non c'è mobilità di codice, e sia la conoscenza per effettuare la computazione, sia le risorse necessarie, sono presenti sul sito S_B.

1.3.2.4.1 Remote Evalutation (REV)

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

verrà in seguito ampiamente utilizzato. Esso infatti consente al Sito A di richiedere lo svolgimento di un determinato compito, quale per esempio l'esecuzione di un protocollo di Riconfigurazione, per cui passa anche le istruzioni, al Sito B. Ovviamente, come già, ampiamente chiarito, l'adozione di questo genere di meccanismi, può portare a problemi di Sicurezza e, pertanto, essi non dovranno essere direttamente accessibili alle applicazioni utente ma nascosti all'interno del middleware.

1.3.2.4.2 Code On Demand (COD)

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

1.3.2.4.3 Mobile Agent (MA)

L'ultimo paradigma considerato è quello degli agenti mobili. In questo caso l'entità A possiede la conoscenza necessaria per portare a termine il servizio, ma necessita di alcune risorse che sono presenti solo sul sito S_B, migra perciò su tale sito e termina l'esecuzione del servizio. Esiste una differenza fondamentale fra questo paradigma e quelli presentati in precedenza, e cioè che, mentre nei casi precedenti ciò che veniva spostato era principalmente codice, qui si muove l'intera EU, col proprio stato, codice, e risorse. Nell'attuale progetto gli agenti mobili vengono utilizzati anche per la configurazione e riconfigurazione del sistema distribuito. Un importante aspetto legato alla realizzazione di questo paradigma è la decisione su come realizzare la comunicazione fra gli agenti. Bisogna innanzitutto supportare l'identificazione degli agenti stessi, in modo che sia possibile per un certo agente esprimere la volontà di aprire la comunicazione con un altro agente. La comunicazione può poi essere realizzata in molti modi.

Possono essere utilizzate operazioni per la spedizioni di semplici messaggi (ad esempio oggetti serializzabili che vengono passati fra gli agenti), che si ispirino al message-passing, ma possono anche essere implementate comunicazioni di tipo stream–based.
Particolare rilevanza riveste poi l'utilizzo del modello a memoria condivisa, per la realizzazione di comunicazioni di gruppo fra più agenti, in quanto astrazione di una risorsa accessibile a tutti.

1.3.2.5 Sicurezza

Da ultimo si vogliono accennare alcun problematiche legate alla sicurezza; se in una rete locale possiamo ipotizzare di avere un elevato grado di fiducia, e quindi la mobilità di codice non implica particolari problemi, quando invece si considerino ambiti più ampi la sicurezza rappresenta un problema.
La paura degli amministratori di sistema è infatti quella di ricevere e far eseguire sulla propria macchina un codice non autorizzato che possa andare a minare le basi del sistema stesso, o anche, più banalmente, lo renda inutilizzabile, appropriandosi di tutte le risorse. Tale problema è stato uno dei fattori che hanno limitato la diffusione dei paradigmi sopra esposti, ed in particolare del paradigma ad agenti mobili. Elenchiamo qui alcuni dei requisiti di sicurezza richiesti ad un sistema che voglia supportare la mobilità di codice (si veda inoltre [TRA98]).

1) **Riservatezza ed integrità**: è spesso necessario mantenere riservate alcune parti dello stato di un agente mentre attraversa la rete, così come bisogna implementare metodi per verificare l'integrità di un agente e più in generale di codice mobile, al suo arrivo in un nuovo sito.
2) **Autenticazione**: ogni pezzo di codice mobile deve essere autenticato, in modo che i CE che ospitano tale codice possano determinare l'identità dell'utente che ne richiede l'esecuzione, questo anche per permettere l'addebito delle risorse utilizzate.
3) **Autorizzazioni e controllo degli accessi**: si vuole dare la possibilità ai proprietari delle risorse di specificare le politiche di accesso alle risorse stesse, basando tali politiche su diverse di-

mensioni, non solo sull'identità dell'utente che ha progettato il codice mobile, ma anche sul suo ruolo, ecc.,inoltre si vuole dare la possibilità agli sviluppatori degli agenti di specificare delle restrizioni ai diritti del proprio agente. Questa funzionalità sarebbe ovviamente molto utile in fase di debugging dell'agente stesso.

1.4 Conclusione.

In questo primo capitolo, abbiamo introdotto alcune nozioni basilari, definendo il concetto di flusso e alcuni parametri ad esso connessi. Abbiamo poi presentato le principali problematiche che si presentano oggi a ci voglia sviluppare applicazioni multimediali. Ci siamo infine soffermati sui modelli di mobilità e di comunicazione in ambito distribuito. L'intento di questo capitolo era, infatti, formare un background che consentisse di capire meglio le tematiche affrontate nello sviluppo di questo progetto. Nel prossimo capitolo ci addentreremo nell'ambito delle località; vedremo innanzi tutto alcuni tra le topologie di rete più diffuse e come su esse si possano realizzare dei sistemi di località, affrontando nel dettaglio un analisi degli aspetti coinvolti.

CAPITOLO 2.

2 Definizione di un modello per un sistema di località e QoS based routing.

La costituzione di un modello di sistema di località da adottare nell'ambito di topologie preesistenti, è di fatto il punto chiave per la realizzazione di servizi di raccolta delle informazioni e quindi per sia per servizi Location Aware e Context Aware che per servizi di QoS management come, l'adattamento dei flussi e la riallocazione distribuita delle risorse.
In questo capitolo, introdurremo il concetto di overlay network e i modelli di topologie a cui, in genere, un'overlay network è riconducibile. Nella seconda parte del capitolo, invece, introdurremo gli aspetti fondamentali da tenere in considerazione nella progettazione di un modello per un sistema di località e, infine, nella terza parte, vedremo quali sono le basi per una gestione intelligente delle risorse del sistema: sia per quanto riguarda l'allocazione delle risorse, sia per quanto riguarda la definizione dei percorsi.
Concluderemo infine il capitolo presentando alcuni esempi.

2.1 Modelli per un Overlay Network.

Qualunque sia il modello di località che si vuole realizzare, esso dovrà poggiarsi sulla topologia effettiva della rete sottostante.
In particolare, se è possibile, identificare un principio organizzativo nella costituzione della topologia, sarà possibile sfruttare tale organizzazione per sviluppare un modello di località più organico e razionale. In genere, è possibile evidenziare la presenza di una topologia strutturata, solo all'interno di reti locali (LAN) poiché nell'ambito delle Wide Area Network e Metropolitan Area Network, si ha in genere a che fare con una rete di reti che dal punto di vista della struttura fisica non realizza una particolare principio organizzativo. E' però vero che, a livello logico, tale rete di reti viene strutturata e riorganizzata, creando reti e sottoreti. Tale

riorganizzazione genera quindi delle “overlay networks”, cioè delle reti costruite su altre. Naturalmente, una rete di questo tipo può essere costruita non solo su reti fisiche ma anche su altre overlay networks, in modo da creare una struttura a livelli, in cui, ogni livello crea una propria visibilità sul sistema di reti sottostante, a seconda delle proprie specifiche esigenze.
Facciamo notare inoltre, che se la ripartizione in sottoreti di una rete più grande, non comporta in genere problemi, non è lo stesso per la creazione di una overlay network che comprenda nodi di reti diverse. In questo caso, potrebbero essere utilizzati, protocolli diversi nelle diverse reti e la struttura logica che definisce l’overlay network(di solito un sistema di nomi) deve tenerne conto per effettuare la eventuali conversioni, ecc..
Le overlay network, possono quindi essere usate per ridurre la visione della rete a livello sottostante, oppure per ampliarla. Ancora possono essere create per fornire al sistema di reti sottostante una struttura logica particolare. Per quanto ci riguarda, il nostro presupposto, è che il sistema di località non venga creato direttamente su una rete fisica ma appunto su una overlay network che organizzi il sistema di reti sottostante secondo una precisa topologia.
Nella prossima parte del paragrafo ci soffermeremo quindi nell’introduzione delle principali organizzazioni logiche e topologie per una rete.

2.1.1 Struttura ad anello.

Un tipo di struttura molto semplice è la struttura ad anello. L’anello rappresenta il mezzo di comunicazione sul quale sono affacciate tutte le macchine della rete. Nel caso che sia gestito con protocolli della famiglia TokenRing, esisterà un messaggio, chiamato token, che verrà passato tra le macchine. Queste comunicano a turno (quando hanno a disposizione il token) e i messaggi hanno un senso di percorrenza dell’anello stabilito. Il tempo di utilizzo della rete per la singola macchina è stabilito all’atto della costituzione del

progetto della rete (dati sincroni) ma sono possibili anche protocolli che prevedano l'eventuale invio di dati asincroni. Esistono numerose varianti a questo modello e vale la pena di notare come non è affatto necessario che le reti sottostanti siano effettivamente ad anello, in quanto, una volta stabilito un ordine tra le macchine della rete, il token può essere passato dall'una all'altra secondo l'ordine stabilito, qualunque sia la topologia sottostante.

2.1.2 Struttura a Stella.

Il caso di una topologia a stella è il tipico caso di molti terminali collegati ad una unità centrale (è per esempio il caso di una rete creata per un servizio di chat, in cui un server centrale faccia da tramite per le comunicazioni tra gli utenti). In questa struttura, un nodo centrale funziona da termine di collegamento per ogni nodo periferico; la comunicazione tra i nodi periferici richiede necessariamente il transito attraverso il centro della stella. Anche in questo caso, i problemi all'aumentare dei nodi nella rete sono dovuti alla rapida diminuzione delle risorse di comunicazione. Oltre a ciò, assicurare la stabilità del nodo centrale è il problema principale per questo modello di rete in quanto da esso dipende la stabilità della rete.

2.1.3 Struttura a Bus unico condiviso.

Concettualmente, il più semplice tipo di topologia possibile. Tutti i componenti della rete si affacciano sullo stesso mezzo di comunicazione e competono tra di loro per l'accesso alla risorsa di comunicazione (fig. 2.1). Ogni nodo vede direttamente tutte gli altri nodi presenti nella rete. La comunicazione può essere gestita secondo diversi protocolli e la differenza sostanziale tra questa organizzazione e quella ad anello (in entrambe un solo nodo alla volta utilizza il mezzo di comunicazione) è che in questo caso non esiste, in genere, un ordine predefinito per l'accesso alle risorse di comunicazione. Problema di questo tipo di organizzazione è che, anche in questo caso, al crescere del numero di macchine affacciate sul bus abbiamo però uno scadere delle prestazioni.

Schema di Topologia a Bus unico condiviso .

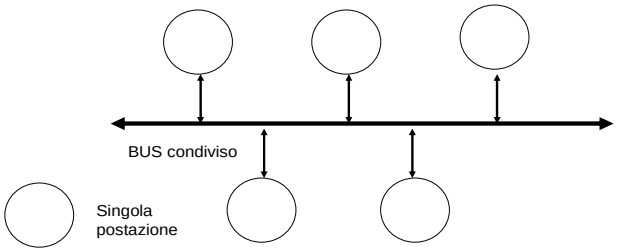


Fig. 2.1

2.1.4 Struttura a Bus Multipli.

Rappresenta una variante della struttura a Bus unico condiviso.
In questo caso si cerca di ridurre la diminuzione di prestazioni all'aumentare delle macchine utilizzando un numero di bus superiore a uno. Il numero di macchine direttamente affacciate su un bus viene mantenuto entro certi valori dipendenti dalle specifiche definite in fase di progetto. In una rete fisica, i bus vengono messi in comunicazione tra di loro in genere tramite un bus trasversale (fig. 2.2) mentre nel caso di una overlay network, dovrà essere presente, per ogni bus, un nodo che effettui un proxy

Schema di Topologia a Bus multipli

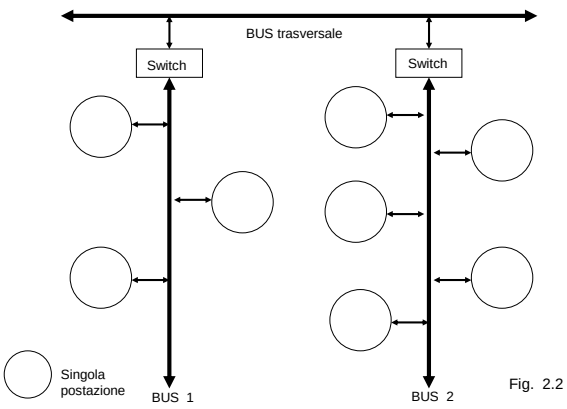


Fig. 2.2

da e verso gli altri bus, in modo da ricevere e propagare sul proprio

bus i messaggi provenienti dagli altri e da effettuare un forward per i messaggi diretti agli altri bus. Al crescere delle postazioni anche questa struttura tende comunque a degenerare le proprie prestazioni e diventa inattuabile per l'aumentare delle comunicazioni attraverso i nodi proxy.

2.1.5 Struttura ad albero

L'organizzazione di una topologia di rete, secondo un a struttura ad albero, è molto comune. Questa struttura consente infatti di ottenere diversi vantaggi rispetto alle altre possibilità tra cui limitare il numero di connessioni e consentire una crescita dei componenti della rete senza conseguire un degrado eccessivo delle prestazioni. La struttura di un albero si compone gerarchicamente per livelli; al primo abbiamo un nodo radice che costituisce il vertice dell'albero. Il nodo radice è connesso direttamente con tutti i nodi del secondo livello. Tali nodi sono chiamati figli, mentre il nodo radice è il padre. Ogni nodo del secondo livello può, a sua volta, essere padre di altri nodi che si collocano quindi al terzo livello e così via (fig. 2.3). Le connessioni richieste per nodo, in questa struttura sono quindi dell'ordine del numero dei figli più quella verso il genitore, mentre i ritardi di comunicazione sono proporzionali al numero dei livelli, consentendo però un fattore di crescita esponenziale relativo al numero di livelli dell'albero.
L'unico svantaggio che tale struttura presenta è la presenza, dati due elementi dell'albero, di un solo cammino tra questi. Oltre a ciò, tale cammino è in genere condiviso, in tutto o in parte per i collegamenti tra gli altri componenti dell'albero. Questa struttura si presta in sostanza bene all'individuazione di un elemento in una rete e all'organizzazione della rete stessa ma non troppo bene per quanto riguarda la gestione dei percorsi tra le singole macchine; per questo motivo, in genere questa struttura viene utilizzata non per l'organizzazione fisica delle reti, ma per la loro gestione (sistemi di nomi,ecc..).

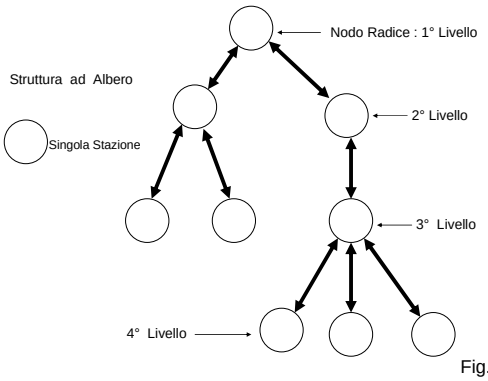


Fig. 2.3

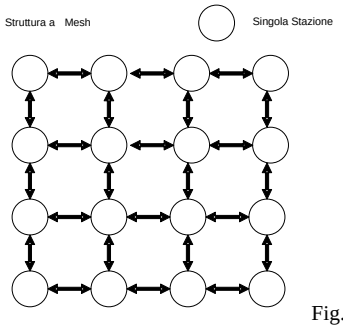


Fig. 2.4

2.1.6 Struttura a mesh e ipercubi

La struttura a mesh, prevede la definizione della struttura di una cella di nodi in comunicazione tra loro e la ripetizione omogenea nelle due dimensioni di tale cella. In genere si considera come cella fondamentale il quadrato, i cui vertici rappresentano i nodi e i cui lati rappresentano le connessioni possibili.
La ripetizione nelle due dimensioni della cella fondamentale porta alla strutturazione di una mesh, anch'essa quadrata in cui ogni postazione è direttamente connessa con le 4 vicine in modo tale da trovarsi al centro di un quadrato composto di lato doppio di quello della cella fondamentale.
La struttura a ipercubi, riprende poi esattamente la stessa idea, utilizzando come cella base, un cubo, e attuando la ripetizione in uno spazio di connessioni n-dimensionale. La creazione di mesh e ipercubi, da un lato ha il vantaggio di creare percorsi multipli tra i nodi, dall'altro ha lo svantaggio dell'alto costo per la gestione delle connessioni.

2.1.7 Struttura a grafo.

La struttura a grafo è una generalizzazione delle strutture precedenti. Non esiste come nella struttura a mesh una cella fondamentale per le connessioni che viene ripetuta spazialmente.
Né esiste una vera e propria strutturazione gerarchica come nel modello ad albero. Ogni macchina della rete può teoricamente es-

sere connessa con ogni altra stazione presente (vedi fig. 2.6); se è così si parla di connettività totale (si hanno n*(n-1) connessioni dove n è il numero di nodi della rete). Naturalmente, nel caso di una rete fisica, i costi di installazione di una rete a connettività totale rendono in genere tale soluzione inaccessibile; nel caso invece di una overlay network rimangono comunque alti i costi di gestione. Spesso le topologie costruite secondo il modello della struttura a grafo, sono riconducibili a strutture con un albero primario e con connessioni incrociate tra nodi non in rapporto di parentela diretto (vedi fig. 2.5). Questi però sono da considerarsi casi particolari. Nell'ambito della costituzione di reti di reti, la struttura a grafo rappresenta comunque il caso più comune in quanto emerge spontaneamente dalla libera connessioni di queste.
Vantaggio di questo tipo di topologia è ovviamente la possibilità di individuare cammini multipli tra due macchine della rete senza avere, nei casi pratici, un aumento delle connessioni (e quindi dei costi) elevato come nel caso delle mesh.

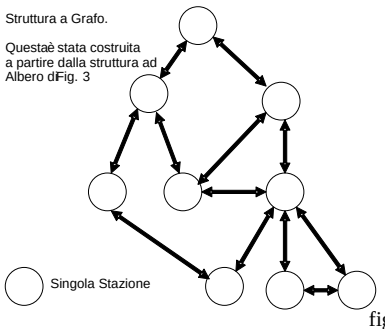


fig 2.5

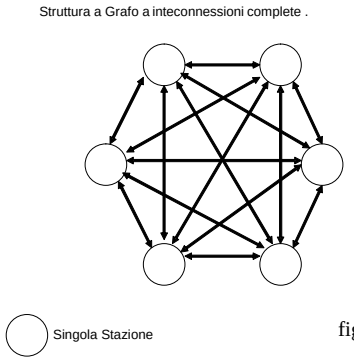


fig. 2.6

L'insieme delle topologie che abbia presentato, non costituisce un catalogo completo delle possibili organizzazioni delle reti, ma rappresenta sicuramente quelle che sono le strutture maggiormente adottate.

2.1.8 Topologie dinamiche e statiche.

Oltre alla configurazione della topologia, aspetto fondamentale di una rete è la sua capacità di reazione rispetto agli eventi che si verificano nel corso della sua esistenza e che possono modificare la topologia stessa. Se una rete prevede la possibilità di gestire tali eventi, si dice dinamica; se invece la configurazione della rete non prevede o gestisce eventuali modifiche della topologia, allora si dice statica. In realtà, attualmente, la maggior parte delle reti, prevede e gestisce in maniera efficiente sia il caso di nuovi ingressi che il caso di postazioni cadute o uscite, a patto però che tali eventi non modifichino la topologia nel suo insieme. Se prendiamo, per esempio, il caso della topologia ad albero, mentre è tollerata senza particolari disagi la caduta di elementi terminali (le foglie), la caduta di un nodo interno dell'albero divide normalmente la rete in due sottoreti non comunicanti, con gli ovvi inconvenienti che ne derivano; allo stesso modo, l'aggiunta di nuovi nodi, è possibile solo se questi si inseriscono come foglie, ossia figli di nodi pre-esistenti.
Naturalmente anche la reattività a questi eventi dipende dalla struttura della rete; se infatti, in una rete ad albero, vi saranno dei vincoli sull'inserimento di nuovi nodi, in una rete a grafo, tali vincoli non saranno presenti e sarà possibile introdurre nuovi elementi in qualunque punto della topologia.

Ora che abbiamo presentato i principali modelli di organizzazione per una rete, passiamo alla definizione degli aspetti fondamentali da tenere in considerazione nella definizione di un modello di località.

2.2 Modello per un Sistema di Località.

I motivi per l'adozione di un sistema di località in una struttura di rete pre-esistene, sono in genere due:
1) la riduzione della visibilità tra le postazioni della rete

2) l’estensione della visibilità tra le postazioni della rete.

Anche se tali motivazioni sembrano contrastanti, in realtà, non è così. In entrambi i casi, infatti, l’obiettivo che ci si prefigge, è quello di ottenere un livello di visibilità equilibrato alle esigenze del sistema. Accade così che per i sistemi in cui ogni macchina ha visibilità diretta di un numero eccessivo di nodi, un sistema di località riduca tale visibilità in modo tale da consentire una gestione migliore dei link rimasti visibili; per i sistemi, invece, in cui le macchine hanno una visibilità troppo limitata delle postazioni vicine, l’adozione di un sistema di località, consenta il superamento di tale limite. Un servizio di località è infatti,in primo un luogo, un servizio di naming.
Naturalmente, ricordiamo ancora una volta, nell’ambito del nostro progetto, sia quando si parla di visibilità che di topologia per una rete, non intendiamo riferirci né alla struttura fisica della rete, né alla sua configurazione a livello di rete e trasporto, ma, in genere ad una sua organizzazione logica sovrastante i livelli menzionati.
In questo paragrafo intendiamo chiarire cosa è, per noi, una località e come un sistema di località può essere realizzato su una rete con una propria topologia.

2.2.1 Concetto di Vicinanza

Inteso in senso lato, il concetto di località è definibile come un insieme di siti che si trovano tutti nelle “vicinanze”. Naturalmente, più definizioni possono essere date per il concetto di “vicinanze”; le più evidenti sono :

1) Vicinanza geografica : due Place (o DefaultPlace) appartengono alla stessa località se sono geograficamente vicini (rimane comunque da stabilire cos’è la vicinanza geografica)
2) Vicinanza temporale : due postazioni appartengono alla stessa località se le comunicazioni tra le due avvengono in un intervallo di tempo stabilito a priori; in questo caso, la località dipenderebbe dinamicamente anche dalle condizioni del traffico sulla rete tra i diversi nodi, oltre che ovviamente dalle stesse

risorse di rete tra i nodi.

3) Vicinanza logica : due nodi di una rete appartengono alla stessa località se è possibile definire una caratteristica funzionale comune per gli elementi della topologia, ed è la stessa per i nodi di una medesima località (ad esempio : tutte le postazioni in cui sono attivi componenti per il calcolo numerico; tutti i nodi in cui sono disponibili certi tipi di risorse, ecc..)
4) Vicinanza in termini di routing : due nodi appartengono alla stessa località se tra i due vi sono a livello di rete, un numero non superiore a una soglia fissata, di intermediari che effettuano routing o forwarding.
5) Vicinanza in termini di elementi di topologia. Due elementi della rete appartengono alla stessa località, se tra i due vi è un numero di elementi della topologia non superiore a una soglia prefissata. Questa definizione sembra ricalcare quella del punto precedente tuttavia non è così; ricordiamo infatti che quando ci riferiamo alla topologia di una rete, non intendiamo indicare necessariamente l'effettiva configurazione delle connessioni fisiche ma che essa può fare riferimento a un'organizzazione logica della rete costruita su una diversa topologia fisica.

Ovviamente altri tipi di "vicinanza" sono ipotizzabili. La nostra scelta comunque è ricaduta sull'ultimo tipo di "vicinanza" considerato.

Il motivo della nostra scelta dipende da considerazioni sulle diverse possibilità; nel primo caso sarebbe stato difficile avere un riferimento geografico delle macchine senza contare che a livello operativo la vicinanza geografica ha un'importanza relativa; nel secondo caso, ogni località sarebbe comunque stata costretta a controllare un numero di nodi prefissato (e stabilito presumibilmente secondo uno degli altri modelli di vicinanza) per garantire che un nodo uscito dalla località vi possa rientrare nel caso che le condizioni di rete tra i due migliorino. Oltre a questo, la dimensione della località sarebbe molto variabile, mentre, per motivi pratici, ha senso località abbastanza stabili. Infine, in caso di sovraccarico della rete, un sito potrebbe trovarsi isolato (cioè in una località con

lui come unico elemento) vanificando quindi rendendo inutile il sistema di Località introdotto.
Il terzo caso sarebbe operativamente interessante. Le difficoltà sarebbero però nell'implementazione; in particolare nell'identificazione di una caratteristica logica che permetta una distinzione in categorie di tutti i siti del sistema; in sostanza, dovrebbe essere introdotto un metadato che descriva le caratteristiche del sito in termini delle attività svolte e che venga aggiornato al modificarsi di tali attività; oltre a ciò, questo tipo di vicinanza sarebbe utilizzabile più da applicazioni di sistema che da applicazioni utenti a cui interesserebbe relativamente sapere in quali altri siti sono ospitate altre applicazioni utenti mentre per quanto ci riguarda è importante che possano trarre vantaggio dall'introduzione delle Località anche e principalmente tali applicazioni che sono di fatto le maggiori consumatrici di oggetti multimediali.
Il quarto caso presenta sicuramente un'ipotesi di "vicinanza" valido, tuttavia le informazioni necessarie per costruire tale tipo di località (cioè le tabelle di routing) sono a un livello non sempre accessibile direttamente; si è pertanto optato per un'ipotesi di vicinanza simile, cioè l'ultimo presentato, realizzabile con le informazioni che sono sempre accessibili direttamente accessibili anche nel caso in cui non si conosca l'effettiva organizzazione della rete sottostante. Definito il concetto di vicinanza, è necessario stabilire esattamente cosa è una località e quali sono le sue caratteristiche e quali servizi dovrà offrire.

2.2.2 I parametri di una Località.

Dare una definizione del concetto di località che intendiamo introdurre significa descriverne le proprietà e le relazioni con la topologia sottostante. In primo luogo, stabilito che il concetto di "vicinanza" è relativo alla topologia di cui si ha visibilità, prenderemo ora in considerazione le località dal punto di vista della sovrapposizione e centratura, della grandezza e dei servizi che dovranno fornire.

Naturalmente, sarà il tipo di topologia sottostante a dettare la scelta sulla tipologia di località; noi intendiamo qui presentare i criteri e gli aspetti fondamentali da tenere in considerazione in una fase di progetto.

2.2.2.1 Sovrapposizione e centratura di una località.

I primi due parametri che prendiamo in considerazione sono centratura e sovrapposizione. Con centratura, intendiamo stabilire se le località saranno centrate su particolari nodi, e se sì su quale tipo di nodi, o meno. Con sovrapposizione, invece ci interessa stabilire se, diverse località possono sovrapporsi, ossia se è possibile che un nodo appartenga a diverse località contemporaneamente. Concettualmente i due parametri sono ortogonali e dipendono fortemente dalla topologia della rete sottostante. Vediamo quindi i quattro casi possibili :

1) località non centrate e non sovrapposte; consideriamo una rete organizzata a bus multipli, ci accorgiamo immediatamente che il modello di località più intuitivo è quello che associa ad ogni bus una località. Tali località non sono centrate in quanto non esiste un particolare nodo da cui dipendono e non sono sovrapposte in quanto non esistono nodi affacciati su più di un bus.

2) località non centrate e sovrapposte; consideriamo ora una rete organizzata a mesh e consideriamo di costruire una località radunando i nodi di una cella fondamentale. Tale tipo di località non è centrata su un nodo particolare e può essere sovrapposta o meno. Nell'esempio in figura abbiamo il caso relativo a località sovrapposte.

3) località centrate e non sovrapposte; per questo esempio possiamo semplicemente riprendere il caso precedente e considerare invece delle singole celle della mesh una nuova cella di lato pari a una volta e mezzo il lato della cella originaria e centrata sul nodo centrale di tale cella. Altri esempi sono possibili in reti ad anello, ad albero, ecc..

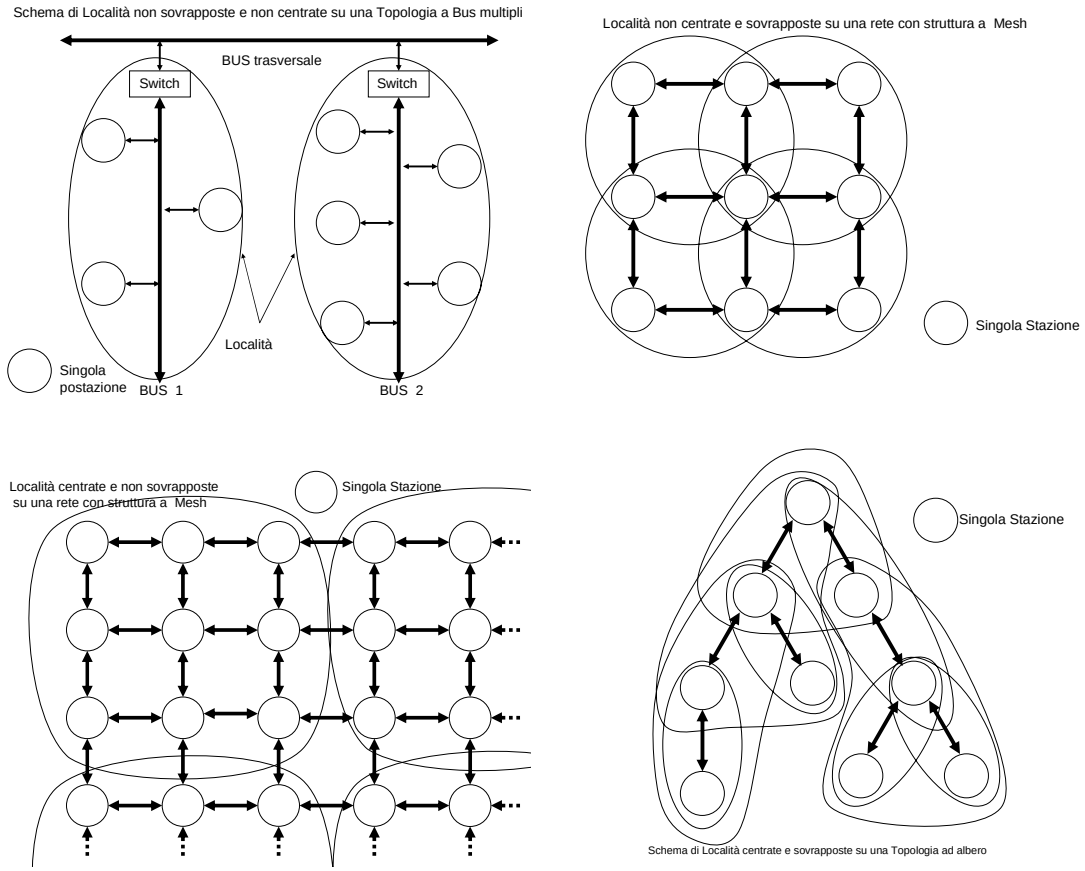


Fig 2.7

4) località centrate e sovrapposte; prendiamo per questo esempio una rete con topologia ad albero. Se in questo consideriamo una località centrata su ogni nodo dell'albero e con un estensione di almeno un passo, abbiamo un sistema di località centrato e sovrapposto, che per ogni nodo, contiene i rispettivi figli e padre.
In figura 2.7 sono riportati i quattro casi esposti.

E' chiaro come la scelta di questi parametri definisce quindi sia i rapporti tra il sistema di località e la topologia sottostante che i rapporti tra le varie località; va inoltre notato che non tutti gli esempi sono riconducibili al concetto di vicinanza che abbiamo

scelto; complessivamente risulta quindi, difficile delineare parametri generali di giudizio in questo ambito. La decisione riguardo alla sovrapposizione e centratura delle località, deve rispondere quindi ad un attento esame delle proprie esigenze e della rete sottostante. Nell'ambito del nostro progetto, il sistema di località che realizzeremo è di tipo centrato e con sovrapposizioni. Le motivazioni che ci hanno portato saranno chiarite nel prossimo capitolo quando entreremo nel dettaglio di questa parte di progetto.

2.2.2.2 Dimensione di una località.

La definizione della dimensione di una località è un parametro che dipende anch'esso dal tipo di vicinanza che si considera.
Poiché, nel nostro caso, consideriamo, come parametro per la distanza, il numero di passi lungo la topologia, possiamo introdurre il concetto di raggio o range di un località. In particolare, abbiamo che appartengono ad una stessa località, centrata su un nodo della topologia, tutti gli altri elementi della rete che sono raggiungibili in un numero di passi pari o inferiore al raggio scelto per la località. Rimane quindi, ora, da definire il valore di tale raggio.
Il valore scelto, anche in questo caso, sarà dipendente più che da una scelta progettuale, dalle caratteristiche della rete sottostante. Una volta stabilito poi un valore per il raggio di località è necessario considerare se tale valore sarà adottato da tutte le località del sistema o se, località diverse, potranno avere raggi diversi. La presenza di località dal raggio diverso, rende sicuramente più flessibile il sistema e meglio adattabile alle diverse situazioni possibili; d'altro canto, questa possibilità introduce il problema della visibilità asimmetrica tra le diverse località che rende più complesso lo sviluppo dei protocolli di gestione. Nell'ambito del nostro progetto si è comunque deciso di consentire questa possibilità.

2.3 QoS-based routing.

Una volta definito il modello per quello che sarà il sistema di località che andremo a creare, passiamo ora al problema dell'utilizzo delle informazioni della località per la gestione di flussi di dati: cioè al problema del QoS-based routing.

Il QoS-based routing è definito come : "Un meccanismo di routing per il quale i percorsi per i flussi sono determinati in funzione di una qualche conoscenza sulla disponibilità delle risorse nella rete, così come per le specifiche di QoS dei flussi" [RFC2386]. Oppure come: "protocollo di routing dinamico che ha espanso i propri criteri di selezione del percorso per includere parametri di QoS come banda disponibile, percentuale di utilizzo di link e percorsi end-to-end, consumo di risorse per nodo, ritardo e latenza e jitter."[QOSF2].

In questo paragrafo intendiamo offrire una panoramica del problema del QoS-based routing; in particolare, nella prima parte intendiamo presentare il problema da un punto di vista generale, mentre nella seconda parte, descriveremo alcuni casi reali di sistemi in cui si è affrontato il problema del QoS routing.

2.3.1 Metriche, euristiche e definizione dei percorsi.

I due temi di base del QoS-based routing sono: per primo come misurare e raccogliere le informazioni sullo stato della rete(problema nel nostro caso risolto dal sistema di località); secondo, come definire percorsi basati sulle informazioni raccolte.

La selezione di una metrica è quindi molto importante in quanto deve rappresentare le proprietà di interesse della rete, inoltre la metrica definisce il tipo di garanzie di QoS che il sistema può fornire. Non esiste, infatti, un modo per supportare richieste di QoS che non possano essere mappate in una qualche combinazione delle metriche esistenti.

Oltre a ciò, la complessità di elaborazione deve essere presa in considerazione nella scelta di una metrica; sfortunatamente, il QoS-based routing è, in genere, sottoposto a vincoli multipli, e il problema della definizione di un percorso in base alla combinazio-

ne di diverse metriche si è dimostrato un problema NP-completo [WC96]. Molti algoritmi sono stati proposti; un metodo abbastanza comune è chiamato : “sequential filtering”, nel quale una combinazione di metriche è ordinata secondo un certo ordine, in relazione all’importanza delle stesse; prima i percorsi vengono calcolati sulle metrica principale, successivamente un sottoinsieme dei percorsi trovati, viene eliminato in funzione di una metrica secondaria e così via fino a che non rimane un solo percorso. In questo modo si ottiene un tradeoff tra l’ottimizzazione delle performance e la complessità di elaborazione. In generale, comunque, le metriche possono essere riunite in tre categorie [CHEN99] definite qui di seguito:

Sia m(n1, n2) una metrica per il link(n1, n2). Per ogni percorso
P = (n1, n2, …, ni, nj) (dove n1, n2, …, ni, nj sono nodi della rete)

1) metriche addittive:
   se m(P) = m(n1, n2)+ m(n2, n3)+ …+ m(ni, nj).
   Esempi di questo tipo di metriche sono quelle relative ai ritardi le cosiddette hop-count, in cui si contano la distanza tra due nodi, in termini del numero di elementi tra i due nodi nella topologia della rete.
2) metriche concave:
   se m(P) = min[m(n1, n2), m(n2, n3), …, m(ni, nj)]
   Esempi sono le metriche basate sulle richieste di banda o sulle capacità dei proxy.
3) metriche moltiplicative:
   se m(P) = m(n1, n2)* m(n2, n3)* …* m(ni, nj).
   Un esempio è la metrica relativa alla volatilità dei proxy: cioè la probabilità che un proxy cada. In questo caso, si cerca il percorso più stabile. La probabilità che un nodo qualunque in un percorso cada (e che quindi il percorso non sia stabile) è data da :
   $1 - \prod_{i=1,..,n} (1 - v(p_i))$
   dove $v(p_i)$ è la probabilità di caduta del proxy i.

Il calcolo dei percorsi è, strettamente legato al problema dell'occupazione delle risorse, infatti, una volta che un possibile percorso è stato scelto, le risorse(banda, memoria, ecc..) devono essere riservate per il flusso, così che non sono disponibili per altri flussi. Il problema principale è dunque stabilire su quale nodo cercare di occupare le risorse.
Per la soluzione di questo problema esistono diversi approcci: il primo consiste nel non porsi alla ricerca di un particolare nodo che sia in grado di soddisfare le richieste in termini di risorse; quando si esaminano i diversi nodi in cui riservare le risorse, si sceglie semplicemente il primo nodo in cui le richieste possono essere soddisfatte. Ovviamente, in questo caso, non vi è nessun tipo di ottimizzazione né di bilanciamento del sistema, la scelta, avviene, non a caso, ma in maniera sistematica. In questo modo si caricherà prima completamente il primo nodo della lista, per poi passare al secondo e così via.. l'unico vantaggio di questo criterio di scelta è che, di fatto non viene eseguita nessun tipo di computazione. Naturalmente, in genere non si adotta una scelta così "grezza" ma si cerca di sfruttare le informazioni sullo stato delle risorse dei nodi, adottando criteri di scelta più complessi.
Una delle euristiche più usate è la cosiddetta wide-fit; con questa euristica, viene scelto il nodo con la maggiore disponibilità delle risorse (o con il migliore valore della metrica di valutazione). Questa scelta che è assolutamente logica, non sempre è la politica migliore.
Consideriamo per esempio il caso in cui si debba scegliere tra due nodi (vedi fig.2.10). Nell'esempio un primo flusso, dal nodo A al nodo B, richiede un buffer di 4Mb su un nodo intermedio.
Utilizzando il protocollo wide-fit, sceglieremo di far passare il flusso per il nodo D, che vedrebbe così ridotta la sua memoria libera a 4Mb. A questo punto un secondo flusso da A a B, richiede su un nodo intermedio un buffer di 5Mb, tale richiesta però non può essere soddisfatta da nessuno dei due intermediari, in quanto in D sono rimasti 4Mb e lo stesso vale per il nodo C. Se avessimo invece, assegnato le risorse per il primo flusso nel nodo C, nel

nodo D sarebbero rimaste risorse sufficienti anche per il secondo flusso. La differenza tra i due casi è che, nel primo si è avuta una maggior frammentazione delle risorse rispetto al secondo.

Allocazione delle risorse :
Devo allocare due componenti, il primo che occupa 4Mb in un istante t1
Il secondo occupa 5Mb in un istante t2 successivo a t1.
Il nodo D ha a disposizione 8Mb, mentre il nodo C ne ha ha disposizione 4.

1) wide-fit : viene selezionato il nodo C per la prima richiesta e la seconda richiesta viene rifiutata perché non ci sono nodi con risorse sufficienti.
2) best-fit : viene selezionato il nodo D per la prima richiesta e il nodo C per la seconda.

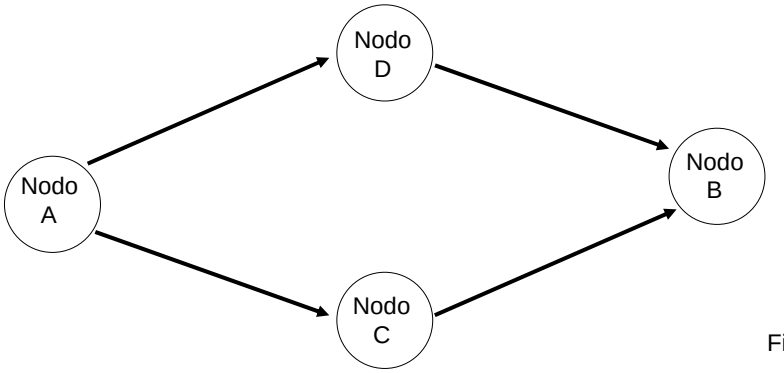


Fig. 2.10

Per evitare dunque la frammentazione delle risorse si userà un protocollo di scelta del tipo best-fit, in cui il nodo scelto sarà quello la cui disponibilità di risorse è più vicina alla quantità di risorse effettivamente richieste.
Un altro svantaggio della wide-fit è che, anche se calcolata su nodi diversi (per es. in figura 2.10 sui nodi A e B) indicherà, in entrambi i casi lo stesso nodo (cioè il nodo D) e vi sarebbe concorrenza per la prenotazione delle risorse e la probabile rifiuto di una delle due.
Per ridurre i rischi di ciò, si può considerare un euristica di tipo probabilistico. In questo tipo di euristiche, per ogni nodo,esiste una possibilità di essere scelto proporzionale a un coefficiente: un peso, calcolato in relazione allo stato delle sue risorse. Normalmente, nel

caso più comune, la probabilità di un nodo di essere scelto (ovviamente se è in grado di soddisfare la richiesta di risorse) è direttamente proporzionale alla quantità di risorse disponibili. (per le euristiche si veda [DQRMT],[IARSQD]).
Esistono poi altri tipi di euristiche e diverse strategie per l'allocazione delle risorse; nel caso di reti ATM wireless, per esempio, viene spesso usata una strategia di allocazione delle risorse di tipo predittivo, in cui le risorse, per un utente/applicazione, vengono allocate prima che ve ne sia la reale necessità, in relazione a quella che è la sua storia passata. L'idea di base (vedi [BASW]) è che i movimenti di un terminale wireless da una cella ad un'altra, siano in generale prevedibili se si è a conoscenza dei suoi spostamenti precedenti; il caso tipico, è quello di un utente in viaggio su un'autostrada in cui è possibile prevedere in quale cella entrerà l'utente, nell'ipotesi che si segua la direzione dell'autostrada.
In sostanza, la scelta dei protocolli e delle strategie per l'allocazione delle risorse e la valutazione dei diversi nodi, è un ambito di ricerca molto ampio, in cui una soluzione efficiente, può essere proposta, di fatto, solo riferendosi a situazioni reali.

Concludiamo questo paragrafo, introducendo gli algoritmi per il QoS-based routing; non intendiamo qui presentare dei veri e propri algoritmi per il QoS-based routing ma delineare quelle che devono esserne le caratteristiche principali.
La maggior parte degli algoritmi proposti, partono, in realtà dall'estensione dei normali protocolli di routing; i loro requisiti di base sono inoltre:
1) essere efficienti e scalabili,
2) essere di una complessità non troppo superiore rispetto a quella degli attuali algoritmi,
3) essere adatti all'architettura della rete.

Si può notare, come alcuni di questi requisiti siano in realtà in conflitto tra di loro, per cui è evidente che sarà necessario cercare un compromesso; in particolare per quanto riguarda efficienza e complessità.

In genere comunque, è possibile suddividere gli algoritmi per il QoS-based routing in tre categorie:
1) hop by hop routing, chiamato anche routing distribuito,
2) source based routing,
3) hierarchical routing.
In sostanza, questi vengono classificati, in relazione al modo in cui le informazioni di stato sono mantenute e a come la ricerca dei possibili percorsi è portata avanti.
Nel source routing, ogni router ha le informazioni sullo stato globale della rete e il percorso è localmente determinato su queste informazioni. Una volta che il percorso è determinato, il source router indica agli altri router lungo il percorso, come effettuare il forward del flusso. Il flusso sarà quindi consegnato nel nodo di destinazione secondo gli accordi stabiliti. Naturalmente è possibile che sia necessaria una fase di negoziazione con gli altri router e deve essere presa in considerazione la possibilità che un router non accetti le richieste effettuate dal source router che quindi dovrà calcolare dei percorsi alternativi. Nell'ambito di un sistema di località, questo tipo di algoritmi non è sempre possibile, in quanto non si ha una visione dell'intera rete; è però sicuramente attuabile, nell'ambito della singola località.
Il routing gerarchico è quello più adatto per reti di grandi dimensioni. il processo di routing è suddiviso in livelli multipli; il livello più basso, mantiene le informazioni sui nodi effettivi attraversati.
In questo caso i nodi sono riuniti in gruppi logici, che possono a loro volta essere riuniti e cosi via, creando tutti i livelli; ogni gruppo è un nodo logico del livello sovrastante. Le informazioni per il routing, sono in questo caso, integrate nei nodi di confine di ogni gruppo. In questo caso, anche se il sistema di località crea un livello sovrapposto alla rete sottostante, è possibile attuare un protocollo di questo tipo solo nel caso in cui le località non siano sovrapposte. Un protocollo di questo tipo è il Private Network-Network Interface (PNNI) che vedremo brevemente nel prossimo paragrafo.

Nel routing di tipo hop by hop, infine, i nodi non sono a conoscenza dell'intero percorso ma conoscono esclusivamente il passo successivo verso la destinazione. Il nodo non fa altro che effettuare un forward del flusso verso il nodo successivo fino a che il flusso non giunge a destinazione. Il vantaggio di questo tipo di protocolli è che riduce notevolmente i problemi di complessità per il calcolo del percorso migliore, distribuendola lungo tutto il percorso. Evoluzioni di questo tipo di protocollo prevedono inoltre la possibilità, arrivati su un nodo, di portare con sé, il codice per la determinazione del passo successivo.

Con questo, abbiamo terminato la parte introduttiva relativa al QoS-based routing; nel prossimo paragrafo vedremo alcuni esempi di sistemi reali.

2.3.2. Approccio al problema della QoS nella pratica effettiva.

In questa sezione, non intendiamo presentare semplicemente dei protocolli per il QoS-based routing; il nostro intento è quello di mostrare come il problema del QoS sia in realtà affrontato a diversi livelli, e non sia limitato soltanto alla definizione di protocolli o algoritmi per il routing. In particolare, le tematiche relative alla QoS, nell'ambito che più ci interessa, cioè quello della distribuzione d flussi di dati, all'interno di un sistema in cui siano presenti utenti mobili e nomadici e in cui sia prevista una riconfigurazione dinamica dei percorsi per adattarsi alle condizioni del sistema, è un problema che deve essere affrontato su diversi livelli. In particolare possiamo distinguere almeno tre livelli di cui il primo riguarda la creazione di protocolli per il QoS routing; il secondo è relativo alla definizione di modelli per la descrizione dei flussi in termini dei servizi e dei componenti che necessitano e infine il terzo è relativo alla progettazione di strutture ed entità (come ad esempio i middleware) in grado di assicurare alle applicazioni che a queste fanno riferimento determinate garanzie, nell'ambito della QoS.
Ovviamente non si può pensare che i tre livelli indicati siano indipendenti l'uno dall'altro; al contrario, vi sono forti relazioni e

sovrapposizioni tra questi com'è infatti intuibile, la definizione di un modello di descrizione dei flussi può essere influenzato dai protocolli esistenti e viceversa ed entrambi sono fattori importanti nella definizione dell'architettura e dei servizi che un middleware può fornire. Tuttavia, in genere, ci si accosta al problema del QoS-based routing, considerando principalmente uno solo di questi aspetti.
Nei prossimi paragrafi intendiamo quindi presentare, per ognuno dei livelli individuati, un soluzione reale al problema posto.
I prossimi paragrafi presenteranno due protocolli per la realizzazione del QoS-based routing, mentre nei paragrafi successivi vedremo come il problema sarà affrontato nella definizione di un modello per la descrizione dei flussi e infine, chiuderemo il capitolo presentando un middleware che si propone di realizzare QoS nell'ambito delle reti ATM wireless.

2.3.2.1 Il protocollo Private Network-Network Interface.

Private Network-Network Interface (PNNI) è un protocollo per routing dinamico di tipo gerarchico, per le reti ATM che supporta il QoS. Il protocollo è basato sull'algoritmo link-state. Le informazioni sulla topologia (incluso le informazioni relative a nodi, links e indirizzi) sono inviate mediante flood attraverso la rete. Le risorse di rete sono definite mediante metriche e attributi (ritardo, banda disponibile, jitter, etc..) che sono raggruppati per classi di traffico. Poiché alcune delle metriche usate cambiano frequentemente (come la banda disponibile), sono utilizzati algoritmi con soglie per stabilire se un cambiamento in una metrica o per un attributo è significante da richiedere la propagazione dell'informazione aggiornata. In sostanza, viene realizzato un meccanismo di notifica di eventi. Poiché la struttura del protocollo è gerarchica, sono previsti i concetti di livelli e di nodi logici. Supporta inoltre l'aggregazione di informazioni relative alla topologia e alla raggiungibilità dei nodi. Altri servizi che mette a disposizione sono l'auto

configurazione della gerarchia di routing e i controlli sui permessi (viene considerato come una parte dell'algoritmo di calcolo dei percorsi).
Caratteristiche negative sono invece le seguenti: il PNNI routing protocol non fornisce un supporto al multicast ed inoltre eredità il problema sostanziale degli algoritmi basati sul link state, cioè come realizzare in maniera efficiente la propagazione, in broadcast, delle informazioni di stato, in particolare, nel caso di risorse velocemente variabili. (Per una visione completa del protocollo vedi [AF-PNNI]).

2.3.2.2 Ticket based probing protocol.

Questo tipo di protocollo, viene proposto per ad-hoc network in cui tutte le comunicazioni sono su media wireless (si veda: [DQSAHN]); in questo tipo di reti la topologia può cambiare frequentemente e le informazioni sullo stato delle risorse possono essere, di conseguenza, non precisamente definite. In questo scenario, si intende realizzare un protocollo distribuito che consideri più percorsi, in funzione di una metrica definita. In particolare, ogni nodo i della rete, mantiene informazioni aggiornate relative ad ogni proprio collegamento di uscita; l'informazione di stato dei link, comprende: il ritardo del canale (ritardo dovuto al mezzo trasmissivo, all'accodamento, ecc..), la banda disponibile e il costo del collegamento. La base di questo protocollo nasce da due osservazioni: la prima è che il QoS routing è orientato alla connessione; da qui l'idea di non utilizzare meccanismi di flooding per la definizione del percorso perché questi inonderebbero tutta la rete mentre si vuole localizzare l'attività di routing in una parte limitata della rete tra la sorgente e la destinazione. La seconda osservazione, è che esistono diversi percorsi tra sorgente e destinazione e non si vuole sceglierne uno a caso, ma effettuare nodo dopo nodo, effettuare una soluzione intelligente. L'idea del protocollo è la seguente: un ticket è un permesso di cercare un percorso; la sorgente inizia a cercare un percorso emettendo un numero N di ticket. Messaggi chiamati Probes vengono allora mandati dalla sorgente verso la

destinazione, ognuno con un certo numero di ticket (nei messaggi di Probe devono sempre esserci uno o più ticket), per cercare il percorso di costo minore che soddisfi le specifiche di QoS. Ad ogni nodo intermedio, un Probe con più di un ticket può suddividersi in più messaggi di Probes (testando quindi cammini multipli) tale che la somma dei ticket complessivamente sia pari al numero di quelli portati dal messaggio originario. Con questo schema, l'overhead dovuto alla ricerca di un percorso è limitata dal numero dei ticket e lo stesso vale per i possibili percorsi individuabili. Inoltre, tramite questo protocollo, è possibile lavorare con informazioni di stato non certe (il grado di incertezza influirà sul numero dei ticket emessi dalla sorgente), di evitare un processo di elaborazione centralizzato del percorso, usando le informazioni locali dei nodi (che in questo modo possono evitare di mantenere le informazioni sulla rete nel complesso). Naturalmente il problema principale di questo tipo di protocollo sarà stabilire appunto il numero di ticket che il nodo sorgente dovrà emettere e come ripartire i ticket tra i messaggi di Probes nei nodi intermedi.

2.3.2.3 Logical Application Stream Model.

Presentiamo qui, un modello di descrizione di flussi di dati preso da [ARDMS]. L'idea di base di questo modello, è che, mentre per le tradizionali applicazioni end-to-end era sufficiente un modello per il flusso di tipo:
source computer ->communication I/O -> sink computer,
Nel caso invece di applicazioni che coinvolgano diverse elaborazioni sul flusso oltre a specifiche di QoS, si deve estendere il modello in questo modo:
source computer ->communication I/O ->intermediate computer-> …..->sink computer.
In questo scenario, il Logical Application Stream Model (LASM) descrive le caratteristiche del carico e le esigenze temporali e di elaborazione della serie di elaborazioni che si vuole eseguire sul flusso.

Il modello è costituito da una serie di passi di elaborazione logici. Ogni passo è un unita di esecuzione su una qualche risorsa (ad es. un thread è un passo di elaborazione che esegue su una risorsa di tipo ”processore”). Esiste poi una serie di vincoli di precedenza tra i passi di elaborazione. Ogni passo di elaborazione è infine descritto mediante degli attributi :

1) categoria: indica il comportamento del passo di elaborazione; sono previste tre categorie e cioè sorgente, destinazione e intermedio. Un passo di elaborazione sorgente genererà un output ogni volta che sarà in esecuzione mentre non avrà mai un input; un passo di tipo intermedio, avrà sempre in ingresso un certo input di dati e allo stesso modo produrrà dei dati in uscita; un passo di tipo destinazione, infine, semplicemente consumerà un certo input di dati e non ne produrrà in output.
2) il tipo di risorsa: specifica il tipo di risorsa su cui il passo di elaborazione viene eseguito.
3) Processing step rate: specifica la velocità a cui il processo di elaborazione esegue. Viene qui specificata anche la massima latenza per un passo i, come la durata massima consentita dall’inizio dell’esecuzione del passo sorgente fino al completamento dell’esecuzione del passo iesimo.
   Il massimo jitter specifica invece la massima variazione assoluta tra il completamento di due istanze di un passo di elaborazione.
4) Il modello di richiesta delle risorse(PRDM): identifica il carico che un passo di esecuzione comporta su una risorsa.

La definizione di questo modello logico di descrizione consente di definire un algoritmo integrato per il routing e l’allocazione delle risorse (vedi sempre [ARDMS])nell’ambito di un intranet per un flusso di dati soggetto a una sequenza di elaborazioni.
Non intendiamo qui presentare tale algoritmo in quanto ci interessa di più portare l’attenzione su come sia stato possibile risolvere il problema della QoS, partendo dalla definizione di un modello per la descrizione delle elaborazioni a cui è soggetto un flusso.

2.3.2.4 Il middleware mobiware.

Mobiware è un middleware in grado di supportare terminali mobili, il cui scopo è trasportare flussi di dati, ridurre cadute e ritardi nelle fasi di hand-off (cioè in cui a sessione viene temporaneamente sospesa per permettere lo spostamento del terminale) e migliorare l'utilizzo delle risorse per i dispositivi wireless. In mobiware è implicita la nozione che i flussi sono rappresentati e trasportati verso i dispositivi mobili come flussi scalabili a più livelli; per i quali cioè esisterà un livello base e una serie di miglioramenti possibili nella qualità (con conseguente maggiore quantità di dati trasportati). mobiware fornisce una piattaforma per facilitare, nella creazione di servizi, il monitoraggio e l'adattamento dei flussi multimediali, dove, con adattamento si intende il processo di scaling dei flussi in caso di risorse limitate (tipicamente di banda).

In questo paragrafo, non intendiamo scendere nel dettaglio relativamente all'architettura e ai meccanismi di mobiware (si veda per questo [QSMMC]), presenteremo qui, infatti, solo gli aspetti che ci sono parsi più interessanti, soprattutto in considerazione del fatto, che anche nell'ambito di questo progetto si intende operare su un middleware.

Mobiware promuove una netta separazione tra segnali dei dispostivi mobili e gestione dell'adattamento dei flussi da un lato e trasporto dei flussi dall'altro. Nel middleware sono presenti dei cosiddetti transport object (ATO) che costituiscono dei componenti attivi del sistema di trasporto del middleware, il quale può distribuire questi ATO nei punti strategici della rete o nei terminali di confine, per fornire un supporto alla QoS. Un interessante esempio dei servizi offerti dagli ATO, è la realizzazione dei filtri mobili per lo scaling dei flussi; durante il passaggio da una cella ad un'altra (quindi durante l'hand off della sessione) il middleware stabilisce se nella nuova cella vi sono risorse sufficienti per mantenere le stesse caratteristiche del flusso, se non è così, genera un ATO che realizzi lo scaling del flusso. Ad un successivo cambiamento,l'ATO potrà seguire il terminale spostandosi verso la

nuova cella in caso di necessità. Sempre a riguardo dei filtri mobili, è interessante la politica adottata dal middleware per la loro disposizione lungo il percorso; se infatti la carenza di risorse è dovuta all'ingresso del terminale in una nuova cella (con risorse inferiori) il filtro viene attivato immediatamente a monte della cella; se invece la carenza di risorse viene segnalata in un nodo lungo il percorso, il filtro viene attivato nel nodo più a monte possibile. La differenza tra i due casi, è che nel primo si suppone, che vi possano essere passaggi frequenti da una cella ad un'altra e non ha senso quindi modificare il flusso lungo tutto il percorso in quanto vi è una possibilità sufficientemente alta di entrare in una nuova cella in cui lo stato delle risorse potrebbe essere migliore (e se così non fosse l'ATO si sposterebbe seguendo il terminale); nel secondo caso, la scelta attuata deriva dalla considerazione che il percorso del flusso, tenderà a cambiare (per i movimenti dell'utente) più dal lato del terminale che non da quello della sorgente e quindi, una mancanza di risorse in un nodo intermedio difficilmente sarà risolta da uno spostamento del client.
Un altro aspetto interessante di mobiware è la gestione degli spostamenti dei terminali e la riconfigurazione dei percorsi. Mobiware infatti associa molto semplicemente, ad ogni terminale, un albero di multicast. Quando l'utente è fisso in una cella, ovviamente l'albero di riduce a un percorso singolo; quando però l'utente è in movimento, il middleware può predisporre una prenotazione anticipata delle risorse nelle possibili celle di destinazione e creare una serie di percorsi alternativi semplicemente aggiungendo dei rami all'albero di multicast (come nel caso in cui si unissero nuovi utenti).Una volta poi che l'utente è transitato nella nuova cella, l'eliminazione dei percorsi inutilizzati viene effettuata semplicemente rimuovendo i rami non coinvolti. Con questo metodo, la gestione del meccanismo di re-routing per seguire il movimento degli utenti è di fatto molto semplice a livello del middleware, sia da un punto di vista logico che pratico.

2.4 Conclusione.

In questo capitolo abbiamo delineato non solo gli aspetti da tenere in considerazione in un modello per un sistema di località ma abbiamo anche visto, come, le informazioni che verranno ottenute grazie a questo, possono essere utilizzate per la realizzazione e la gestione di specifiche di QoS per flussi di dati.

Nella prima parte del capitolo, abbiamo visto come l'organizzazione delle reti possa essere strutturata secondo principi logici che ne determinano la topologia. Nella pratica, una rete di reti, raramente è realizzata secondo uno specifico modello di topologia per quanto riguarda il lato fisico. Al livello in cui noi intendiamo operare, invece, è spesso possibile individuare una overlay network che adotti un tale modello,

Nell'ultima parte, infine, abbiamo introdotto gli aspetti fondamentali relative alla QoS: sono state presentati alcune metriche e alcuni criteri per la valutazione dei percorsi e infine, nell'ultima parte sono stati presentati alcuni casi in cui si è evidenziato come esistano, di fatto, diversi approcci al problema della QoS, ognuno dei quali in grado di mettere in luce e realizzare interessanti risultati. Nel prossimo capitolo intendiamo concludere la parte introduttiva, relativa a questo progetto, presentando gli strumenti che verranno utilizzati per la sua realizzazione. I capitoli dal quattro fino al sei saranno invece dedicati alla descrizione del progetto realizzato e alle conclusioni tratte.

CAPITOLO 3.

3. SOMA e MUM.

In questo capitolo, intendiamo introdurre gli strumenti che verranno utilizzati per la realizzazione del nostro progetto.
In particolare introdurremo nella prima parte la piattaforma SOMA (Secure and Open Mobile Agent). Il sistema di località verrà infatti realizzato come estensione di questa piattaforma; è da notare, come SOMA offra già un'astrazione di località che chiama domini. Tali domini vengono poi organizzati in una struttura ad albero in modo da consentire un'organizzazione efficiente del sistema. E' tuttavia questa stessa scelta che rende utile la sovrapposizione del sistema di località all'organizzazione esistente.
Nella seconda parte introdurremo invece il middleware MUM (Multimedia agent based Ubiquitus Multimedia Middleware), in fase di sviluppo presso il Dipartimento di Elettronica, Informatica e Sistemistica (DEIS) dell'Università di Bologna. Il middleware è stato scelto in quanto è già prevista, a livello di architettura, sia la possibilità di servire utenti nomadici e terminali mobili, che la presenza di un sistema di riconfigurazione per i singoli flussi che verrà integrato nell'ambito del nostro progetto.

3.1 La Piattaforma SOMA.

SOMA [SOMA] è un ambiente ad agenti mobili realizzato presso il Dipartimento di Elettronica Informatica e Sistemistica (DEIS) dell'Università di Bologna. Qui se ne vogliono delineare semplicemente le caratteristiche architetturali principali, mantenendo come riferimento la teoria generale esposta nei paragrafi precedenti.
L'esigenza, a livello realizzativo, di SOMA era il superamento del problema dell'eterogeneità delle reti, e la realizzazione di un meccanismo di migrazione per le applicazioni che su tale piattaforma intendessero appoggiarsi. Entrambi questi obiettivi sono stati realizzati tramite l'implementazione mediante il linguaggio Java.

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

1) essendo interpretato, può eseguire su qualunque macchina possegga una Java Virtual Machine (JVM) in grado di trasformare il bytecode in istruzioni macchina; è, cioè, portabile su diverse architetture software/hardware;
2) è un linguaggio object-oriented che permette uno sviluppo modulare del codice per estensione ed ereditarietà, così che l'utente possa scrivere i propri agenti con uno sforzo limitato a partire dalle classi messe a disposizione dalla piattaforma;
3) prevede un meccanismo di caricamento dinamico delle classi anche da sorgenti remote e di collegamento dinamico all'applicazione corrente;
4) fornisce la possibilità di serializzare gli oggetti, cioè di rappresentarli come stream di byte e di trasferirli sulla rete;
5) ha caratteristiche di sicurezza built-in, la più nota è quella associata alle applet che, originariamente, potevano essere scaricate da un Web server, ma non avevano diritto di accedere alle risorse del sistema locale. Dalla versione 1.2 del linguaggio si ha una architettura di sicurezza rivisitata, molto più flessibile ed espressiva, fatta di domini di protezione, controlli d'accesso e permessi da associare sia a codice remoto che a codice locale. D'altra parte, come evidenziato in precedenza, il linguaggio obbliga alla scelta di un determinato modello di mobilità. Essendo un linguaggio interpretato, una parte dello stato di esecuzione rimane incluso nello stato dell'interprete; la cattura di tale stato diventa praticamente impossibile se non modificando l'interprete, ma questo, oltre a problemi intrinseci, porterebbe alla perdita della portabilità, che è una delle caratteristiche fondamentali di un sistema ad agenti.

facendo uso della terminologia e delle definizioni introdotte nel primo capitolo, SOMA è allora un sistema a mobilità debole, in cui cioè, ciò che si muove non è l'immagine dello stato di esecuzione di un processo in un istante qualsiasi ma un suo stato di esecuzione ben definito e creato a livello applicativo; si può quindi dire che:

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

3.1.1 Astrazioni di località e topologia in SOMA

Come abbiamo anticipato, SOMA fornisce un' astrazione di località. Tali località, che vengono chiamate domini, sono adatti alla descrizione di diversi scenari di connessione e la loro idea è mutuata da Internet; esistono, infatti, diversi domini all'interno dei quali vengono distinti sotto-domini, fino a giungere al semplice nodo.
Il mondo in cui vivono gli agenti è allora costituito di Place e Domini (vedi figura 3.1).

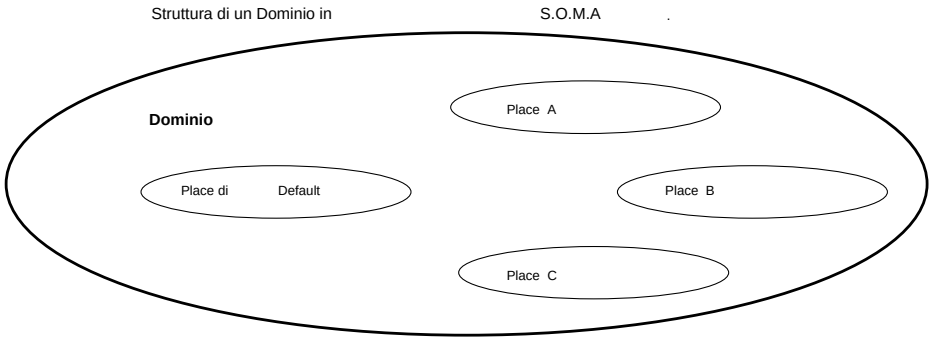


Fig. 3.1

I domini sono quindi organizzati secondo un'organizzazione gerarchica. Tutti gli elementi all'interno di un dominio sono infatti considerati "figli" di quel dominio, che di conseguenza ne è il padre. L'organizzazione che si viene a creare, è quindi quella di una struttura ad albero, in cui i Place normali costituiscono le foglie (cioè nodi terminali) mentre i Place di Default possono rap-

presentare sia nodi che foglie. Adottare questo tipo di topologia, permette l'identificazione univoca di un certo place/default place, tramite il percorso che lo lega al default place "radice", cioè tramite la lista di DefaultPlace che devono essere attraversati, partendo dal DefaultPlace radice, fino al place/default place stesso.
La struttura così delineata, deve corrispondere ad una rete reale, vediamo quindi ora qual è, in generale, la corrispondenza tra gli elementi della topologia e quelli di una struttura reale.
Il **Place** è il contesto di esecuzione dell'agente e sussume il concetto di nodo: il place può infatti corrispondere ad una macchina fisica, ma su un nodo possono convivere anche più place, permettendo, ad esempio, la definizione di località di protezione delle risorse. Anche se generalmente, viene assunto che un Place sia costituito da una postazione fissa, SOMA in realtà fornisce anche il supporto alle postazioni mobili; in particolare, tali postazioni sono rese tramite l'astrazione di un **Mobile Place**. Concettualmente il Mobile Place viene trattato come un Place normale quando si trova in un dominio ma è comunque necessario tenere presente che esso può spostarsi e registrarsi presso un altro dominio durante il corso della sua esistenza.
Il **Dominio** è un'aggregazione di place che può rispecchiare un contesto reale, come una LAN, oppure una caratteristica logica, ad esempio l'insieme dei dispositivi dello stesso tipo o appartenenti ad uno stesso dipartimento all'interno di un'organizzazione.
Nell'implementazione, il concetto di Dominio si riscontra solo nella distinzione tra place generici e place cosiddetti di "default"; questi ultimi hanno conoscenza dei siti costituenti un dominio e sono punto d'accesso da e verso l'esterno. Per quello che riguarda l'implementazione del sistema oggetto di questo lavoro di tesi, la corrispondenza fra place e nodi è di tipo 1:1. Il motivo di questa scelta è che in questo modo il place può essere utilizzato come utile astrazione di località fisica. Imponendo questa corrispondenza 1:1 il place rappresenta, per le applicazioni sviluppate al di sopra del middleware basato su SOMA, l'unico punto di accesso alle risorse di sistema di un certo nodo, risolvendo alcune delle

problematiche di gestione delle risorse. Ad esempio, come vedremo, alcuni servizi di base, i sottosistema per la prenotazione delle risorse, vengono realizzati a livello di place, in questo modo si ha, come è giusto, un unico gestore delle risorse per singola macchina.

Come abbiamo anticipato nel capitolo precedente, la struttura della topologia, che si viene a creare, ha lo svantaggio di determinare un solo percorso diretto tra due nodi dell'albero. In SOMA, questo problema, è parzialmente risolto nella realizzazione dei servizi di informazione tuttavia, la soluzione che viene adottata, come vedremo, non è adatta alle esigenze di applicazioni che intendano riconfigurare il percorso di flussi multimediali da cui la necessità della realizzazione di un'estensione di tale servizio di informazioni.
In figura 3.2 riportiamo un esempio che utilizza una gerarchia di semplice comprensione. Il default place root sarà il default place Mondo, e poi ci sono tutti i vari sotto-domini, ecc.; ad esempio il place "Toscana" è univocamente determinato dal percorso [Mondo, Europa, Italia, Toscana].

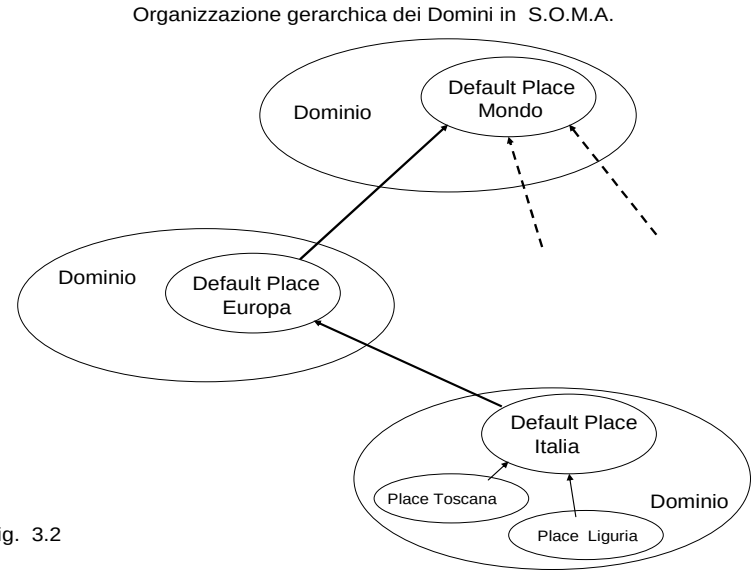


Fig. 3.2

3.1.2 Introduzione all'architettura di SOMA

In questo paragrafo intendiamo presentare l'architettura di SOMA a livello dei modelli e componenti logici con cui è realizzata; lo scopo è fornire una veloce panoramica sull'architettura di SOMA; nel paragrafo successivo invece ripresenteremo i componenti dell'architettura che verranno usati o modificati nell'ambito di questo progetto di tesi, in particolare ci soffermeremo sui dettagli dell' *Information Service*.

3.1.2.1 L' ambiente di esecuzione

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

1) *Agent Manager*, che gestisce gli agenti permettendone l'esecuzione, l'ingresso e uscita da un place; l'insieme delle funzionlità dell'AgentManager non verrà sostanzialmente modificato nell'ambito di questo progetto; l'AgentManager è tuttavia lo strumento attraverso il quale, l'agente ha accesso all'ambiente del Place in cui si trova e pertanto anche alle informazioni relative alla località del Place in cui si trova.
2) *Network Manager*: che gestisce le interazioni tra i place, mantenendo i necessari canali di comunicazione; in SOMA, la comunicazione avviene direttamente, esclusivamente tra Place che hanno visibilità diretta; il Network Manager dovrà pertanto essere modificato per tenere in considerazione l'ampliamento di visibilità dovuto al sistema di località.
3) *Information Service*: che gestisce le informazioni sui place appartenenti ad un dominio e sui domini ad esso noti e si distingue in DomainNameService e PlaceNameService.
   All'interno dell'Information Service verrà realizzato il Locality Service che manterrà le informazioni necessarie per la località e fornirà gli strumenti di accesso a queste.

3.1.2.2 Identificazione di Agenti e Place

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

3.1.2.3 Comunicazione in SOMA

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

realizzabile grazie al servizio di localizzazione degli agenti offerto dal supporto e attivabile su necessità.

3.1.2.4 Sicurezza in SOMA

SOMA garantisce il rispetto di una politica di autorizzazione così che agenti maliziosi non interagiscano in modo incontrollato con le risorse e i servizi messi a disposizione dall'ambiente. La definizione di diverse astrazioni di località, inoltre, consente di introdurre politiche di sicurezza nelle quali le azioni siano controllate sia a livello di dominio, sia a livello di place. Il dominio definisce una politica di sicurezza globale che impone autorizzazioni e proibizioni generali; ogni place, però, può applicare restrizioni ai permessi consentiti a livello di dominio. Questo modello di sicurezza si basa sui meccanismi offerti dal linguaggio Java. In particolare, il ClassLoader e la gestione delle politiche sono adattati alle specifiche esigenze di un sistema ad agenti; così, ad esempio, è stato definito un *AgentClassLoader* ad hoc.
Agli agenti vengono attribuiti specifici permessi in base alle azioni che devono compiere; ogni permesso è caratterizzato da un target, cioè una risorsa locale, e da un numero di azioni permesse. Esistono permessi predefiniti come, per esempio, quello di accesso ad un Place (PlaceAccessPermission) e quello per ottenere il riferimento all'ambiente del Place (AgentPermission). Per consentire l'attribuzione dei permessi giusti ad ogni agente, il Sistema deve identificare il relativo Principal, cioè l'entità che rappresenta il responsabile per l'agente (sia esso uno user, una compagnia, ecc...). Tale Principal, in Java viene rappresentato mediante un CodeSource composto da un URL e da un insieme di chiavi crittografiche pubbliche. Le chiavi consentono la verifica delle credenziali associate all'Agente (che sono firme digitali applicate al codice). Una volta nota e autenticata l'identità del Principal di un agente, quest'ultimo possono essere associati i permessi previsti dalla politica del Place.

3.1.3 Dettagli della Piattaforma SOMA.

Affrontiamo in questo paragrafo i dettagli di realizzazione delle componenti dell'architettura di SOMA che verranno principalmente utilizzate nell'ambito di questo progetto di tesi. In particolare, ci soffermeremo sui componenti che realizzano l'Information Service.

3.1.3.1 Gli Agenti

Un agente è realizzato come classe derivata dalla superclasse astratta *Agent* ed è un semplice oggetto passivo serializzabile, proprio perchè la JVM non permette la migrazione di unità di esecuzione, quindi di thread. Così quando un agente nasce viene affidato ad un thread (*AgentWorker*) che gli associa un flusso di esecuzione, lo stesso avviene all'arrivo dell'agente su un nuovo nodo. L'attributo principale di un agente è il suo identificatore unico (*AgentID*) da cui è possibile dedurne l'origine, ovvero il place di istanziazione. Un altro attributo è la *Mailbox*, mezzo di invio/ricezione di messaggi ad altri agenti. L'agente può, infatti, essere creato *Traceable* oppure no. Da questo dipende la possibilità di rintracciarlo, quindi di recapitargli i messaggi. Se un agente è traceable il gestore degli agenti del suo place di origine è responsabile della conoscenza della sua posizione corrente, l'informazione è mantenuta aggiornata grazie ad un meccanismo di notifica da parte di ogni place ricevente al place di origine. L'agente può interagire con l'ambiente di esecuzione solo tramite il campo *agentSystem* di cui è dotato: esso rappresenta l'interfaccia tra agente e sistema, cioè il punto di accesso a risorse e servizi. Si tratta di un campo *transient*, perché non sopravvive alla migrazione e viene assegnato all'agente dal place in cui di volta in volta si trova. Tutte le migrazioni avvengono tramite invocazione del metodo *go*(*PlaceID*,*String*) dove il programmatore deve specificare l'identificatore del place di destinazione e il nome del metodo da cui ripartirà l'esecuzione; in questo modo si riesce a simulare il controllo di flusso tipico della mobilità forte. La migrazione del codice dell'agente e di tutte le eventuali classi da questo riferite

avviene ad opera di un ClassLoader specializzato, e cioè l' *Agent-ClassLoader*, che richiede trasferimento del codice da remoto solo su necessità (paradigma COD) e lo mantiene in una cache locale per utilizzi successivi.

3.1.3.2 L'accesso ai Servizi : l' Environment

Come anticipato il place è l'ambiente di gestione ed esecuzione degli agenti e, in quanto tale, deve rendere disponibili tutti servizi di comunicazione, naming, sicurezza e migrazione. Per consentire ciò, ogni Place fornisce un contenitore che raggruppa i gestori di tali servizi. Tale contenitore è l'*Environment* del Place (che è realizzato da un'omonima classe) e, di fatto si può identificare con il Place a tal punto che, a livello implementativo, l'istanziazione di un nuovo Environment comporta l'effettivo avvio di un place.
Possedere un riferimento all'environment locale significa avere completo accesso alle funzionalità del supporto; ogni attributo di classe, infatti, corrisponde ad un fornitore di servizi logicamente correlati o ad un data base di informazioni (vedi figura 3.3).

Componenti principali della classe Environment

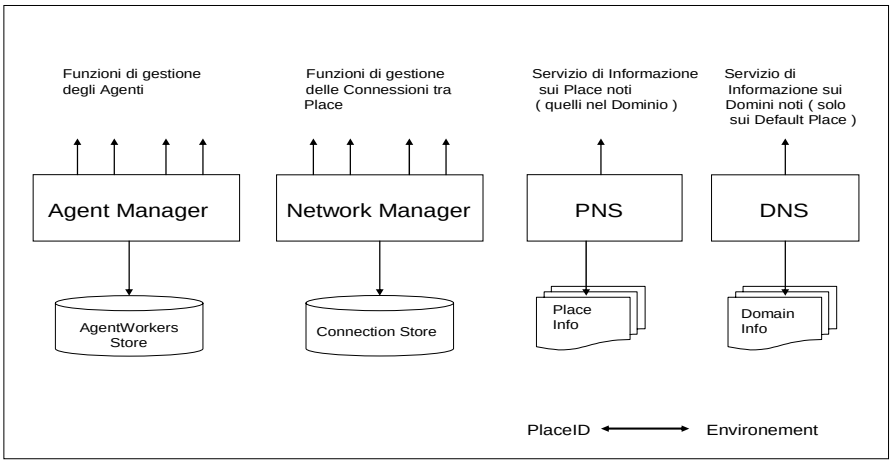


Fig. 3.3

Risulta allora evidente perchè non si sia stato reso direttamente accessibile dalla classe AgentID: è necessario discriminare gli accessi e lo si può fare a partire da AgentSystem che consente di creare "viste" differenziate dell'ambiente. Si vogliono ora

descrivere gli attributi della classe Environment che corrispondono ad altrettanti moduli del supporto SOMA

3.1.3.2 Gestore degli Agenti

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

3.1.3.3 Informazioni su Place e Domini : l'Information Service

Come abbiamo anticipato, l'Information Service, è realizzato da due componenti : il PlaceNameService e il DomainNameService; come è facilmente intuibile dai nomi stessi, il primo di questi componenti fornisce le informazioni su quelli che sono i Place semplici che appartengono a un dominio del sistema mentre il secondo mantiene le informazioni su i Place che rappresentano l'astrazione di un Dominio e che vengono chiamati defaultPlace. La presenza di due sistemi di nomi è dovuta, come abbiamo visto, all'organizzazione della topologia di SOMA. In particolare, si vuole mantenere limitata al solo dominio in cui si trovano, la visibilità per i Place normali. Per i DefaultPlace invece, la visibilità non viene estesa, tramite appunto al DomainNameService, a tutti gli altri DefaultPlace del sistema. Tale forma di visibilità, in realtà non rispecchia la struttura della topologia in quanto, in questo modo, ad un agente risultano accessibili, direttamente, tutti i DefaultPlace

del sistema, come se vi fosse una connessione diretta tra questi (connessione che nella pratica può esistere ma che non è rispecchiata dalla topologia del sistema). In sostanza, ai Default Place viene impedita la vista degli elementi appartenenti a un dominio al di fuori del proprio, ma viene fornita la possibilità di comunicazione diretta con ogni altro DefaultPlace in una qualunque posizione nella topologia. La scelta di fornire una tale visibilità è dovuta ovviamente a ragioni di praticità in quanto limitare la visibilità, renderebbe necessaria la costruzione in ogni DefaultPlace, di tabelle di instradamento per ogni comunicazione replicando un'elaborazione già svolta di fatto ai livelli sottostanti. Tale scelta tuttavia presenta di fatto anche degli inconvenienti in particolare per quanto riguarda la necessità di reperire informazioni sullo stato delle risorse negli elementi della topologia più vicini; se ci si vuole limitare alle informazioni dei nodi padre e figli, tale operazione è ancora possibile in quanto il DNS mantiene separatamente le informazioni relative a tali DefaultPlace, se si vuole estendere invece la propria conoscenza in un raggio più ampio, ciò non è più possibile in quanto non si hanno informazioni sulla propria posizione nella topologia rispetto agli altri nodi (in realtà, sarebbe comunque possibile se si interrogassero i nodi direttamente connessi : padre e figli, e questi interrogassero a loro volta i confinanti, e così via in modo recursivo fino alla distanza desiderata; ovviamente è evidente che questo protocollo sarebbe piuttosto inefficiente sia a livello di tempo che di risorse di rete impegnate). Questa incapacità di stabilire la propria posizione rispetto agli DefaultPlace risulta quindi, nell'ottica del nostro progetto, il primo limite dell' Information Service.

Entrambi sono comunque assimilabili a delle tabelle i cui elementi sono, nel primo caso, degli oggetti PlaceInfo mentre, nel secondo caso, degli oggetti DomainInfo; pur rappresentando entità concettualmente diverse, all'atto pratico DomainInfo e PlaceInfo sono sostanzialmente uguali in quanto le informazioni che mantengono sono : l'identificatore del Place a cui si riferiscono (il PlaceID) e

l'indirizzo di rete (nel nostro caso indirizzo ip + porta) su cui il Place è in ascolto per le comunicazioni con gli altri Place. La differenza fondamentale quindi tra i due servizi di nomi è sul tipo di visibiltà che offrono della topologia di SOMA in quanto all'atto pratico le stesse informazioni descrivono sia un dominio che un Place comune. Tale omogeneità, nella descrizione di domini e place normali, è, a nostro avviso, il secondo limite dell'Information Service di SOMA. Vediamo ora quali sono le funzionalità dei due servizi di nomi. Il PlaceNameService, come abbiamo visto, fornisce le indicazioni relative ai Place nel dominio, fornisce quindi, in particolare, i metodi per registrare e cancellare un Place nella tabella dove mantiene le proprie informazioni. Allo stato attuale tuttavia, il meccanismo di cancellazione di un Place non è realizzato; si assume, pertanto, che una volta che un nuovo Place sia entrato in un dominio, la sua postazione sia fissa. Tale ipotesi, che se non consideriamo l'eventualità di caduta di un Place può ritenersi non particolarmente limitante, diviene però un vincolo eccessivamente forte nel caso di utenza collegata tramite terminali mobili : ossia nel caso dei Mobile Place.
Una situazione analoga, si presenta nel caso del DomainName-Service; anche in questo caso, sono messi a disposizione i meccanismi di inserimento e rimozione di una DomainInfo (oltre a servizi di utilità generale) tuttavia nel Sistema non viene implementato un Sistema per il controllo della reale validità di tali informazioni. In particolare, anche se diversamente dai Place normali, i DefaultPlace non possono essere terminali mobili, la necessità di un meccanismo di controllo nasce dall'eventualità di caduta di un nodo. Se infatti tale ipotesi era trascurabile per quanto riguardava i Place normali in quanto non comportava danni eccessivi per il sistema nel suo complesso, la caduta di un Default Place comporta concettualmente la divisione della topologia in due sottoreti disgiunte. Nella pratica questo problema viene parzialmente risolto grazie alla visibilità totale che offrono i DNS ma per agenti che si spostino di nodo in nodo o per protocolli / applicazioni che si basino sulla topologia del Sistema il problema

rimane. In questo caso, pertanto, non solo sarebbe necessario un meccanismo per verificare che un DefaultPlace non sia caduto ma anche, nel caso, il successivo ripristino della topologia.
Abbiamo fino a qui presentato i limiti e le carenze dell'Information Service; tali limiti verranno superati grazie all'introduzione del nuovo sistema di località.

3.1.3.4 Interazione tra Place : gli oggetti Command

Le interazioni tra place diversi avvengono tramite normali comunicazioni via socket, ma ciò che è particolare è l'oggetto dello scambio: un comando. I comandi derivano dalla classe astratta *Command* che ne presenta l'interfaccia di riferimento. Il metodo *start()* è quello invocato da un place all'atto della ricezione, al suo interno si ha la creazione di un nuovo thread cui si affida l'esecuzione del metodo *run()*, dove viene racchiusa la computazione associata al comando concreto (paradigma REV). La registrazione di un place presso il place di default può essere un esempio di utilizzo di comando: il PNS richiede al NetworkManager l'invio di un'istanza di una particolare sottoclasse di Command, il *PlaceRegisterCommand*; questo comporta il recupero o la creazione della Connection verso il place di default, per poi invocarne il metodo *send(Command)*. All'altro capo della comunicazione il demone responsabile (un'altra istanza di Connection) accetterà il comando e ne avvierà l'esecuzione che comporterà una chiamata di registrazione al PNS locale.
Nell'ambito della realizzazione del sistema di località in SOMA, la classe Command rivestirà un ruolo molto importante in quanto sarà tramite specifici comandi che verranno realizzati tutti i protocolli per la creazione, aggiornamento e recovery delle località. Tale scelta è dovuta, in primo luogo, a ragioni di omogeneità con le strutture preesistenti, che fanno appunto utilizzo di questo strumento, e in secondo luogo al fatto che, l'utilizzo dei Command, consente una soluzione concettualmente più flessibile rispetto all'utilizzo di altri meccansimi.

3.1.3.5 Gestore di Rete

La gestione delle comunicazioni relative ad un place è logicamente a sé stante ed anche nell'implementazione è stata associata ad una classe specifica: il *NetworkManager*, anch'esso raggiungibile tramite l'Environment. Il networkManager ha conoscenza delle connessioni stabilite con altri place dello stesso dominio grazie alla struttura *ConnectionStore*, una sorta di contenitore delle associazioni tra connessioni, istanze della classe Connection, e identificatori di place coinvolti. All'inizializzazione di un place viene anche creato un *ConnectionServer*, demone che attende richieste ed attiva connessioni. A loro volta le istanze di *Connection* non sono altro che demoni responsabili delle comunicazioni via socket con un altro place. Oggetto della comunicazione gestita dai networkManager sono i Comandi presentati nel paragrafo precedente. Ogni comando viene indirizzato verso un particolare Place di destinazione; il network manager reperisce le informazioni relative al Place tramite l'InformationService del Place ed eventualmente richiede l'attivazione di una connessione con tale Place. Una volta stabilita la connessione invia quindi il comando. Nel caso in cui il comando sia invece indirizzato verso un Place non conosciuto dall'InformationService (caso di un Place normale di un altro dominio), il networkManager genera un Comando di Trasporto (TransportCommand); tale comando, ingloba il comando originario e viene spedito verso il DefaultPlace proprietario del dominio in cui è presente il Place non noto all'InformationService. Una volta giunto sul DefaultPlace, il TrasnportComand provvede a rispedire il comando originario al Place giusto (questa volta sicuramente noto in quanto si è nel suo dominio).

3.2 Il middleware MUM.

Passiamo ora a presentare il middleware MUM. Come abbiamo già anticipato all'inizio di questo capitolo, MUM è un middleware, costruito su SOMA, il cui intento è quello di fornire le infra-

strutture e i servizi necessari per la realizzazione di applicazione per la consegna e la fruizione di materiale multimediale in un sistema distribuito. Tra i servizi offerti, MUM mette a disposizione delle applicazioni utente la possibilità di monitorare lo stato della sessione, cioè di verificare se i requisiti richiesti per la fruizione del materiale multimediale sono o meno rispettati e, in caso non lo siano, di attivare una fase di riconfigurazione. Nel prossimo capitolo, vedremo come, il servizio di riconfigurazione offerto da MUM, verrà esteso per consentire una gestione più flessibile di questa fase. In questa sezione intendiamo invece presentare la struttura e il funzionamento del middleware.

3.2.1 Scenario per lo sviluppo del MiddleWare

Il middleware intende supportare la pubblicazione e fruizione di **presentazioni multimediali** di varia natura e qualità diverse su rete fissa e mobile. Con presentazione multimediale intendiamo un insieme di diversi **oggetti multimediali** (cioè istanze di un dato multimediale. Ogni oggetto multimediale, si riferisce infatti, ad un unico tipo di media (audio o video) e contiene informazioni di formato, compressione e durata; in pratica, nel sistema, saranno file) i cui singoli elementi facciano riferimento allo stesso contenuto multimediale. In sostanza, un particolare filmato audio video, per esempio, rappresenta un **contenuto multimediale**; se esso è presente nel sistema in diversi formati, con diverse qualità, ecc.. ad ognuna di queste corrisponde una presentazione multimediale. Queste, a loro volta, sono composte da oggetti multimediali (per esempio, nel caso in cui l'audio di un filmato, sia disponibile in diverse lingue, la presentazione multimediale relativa, sarà costituita dall'oggetto multimediale che mantiene le informazioni video e dai diversi oggetti multimediali che contengono le informazioni audio nelle diverse lingue).
Gli utilizzatori del sistema, saranno inoltre preventivamente registrati e saranno caratterizzati mediante un **profilo** che conterrà

le informazioni relative come l'insieme dei terminali dal quale l'utente può accedere, la loro posizione, ecc..
Essi potranno richiedere il delivery dei titoli presenti all'interno del sistema distribuito, comandare la presentazione, attraverso una interfaccia simile a quella di un videoregistratore, richiedere lo spostamento della sessione verso un terminale diverso da quello che stanno attualmente utilizzando. Quando l'accesso avvenga da rete mobile, cioè attraverso un device di tipo wireless, si vuole che l'infrastruttura supporti il delivery delle presentazioni, riorganizzandosi per seguire i movimenti dell'utente.
L'infrastruttura in progetto si occupa inoltre della gestione delle risorse supportando prenotazione e monitoraggio delle stesse, nonché della scelta della presentazione più adatta alla piattaforma computazionale dalla quale un certo utente sta accedendo al sistema. Le caratteristiche delle diverse piattaforme computazionali saranno salvate in dei descrittori, e i profili utenti manterranno delle coppie con locazione della piattaforma dalla quale si accede e riferimento al descrittore della piattaforma. Da ultimo si vuole fare in modo che sia possibile, supportare il downloading e l'inizializzazione di software a runtime.

3.2.2 Entità per la fruizione del materiale multimediale

Il modello computazionale adottato è un'integrazione del tipico modello client-server con il modello ad agenti mobili.
La tipica architettura client-server è stata, poi, integrata con la presenza di entità intermediarie dai compiti generici : i proxy.
Abbiamo quindi:

1) **Client**: il client è l'end-point che riceve i flussi multimediali richiesti e presenta l'interfaccia di comando all'utente. Tramite tale interfaccia, l'utente è in grado di interagire con il middleware e di gestire la presentazione richiesta.
2) **Server**: il Server è l'altro end-point della comunicazione dei flussi multimediali (spedisce i flussi richiesti); è un' entità fissa scaricabile secondo necessità (COD). Inoltre, una volta lanciato su una certa macchina, verrà utilizzato per servire tutte

le richieste che da quel punto in poi arriveranno a quella macchina per quel tipo di server.

3) **Proxy**: i proxy sono entità che svolgono attività generiche o direttamente richieste dall'utente o comunque necessarie per la presentazione/consegna del contenuto multimediale richiesto; un caso tipico in cui è prevista la presenza di un componente proxy è quella in cui si desidera la fruizione del materiale multimediale tramite un terminale dalle ridotte capacità di elaborazione; in questo caso, un componente proxy può effettuare un adattamento della qualità, e riducendo quindi le esigenze in fatto di elaborazione, operando sul flusso, prima che questo venga consegnato al terminale utente.

In realtà, come vedremo successivamente, la realizzazione effettiva delle applicazioni per la fruizione del materiale multimediale prevede una struttura più complessa rispetto alla semplice presenza di client, proxy e server. L'architettura effettiva per le applicazioni utenti verrà presentata in seguito nell'ultima parte del capitolo.

3.2.3 Le caratteristiche del Middleware MUM.

Prendiamo ora in esame le caratteristiche e i servizi principali che il middleware intende offrire.

3.2.3.1 Movimento degli utenti.

L'architettura vuole supportare il movimento dei clienti fra diversi terminali. L'utente può iniziare la sessione su un terminale, e poi richiedere lo spostamento su di una altro terminale, verso il quale si sposta fisicamente. In modo trasparente all'utente stesso l'infrastruttura provvederà allo spostamento della sessione. In MUM, le postazioni in cui il client può spostarsi, devono essere indicati nel profilo utente. Esiste quindi un servizio per la gestione dei profili.

3.2.3.2 Movimento dei terminali.

L'utente si muove insieme al proprio terminale, e l'infrastruttura, interagendo col terminale stesso, al verificarsi del movimento si

modifica in modo da effettuare lo streaming non più dalla vecchia località, ma dalla nuova località; il tutto in modo trasparente per l'utente finale. Per questo tipo di servizio, il middleware, crea un Proxy che effettua un forward del flusso come ultimo elemento del path prima del client. Quando il terminale del client entra in un nuova località, un servizio di notifica di ingresso realizzato dal middleware, avverte tutti le entità registrate (i clientAgent per esempio) che possono quindi avviare la riconfigurazione del percorso del flusso.

3.2.3.3 Inizializzazione e riorganizzazione dinamica del sistema.

Quando si inizia una sessione di streaming, dapprima (se non è già presente) verrà scaricato il software necessario ad effettuare la trasmissione stessa, poi verrà configurato il path dal server al client, dopo di ché verrà iniziata la trasmissione vera e propria.
Il sottosistema di inizializzazione è in grado di scaricare e inizializzare le varie entità che partecipano al multimedia streming lungo tutto il path senza conoscerle a priori, in quanto, tali entità devono essere realizzate dallo rispondendo a delle interfacce prestabilite; in particolare, un decisore, creerà un piano di inizializzazione che specializzi tali entità nel percorso. Il decisore inoltre crea, se possibile, piani alternativi in modo che, nel caso che l'inizializzazione di un piano fallisca, si possa ricorrere al piano alternativo.

3.2.3.4 Gestione delle risorse

Il sistema di gestione delle risorse consente di prenotare preventivamente le risorse per un applicativo negoziando inizialmente la qualità di servizio con una entità che si occupi della ***prenotazione*** delle risorse, poi monitorando l'andamento della sessione, rinegoziando eventualmente a tempo di esecuzione la qualità di servizio. Vengono considerate le risorse più sensibili ai fini della fruizione di contenuti multimediali cioè la ***banda trasmissiva*** e la ***CPU***. Come abbiamo visto nei precedenti capitoli, per il sistema che si occupa del reperimento delle informazioni sullo stato delle

risorse nelle Località, sono state considerate anche la memoria primaria e la memoria secondaria. Ovviamente ciò non genera problemi ma al contrario, consente eventualmente di anticipare le possibili modifiche nel modello di prenotazione delle risorse del middleware. La gestione della qualità di servizio è poi definita in un dominio discreto, nel quale si "salta" da una qualità ad un'altra in modo discontinuo.
Viene realizzato un gestore delle risorse per ogni singola macchina in modo che più clienti lanciati sulla stessa macchina fanno riferimento allo stesso gestore delle risorse. Tale gestore verrà descritto molto dettagliatamente nei successivi paragrafi in quanto, sarà tramite il monitoraggio delle risorse che effettuerà che verrà attivato in generale la riconfigurazione dei componenti del sistema.

3.2.3.5 Struttura a livelli.

Il sistema viene infine organizzato in tre livelli, vedi figura 3.4. In questo modo, si è inteso creare una struttura modulare consentendo lo sviluppo ai livelli superiori di servizi realizzati dai livelli inferiori. I tre livelli realizzano quindi dal più basso al più alto specifiche e servizi via via più complessi; il **MUM Services Layer** implementa i servizi base, delle sorte di primitive che possono essere accedute dagli oggetti facenti parte del **MUM Middleware Layer**.

Suddivisione del Middleware (MUM) in livelli.

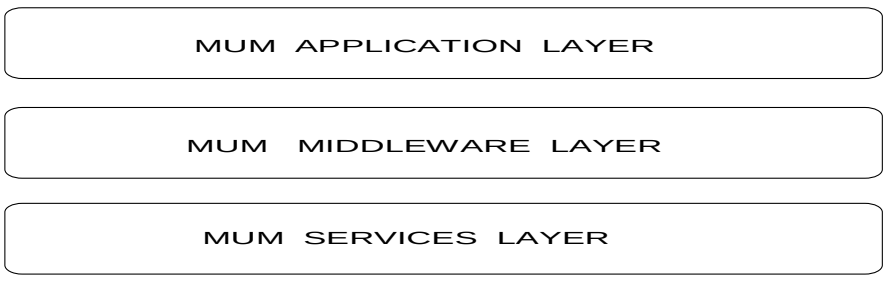


Fig. 3.4

Questo secondo livello, utilizzando i servizi offerti realizza funzionalità più complesse che mette a disposizione dello sviluppatore dell'applicativo. Il terzo ed ultimo livello (**MUM Application Layer**) racchiude invece tutte le entità che realizzano l'applicazione vera e propria, e l'interazione con l'utilizzatore del sistema.

3.2.4 La realizzazione del Middleware.

Nel precedente paragrafo, abbiamo introdotto le caratteristiche del Middleware MUM a livello dei requisiti che intende soddisfare; in questo paragrafo intendiamo invece soffermarci sui meccanismi che realizzano tali servizi. In particolare, non intendiamo fornire una presentazione esaustiva di ogni componente del Middleware (per una visione completa di tale middleware, si veda [GFRMI]) ma intendiamo soffermarci principalmente su quelli che saranno i meccanismi e le entità con cui si interagirà nell'ambito della realizzazione di questo progetto.

3.2.4.1 Servizio per l'istanziazione di entità.

Il middleware mette a disposizione un servizio per l'istanziazione delle entità. Esso è, in sostanza, una factory per la creazione di oggetti che debbano essere passati al livello applicativo o utilizzati durante lo svolgimento di uno dei servizi del sistema.

3.2.4.2 Servizio di downloading del software

Per la realizzazione di questo servizio si è deciso di utilizzare un agente mobile. L'interazione fra il cliente di questo servizio e il servitore è basata sul modello a eventi. Il cliente richiede cioè il downloading di un certo SW al servitore e il servitore, quando il SW sia stato scaricato, notifica l'avvenuta esecuzione del servizio scatenando un evento al cliente che si era preventivamente registrato presso di lui. Le ipotesi del sistema sono che tutto il codice scaricabile (COD) all'interno del sistema, sia presente almeno nel

nodo radice. Compito dell'agente è quello di andare alla ricerca del codice richiesto, ritornare al place di partenza, aggiornare il database locale. Non ci addentriamo oltre nella descrizione del meccanismo che implementa il downloading del codice, in quanto esso non viene direttamente coinvolto nell'ambito di questo progetto.

3.2.4.3 Servizio di attivazione delle entità multimediali.

Questo servizio viene utilizzato per attivare i server multimediali in uno dei nodi facenti parte del sistema, e, una volta attivato il server, per mantenere un riferimento al server, in modo da non dovere attivare un nuovo server per ogni nuova richiesta, ma avere, invece, una sola istanza di server per ogni tipologia di server multimediale. Ogni nodo mette a disposizione il servizio. Il servizio viene anche utilizzato per salvare i riferimenti ai Client istanziati, mantenendo una tabella che fa corrispondere ad ogni ClientAgent il proprio Client.
Questo servizio necessita di una struttura dati per la memorizzazione dei riferimenti ai server e ai client attivi. Tale struttura dati verrà conservata in memoria, e sarà distrutta ad ogni disattivazione del nodo. L'architettura logica di questo servizio è assai semplice, è infatti costituita da un unico manager IEntityManager (MultimediaServicesManager) che realizza le funzionalità descritte e mantiene aggiornata la struttura dati. Tale oggetto riceve in ingresso l'identificatore unico dell'unità server da avviare, costituito dal nome del package e dell'interfaccia, e avvia il server richiesto. Una volta avviato il server, sarà possibile ottenere un riferimento (indirizzo IP e porta) al server stesso, incapsulato in un oggetto EntityInfo. Per il salvataggio delle varie entità server si è fatto uso, di un DirExplorerItem, che racchiude vari MultimediaServerExplorerItem. Un DirExplorerItem è un menù introdotto dall'architettura SOMA, accedibile a riga di comando, che può contenere diverse voci di menù, in questo caso i vari MultimediaServerExplorerItem. Ognuna di queste voci permette di avviare i Server anche da riga di comando da parte di un amministratore di

sistema e questi ultimi possono essere fermati e riavviati, come un qualsiasi demone facente parte dell'architettura SOMA, essendo, essi stessi, dei demoni. Di solito però saranno avviati dai Plan Visitor Agent, attraverso il servizio descritto in questo paragrafo.
Nel caso invece dei Client, questi vengono creati da un PlanVisitor Agent, che li registra presso il manager per consentire che questi siano poi recuperati da chi ne ha necessità (in genere un ClientAgent). La chiave per il recupero del Client è rappresentata dall'identificatore del ClientAgent per il quale il client è stato istanziato, che corrisponde, in SOMA, all'AgentID di tale agente.

3.2.4.4 Schema per la gestione delle risorse.

La richiesta di risorse, un oggetto IResourceRequest, contiene le richieste di CPU e banda. In risposta ad una richiesta di risorse il gestore delle risorse risponde ritornando, al richiedente, un oggetto IResourceResponce. Le risorse delle diverse entità che partecipano all'architettura distribuita sono poi gestite dai QoSManager.
La IResourceResponce contiene informazioni circa la riuscita della prenotazione. Potrebbe essere una risposta affermativa, in caso siano disponili le risorse, negativa, se non ci sono risorse disponibili, oppure positiva con riserva, se il gestore è stato informato di un prossimo rilascio di risorse. In questo caso cioè la risorsa viene per così dire "anticipatamente prenotata", e solo quando è realmente disponibile ciò viene comunicato al QoSManager che aveva effettuato la prenotazione. Se il QoSManager decide di non attendere la liberazione delle risorse dovrà disdire la propria prenotazione anticipata. Internamente il servizio manterrà perciò una tabella che per ogni QoSManager memorizzerà l'ammontare di risorse richiesto. Al termine dello streaming è dovere del QoSManager procedere alla liberazione delle risorse, che avverrà in due fasi: dapprima viene comunicata una prossima liberazione delle risorse. Poi, al termine vero e proprio dello streaming, le risorse vengono effettivamente liberate. Naturalmente è possibile anche effettuare l'immediato rilascio delle risorse, se non si è in grado di sapere

quando di sta per terminare. Il sottosistema di gestione delle risorse permette inoltre ai QoSManager di registrarsi per ottenere informazioni riguardo l'utilizzo della CPU sul nodo corrente, mediante la notifica di eventi di monitoring.
Il sottosistema di gestione delle risorse è costituito da un unico oggetto presente in ogni Place che prende il nome di IResource-Broker . Gli identificatori utilizzati per la prenotazione e il rilascio delle risorse devono essere unici per il singolo broker, e vengono perciò utilizzati, a tale scopo, gli AgentID dei ClientAgent.

3.2.4.5 Architettura delle entità che effettuano lo streaming.

Le entità che gestiranno effettivamente gli stream, come già detto, sono client, proxy e server. Ognuno di questi oggetti può gestire uno o più flussi, agendo a livello di sessione; la struttura è inoltre simile per tutti ed è un aggregato dei seguenti oggetti:
1) Gestore della sessione;
2) Uno o più agenti di gestione dei singoli flussi;
3) Un protocollo per gestire il controllo di sessione;
4) Un manager della qualità di servizio;

In figura 3.5 riportiamo l'architettura di un generico client. Le architetture del proxy e del server sono simili a quella del client.
L'unica differenza sta nel fatto che, mentre gli *Streaming Agent* del client dovranno gestire solo flussi in ingresso, quelli del server dovranno gestire unicamente flussi in uscita, e quelli del proxy dovranno riportare in uscita tutti i flussi che arrivano in ingresso. Il *QoS manager* interagisce con il *Resource Broker* presente sui vari place, per la prenotazione delle risorse, inoltre monitora la situazione dello streaming osservando gli streaming agent. Il resource broker può poi notificare al QoS manager eventuali problemi dovuti all'eccessiva occupazione di CPU, che potrebbero richiedere un riconfigurazione della sessione. Da ultimo le *Protocol Unit* incapsulano i protocolli di comunicazione fra le varie entità distribuite.

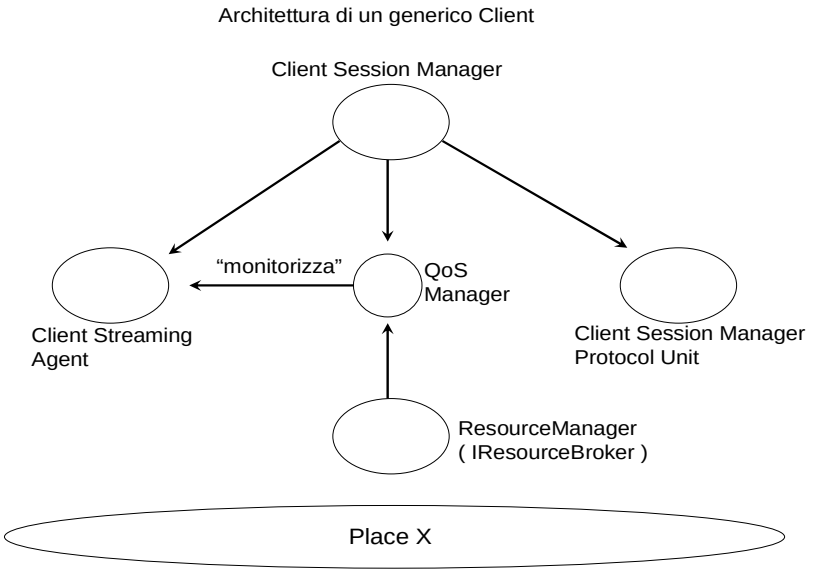


Fig. 3.5

L'insieme dei componenti è poi creato, tramite la factory messa a disposizione dal middleware, dal PlanVisitorAgent che ne conosce esclusivamente le interfacce. La definizione delle componenti tramite le loro interfacce è infatti una delle caratteristiche del middleware; ciò consente di effettuare facilmente, in caso di necessità, la modifica del codice e delle funzionalità dei componenti senza che l'architettura generale debba essere modificata.

3.2.4.6 Servizio di inizializzazione e riconfigurazione dinamica del sistema

Il servizio di inizializzazione impone un modello per il collegamento fra le varie entità, e impone che in tali entità siano implementati alcuni metodi, lasciando al progettista dell'applicativo il compito di concentrarsi sulla realizzazione delle entità stesse, e sollevandolo dal compito di progettare un sottosistema per la distribuzione e l'inizializzazione di tali componenti.
Le varie entità come già detto saranno distribuite su un Path che parte dal cliente e termina con un server. Tutte le entità introdotte agiranno a livello di sessione, gestendo tutti i flussi appartenenti a

una certa sessione. L'inizializzazione sarà portata a termine da agenti mobili detti PlanVisitorAgent. Alla ricezione di una richiesta infatti il ClientAgent, cioè un agente attivato dall'utente, richiede l'inizializzazione del sistema indicando il proprio DecisionMaker (il *decisore* introdotto nei requisiti al punto 3.2.3.3), da utilizzare e passando la richiesta dell'utente (il titolo richiesto). Fatto ciò attende, la notifica di avvenuta inizializzazione.
Il servizio di inizializzazione spedisce lungo il percorso i Plan VisitorAgent, passando ad essi un Plan, che contiene le informazioni riguardanti i componenti da scaricare e inizializzare lungo il percorso. Per l'interpretazione dei Plan, anche per permettere future estensioni, si è fatto uso del pattern Visitor [VisPat]. I Visitor Agent si coordinano tra loro, quando cioè il server è stato inizializzato le informazioni riguardanti l'end-point del server vengono passate indietro agli altri visitor agent sul percorso che possono inizializzare le altre entità fino al client, come visualizzato dal collaboration diagramma in figura 3.6; è da notare che, questo stesso meccanismo verrà poi riutilizzato per quanto riguarda la riconfigurazione del sistema.
Il servizio di inizializzazione provvede, poi, anche ad effettuare un eventuale fase di negoziazione iniziale della QoS e prenotazione delle risorse. Appena giunto in una nuova località infatti il Visitor Agent, per prima cosa, vede se ci sono risorse sufficienti per la presentazione richiesta. In caso affermativo continua il processo di inizializzazione inviando un altro Plan Visitor Agent sul prossimo place, e se ci sono sufficienti risorse su tutto il Path, il processo di inizializzazione termina con successo, come mostrato in figura 3.6. Se invece in un certo place non sono disponibili sufficienti risorse, il Visitor Agent consulta il proprio piano di inizializzazione per vedere se esistono presentazioni alternative a qualità più bassa e tenta di istanziare il percorso per tali presentazioni.

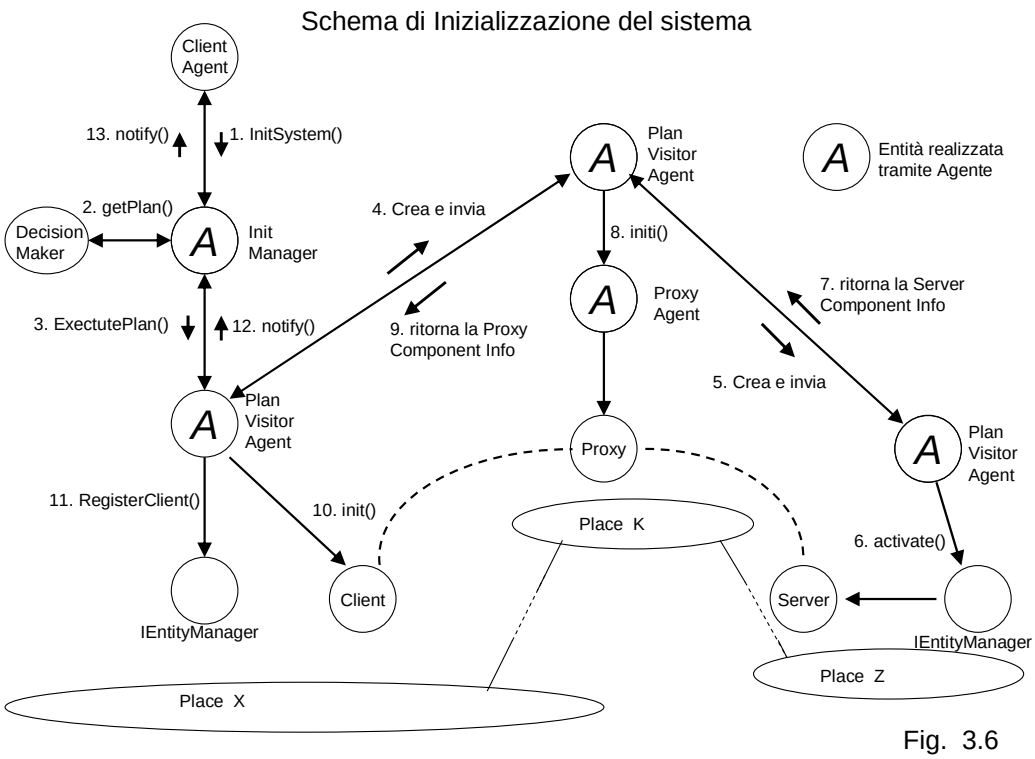


Fig. 3.6

In caso non si riesca a inizializzare il sistema per nessuna delle presentazioni incluse nel Plan ciò viene notificato all'indietro, fino al Client Agent.
Elemento fondamentale dei Plan, durante la fase di Inizializzazione del percorso, sono le **PlanEntry**. Ogni PlanEntry è un oggetto che mantiene le informazioni relative, per ogni Place, alle entità che il VisitorAgent deve attivare; in particolare, queste vengono identificate tramite una tupla del tipo: [packageName, interfaceName].

3.2.4.7 Sottosistema di gestione della qualità di servizio

Per la gestione della qualità di servizio vengono fissate N classi di qualità di servizio. Quando l'utente richiede un certo titolo, vengono richieste tutte le presentazioni multimediali che corrispondono a

tale titolo, si procede quindi ad una analisi delle stesse, in modo da costruire dei Plan per l'inizializzazione e la riorganizzazione del sistema in caso di degrado delle risorse. I criteri adottati sono i seguenti :

1) Fra le presentazioni per ogni classe vengono ricercate le presentazioni che si trovano più vicine al client, in termini dell'astrazione di località introdotta.
2) A questo punto rimane da decidere come disporre le entità lungo il percorso. La scelta fatta viene effettuata tenendo conto delle esigenze, delle richieste e del tipo di client. Tenendo presente che per ogni piano, nel primo nodo andrà istanziato un client e, sull'ultimo, un server.

Le due entità per la gestione delle risorse che introduciamo sono il QoS Manager ed il CPU Monitor. Il CPUMonitor realizza il monitoraggio della CPU e, nel caso in cui l'utilizzo della CPU nell'ultimo intervallo considerato superi una certa soglia, lancia un evento alla volta dei QoSManager, i quali, in risposta a tale evento, cercheranno di passare ad una presentazione a qualità più bassa.
Il CPUMonitor è un Thread che, con una prefissata frequenza, monitora la situazione. Tale oggetto dovrà inoltre rendere disponibili metodi per l'aggiunta e la rimozione di Listener ai quali, come visto sopra, verrà notificata la situazione critica di utilizzo della CPU. In particolare vengono definite alcune soglie, al superamento di ciascuna delle quali viene notificato l'evento con l'indicazione della soglia superata. Ognuna delle entità partecipanti all'architettura avrà poi un proprio QoSManager che è incaricato della gestione della QoS a tempo di esecuzione. Oltre a ricevere gli eventi relativi all'utilizzo della CPU dal CPUMonitor realizza esso stesso il monitoraggio della banda trasmissiva.
Dipendentemente dalle informazioni ricevute relativamente a banda e CPU il QoS Manager può decidere di richiedere una fase di riconfigurazione del sistema. La rilevazione di una situazione critica può riguardare il Client, così come il Proxy o anche il Server; poiché, inoltre, si vogliono evitare conflitti tra due entità,

di uno stesso flusso, che richiedano contemporaneamente una riconfigurazione del sistema, viene stabilito che, se il QoSManager comunica la necessità di una configurazione al proprio ProxyAgent quest'ultimo, per prima cosa, richieda al ClientAgent il permesso di iniziare una fase di riconfigurazione. Il ClientAgent risponde in modo affermativo, se non è già stata fatta una simile richiesta, oppure negativamente, nel caso in cui tale richiesta sia già stata fatta, nel cui caso, abbandona il suo intento. Il ProxyAgent che riceve risposta affermativa, può fare partire la fase di riconfigurazione. Anticipiamo qui che, nell'ambito del nostro progetto, questo meccanismo per l'avvio della fase di riconfigurazione verrà modificato e integrato in uno schema più generale e flessibile.

3.2.4.8 Schema di supporto per la mobilità utente

Lo schema previsto per la mobilità utente è basato sul sottosistema di inizializzazione e riconfigurazione. Quando l'utente decide di muoversi un nuovo PlanVisitorAgent viene spedito sulla nuova località dove istanzia un nuovo Client, inizializzandolo in modo che venga collegato alla seconda entità presente sul Path già stabilito e, a inizializzazione finita, richiama l'agente del Client presente sul vecchio Place. Il protocollo previsto comporta, in particolare, le seguenti fasi :

1) il ClientAgent, dopo aver richiesto all'utente il place verso il quale si vuole muovere, ed aver ricavato il Path dallo User Profile, richiede poi l'inizio dell'hand-off al sottosistema, indicando il Path che localizza il Place verso il quale l'utente si vuole muovere e l'AgentID dell'ultimo Proxy nel percorso;
2) viene quindi creato un PlanVisitorAgent a cui è passato il Plan per la configurazione del percorso verso il nuovo place. In particolare, in questo piano sono contenute le informazioni relative al Client da istanziare e al primo ProxyAgent nel percorso;
3) all'atto dell'inizializzazione sul Target Place, il PlanVisitor Agent contatta il ProxyAgent indicato nel Path, inizializza la

sessione fra il Proxy e il nuovo Client e ritorna un messaggio per notificare l'avvenuta inizializzazione;

4) il sottosistema notifica la terminazione dell'hand-off della sessione stimolando il movimento del ClientAgent alla volta del Target Place;
5) giunto sul nuovo Place, utilizzando il servizio di attivazione delle entità multimediali, l' agente dell'utente si procura il riferimento al nuovo client.

3.2.4.9 Servizi di supporto alla mobilità del terminale

La realizzazione della mobilità del terminale viene supportata nel middleware da un insieme di servizi; in particolare, uno notifica l'ingresso in una nuova località, e l'altro interroga il Target Localizer (il target localizer è l'entità che ci consente di scoprire la locazione della macchina che, nella località obiettivo verso cui ci si sta muovendo, può accettare l'agente mobile proxy che migrerà).
All'ingresso in una nuova località verrà notificato un evento a tutti i listener registrati presso il TerminalMobilityManager (componente che sarà installato in tutti i Place wireless). L'evento contiene informazioni riguardanti i vari Place BSS (Basic Service Set) sui quali è stato previsto che si potrebbe saltare, riportando per ciascuno di essi il Path che li identifica univocamente.
In particolare, gli eventi che si possono generare sono i seguenti :

- *Nessun evento*: questo è il caso in cui il terminale non si è spostato tanto da richiedere una notifica del fatto;
- *Potenziale ingresso* in un nuovo BSS: questo evento viene lanciato quando si pensa che il terminale stia entrando in un nuovo BSS. In questo caso l'evento viene notificato a tutti i ClientAgent, che possono perciò iniziare l'inizializzazione del Proxy sul/sui Place indicati.
- *Ingresso avvenuto* in un nuovo BSS: questo evento viene lanciato quando, verificando il BSS identifier del terminale, si vede che effettivamente la stazione risulta associata ad un nuovo BSS.

- *Ingresso non avvenuto*: questo evento viene lanciato quando l'ingresso che ci si aspettava in un nuovo BSS non è avvenuto.

Nelle ipotesi per la gestione dell'hand-off di sessione è che non ci siano disconnessioni per quanto riguarda la fruizione del servizio sulla rete wireless.

3.2.4.10 Schema per la mobilità del terminale

Lo schema elaborato per la mobilità del terminale viene costruito sfruttando i servizi presentati sopra. Nel caso della mobilità utente, l'entità che veniva inizializzata nel target place era il Client, questa volta invece sarà il primo Proxy che si trova sul Path dal client al server. Inizialmente viene notificato ai vari Client un *potenziale ingresso* in una nuova località, poi se effettivamente si entra nella nuova località, verrà notificato un *ingresso nella nuova località*. I Client possono perciò, alla ricezione del primo evento, cominciare l'inizializzazione del Proxy sui possibili place remoti, inviando su tali Place un Plan Visitor Agent. Una volta terminata l'inizializzazione, se effettivamente il terminale è entrato nella nuova località, verrà fatta partire la riconfigurazione dell'ultima parte del Path, stimolando il salto dell'ultimo ProxyAgent dal vecchio Place Wireless Enabled al PlaceWirelessEnable target, dove attraverso il servizio di attivazione delle entità multimediali (vedi 3.2.4.3) potrà recuperare il riferimento al Proxy già istanziato e inizializzato.
Se creato il percorso alternativo viene notificato infine il segnale di ingresso non avvenuto, i client dovranno avvisare i Visitor Agent che, a loro volta, dovranno, occuparsi di eliminare i componenti non più necessari.

3.3 Conclusione.

In questo capitolo abbiamo introdotto gli strumenti che verranno utilizzati nella realizzazione del nostro progetto. Per prima cosa abbiamo descritto SOMA: la piattaforma su cui il nostro sistema di località è stato realizzato; nella seconda parte del capitolo abbiamo invece introdotto MUM : un middleware realizzato su SOMA e progettato per fornire un supporto per la programmazione di applicativi per il delivery di flussi multimediali in un ambiente distribuito.

Nei prossimi capitoli, vedremo, infine come, il sistema di località sarà effettivamente realizzato su SOMA; vedremo anche quale modello di riconfigurazione intendiamo realizzare per il middleware MUM e come questo potrà essere utilizzato per la gestione di un caso pratico di riconfigurazione del percorso di un flusso multimediale.

CAPITOLO 4.

4. Realizzazione del sistema di località in SOMA.

In questo capitolo intendiamo descrivere come, il modello per un sistema di località, introdotto nel secondo capitolo, è stato realizzato sulla piattaforma SOMA, presentata nel capitolo precedente.
Come abbiamo visto, SOMA offerta già un'astrazione di località che chiama domini. Tali domini vengono poi organizzati in una struttura ad albero per consentire un'organizzazione efficiente del sistema. Tuttavia questa scelta rende necessaria la sovrapposizione di un nuovo sistema di località, all'astrazione presente in SOMA, per consentire un'estensione alla visibilità delle piattaforme del sistema. Il nostro sistema di località è stato poi pensato come concettualmente distino dai sistemi di nomi presenti; la motivazione di ciò, insieme alla descrizione delle modifiche della piattaforma è oggetto della prima parte del capitolo. Nella seconda parte invece verranno presentati i dettagli di realizzazione del sistema di località vero e proprio.

4.1 Motivazioni per una una nuova entità come gestore delle località.

Dal punto di vista realizzativo, avremmo potuto impostare il sistema di località, come un estensione dei sistemi di nomi di SOMA; si è invece optato per la creazione di un entità distinta, il cui scopo fosse oltre alla creazione e alla gestione delle strutture destinate a mantenere le informazioni sulle località, la gestione dei meccanismi di registrazione e accesso a tali informazioni, e l'integrazione con i sistemi di nomi già presenti.
Questa scelta è stata effettuata tenendo in considerazione diverse considerazioni :

- il sistema di località non è un semplice sistema di nomi; il suo scopo, in questo progetto, è quello di consentire un riadattamento run-time dei percorsi degli stream multimediali. Se prendiamo in considerazione le informazioni presenti in un semplice sistema di nomi, tale obiettivo non è raggiun-

gibile in quanto non vi vengono registrate indicazioni sullo stato delle risorse, informazioni invece importanti per una riconfigurazione efficacie.

- l'entità che implementeremo dovrà essere in grado di gestire anche i rapporti con le altre località stesse. Se possiamo pensare, infatti, le località come entità distinte e concettualmente indipendenti, a livello dei servizi offerti; a livello realizzativo l'entità che gestisce una determinata località dovrà essere a conoscenza anche di informazioni relative alle località adiacenti e con esse dovrà coordinarsi per la realizzazione di alcuni servizi come la creazione di una nuova località, il recovery della topologia, ecc..
- Una ulteriore differenza, tra la realizzazione dei sistemi di nomi di SOMA e il gestore delle località, è la seguente: mentre i sistemi di nomi sono localmente presenti nei place, per quanto riguarda la tabella che mantiene le informazioni relative alla località, si è deciso di mantenerla esclusivamente sui DefultPlace e si è poi realizzato un meccanismo di consultazione di tipo client-server tra i Place normali e i DefaultPlace. Questa scelta è stata adottata per poter garantire una minore occupazione di risorse nei Place normali e mobili del sistema che spesso saranno dispositivi dalle risorse limitate.

Sulla base di queste considerazioni si è quindi deciso di creare un sistema che sia indipendente dai preesistenti sistemi di nomi ma che con essi interagisca comunque in maniera efficace.

4.2 Modifiche alla piattaforma SOMA.

Prima di procedere nei dettagli dell'implementazione del sistema di località, introduciamo quelle che sono state le principali modifiche della piattaforma SOMA; tali modifiche riguardano oltre ad alcune entità, anche il meccanismo di registrazione dei Default Place.

4.2.1 La classe Environment.

Come abbiamo visto, l'Environment rappresenta l'accesso ai servizi di un Place generico in SOMA; per tale motivo, esso dovrà fornire anche l'accesso ai servizi di località. Alla creazione di un environment verrà quindi creato anche un'entità che sarà il manager di località. Questo implementerà un interfaccia chiamata *LocalityService*. Nell'environment vengono poi definiti le costanti fondamentali per la creazione della località stessa quali il suo raggio e la politica di aggiornamento. Devono essere infine, implementati i metodi che consentano di ottenere le informazioni sullo stato iniziale delle risorse della postazione in cui risiede il Place, e sulle soglie di allarme per tali risorse. Oltre a ciò infine viene definito un metodo per ottenere un marcatore temporale da associare allo stato delle risorse ogni qual volta questo viene letto.

4.2.2 Modifiche per il DomaniNameService.

Il servizio che ha subito la maggior parte delle modifiche in SOMA è il DNS. In particolare, come abbiamo visto, esso rappresenta il servizio di nomi per i DefaultPlace presenti nel sistema. Gli oggetti memorizzati sono DomainInfo che rappresentano un indirizzo di rete(indirizzo ip + porta) di un DefaultPlace del sistema. Nell'ambito del nostro progetto, tali oggetti sono stati modificati comprendendo altre informazioni comunque compatibili con la struttura di un servizio di nomi. Per quanto riguarda le DomainInfo in particolare, sono stati aggiunti i seguenti campi :

1) un oggetto che indica la posizione del DefaultPlace dalla radice,
2) la distanza del DefaultPlace dal Place in cui è mantenuta la DomainInfo,
3) l'indicazione se, su tale DefaultPlace, il servizio di località è attivo o meno.

L'insieme di queste informazioni consente una gestione dei protocolli di creazione e recovery del sistema più agevole senza modificare la natura del servizio del DNS, tuttavia, per ottenere tale estensione nelle DomainInfo, è stato necessario modificare il mec-

cansimo di registrazione dei DefaultPlace. Inoltre, durante la riconfigurazione della topologia, tali informazioni cambiano e deve quindi essere previsto un meccanismo di modifica.
Vediamo ora in cosa i meccanismi del DNS si differenziano dalla versione precedente.

4.2.2.1 La registrazione di un DefaultPlace nel DNS.

La registrazione di un nuovo DefaultPlace nei DNS del sistema avviene tramite l'uso dei Command; in particolare del DomainRegisterCommand. Quando un nuovo DefaultPlace viene creato, questo invia tale comando al DefaultPlace padre. Una volta a destinazione, tale comando registra la DomainInfo del place appena creato e invia ai DefaultPlace figli e padre, il comando PutDomainCommand con la DomainInfo ricevuta, dopodichè ritorna al nuovo DefaultPlace il comando DomainRefreshCommand (DRC) che possiede una copia della tabella del DNS. I comandi PutDomainCommand invece, quando giungono su un DefaultPlace, controllano se qui è già presente la DomainInfo che portano con se; se è presente terminano mentre se non lo è, la aggiungono e nuovamente viene inviato un PutDomainCommand (PDC) a tutti i DefaultPlace confinanti (padre e figli).
In questo modo la DomaniInfo del nuovo DefaultPlace si propaga nella topologia fino a tutti i DefaultPlace.
La modifica che abbiamo apportato, in questo meccanismo, riguarda il DomainRegisterCommand. Quando un DefaultPlace viene creato, non conosce da subito il percorso che lo congiunge al DefaultPlace radice; in particolare, lo otterrà quando il DRC (DomainRegisterCommand) arriverà sul nodo padre che, essendo già nella topologia, è già a conoscenza di tale informazione.
Calcolato il percorso tra il nodo radice e il nuovo DefaultPlace, il DRC ottiene una copia della tabella del DNS con le distanze dei DefaultPlace incrementate di 1, registra quindi la DomainInfo del DefaultPlace, invia i PDC agli altri DefaultPlace e il DRC con la tabella modificata, al nuovo iscritto.

La propagazione della DomainInfo avviene poi nel modo precedentemente descritto tuttavia, a ogni passo, la distanza indicata dalla DomainInfo viene incrementata di un'unità.

4.2.2.2 La fase di Recovery e le modifiche al DNS.

Nel caso di un cambiamento della topologia causato dalla caduta di un DefaultPlace, le informazioni delle DomainInfo devono essere modificate.
La versione precedente della piattaforma SOMA, non offriva un supporto per la riconfigurazione della topologia e anche se era prevista la possibilità di rimuovere una DomainInfo dalla tabella del DNS, non esistevano meccanismi che consentissero di verificare la caduta di un DefaultPlace.
Attualmente tali meccanismi saranno sviluppati grazie al sistema di località realizzato che, nel caso, attiverà la modifica del DNS specificando quale DefaultPlace è caduto.
In questo caso, le operazioni svolte dal DNS sono le seguenti :

1) rimuove la DomainInfo corrispondente e da questa ottenere il percorso tra tale DefaultPlace e la radice.
2) individua la posizione del DefaultPlace caduto rispetto alla propria; in particolare può accadere che :
   a) il DefaultPlace caduto sia in un sottoramo di questo DefaultPlace,
   b) questo DefaultPlace sia in un sottoramo del DefaultPlace caduto,
   c) Il DefaultPlace caduto e questo sono su rami diversi.
3) individua le DomainInfo di tutti i DefaultPlace per cui il percorso verso la radice è cambiato (era presente nel percorso il Place caduto). Per tali Place viene modificato il percorso ed eventualmente la distanza se è cambiata (i DefaultPlace per cui cambia la distanza dipendono dalla posizione del DefaultPlace caduto rispetto a questo).
4) Vengono aggiornate, eventualmente le informazioni relative al proprio percorso, al proprio DefaultPlace padre e ai propri DefaultPlace figli.

Infine la riconfigurazione del DNS termina.

4.2.3 Il NetworkManager.

Le modifiche relative al NetworkManager riguardano la possibilità di evitare il meccanismo standard di comunicazione con Place normali al di fuori del proprio dominio. Come abbiamo visto, in questi casi, veniva utilizzato un passaggio intermedio che consisteva nell'utilizzo di un TransportCommand verso il Default Place gestore del dominio in cui il Place normale si trovava. Grazie alla creazione della località, esiste ora una visibilità diretta dei Place normali che si trovano nella località; per questo motivo, prima di ricorrere al TransportCommand, viene ora controllato se il Place di destinazione è visto nella Località, e nel caso, viene inviato direttamente il comando.

4.3 Implementazione del sistema di località.

4.3.1 Centratura e raggio delle località.

La prima cosa di cui ci siamo preoccupati, è stato stabilire quanto saranno grandi le località; considerando la distanza in termini di passi lungo la topologia di SOMA, abbiamo che appartengono ad una stessa località, centrata su un elemento della topologia stessa, tutti i Place e i DefaultPlace che sono raggiungibili in un numero di passi pari o inferiore al raggio della località. Per tale raggio, si sono provati valori da 2 a 6 passi. Per mantenere dimensioni accettabili delle località, ed essere in questo modo anche più omogenei con il concetto tradizionale di località, è comunque necessario limitare tale valore.
Considerando un raggio pari a tre, i Place che entreranno a far parte di una località centrata su un DefaultPlace X saranno, adottando una terminologia genealogica :

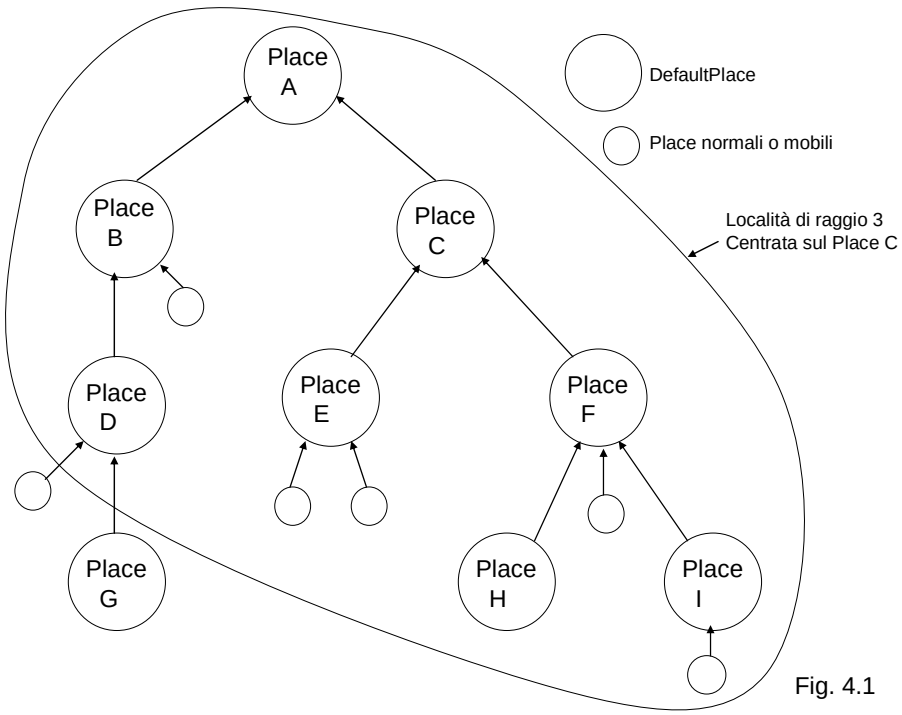


Fig. 4.1

A scendere da X :
1) i Place normali del dominio di X (distanza 1).
2) i DefaultPlace figli di X (distanza 1).
3) i Place normali del dominio dei DefaultPlace figli di X (distanza 2).
4) i DefaultPlace nipoti di X (distanza 2).
5) i Place normali del dominio dei DefaultPlace nipoti di X (distanza 3) .
6) i DefaultPlace figli dei nipoti di X (distanza 3).

A salire da X :
1) il DefaultPlace padre di X (distanza 1).
2) i Place normali registrati nel DefaultPlace padre di X (distanza 2).
3) i DefaultPlace fratelli di X (distanza 2).
4) il DefaultPlace nonno di X (distanza 2).
5) i Place normali del dominio dei fratelli di X (distanza 3).

6) i DefaultPlace registrati presso i fratelli di X (distanza 3).
7) i Place normali registrati presso il nonno di X (distanza 3).
8) i DefaultPlace registrati presso il nonno di X (distanza 3).

Una dimensione di 3 per il raggio delle località è stata quindi ritenuta, in generale, sufficiente (vedi fig. 4.1).
Il secondo aspetto di cui ci siamo preoccupati è stato stabilire quante località avere nel sistema e centrate su quali nodi.
La prima ipotesi che possiamo fare è creare una località per ogni elemento della topologia, quindi sia per Place normali che per DefaultPlace. In questa ipotesi tuttavia avremmo avuto un forte overhead e un eccessivo utilizzo delle risorse di rete, per gestire gli eventuali aggiornamenti e comunicazioni di servizio tra le varie località.
Una seconda ipotesi è invece quella di creare delle località ogni intervallo predefinito di passi; in questo modo per esempio, scegliendo di creare una località di R passi centrata su nodi a 2R passi, potremmo avere località non sovrapposte. Ovviamente in questo caso, ogni Place o DefaultPlace apparterrebbe in maniera esclusiva a una località, e tutti i Place della località avrebbero la stessa visibilità, diversamente dal caso sopra. In questo caso tuttavia, la visibilità dei membri della località sarebbe fortemente orientata (a meno di non essere sul nodo su cui la località è centrata) in un verso o nell'altro della topologia. Oltre a ciò, nel caso che si aggiungessero al sistema nuovi nodi al di fuori della località, si dovrebbero creare nuove località, di cui sarebbero l'unico elemento, vanificando quindi l'obiettivo di ottenere maggiore visibilità.
La terza ipotesi, che è quella che abbiamo adottato, consiste nell'utilizzare località sovrapposte centrate solo sui DefaultPlace.
Questo in quanto, si può ragionevolmente pensare che Place normali e i rispettivi DefaultPlace, siano già tra loro "locali", e possano quindi avere la stessa località.
Un'osservazione che vale la pena di fare è che in questo modo, si ottiene una visibilità asimmetrica nel sistema. Supponendo di avere

un raggio di località pari a due; in questo caso, i Place normali hanno visibilità sui DefaultPlace distanti due passi dal DefaultPlace X cui sono registrati; tali DefaultPlace (cioè quelli a distanza due) non hanno tuttavia visibilità di questi Place normali in quanto le loro località hanno visibilità solo fino al DefaultPlace X. Questo tuttavia non pare essere un inconveniente, in quanto comunque avvantaggia i Place normali che si presume siano su macchine con risorse in genere limitate e quindi con maggiore necessità di una visibilità ampia e su macchine (quali possono essere quelle che ospitano i DefaultPlace) dalle risorse maggiori.

4.3.2 I servizi di una località.

Per la creazione e gestione delle località dovranno esistere su ogni nodo diverse entità che collaboreranno per fornire i servizi necessari; in particolare, tali servizi verranno raccolti tramite un oggetto che costituirà il front end per la località nel sito considerato.

L'insieme dei servizi che verranno messi a disposizione, è concettualmente divisibile in due gruppi (fig. 4.2); il primo gruppo è composto da tutti i servizi che la località dovrà rendere disponibile alle applicazioni sovrastanti, e cioè in particolare, i servizi di consultazione delle informazioni sulla risorse nei nodi della località stessa; il secondo è invece composto dall'insieme di funzionalità e servizi necessari per quelle attività che non sono visibili alle applicazioni ma che sono tuttavia necessarie al corretto funzionamento del sistema di Località. In particolare queste attività possono essere distinte:

1) servizio di creazione di una località
2) servizio per il reperimento delle informazioni
3) servizio per il recovery del sistema di località.

Nei prossimi paragrafi entreremo più nel dettaglio relativamente ai diversi servizi offerti.

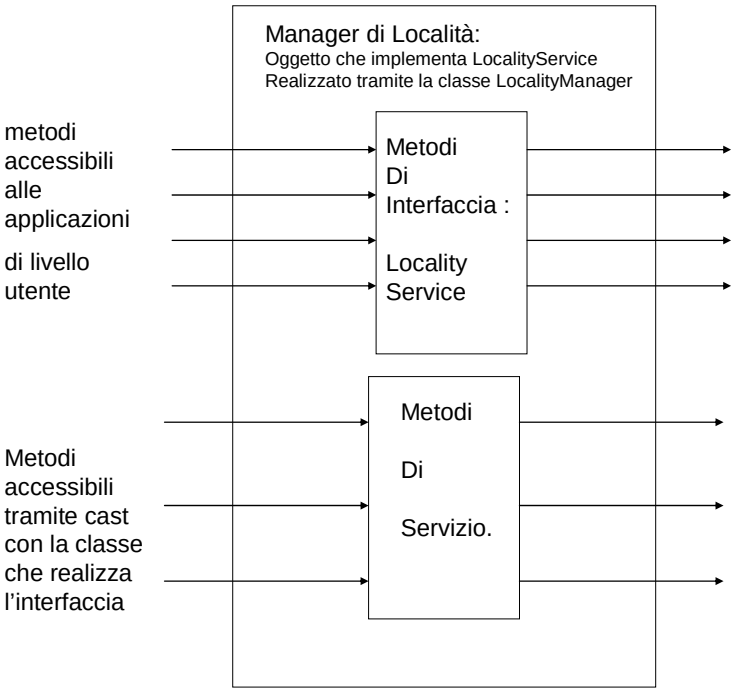


Fig. 4.2

4.3.3 Le entità che compongono una località.

L'entità che si occupa della gestione della località e ne rappresenta, per le applicazioni, il front-end, è il manager di località (*Locality Manager)*. Il manager di località gestisce e mantiene fisicamente le informazioni relative allo stato delle risorse delle postazioni della località; in particolare, una tabella di località (*Locality Table*) in cui, per ogni elemento della località esiste una entry corrispondente (*LocalityTableEntry*).

Oltre alle informazioni sullo stato delle piattaforme della Località, il manager di località, mantiene le informazioni relative alle località la cui estensione è tale da estendersi fino a comprendere il DefaultPlace in cui ci si trova (in sostanza tutte le località che lo vedono). L'esatta struttura di questi oggetti sarà definita nei prossimi paragrafi.

All'interno del manager di località sono poi presenti altri tre

manager che si che si occupano della realizzazione dei servizi per la gestione delle località :
1) Manager di Topologia (*Topology Manager*)
2) Manager per l'aggiornamento delle informazioni (*UpdatManager*).
3) Manager per il controllo periodico della topologia. (*AYAManager*).

Il primo si occupa di fornire le informazioni riguardanti la topologia conosciuta del sistema (in genere almeno la parte per cui si estende la località) e gestisce il servizio di recovery (i protocolli di riconfigurazione della topologia nel caso di caduta di un nodo). Il secondo si occupa invece dell'aggiornamento delle informazioni della tabella di località. Il terzo manager, infine, si occupa di verificare che i nodi della topologia presenti nella località siano ancora attivi attraverso un semplice protocollo di tipo Are You Alive; la presenza di questo manager nasce in quanto l'aggiornamento, delle informazioni sullo stato delle risorse nei nodi della località sarà, a default, attuato con una politica di notifica degli eventi.

4.3.4 I servizi dell' interfaccia LocalityService.

Come abbiamo visto, i servizi messi a disposizione ai programmatori di alto livello, sono state raccolti in un'interfaccia che viene messa a disposizione tramite la variabile di ambiente di ogni Place. Il nome dell'interfaccia è LocalityService (e viene implementata dal manager di località) ed è stato scelto in omogeneità con i nomi dei sistemi di nomi esistenti. I servizi messi a disposizione nell'interfaccia, vengono qui specificati, in funzione delle caratteristiche di SOMA :

1. specificato un PlaceID, ottenere l'entry contenente le informazioni relative al Place.
2. ottenere un vettore con le informazioni relative a tutti i Place nella località.
3. ottenere un vettore con tutti i PlaceID degli elementi della

località.

4. ottenere un vettore contenente le entry relative ai Place presenti nello stesso dominio, cioè ai Place specificati nel proprio PNS.
5. ottenere un vettore contenente le entry relative ai Place presenti in un dominio specificato; ovviamente, sono ritornate solo le entry relative ai Place visti nella località.
6. ottenere il numero di elementi presenti nella località.
7. ottenere, specificato il PlaceID, un valore booleano indicate se tale Place è presente nella località.
8. ottenere un vettore con le entry migliori, dei Place nella località, rispetto un parametro specificato.
9. ottenere un vettore con le entry migliori rispetto ad un parametro specificato e ad un range indicato.
10. ottenere un booleano che indica se sul place, il manager di località che si occupa delle località è stato inizializzato e se è attivo.
11. ottenere un booleano che indica se si è su un DefaultPlace.
12. ottenere il PlaceRange della località in cui ci si trova.
13. ottenere lo stato delle risorse sul Place in cui si risiede.

4.3.5 I servizi di località non accessibili in LocalityService.

A differenza dei servizi resi disponibili tramite l'interfaccia LocalityService che sono necessari per le applicazioni che intendano avvalersi delle informazioni sulla località, le seguenti funzionalità sono necessarie per la gestione dei meccanismi di coordinamento e aggiornamento del sistema di località e, pertanto, non sono resi accessibili alle applicazioni utente.

L'entità che fornisce questi servizi è ottenuta mediante cast con la classe che implementa l'interfaccia LocalityService; inoltre, questi servizi sono accessibili esclusivamente sui DefaultPlace, nei quali ve ne è realmente necessità, e non sui Place normali.

Questi servizi consentono :

1. l'inserimento, la cancellazione o l'aggiornamento di una entry contenente le informazioni relative ad un Place specificato dal

relativo PlaceID.

2. di ottenere un vettore i cui elementi sono PlaceID relativi alle località con cui può interagire, cioè l'insieme di quelle che hanno visibilità su questo DefaultPlace e quelle di cui il Default Place ha visibilità tramite la Località.
3. di aggiungere o rimuovere un PlaceRange relativo ad una località.
4. di ottenere un vettore contenente i PlaceID relativi a località che hanno ancora, da questo Place, un numero specificato come parametro. Specificato il numero di passi P, questo servizio ritorna il PlaceID di località che hanno un raggio R e che si trovano ad una distanza D da questo DefaultPlace tale che $P =< R - D$.
5. di ottenere un vettore con i PlaceID di tutte le località che si trovano entro una distanza specificata come parametro da questo DefaultPlace.
6. di ottenere un vettore con i PlaceID di tutte le località che, nella topologia, si trovano al di sopra del DefaultPlace in cui il servizio viene chiamato. Se viene specificato un numero di passi, il manager di località ritorna i PlaceID relativi alle località superiori la cui distanza è inferiore o uguale al numero di passi specificato.
7. di ottenere un vettore con i PlaceID di tutte le località che si trovano, nella topologia, al di sotto del DefaultPlace in cui il servizio viene chiamato. Anche in questo caso è possibile specificare, come parametro, il numero di passi per cui deve essere presa in considerazione la parte inferiore della topologia.
8. di ottenere un vettore con i PlaceID di tutte le località che si trovano, nella topologia, al di sotto del DefaultPlace in cui il servizio viene chiamato ad esclusione di quelle che si trovano in un particolare ramo della topologia. Anche in questo caso è possibile specificare come parametro un numero di passi per cui esplorare la topologia.
9. di attivare e disattivare il processo di recovery della topologia nel caso di caduta di un Place o di un DefaultPlace.
10. abilitare, disabilitare e settare i range per la politica di notifica

di eventi relativi ad uno specificata risorsa .

11. abilitare e disabilitare il funzionamento del manager stesso.
    In generale, il manager quando viene creato non possiede le informazioni per rispondere a eventuali interrogazioni da parte di utenti ed è quindi disabilitato; una volta registrato e ottenute le informazioni viene attivato. Anche durante una fase di recovery, il manager viene momentaneamente disabilitato per non fornire informazioni errate; viene poi riattivato a recovery terminato.
12. ottenere il riferimento alla variabile Environment del Place in cui ci si trova.
13. bloccare e liberare il manager. Tipicamente durante una fase di transizione, per esempio durante l'iscrizione di nuovi Place nella topologia il manager viene temporaneamente bloccato.
    Con "bloccato" in particolare intendiamo in questo caso, che non possono avvenire in contemporanea altre iscrizioni o altre operazioni che ne modificherebbero lo stato.
14. gestire le comunicazioni tra i manager di località sui Default Place e quelli sui Place normali.

4.3.6 Le strutture dati delle località.

Le informazioni relative allo stato dei Place della località stessa sono mantenute nella tabella chiamata *LocalityTable*. Questa, come anticipato, viene per ragioni di occupazione di risorse, mantenuta esclusivamente nei DefaultPlace e viene consultata tramite un meccanismo di client-server sincrono e bloccante dai manager di località dei Place normali. Ogni manager di località su Default Place implementa quindi un demone/thread in ascolto per possibili richieste. I manager di località su Place normali dovranno invece implementare un ponte tra il sistema locale e il DefaultPlace a cui si è registrati, rimbalzando le richieste sulla località al DefaultPlace (l'assunzione di base, è che le comunicazioni all'interno dello stesso dominio siano facili e poco costose).
La LocalityTable, oltre a mantenere le informazioni relative ai

place nella località, fornisce anche i servizi necessari per la gestione degli stessi. In particolare, implementa le funzionalità richieste dal manager di località per l'accesso, l'aggiornamento e l'aggiunta di informazioni relative a uno o più Place nella località, ecc..
Le entry della LocalityTable sono costituite da oggetti chiamati *LocalityTableEntry* (vedi fig. 4.4.) che mantengono le informazioni relative al Place considerato. Lo stato delle risorse di un Place è contenuto nel *PlaceResourcesStatus* e oltre a questo, per ogni entry vengono mantenute altre informazioni di natura diversa.
Le informazioni del PlaceResourcesStatus, sono infatti, per natura variabili velocemente mentre la parte rimanente di informazioni è di natura statica o dinamica ma con ritmi di variazioni molto più lenti rispetto alle informazioni di stato. In particolare, ogni LocalityTableEntry, contiene il PlaceID del Place relativo, che è la chiave di identificazione della entry nella tabella, la sua Place/DomainInfo e il percorso assoluto o Path che lo identifica univocamente sulla topologia a partire dal nodo root.
Per quanto infine riguarda il PlaceResourcesStatus; lo scopo di questa struttura è mantenere le informazioni relative allo stato delle risorse di un Place. Le risorse prese in considerazione sono già state introdotte nel primo capitolo; poiché però i parametri che possono essere utilizzati nella descrizione delle risorse di un Place possono variare a seconda degli obiettivi che ci si prefigge si è cercato di mantenere per questo oggetto una struttura il più flessibile possibile. Per chiarire, cosa intendiamo facciamo un semplice esempio: è possibile pensare al caso in cui un agente, il cui scopo sia stampare un file, cerchi tra i Place della località quelli che sono dotati di una stampante; in questo caso, è possibile pensare di caratterizzare i Place anche in funzione della presenza o meno di un dispositivo stampante e di quanto questo sia utilizzato.
In sostanza, diverse esigenze, possono portare a diversi parametri di valutazione delle risorse un Place. Si è quindi cercato di creare un oggetto che consentisse di ottenere facilmente un ampliamento dei parametri considerati.

Schema generale per la consultazione delle informazioni di località

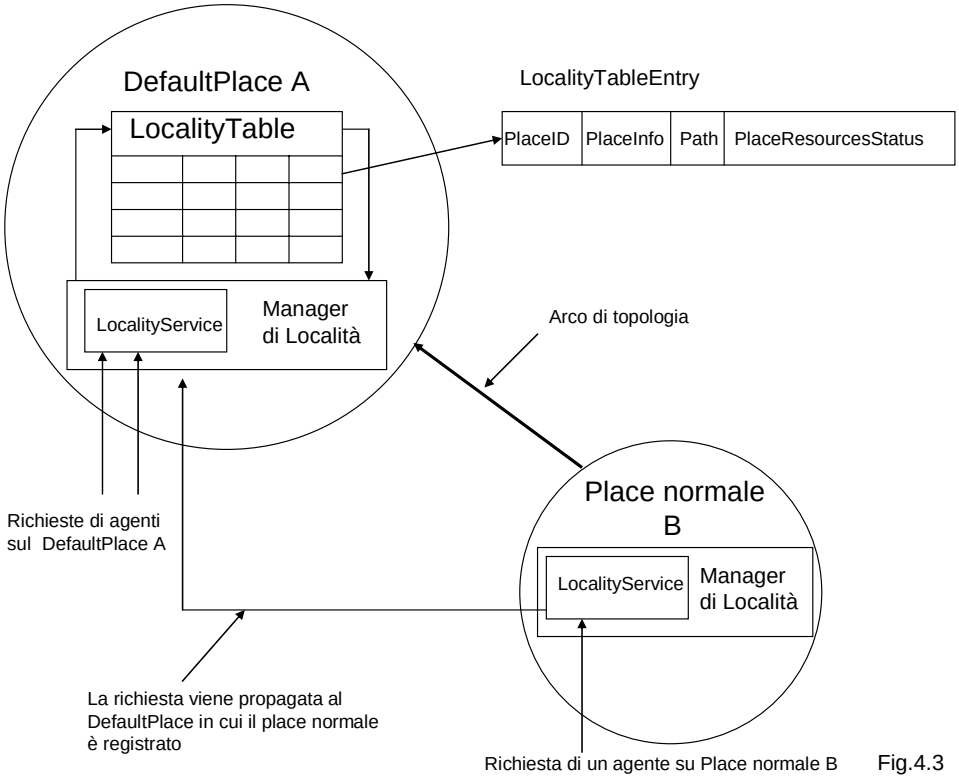


Fig.4.3

Fig. 4.4

Elementi presenti in una LocalityTableEntry.

Oggetto :	Informazione mantenuta :
PlaceID	Identificatore del Place.
Place/DomainInfo	Informazioni sul Place (indirizzo di rete)
Path	Percorso dal nodo root al Place.
PlaceResorcesStatus	Informazioni relative allo stato delle risorse sul Place

Nell'oggetto PlaceResourcesStatus, esiste infatti un metodo unico per l'accesso ai valori dello stato delle risorse; tale metodo, specifi-

cato l'identificatore della risorsa interessata, restituisce un oggetto che mantiene tutte le informazioni necessarie alla caratterizzazione dello stato della risorsa specificata.
Nel caso che si sia specificato l'uso della CPU, l'oggetto ritornato potrà essere semplicemente un intero con il valore percentuale della CPU utilizzata, nel caso invece che si sia interessati allo stato della stampante, l'oggetto restituito potrà essere un record indicante il numero di stampanti accessibili dal nodo e, per ognuna, lo stato. Nell'eventualità, infine, che si indichi un identificatore non conosciuto, l'oggetto ritornato sarà un oggetto nullo.
Oltre a ciò, come fatto notare nel secondo capitolo, è necessaria la presenza in ogni PlaceResourcesStatus di una marca temporale (timeStamp) e di un metadato che descriva la piattaforma a cui il PlaceResourcesStatus si riferisce (PlatformDescriptor) che sono sempre accessibili tramite il metodo getParam( ) (vedi fig. 4.5).
La presenza del metadato è fondamentale in quanto, lo stato delle risorse può essere espresso come un valore normalizzato e cioè riferito alla quantità complessiva della risorsa considerata (per esempio il caso dell'uso della CPU che viene indicato in percentuali); in questo caso, il descrittore della piattaforma consente di traslare il valore percentuale in un valore effettivo qualora ve ne sia la necessità; la presenza del marcatore temporale è invece importante quando si vuole stabilire una relazione temporale tra due "snapshot" dello stato delle risorse (per esempio un'applicazione che viaggia da un nodo ad un altro può raccogliere informazioni di stato sugli stessi elementi della topologia e per sapere quale è la più recente potrà confrontare le marche temporali).
In particolare, il marcatore temporale viene associato allo stato delle risorse ogni volta che viene letto tale stato e verrà, ad ogni lettura, incrementato in modo tale che letture effettuate in tempi successivi abbiano sempre marcatori distinti.
Oltre alle informazioni sullo stato delle piattaforme della località, il manager di località, mantiene anche le informazioni relative alle località la cui estensione è tale da estendersi fino a comprendere il nodo in cui si trova (in sostanza tutte le località che lo vedono).

Per ogni località, le informazioni sono mantenute in un oggetto chiamato *PlaceRange* che possiede le informazioni riguardanti: il nodo su cui la località è centrata, il raggio di tale località e la distanza tra il nodo proprietario di tale località e questo nodo.

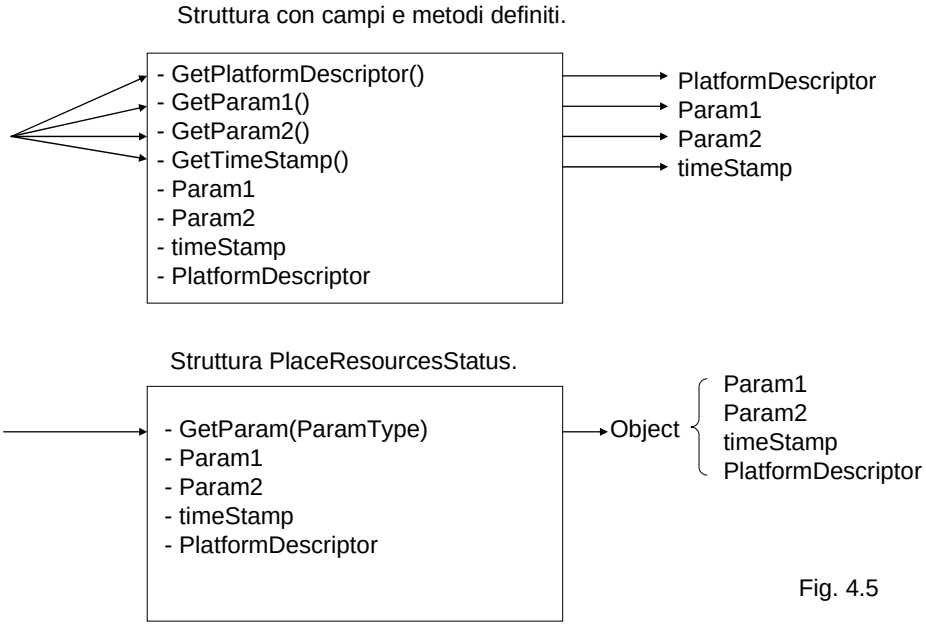


Fig. 4.5

I PlaceRange non sono accessibili al livello delle applicazioni poiché servono esclusivamente per i meccanismi di gestione del sistema di località.

4.3.7 I componenti del manager di località.

Abbiamo visto precedentemente che il manager di località include al suo interno oltre alla tabella di località, altri manager per la gestione di alcuni servizi. In realtà la struttura di tale manager è piuttosto complessa; in particolare, ogni manager di località è composto dalle seguenti entità :

1. la tabella in cui vengono mantenute le informazioni sullo

stato dei Place nella località.

2. un vettore in cui vengono mantenute le informazioni, cioè i PlaceRange, relativi alle diverse località in relazione con la località gestita dal manager.
3. la parte di accesso alla tabella dei Place della località. Tale parte è realizzata direttamente nel manager di località ed è in sostanza quella accessibile dall'interfaccia LocalityService.
4. un manager che si occupa, di aggiornare lo stato delle risorse dei Place nel dominio. Tale manager chiamato UpdateManager, è attivo solo sui DefaultPlace.
5. un manager che attua un meccanismo di controllo del tipo Are You Alive nei confronti degli altri DefaultPlace presenti nella località. Tale manager è chiamato AYAManager.
6. un thread chiamato answerThread, presente solo sui DefaultPlace, il cui scopo è quello di rispondere alle interrogazioni provenienti dai Place normali o mobili del dominio del DefaultPlace. Questo thread, in sostanza, implementa un server multi thread; quando giunge una richiesta da un Place del domino, crea un thread che soddisfi la richiesta e si rimette in attesa (per lo schema di funzionamento vedi fig. 4.3).
7. un componente chiamato requestSender, presente solo sui Place normali o mobili, il cui scopo è quello di intercettare le chiamate al manager di località e propagarle al Default Place a cui si è registrati. Il requestSender è l'oggetto che comunica con l'answerThread, di cui conoscerà quindi indirizzo e porta (vedi fig. 4.3). La comunicazione è sincrona e bloccante. Bisogna quindi tener conto che ogni agente che utilizzi i servizi di località non direttamente rimane in attesa per un periodo non determinato.
8. un manager il cui scopo è quello di gestire informazioni riguardanti la topologia del sistema; tale manager che ha nome TopologyManager è attivo esclusivamente sui Default Place.

4.3.8 I meccanismi di Inizializzazione e Registrazione .

Il LocalityManager viene creato all'atto della creazione del Place; quando questo si registra nella topologia, avvengono anche le fasi di inizializzazione e registrazione. In figura 4.6 e4.7 vediamo una rappresentazione di tali fasi.

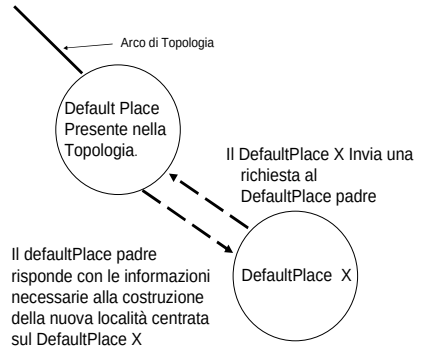


Fig 4.6

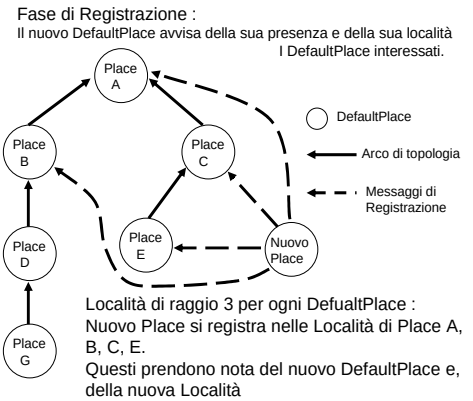


Fig 4.7

Per fase di inizializzazione, intendiamo la fase in cui il nuovo nodo ottiene le informazioni relative alla propria località: quindi l'insieme delle entry per i nodi che appartengono alla località, e le informazioni sulle località in cui il nuovo nodo è inserito. Per quanto riguarda, invece, la fase di registrazione, essa consiste nel processo per cui si avvisano le altre località che si estendono fino al nuovo nodo, del suo ingresso, e della presenza di una nuova località. Prima di passare al dettaglio del protocollo realizzato vediamo ora, in breve, in cosa consistono queste fasi.

4.3.8.1 Fase di inizializzazione.

La fase di inizializzazione di una nuova località avviene quando, il nodo su cui tale località è centrata deve ottenere le informazioni relative agli elementi che di tale località faranno parte.
Poiché, per ipotesi, assumiamo che tale nodo sia già inserito all'interno della topologia della rete, esso avrà visibilità di un

numero di nodi imprecisato, di tale topologia, variabile da uno a n, dove n è il numero di nodi presenti nella rete. Per ottenere, quindi, le informazioni necessarie per la creazione della propria località si potrà agire in due modi :

1) consultare tutti i nodi di cui si ha visibilità diretta, raccogliere le informazioni ottenute, elaborarle (per esempio per eliminare le informazioni ripetute, ecc..) e costruire su queste la propria località.
2) individuare un nodo a cui richiedere le informazioni necessarie, (che si incarichi anche di raccoglierle, se non le possiede) e costruire quindi la propria località.

Nel nostro caso, si è optati verso la seconda scelta. I motivi sono diversi; il primo motivo è che con il primo metodo, sarebbe stata necessario un uso elevato delle risorse di rete; un secondo motivo è che, spesso, è possibile individuare un nodo che possieda già tutte le informazioni necessarie per la nuova località.
In realtà, quest'ultima affermazione dipende dalla struttura della topologia della rete sottostante e dalla presenza di località con raggi diversi, in particolare essa è sempre verificata nei casi in cui la rete consenta l'inserimento di nuovi nodi solo nella parte terminale della propria topologia (la rete ad albero di SOMA per es.) e le località abbiamo tutte lo stesso raggio.
In entrambi i casi comunque si assume che la struttura della topologia non cambi mentre avviene la creazione di una nuova località. Per ipotesipossiamo quindi escludere il caso di nodi che cadano durante questa fase, non possiamo però escludere il caso della creazione in simultanea di due località; in questo caso, per essere sicuri di ottenere informazioni corrette, le due operazioni dovranno essere, come vedremo, sequenzializzate.

4.3.8.2 Fase di registrazione.

Contrariamente alla fase di inizializzazione, anche nel caso di località con raggi diversi, il processo di realizzazione rimane molto semplice. Una volta che il nuovo nodo invia la propria richiesta di ingresso nel sistema di località, essa già contiene tutte le informa-

zioni con cui la nuova località sarà registrata; considerando l'ipotesi per cui la registrazione avvenga solo in un nodo, quando, a questo, giunge la richiesta di registrazione, esso è subito in grado di stabilire in quali località il nuovo nodo sarà inserito e quindi sarà in grado di procedere immediatamente alla segnalazione della presenza di questo nuovo nodo in tali località.

In realtà, nella nostra implementazione, il meccanismo che realizza l'una è utilizzato per ottenere anche l'altra. Tali fasi sono infatti concomitanti. Va inoltre notato che, la creazione di una nuova località avviene solo con la creazione di un nuovo DefaultPlace; nel caso invece della creazione di un nuovo Place normale, nonostante il manager e il Place si registrino comunque presso il DefaultPlace padrone del dominio, il protocollo attuato è molto più semplice in quanto non necessità la creazione di una nuova località.
Temporalmente, il nuovo Place / DefaultPlace si registra nella topologia secondo i meccanismi si SOMA e, una volta entrato nella topologia, costruisce il proprio manager di località che si occuperà delle successive fasi. Vediamone ora i dettagli nel caso di un DefaultPlace e successivamente nel caso di un Place normale.

4.3.8.3 Creazione di una nuova località.

All'atto della creazione di un nuovo DefaultPlace e della sua registrazione nella topologia, viene creata anche una nuova Località; il nuovo DefaultPlace, cercherà allora di ottenere, dal DefaultPlace presso cui si registra e cioè il suo DefaultPlace padre, l'insieme di informazioni necessarie alle successive fasi in cui creerà la propria località e si registrerà presso le località altrui.
Nel caso che il DefaultPlace sia un nodo radice, c'è una sostanziale semplificazione; non si necessita di nessun tipo di informazione per la creazione della località che sarà vuota e l'unica del sistema.

In questo caso, quindi, il sistema si limita a creare l'ambiente del place e il manager di località che, a sua volta, si occuperà di creare la LocalityTable, il vettore dei PlaceRange, che conterrà solo il PlaceRange della propria località, il thread answerThread e i manager : AYAManager, l' UpdateManger e il TopologyManager.

Prendiamo ora in considerazione il caso della creazione di un DefaultPlace X qualunque. Oltre alle azioni del caso precedente, il sistema(in particolare la classe Creatore di SOMA) invia un comando di registrazione al DefaultPlace padre. Questo comando, chiamato: LocalityTableRegisterCommand (LTRC), contiene lo stato delle risorse del DefaultPlace e le informazioni sulla nuova località, cioè il suo PlaceRange. A questo comando, il manager presente sul DefaultPlace padre, risponde sempre con un comando: LocalityTableRegisterAswerCommad (LTRAC).

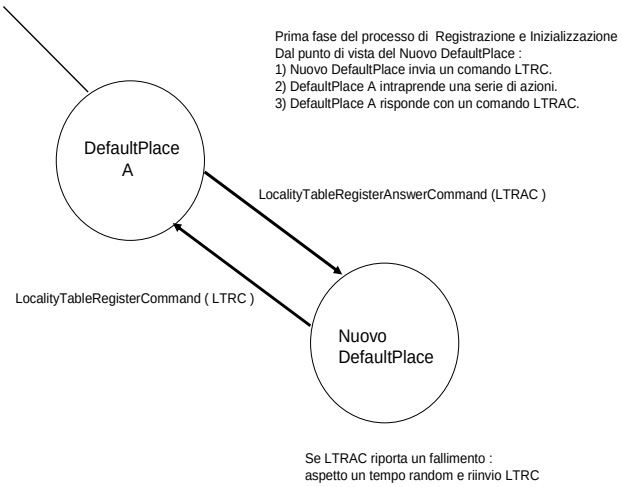


fig 4.8

Prima di rispondere il manager svolgerà una serie di azioni che costituiscono la prima fase del meccanismo di inizializzazione e registrazione (vedi fig. 4.8).
La prima azione del comando consiste nel riservare (bloccare) il manager di località del DefaultPlace affinché X sia l'unico Default

Place che si sta registrando. Se riesce, X sarà l'unico DefaultPlace che potrà registrarsi presso quel manager di località fino a che non verrà liberato(naturalmente, il manager di località viene riservato esclusivamente per quanto riguarda la possibilità di registrare nuovi Place, mentre rimane disponibile per i servizi di informazione sulla località). Se X invece fallisce, viene immediatamente generato il comando LTRAC di ritorno. Questo comando informa X sull'esito della prima parte della sua registrazione; nel caso di

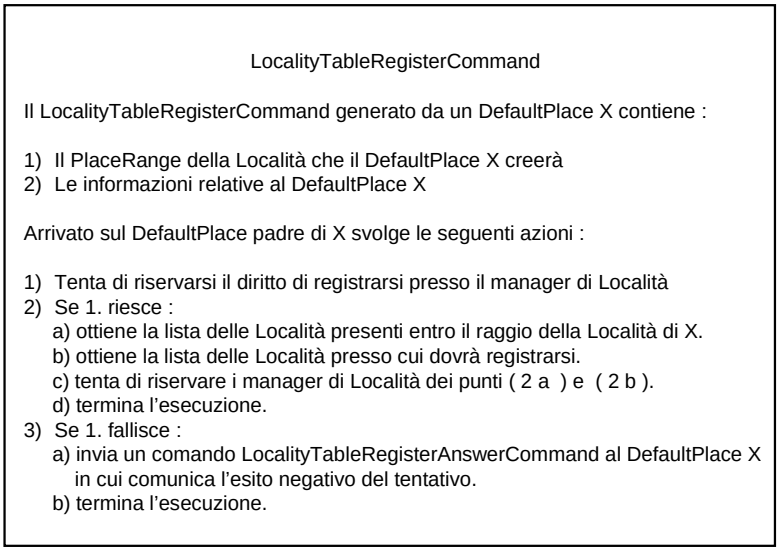
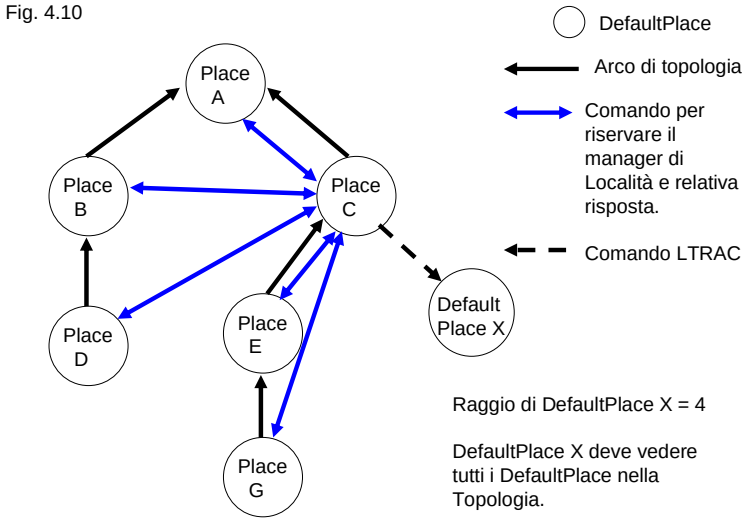


fig 4.9

fallimento, cioè quello che stiamo esaminando, le azioni intraprese sul nodo X, al ricevimento di LTRAC, consistono nell'attesa di un tempo random tra 0 e 3 secondi e nell'invio di un secondo messaggio LTRC al DefaultPlace padre allo scadere del tempo. Se X invece riesce a riservare il manager di località sul DefaultPlace padre, il comando ottiene due vettori: uno contenente le informazioni relative alle località che saranno entro il suo raggio e uno con le informazioni relative alle località da cui X sarà visto.
Naturalmente se nel sistema ogni località ha lo stesso raggio, le due liste coincideranno. Se invece il sistema prevede la possibilità di località con raggi diversi le due liste potrebbero differire.
A questo punto è necessario però prendere in considerazione il ca-

so in cui il DefaultPlace padre di X abbia una località il cui raggio sia inferiore a quello necessario; indicando cioè con R1 il raggio della località di X e con R2 il raggio della località del padre di X, se R1 > R2+1, allora il padre di X non possiede una località abbastanza grande per fornire direttamente tutte le indicazioni necessarie e si rende pertanto necessario un'esplorazione della topologia di cui si occuperà il manager di Topologia. Tale manager, in particolare risponderà alle richieste interagendo con il DNS che mantenendo per ogni DefaultPlace, sia la distanza da questo DefaultPlace che il percorso dalla radice, ha tutte le informazioni necessarie.

Fig. 4.10



Una volta ottenute le informazioni necessarie, si deve procedere a riservare i manager delle località individuate. Anche questa necessità è conseguenza dell'esigenza di sequenzializzare le registrazioni e verrà chiarita successivamente.
Per riservare i manager delle località coinvolte, il manager del DefaultPlace padre, nell'esempio in fig. 4.10, il manager di località di Place C, invia loro un comando. In tale comando viene specificato, il DefaultPlace che ha originato il comando e l'identificatore del DefaultPlace che si sta registrando; quando il comando arriva su un DefaultPlace cerca di riservare il manager di località per la

registrazione del nuovo DefaultPlace. Se il manager non è già stato riservato per la registrazione di altri Place, il comando ha successo; se invece il manager è già occupato nella registrazione di un'altra località, il comando ha esito negativo. In ogni caso, viene generato un comando di ritorno che specifica l'esito dell'operazione e viene inviato al Place padre.

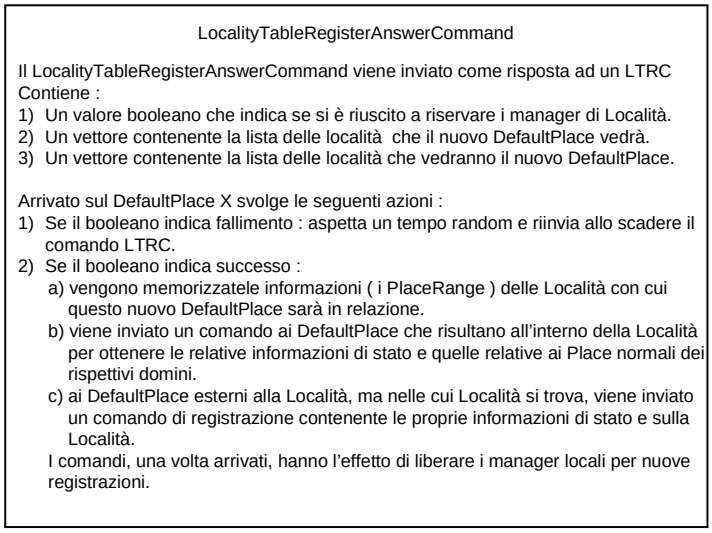


fig. 4.11

Quando tale Place riceve una risposta da uno dei manager a cui ha inviato il comando, controlla quante sono state le risposte e memorizza l'esito riportato. Se il numero di risposte è pari al numero di comandi inviati, controlla l'esito complessivo dell' operazione: se tutti i manager sono stati riservati, l'operazione ha avuto successo, viene inviato al nuovo DefaultPlace il comando:
LocalityTableRegisterAnswerCommand
con indicato l'esito positivo e le informazioni su tutte le Località con cui sarà in relazione; se invece l'operazione non ha avuto successo, viene inviato un comando LTRAC che riporta l'esito negativo della registrazione che verrà tentata dal Place nuovamente allo scadere di un tempo scelto random.
In questo caso, oltre ad avvisare il DefaultPlace che ha tentato la registrazione, il manager del DefaultPlace padre deve anche sbloc-

care i manager che erano stati riservati; per farlo invia a tutti le località coinvolte un comando di sblocco, il cui effetto sarà quello di liberare il manager relativo. (Per lo schema dei comandi LTRC e LTRAC vedi fig. 4.9 e 4.11).
Una volta che il nuovo DefaultPlace riceve un comando LTRAC con esito positivo, possiede tutte le informazioni necessarie per proseguire la registrazione da solo.
Esegue quindi i seguenti passi :
1) fornisce le proprie informazioni alle località che lo vedono.
2) ottiene le informazioni relative a tutti Place e DefaultPlace visti nella sua Località.

In particolare, per realizzare le due fasi, viene inviata a ogni località un comando, tale comando, inserisce il PlaceRange e la Locality TableEntry del nuovo DefaultPlace presso il manager di località e ritorna le informazioni sullo stato delle risorse ed eventualmente il PlaceRange della località (nel caso il proprio DefaultPlace padre non lo avesse avuto). Una volta ottenute tutte le risposte dalle diverse località e costruita la tabella di località, viene attivato il manager di località; in sostanza questo consiste nella creazione dei manager e dell'answerThread e nel loro avvio.
Oltre a questo viene poi segnalato al proprio DNS che la località è attiva, creato il comando LocalityONCommand indicando il proprio PlaceID e inviato al DefaultPlace padre. Tale comando segnala al DNS del DefautlPlace su cui arriva che la località sul nodo indicato è stata attivata. Se su tale DefautlPlace la località risultava già attiva, il comando termina, altrimenti si auto-invia automaticamente ai DefaultPlace padre e figli (il meccanismo di propagazione è quello dei comandi tipici del DNS). Quando giunge inoltre su uno dei DefaultPlace che erano stati bloccati per la registrazione della nuova località, lo sblocca.

4.3.8.4 Motivi della sequenzializzazione.

Prima di passare alla discussione dello stesso meccanismo per un Place normale, chiariamo come nasce l'esigenza di sequenzializza-

re le registrazioni di nuove Località. Innanzitutto, tale esigenza non è su tutto il sistema, le registrazioni possono avvenire anche in maniera concomitante se avvengono in zone della topologia sufficientemente distanti tra di loro. Con distanza sufficiente intendiamo che le due località non devono avere visibilità l'una dell'altra. Se questa condizione è rispettata, vi possono essere anche diverse registrazioni in parallelo nel sistema; se, invece tale condizione non è verificata, allora nasce l'esigenza di sequenzializzazione.

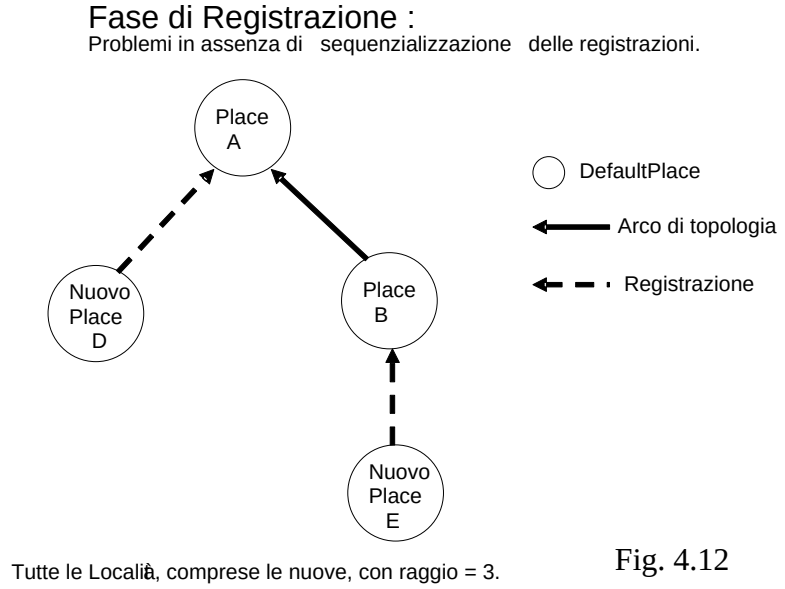


Fig. 4.12

Vediamo l'esempio riportato in figura 4.12; prendiamo in considerazione un DefaultPlace radice e un DefaultPlace figlio già presenti nella topologia e con i rispettivi LocalityManager attivi. Se contemporaneamente cercano di registrarsi due DefaultPlace, uno presso la radice e uno presso il figlio, in un ipotesi di non sequenzializzazione, entrambi inviano il comando LTRC al rispettivo DefaultPlace padre, questo, sulla base delle proprie conoscenze, risponde con LTRAC in cui informa i nuovi Place che gli unici DefaultPlace presenti nel sistema sono quello radice e il figlio; ricevuto tale comando, Nuovo Place D e Nuovo Place E passano a richiedere a Place A e Place B le loro entry e quelle dei Place dei

rispettivi domini ma non si accorgono reciprocamente l'uno dell'altro anche se, le rispettive località hanno un raggio sufficientemente ampio. Ovviamente ciò avviene perché i DefaultPlace padre, rispondono con le informazioni locali che però possono non sono ancora state aggiornate e quindi non sono consistenti. La sequenzializzazione delle registrazioni impedisce, quindi, ai manager di rispondere con informazioni che potrebbero non essere consistenti con lo stato del sistema.

4.3.8.5 Creazione di un Place normale.

Il procedimento di inizializzazione per un Place normale è molto più semplice rispetto a quello di un DefaultPlace. Il Place invia al DefaultPlace padre un LTRC con il proprio stato; una volta arrivato, il comando cerca di bloccare il LocalityManager, se vi riesce, viene aggiornata la tabella della località e viene mandato un comando a tutti i DefaultPlace che vedranno, all'interno della loro località, il nuovo Place, infine viene liberato il manager di Località. Se con il comando LTRC non si riesce a riservare il manager del DefaultPlace, viene inviato al Place un comando LTRAC in cui si riporta l'esito negativo del tentativo di registrazione;

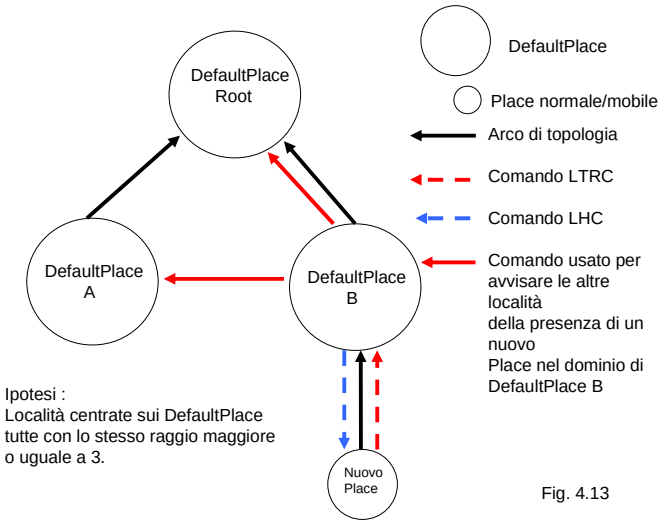


Fig. 4.13

una volta arrivato, questo comando, attiva un timer allo scadere del

quale la registrazione viene ritentata. Il tempo di attesa del timer è determinato in maniera random ma in un intervallo più ampio che nel caso di un DefaultPlace; questo per garantire la precedenza a questi durante la fase di registrazione e in caso di eventuali collisioni. Nel caso invece che la registrazione avvenga con successo, il Place normale deve agganciarsi al manager del Default Place; in particolare, il DefaultPlace indica la porta su cui è attivo l'answerThread. Viene quindi mandato al Place un comando LocalityHookCommand (LHC) con indirizzo e porta del thread. Tale comando ha come effetto quello di attivare il componente RequestSender del manager e di attivare il manager, concludendo la registrazione (vedi fig. 4.13 per lo schema dei messaggi).
NOTA : nel caso che nel sistema non vengano usate le località, la registrazione di un DefaultPlace o di un Place normale non ha nessun effetto; quando un comando LTRC arriva su un Default Place in cui non è presente un manager di Località informa infatti il suo nodo di origine che nel sistema non sono utilizzate località e la registrazione termina.

4.3.9 Aggiornamento delle località.

L'aggiornamento delle informazioni relative allo stato delle risorse dei nodi della località è il servizio principale nell'ambito del nostro sistema di località. Essenzialmente esistono due modalità per l'ottenimento delle informazioni di stato necessarie in una località; la prima consiste nell'attuare un meccanismo di probing degli elementi della località. La seconda consiste invece nel realizzare un meccanismo di notifica di un evento. Nella parte iniziale di questo paragrafo vedremo in dettaglio queste possibilità, nella seconda fase invece descriveremo i protocolli realizzati effettivamente per il sistema. Nel nostro progetto sono, infatti, stati sviluppati tre protocolli, dal più semplice al più complesso ed efficiente, realizzando infine un ibrido basato sulla particolare topologia di SOMA che coniugasse i vantaggi del modello a notifica di eventi con il model-

lo di probing.

4.3.9.1 Il probing.

Generalmente parlando, il probing di un nodo, è un meccanismo per cui, periodicamente, con una frequenza prefissata, viene inviato un messaggio di test al nodo oggetto del probing. I motivi per cui si adotta questo meccanismo, che può essere rivolto a un singolo nodo come ad un insieme di nodi, possono essere diversi e, in relazione a questi, il comportamento del messaggio di test sarà diverso da caso a caso. In genere, il probing viene utilizzato per ottenere informazioni di natura dinamica che devono essere periodicamente aggiornate per essere mantenute coerenti (nel nostro caso, per esempio, ci interessano le informazioni sullo stato delle risorse). Un altro utilizzo frequente del probing è mantenere sotto controllo la topologia della rete; inviando semplicemente un messaggio che richieda una risposta è infatti possibile, sotto certe ipotesi (per esempio che i messaggi non si perdano), verificare se un nodo della rete è attivo o meno ed attivare quindi un'eventuale fase di riconfigurazione della rete. Nel caso, invece, che non sia possibile assumere l'ipotesi che i messaggi in rete arrivino sempre a destinazione, il meccanismo di probing dovrà prevedere un meccanismo di timing per cui si attenderanno i messaggi di risposta solo per un intervallo di tempo predeterminato scaduto il quale, si considereranno caduti i nodi da cui non è arrivata risposta.
Un problema strettamente legato al probing è l'utilizzo delle risorse di rete. In particolare l'impegno delle risorse di rete è in relazione al numero di nodi che effettuano probing, al numero di nodi coinvolti e alla frequenza con cui il probing è effettuato.
Se per esempio consideriamo il caso di una rete con n nodi e in cui ogni nodo attua il meccanismo di probing sugli altri nodi abbiamo che il per ogni ciclo di probing, il numero di messaggi nella rete è pari a 2*n*(n-1). E' quindi sempre necessario cercare di ridurre il più possibile i messaggi di probing.
Ciò è possibile se si dilatano gli intervalli con cui il probing viene attuato, tuttavia questo è a scapito dell'aderenza delle informazioni

ottenute con lo stato reale delle risorse.
La possibilità di effettuare un probing parziale (limitando il numero di nodi) è quindi sicuramente vantaggiosa dal punto di vista del risparmio delle risorse di rete; un tipico caso di politica di probing parziale è il caso in cui si interrogano esclusivamente i nodi confinanti (con i quali si suppone che la comunicazione sia poco costosa). Inoltre, anche nel caso di una politica di probe parziale, è possibile rendersi conto della presenza di macchine cadute in quanto, ogni nodo continua a essere oggetto di controllo di almeno un altro nodo confinante (che se ne rileva la caduta avvierà la riconfigurazione per tutto la rete).
Tramite l'adozione di politiche di probing parziale, il numero dei messaggi si può ridurre notevolmente, tuttavia il probe viene attuato sia che ve ne sia bisogno, sia che le condizioni del sistema siano sostanzialmente costanti e, non ve ne sia quindi una reale necessità. Per evitare questo inconveniente si dovrà attuare una politica di notifica di eventi.

4.3.9.2 La politica di notifica degli eventi.

Concettualmente diversa dal meccanismo di probing è il meccanismo di notifica di un evento; esso prevede che ogni nodo monitorizzi, a intervalli regolari, il proprio stato; nel caso, poi, che vi siano stati cambiamenti significativi viene generato un evento, che verrà segnalato a tutti i nodi che ne hanno visibilità. Per realizzare questo tipo di politica è necessaria una struttura più complessa rispetto al caso del probing, tuttavia la maggiore complessità è compensata da un minore uso delle risorse di rete, che questa volta, vengono impegnate esclusivamente quando ve ne è reale necessità. In particolare, per potere notificare correttamente gli eventi relativi al cambiamento di stato delle proprie risorse, ogni nodo dovrà mantenere traccia di tutti i nodi che dovranno essere avvertiti, mentre nella politica di probing, tale necessità non sussiste (nel nostro modello di località, tali informazioni sono comunque mantenute a prescindere dalla politica di aggiornamento delle informazioni adottata). Oltre a ciò, ogni nodo dovrà definire delle soglie

per il cambiamento dello stato delle risorse; tali soglie saranno utilizzate per determinare se l'entità del cambiamento nello stato della risorsa è sufficientemente ampio da dover essere comunicato o meno. In particolare poi, lo stato delle risorse che viene notificato, dovrà essere memorizzato e la variazione delle risorse sarà in rapporto a tale stato.
L'adozione di questa politica, come abbiamo anticipato, consente una riduzione dei messaggi di aggiornamento nella rete rispetto al caso del probing, ha però lo svantaggio di non consentire più direttamente il controllo della topologia. Poiché infatti, in questo caso il silenzio di un nodo può essere dovuto ad uno stato stazionario delle sue risorse, non vi è modo di accorgersi della sua eventuale caduta. Nel caso quindi che si decida di adottare questa politica per l'aggiornamento delle informazioni di stato dei nodi della località si dovrà tenere conto che sarà necessaria la presenza di una ulteriore entità per il controllo della topologia (notiamo comunque che il controllo della topologia, può essere attuato a intervalli molto più lunghi rispetto a quelli di un meccanismo di probing; per questo, anche con gli eventuali messaggi necessari per tale controllo, in generale questa politica può offrire risultati migliori in termini di occupazione delle risorse di rete).

4.3.9.3 Politica di probing globale.

Questa politica di probing viene chiamata globale e costituisce sostanzialmente la politica di probing più semplice ipotizzabile. In sostanza, il DefaultPlace su cui è centrata la località invia, a intervalli regolari e prefissati, un comando di probing ad ogni nodo della località, sia esso un Place normale o un DefaultPlace. Il Place a sua volta, risponde con le informazioni relative allo stato delle sue risorse. Una volta ricevuta la risposta, il DefaultPlace che ha comandato il probe, aggiorna la tabella. Nel caso, infine, che un comando di probe si perda oppure che uno dei Place interrogati sia in realtà caduto, il DefaultPlace che ha comandato il probe, non otterrà risposta e attiverà il meccanismo di recovery.

All’atto pratico, all’avvio del meccanismo di probing globale, il DefaultPlace attiva un timer e memorizza quanti sono i Place interrogati. Se allo scadere del timer, non sono pervenute tutte le risposte, viene attivata la fase di Recovery.

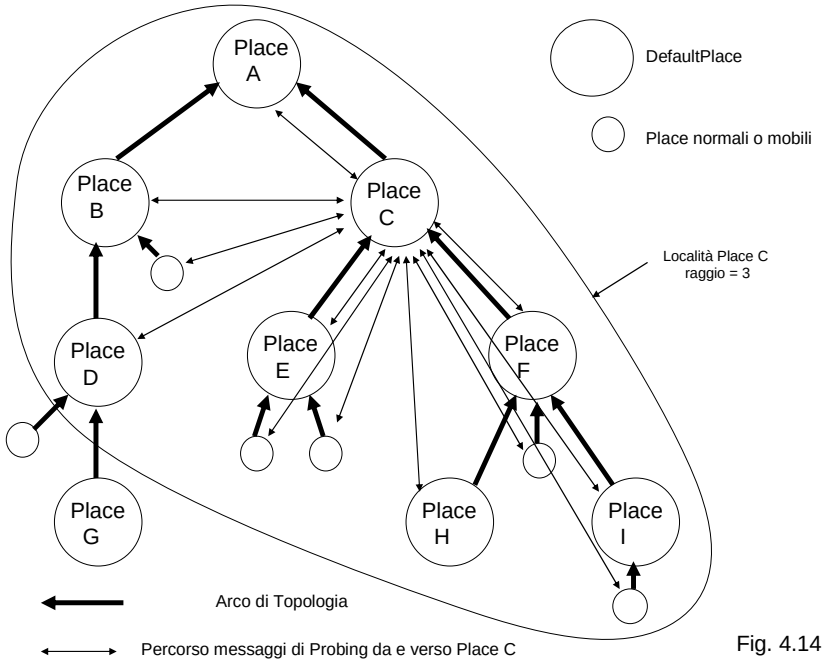


Fig. 4.14

E’da notare che, il numero dei Place interrogati, viene memorizzato perché il numero di elementi nella località potrebbe variare durante la fase di probing e il numero di risposte attese potrebbe quindi essere diverso da quello degli elementi della località.
Lo svantaggio di questo protocollo di probing è evidente: esso inonda periodicamente la rete con messaggi di probing e relative risposte causando possibili congestioni (in fig. 4.14 è mostrato un esempio dei messaggi usati da un singolo nodo).
Il vantaggio di questo meccanismo, oltre a consentire un controllo sulla topologia, è che consente di ottenere un valore relativo al tempo di risposta per l’interrogazione e quindi di poter in qualche modo stimare le condizioni della rete e/o la vici-nanza reale dei Place; in sostanza, quando il comando di probing parte verso un Place, porta con sé l’indicazione dell’ora di Sistema del Default

Place; tale valore viene riportato anche nella risposta del Place interrogato, in questo modo la differenza tra l'ora di Sistema attuale e quella di partenza consentono di avere il tempo di risposta dei Place della località e quindi la loro "distanza" in funzione delle condizioni reali della rete e dello spazio fisico.

4.3.9.4 Politica di Probing locale.

Una versione modificata del protocollo visto, volta a ridurre l'utilizzo delle risorse di rete è quella che presentiamo ora; come nel caso precedente, il DefaultPlace attua il protocollo a intervalli regolari; in questo caso, tuttavia, il comando di Probe non viene in-

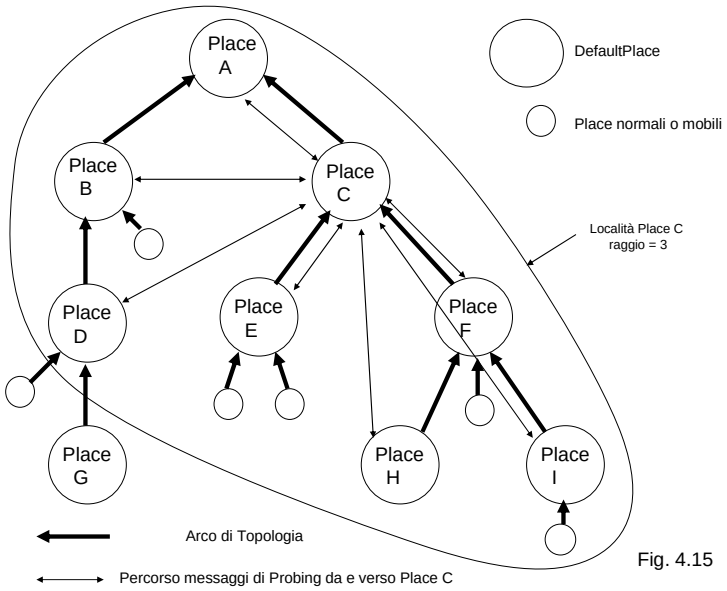


Fig. 4.15

viato a tutti i Place della località ma esclusivamente ai Default Place e ai Place normali e mobili del proprio dominio. I place del dominio rispondono con il loro stato, i DefaultPlace della località rispondono con un vettore contenente sia il loro stato che quello dei Place normali e mobili del proprio dominio. Nell'esempio in figura 4.15, il PlaceE risponderà con le informazioni relative sia al proprio stato che a quello dei Place figli e lo stesso varrà per

PlaceB, PlaceF e PlaceI. Per quanto invece riguarda PlaceD, poiché conosce il raggio della località di PlaceC, sa che i Place normali del suo dominio non sono nella sua località e risponderà solo con le infor-mazioni sul proprio stato. Con questa politica, il controllo sulle in-formazioni ottenute è diverso dal caso precedente. Nel caso pre-cedente, il DefaultPlace che attuava il probing, si limitava a me-morizzare il numero di comandi inviati e a controllare il numero delle risposte; in questo caso, ciò non è più sufficiente. Vediamo quali casi si possono verificare :

1) la località è rimasta tale e quale e tutti i DefaultPlace rispondono con le informazioni sul proprio stato e su quello dei propri Place. In questo caso, naturalmente non accade nulla se non l'aggiorna-mento della tabella della Località
2) un Place del proprio dominio è caduto e quindi non risponde. In questo caso non viene attivato il meccanismo di recovery ma semplicemente viene eliminata la rispettiva entry della tabella di località. Quando un altro DefaultPlace attuerà il probing otterrà un vettore in cui non saranno presenti le informazioni relative al Place caduto e pertanto, si troverà nel caso 5.
3) un DefaultPlace nella località è caduto e quindi non risponde. In questo caso viene attivato il meccanismo di recovery.
4) un Place si è aggiunto ad un altro dominio e quindi ottengo dal DefaultPlace proprietario del dominio anche le sue informazioni. In questo caso, quando arriva la risposta al comando di probe, le informazioni relative al Place nuovo vengono semplicemente ignorate. Questo perché l'inserimento del nuovo Place sarà effet-tuato dal meccanismo di registrazione.
5) un Place di un altro dominio ma visto nella località è caduto e quindi non ottengo le informazioni relative.
   Quando un DefaultPlace riceve da un altro DefaultPlace la ri-sposta al comando di Probe, per prima cosa deve riprendere dalla propria tabella la lista dei Place appartenenti al dominio in que-stione. Ottenuta tale lista semplicemente effettua un controllo per verificare che i Place presenti nella propria tabella siano tutti in-

dicati nel messaggio ottenuto. Se un Place è presente nella tabella ma non nella risposta ottenuta viene cancellato dalla tabella altrimenti semplicemente ne viene aggiornato lo stato.

Naturalmente, anche se qui sono stati presentati individualmente, si può presentare una qualunque combinazione di questi casi, tenendo però sempre in considerazione che il protocollo di recovery è stato sviluppato nell'ipotesi che vi sia, al più, la caduta di un Default Place.

Questa versione "locale" del precedente protocollo, nasce dalla considerazione che l'interrogazione dei Place del proprio dominio ha comunque un costo accettabile e si cerca di ridurre invece lo scambio di messaggi extra dominio. Oltre a ciò, le considerazioni del paragrafo precedente sulla possibilità di ottenere un parametro di valutazione per le condizioni di distanza/intasamento della rete, rimangono, seppur più grossolanamente, valide in quanto, nell'ottica di un dominio come un contesto in cui la comunicazione è veloce e poco costosa, i tempi delle comunicazioni all'interno del dominio possono essere sostanzialmente trascurati rispetto ai tempi per le comunicazioni extra dominio.

4.3.9.5 Politica di notifica di eventi.

Questo terzo protocollo, non realizza un meccanismo di notifica di eventi "puro" ma come vedremo lo fonde con il meccanismo di probing.
In particolare, i Place soggetti al probing sono esclusivamente i Place del proprio dominio. Come abbiamo già evidenziato, infatti, è ragionevole ipotizzare che le comunicazioni all'interno di un dominio siano poco costose e pertanto, il probing limitato al dominio è accettabile. Quando il DefaultPlace riceve una risposta da un Place del dominio, aggiorna il relativo stato nella tabella e controlla quanto grande è stata la variazione nei parametri di stato del Place. Se vengono superate certe soglie prefissate, il DefaultPlace

si occuperà di notificare il cambiamento di stato a tutti gli altri DefaultPlace che lo vedono all'interno della rispettiva località. Si verifica in sostanza un evento rilevante, cioè un cambiamento dello stato delle risorse tale che deve necessariamente essere comunicato. Naturalmente nel messaggio di notifica verrà indicato lo stato di tutti i Place del dominio e non solo quello del Place che ha generato l'evento. Inoltre il DefaultPlace prima di inviare l'eventuale notifica dovrà aspettare che tutti i Place del proprio dominio abbiano risposto in modo da non dover inviare più notifiche nel caso che più di un Place abbia una variazione di stato da segnalare.

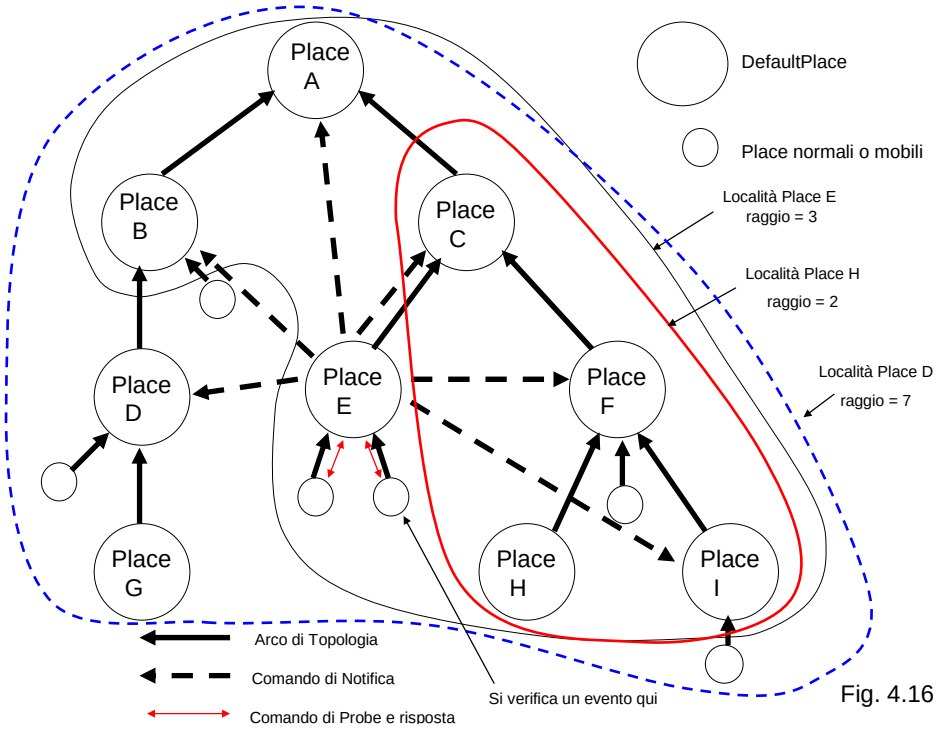


Fig. 4.16

Nell'esempio mostrato in fig 4.16, vediamo un esempio di questo protocollo. In particolare, notiamo che PlaceD riceve il messaggio di notifica del cambiamento di stato di uno dei Place del dominio di PlaceE pur essendo esterno alla sua località. Al contrario, PlaceH, anche se si trova all'interno della località di PlaceE non riceve tale notifica in quanto la sua località non ha raggio sufficien-

te. Nel caso infine in cui, in nessun Place vi sia una variazione di stato sufficientemente ampia, non viene inviata nessuna notifica. Nel caso in cui un Place o un DefaultPlace cadano, tenendo come riferimento la figura, e ipotizzando che si consideri la Località di DefaultPlace PlaceE si possono verificare i seguenti casi :

1) cade un Place appartenente al dominio di PlaceE
2) cade un Place appartenente al dominio di un altro DefaultPlace visto nella località di PlaceE
3) cade un DefaultPlace visto nella località di PlaceE

Nel primo caso, non ci sono problemi, il meccanismo di probing attiva un timer e la situazione è come nei casi precedenti; se un Place non risponde viene considerato caduto e lo si rimuove dalla tabella della località (e dal PNS); ciò inoltre viene considerato un evento rilevante e viene quindi inviato un comando di notifica ai DefaultPlace interessati.
Nel secondo caso PlaceE, il DefaultPlace responsabile del dominio invia un comando in cui notifica l'evento la caduta di un Place; il manager di località si limita quindi a eliminare l'entry relativa dalla propria tabella di Località.
Nel terzo caso, poiché è impossibile distinguere l'assenza di comunicazione dovuta ad assenza di eventi dall'assenza di comunicazione dovuta alla caduta di un nodo, per avviare il recovery è necessaria la presenza dell'AYA Manager.
Notiamo qui che, se il periodo con cui viene effettuato il probe e il periodo con cui viene mandato il messaggio Are You Alive, coincidono, l'occupazione delle risorse di rete di questo protocollo è identico o superiore al caso precedente. Il vantaggio reale consiste nel fatto che tale messaggio può essere inviato ad intervalli molto più laschi di quanto non sia necessario per i meccanismi di probing. Complessivamente, questa politica consente di minimizzare la comunicazione extra dominio e pertanto, nell'ottica del risparmio delle risorse di rete, ci è sembrata la scelta migliore ed è stata pertanto quella adottata a default

4.3.10 Il servizio di recovery.

A seconda di quale meccanismo si utilizzi per ottenere le informazioni relative allo stato delle risorse dei nodi di una località, il meccanismo che attiverà la fase di recovery sarà diversa. In generale, se viene effettuato un meccanismo di probing della località, sarà tramite questo servizio che si attiverà eventualmente il servizio di recovery del sistema. Se invece si adotta un meccanismo di notifica degli eventi, sarà necessario prevedere anche un controllo separato (tramite probing) della località. In entrambi i casi, il servizio di recovery verrà attivato allo scadere di un tempo prefissato (time out) in cui si attenderà l'arrivo delle risposte dei nodi controllati.

Nella maggior parte dei casi, il recovery della topologia comporta una ricostruzione globale o locale della topologia stessa e, un successivo adattamento delle località del sistema alla nuova topologia.

I protocolli/meccanismi di recovery possono essere poi sviluppati secondo due linee di principio fondamentalmente distinte:

1) protocolli che richiedono un coordinamento dei nodi superstiti per la ricreazione della topologia.
2) protocolli che attuano la riconfigurazione della topologia in modo autonomo per i singoli nodi della rete.

Nel primo caso, abbiamo, in genere protocolli complessi che possono impiegare meccanismi di negoziazione a più fasi ma che, d'altro canto possono realizzare diverse politiche a seconda dello stato del sistema; nel secondo caso, invece, abbiamo in genere, protocolli più snelli e veloci, che però hanno, un grado di espressività e flessibilità minore.

Come sempre, non esiste una soluzione universalmente valida. Nell'ambito del nostro progetto è stato realizzato, per il recovery, sia un protocollo che richiede un coordinamento tra i nodi rimanenti, che un protocollo che consente una riconfigurazione della topologia in modo quasi autonomo per i singoli nodi. All'atto del confronto, è stato infine scelto quest'ultimo protocollo per il progetto definitivo in quanto, per le nostre esigenze, la riconfigurazio-

ne della topologia non presentava particolari necessità e si è preferito quindi l'adozione di un protocollo che impegnasse al minimo le risorse del sistema e che fosse il più snello possibile. In appendice A, viene comunque presentato il secondo protocollo realizzato.

4.3.10.1 Il protocollo di recovery del sistema.

Con politica di recovery si intende l'insieme di operazione che vengono attuate nel sistema quando si verifica la caduta, reale o presunta, di un DefaultPlace. Come abbiamo già spiegato nei paragrafi precedenti non si vuole qui distinguere i casi in cui un messaggio non giunge a destinazione per motivi di congestione della rete o perché non è stato affatto inviato in quanto il nodo è caduto; attualmente ci si limita a prendere atto del verificarsi di un evento. Tale evento viene, nel nostro caso, generato dal manager AYA Manager che a intervalli regolari invia a tutti i DefaultPlace presenti nella località, il comando Are You Alive e attiva un timer. Se allo scadere del timer non sono pervenute tutte le risposte viene generato un evento che attiva la fase di recovery.
Poiché lo scopo del recovery è quello di aggiornare le informazioni di sistema affinché rimangano coerenti e ricostruire se possibile la topologia, è necessario:

1) aggiornare i DNS di tutti i DefaultPlace;
2) riorganizzare la struttura delle località; se infatti ricostruiamo la topologia dopo la caduta di un DefaultPlace, inevitabilmente la nuova topologia presenterà un ramo più corto rispetto a quella precedente; ciò necessiterà l'aggiornamento di tutte le distanze, dei Path dei vari DefaultPlace e l'eventuale ampliamento delle località nella direzione in cui l'albero della topologia si è accorciato.

Facciamo infine nuovamente presente che, nelle nostre ipotesi, è possibile la caduta di un solo DefaultPlace alla volta.

Per la descrizione del meccanismo di recovery di rifaremo in

particolare alla situazione presentata nelle figure successive, in cui viene mostrato il caso in cui in un sistema, con località tutte con lo stesso raggio, cada il DefaultPlace PlaceC. Le due immagini della topologia mostrano la configurazione originale e la topologia ricostruita, nelle tabelle invece viene indicata la visibilità dei Default Place nelle varie località, prima della caduta del nodo e dopo il recovery.
In generale, dobbiamo tenere in considerazione che ognuno dei DefaultPlace che vedevano PlaceC può accorgersi in maniera indipendente dagli altri della sua caduta e quindi avviare la fase di recovery. La scelta è quindi tra la definizione di un protocollo di recovery idempotente, tale cioè per cui anche se avviato da diversi DefaultPlace in tempi diversi, l'effetto complessivo è lo stesso che se fosse eseguito una sola volta; oppure la definizione di un protocollo con coordinamento tra i DefaultPlace. Con l'estensione delle informazioni presenti nel DNS è stato possibile ottenere una gestione della riconfigurazione della topologia in "locale",e si è optati per la prima opzione.

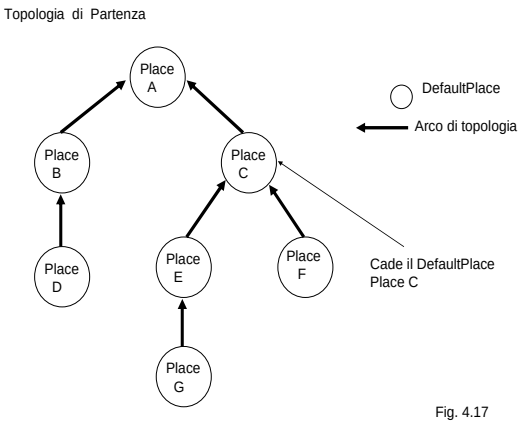


Fig. 4.17

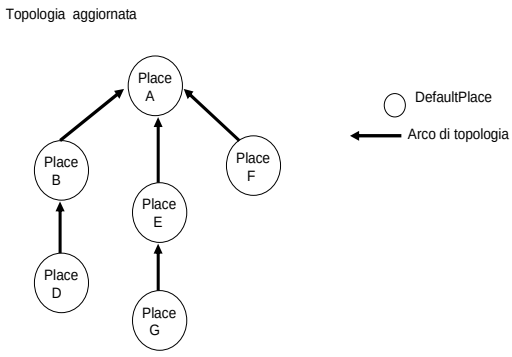


Fig. 4.18

TABELLA DELLE LOCALITA' NEL SISTEMA

Place propietario della Località	Place visti nella Località (ogni località ha raggio = 2)					
Place A	Place A	Place B	Place C	Place D	Place E	Place F
Place B	Place A	Place B	Place C	Place D		
Place C	Place A	Place B	Place C	Place E	Place F	Place G
Place D	Place A	Place B	Place D			
Place E	Place A	Place C	Place E	Place F	Place G	
Place F	Place A	Place B	Place E	Place F		
Place G	Place C	Place E	Place G			

Tabella 1.

TABELLA DELLE LOCALITA' AGGIORNATA

Place propietario della Località	Place visti nella Località (ogni località ha raggio = 2)					
Place A	Place A	Place B	Place D	Place E	Place F	Place G
Place B	Place A	Place B	Place D	Place E	Place F	
Place D	Place A	Place B	Place D			
Place E	Place A	Place B	Place C	Place E	Place F	Place G
Place F	Place A	Place B	Place E	Place F		
Place G	Place A	Place E	Place G			

Tabella 2.

I vantaggi di protocolli che agiscano in locale sono evidenti, tuttavia questi hanno richiesto l'estensione delle informazioni del DNS, mentre il protocollo basato sul coordinamento dei DefaultPlace, oltre alla possibilità di attuare politiche in genere più flessibili (per esempio con l'adozione di forme di negoziazione per la ricostruzione della topologia), non necessitava l'estensione delle informazioni mantenute nel DNS.

Vediamo come si procede :

1) Un DefaultPlace K attiva la fase di riconfigurazione della topologia tramite il LocalityManager chiamando il metodo startReconfiguration(place) dove place è il place da rimuovere. Questo metodo può essere chiamato o dal manager AYAManager o da un comando di SOMA:
   RemoveDomainCommand
2) Per prima cosa, il LocalityManager controlla se esiste nel DNS una DomainInfo per tale PlaceID; è possibile, infatti, che la fase di Recovery per il DefaultPlace K sia già stata attivata e la DomainInfo rimossa. In questo caso il metodo chiamato ritorna senza fare nulla.
3) Se è presente la DomainInfo, il manager controlla se è presente una entry corrispondente nella tabella. Se non è così, è perché ci si trova lontani dal DefaultPlace caduto (K non lo vede nella sua località). In questo caso, si passa al punto 4 altrimenti si passa al punto 5.
4) In questa situazione si deve riconfigurare solamente il DNS ed eventualmente i PlaceRange di località che vedono il DefaultPlace K; La località di K non viene comunque modificata dal recovery. Viene chiamato il metodo Remove Domain nel DNS, e quindi si verifica se tra i PlaceRange presenti ve ne sono alcuni che indicano una distanza del DefaultPlace diversa da quella indicata nelle DomainInfo aggiornate. Se ve ne sono, vengono rimossi e sostituiti con nuovi PlaceRange con le distanze corrette. Dopodichè la riconfigurazione termina.
5) Vengono bloccati il manager UpdateManager e l'AYAManager e viene chiamato il metodo startReconfiguration del TopologyManager indicando il DefaultPlace caduto.
6) Il manager di topologia per prima cosa chiama il metodo removeDomain del DNS. Viene poi chiamato il metodo removeEntry del LocalityManager (tale metodo quando viene specificato un DefaultPlace rimuove anche tutti i Place

del rispettivo dominio e aggiorna le altre entry). Inizia poi la riconfigurazione della località.

7) Si controlla, per ogni PlaceRange, se la distanza è diversa rispetto a quella indicata dal DNS; se è così, si sostituisce il PlaceRange (come nel punto 4). Se la distanza indicata dalla DomainInfo è pari al raggio di località meno uno devo ottenere le entry relative ai Place normali di questi domini (che prima non vedevo); per ottenerli, creo quindi un comando ReconfigureLocalityCommand, lo invio al Default Place in esame; inoltre segnalo al manager di località che sono in attesa di una risposta.
8) Vengono ora controllati i DefaultPlace la cui distanza è pari al raggio di località; se non era già presente il rispettivo PlaceRange significa che tali DefaultPlace si sono avvicinati e ora li vedo nella località mentre prima no; devo quindi recuperare il loro stato e inserire il PlaceRange della loro località. Viene quindi creato, anche in questo caso, un comando ReconfigureLocalityCommand e inviato ai Default Place in esame; infine viene segnalato al manager di località che si è in attesa delle risposta. Con questa fase termina la prima parte riconfigurazione locale.
9) I ReconfigureLocalityCommand, una volta a destinazione, recuperano le informazioni necessarie, attivano la riconfigurazione (se non è già stata attivata) e si rispediscono al DefaultPlace mittente dove una volta giunti aggiornano le informazioni sulla località e avvisano il manager dell'arrivo di una risposta ad un ReconfigureLocalityCommand. Una volta infine che tutte le risposte sono giunte, la riconfigurazione termina e viene quindi chiamato il metodo ReconfigurationEnded del LocalityManager che riattiva i manager UpdateManager e AYAManager e si ritorna alla gestione normale della Località.

E' necessario sottolineare come tale protocollo non sia attuabile nel caso della caduta del nodo radice poiché di fatto, in questo caso,

non esiste un nodo padre di riferimento. La caduta del nodo radice deve essere d'altronde considerata un evento raro nel sistema.
Oltre a ciò, la caduta del nodo radice determina diversi scenari : la generazione di N Topologie distinte, e non comunicanti tra loro, dove N è il numero dei figli del nodo Radice; oppure l'elezione di una nuova Radice tra i suoi figli, sono solo alcune delle possibilità tra le quali si deve necessariamente effettuare una scelta prima di trattare il caso.

4.4 Conclusione.

In questo capitolo ci siamo occupati della realizzazione del sistema di località sulla piattaforma SOMA. Per prima cosa abbiamo introdotto evidenziato quelle modifiche che è stato necessario apportare a alla piattaforma e infine, abbiamo descritto l'architettura realizzata per lo sviluppo del sistema di località ideato. Con questo capitolo, si chiude quindi la parte relativa al reperimento delle informazioni necessarie per una politica di adattamento intelligente dei flussi multimediali alle condizioni del sistema; si apre, invece, quella destinata alla realizzazione di un modello di riconfigurazione del sistema. Nel prossimo capitolo presenteremo il modello per la riconfigurazione che intendiamo integrare in MUM; in tale capitolo, presenteremo anche un protocollo da noi ideato, per la riconfigurazione dei percorsi dei flussi che cerchi di sfruttare le conoscenze sullo stato delle risorse della località.
Nel capitolo sei, infine, vedremo come il modello per la gestione della riconfigurazione del middleware verrà realizzato e trarremo le conclusioni sul lavoro effettuato.

CAPITOLO 5.

5 Modello per l' Adattamento dei Flussi Multimediali alle Condizioni del Sistema.

Il problema della riconfigurazione di flussi multimediali in relazione alle condizioni del sistema, è una questione che può essere sostanzialmente affronatata in due fasi. La prima fase consiste nella creazione di un modello di descrizione degli stessi flussi, la seconda nella definizione delle struttre e dei meccanismi per la realizzazione dell'adattamento stesso. In questo capitolo intendiamo proporre, nella prima parte, il modello per la descrizione dei flussi che abbiamo usato e il modello per la riconfigurazione del sistema e l'adattamento degli stessi; successivamente vedremo come tale modello sia stato sviluppato sulla base del middleware MUM.

5.1 Modello per la riconfigurazione del sistema e l'adattamento dei flussi.

L'idea di base del modello che intendiamo presentare, è che non si vuole attuare un particolare protocollo di riconfigurazione ma, piuttosto, si intende creare una struttura che consenta lo svolgimento di diversi protocolli a seconda delle condizioni del sistema e delle esigenze del flusso; riconducendo la gestione di diversi eventi ad un unico sottosistema. Fondamentalmente, alla base di tale struttura vi è la necessità di poter gestire diversi protocolli di riconfigurazione in maniera omogenea e di avere gli strumenti per poter decidere al meglio, quale protocollo attuare. In questo paragrafo, vedremo in che modo il modello che abbiamo realizzato risponde a tali esigenze.

5.1.1 Riconfigurazione del sistema.

In questo paragrafo, intediamo definire cosa intendiamo con riconfigurazione del sistema a seconda delle condizioni dello stesso.

La fruizione di un flusso di dati, in MUM, da parte di una generica applicazione utente, è un caso particolare, tuttavia, considereremo tale caso per chiarire i concetti che verranno presentati.
Quando, in MUM, un client indica una presentazione, il sistema provvede, in maniera trasparente, a generare un piano; tale piano, eseguito dai PlanVisitorAgent, a runtime, stabilisce il percorso tra il client e un server/place, in cui è presente la presentazione indicata, e, oltre a ciò, determina la posizione degli eventuali componenti necessari lungo il percorso stesso tra client e server. Ogni componente, quando viene piazzato su un Place ottiene una certa quantità di risorse dal sistema, dipendente dal tipo di elaborazione svolta. Tipicamente, tali risorse consistono in una precentuale di uso della CPU, una parte della banda disponibile, una certa quantità di RAM necessaria per il mantenimento in memoria e l'esecuzione dell'attività prevista ed eventualmente un certo quantitativo di memoria secondaria. Quando però, a run time, le condizioni necessarie allo svolgimento delle attività programmate vengano a mancare, si presenta la necessità di riconfigurare il sistema.
Con riconfigurare il sistema intendiamo quindi cambiare la configurazione del sistema in uno stato T che possiamo chiamare C(T), in una seconda configurazione che possiamo considerare C(T1) , con T1 successivo a T, tale che per C(T1) sia verificata una condizione non verificata per C(T).
Generalmente, le condizioni che si devono verificare sono inerenti la richiesta di risorse per l'elaborazione del flusso; non è tuttavia sempre così: esistono infatti diversi altri casi in cui può essere necessario adattare i flussi del sistema. Vediamo allora in quali situazioni, il processo di riconfigurazione può essere attivato.

5.1.2 Avvio del processo di riconfigurazione del sistema.

Come abbiamo visto, il processo di riconfigurazione viene attivato quando nel sistema non è verificata una particolare condizione.

L’insieme di condizioni che un sistema, come un middleware, deve soddisfare sono generalmente implicite, cioè non formulate espressamente, (mediante un linguaggio dichiarativo per esempio) e dipendono dalle possibilità che il sistema stesso mette a disposizione. Nel nostro caso, per i presupposti di questo progetto, prenderemo in considerazione il middleware MUM che consente, a livello teorico, l’attivazione dinamica dei servizi di elaborazione dei flussi (es. conversione/riduzione di formato per file audio/video, ecc..), la mobilità sia di utente che di terminale, e la possibilità di riconfigurazione dei percorsi dei flussi; in tale caso, si possono individuare tre situazioni distinte che danno origine al processo di riconfigurazione; tali situazioni, sono concettualmente diverse, e, pertanto generano eventi distini e indipendenti l’uno dall’altro.

5.1.2.1 Riconfigurazione su richiesta del client.

Questo tipo di riconfigurazione è generato da una specifica richiesta del client. In particolare un client nomadico può decidere di spostare la sessione creata su un primo terminale in un secondo, oppure può essere richiesta una migliore qualità della presentazione richiesta, oppure, ancora, una particolare elaborazione della stessa (ad esempio un’elaborazione della traccia audio per ridurre i rumori di sottofondo, ecc..). In questo caso, è necessario sottolineare come che le richieste del client possono avvenire in un qualunque momento e qualunque sia lo stato del sistema.

5.1.2.2 Riconfigurazione del singolo flusso.

Una volta definito il percorso del flusso multimediale, l’applicazione utente che ne usufruisce può, e in genere lo fa, controllare l’andamento della sessione per quanto riguarda la QoS.
In tal caso, sarà prevista l’esistenza di un QoS manager che controlla le risorse assegnate ed, eventualmente, segnala una situazione di allarme, attivando la riconfigurazione per il flusso. Il processo di riconfigurazione potrà quindi consistere in diverse azioni, come ad esempio, l’adozione di un filtro, la selezione di una ogget-

to multimediale equivalente ma di diverso formato, il cambiamento del percorso del flusso, ecc..
Tale forma di controllo sul singolo flusso può però essere inadatta in alcune circostanze; nel caso infatti, in cui siano presenti su uno stesso nodo del sistema diversi flussi, l'eventuale carenza di risorse del Place sarebbe rilevata da tutti i relativi QoS manager, attivando così la riconfigurazione di tutti i flussi mentre sarebbe più ragionevole se venisse identificato un unico flusso da riconfigurare e gli altri aspettassero di verificare lo stato delle risorse dopo la riconfigurazione del flusso scelto. In questo modo si eviterebbero, per esempio, situazioni di "fuga di massa" e problemi di oscillazione dello stato sistema.

5.1.2.3 Riconfigurazione di un insieme di flussi.

Come abbiamo visto, se si verifica il caso per cui le risorse disponibili di un nodo in cui sono presenti più flussi, scarseggiano, nasce l'esigenza di riconfigurare non il singolo flusso quanto l'insieme dei flussi passanti per il nodo stesso. Ovviamente questo non è che un caso particolare di una riconfigurazione di un insieme determinato di flussi del sistema. In generale, a meno che non vi sia una fase di coordinamento tra le applicazioni utenti, la gestione della fase di riconfigurazione per un insieme di flussi può essere attuata esclusivamente dal sistema stesso, mentre nei due casi precedenti poteva essere svolta direttamente dall'applicazione utente; In particolare, il sistema dovrà svolgere le seguenti funzioni:

1) Monitor delle risorse del sistema; parallelamente ai QoS manager presenti nelle applicazioni utenti, anche il sistema dovrà impegnarsi nel monitoraggio delle risorse. La presenza di un proprio monitor delle risorse consente infatti di implementare politiche più flessibili e di entrare in azione prima che le applicazioni stesse ne segnalino l'esigenza. Facciamo comunque presente, nel caso del middleware MUM, è già presente un monitor delle risorse a livello del middleware e che, a questo si

appoggiano, per esempio per il controllo della CPU, i QoS manager delle applicazioni.
2) Identificazione della miglior politica di riconfigurazione possibile in funzione della/e risorsa/e carenti.
3) Identificazione dell'insieme di flussi maggiormente adatto alla Riconfigurazione scelta.
4) Attivazione e gestione della riconfigurazione per l'insieme di flussi individuato.

Lo svantaggio evidente di tale modalità di riconfigurazione, è l'estrema complessità del sistema; in particolare, l'individuazione dell'insieme di flussi, la loro riconfigurazione simultanea e la conseguente riallocazione delle risorse, è un problema di sicuro interesse nell'ambito sia della programmazione distribuita che dell'intelligenza artificiale e tuttavia esula dall'ambito ristretto di questo progetto nel quale, questa modalità di riconfigurazione verrà presa in considerazione ma con le forti limitazioni esposte nell'esempio del paragrafo precedente; in sostanza, si considererà l'insieme di flussi da riconfigurare come l'insieme dei flussi presenti sul nodo in cui è stata segnata una carenza di risorse e di questi verrà scelto un particolare flusso e attuata la fase di riconfigurazione esclusivamente per questo.

5.1.2.4 Concorrenza tra gli eventi di riconfigurazione.

Concludiamo con un'osservazione sul caso della concorrenza dei tre tipi di riconfigurazione possibili. Queste sono in genere concorrenti. E' cioè possibile che siano attuate in contemporanea (non relativamente allo stesso flusso però).
Se da un punto di vista teorico, ciò è lecito e non costituisce un problema. Da un punto di vista pratico, ciò non è utile.
Trascurando il caso in cui è l'utente a richiedere la riconfigurazione dei componenti del flusso, negli altri due casi, il motivo scatenante è, generalmente, lo stesso: viene rilevata una carenza di risorse. In questo caso, se lo stesso stato delle risorse genera entrambi i tipi di riconfigurazione, la presenza del meccanismo di riconfigurazione globale si rivela inutile. Tale meccanismo dovrebbe, in-

fatti, attivarsi con un buon anticipo rispetto a quello dei singoli flussi. In modo tale che questi entrino in azione solo se, nonostante i tentativi svolti dal sistema, le risorse continuino a non essere sufficienti (per esempio nel caso in cui un flusso prioritario, si appropri sempre di tutte le risorse che si liberano). Nella pratica però saranno le singole applicazioni utenti a fissare le soglie per le risorse necessarie. Vale quindi la pena di fissare delle soglie di trigger, per il meccanismo di sistema, piuttosto basse, in modo tale che questo agisca anticipando le richieste delle applicazioni.
Questa scelta, d'altro canto, ha uno svantaggio evidente; con soglie basse per la riconfigurazione, si rischia di attivare una fase di riconfigurazione senza che ve ne sia la reale necessità. Questo svantaggio può essere parzialmente colmato dividendo la fase di riconfigurazione in due parti, come vedremo.

5.1.3 Fasi della Riconfigurazione.

Come abbiamo anticipato in chiusura del precedente capitolo, la fase di riconfigurazione viene suddivisa in due parti; una prima fase, che chiamiamo fase di preparazione, in cui viene preparata la configurazione alternativa del sistema e una seconda fase, che chiamiamo fase di attivazione, in cui, o tale configurazione diventa effettiva o in cui il sistema elimina i cambiamenti che sono stati preparati.
Le due fasi andranno attivate quindi in sequenza in caso di un peggioramento delle condizioni del sistema; avremo quindi due soglie: la prima genererà un segnale di Attenzione e la seconda genererà infine un segnale di Allarme. Superata la prima soglia, il sistema attiverà la fase di preparazione. Superata la seconda, i cambiamenti attuati in fase di preparazione verranno realizzati. Nel caso invece che, una volta segnalato lo stato di Attenzione, lo stato delle risorse migliori, i cambiamenti generati durante questa fase dovranno essere annullati, in questo caso si attiverà una fase che chiamiamo di roll back (usiamo questo nome perché di fatto, in questa fase, devono essere cancellati le azioni compiute, con suc-

cesso, nella fase di preparazione; il sistema ritorna indietro quindi: effettua cioè un roll-back verso il precedente stato stabile; mentre lo stato, in cui i cambiamenti sono preparati ma non attuati è sempre uno stato transitorio). L'annullamento di un piano di riconfigurazione già preparato non avverrà però immediatamente, nel caso che le condizioni del sistema migliorino in quanto si cercherà di verificare che tale miglioramento sia "stabile". In particolare, se le condizioni del sistema si manterranno sufficientemente buone (cioè tali da non generare né segnali di attenzione né di allarme), per un numero stabilito di controlli successivi, allora lo stato di attenzione rientrerà e i cambiamenti preparati nella prima fase di riconfigurazione verranno annullati.

Per la gestione di questa evenienza esisterà un terzo segnale che avvertirà il sistema della possibilità di far rientrare lo stato di attenzione. Naturalmente se tale segnale dovesse essere generato mentre la fase di preparazione non è ancora terminata, il sistema dovrà tenerne conto e una volta terminata tale fase procedere immediatamente alla fase di roll back.

Naturalmente è anche possibile che da uno stato di disponibilità, si passi direttamente ad uno stato di Allarme. In questo caso, la fase di preparazione e realizzazione verranno entrambe attuate in ordine. Infine è possibile che ognuna delle fasi di riconfigurazione termini con successo oppure fallisca a causa di elementi non previsti (per es. la caduta di un nodo, il blocco delle comunicazioni, ecc..); poiché il fallimento di una delle tre fasi potrebbe lasciare il sistema in uno stato non consistente, è necessario prevedere e gestire, all'interno delle fasi stesse tale possibilità, poiché, infatti, in caso di fallimento di una delle fasi, lo stato non è prevedibile a priori, e non è possibile quindi attuare la fase di roll back.

Oltre a ciò, infine, facciamo presente che ogni fase impiega un certo tempo, non prevedibile a priori, per essere completata. Relativamente a tali fasi, lo stato delle risorse, e quindi i segnali di Allarme e Attenzione sono asincroni e il sistema ne deve quindi eventualmente tenere traccia. Nel diagramma in figura 5.1 riportiamo per maggiore chiarezza le possibili transazioni di stato e,

nella tabella di figura 5.2, indichiamo l'effetto dei vari segnali a seconda delle varie fasi.

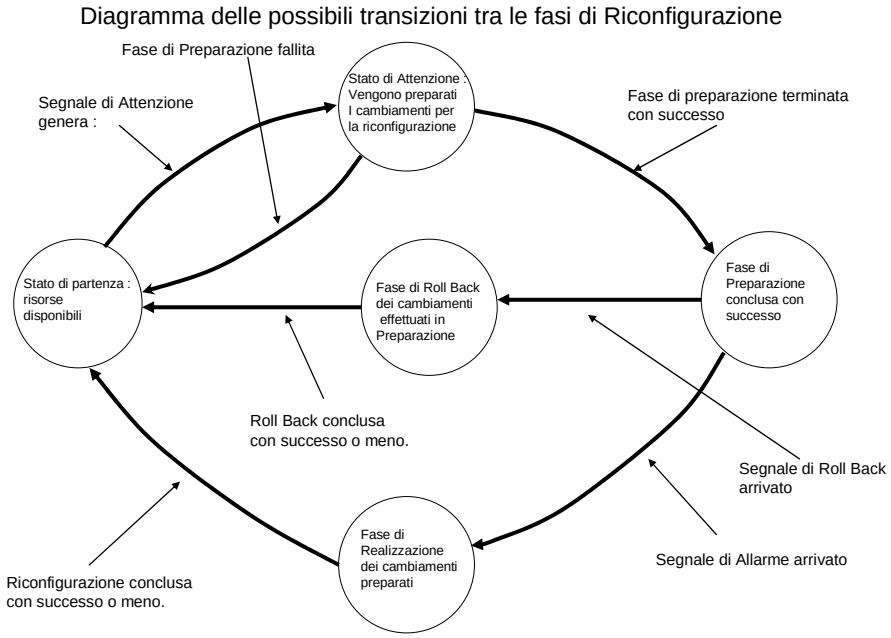


Fig. 5.1

Da notare che, secondo il diagramma, durante la fase di roll back, e la fase di attuazione dei cambiamenti preparati, i segnali di Attenzione e Allarme non sono considerati né vengono registrati per essere presi in considerazione al termine della fase (all'atto pratico essi verranno infatti disabilitati fino alla fine della fase di tali fasi); questa scelta è stata adottata in quanto, se si considera che il segnale di roll back viene generato solo se, per un certo tempo, lo stato delle risorse si è mantenuto sotto i valori di soglia, il nuovo segnale, di attenzione o di allarme, non è più correlato a quello iniziale e quindi non avrebbe senso riprendere la stessa azione di riconfigurazione ma deve esserne elaborata una nuova. Per quanto riguarda invece la mancata sensibilità a questi segnali in fase di attuazione dei cambiamenti preparati, essa è dovuta al fatto che sembra essere ragionevole, prima di attuare una seconda fase

di riconfigurazione, verificare quali sono le condizioni del sistema, al termine della prima. Questi segnali, inoltre, non vengono memorizzati dal sistema; in questo modo, si costituisce infatti un filtro per eventuali segnali sporadici non connessi ad uno stato di reale attenzione / allarme.

EFFETTO DEI SEGNALI NEI VARI STATI

Stato del Sistema / Segnale Ricevuto	Nessun evento segnalato	Preparazione In atto	Preparazione Conclusa con successo	Realizzazione dei cambiamenti In atto	Fase di Roll Back in atto
Segnale di Attenzione	Inizia la fase di Preparazione	Nessun effetto	Nessun effetto	Nessun effetto	Nessun effetto
Segnale di Allarme	Inizia la fase di Preparazione e di seguito quella di Realizzazione	Terminata la fase di Preparazione attiva quella di Realizzazione	Attiva Immediatamente la fase di Realizzazione	Nessun effetto	Nessun effetto
Segnale di Roll- Back	Nessun effetto	Terminata la fase di Preparazione attiva quella di Roll Back	Attiva Immediatamente la fase di Roll Back	Nessun effetto	Nessun effetto

Fig. 5.2

5.2 Modello di descrizione dei flussi.

Prima di presentare le entità che realizzeranno le fasi di riconfigurazione introdotte nel precedente paragrafo, è necessario delineare il modello di descrizione dei flussi che intendiamo adottare. La creazione di un modello dei flussi, è, infatti, fondamentale per definire quelle che sono le risorse a cui si deve prestare attenzione; inoltre per poter effettivamente adattare un flusso di informazioni alle condizioni del sistema, è necessario stabilire quelle che sono le unità fondamentali del flusso sulle quali si può agire. Precisiamo, che il modello che intendiamo usare per descrivere i flussi, non è

un modello general purpose, ma è ristretto ed orientato alle azioni che si possono intraprendere sul flusso stesso. In questo senso, non viene data una caratterizzazione fisica del flusso quanto piuttosto una descrizione dell'elaborazione a cui il flusso è soggetto.
In particolare, vogliamo realizzare la descrizione dei singoli flussi in funzione di un numero finito di elementi concreti sui quali sia possibile operare per ottenere la riconfigurazione del flusso stesso.
La scelta del modello da adottare nella descrizione di uno stream multimediale naturalmente può, infatti, dipendere da vari fattori, (per es. le specifiche dei protocolli utilizzati, il livello a cui questi operano, il livello di astrazione che si vuole avere, ecc..). Se potessimo agire a livello del sistema operativo, gestendo le risorse in ogni singolo nodo attraversato dal flusso, potremmo certamente considerare i singoli nodi come l'elemento di base per il nostro modello. Naturalmente ciò significherebbe agire a un livello molto basso di programmazione e in realtà le soluzioni sarebbero attuabili solo all'interno di reti locali o dedicate. Se si decide invece di creare un modello, a un livello di astrazione superiore, si riesce a compensare lo svantaggio di non poter lavorare a livello dei nodi lungo il percorso effettivo del flusso, con i vantaggi di una maggior interoperabilità e portabilità. Per quanto riguarda il modello che noi intendiamo utilizzare in questo progetto, esso prende spunto dal modello presentato nel capitolo due e dalla teoria del service routing.
Obiettivo del service routing è quello di ottimizzare il percorso di un flusso di dati attraverso diversi servizi messi a disposizione nel sistema. In sostanza, è possibile vedere un nodo come un insieme di servizi, ognuno dei quali descritto in termini della elaborazione che svolge, delle risorse che richiede, dei ritardi che produce, ecc..
Da qui, si cerca, tenendo anche conto di quelli che sono eventuali vincoli di priorità, di costruire un percorso che minimizzi/massimizzi una funzione di valutazione. Dal nostro punto di vista invece, all'atto della creazione del percorso del flusso, tale problema non sussiste in quanto i servizi eventualmente richiesti sono realizzati come componenti mobili, che è possibile attivare su qual-

siasi nodo del sistema, eventualmente scaricandone il software; può comunque rimanerci quella che è l'idea di base del modello e cioè la descrizione dei flussi in termini dei componenti/servizi che esso utilizza nel suo percorso.
In particolare, definiamo come unità di base per la descrizione di un flusso multimediale, l'oggetto MultimediaActiveComponent.
Questo oggetto rappresenta nel sistema, l'astrazione per un qualunque componente che svolga un elaborazione su un flusso, impegnando le risorse del sistema stesso. In particolare, quando un flusso viene creato e vengono disposti gli elementi necessari per la sua elaborazione/delivery lungo il percorso, verranno anche creati e registrati nel sistema, i rispettivi MultimediaActiveComponent che ne specificheranno le caratteristiche.
Riferendoci al caso del middleware MUM, anche se è vero che, in ogni Place il ResourceBroker, che si occupa appunto della prenotazione e dell'assegnazione delle risorse del Place, tiene traccia delle richieste generate dai componenti presenti sullo stesso Place, tali richieste si esauriscono in una pura e semplice specificazione delle risorse necessarie al componente installato ma non danno nessuna ulteriore informazione sul componente. Con l'introduzione del MultimediaActiveComponent, il nostro obiettivo è quello di creare un oggetto che realizzi un'astrazione totale, per il sistema, del componente, specificandone ogni possibile aspetto. In particolare dovrà fornire informazioni sul tipo di elaborazione che l'oggetto svolge sul flusso e le proprie specifiche e necessità in caso di adattamento e riconfigurazione, dovrà inoltre costituire un ponte per il sistema in modo che sia possibile, operando su questo oggetto, agire anche sul componente.
Come oggetto per la rappresentazione di un componente, tuttavia, è troppo generale per risultare direttamente utile. Si deve quindi procedere ad una specificazione dei MultimediaActiveComponent nei termini delle tipologie di componenti/servizi che è possibile identificare nel sistema. Intendiamo cioè identificare le tipologie di servizio che è possibile attuare, genericamente, nel sistema; da un

analisi immediata, risultano chiaramente evidenti che tali tipologie sono almeno tre e ovviamente sono :
il servizio di elaborazione/presentazione del flusso dalla rete,
il servizio di generazione del flusso e invio sulla rete,
il servizio di elaborazione generica del flusso dalla rete e ri-invio sulla rete.
Ovviamente questi tre servizi corrispondono in linea di massima a componenti di tipo Client, Server e Proxy.
Queste tipologie di servizio, pur essendo accomunabili in quanto richiedono risorse al sistema e svolgono un elaborazione sul flusso, non possono essere completamente descritte se non tendendo conto delle rispettive diversità; è quindi necessario introdurre i relativi ActiveComponent :
ActiveClient,
ActiveServer,
ActiveProxy.
Oltre a questi componenti è però possibile identificare un quarto tipo di componente che può essere presente nel sistema. L'idea, è quella di un componente che generi un flusso ma non lo invii sulla rete: che svolga, cioè, l'elaborazione e la presentazione del flusso in locale. Naturalmente, tale servizio è un ibrido di quelli presentati e in particolare lo si può sempre pensare come composto da un servizio di Server e da uno di Client (ed eventualmente anche un Proxy), tale scomposizione però è valida solo a livello teorico. Nella realtà è infatti possibile che un'applicazione, scoprendo che il file multimediale a cui è interessata si trova nello suo stesso Place decida di aprire una sessione locale invece di attivare un Server, creare un client, e metterli in comunicazione. In tali casi è, inoltre, possibile attuare politiche di riconfigurazione diverse dal caso distribuito, vale la pena di considerare la presenza nel sistema di possibili componenti a se stanti. Tali componenti saranno quindi descritti da uno : StandAloneComponent, che insieme agli altri ActiveComponent, specifica ogni tipo di possibile elaborazione su un flusso multimediale (nell'ambito del nostro progetto, i componenti di questo tipo, sono stati utilizzati semplicemente per effet-

tuare una prenotazione di risorse su un nodo e quindi, avere un modo per attivare la fase di riconfigurazione dei flussi a piacere). Abbiamo quindi ridotto il problema della definizione di un flusso a quello della definizione e caratterizzazione dei servizi/elaborazioni svolte sul flusso stesso e identificato quattro tipologie di servizio possibile (vedi figura 5.3 e 5.4).

Specificazione del Multimedia Active Component per i componenti del Sistema

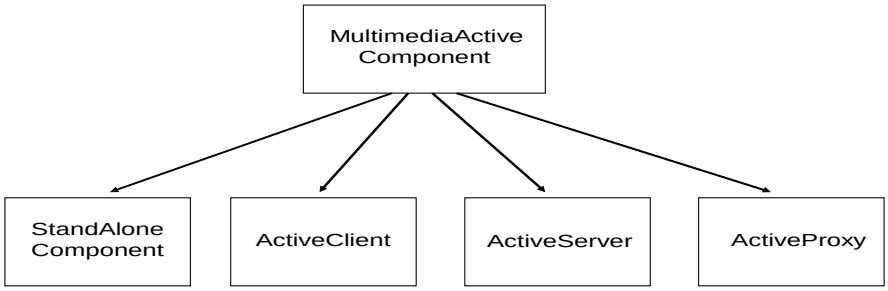


Fig. 5. 3

Prendiamo ora brevemente in considerazione le caratteristiche dei singoli ActiveComponent; la loro struttura è tuttavia molto legata

Associazione tra un componente e il relativo metadato (Active Component relativo)

MultimediaActive Component	Oggetto che identifica un generico servizio svolto su un Flusso multimediale
ActiveClient	Oggetto che identifica un servizio di presentazione di un Flusso multimediale ottenuto dalla rete
ActiveServer	Oggetto che identifica un servizio di generazione e invio di un Flusso multimediale sulla rete
ActiveProxy	Oggetto che identifica un servizio di ricezione, elaborazione e invio di un Flusso multimediale sulla rete
StandAlone Component	Oggetto che identifica un componente che non comunica con altri, in quanto il flusso multimediale è generato e consumato localmente; viene comunque registrato nel Sistema per le risorse occupate e come oggetto di eventuale riconfigurazione.

Fig. 5.4

all'implementazione del sistema sottostante, pertanto verranno qui solo introdotti mentre saranno specificati nel dettaglio nella parte relativa all'implementazione realizzata.

5.2.1 Il Multimedia Active Component.

Come abbiamo detto, un multimediaActiveComponent è un oggetto che rappresenta l'astrazione di un qualunque servizio multimediale che possa essere attuato nel sistema. Pur essendo troppo generico per un uso diretto, esso definisce la struttura di base per gli altri oggetti che rappresentano astrazioni di componenti del sistema ed è quindi la radice comune per la definizione di tali oggetti.

Nel sistema, ogni possibile componente attivo dovrà essere caratterizzato in funzione delle possibili azioni che è possibile attuare su di esso in fase di riconfigurazione; poiché questa è una caratteristica generale di tutti i possibili componenti, l'oggetto che manterrà tali informazioni è presente nel MultimediaActiveComponent (verrà poi anch'esso specializzato a seconda del componente creato). Tale oggetto viene chiamato: ActionBox.

Lo scopo dell'ActionBox è quindi mantenere le informazioni necessarie per sapere come agire sul componente che realizza un particolare servizio, a seconda della situazione. All'atto della creazione di un qualunque active component, viene quindi creata la relativa ActionBox, specificando il tipo di componente in cui è inserita; l'ActionBox verrà quindi generato, a seconda, del compito che il componente attua ed è una vera e propria "scatola" che contiene la descrizione dei comportamenti realizzabili per il componente; in particolare, per ogni ActionBox, viene stabilito :

1) l'azione di riconfigurazione di default del componente
2) il set di azioni di riconfigurazione che non possono essere attuate sul componente.

Sulla base di tali azioni, sarà poi stabilito come agire (e se agire) sul componente. Oltre alla presenza dell'ActionBox, il Multimedia ActiveComponent non è ulteriormente specificato.

5.2.2 Active Client.

L'ActiveClient è l'oggetto che per il sistema definisce l'astrazione di un componente Client; in particolare, la struttura di ogni oggetto ActiveClient dovrà contenere almeno i seguenti oggetti :

1) un oggetto ActionBox.
   Le azioni di riconfigurazione permesse su entità client in realtà sono molto limitate; non si può, in genere, pretendere che un Client si sposti su un altro nodo, è però possibile intervenire introducendo a monte dei filtri o in altri modi, comunque sia, poiché attualmente lo scopo del progetto non è quello di definire le migliori azioni di riconfigurazione nei possibili casi ma di delineare l'infrastruttura per il sistema di riconfigurazione, l'azione di riconfigurazione di Default è nessuna azione e il set di azioni proibite per il client è vuoto.
2) un riferimento all'applicazione cliente che consenta di operare direttamente su questa.
3) un riferimento al componente successivo nel flusso. Tale riferimento è importante per poter eventualmente interagire con gli altri componenti del flusso.

5.2.3 Active Proxy .

L'oggetto ActiveProxy è utilizzato nel sistema come astrazione di un componente Proxy. Un componente proxy è un componente che preleva un flusso dalla rete, esegue una generica elaborazione su questo, e ripropone il flusso alla rete. Per quanto riguarda la possibilità di adattare i flussi alle condizioni del sistema e di riconfigurare il sistema stesso, i componenti Proxy sono sicuramente quelli maggiormente coinvolti. Poiché per questo tipo di componenti, è necessario tenere bene in considerazione il tipo di elaborazione che il proxy svolge, in ogni ActiveProxy dovrà esserne tenuta traccia.

A prescindere comunque dal tipo di elaborazione svolta, un Active Proxy mantiene le seguenti informazioni :

1) un oggetto ActionBox
2) un riferimento all'applicazione che svolge l'elaborazione del

flusso
3) un oggetto che consenta l'identificazione dell'applicazione utente per cui si elabora il flusso.
4) un riferimento al componente successivo nel flusso.
5) un riferimento al componente precedente nel flusso.

5.2.4 Active Server.

L'oggetto ActiveServer è utilizzato dal sistema come astrazione di un servizio offerto dal sistema; Una volta attivato, un servizio è realizzato, in genere, mediante un thread demone che attende una o più richiesta e le esegue, in generale in multithreading, prelevando o creando un flusso sul Place e inviandolo sulla rete. Ciò significa che un server, a meno di non servire un unico client, non gestisce un solo flusso, ma ne avrà uno per ogni client con cui è in contatto. In particolare, un ActiveServer dovrà quindi mantenere le informazioni relative a ogni client, consentendo di operare singolarmente su uno di essi (su uno dei flussi). In generale, è comunque vero che, i server, non prendono parte alla riconfigurazione del sistema a meno che non il client non opti verso un'altra presentazione a qualità inferiore/superiore, o che il sistema non prenda in considerazione la chiusura forzata di un flusso verso uno dei client serviti (per liberare risorse necessarie).
Abbiamo quindi che un ActiveServer contiene :
1) un oggetto che definisce univocamente il servizio svolto.
2) un oggetto ActionBox.
3) un vettore contenente le informazioni relative ai singoli client che sta servendo; in particolare, per ogni client, dovrà essere presente:
    1) un oggetto che identifichi l'applicazione utente che si serve
    2) l'identificatore del componente attivo precedente, lungo il percorso del flusso.
    3) l'identificatore dell'agente client avviato dall'utente.
    4) un riferimento al thread che gestisce il flusso dati verso il client.

5.2.5 Stand Alone Component.

Gli StandAloneComponent rappresentano, nel sistema, l'astrazione per componenti dalla natura ibrida di Server e Client. Per questi componenti, il flusso multimediale viene generato e consumato localmente. In caso di riconfigurazione, le possibili azioni su un componente di tipo StandAlone sono abbastanza limitate, anche se è sempre possibile decidere di chiudere forzatamente la sua sessione per liberare le risorse impegnate.
Ogni oggetto StandAloneComponent dovrà mantenere :
1) un riferimento all' applicazione utente rappresentata dallo Stand Alone Component.

5.3 I Protocolli di Riconfigurazione.

Come abbiamo anticipato, nel modello che intendiamo realizzare, il sistema non prende in considerazione, né implementa, un particolare protocollo per la riconfigurazione dei flussi; esso fornisce però gli strumenti per la gestione di oggetti che incapsulano al loro interno tali protocolli. Tali oggetti sono :
Azioni di Riconfigurazione (Reconfiguration Action), e Azioni di Riconfigurazione Remota (Remote Reconfiguration Action) .

5.3.1 Reconfiguration Action.

Nell'ambito della riconfigurazione di un flusso, la determinazione di un unico protocollo, adatto ad ogni circostanza, e per ogni componente, non consente una politica di riconfigurazione realmente efficace.
Quando ci si occupa della riconfigurazione di un flusso su cui viene svolta una generica elaborazione è in realtà necessario, tenere in considerazione tre aspetti :
1) il tipo di evento che ha generato la fase di riconfigurazione.
2) le esigenze del componente da riconfigurare in termini di risorse
3) il tipo di elaborazione svolta dal componente.

Al variare di questi tre parametri dovrà quindi essere identificato il protocollo di riconfigurazione più adatto al caso.
Una Reconfiguration Action intende essere, in questo senso, l'astrazione per un protocollo di riconfigurazione allo stesso modo in cui un MultimediaActiveComponent è l'astrazione di un componente generico del sistema.
Poiché i protocolli idealmente attuabili in fase di riconfigurazione possono essere diversi, allo stesso modo dovranno esistere diversi tipi di ReconfigurationAction i quali però dovranno presentare la stessa interfaccia e struttura; in particolare ogni Reconfiguration-Action, dovrà :

1) indicare quale tipo di protocollo di riconfigurazione esegue.
2) fornire un metodo che prepari il sistema alla riconfigurazione. Creando una configurazione alternativa del sistema ma non rendendola attiva.
3) fornire un metodo che renda definitive le modifiche attuate in fase di preparazione.
4) fornire un metodo che annulli le modifiche attuate in fase di preparazione.
5) essere in grado di indicare quale componente è soggetto alla riconfigurazione.

5.3.2 Remote Reconfiguration Action.

Simili alle Reconfiguration Action, sono le : Remote Reconfiguration Action.
Come abbiamo visto, una Reconfiguration Action è l'astrazione per un generico protocollo di riconfigurazione. Il sistema, all'inizio della fase di riconfigurazione, determina quale azione di riconfigurazione attuare e sceglie un componente, in modo che questa azione venga attuata per il componente scelto.
In generale, questo avviene localmente. Con questo intendiamo dire che, il componente, per cui viene attuata la riconfigurazione, viene scelto nello stesso luogo in cui viene attivata tale richiesta. In questo senso, intendiamo che la riconfigurazione sia locale (naturalmente, il fatto che la riconfigurazione sia locale, non significa

che non coinvolga Componenti su altri Place ma che il componente locale è quello maggiormente coinvolto) .
In alcuni casi, però, come, per esempio, per alcuni tipi di riconfigurazione da richiesta del client, la riconfigurazione non coinvolge principalmente il componente locale ma deve avvenire in remoto.
Un tipico esempio è dato dalla richiesta di cambio di formato dello stream presentato per poter ridurre le esigenze di banda. Un manager di QoS che avverta una diminuzione eccessiva della banda trasmissiva infatti, può richiedere che venga creato, il più a monte possibile, un proxy che esegua una conversione verso un formato che occupi meno banda in trasmissione. In questo caso, la riconfigurazione dei componenti del flusso, non coinvolge direttamente il componente che l'ha attivata ma il componente più a monte possibile nel flusso. Un altro caso è il caso di un terminale mobile che entri in una nuova cella (o per semplificare, si registri in un altro dominio); anche in questo caso, l'esigenza di riconfigurazione, viene attivata sul client ma il componente per cui questa deve essere attuata è il Proxy che effettua il forward del flusso dall'ultima postazione fissa (dal dominio) che deve spostarsi verso la nuova cella(verso il nuovo dominio). E' necessario quindi che questa richiesta generi una RemoteReconfigurationAction, cioè un'azione di riconfigurazione che verrà attivata non localmente.
Tale oggetto, oltre alle specifiche di una Reconfiguration Action dovrà anche consentire di:

1) indicare dove spostarsi.
2) indicare il nodo in cui ci si trova è adatto (quello giusto) per attuare la riconfigurazione.
3) indicare se è necessario creare un nuovo componente o è possibile agire su un componente già presente.
4) creare, in caso di necessità, un nuovo componente necessario per attuare la riconfigurazione.
5) indicare il nodo di origine

5.3.3 Possibili Azioni di Riconfigurazione locali e remote.

Per completezza, indichiamo ora alcuni possibili protocolli realizzabili mediante le azioni di riconfigurazione. La seguente lista non intende essere ovviamente esaustiva dei possibili protocolli realizzabili, ma semplicemente dare un'idea della versatilità che le azioni di riconfigurazione rendono possibile.

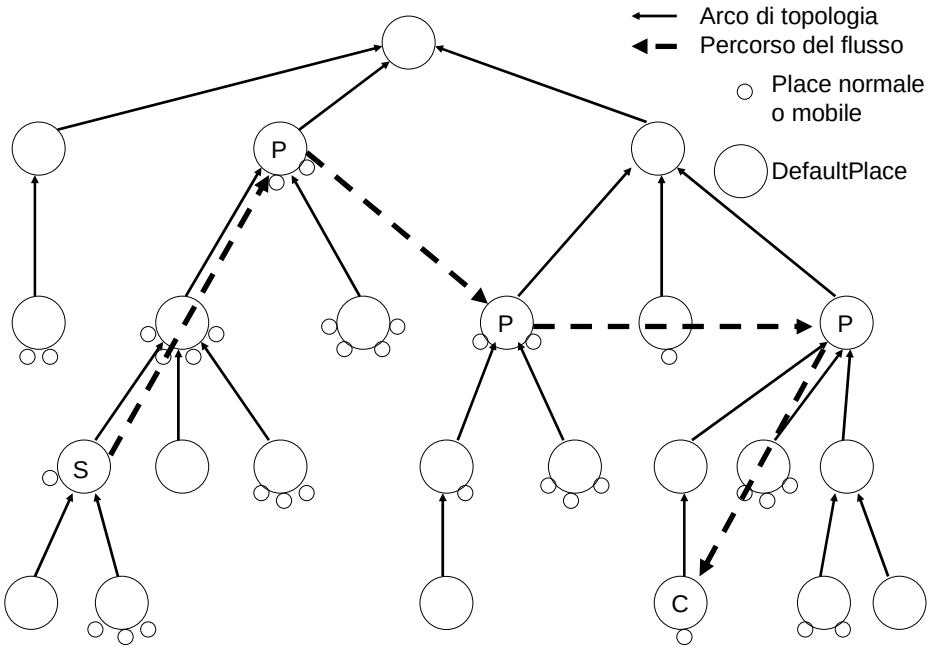
1) Nessuna operazione di riconfigurazione.
   Il caso più semplice in assoluto, il middleware non opera alcuna riconfigurazione ma attende per vedere se la situazione non migliora da sé.
2) Salta il componente. Riferito a un componente di tipo Proxy.
   Se il componente svolge un'elaborazione di cui non si sente particolarmente la necessità, il sistema può decidere che non vale la pena mantenerlo e che si possano liberare le risorse a lui dedicate. In tal caso, il sistema metterà in comunicazione diretta il componente a monte con quello a valle del componente scelto.
3) Muovi il componente. Riferito a un componente di tipo Proxy.
   E' il tipico caso di riconfigurazione del percorso dello stream. Il componente viene spostato in un Place diverso da quello in cui si trova attualmente.
4) Crea un nuovo percorso. Riferito a un componente Client
   può rappresentare il caso di un utente nomadico o di un dispositivo wireless. Identificata la destinazione del Client, viene creato un percorso che congiunge tale destinazione con l'attuale percorso del flusso.
5) Adotta nuova presentazione equivalente. Riferito a un componente Proxy. Viene cercata una presentazione equivalente, ma per esempio in un Server con maggiori risorse, oppure una versione in un diverso formato. Viene quindi attivato il percorso tra uno dei componenti già presenti del flusso e il luogo in cui risiede tale riconfigurazione.
6) Inserisci Nuovo Componente. E' il caso in cui, sia necessario, o si desideri applicare una qualche elaborazione/filtro al flusso. Verrà creato un nuovo componente che svolge l'elaborazione richiesta e verrà inserito nel flusso.

Pur non comprendendo ogni possibile protocollo di riconfigurazione, queste azioni di per sé potrebbero già garantire una copertura sufficiente per la maggior parte dei possibili casi.
Notiamo inoltre che l'insieme dei protocolli di riconfigurazione che necessitano la valutazione dello stato delle risorse di altri siti, e che quindi, necessitano l'interazione con il sistema di località, sono solo un sottoinsieme dei possibili protocolli realizzabili; essi tuttavia rivestono un'importanza notevole nei casi pratici e pertanto giustificano largamente l'introduzione del sistema di località descritto nei precedenti capitoli. Nel prossimo paragrafo, prima di procedere alla definizione dell'architettura generale per il modello proposto, intendiamo presentare quindi un esempio di un possibile protocollo.

5.3.4 Protocollo per la scelta del percorso di un flusso nell'ambito di una località.

In questo paragrafo, intendiamo presentare un protocollo per il QoS routing di un flusso di dati, nell'ambito del sistema di località da noi realizzato. Si è deciso di presentare questo protocollo in questo capitolo e non nel capitolo precedente (dedicato alla discussione del sistema di località) perché esso si basa sul modello dei flussi qui introdotto e perché questo stesso protocollo può essere utilizzato in fase di riconfigurazione del sistema.
Lo scopo del protocollo è quello di determinare un percorso tra due entità: un client e un server, lungo il quale siano distribuiti le entità proxy necessarie; il flusso dei dati sarà quindi definito puntualmente dalla disposizione di questi componenti (vedi fig. 5.5).
Facciamo notare che non considereremo il caso in cui le entità coinvolte siano esclusivamente un client e un server in quanto, in questo caso, la comunicazione sarebbe diretta tra i due e non si potrebbe realizzare la gestione del percorso del flusso di dati che sarebbe demandato ai protocolli di livello inferiore. L'idea di base che ci interessa realizzare è quella di distribuire le entità che gestiranno il flusso (client, proxy e server) in maniera uniforme lungo il

percorso, cercando di distribuire l'occupazione delle risorse all'interno delle località, tra i vari nodi.

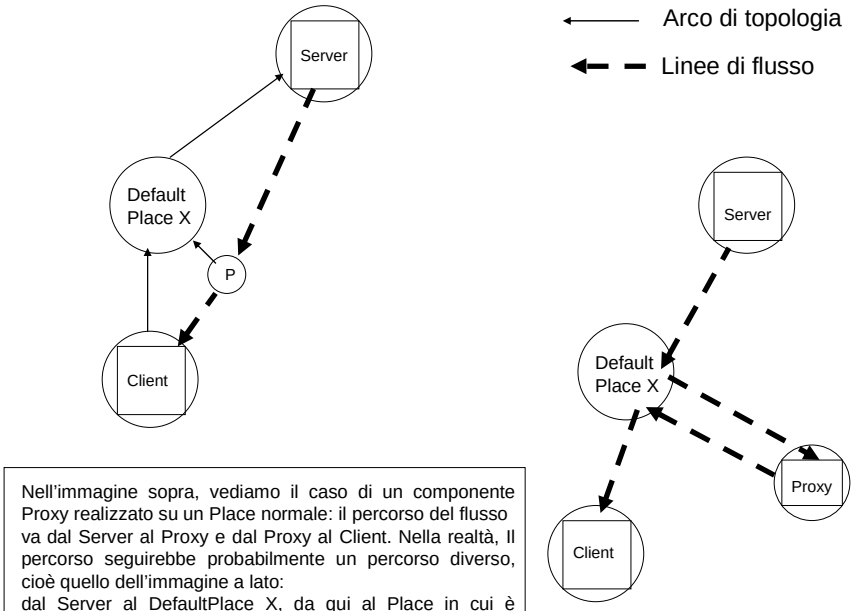


Esempio di un percorso per un flusso di dati, le lettere indicano i componenti Client, Proxy e Server

Fig. 5.5

Il primo aspetto che dobbiamo considerare, è la definizione di una metrica per la valutazione dei diversi Place. Le informazioni raccolte dal sistema di località sono relative a CPU, banda disponibile, memoria secondaria e primaria; queste informazioni definiscono quindi le grandezze che possono essere prese in considerazione nella valutazione di un nodo. Oltre a questo dobbiamo poi tenere in considerazione che i nodi, per la piattaforma SOMA, non sono tutti uguali; in particolare sono presenti, come abbiamo visto, tre tipi di nodi: i DefaultPlace, i Place normali e i Place mobili. La differenza tra questi nodi non è limitata esclusivamente alle capacità delle macchine ma anche alla loro natura; i Place mobili, per esempio, rappresentano siti la cui appartenenza a un dominio si può presupporre temporanea, per cui sono poco adatti ad essere coinvolti nel percorso di un flusso, anche se avessero disponibilità di risorse. I DefaultPlace possono invece rappresentare gateway o

firewall di una rete locale; in questo caso, se l'utente si trova nel loro dominio, il flusso di dati verso il client sarà comunque costretto a transitare attraverso il DefaultPlace e può quindi risultare conveniente disporvi un Proxy. Per quanto riguarda i Place normali invece, teniamo in considerazione che, se vale la considerazione fatta per i DefaultPlace, coinvolgere un Place normale nel percorso di un flusso, comporterebbe la necessità di passare dal Default Place presso cui il Place è registrato, per raggiungerlo e poi di ripassare attraverso lo stesso DefaultPlace, per uscire dalla rete locale (vedi fig. 5.6). Ciò comporterebbe non solo un allungamento del percorso effettivo (con eventuali problemi di ritardi) ma anche un'occupazione doppia di risorse di banda sul DefaultPlace a meno che, tali Place non siano nel dominio del client o nel dominio del server (in realtà la considerazione che abbiamo fatto continua a valere se il server o il client sono su un DefaultPlace e non su un Place del dominio).



Nell'immagine sopra, vediamo il caso di un componente Proxy realizzato su un Place normale: il percorso del flusso va dal Server al Proxy e dal Proxy al Client. Nella realtà, Il percorso seguirebbe probabilmente un percorso diverso, cioè quello dell'immagine a lato:
dal Server al DefaultPlace X, da qui al Place in cui è presente il Proxy, poi di nuovo verso il DefaultPlace X e infine al Client.

Fig. 5.6

In quest'ultimo caso, infatti, il flusso deve comunque entrare nel dominio e, anzi, visto che le comunicazioni intra dominio sono poco costose, è conveniente disporre qui i componenti proxy.
La scelta di un Place per la creazione di un componente allora avviene secondo il seguente schema:

1) Si ottengono dalla tabella di località le entry relative ai Place visti.
2) Si ordinano le entry così ottenute nei seguenti gruppi:
   1) i Place normali appartenenti allo stesso dominio del client
   2) i Place normali appartenenti allo stesso dominio del server
   3) i DefaultPlace che si trovano lungo il percorso tra client e server
   4) gli altri DefaultPlace visti nella località
   5) i Place normali appartenenti a domini diversi da quello del client e da quello del server
3) Si ordinano i gruppi così ottenuti (alcuni potranno anche essere vuoti) a seconda di dove è il componente precedente e quello successivo (se si dispone di questa informazione) altrimenti li si considera secondo il seguente ordine 3, 4, 1, 2, 5.
   Nel caso che si sappia invece la posizione dei componenti precedente e/o successivo (cosa che può accadere se si sta riconfigurando il flusso; in questo caso si può spostare un singolo componente mentre gli altri rimangono dove sono) si deve tenere conto del fatto che, se si è connessi con un componente all'interno di un dominio, conviene prima cercare tra i Place dello stesso dominio e sul relativo DefaultPlace e solo dopo riprendere la ricerca secondo l'ordine visto.
4) Si ricercano, all'interno del singolo gruppo i Place che meglio si adattano alle richieste di risorse del componente; per la scelta si adotta il principio del sequential filtering visto nel capitolo 2; in particolare si considerano le risorse nell'ordine banda, CPU, memoria primaria e memoria secondaria. Il criterio di scelta è di tipo può essere sia di tipo best-fit o wide-fit. Se nessun Place ha risorse sufficienti, si passa ai Place del gruppo successivo

5) Si ripete poi il processo per ogni altro componente, tenendo conto del fatto che, nel Place selezionato al punto 4), le risorse saranno diminuite.

Vediamo ora come funziona il protocollo ideato:

Le ipotesi di partenza sono le seguenti:

1) il flusso di dati comporta la presenza di K entità dove K > 2 e le entità terminali sono da un lato un client e dall'altro un server mentre le entità intermedie sono di tipo proxy.
2) La distanza, in termini di passi sulla topologia di SOMA tra client e server è pari a N.
3) Le località hanno tutte lo stesso raggio pari a R passi sulla topologia di SOMA.
4) Le soglie per la notifica degli eventi relativi allo stato delle risorse sono uguali in tutto il sistema; si considera quindi che due Place hanno le stesse risorse se i valori indicati non differiscono per un valore maggiore al valore della soglia per la risorsa in esame.
5) La definizione del percorso avviene partendo dal nodo client procedendo verso il server secondo il meccanismo previsto in MUM; i componenti sono cioè piazzati tramite VisitorAgent e il loro indirizzo è dato da ComponentInfo di MUM.
6) La posizione di server e client sono stabilite.

Per prima cosa si calcola il percorso, lungo la topologia tra client e server. Questo è sempre possibile, perché il DNS mantiene il percorso di ogni Place verso la radice. Il percorso, di lunghezza N, è della forma [$Place_{Client}$, $DefaultPlace_i$, .., $DefaultPlace_j$, $Place_{Server}$].

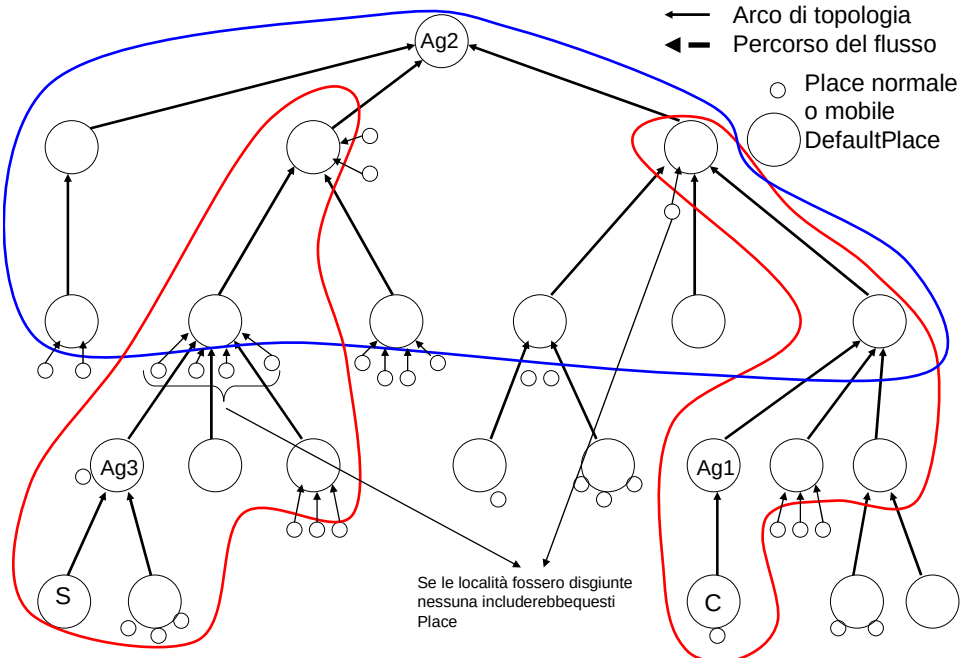
Si possono ora verificare tre casi:

1) R >= N,
2) N > R e N < 2R -1,
3) N > 2R -1.

Nel primo caso, il server si trova nella località del client e viceversa; si cercherà di disporre i componenti tutti nella stessa località (cioè quella del client). Il protocollo esegue, quindi, in questo caso, un source routing, in cui l'insieme dei nodi attraversati viene

definito dal nodo di partenza (anche se in questo caso è il nodo di arrivo del flusso quello in cui viene definito il percorso).
In particolare viene creato un piano in cui i componenti sono distribuiti tra i Place della località. Tenendo in considerazione che il primo elemento sarà il client e l'ultimo il server, che per ipotesi sono su Place prefissati, gli altri verranno stabiliti secondo il procedimento visto sopra. Il piano viene poi passato ad un VisitorAgent creato dal client secondo lo schema classico di MUM.
Consideriamo ora, invece, il secondo caso; questa volta, la distanza tra client e server è tale che questi non si vedono l'un altro nella propria località tuttavia, poiché la distanza tra i due è inferiore a 2R, per ipotesi esiste sempre un DefautlPlace la cui località comprenda sia il client che il server. In questo caso, il VisitorAgent creato dal client si sposterà sul DefaultPlace che si trova a distanza R-1 dal client, lungo il percorso client/server e qui agirà come visto nel caso precedente creando un piano per la disposizione dei componenti e affidandolo ad un altro VisitorAgent la sua esecuzione. Una volta che i componenti saranno disposti lungo la località e il primo VisitorAgent avrà ottenuto la ComponentInfo del primo componente non farà altro che riportarla al client.
Facciamo notare che esiste sempre una località che comprenda client e server anche quando la loro distanza è uguale a 2R; in questo caso però, se il client si trova su un DefaultPlace, in tale località (centrata sul DefautlPlace del percorso client/server a distanza R) non si ha visibilità dei Place normali del dominio del client; visto che in realtà, averne visibilità può essere conveniente si è deciso di limitare questo caso, a quando la distanza client/server sia strettamente inferiore a 2R.
Vediamo ora il terzo caso. La situazione che affrontiamo questa volta è più complessa perché visto che la distanza tra server e client è tale che non esiste una località che li comprenda entrambi.
L'idea è quella di identificare un numero definito di località lungo il percorso e di attivare i componenti necessari al flusso in tali località, distribuendoli in modo il più omogeneo possibile lungo il flusso. Si nota a questo punto che se consideriamo, lungo il

percorso client/server, località a distanza 2R l'una dall'altra, cioè località completamente disgiunte l'una dall'altra, i Place normali registrati presso i DefaultPlace ai bordi di tali località non sarebbero inclusi nel protocollo anche se topologicamente vicini al percorso client/server (vedi fig. 5.7). Per questo motivo consideriamo località a distanza 2R-1 l'una dall'altra.



Località di raggio 2; gli agenti Ag1,2,3 creano i piani per la disposizione dei componenti nelle rispettive località.

Fig. 5.7

Il numero di località (L) che dobbiamo considerare quindi ci è dato da: L = N / (2R – 1) approssimato per eccesso. La prima località che con-sideriamo, come nel caso precedente, è a distanza R-1 dal Place del client; le altre fino alla penultima sono a distanza 2R-1 dalla precedente. L'ultima invece dipende dalla distanza rimasta per arrivare al server: tale distanza è il resto della formula:
N/(2R – 1).
Se è superiore o uguale a R-1 allora consideriamo la località individuata dalla regola esposta precedentemente, mentre se è inferiore, consideriamo la località a distanza R+1 passi dalla precedente

(vedi fig 5.7; le località evidenziate sono centrate sui DefaultPlace in cui sono gli agenti Ag1,2,3 e la località di Ag3 è a R+1 passi da quella di Ag2); in questo caso la distanza tra client e server è di otto passi e il raggio delle località è 2, abbiamo che 8 / 3 ha come resto 2 (2 > 1), quindi, in questo caso, non ci sarebbe lo spazio per localizzare una località a fine percorso, a distanza 2R-1 dalla precedente, e si prende quindi in considerazione la prima località centrata su un Default Place esterno all'ultima considerata.
Una volta che si è stabilito su quante località si andrà ad operare e su quali DefaultPlace sono centrate (si riesce a stabilire su quali DefaultPlace queste località sono centrate perché si conosce il percorso client/server), si ottiene il numero di componenti da attivare per località dal rapporto M = K / L (naturalmente si considererà un'approssimazione del rapporto se questo è frazionario) e infine si passa alla parte operativa. Il client lancia un primo VisitorAgent verso il DefaultPlace su cui è centrata la prima località (in fig. 5.7 è Ag1); questo agente crea un VisitorAgent e lo invia al Default Place su cui è centrata la seconda località (in fig. è Ag2); il processo viene quindi ripetuto fino a che, per ogni località, non è presente un VisitorAgent. L'ultimo VisitorAgent creato, una volta arrivato sul DefaultPlace di propria competenza, creerà un piano per i suoi M componenti utilizzando il procedimento visto nei casi precedenti e attiverà un nuovo agente che avrà lo scopo di attuarlo. Riferendoci all'esempio di fig. 5.8, una volta che il piano creato da Ag3 è completato, Ag3 riceve la ComponentInfo del primo componente di questa parte di percorso (in questo caso viene attivato solo il Server e P1 quindi riceve la ComponentInfo di P1) e la passa ad Ag2. Ag2, a sua volta, crea un piano per determinare la posizione dei componenti che gli sono assegnati e attiverà un VisitorAgent per realizzarlo. In questo piano verrà anche inclusa la ComponentInfo che gli è stata segnalate (quella di P1). I componenti attivati da questo piano si creeranno un percorso e lo collegheranno al componente indicato dalla ComponentInfo indicata (in particolare, in fig. 5.8, a P2 sarà indicata la ComponentInfo di P1). Il processo verrà poi ripetuto

fino ad arrivare all'attivazione del client. Nel caso in fig. 5.8, una volta attivati P2 e P3, Ag2 riceverà la ComponentInfo di P3 e la passerà ad Ag1; Ag1 sempre secondo lo stesso procedimento attiverà P4 (a cui ha passato la ComponentInfo di P3) e infine notificherà al client C che ha concluso. Il client ha questo punto (attiverà il percorso) C, P4, P3, P2, P1, S secondo i classici meccanismi di MUM.

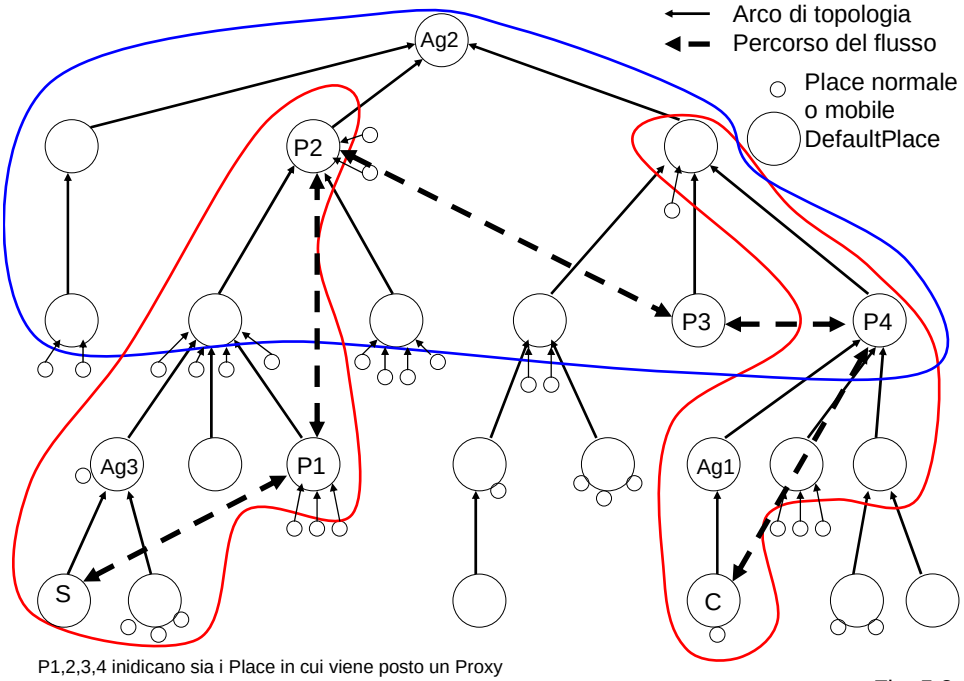
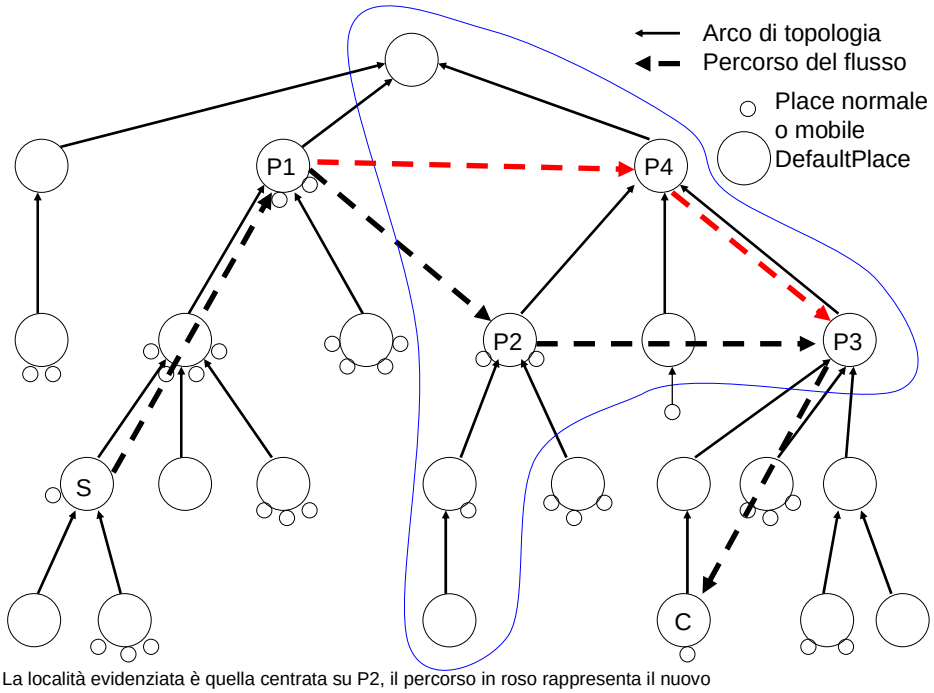


Fig. 5.8

Come abbiamo visto, in questo modo è possibile, ottenere una distribuzione abbastanza equilibrata dei componenti lungo le località attraversate dal percorso client/server. Nei termini introdotti nel capitolo 2, questo tipo di protocollo rappresenta un ibrido tra un protocollo di source routing e un protocollo distribuito cercando di coniugare i vantaggi di entrambi. In particolare, si limita il source routing alle località in modo da non appesantire eccessivamente questa fase dal punto di vista dell'elaborazione.

Un tema importante è rimasto però fuori dalla trattazione esposta: cosa succede, cioè, in caso che non si riesca ad attivare uno dei piani delle località. In questo caso sarebbero possibili, in realtà, diverse strategie come effettuare una rielaborazione del piano, passare i componenti non attivati all'agente che si trova nella località precedente, ecc..
L'intento di questo progetto di tesi non è tuttavia la definizione esatta di un protocollo: quello presentato vuole semplicemente essere un esempio dimostrativo di come sia possibile implementare un protocollo di QoS routing sulla base del sistema di località realizzato per cui, di tali argomenti non si tratterà (è da notare che un argomento altrettanto interessante sarebbe la definizione di un criterio per l'assegnazione degli M componenti alle località lungo il percorso).
Vediamo ora come questo protocollo possa essere usato in fase di riconfigurazione del sistema. In realtà è molto semplice. La riconfigurazione del sistema coincide di fatto con la riconfigurazione di un componente situato lungo il percorso di un flusso. In questo caso, se si considera la località centrata sul Place che ospita tale componente, si può direttamente applicare l'algoritmo si selezione di un Place visto a inizio capitolo, per determinare il Place più idoneo per lo spostamento del componente. In questo caso, se non si è conoscenza dell'esatto percorso client/server, si può considerare il percorso che congiunge direttamente i componenti precedente e successivo a questo che, per la struttura degli Active Component sono sempre noti. Nell'esempio di fig. 5.9 vediamo che viene attivata la riconfigurazione per il componente P2 del flusso, viene quindi selezionato un Place alternativo per il flusso all'interno della località centrata su P2 e viene quindi attivato il nuovo percorso.



La località evidenziata è quella centrata su P2, il percorso in roso rappresenta il nuovo tratto del percorso che non passa più da P2 ma da P4

Fig. 5.9

5.4 Architettura del modello.

Presentato un esempio di protocollo basato sul sistema di località e visto come possa essere utilizzato anche in fase di riconfigurazione dei flussi, presentiamo adesso l'architettura che realizza il modello di riconfigurazione presentato. Questa è composta fondamentalmente da quattro elementi :

- il sottosistema per la generazione dei segnali
- il reconfiguration planner
- il reconfiguration manager
- i reconfiguration listeners

L'insieme di questi quattro elementi consente, come vedremo, la gestione delle azioni di riconfigurazione locali e remote; di seguito introduciamo il ruolo e i compiti dei singoli componenti, che invece verranno dettagliati nel prossimo capitolo.

5.4.1 Il sottosistema per la generazione dei segnali.

Il sottosistema per la generazione dei segnali di riconfigurazione, è sostanzialmente composto da un componente che effettua un controllo periodico dello stato delle risorse sul nodo in cui si trova.
Tale componente verifica se lo stato delle risorse è entro le soglie fissate sia per lo stato di Attenzione che per lo stato di Allarme, se una delle due viene superata, viene generato il segnale corrispondente (se viene superata la soglia di allarme, il segnale di attenzione non viene attivato, perché il segnale di allarme lo comprende). Se viene generato il segnale di attenzione, il componente riprende il controllo delle risorse, evitando però di generare altri segnali di attenzione. Se per un numero prefissato di cicli, non viene superata nessuna soglia, viene generato il segnale di rientrato allarme; se invece, segnalato uno stato di attenzione, lo stato delle risorse peggiora, viene generato il segnale di allarme.

5.4.2 Reconfiguration Planner.

Il ReconfigurationPlanner è l'entità che si occupa di delineare l'azione di riconfigurazione da utilizzare, nel caso di riconfigurazione dell'insieme dei flussi. Esso fornisce all'esterno un solo servizio che consiste nell'elaborare la ReconfigurationAction.
Al Planner viene indicato lo stato delle risorse che ha generato la richiesta di riconfigurazione, il planner determina di conseguenza, a seconda delle risorse carenti, la miglior azione da intraprendere;
vengono poi determinati quali active component sono compatibili con tale azione, infine si cerca tra i componenti locali quale sia il migliore per la riconfigurazione. Trovato il componente, viene ritornata l'azione di riconfigurazione; se non viene invece trovato nessun componente per cui è possibile attuare l'azione scelta, la riconfigurazione fallisce.

5.4.3 ReconfigurationManager.

Il ReconfigurationManager è il componente che si occupa di gestire le richieste di riconfigurazione; sia per quanto riguarda le richieste provenienti dalle applicazioni, che le richieste generate

dal sistema. Per quanto riguarda le richieste provenienti da applicazioni utente, il manager registra l'azione richiesta, il componente per cui viene attuata e il listener per l'azione. Tale operazione fallisce però nel caso in cui il componente sia già soggetto alla riconfigurazione di sistema. Se ha successo viene attivato il metodo dell'azione di riconfigurazione che prepara i cambiamenti del sistema. L'azione di riconfigurazione sarà poi gestita dal listener associato.
Per le azioni di riconfigurazione del sistema, il manager le gestisce direttamente in quanto ne è anche il listener e ne mantiene, quindi, lo stato di svolgimento. Inizialmente, si trova in uno stato in cui è libero; cioè uno stato in cui non sta svolgendo nessuna riconfigurazione. Quando, il sottosistema di generazione dei segnali delle risorse, richiede l'avvio della riconfigurazione, il manager registra l'azione e il componente scelto e avvia la fase di preparazione della ReconfigurationAction. Quando tale fase termina, se ha avuto successo, il manager determina la fase seguente che potrà essere di attesa (nel caso non siano stati generati segnali di allarme o di roll back), di roll back, o di attuazione dei cambiamenti preparati. Nel caso, infine, che una delle fasi fallisca, o che, la riconfigurazione termini, il manager cancella i dati registrati, segnala la fine della riconfigurazione e ritorna nello stato free.

5.4.4 I Reconfiguration Listener.

Le fasi attraverso cui transita il processo di riconfigurazione, non hanno in genere una durata prevedibile a priori; il componente (sia esso il sistema o un applicativo utente) che ha attivato tale azione, allora, non si blocca in attesa che le singole fasi finiscano ma registra un listener presso il Reconfiguration Manager. Una volta terminata la fase in atto, la ReconfigurationAction recupera il listener (tramite l'indicatore del componente per cui sta operando) dal sistema e notifica l'esito della fase svolta.
In particolare quindi un Reconfiguration Listener dovrà gestire i casi in cui:
1) la fase di preparazione è terminata con successo

2) la fase di preparazione è fallita.
3) la fase di attuazione della riconfigurazione è terminata con successo.
4) la fase di attuazione della riconfigurazione è fallita.
5) la fase di roll back è terminata con successo
6) la fase di roll back è fallita.

5.5 Conclusione.

In questo capitolo, abbiamo presentato lo schema di funzionamento del progetto che intendiamo realizzare. Tale schema si presta a situazioni diverse in maniera molto flessibile e lascia aperte ampie possibilità per lo sviluppo delle componenti principali cioè le azioni di riconfigurazione.
Esse rappresentano, infatti, il punto cardine per una gestione valida ed efficiente di ogni fase di riconfigurazione del sistema.
La definizione e, la disponibilità, di molti protocolli di riconfigurazione consentirebbe l'applicazione di questa architettura ad un elevato numero di situazioni mentre la disponibilità di pochi protocolli per la riconfigurazione renderebbe, di fatto, questo modello, poco efficiente. Abbiamo poi visto, un protocollo per il QoS-based routing dei flussi di dati e abbiamo visto come l'algoritmo per la scelta dei nodi del percorso possa essere riutilizzato nell'ambito della riconfigurazione dello stesso percorso.
Nel prossimo capitolo presentermo invece i dettagli della realizzazione del modello presentato e il caso particolare di una azione di riconfigurazione.

CAPITOLO 6.

Realizzazione del modello di Riconfigurazione su MUM.

In questo capitolo, intendiamo descrivere come il modello per la riconfigurazione del sistema e l'adattamento dei flussi è stato effettivamente implementato all'interno del middleware MUM.
In particolare vedremo come l'architettura preesistente è stata integrata per consentire la realizzazione dei servizi descritti nel precedente capitolo. Vedremo poi la realizzazione di un esempio pratico di azione di riconfigurazione e infine concluderemo il capitolo con una breve analisi sul lavoro svolto.

6.1 L'implementazione del modello di riconfigurazione.

Come abbiamo visto nel precedente capitolo, i componenti che realizzano l'architettura del modello sono quattro:

- il sottosistema per la generazione dei segnali
- il reconfiguration planner
- il reconfiguration manager
- i reconfiguration listeners

Come vedremo, questi componenti verranno realizzati in maniera integrata nell'architettura di MUM, cercando di lasciare il più possibile inalterata la struttura del middleware e realizzando solo dove necessario, le dovute estensioni. Oltre a queste elementi, nel precedente capitolo si sono anche introdotti i Multimedia Active Component e le Azioni di Riconfigurazione; in questa parte del capitolo, intendiamo presentare l'implementazione di questi oggetti.
Premettiamo inoltre che, si è considerata la riconfigurazione del sistema, come uno dei servizi multimediali che il middleware mette a disposizione e pertanto, all'atto pratico, l'insieme delle classi presentate sarà riunito in un package interno del Multimedia ServicesService di MUM.

6.1.1 Il sottosistema per la generazione dei segnali.

Come abbiamo visto nel capitolo precedente, questo sottosistema prevede il monitoraggio delle risorse del Place e l'identificazione delle condizioni per la generazione dei tre segnali visti :

1) il segnale di attenzione
2) il segnale di allarme
3) il segnale di rientrato allarme

Poiché, come abbiamo visto, nel capitolo relativo, MUM esegue già un monitoraggio delle risorse, l'idea fondamentale è stata quella di espandere il modulo che eseguiva tale funzione in MUM per ottenere tali servizi. L'entità che in MUM implementava il controllo per le risorse è un oggetto che implementa l'interfaccia IResourcesBroker che prendiamo qui in esame. Tralasciando, la parte relativa alla prenotazione anticipata, non coinvolta nel nostro esame, questa interfaccia consente la prenotazione e il rilascio delle risorse; queste funzioni, sufficienti per un funzionamento di base, non consentono di realizzare i nostri intenti; in particolare si è quindi cercato di rendere possibile un maggiore scambio di informazioni. Si è quindi ampliata l'interfaccia per consentire :

- di ottenere lo stato delle risorse prenotate sul Place
- di ottenere lo stato delle risorse, nel caso peggiore risultato dal confronto tra, lo stato effettivo delle risorse e le risorse prenotate. Si deve infatti tenere in considerazione il peggiore dei due quando si valuta la condizione delle risorse in un Place.
- ottenere la soglia per lo stato di attenzione
- ottenere lo soglia per lo stato di allarme
- rendere effettiva una prenotazione

I primi due servizi realizzati consentano di ottenere informazioni che prima erano incapsulate all'interno dell'oggetto e non accessibili all'esterno; ciò si rivela utile anche nel contesto del sistema di località che in questo modo ha un facile accesso alle informazioni di stato. Il terzo e quarto servizio realizzato consentono di conoscere quali sono le soglie per l'attivazione dei segnali di attenzione e allarme; i valori di tali soglie, sono mantenuti nell'environment di

ogni Place in quanto sono parte della descrizione di tale ambiente ma vengono resi accessibili anche qui (il resourceBroker li preleva dall'environment) perché si è voluto creare un oggetto unico che raccogliesse in sé tutte le funzioni necessarie per la gestione dello stato delle risorse e delle informazioni relative.
L'ultimo servizio introdotto è invece completamente nuovo; secondo il modello originale, le risorse su un Place venivano prenotate in un'unica fase dai PlanVisitorAgent che creano i componenti lungo il percorso del flusso; si è deciso di cambiare tale modalità di prenotazione perché, per ogni componente registrato, la chiave per la prenotazione era data dall'AgentID del client; nel caso però che un client sia in grado di gestire più flussi, e due componenti, legati a flussi diversi ma dello stesso client, si fossero trovati sullo stesso Place, non si sarebbe potuto distinguere tra i due. Si è deciso pertanto di adottare come chiave per i componenti proxy l'insieme dell'AgentID del ClientAgent e del ProxyAgent. Il problema che è nato da questo è che i Visitor Agent creano i Proxy Agent esclusivamente quando sanno che nel Place vi sono risorse sufficienti, e quindi, a prenotazione eseguita. E' stato, pertanto, necessario realizzare la prenotazione in due fasi : la prima in cui vengono riservate le risorse, con chiave l'agentID del ClientAgent, la seconda in cui la prenotazione provvisoria diventa effettiva con la chiave definitiva.
Oltre all'interfaccia, come abbiamo detto, l'oggetto che effettivamente implementa il resourceBorker realizza anche un controllo delle risorse (della CPU in MUM) e implementa l'interfaccia ICPUStatusNotifier, è cioè l'entità che notifica ai QoS Manager delle applicazioni, eventuali problemi dovuti a un sovraccarico di CPU. Tale interfaccia è rimasta identica per compatibilità con la versione precedente ma si è implementata una seconda interfaccia IResourcesStatusNotifier; questa interfaccia è quella che consente il delivery dei segnali necessari alla riconfigurazione. In particolare i metodi realizzati consentono :

- di aggiungere / rimuovere un listener all'entità notifier
- di notificare uno stato di attenzione / allarme

- di notificare la fine dello stato di attenzione (rientrato allarme)
- di segnalare al notifier la fine della riconfigurazione

I primi tre servizi indicati non necessitano ulteriori commenti; per il quarto, è necessario invece presentare qual'è la logica che governa la generazione dei segnali. Il notifier, controlla ciclicamente lo stato delle risorse; quando viene segnalato uno stato di attenzione, ciò viene memorizzato anche al suo interno. Per non generare segnali di attenzioni multipli, prima di poterne generare nuovamente uno, tale stato deve essere disattivato. La disattivazione dello stato di attenzione avviene dopo un numero di cicli prefissato in cui non lo stato delle risorse è contenuto nelle soglie. Nel caso invece in cui venga generato lo stato di allarme, il controllo delle risorse viene momentaneamente disattivato, sempre per evitare la generazione di segnali multipli. Si rende però così necessaria la sua riattivazione, tramite appunto l'ultimo servizio visto, quando la riconfigurazione è conclusa.

6.1.2 Il Multimedia Services Manager.

Passiamo ora alla dettaglio delle estensioni portate al Multimedia Services Manager. Come abbiamo anticipato nell'introduzione, si è inteso realizzare il processo di riconfigurazione come un servizio multimediale che viene offerto alle applicazioni utenti; questo perché, la riconfigurazione, come abbiamo visto, non è necessariamente richiesta solo dal sistema, ma può anche essere attivata dalle applicazioni a livello utente, ed è quindi possibile pensare ad essa come a un servizio. Poiché il manager che esaminiamo ora è il front-end del middleware per tali servizi, lo si è incluso nell'implementazione del nostro modello. Esso è quindi il riferimento sia per le richieste esplicite degli utenti, sia per quanto riguarda le richieste provenienti dai QoS manager delle applicazioni, sia per la gestione della riconfigurazione dei flussi passanti per il Place (operata dal middleware). Questa scelta progettuale consente quindi la realizzazione di un meccanismo omogeneo per i tre possibili casi di riconfigurazione.

Nella sua versione originale, il manager era utilizzato per registrare i manager di sessione dei componenti che venivano creati dai Visitor Agent (client, proxy) e per l'attivazione dei servizi (server) su un Place. Nella nostra ottica, il manager dei servizi multimediali, diviene quindi l'entità che mantiene, registra e gestisce gli Active Component presentati nel precedente capitolo. Per ottenere ciò, in particolare, sono stati sviluppati i seguenti servizi :

1) Attivare i componenti Server richiesti sul Place.
2) Registrare la presenza di qualunque componente attivo presente sul Place.
3) Fornire un riferimento ad un particolare componente attivo richiesto o una lista dei componenti attivi presenti in locale.
4) avviare la fase di riconfigurazione secondo per uno specifico componente e con una determinata Reconfiguration Action o Remote Reconfiguration Action. In questo caso, l'azione di riconfigurazione che si intende intraprendere è già definita alla richiesta. E' il caso di riconfigurazioni avviate da applicazioni o da utenti. Nel primo caso è l'applicazione che determina cosa è meglio fare, nel secondo è l'utente che indica cosa vuole (ed eventualmente l'applicazione determina quale azione intraprendere). In entrambi i casi, la prima operazione che si effettua è registrare azione, componente e listener presso il manager di riconfigurazione; tale registrazione ha successo solo se il componente non è soggetto ad altre azioni di riconfigurazione. Se si tratta di una azione di riconfigurazione remota, viene anche indicato un Place diverso da quello in cui ci si trova; l'azione viene quindi incapsulata in un Reconfiguration Command e inviata sul Place. Se su tale Place, l'azione può essere attuata, viene attivata la fase di riconfigurazione tramite il manager dei servizi multimediali, altrimenti viene identificato un nuovo Place e reiterato il processo. Se non è possibile invece identificare un nuovo Place, l'azione richiesta fallirà. Nel caso, invece, che l'azione di riconfigurazione sia da attuare in locale, dopo averla registrata viene attivata direttamente tramite il manager di riconfigurazione.

5) avviare la fase di riconfigurazione su richiesta di un componente attivo non locale. E' questo il caso indicato nel punto precedente, quando si è introdotto il Reconfiguration Command. Un'azione di riconfigurazione remota, giunge sul Place tramite questo comando e, qui può richiedere al manager di avviare la riconfigurazione per uno dei componenti locali (il componente locale deve essere legato a quello che ha generato l'azione di riconfigurazione, devono cioè operare sullo stesso flusso; se invece non vi sono componenti legati al flusso sul Place, ne deve essere creato uno prima di richiedere la riconfigurazione al manager).
6) indicare su un componente specificato è già in fase di riconfigurazione o meno.

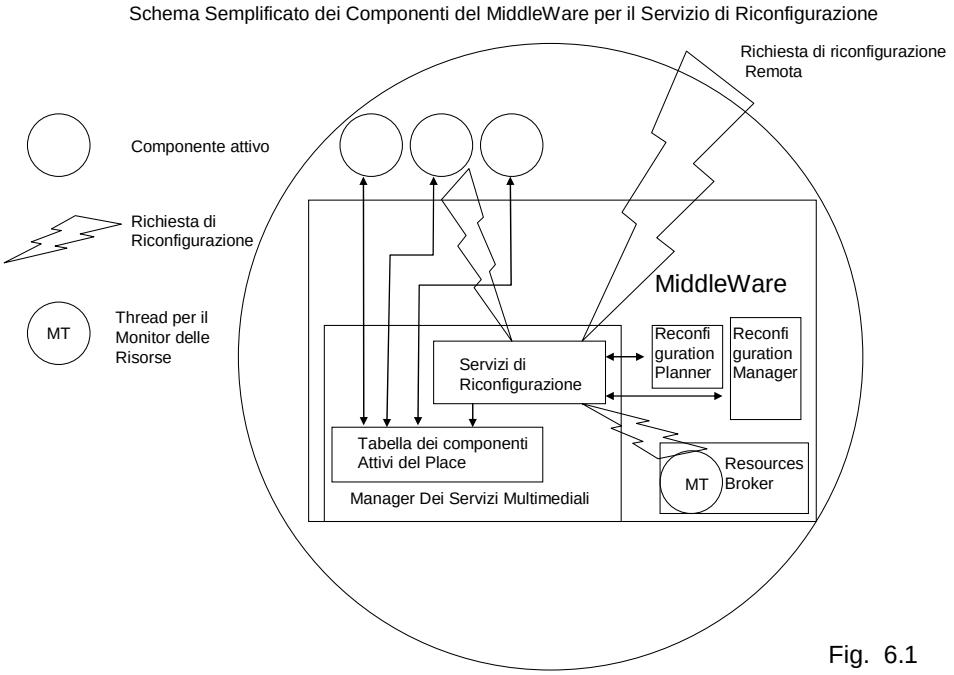


Fig. 6.1

Oltre a questi, il manager deve fornire anche i servizi per la gestione della riconfigurazione dei flussi del Place; sono quindi realizzati servizi per :

7) attivare la fase di preparazione.
   In questo caso, attivare la fase di preparazione, necessita, per prima cosa, l'identificazione del tipo di azione di riconfigurazione da attuare, e successivamente, nell'identificazione di un componente attivo tra quelli presenti che possa essere riconfigurato secondo l'azione scelta.
   Oltre a ciò, attivando tale fase, deve specificare se, terminata la preparazione, deve seguire la fase di attuazione o se deve essere eseguita esclusivamente questa fase.
8) attivare la fase di realizzazione dei cambiamenti.
9) attivare la fase di roll back. Ossia cancellare i cambiamenti effettuati durante la fase di preparazione se lo stato di attenzione rientra.

Tali servizi vengono attivati dal ResourcesBroker in quanto il manager implementa l'interfaccia di IResourcesStatusListener e si registra presso il broker all'atto della sua creazione.
Il manager, ovviamente, non realizza direttamente tutti i servizi indicati ma esclusivamente quelli che possono essere di carattere globale mentre gli altri vengono realizzati da entità con cui interagisce che sono di fatto quelle già introdotte a inizio capitolo :
1) Il ReconfigurationPlanner, e il
2) Il ReconfigurationManager.

Facciamo infine presente come in questo paragrafo sia stato introdotto un oggetto nuovo : il Reconfiguration Command; esso non è stato presentato nel capitolo in cui abbiamo discusso il modello di riconfigurazione perché è strettamente connesso all'implementazione e alla piattaforma SOMA e ci è sembrato quindi più opportuno presentarlo in questa sezione.

6.1.3 Il Reconfiguration Planner.

Il funzionamento del Reconfiguration Planner è stato introdotto nel capitolo precedente; in questo paragrafo intendiamo soffermarci su alcuni dettagli dell'implementazione. Come abbiamo detto, l'unico servizio che il Planner mette a disposizione è quello per elaborare, dato lo stato delle risorse, un'azione di riconfigurazione per il

sistema. Attualmente non sono implementati particolari protocolli per l'identificazione della miglior azione possibile in funzione dello stato delle risorse segnalato seppure questo sia sicuramente un ambito di ricerca interessante esula dagli obiettivi del progetto.
Al momento, quindi, l'azione viene determinata staticamente. Il Planner determina quale dei componenti registrati presso il manager dei servizi multimediali può essere soggetto all'azione scelta e per determinare ciò tiene in considerazione sia quella che è la sua azione di Default Action sia delle sue Forbidden Actions.
Il Planner ritorna quindi una ReconfigurationAction o una Remote Reconfiguration Action che mantiene già tutte le informazioni necessarie per la riconfigurazione.

6.1.4 Il Reconfiguration Manager.

Il Reconfiguration Manager è il componente che si occupa dell'attivazione delle azioni di riconfigurazione sia per il sistema che per le applicazioni utenti. Le richieste generate da applicazioni utenti, vengono mediate dal manager dei servizi multimediali, che come abbiamo detto è il front-end per questo servizio in MUM; in particolare, esse attivano il metodo :
external_request_reconfiguration
sia nel caso che l'azione sia stata generata localmente sia nel caso che sia giunto un ReconfigurationCommand e che abbia richiesto l'attivazione dell'azione che trasportava. Il metodo indicato si limita comunque a fornire all'azione un riferimento all'ambiente e ad attivare il suo metodo di preparazione dei cambiamenti.
Nel caso invece che debba essere attivata un'azione di riconfigurazione per il Place, il manager dei servizi multimediali chiama il metodo prepare_reconfiguration. In questo caso, il manager di riconfigurazione realizza anche il listener per l'azione stabilita dal Reconfiguration Planner; viene inoltre indicato al manager quale notifier ha attivato la riconfigurazione e se, alla preparazione dell'azione, deve seguire la sua attuazione. Il manager chiama poi, come nel caso precedente, il metodo dell'azione di riconfigurazio-

ne che prepara i cambiamenti e poi ritorna al normale funzionamento.
Durante lo svolgimento della fase preparatoria, potranno arrivare altri segnali che modificheranno lo stato del manager; quando la fase preparatoria terminerà, a seconda dello stato in cui si troverà, il manager deciderà se effettuare la realizzazione dei cambiamenti, effettuare un roll back o non operare alcuna operazione. La possibilità di non operare alcuna operazione è tuttavia possibile solo per un periodo di tempo limitato dopodichè il sistema necessariamente indicherà al manager quale comportamento intraprendere. Al termine del fase di realizzazione o di roll back, il manager segnalerà la fine della riconfigurazione, eliminerà i riferimenti all'azione, e notifier e infine resetterà il proprio stato.

6.1.5 I Reconfiguration Listeners.

Come abbiamo visto nel paragrafo 5.4.4 i Reconfiguration Listener sono quegli oggetti a cui vengono notificati gli esisti delle fasi di riconfigurazione; in realtà poiché il comportamento di tali oggetti dipende dall'azione in particolare e dalle politiche adottate, non sono stati realizzati degli oggetti specifici che realizzassero i listeners ma è semplicemente stata creata un'interfaccia costituita da due metodi, qualunque oggetto implementi tale interfaccia può quindi essere utilizzato come listener per gli esiti delle fasi di riconfigurazione. I metodi dell'interfaccia sono:

- reconfigurationEnded(ReconfigurationAction, boolean)
- reconfigurationPrepared(ReconfigurationAction, boolean)

In questi metodi, il campo ReconfigurationAction indica l'azione che si è conclusa e il valore boolean indica se la fase è terminata con successo o meno. Notiamo che non è presente un metodo per notificare la conclusione della fase di roll back in quanto, per un listener questa coincide con il caso in cui fallisca la fase in cui si attuano i cambiamenti preparati (ricordiamo ancora che eventuali errori durante lo svolgimento di una qualunque fase non vengono gestiti dal listener ma devono essere gestiti all'interno dell'azione di riconfigurazione).

Un esempio di listener è il manager di riconfigurazione che abbiamo visto nel paragrafo precedente; come listener, il manager, adotta una politica di azione che dipende dai segnali che gli vengono notificati. Un altro esempio di listener verrà presentato in 6.2 quando verrà discusso un caso effettivo di riconfigurazione del percorso di uno stream multimediale.

6.1.6 Gli Active Component.

In questo paragrafo ci occupiamo della descrizione degli Active Component; come abbiamo già detto, gli Active Component sono oggetti che derivano tutti dal MultimediaActiveComponent; nella nostra implementazione quindi, abbiamo che quest'ultimo oggetto rappresenta la classe madre di ogni altro Active Component. All'interno di tale classe sono definite le costanti per l'identificazione dei componenti specifici (CLIENT, PROXY, ecc..) ed è presente un oggetto ActionBox. Quando viene creato un nuovo MultimediaActiveComponent (o un qualunque altro componente, infatti, in ogni costruttore la prima chiamata è al costruttore della classe padre) viene creata l'ActionBox relativa al tipo di componente, specificato mediante una delle costanti. La determinazione delle azioni che caratterizzino il comportamento di un componente in fase di riconfigurazione, è sicuramente un tema interessante per ulteriori sviluppi, ma poiché esula dall'ambito di questo progetto, si è scelto di assegnare in maniera statica le azioni ai componenti e in particolare, l'azione di default in caso di riconfigurazione per client e server è nessuna azione, mentre per il Proxy è spostare il componente e per lo StandAloneComponent è chiudere la sessione del componente. Presentiamo ora, prima di procedere con la descrizione dei singoli Active Component, un nuovo oggetto:
il ComponentIdentifier.

6.1.6.1 I Component Identifier.

Scopo dei ComponentIdentifier è quello di mantenere le informazioni relative ad un ActiveComponent; i ComponentIdentifier sono quindi strutturati come gli ActiveComponent. Esiste un oggetto

generale chiamato appunto ComponentIdentifier che mantiene le caratteristiche che sono comuni per tutti i componenti del sistema e, da questo oggetto derivano, come nel caso del Multimedia ActiveComponent, oggetti più specifici, relativi alle tipologie dei componenti. Abbiamo quindi (vedi fig. 6.2):
ClientIdentifier,
ProxyIdentifier,
ServerIdentifier e
StandAloneComponentIdentifier.

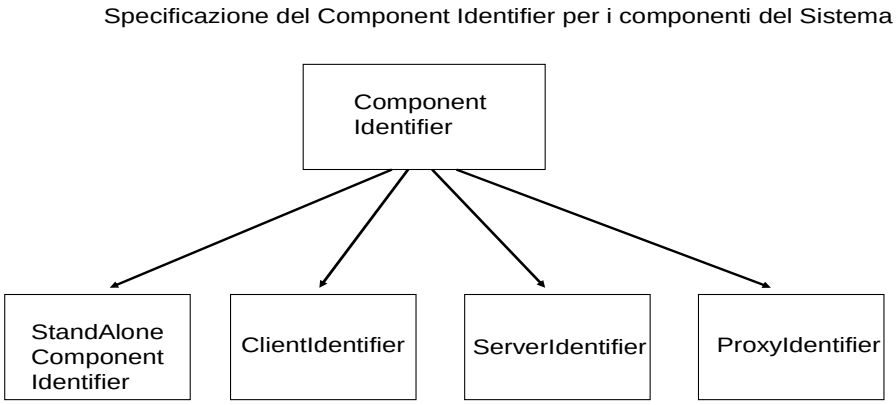


Fig. 6.2

La differenza con gli Active Component è che ricoprono ruoli diversi : mentre gli Active Component rappresentano per il sistema un'astrazione dell'oggetto a cui si riferiscono, i Component Identifier sono esclusivamente utilizzati come identificatori del Componente. A livello pratico tale distinzione, si manifesta nel fatto che nei ClientIdentifier non è presente il riferimento al manager di protocollo dell'oggetto e pertanto, come oggetti sono molto più maneggevoli dei rispettivi Active Component i quali per esempio non potrebbero essere impacchettati in un comando e inviati da

qualche parte poiché mantengono riferimenti a strumenti di sistema per l'interazione con fileSystem, rete, ecc..
La necessità di mantenere le informazioni sul componente che ha generato la richiesta di riconfigurazione (o che viene riconfigurato) anche al di fuori del Place in cui si trova, ha determinato l'esigenza dei ComponentIdentifier.

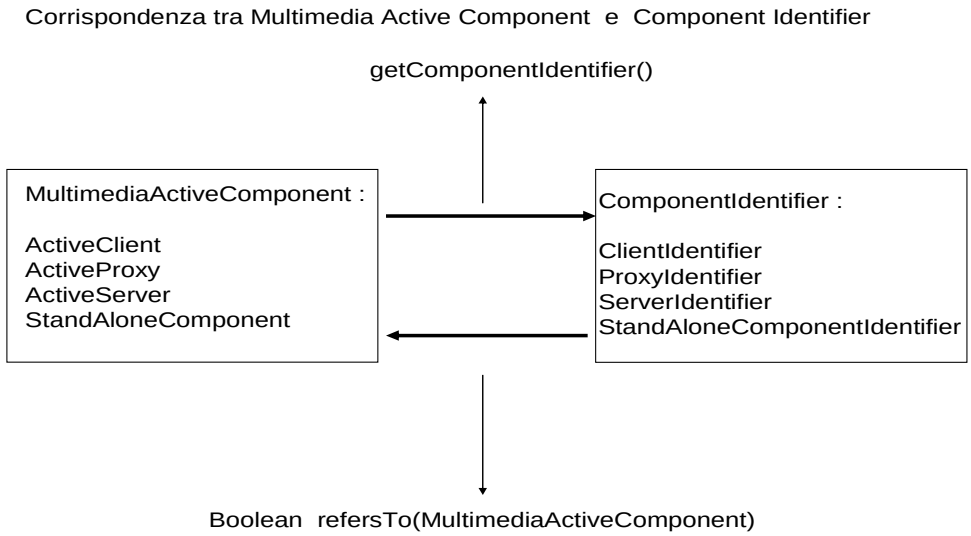


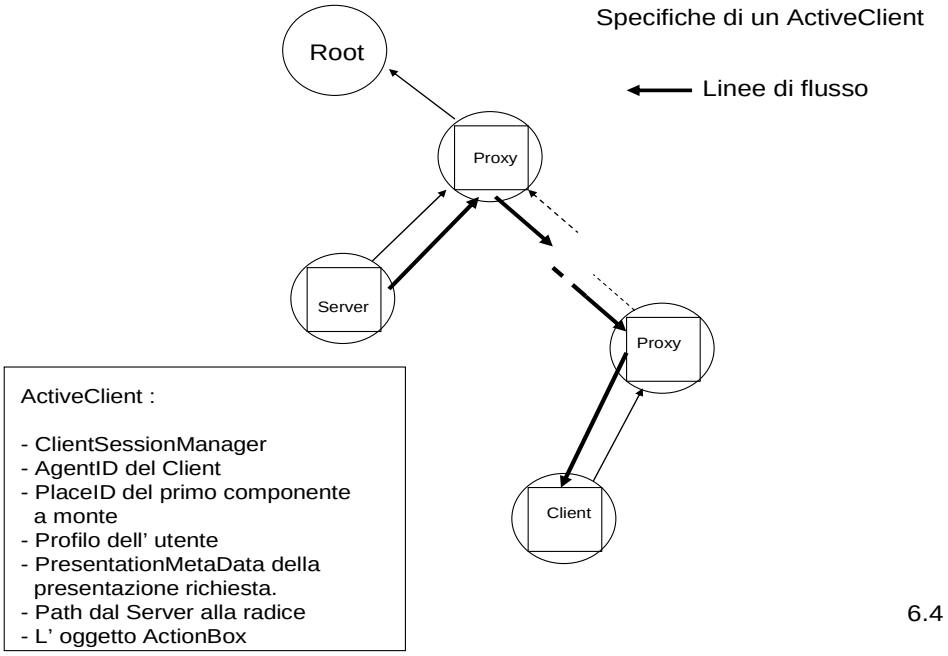
Fig. 6.3

La corrispondenza tra Active Component e ComponentIdentifier relativo è mostrata in figura 6.3; gli ActiveComponent avranno un metodo: getComponentIdentifier(), con cui potranno generare il ComponentIdentifier relativo e ogni ComponentIdentifier avrà un metodo: refersTo con cui potrà riconoscere l'ActiveComponent a cui si riferisce. Passiamo ora alla specifica dei singoli componenti.

6.1.6.2 Active Client, Proxy, Server e StandAloneComponent.

Nel paragrafo 5.2.2 abbiamo visto quali sono i servizi minimi, indipendenti dalla struttura del sottosistema, che un oggetto Active Client deve implementare; in questo paragrafo, e nei successivi,

presentiamo invece la struttura con sui essi sono stati effettivamente realizzati; ogni oggetto ActiveClient contiene :



6.4

1) un oggetto ActionBox che come abbiamo visto ha come default action una no_action e nessuna forbidden_action impostata.
2) l'identificatore dell'agente che è stato attivato dall'utente, cioè l'AgentID del ClientAgent.
3) l'identificatore del Place (PlaceID) in cui si trova il primo componente che svolge un servizio (sia esso un Proxy o un Server) a monte del Client.
4) Il profilo dell'utente che ha attivato il ClientAgent.
5) Il metadato della presentazione che l'utente ha richiesto.
6) Il path dal nodo radice al nodo in cui si trova il componente server.
7) Il manager di protocollo del client.

In questo modo, agendo sull'ActiveClient, il sistema è in grado di agire sul flusso. La presenza del manager di protocollo, che

ricordiamo deve presentare delle interfacce prestabilite,consen-te al sistema di intervenire sul flusso tramite i singoli compo-nenti. (ciò avverrà appunto tramite i metodi di interfaccia del manager di protocollo, come vedremo nella prossima parte del capitolo).

All'atto della creazione di un oggetto ActiveClient, non tutti i parametri indicati possono essere noti; La determinazione del Place in cui risiede il Server, per esempio, è possibile solo una volta che il percorso tra Client e Server è stato stabilito (nel caso che si esegua un IMultiplePlan). Lo stesso dicasi del metadato che mantiene le informazioni sulla presentazione scelta nel caso che ve ne siano più di una disponibili. Oltre ai metodi per ottenere i singoli oggetti, l'ActiveClient implementerà quindi anche dei metodi per settare, una volta ottenuti, tali parametri .

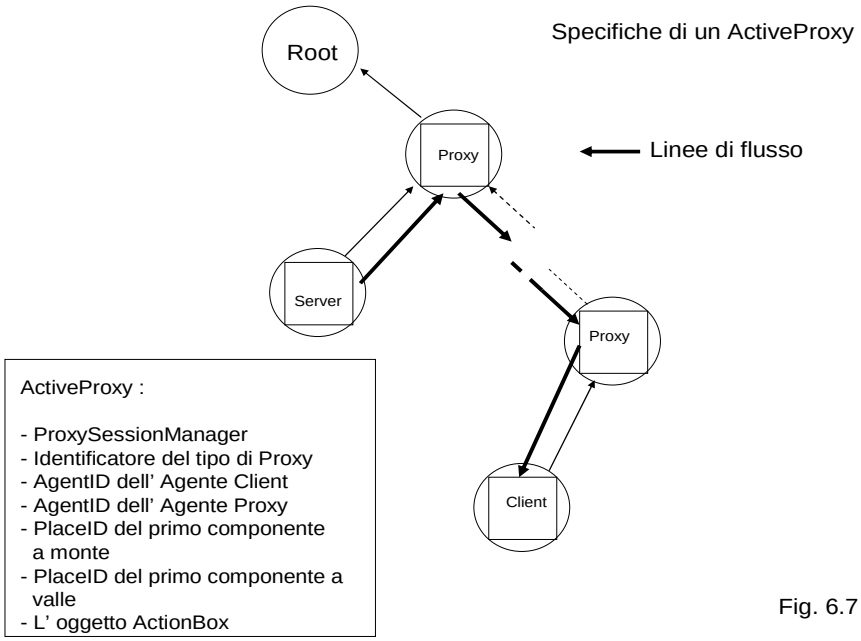


Fig. 6.7

Passiamo ora ai rimanenti componenti di cui abbiamo visto la struttura nei paragrafi 5.2.3 – 5.2.5.

Per quanto riguarda il componente Proxy, esso contiene i seguenti oggetti :

1) un oggetto ActionBox
2) l'identificatore dell'agente (AgentID) che svolge il servizio di proxy
3) l'identificatore dell'agente (AgentID) per il quale il flusso è stato creato (in genere l'AgentID del ClientAgent)
4) l'identificatore del primo Place (PlaceID) a monte in cui si trova un componente attivo, lungo il percorso dello stream multimediale; su tale place si potrà trovare o un componente Proxy oppure un componente Server .
5) l'identificatore del primo Place (PlaceID) a valle in cui si trova un componente attivo, lungo il percorso dello stream multimediale; su tale place si troverà invece o un componente Proxy o il componente Client.
6) Il manager di protocollo del componente proxy. In particolare tale manager di protocollo è quello che si occupa della gestione del flusso in ingresso, della gestione del flusso in uscita e della coordinazione dei due.

Prendiamo infine in considerazione l'oggetto ActiveServer. Nel middleware, ogni servizio viene univocamente identificato mediante il classpath delle classi che lo realizzano e l'interfaccia che

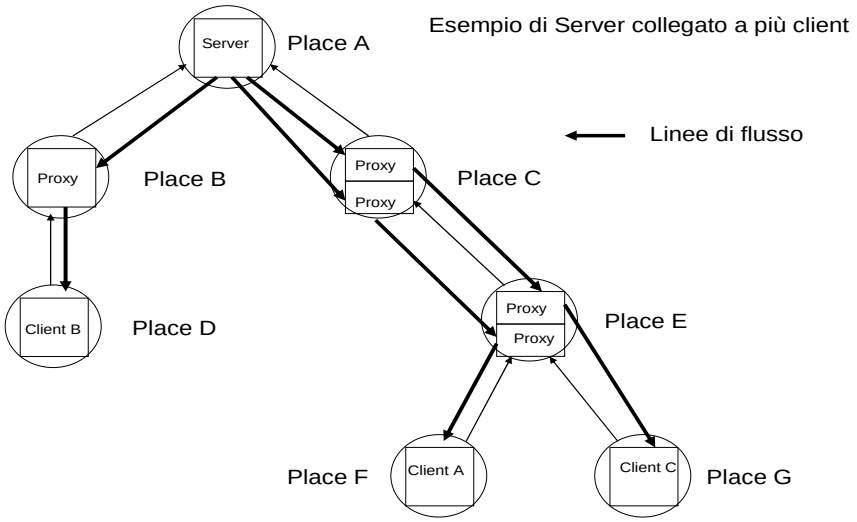


Fig. 6.8

il servizio implementa. Oltre a questo, come si vede in figura 6.8 e 6.9 e come abbiamo già detto, un server può essere connesso a più utenti e deve comunque consentire di operare con i singoli flussi.

Specifiche mantenute in un Active Server

Active Server contiene :

1) ClassPath + Interface del servizio.
2) Parametri di Inizializzazione del Servizio.
3) Oggetto ActionBox.
4) Riferimento al Thread Demone che svolge il servizio vero e proprio.
5) Un riferimento all'ambiente del Place in cui si trova.
6) Un vettore che contiene :

Per il Client A :	Per il Client B :	Per il Client C :
1) Place F	1) Place D	1) Place G
2) Place C	2) Place B	2) Place C
3) ClientAgentID del Client A	3) ClientAgentID del Client B	3) ClientAgentID del Client C
4) Manager di protocollo per il Client A	4) Manager di protocollo per il Client B	4) Manager di protocollo per il Client C

Fig. 6.9

Per permettere ciò, l'ActiveServer non mantiene informazioni solo sul servizio in sé ma deve anche mantenere un riferimento a tutti i thread che vengono creati per soddisfare le richieste dei client.
Abbiamo quindi che un ActiveServer contiene :

1) classpath e interface che definiscono univocamente il servizio svolto.
2) i parametri che sono necessari all'inizializzazione del Server quando viene attivato.
3) un riferimento al thread demone che effettivamente svolge il servizio.
4) un riferimento all'ambiente in cui si trova.
5) un oggetto ActionBox.
6) un vettore contenente le informazioni relative ai singoli client che sta servendo; in particolare, ogni elemento di tale vettore è un array di oggetti che descrivono un client. Le informazioni mantenute sono :
   1) l'identificatore del Place in cui si trova il client
   2) l'identificatore del primo Place a valle del Server in cui si

trova un componente attivo, lungo il percorso dello stream multimediale.
3) l'identificatore dell'agente client avviato dall'utente.
4) un riferimento al manager di protocollo creato per gestire la sessione con il client specificato.

Passiamo infine agli StandAloneComponent; poiché per questi componenti non c'è flusso sulla rete, non è necessario mantenere particolari informazioni; teoricamente, poiché il flusso viene generato e consumato localmente e non serve nemmeno un manager di protocollo ma al limite, un manager di sessione. In caso di riconfigurazione, per default si è scelto di chiudere forzatamente la sessione del componente. Uno StandAloneComponent mantiene, comunque, i seguenti oggetti :
1) l'identificatore dell'agente che realizza il componente
2) un riferimento allo StandAloneSessionManager, cioè il manager del componente.

6.1.7 Le Azioni di Riconfigurazione.

Delineiamo ora la struttura delle azioni di riconfigurazione; come abbiamo spiegato nel capitolo precedente esse rappresentano un involucro per i protocolli di riconfigurazione. Quando i protocolli devono essere eseguiti in locale avremo una Reconfiguration Action mentre quando questi devono essere eseguiti in un altro Place avremo una Remote Reconfiguration Action. In realtà però, la distinzione non è così netta, infatti, si è voluto, in fase realizzativa, rendere il più facile possibile l'eventuale trasformazione di una Reconfiguration Action in una Remote Reconfiguration Action. All'atto pratico, ciò è stato ottenuto creando un solo oggetto e cioè le Reconfiguration Action e realizzando per le azioni remote un'interfaccia. In questo modo è possibile pensare di implementare le azioni di riconfigurazione dapprima solo in locale, poi qualora ve ne sia la necessità, senza modificare il codice già scritto, limitarsi a

implementare i metodi dell'interfaccia e ottenere un'azione di riconfigurazione remota.
Nel dettaglio, la classe che implementa le reconfiguration action definisce le costanti per la definizione delle sue sottoclassi, allo stesso modo dei Multimedia Active Component e dei Component Identifier. Essa è per il resto un contenitore vuoto, che definisce i metodi fondamentali, che già abbiamo presentato in 5.3.1, e che le sottoclassi dovranno implementare. Per quanto riguarda i metodi che dovranno essere implementati per l'interfaccia delle azioni remote essi dovranno

1) indicare un Place in cui si ipotizza sia possibile attuare la riconfigurazione, nell'esempio che vedremo, si risale il flusso fino a che non si trova un Place giusto o non si arriva al Server.
2) indicare se un Place è adatto alla riconfigurazione richiesta.
3) indicare se sul Place in cui si vuole svolgere l'azione di riconfigurazione, è necessario installare un nuovo componente o è possibile agire su un componente già presente.
4) Creare, in caso di necessità, il componente necessario per attuare la riconfigurazione (vedi punto 3), nel Place scelto.
5) indicare il suo Place di origine.

6.1.8 I Reconfiguration Command.

Introduciamo in questo paragrafo degli oggetti nuovi; questi non sono stati presentati nella discussione del modello di riconfigurazione del sistema perché in tale modello non si è supposto di adottare un meccanismo piuttosto che un altro per il trasferimento delle Remote Reconfiguration Action. Poiché nell'implementazione ci si è basati però su una piattaforma reale, il problema di quale vettore utilizzare è stato posto. La scelta di utilizzare i Command di SOMA è stata quasi immediata in quanto questo evitava di dover creare, su ogni Place, un server multithread che gestisse le richieste di riconfigurazione remota, mantenendo quindi, l'architettura il più semplice possibile. I comandi implementati effettivamente sono tre:

- ReconfigurationCommand
- ReconfigurationPreparedCommand
- ReconfigurationEndedCommand

Di questi però solo il primo è fondamentale in quanto incapsula una parte della logica per la riconfigurazione del sistema. I due rimanenti si limitano a tornare sul Place di origine, dove recuperano il listener e gli notificano la riuscita o meno dell'operazione. Facciamo presente che il comando ReconfigurationEndedCommand, viene utilizzato anche nel caso del roll back e che, oltre a notificare il successo o meno della fase appena conclusa libera il ReconfigurationManager dalla registrazione fatta all'inizio della fase di riconfigurazione.
Il ReconfigurationCommand viene creato dal manager dei servizi multimediali e poi inviato su un Place indicato dal client; quando arriva sul Place, la prima cosa che fa è controllare tramite i servizi dell'interfaccia delle azioni remote se sul Place si può attuare l'azione. Se non è così, cerca tra i componenti registrati sul Place uno che sia legato al componente da cui è stato avviato (cioè, in sostanza, cerca un componente dello stesso flusso); se lo trova ed è un Proxy, gli richiede su quale Place si trova il prossimo componente e su tale Place si sposta. Se il componente non c'è oppure è un Server, significa che o il comando si è perso in qualche modo, oppure si è arrivati al termine del percorso senza trovare un Place dove attuare l'azione. In entrambi i casi, viene creato un comando ReconfigurationEndedCommand in cui viene indicato l'esito negativo della riconfigurazione e questo viene spedito al place da cui si è partiti, dopodichè si termina. Se invece si può attuare l'azione su questo Place, si verifica, sempre tramite i metodi dell'interfaccia delle azioni remote se, per questa azione e su questo place, devo creare un nuovo componente o ne devo usare uno già presente. In questo caso, recupero l'identificatore del componente che deve riconfigurare e richiedo l'avvio della fase di riconfigurazione al manager dei servizi multimediali, indicando l'azione, il componente e come listener, il comando stesso.
Naturalmente questa è una scelta implementativa, a livello teorico,

si potrebbe certamente pensare di passare al comando, all'atto della sua creazione, oltre all'azione di riconfigurazione anche un oggetto listener che implementi la politica voluta. Poiché però al momento non vi è necessità di particolari politiche nei listener si è ritenuto più che adatta questa scelta.
Nel caso che sia invece necessaria la presenza di un componente, il componente viene creato grazie al metodo dell'interfaccia per le azioni di riconfigurazione remote e poi viene registrato nel sistema. Infine viene richiesta la riconfigurazione. E' da notare che il componente che viene creato è in genere un componente temporaneo la cui esistenza sarà limitata alla fase di riconfigurazione; la sua utilità consiste nell'essere un referente per le fasi successive della riconfigurazione. Un esempio di ciò sarà dato proprio nella prossima parte del capitolo in cui si discute di un caso pratico di riconfigurazione di uno stream multimediale.

6.2 Un esempio pratico : la New Path Action.

In questo paragrafo intendiamo presentare un caso particolare effettivamente realizzato di azione di riconfigurazione. Il nome dell'azione di riconfigurazione è New Path Action; in sostanza, tale azione di riconfigurazione consente di gestire la mobilità nel caso di un utente nomadico che decida di cambiare, durante la presentazione di un oggetto multimediale, il terminale da cui accedere all'oggetto. Per lo sviluppo di tale azione di riconfigurazione, è stato utilizzato un prototipo di applicazione utente sviluppata in [GFRMI]. Nella prima parte del paragrafo, riprendiamo brevemente la descrizione di tale applicazione, mentre nella seconda intendiamo descrivere il meccanismo realizzato dall'azione di riconfigurazione. Premettiamo che, essendo l'azione di spostamento della sessione da un terminale ad un altro direttamente comandata dall'utente, per essa, non vi è necessità di effettuare la riconfigurazione in due fasi, ma ne basterà una.

6.2.1 Presentazione dell'applicazione utente.

Le entità che sono state sviluppate per l'applicazione sono tre:

1) il Client;
2)il Proxy;
3) il Server.

Nel sistema è stato progettato anche il decision maker in modo che realizzi piano per queste tre entità e un QoS manager di cui però, in questo caso non ci interessiamo particolarmente in quanto non è coinvolto nei meccanismi di riconfigurazione.

Poiché l'applicazione fa largo uso del Java Media Framework (JMF), vale inoltre la pena di soffermarsi anche su questo componente così come sul protocollo RTP, usato nel nostro caso, per la gestione di dati multimediali in Java.

6.2.1.1 Java Media Framework.

Il Java Media Framework sono API (*application programming interface*) create per permettere di incorporare tipi di dati Multimediali in applicazioni o applet Java; si tratta di un pacchetto opzionale, installabile per estendere le funzionalità della piattaforma JAVA2SE™, sviluppato congiuntamente da Sun e IBM.

Poiché Java utilizza una Java Virtual Machine (JVM) che interpreta il byte-code generato dal compilatore Java, nel caso in cui sia richiesta un'elevata velocità di esecuzione, ciò impone dei seri vincoli di prestazioni. Il trattamento di dati Multimediali, in particolare, è uno dei casi in cui è richiesta un'elevata velocità computazionale (decompressione delle immagini, rendering, ecc.), e quindi, non è sufficiente la sola emulazione della CPU per ottenere delle prestazioni ottimali.

Per ottenere delle prestazioni migliori si deve, necessariamente, ricorrere a codice nativo della piattaforma alla quale è interessato. Questo procedimento comporta però due problemi:

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

L’API JMF tenta di risolvere questi problemi, mettendo a disposizione del programmatore una serie di chiamate ad "alto livello" per la gestione del codice nativo e fornendo un supporto per i più comuni standard di memorizzazione dei dati multimediali, quali: MPEG-1, MPEG-2, Quick Time, AVI, WAV, AU e MIDI.
Usando JMF, l'applicazione o applet non ha necessità di conoscere quando e se deve sfruttare particolari metodi nativi per svolgere una determinata azione. JMF è stato progettato, in particolare, per:

- Facilitare la programmazione
- Mettere a disposizione del programmatore un player JMF per la riproduzione dei dati multimediali.
- Semplificare notevolmente l’integrazione di sorgenti multimediali in applet o applicazioni fornendo tutta una serie di classi e metodi per la gestione temporale di stream di dati.
- Connettersi a host remoti e instaurare sessioni http, piuttosto che RTP/RTCP o RTSP (Real Time Streaming Protocol).
- Permettere lo sviluppo di applicazioni di audio e video conferenza in Java.
- Permettere a programmatori avanzati di implementare soluzioni personalizzate basate sulle API esistenti e di integrare le nuove caratteristiche nella struttura esistente.

Tornando invece all’applicazione utilizzata, la ricezione dei dati sul client avviene tramite un Player che è la struttura che le API di JMF mettono a disposizione del programmatore per la riproduzione di dati multimediali. In figura 6.10 viene rappresentato un tipico scenario di utilizzo del JMF, per la ricezione di dati dalla rete. Questa figura è stata tratta da una documentazione che purtroppo è piuttosto datata (vedi [JMFPG]): il SessionManager nell’attuale versione 2.1.1 è stato sostituito dall’RTPManager, comunque concettualmente non è cambiato molto. Per ogni flusso multimediale ricevuto viene impiegato un RTPManager che gestisce la sessione RTP. Ad alto livello, dal lato del ricevente possiamo, attraverso l’uso di opportune API ricavare, all’arrivo dei

dati, un DataSource che verrà utilizzato per l'inizializzazione del Player.

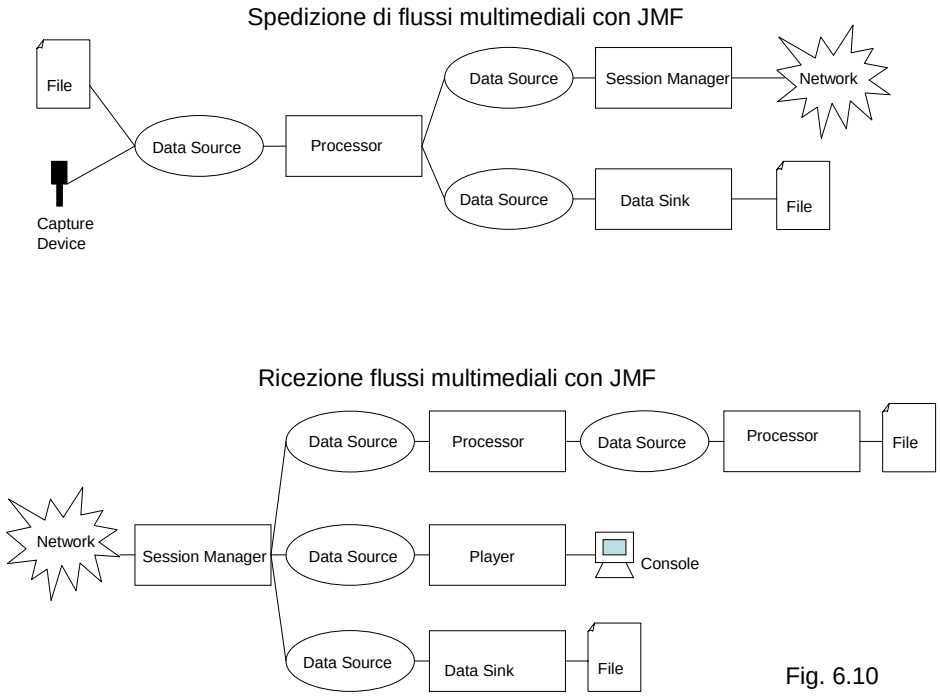


Fig. 6.10

Una volta inizializzato il Player potremo poi ricavare dal Player stesso un componente visuale (se presente), che utilizzeremo per la fruizione del materiale multimediale; nel nostro caso, dal momento che realizzeremo uno streaming video, tale componente sarà, ovviamente, presente. Per quanto riguarda invece l'invio dei dati viene utilizzato un Processor (vedi sempre fig. 6.10). Il Processor estende il Player ed in più di quest'ultimo offre metodi per la gestione e la trasformazione dei flussi multimediali e per l'ottenimento di un DataSource. Tale metodo è di fondamentale importanza per la successiva inizializzazione dell'RTPManager; infatti, all'atto della creazione di un SendStream, che rappresenta appunto l'astrazione del flusso dati che verrà trasmesso in rete, bisognerà passare all'RTPManager un DataSource dal quale attingere il contenuto multimediale che sarà, per l'appunto, ottenuto dal

Processor. Il JMF mette, infine, a disposizione un set di API per stabilire sessioni RTP, offrendo al livello applicativo tutte le astrazioni necessarie a gestire tale protocollo in modo piuttosto semplice. Per avere ulteriori informazioni è possibile consultare la suddetta documentazione, oppure fare riferimento alla home page [JMFH].

6.2.1.2 Il Protocollo RTP.

Nelle applicazioni che devono garantire un servizio in tempo reale, è più facile compensare la perdita di una parte dei dati, rispetto ad un ritardo troppo elevato. Di conseguenza i protocolli che ben si adattano alla trasmissione di dati statici, non si adattano altrettanto bene al supporto di applicazioni multimediali.
Il *Real Time Protocol* (RTP) [RFC1889] rappresenta lo standard proposto dall'*Internet Engineering Task Force* per la distribuzione di flussi multimediali su Internet. Esso provvede alla distribuzione di servizi punto-punto per i dati che necessitano di trasferimento in tempo reale; cioè stream audio e video, ma anche per altri tipi di dati. Questi servizi includono l'identificazione del formato dei dati trasportato (*payload type)*, la numerazione dei pacchetti (*sequence numbering*), l'assegnazione di un *timestamp*, e il monitoraggio della sessione. Le applicazioni in genere implementano RTP al di sopra di UDP, che fornisce le operazioni di *multiplexing* e *checksum*, anche se, per le sue caratteristiche, RTP può essere usato con altri protocolli di rete e di trasporto. Inoltre RTP supporta il trasferimento dati verso molteplici destinazioni usando, una distribuzione di tipo multicast. È da notare che questo protocollo non prevede nessun meccanismo che assicuri una corretta trasmissione o garantisca la qualità del servizio, ma per quello che riguarda la gestione della QoS la facilita, dotando il livello applicativo di maggiore visibilità. RTP non si interessa, inoltre, del mancato arri-/vo a destinazione in ordine dei pacchetti né dell'affidabilità con cui i livelli di rete prevedono il riordino. La numerazione, permette al ricevente di ricostruire la corretta sequenza dei pacchetti inviati

dal mittente; inoltre i *sequence number* possono essere usati per determinare la corretta posizione di un pacchetto all'interno di una sequenza senza necessariamente decodificarlo.
RTP consiste di due parti principali:

- il Real-Time Protocol (RTP), per trasportare dati che hanno vincoli di real-time
- l'Real-Time Control Protocol (RTCP), per monitorare la qualità del servizio e fornire informazioni sui partecipanti di una sessione in atto.

Quest'ultimo aspetto di RTCP può essere sufficiente per applicazioni dove non esiste un esplicito controllo dei partecipanti, ma non è sufficiente per implementare meccanismi di management dei gruppi. In una trasmissione RTP che preveda l'utilizzo contemporaneo di diversi media (ad esempio audio e video) essi sono trasmessi per mezzo di sessioni RTP separate e i pacchetti RTCP relativi usano due differenti coppie di porte UDP o indirizzi multicast, nel caso di comunicazione multicast. Le informazioni estratte dal protocollo RTCP verranno utilizzate per il monitoring della banda passante.

6.2.1.3 Le unità Client, Proxy, Server.

In figura 6.11 viene riportata l'architettura del Client, compreso il QoS manager anche se in questo caso non è direttamente coinvolto; abbiamo però preferito mostrare l'architettura completamente sviluppata in modo che sia possibile usarla come riferimento per evidenziare i cambiamenti dei meccanismi rispetto al precedente sistema di Riconfigurazione che veniva adottato in MUM.
Tornando al client, ricordiamo che tutte le entità che vogliano usufruire dei servizi offerti dal middleware dovranno implementare delle interfacce prestabilite (vedi sempre [GFRMI]). L'architettura del Client è, comunque, organizzata come segue: esso svolge il ruolo di coordinatore delle varie entità ed utilizza la Client Protocol Unit per inviare in avanti comandi alle altre entità.
I comandi che il client mette a disposizione, sono costituiti dal classico set di comandi per il controllo; oltre a questi comandi,

viene messo a disposizione anche il comando per muovere la sessione in un altro Place in cui l'utente sia registrato. Tale comando tuttavia non è implementato nell'applicazione di base e non viene quindi qui preso in considerazione. Il ClientVideoAgent, è invece l'entità preposta alla gestione del flusso multimediale, incapsula infatti il gestore del protocollo RTP e il Player che realizza il rendering del video, e presso il quale ci si potrà procurare l'oggetto visuale presentato all'utente. L'ultimo oggetto che consideriamo è il QoSManager che, utilizzando le API messe a disposizione dal JMF, monitora la sessione RTP e, ricevendo anche notifiche sulla condizione della CPU da parte del CPU Monitor (realizzato in MUM all'interno del ResourceBroker), richiede eventuali riconfigurazioni al ClientAgent.

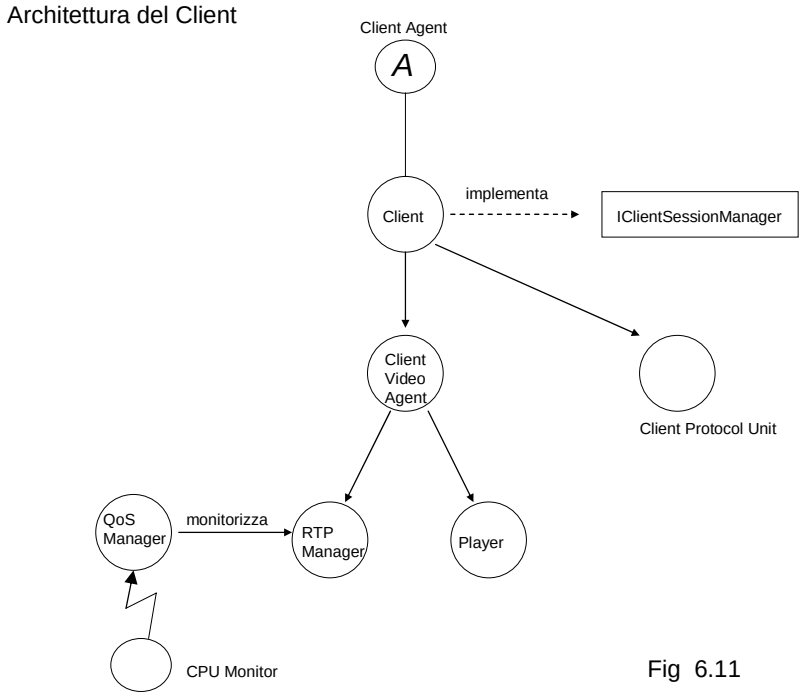


Fig 6.11

Le connessioni aperte dal Client sono tre, due come visto sopra sono connessioni richieste dal protocollo RTP, che tipicamente

saranno connessioni di tipo UDP, mentre la terza è una connessione TCP per l'invio dei comandi.
Passiamo ora ai componenti server e proxy, la cui architettura vediamo in figura 6.12. Presentiamo prima il componente Server, perché l'architettura del Proxy riprende alcuni elementi presenti in questa. Il Server è assai simile al Client; come per il Client si è strutturato il lato servitore in modo da dividere la gestione della sessione, cioè l'invio dei comandi dalla gestione dei flussi, che viene incapsulata, in questo caso, dal ServerVideoAgent. Come sopra abbiamo poi il QoSMonitor che monitora la situazione. La vita del Server, una volta attivato, è però superiore a quella dei client; il Server è, infatti, multithreaded, e ad ogni nuova richiesta ricevuta crea e fa partire un nuovo thread che gestirà da quel mo-

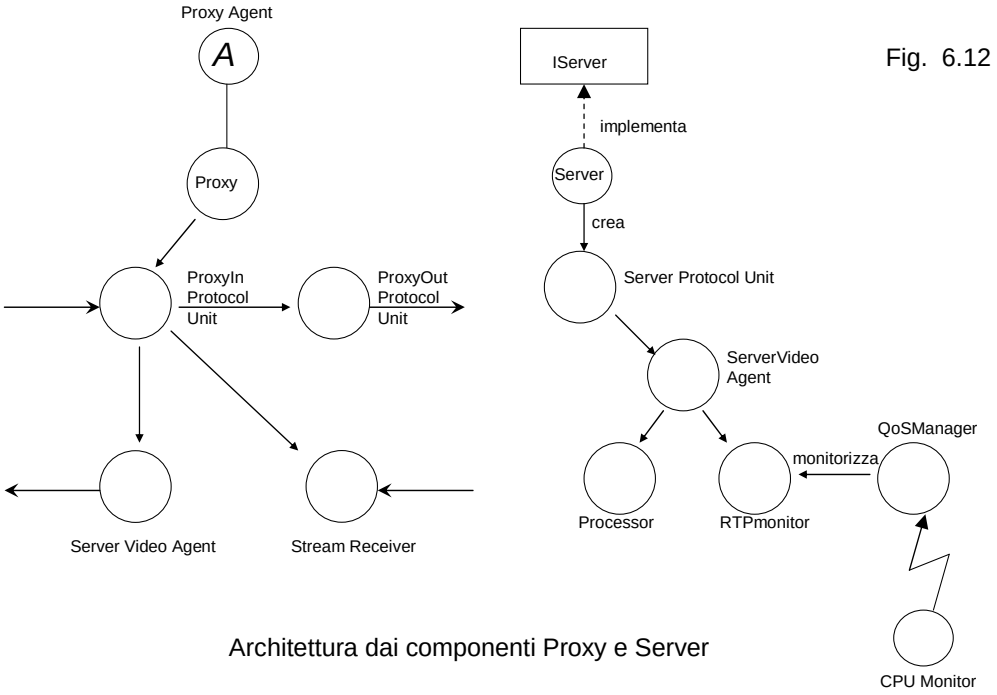


Fig. 6.12

Architettura dai componenti Proxy e Server

mento in poi tutta la sessione col cliente, cioè una Server Protocol Unit. L'ultima entità che ci rimane da analizzare è il Proxy. Tale entità riceverà dalle entità a valle i comandi, che dovrà inoltrare alle entità a monte, mentre dalle entità a monte i flussi, che dovrà

inoltrare a quelle a valle. Le Protocol Unit di ingresso e di uscita sono assai simili a quelle di Server e Client. La differenza più rilevante è che si è scelto di rivestire, un po' come accade anche per l'entità Server, la ProxyInProtocolUnit del ruolo di coordinatore per le altre entità costituenti il Proxy.

6.2.2 La realizzazione della New Path Action.

In questo paragrafo ci soffermiamo sulla realizzazione dell'oggetto NewPathAction. Questo oggetto è una ReconfigurationAction che implementa l'interfaccia per la riconfigurazione remota. Oltre a ciò, la classe implementa, come vedremo, i servizi importanti per il protocollo.

Quando nell'applicazione client (il MUMClientAgent) viene attivato il trasferimento della sessione, viene creato un oggetto New PathAction in cui vengono specificati i parametri necessari per il protocollo :

- il ComponentIdentifier del ClientAgent,
- il Place di destinazione,
- Place attuale in cui si trova il client,
- il percorso dal server alla nuova destinazione (ottenuto tramite uno dei servizi offerti dalla classe),
- il percorso per questo Place

L'azione viene poi passata al manager dei servizi multimediali, a cui viene richiesta l'attivazione dell'azione e al quale viene indicato come Place a cui inviare l'azione il place di destinazione del client. Quando il comando arriva sul Place chiama il metodo *doReconfigurationHere* dell'azione che, in questo caso, ritorna true se il place in cui si trova è il "destination place", indicato come parametro. Come abbiamo visto, a questo punto il comando chiede all'azione se utilizzerà uno dei componenti presenti oppure dovrà crearne uno nuovo; questo tipo di azione, richiede infatti la presenza di un componente sul place che gestisce le interazioni con i visitor agent che dovranno costruire un nuovo percorso parziale che si colleghi al vecchio; secondo l'architettura di MUM, i visitor agent sono progettati per fare riferimento a un initManager che, a

sua volta, riferisce dell'esito dell'inizializzazione a un initListener che è l'applicazione utente. Nella realizzazione dei meccanismi dell'azione, si è voluto mantenere lo stesso schema di architettura di MUM, tuttavia non si poteva avere a disposizione il client agent come riferimento per l'initManager su questo place perché si vuole anche effettuare il trasferimento, esclusivamente, quando il nuovo percorso è creato. Si è ricorso pertanto all'utilizzo di un finto client (fakeClient) che svolge momentaneamente la funzione del client.

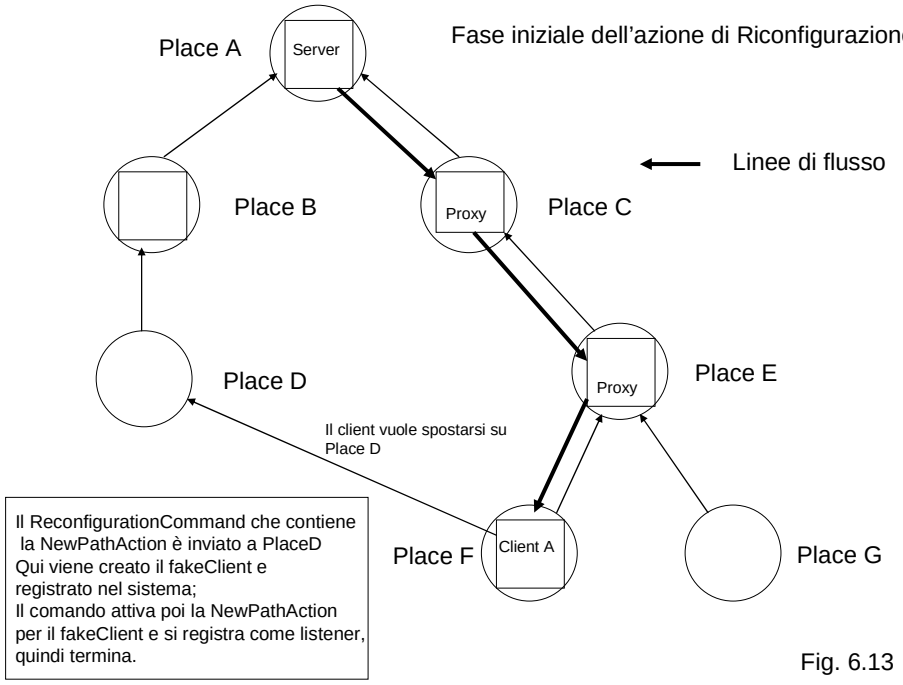


Fig. 6.13

Abbiamo quindi che il ReconfigurationCommand, ottiene questo componente dall'azione di riconfigurazione tramite il metodo getComponent e lo registra nel sistema, dopodichè attiva l'azione di riconfigurazione per il componente registrato, si segnala come listener e termina (vedi fig. 6.13). Il manager locale attiva l'azione di riconfigurazione, questa volta locale, per il componente; viene cioè chiamato il metodo prepare_action della NewPathAction. Tale metodo si limita ad attivare il fakeClient. Una volta attivato, il fakeClient, come se fosse un ClientAgent scarica sul Place, se non

c’è, il software necessario all’applicazione, crea l’InitManager il quale attiva il piano di riconfigurazione del percorso.
Tale piano è un oggetto che implementa le interfacce Plan e ISinglePlan realizzate nel middleware. Il piano prevede di risalire il percorso tra il place in cui ci si trova e il server, fino al place in cui vi è un’intersezione del nuovo percorso con il vecchio. Come si può vedere in figura 6.14, a seconda della posizione del fakeClient, l’intersezione con il vecchio percorso sarà, o al place dello stesso server, o in un Place in cui sia presente un proxy. Il piano prevede allora la disposizione di seguenti componenti :

- un clientSessionManager nel place del fakeClient
- un proxySessionManager nei place “liberi”, lungo il percorso verso il Server; su tali place sarà anche attivato il corrispondente ProxyAgent
- la duplicazione dell’uscita per il sessionManager che gestisce il flusso sul Place in cui si verifica l’intersezione dei percorsi; nel caso che l’intersezione sia al place del server, i visitor agent dovranno recuperare il record che mantiene le informazioni sul client dall’activeServer e chiederanno la duplicazione dell’uscita alla protocol unit

Facciamo presente, che l’insieme delle operazioni di riconfigurazione possibili su un componente può essere realizzato mediante poche operazioni attuate sui manager di protocollo dei componenti; in particolare le seguenti operazioni :

- la duplicazione dell’uscita
- la duplicazione dell’ingresso
- la chiusura del manager di protocollo

consentono di realizzare praticamente qualunque tipo di riconfigurazione e adattamento di un flusso, compreso l’inserimento di filtri, ecc..
Tali operazioni devono quindi essere previste nell’interfaccia dei manager di protocollo; non però necessariamente implementate.

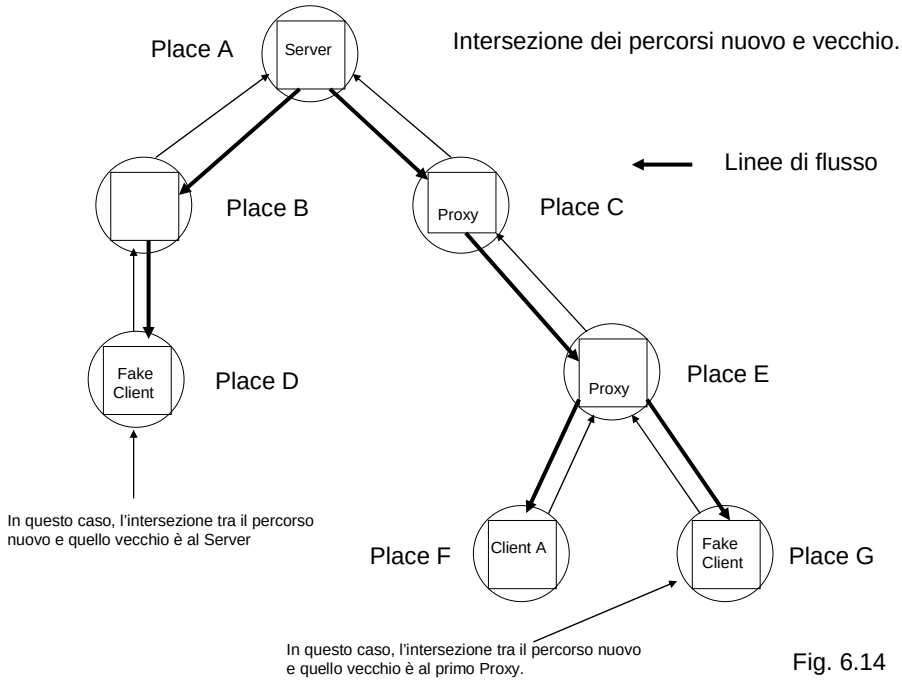


Fig. 6.14

Nel nostro caso, delle tre operazioni, l'unica fondamentale è la prima mentre non è necessario che i componenti dell'applicazione realizzino anche le altre.
Torniamo ora al meccanismo realizzato dalla NewPathAction. Una volta che il manager di protocollo ha duplicato la sessione in uscita, il percorso verso la nuova destinazione viene realizzato secondo il meccanismo previsto da MUM; viene quindi creato, in caso di riuscita, il ClientSessionManager e un ActiveClient relativo all'applicazione che si deve spostare viene registrato sul Place.
Viene infine notificato l'esito all'InitManager e quindi al Fake Client, il FakeClient a questo punto recupera il listener, gli notifica l'esito della fase di preparazione, si de-registra dal place e termina; l'esito viene quindi rimbalzato fino al MUMClientAgent tramite il comando ReconfigurationEndedCommand. Non viene, infatti, in questo caso attuato un processo in due fasi poiché si sa già che l'utente vuole spostarsi. L'agente del client invia allora lungo il suo vecchio percorso il comando di chiusura parziale del flusso;

tale comando chiude i manager di protocollo fino al primo in cui trova una sessione doppia; in tale manager chiude la sessione relativa alla parte di percorso da cui proviene.
Inviato il comando, l'agente del client si sposta sul place di destinazione dove "aggancia" la sessione recuperando il ClientSession Manager dall'ActiveClient presente e riprende quindi la sessione senza ritardi.

6.2.3 Le modifiche ai componenti Client, Proxy e Server.

In questo paragrafo presentiamo le modifiche apportate alla struttura dei componenti per la realizzazione della NewPathAction. Sostanzialmente, questi cambiamenti sono relativi alla struttura del manager di protocollo; come abbiamo detto, per la realizzazione delle azioni di riconfigurazione, è necessario ampliare le interfacce delle unità che gestiscono i protocolli. In realtà, per quanto riguarda il ClientSessionManager, non ci sono state grosse modifiche nella sua struttura in quanto non viene coinvolto particolarmente nella riconfigurazione. Per quanto invece riguarda il Proxy SessionManager e i componenti del server, vi sono stati alcuni cambiamenti che vale la pena di evidenziare. Queste unità infatti devono realizzare il servizio di duplicazione della sessione in uscita. I cambiamenti hanno quindi coinvolto un'estensione delle interfacce ma oltre a questa è stata modificata la struttura interna. Come abbiamo visto, nella fig. 6.12, i componenti server e proxy, gestiscono il flusso tramite un manager di protocollo (nel caso del Proxy è un oggetto : ProxySessManOnlyVideo mentre nel caso del Server è un oggetto SimpleServerVideoProtocolUnitBase). Questi manager, a loro volta contengono i moduli necessari per il funzionamento del componente; così, nel proxy vi saranno i componenti che stanno in attesa dei comandi del VCR (InProtocolUnit) , che li propagano al componente successivo, che ricevono il flusso in ingresso e che lo rimandano in uscita (ServerVideoAgent). Per la duplicazione della sessione in uscita, si dovrà mantenere il componente che riceve il flusso e quello che propaga i comandi

ricevuti dal client mentre gli altri due andranno sostituiti. Esiste però una fase transitoria in cui i due percorsi saranno attivi contemporaneamente; questo accade quando il ClientSession Manager è stato creato sul nodo di destinazione ma il ClientAgent non si è ancora spostato. Durante tale fase, il manager di protocollo dovrà quindi mantenere doppi questi componenti coordinandoli tra loro in modo tale che quando l'agente del client lascia il place orginario, la InProtocolUnit sul vecchio percorso venga chiusa e venga considerata unicamente quella sul nuovo percorso. Naturalmente lo stesso vale per il Server, che si limiterà a duplicare il componente che riceve i comandi dal client e a creare un nuovo VideoAgent collegato al nuovo percorso. Queste sono quindi le operazioni necessarie in questo scenario. Abbiamo quindi la creazione di un nuovo ServerVideoAgent e la sostituzione della inProtocolUnit; nella nostra implementazione si è però scelto di non sostituire la InProtocolUnit ma di realizzare in essa un meccanismo di duplicazione. In questo caso, la creazione del percorso originario attiva non la InProtocolUnit, ma un thread al suo interno. Allo stesso modo, l'attivazione del nuovo percorso, attiva un secondo thread sempre all'interno della InProtocolUnit; di questi due thread, uno è il dominante, cioè quello che propaga i propri comandi al componente successivo mentre il secondo è inattivo; ovviamente, inizialmente sarà il thread sul vecchio percorso a essere dominante, quando però viene richiesta la chiusura parziale del flusso, il thread dominante viene chiuso e al suo posto viene utilizzato il secondo thread che da quel momento propaga i comandi che riceve verso il server. Dopo aver richiesto la chiusura del tratto non più utilizzato del percorso, il client migra e si aggancia, come abbiamo visto, al nuovo percorso. Ovviamente, con le dovute modifiche, lo stesso vale per il server. Le modifiche sono dovute al fatto che il server non riceve un flusso e che i comandi che riceve non vengono propagati ma eseguiti, ma, per quanto riguarda la struttura dei componenti, le modifiche e l'architettura, sono le stesse illustrate nel caso del proxy. In figura 6.15, vediamo un rappresentazione sommaria di quanto esposto.

Schema di duplicazione della sessione in uscita in un ProxyAgent

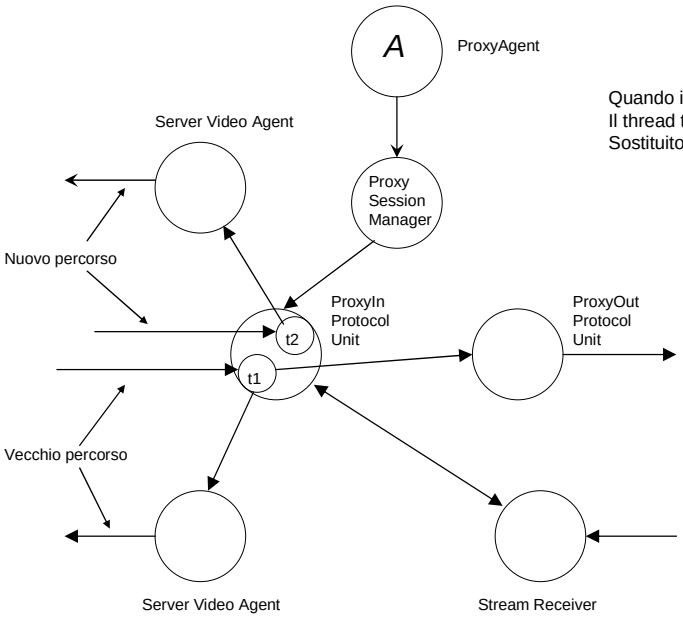


Fig. 6.15

6.2.4 Test dell'azione NewPathAction.

Quest'ultimo paragrafo viene dedicato alla descrizione di alcuni primi test per valutare i tempi di riconfigurazione nel caso dell'azione di riconfigurazione presentata. In particolare si è voluto valutare il tempo tra l'attivazione della richiesta da parte del client e la ripresa della sessione da parte dell'applicazione utente sul nuovo Place.

Poiché, in questo caso, l'azione ha origine su un Place e si conclude su un altro, per la misurazione del tempo non si è utilizzato, come sistema di riferimento l'orologio dei Place coinvolti nella riconfigurazione ma si è fatto ricorso ad un terzo Place. All'inizio della fase di riconfigurazione del percorso, viene generato un comando TimeTestCommand che viene inviato al DefaultPlace radice. All'arrivo del comando viene semplicemente registrata l'ora

di sistema. Quando la riconfigurazione del percorso è conclusa, un secondo comando TimeTestCommand viene inviato verso la radice; qui preleva l'orario registrato dal primo comando che viene sottratto dall'ora attuale; il risultato è il tempo considerato nel nostro test.

6.2.5.1 Configurazione del test e risultati.

Due computer sono stati coinvolti nell'esecuzione dei test. Essi erano così equipaggiati:
PC1 fisso
-processore Intel Pentium IV 2600 MHz, 512 MB RAM
-sistema operativo Windows XP
PC2 portatile acer
-processore Pentium III 450 MHz, 64 MB RAM
-sistema operativo Windows 98
Rete i due computer sono stati poi collegati da una rete LAN Ethernet da10 Mbps

Codice da scaricare: il codice sviluppato per la realizzazione del Client, del Proxy (il Server non viene considerato perché nei nostri test deve necessariamente già essere attivo). Naturalmente si ipotizza che le librerie multimediali (la JMF), siano già presenti sui vari place coinvolti. Per quello che riguarda il codice da scaricare le dimensioni sono le seguenti:

- *Client*: 47 KB;
- *Proxy*: 47KB;

I test presentati si riferiscono al seguente caso (vedi fig. 6.16): la topologia è realizzata interamente sul computer fisso tranne il nodo in cui il client si sposterà che risiede sul computer portatile.
1. Il caso in cui nei nodi coinvolti nel nuovo percorso non sia presente ancora alcun software.
2. Un secondo caso in cui il codice è già presente su tutti i place coinvolti nel nuovo percorso.

Entrambi i casi sono poi stati verificati sia nell'evenienza che l'intersezione tra il nuovo percorso e il vecchio sia in un Proxy, sia che tale intersezione si verifichi direttamente al Server.

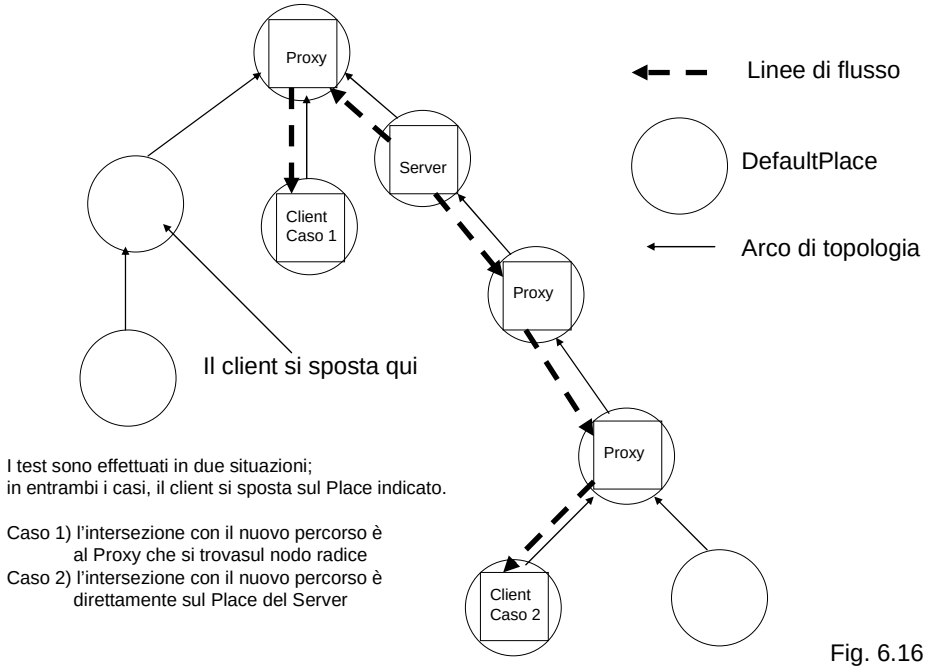


Fig. 6.16

Di seguito riportiamo i risultati per le prove svolte (i tempi sono riportati in msec).
Caso uno: intersezione dei percorsi al Place del server:
Con il software già presente sui nodi del nuovo percorso:
Prova1 Prova2 Prova3 Media
2531 2188 2438 2386
Senza il software presente nei nodi del nuovo percorso:
Prova1 Prova2 Prova3 Media
3032 3328 3538 3299

Caso due: intersezione dei percorsi al Place di un proxy:
Con il software già presente sui nodi del nuovo percorso:
Prova1 Prova2 Prova3 Media

2088 1987 1684 1920
Senza il software presente nei nodi del nuovo percorso:
Prova1 Prova2 Prova3 Media
2437 2378 2564 2459

I dati riscontrati evidenziano una lieve differenza temporale nei casi in cui l'intersezione del nuovo percorso con il vecchio avvenga al nodo del server o al nodo di un proxy; tale differenza sembra però essere più riconducibile alla diversità dei componenti necessari nei due casi; nel primo caso infatti deve essere installato un nuovo componente nel DefaultPlace radice mentre nel secondo caso non ci sono nuovi componenti da installare. Si è quindi ripetuta la simulazione in modo che i percorsi fossero simili e si è infatti verificato che i tempi sono in questo caso assimilabili.
Notiamo infine che, seppur non in maniera pesante, il tempo per lo scaricamento del codice, ha un'influenza sulle prestazioni complessive.
Futuri test dovranno naturalmente essere svolti per verificare se questi primi risultati sono estendibili ad un sistema distribuito costituito realmente da più nodi.

6.3 Lo schema di integrazione dei QoS Manager.

In questa ultima sezione, ci soffermiamo brevemente, su come il meccanismo originale di avvio della fase di riconfigurazione presente in MUM possa essere integrato con il modello di riconfigurazione realizzato.
Come abbiamo visto nel capitolo relativo a MUM, ogni applicazione utente è dotata di un QoSManager che interagisce con l'RTPManager in modo tale che, se vi sono problemi di banda, viene generato un segnale per il QoSManager. Allo stesso modo, quando il QoSManager viene creato, esso si registra come listener presso il thread che monitorizza la CPU ed è pertanto in grado di accorgersi di eventi relativi a banda e CPU. Quando poi verifica l'insorgenza di un problema attivava il meccanismo di riconfigura-

zione dell'applicazione stessa, elaborava un piano di riconfigurazione e lo attuava. Senza modificare assolutamente la struttura vista, adesso, il QoSManager si può registrare presso il resourcesBroker e quando vi è necessità di attivare la riconfigurazione, invece di agire autonomamente, l'applicazione utente dovrà, elaborata la migliore azione di riconfigurazione, locale o remota, richiederne l'avvio al manager dei servizi multimediali (registrandosi come listener). Come si vede, in questo modo, l'architettura prevista per il QoS management riesce a integrarsi perfettamente con il modello di riconfigurazione realizzato.

Conclusioni

Nel corso del lavoro di tesi sono state toccate diverse tematiche legate alla gestione della QoS e alla riconfigurazione dei flussi multimediali, con lo scopo di estendere una preesistente infrastruttura di supporto all'erogazione di servizi multimediali. In particolare, è stata aggiunta una funzionalità per il QoS routing dei flussi che, attraverso l'utilizzo di opportuni protocolli e l'introduzione di nuovi componenti, opera un costante monitoraggio delle risorse a tempo di esecuzione riconfigurando i percorsi dei flussi in modo da rispondere a eventuali peggioramenti nella disponibilità delle risorse.

La prima parte della tesi, è stata quindi dedicata allo studio della letteratura relativa al QoS routing e alla decisione su come realizzare il reperimento di informazioni sullo stato delle risorse nel sistema. In particolare, visto che ottenere informazioni sullo stato complessivo del sistema risulta, nella maggior parte dei casi, troppo oneroso, si è deciso di circoscrivere tale operazione. Si è quindi introdotto il concetto di località come un insieme di nodi/piattaforme per cui vale una relazione di vicinanza. Già in questa fase iniziale si è quindi affrontato un primo tema di importanza non marginale, cioè la definizione del concetto di vicinanza. Tra le diverse scelte, si è adottata quella intuitivamente ed implementativamente più accessibile e quella che rispetto alle altre prese in considerazione ha presentato i maggiori vantaggi per la realizzazione del protocollo di probing. Si è poi visto come realizzare nell'ambito di una località il reperimento e l'aggiornamento delle informazioni sullo stato delle risorse; a tal fine, si è adottato un protocollo ibrido tra un semplice probing e un meccanismo di notifica degli eventi. Il protocollo adottato è stato progettato specificamente per l'architettura di SOMA, l'infrastruttura distribuita che abbiamo esteso, e ci è sembrato la scelta con il miglior compromesso, tra prestazioni ed efficienza nell'uso delle risorse.

Il meccanismo di monitoraggio realizzato, ha poi consentito l'identificazione delle situazioni di caduta di un nodo, o di un uscita di un Place da un dominio: situazioni previste nella piattaforma da un punto di vista teorico ma precedentemente senza un supporto. Il passo successivo è stato quello di realizzare un servizio per ottenere la riconfigurazione della topologia dell'infrastruttura distribuita a fronte di tali rilevazioni. Anche in questo caso ci si è trovati di fronte alla possibilità di realizzare protocolli con approcci al problema concettualmente diversi. La nostra scelta è caduta quindi sul protocollo che ci è sembrato più appropriato a uno scenario in cui l'impiego di risorse deve essere il più possibile limitato e con impatto locale. Sul sistema così realizzato è stato poi impiegato il middleware MUM per la distribuzione di stream mul-timediali in ambito distribuito. Appoggiandosi all'architettura del middleware, abbiamo quindi progettato un protocollo per il QoS based routing di flussi multimediali sulla base del sistema di località precedentemente realizzato. Si noti inoltre che il servizio di località realizzato non è utile esclusivamente ad applicazioni per la gestione e delivery di flussi multimediali, ma può essere utilizzato anche da ltre tipoligie di applicazioni che necessitino una visibilità ampliata sul sistema.
Un'insieme di test sulla riconfigurazione del percorso di un flusso conclude il lavoro di tesi. Tali test, pur avendo esito incoraggiante, andrebbero però estesi con un insieme di nuovi test su reti più estese rispetto a quella locale nella quale si è finora andati ad operare. Ciò non solo per verificare la validità dell'approccio adottato, ma anche per operare una vera e propria messa a punto del sistema. Una conclusione importante alla quale si è giunti è infatti che un sistema di località non può essere "programmato" a prescindere dalle sue effettive condizioni di impiego in quanto parametri importanti, come il raggio di località, i tempi per l'aggiornamento, ecc., non possono essere valutati a priori.
Riteniamo infine che il presente lavoro apra interessanti direzioni per un futuro lavoro. Nell'ambito del sistema di località si può ad esempio pensare di estendere il lavoro svolto nell'ambito della

riconfigurazione della topologia con soluzioni per la clusterizzazione dei nodi ai livelli più alti della topologia di SOMA, in modo da rendere l'infrastruttura sempre più solida nei confronti di possibili malfunzionamenti. Un'altra direzione di lavoro potrà invece estendere gli algoritmi di gestione della riconfigurazione da parte delle entità del middleware che per ora si limitano a una scelta assoultamente statica e "non intelligente" del componente da riconfigurare definendo inoltre riconfigurazioni "ad-hoc" per le diverse tipologie di entità.

Appendice A.

Come abbiamo anticipato nel capitolo quattro, la politica di recovery può essere realizzata tecnicamente in due modi; nel capitolo quattro abbiamo presentato un protocollo di recovery che limitasse al minimo le interazioni tra Place durante la fase transitoria; la realizzazione di tale protocollo è stata resa possibile tramite l'ampliamento delle informazioni mantenute nel DNS di SOMA. Il protocollo che presenteremo ora invece, non necessità di tale estensione; esso ha tuttavia lo svantaggio di richiedere un forte coordinamento globale tra i Place coinvolti.

Lo scopo della fase di recovery, come abbiamo visto, è quello di aggiornare le informazioni di sistema affinché rimangano coerenti e ricostruire se possibile la topologia. Nella trattazione successiva useremo come riferimento, gli esempi mostrati nelle figure.

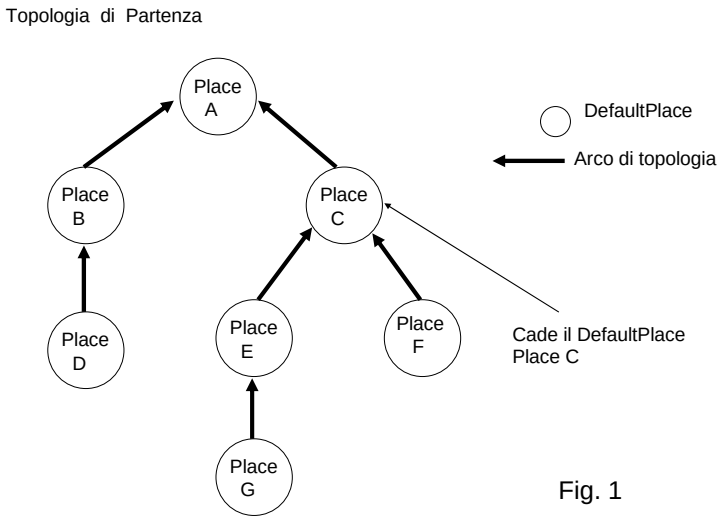


Fig. 1

Abbiamo anticipato che il meccanismo di recovery comporta una fase di coordinamento globale. Con globale intendiamo che vengono coinvolti tutti i DefaultPlace proprietari di località il cui raggio era tale da includere in sé il DefaultPlace caduto. E' stato tuttavia deciso che sia solo un DefaultPlace a occuparsi della gestione del recovery in modo da centralizzare le operazioni.

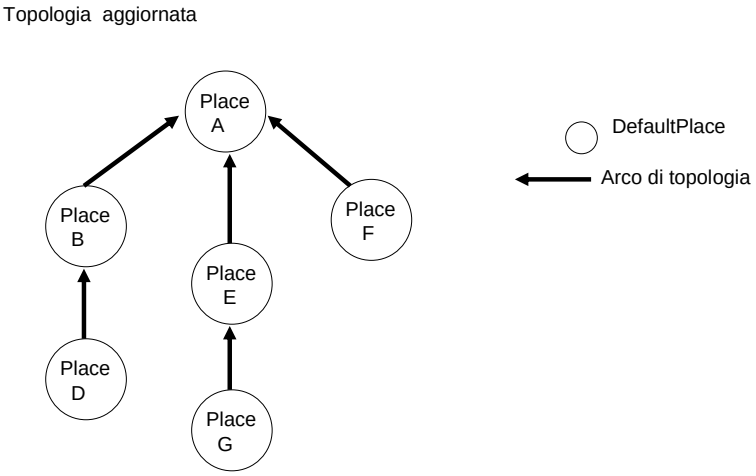


Fig. 2

Particolare attenzione è stata posta nella scelta di questo Default Place; in particolare si è scelto il DefaultPlace per cui il reperimento delle informazioni mancanti per la ricostruzione della topologia sia minimo. Con il termine informazioni mancanti, ci riferiamo alle informazioni relative a tutte le località coinvolte nel processo e non presenti direttamente nel Place. Nell'esempio delle fig. 1 e 2 è necessario ottenere le informazioni su tutte le località centrate su Place A, B, C, E, F, G. In questo caso, il manager su PlaceA dovrebbe ottenere le informazioni esclusivamente sulla località di PlaceG e insieme al manager su PlaceE, a cui mancano le informazioni su PlaceB, è quello a cui manca meno informazioni. In sostanza, nelle ipotesi che tutte le località abbiano la stessa estensione, il DefaultPlace padre e i DefaultPlace figli del Place caduto sono quelli più adatti ad attuare il Recovery.Tra questi abbiamo poi scelto il DefaultPlace padre in quanto, a parte il nodo radice, tutti gli altri hanno un DefaultPlace padre mentre non tutti i nodi hanno dei figli; inoltre così evitiamo di dover coordinare i figli per stabilire chi tra loro, se ve ne è più di uno, si sarebbe dovuto occupare del recovery. Stabilito quale è il DefaultPlace che si occupa del recovery vediamo come si procede :

TABELLA DELLE LOCALITA' NEL SISTEMA

Place propietario della Località	Place visti nella Località (ogni località ha raggio = 2)					
Place A	Place A	Place B	Place C	Place D	Place E	Place F
Place B	Place A	Place B	Place C	Place D		
Place C	Place A	Place B	Place C	Place E	Place F	Place G
Place D	Place A	Place B	Place D			
Place E	Place A	Place C	Place E	Place F	Place G	
Place F	Place A	Place B	Place E	Place F		
Place G	Place C	Place E	Place G			

TABELLA DELLE LOCALITA' AGGIORNATA

Place propietario della Località	Place visti nella Località (ogni località ha raggio = 2)					
Place A	Place A	Place B	Place D	Place E	Place F	Place G
Place B	Place A	Place B	Place D	Place E	Place F	
Place D	Place A	Place B	Place D			
Place E	Place A	Place B	Place C	Place E	Place F	Place G
Place F	Place A	Place B	Place E	Place F		
Place G	Place A	Place E	Place G			

Fig. 3

1) Un DefaultPlace K si accorge che uno degli altri DefaultPlace nella località non ha risposto al comando Are You Alive; viene attivata la riconfigurazione locale che consiste nell'eliminare dalla tabella della località tale DefaultPlace e i Place del suo dominio, che vengono considerati non raggiungibili. Vengono disattivati il manager

UpdateManager e AYAManager fino alla fine della fase di Recovery. Viene attivato il manager di topologia perché attui la riconfigurazione della Topologia.

INFORMAZIONI MANCANTI PER IL RECOVERY

Place propietario della Località	Mancano le informazioni relative alle seguenti Località					
Place A	Place G					
Place B	Place E	Place F	Place G			
Place D	NON E' COINVOLTO NEL RECOVERY					
Place E	Place B					
Place F	Place G	Place B				
Place G	Place A	Place F	Place B			

Fig. 4

2) Il manager di topologia controlla che la riconfigurazione non sia già in atto. Ciò è possibile in quanto se già un altro DefaultPlace ha avviato il recovery è possibile che il Place sia già stato avvisato. In questo caso semplicemente non si prosegue. Se invece la riconfigurazione non è ancora stata attivata, viene attivata ora.
3) Controllo se sono il DefaultPlace padre, uno dei figli del Place caduto, o se non ho alcuna relazione diretta con il Place.
   a) Nel caso che non sia in relazione diretta con il Place caduto, mi limito ad inviare un comando per notificare l'avvio della riconfigurazione al DefaultPlace che mi collega al Place caduto; in sostanza, i DefaultPlace che si trovano, nella topologia, al di sotto del Place caduto

avviseranno i figli del Place caduto, mentre quelli che si trovano al di sopra avviseranno il padre del Place caduto. Questo viene fatto perché si vuole ottenere la modifica dei DNS il più velocemente possibile e tale modifica è gestita da questi DefaultPlace.
In particolare i Default Place figli inviano un comando RemoveDomainCommand nel proprio sottoalbero mentre il DefaultPlace padre si occupa della parte superiore della topologia.

b) Nel caso che il DefaultPlace sia un figlio del Place caduto, aggiorno il DomainNameService, modificando anche l'indicazione del DefaultPlace padre e invio il comando RemoveDomainCommand nel mio sottoalbero. Oltre a ciò, invio un comando al DefaultPlace padre del Place caduto per attivare la fase di ricostruzione delle località.
c) Nel caso che il defaultPlace sia il Place padre del Place caduto aggiorno il DomainNameService, adottando anche i figli del Place caduto come figli propri, e invio il comando RemoveDomainCommand nella parte superiore della topologia e nei sottoalberi che nascono dal Place diversi da quello del Place caduto. Proseguo poi punto 4).

4) Siamo sul DefaultPlace padre del Place caduto. Vediamo per prima cosa come cambia la sua Località. Nella parte superiore della topologia, tutto è rimasto uguale. Nella parte inferiore della topologia che non passa dal Place caduto anche.
L'unica parte in cui cambia qualcosa è il sottoalbero indicato dal Place caduto. In tale sottoalbero, le località si sono avvicinate di un passo e, pertanto, entrano a far parte di questa località anche quelle che si trovavano a una distanza pari a R+1 con R il raggio della località. Ottenute le informazioni su tali Località, sempre tramite il manager di Topologia, ho le informazioni sulle Località viste dal Place, ma non è ancora stata effettuata l'aggiornamento della tabella della località che comporta anche altri passaggi. Il DefaultPlace deve

anche aggiornare ogni altro DefaultPlace coinvolto nel recovery e per prima cosa, li deve individuare; in particolare distinguiamo i casi delle località :

a) che vedevano il Place caduto tramite il DefaultPlace padre (parte superiore della topologia e altri sottoalberi fratelli del Place caduto)
b) che vedevano il Place caduto tramite i figli dello stesso.

Nel caso a), il Place sa già quali località sono coinvolte; questo è dovuto al fatto che vedendo il Place caduto tramite lui, necessariamente anche il DefaultPlace padre è incluso nelle stesse località e quindi ne mantiene le informazioni.
Nel caso b), il Place non è in grado di sapere, in funzione delle informazioni locali, quali Località sono coinvolte perché è possibile che alcune località, nella parte inferiore della topologia vedessero il Place caduto ma non il padre di questo. Consideriamo per esempio la topologia di fig. 5. Se il raggio della località del sistema è pari a 2, le località di PlaceI e PlaceH includevano PlaceC e quindi sono coinvolte nel recovery ma non PlaceA che pertanto non ha informazioni su di loro e le deve chiedere ai figli del Place caduto, i quali, necessariamente mantengono tali informazioni.
In questo caso, il Place padre interroga quindi tutti i figli per sapere quali altre Località sono coinvolte per il tramite di questi. Una volta ottenute tali informazioni, il Place padre è a conoscenza di tutte le località interessate nel recovery e deve procedere al loro aggiornamento.

5) Adesso devo aggiornare la tabella della Località. Per prima cosa elimino dalla tabella il DefaultPlace caduto e ogni Place del suo dominio. Richiedo poi le informazioni di stato dei DefaultPlace ora visibili nella parte bassa della topologia e le informazioni di stato dei Place normali o mobili dei domini che si trovavano a distanza pari al raggio della località. Tali Place non erano infatti visibili precedentemente ma lo diventano ora.
6) Passo ora ad aggiornare i DefaultPlace indicati dalle due

liste del punto 4). Consideriamo la lista indicata dal caso a). Come prima cosa, controllo, per ogni località indicata, se possiedo, localmente, tutte le informazioni necessarie all'aggiornamento delle entry della località.

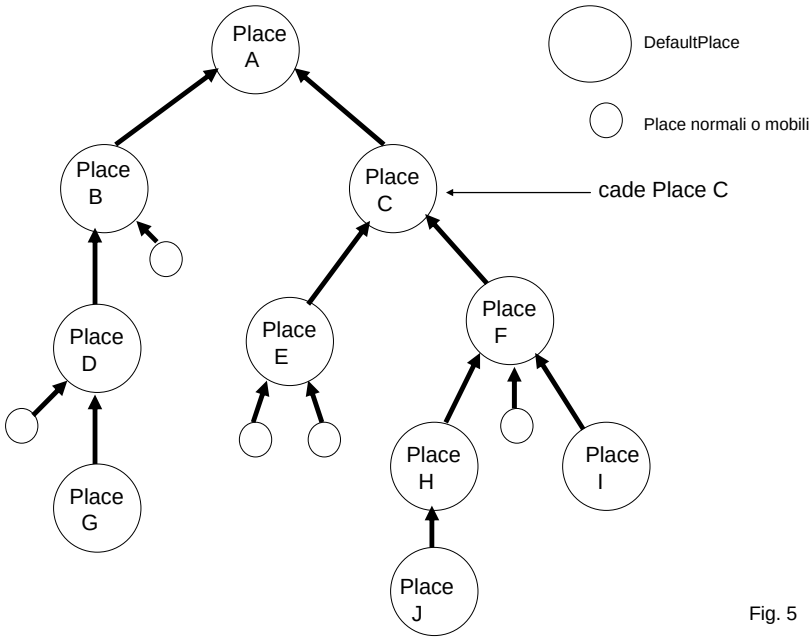


Fig. 5

Nel caso in cui tutte le località abbiano lo stesso raggio, ciò è sempre verificato ma è possibile, se i raggi sono diversi che la visibilità locale non sia sufficiente. Se per esempio consideriamo la topologia di fig. 6 e ipotizziamo che PlaceB abbia un raggio di località pari a 4 mentre PlaceA abbia un raggio di 2 è facile vedere come PlaceA non sia in grado di fornire direttamente le informazioni sui nuovi Place che vedrà PlaceB. Mentre infatti PlaceA vede fino a PlaceH nella nuova topologia, PlaceB vedrà anche PlaceJ. Se questa condizione è verificata, preparo un comando:
ReconfigureLocalityCommand con tutte le informazioni necessarie all'aggiornamento e lo invio al DefaultPlace. Se invece la condizione precedente non è verificata nel

ReconfigureLocalityCommand indico quali sono i Default Place che dovrà interrogare per aggiornarsi. Per quanto riguarda invece la lista indicata al punto 4)b procedo analogamente; se possiedo tutte le informazioni relative ai nuovi Place che saranno visti, le invio direttamente, altrimenti indico quali Default Place dovranno interrogare per ottenere gli aggiornamenti.

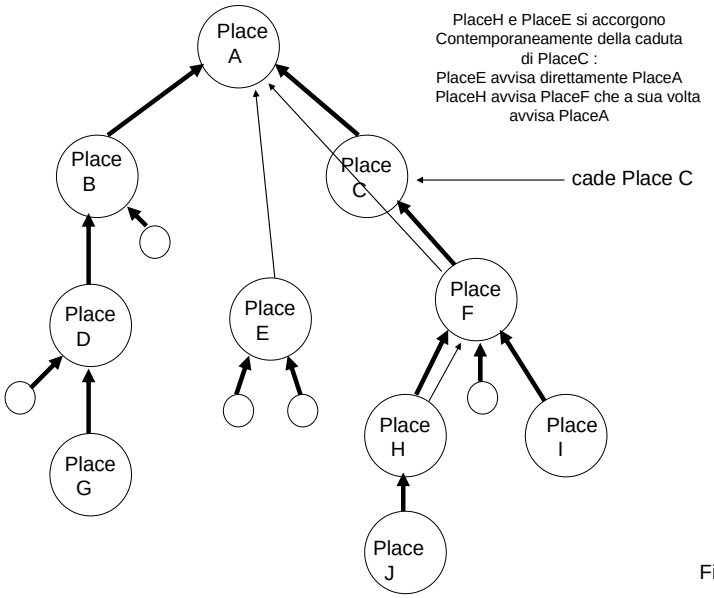


Fig. 6

In sostanza qui terminano le azioni svolte dal DefaultPlace padre che a questo punto termina la propria fase di riconfigurazione e riattiva i manager per la gestione degli aggiornamenti e per il protocollo AreYouAlive.

7) Vediamo ora cosa succede all'arrivo di un comando LocalityReconfigureCommand in un DefaultPlace. Nel caso che il comando possieda tutte le informazioni necessarie, viene semplicemente aggiornata la tabella della località e conclusa la fase di riconfigurazione della località, riavviando quindi il normale funzionamento. Se invece il comando indica di che l'aggiornamento del Place deve essere effettuato dal Place stesso, viene inviato un comando GetEntries Command a tutti i DefaultPlace la cui visibilità rispetto alla

località è cambiata.

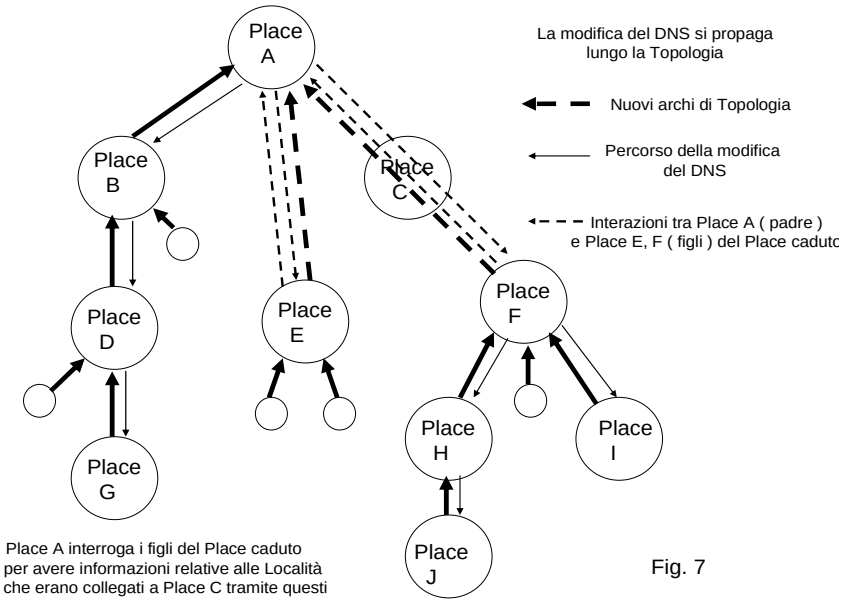


Fig. 7

In sostanza, in questo caso, nel comando vengono indicati i DefaultPlace che prima non erano visti e quelli che si trovavano ad una distanza pari al raggio di località; di questi ultimi saranno, infatti ora visibili i Place del dominio. Una volta che saranno giunte tutte le risposte, la tabella della località sarà aggiornata e si chiuderà la fase di riconfigurazione della topologia ritornando al funzionamento normale.

La fase di recovery, secondo questo protocollo, termina quindi quando la fase 7) è terminata per ogni Place coinvolto e tutte le località sono state riconfigurate secondo la nuova topologia. Anche in questo caso, evidenziamo come questo protocollo non sia attuabile nel caso della caduta del nodo radice poiché di fatto, in questo caso, non esiste un nodo padre di riferimento. Facciamo infine notare come il funzionamento dei ReconfigureLocality

Commad sia diverso da quello descritto nel capitolo quattro; tale diversità è dovuta alla diversità ovviamente alle diverse esigenze del protocollo.

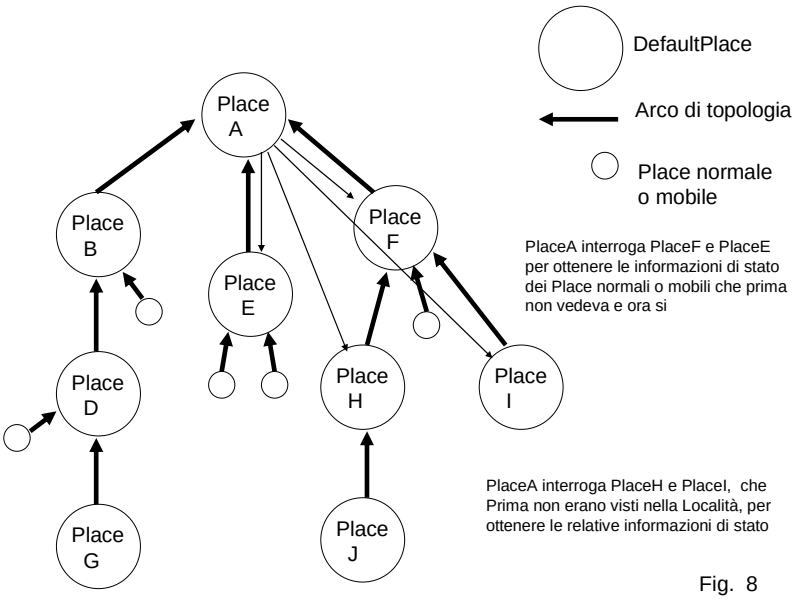


Fig. 8

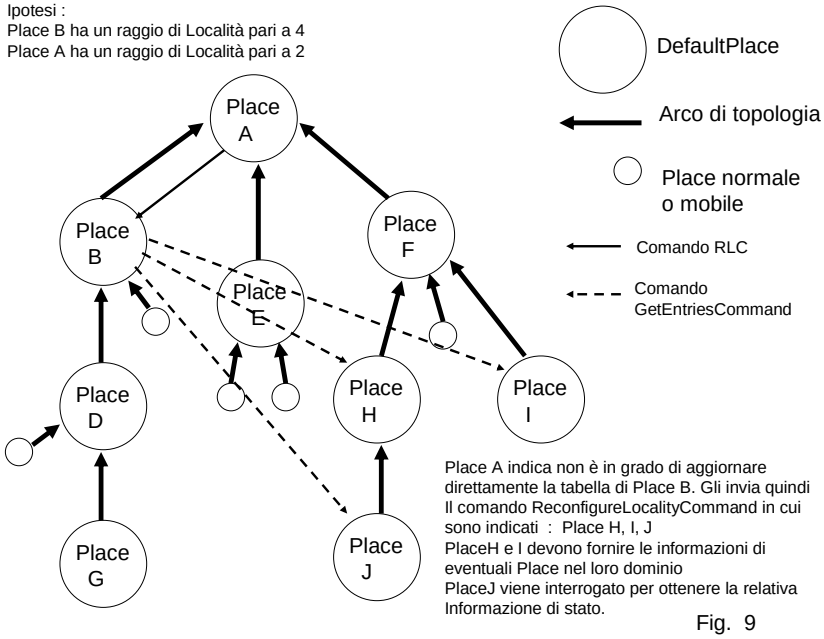


Fig. 9

Nota :
Nella realizzazione di questo protocollo, altri componenti sono stati realizzati con meccanismi diversi rispetto a quelli presentati nella presentazione del progetto; su tali diversità non ci soffermiamo in quanto la versione scelta definitivamente è stata quella proposta.

Bibliografia.

[BAF01] Baschieri F., Tesi di Laurea, 2001.
[SHP02] Shenoy P., Hasan S., Kulkarni P., Ramamritham K., Middleware versus Native OS Support: Architectural Considerations for Supporting
Multimedia Applications, Proceedings of Real-Time Applications and Systems (RTAS), 2002
[OSI] Basic Reference Model of Open Distributed Processing, Part 1: Overview and guide to use. ISO/IEC JTC1/SC212/WG7 CD 10746-1, International Standard Organization, 1992.
[RFC2210] Resource ReSerVation Protocol (RSVP). RFC2210, si veda http://www.ietf.org/rfc/rfc2210.txt
[RFC2205] Resource ReSerVation Protocol (RSVP). RFC2205, si veda http://www.ietf.org/rfc/rfc2205.txt
[RFC2475] An Architecture for Differentiated Services. RFC2475, si veda http://www.ietf.org/rfc/rfc2475.txt
[RFC1889] RTP: A Transport Protocol for Real-Time Applications. RFC1889, si veda http://www.ietf.org/rfc/rfc1889.txt
[RFC2326] Real Time Streaming Protocol (RTSP). RFC2326, si veda http://www.ietf.org/rfc/rfc2326.txt
[GEK01] Geihs K., Middleware Challanges Ahead, IEEE Computer, June 2001 (Vol. 34, No. 6).
[FUA98] Fuggetta A., Picco G.P., Vigna G., “Understanding Code Mobility”, IEEE Transactions On Software Engineering, Vol.24, No.5, May 1998
[TRA98] Tripathi A.R. and Karnik N.M., “Design Issues in Mobile-Agent Programming Systems”, IEEE Concurrency, July-September 1998
[RFC2386] E. Crawlery, R. Niar, B. Rajagopalan, H Sandick,”A Framework for QoS-based routing in the Internet”, RFC 2386, Aug 1998 37 pages, http://www.ietf.org/rfc/rfc2386.txt
[DQRMT], Venkatesh Sarangan, Donna Ghosh, raj Acharya,”Distribuited QoSRouting for multimedia Traffic”,

Department of Computer Science and Engineering, State University of New York at Buffalo
[IARSQD]) A. R. Mohamed Tasir N Linge, “Intelligent Active routing for Supporting QoS demands”, Centre for Networking and Telecommunication reserch, University of Stanford.
[QOSF2] “Quality of Service – Glossary of Terms”, May 1999, 26 pages, http://www.qosforum.com/white-papers/qos-glossary-v4.pdf
[WC96] Wei Sun, “QoS/Policy/Constraint Based routing”, Ohio State University
[CHEN99] Shigang Chen, “Routing Support for Providing Guaranteed End-toEnd Quality of Service”, Ph.D.thesis, UIUC, May 1999.
[BASW] William Su, Merio Gerla, “Bandwidth Allocation strategies for wireless ATM netowrks using Predective Reservation”, Department of Computer Science, University of California.
[AF-PNNI] The ATM Forum, “Private Network-Network Interface Specification Version 1.1 (PNNI 1.1)”, Technical Cometee, April 2002.
[DQSAHN] Shiang Chen, Klara Nahrstedt, “Distribuited Quality-of-Service Rounting in Ad-Hoc Networks”,Journal on selected areas of communication, vol. 17, No. 8, August 1999
[ARDMS] Saurav Chatterjee, “Allocation and Rounting for Distribuited Multimedia Systems”, SRI International
[QSMMC] Andrew T. Campbell,” QoS aware Middleware for Mobile Multimedia Communciations”, Multimedia Tools and Applications 7, 67-82 (1998)
[SOMA] Bellavista P., Corradi A., Stefanelli C., “Mobile Agent Middleware to Support Mobile Computing”, IEEE Computer, Vol. 34, No. 3, pages 73-81, March 2001
[GFRMI] Foschini, L., Tesi di Laurea 2003
[VisPat] Pattern Visitor, si veda http://patterndiget.com/patterns/Visitor.html
[JMFPG] Java Media Framework, Programmers Guide:

http://java.sun.com/products/java-media/jmf/2.1.1/specdownload.html
[JMFH] Java Media Framework home page:
http://java.sun.com/products/javamedia/jmf/