INDICE

INTRODUZIONE **1**

1 CACHING DISTRIBUITO PER MATERIALE MULTIMEDIALE **3**

1.1 DIVERSE TIPOLOGIE DI CACHING 3
1.2 MODELLO CLIENT - SERVER E RELATIVI SVILUPPI 6
1.3 INTRODUZIONE AL WEB CACHING 8
1.4 MULTIMEDIA PROXY CACHING 10
1.5 PARAMETRI DI VALUTAZIONE DI UNA CACHE 12
1.6 POLITICHE DI GESTIONE 14
1.7 CONCLUSIONI 15

2 REMOVAL POLICIES: ANALISI E VALUTAZIONI **17**

2.1 CLASSIFICAZIONE DELLE POLITICHE 17
2.2 STRATEGIE RECENCY 19
2.3 STRATEGIE FREQUENCY 22
2.4 STRATEGIE RECENCY/FREQUENCY 24
2.5 STRATEGIE BASATE SU FUNZIONI 25
2.6 STRATEGIE RANDOM 29
2.7 CACHING DI OGGETTI MULTIMEDIALI 31
2.8 CONCLUSIONI 32

3 GLI AGENTI MOBILI E LE TECNOLOGIE UTILIZZATE: SOMA, MUM E JMF **33**

3.1 LA MOBILITÀ DI CODICE E IL PARADIGMA AD AGENTI MOBILI 33
3.2 SOMA 34
3.2.1 Caratteristiche principali dell'ambiente SOMA *36*
3.3 MUM 38
3.3.1 Fruizione del materiale multimediale *39*
3.3.2 Gestione dei contenuti multimediali *40*
3.3.3 Gestione dello streaming *41*
3.4 JAVA MEDIA FRAMEWORK (JMF) 42
3.4.1 Il componente Player *44*
3.4.2 Il componente Processor *46*
3.4.3 Il componente DataSource *48*
3.4.4 Il componente DataSink *48*
3.5 CONCLUSIONI 48

4 ANALISI E PROGETTAZIONE DEL SISTEMA DI CACHING PER MATERIALE MULTIMEDIALE **50**

4.1 INQUADRAMENTO GENERALE DEL PROBLEMA 50
4.2 ANALISI DEI REQUISITI 51
4.2.1 Politiche di gestione *52*
4.3 PROBLEMATICHE AFFRONTATE 52
4.3.1 I formati video *53*
4.3.2 Organizzazione interna della cache *53*
4.3.3 Memorizzazione dei video *54*
4.3.4 Politiche di rimpiazzamento *55*
4.3.5 Opzioni di personalizzazione *56*
4.4 PROGETTAZIONE 56
4.4.1 La cache *58*

4.4.2 Il gestore del sistema *58*
4.4.3 La configurazione della cache *60*
4.4.4 Le politiche di gestione *61*
4.4.5 La gestione delle liste *67*
4.5 Conclusioni *68*

5 IMPLEMENTAZIONE E TESTING DEL SISTEMA DI CACHING 69

5.1 IMPLEMENTAZIONE DEL SISTEMA DI CACHING 69
5.1.1 Gestione dei contenuti multimediali *70*
5.1.2 Le politiche di gestione *72*
5.1.3 La gestione delle liste *76*
5.1.4 La configurazione del sistema *76*
5.2 TESTING 77
5.2.1 LRU vs LRU-Min *78*
*5.2.2 LRU vs LRU** *80*
5.3 CONCLUSIONI 80

CONCLUSIONI 81

BIBLIOGRAFIA 83

Introduzione

Il mondo dell'Information Technology è in continuo e rapido sviluppo. Oltre ai processori, questo sviluppo ha coinvolto, e coinvolge, tutti gli altri componenti ad esso strettamente legati. Il settore che più ha risentito di queste rapide evoluzioni è quello relativo alle infrastrutture di rete. La motivazione principale è che si è cercato di utilizzare infrastrutture di collegamento già esistenti e che permettessero di raggiungere un elevato numero di utenti. Lo studio di nuovi mezzi di collegamento è sempre progredito, ma il problema sostanziale è rimasto quello dell'impossibilità di estendere in maniera globale le nuove tipologie di cablaggio. Tutto questo parallelamente al costante aumento di prestazioni che viene richiesto a tali infrastrutture. Contribuisce, in maniera fondamentale, a questo incremento di prestazioni richieste, la diffusione di materiale di tipo multimediale. Questa categoria comprende, ad esempio, materiale audio/video di tipo digitale. È possibile pensare a servizi quali la televisione digitale interattiva o le videoconferenze. Questi servizi necessitano di una gestione in tempo reale e per questa motivazione risulta naturale pensare i sistemi multimediali come sistemi di tipo real-time. La diffusione di questa tipologia di dati richiede che, il sistema utilizzato per la loro gestione, garantisca una certa qualità di servizio. Questa garanzia è necessaria per assicurare il rispetto dei tempi a cui i dati sono vincolati. Diverse sono le risorse coinvolte nella gestione della qualità di servizio. Una di queste è la larghezza di banda. Proprietà importante relativa al cammino che le informazioni devono compiere per passare dal server di origine al client.

Per cercare di risolvere i problemi introdotti dalle limitazioni relative alla larghezza di banda, vi sono due possibili approcci: la replicazione e il proxy caching. Il primo approccio è quello che riguarda l'introduzione di cosiddetti mirror server. Si tratta, in pratica, di replicare le informazioni contenute nel server di origine, in altri server sparsi all'interno della rete. Questo approccio di tipo statico risulta essere molto dispendioso, specialmente nel caso in cui il contenuto del server originario cambi frequentemente. Il secondo approccio, il proxy caching, risulta migliore, in

quanto adattabile alle caratteristiche della rete e delle informazioni che viaggiano all'interno di essa. Questo secondo approccio è quello che verrà affrontato all'interno di questa tesi. Vengono introdotte delle entità denominate proxy, intermedie fra client e server, attraverso le quali passano le informazioni transitanti tra i due end point della comunicazione. Queste informazioni possono essere intercettate e memorizzate nella memoria cache del proxy per migliorare le prestazioni di accessi successivi.

Coinvolgendo la tipologia di dati multimediali precedentemente introdotta, verranno trattate principalmente le problematiche relative al *Multimedia Proxy Caching*. Un ruolo importante assumeranno alcuni parametri tipici dei contenuti multimediali, quali, ad esempio, l'occupazione di memoria e la qualità. L'obbiettivo finale di questo documento è la realizzazione di un sistema di caching.

I primi capitoli riguarderanno le argomentazioni generali coinvolte nella realizzazione del progetto. In particolare, il primo capitolo introdurrà le diverse tipologie di caching realizzabili, con particolare attenzione al multimedia proxy caching. Vengono presi in considerazione i parametri di valutazione relativi ad una memoria cache e le politiche coinvolte nella sua gestione. La categoria di politiche che principalmente sarà presa in considerazione è quella delle politiche di rimozione, affrontata all'interno del secondo capitolo. Nel terzo capitolo ci si occuperà dell'introduzione delle tecnologie utilizzate o in qualche modo coinvolte: SOMA, MUM e il Java Media Framework. Risulta necessaria, per queste tecnologie, una breve introduzione, individuandone gli aspetti più significativi per il lavoro di tesi. All'interno del quarto capitolo ci si occuperà delle fasi di analisi e progettazione riguardanti il sistema di caching da realizzare, evidenziando le scelte effettuate. Il quinto ed ultimo capitolo esporrà la fase di implementazione del progetto ed alcune considerazioni relative ad una successiva fase di testing.

CAPITOLO 1

1 Caching distribuito per materiale multimediale

Questo primo capitolo si pone lo scopo di introdurre ed illustrare le problematiche ed alcune delle possibili scelte implementative riguardanti il *caching* in ambiente distribuito. Considerando i motivi per cui è auspicabile intraprendere questa strada ed i vantaggi che questo comporta, ponendo particolare attenzione alle tipologie di dati che più interessano questo progetto di tesi: i dati multimediali.

La *cache* è solitamente intesa come una memoria di non elevate dimensioni, utilizzata per la memorizzazione di informazioni, che si ritiene possano essere richieste in istanti di tempo abbastanza vicini, in modo da renderne più rapido il reperimento. L'idea della realizzazione di questa tipologia di memorie nacque inizialmente per colmare il divario, in termini di velocità, tra le CPU e la memoria principale. Questi concetti sono oggi applicati, con opportune modifiche e particolarità, a diversi ambiti informatici, che andiamo a descrivere brevemente nel prossimo paragrafo.

1.1 Diverse tipologie di caching

Il caching, con la sua idea fondamentale di memorizzazione, si è sviluppato in diversi settori, seguendo lo sviluppo della tecnologia. Lo scopo fondamentale di questa operazione è la memorizzazione di informazioni atta a consentirne un più rapido reperimento, in modo da ottenere un incremento delle prestazioni globali del sistema preso in considerazione. Solitamente queste memorie sono gestite tenendo presenti due concetti: la località spaziale e la località temporale. Qui ci poniamo l'obbiettivo di analizzare rapidamente i diversi ambiti in cui vengono normalmente effettuate operazioni di caching.

Sistemi monoprocessore

Qui la cache è solitamente una memoria veloce e di dimensioni abbastanza ridotte, anche se può essere organizzata, come avviene nei microprocessori attualmente prodotti, su più livelli. L'idea di fondo è la stessa delle memorie principali, ma grazie alle dimensioni limitate riescono ad essere gestite in maniera più veloce. La località spaziale è intesa come il fatto che i dati probabilmente richiesti saranno quelli immediatamente successivi, in termini di indirizzo, a quelli utilizzati al momento e che vengono quindi precaricati nella memoria. La località temporale afferma che l'ultimo dato usato sarà probabilmente il prossimo richiesto.

Sistemi multiprocessore a memoria condivisa

Questa architettura prevede di avere più CPU che condividono una stessa memoria, tramite l'utilizzo di un comune bus di accesso. Ogni CPU, in questo caso, sarà dotata di una propria memoria di cache. Oltre al rispetto dei concetti di località citati per i sistemi monoprocessore, si deve tenere presente anche il rilevante problema della consistenza dei dati contenuti nelle singole cache. È necessario gestire questo tipo di sistema in modo che venga garantito l'utilizzo dell'istanza di dati più aggiornata. Effettuando il caching di una grande quantità di dati, le più recenti versioni di questi sistemi, sono in grado di ridurre notevolmente i tempi di accesso alla memoria e di ridurre la banda necessaria al bus che connette le memorie condivise, dal momento che la maggior parte delle richieste viene soddisfatta dal sistema di cache.

File system distribuiti

I file system distribuiti devono essere in grado di gestire operazioni concorrenti su informazioni presenti su più nodi facenti parte della stessa infrastruttura come se queste avvenissero su file locali. Anche se i tempi di accesso a risorse locali non potranno mai essere equiparati ai tempi di accesso a risorse remote, l'operazione di caching aiuta in maniera considerevole nel ridurre questi ultimi. L'obbiettivo delle operazioni di caching è quello di ridurre il numero di trasmissioni fra le parti coinvolte e di limitare il trasferimento di grandi quantità di dati. Posso avere, nel caso

di modello client-server, un caching sul server, che diminuisce gli accessi al disco, guadagnando velocità, ed un caching sul client che riduce il numero di accessi al server, guadagnando sempre in termini di tempi di risposta.

Una frequente organizzazione di questo modello prevede che la cache venga mantenuta sul client mediante l'utilizzo di caching su disco, che consente un miglioramento delle prestazioni nel caso di accessi successivi al sistema. La cache su disco ha il grande vantaggio di poter essere di dimensioni notevolmente superiori rispetto a quella posta in memoria principale.

World Wide Web

L'espansione capillare della "rete delle reti" ha portato a dover affrontare problemi di trasporto relativi alla mole di dati circolanti sulla rete. La più naturale soluzione per far fronte a questo problema sarebbe quella di intervenire a livello hardware nella rete, aumentando, ad esempio, la larghezza di banda. Questa strada risulta però impraticabile sia dal punto di vista tecnologico che dal punto di vista economico. L'attenzione viene rivolta allora al miglioramento dei protocolli di comunicazione e soprattutto all'utilizzo delle tecniche di caching. Così facendo si riesce a ridurre il traffico nella rete e ad aumentare la velocità di risposta nei confronti degli utenti.

Esistono inoltre, altri ambienti distribuiti oltre a quelli citati, ognuno dei quali può presentare esigenze e caratteristiche diverse nella gestione di un proprio sistema di caching. Queste caratteristiche possono variare a seconda del tipo di informazioni che vengono trasmesse, e quindi memorizzate, nel sistema, a seconda del traffico presente in media sull'infrastruttura di interconnessione oppure a seconda della frequenza delle richieste.

1.2 Modello client - server e relativi sviluppi

In una parte del paragrafo precedente si è accennato ad un modello di interconnessione fra unità di elaborazione denominato *client - server*. Andiamo a vederne brevemente le caratteristiche principali, che risultano utili per introdurre concetti utilizzati nel proseguio della trattazione.

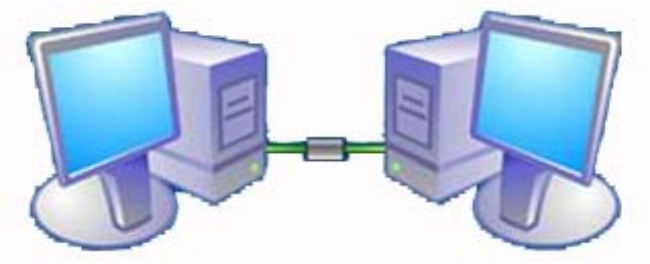
Client  Server

Figura 1.1: classico modello Client-Server.

Questo è probabilmente il modello di comunicazione divenuto più famoso nell'ambito delle reti di computer, grazie anche alla capillare diffusione di Internet, realtà ormai divenuta parte integrante della nostra vita quotidiana. Consiste principalmente in due parti logiche che assumono, uno l'identità di *server* e l'altro quella di *client*. Il componente chiamato server ha solitamente il ruolo di fornitore di servizi che attende, in maniera passiva, l'arrivo di una richiesta da parte di uno o più client, la serve e ritorna al richiedente il risultato della sua elaborazione. Il componente client è invece colui che richiede un determinato servizio all'entità server a cui si collega. Ciò presuppone la conoscenza da parte del client dell'indirizzo del server, mentre la conoscenza da parte del server dell'indirizzo del client non risulta essere necessaria a priori.

Lo stato interno del server risulta nascosto ai vari client che vi si collegano, i quali possono utilizzare il server solamente attraverso i servizi da questo offerti. Il vantaggio principale di questo modello è la semplicità di realizzazione, anche se si può arrivare ad avere sovraccarichi del server e del canale di comunicazione. Questa classica architettura è anche individuata dalla denominazione **client-server 2-tier**, dove il client

interagisce direttamente con il server, senza passaggi intermedi. Una sua caratteristica negativa risulta essere però la bassa scalabilità, in quanto al crescere del numero di utenti collegati, decrescono le prestazioni del server.

Un modello che risulta essere più interessante ed applicabile vede l'inserimento di un'ulteriore unità. Un server può anche agire come client nei confronti di un altro server, nel caso in cui, ad esempio, deleghi a questo l'esecuzione di un sottoservizio. Allo stesso modo un client può fungere da server nei confronti di un altro client. Il caso in cui fra il client ed il server ci sia la presenza di un componente intermedio, ci porta a definire una architettura **client-server 3-tier**. Questo componente intermedio può svolgere vari ruoli, come ad esempio quello di filtro o di bilanciamento del carico di lavoro su più server per ovviare al problema di diminuzione delle prestazioni. Tale unità intermedia può essere conosciuta anche con il nome di **proxy server**, entità che si comporta da server nei confronti dei client, e da client nei confronti dei server. Se le unità intermedie sono più di una si parla in maniera generica di modelli **n-tier**.

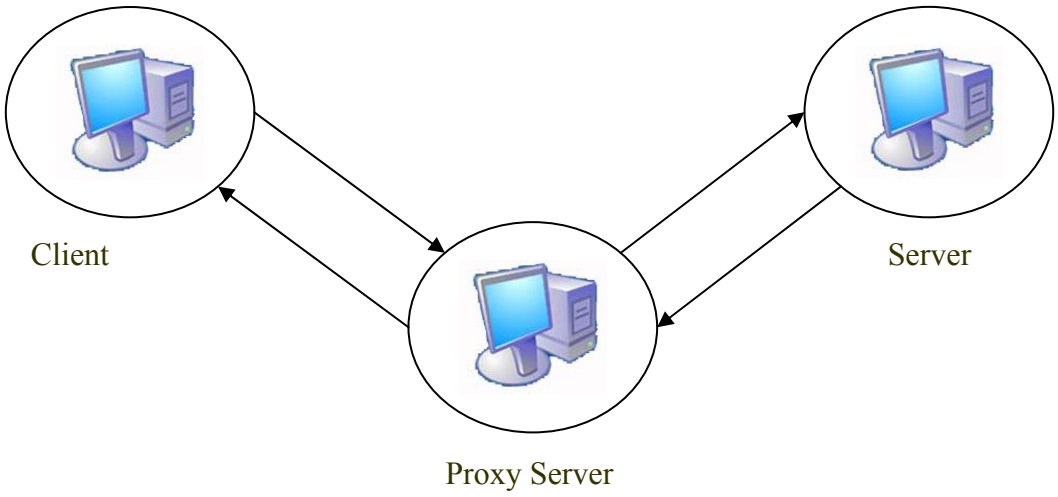


Figura 1.2: Architettura Client-Server 3-tier.

In queste architetture il client può comportarsi principalmente in due modi: può effettuare la richiesta di un servizio e porsi in attesa della risposta oppure effettuare la richiesta e poi svolgere altri compiti. Questo

ultimo modo di operare identifica due schemi di funzionamento caratterizzati da diverse responsabilità delle entità coinvolte:

- modello *pull*, in cui è il client che si carica della responsabilità di ottenere le informazioni richieste, mediante interrogazioni cicliche;
- modello *push*, in cui è il server ad essere incaricato di inviare l'informazione se e quando è disponibile.

Il paradigma client-server sta alla base di successive tecniche di comunicazione e fruizione di servizi da terminale remoto, quali le chiamate a procedure remote (RPC) e le invocazioni di metodi remoti (RMI).

1.3 Introduzione al web caching

In generale ogni richiesta che viene effettuata ad una cache, indipendentemente dalla sua tipologia specifica, può portare a due tipi di risultato denominati *cache hit* o *cache miss*. Quando il documento, o più in generale l'oggetto, richiesto è presente all'interno della cache e può quindi da questa essere reperito si verifica un cache hit. Al contrario, se l'oggetto non è presente si verifica un cache miss, che comporterà l'inoltro della richiesta non soddisfatta verso un'altra entità.

La grande crescita della rete Internet ha portato ad affrontare problemi di sovraccarico della rete e problemi di eccessivo ritardo nel reperimento di informazioni. Per ovviare a questi inconvenienti sono state sviluppate evolute tecniche di caching.

Il web può essere visto come un insieme distribuito di server e client, connessi fra loro mediante una infrastruttura di rete. In questo insieme posso individuare tre livelli di caching: un caching effettuato sui client, uno sui server ed uno realizzato all'interno della rete stessa.

Client cache

Per quanto riguarda questa tipologia di caching, si può dire che essenzialmente è effettuata all'interno del web browser utilizzato dall'entità client per il reperimento di informazioni sulla rete. La sua politica di gestione si basa essenzialmente sul fatto che viene fatta una copia degli

oggetti caratterizzati da un elevato numero di accessi. Così, ad esempio, vengono memorizzati tutti i vari componenti (immagini, testo, …) contenuti in una determinata pagina web, puntando sul fatto che un determinato utente tende a visionare gli stessi documenti con una certa frequenza. Anche questa operazione deve tenere presente la consistenza delle informazioni contenute nella cache, in modo che, se risultano obsolete, venga previsto l'inoltro della richiesta attraverso l'infrastruttura di rete, per ottenere i documenti aggiornati.

Server cache

Questa tipologia di soluzione può essere implementata effettuando il caching del file system del server ottenendo così una riduzione del numero di accessi al disco (che introducono ritardo) sul lato server. Verranno mantenute in memoria principale le informazioni maggiormente richieste, in modo da avere tempi di risposta più brevi. Anche se a prima vista questa potrebbe sembrare una valida soluzione, per come è realizzato e per come opera un file system, non risulta assolutamente sufficiente per sopportare il traffico presente sul web. È quindi necessario realizzare una cache autonoma che non si appoggi sul file system, ma che sia gestita a più alto livello, in modo da consentire al server di gestire i documenti in essa contenuti senza coinvolgere il proprio file system.

In sostanza, il server caching è principalmente utilizzato per ridurre il carico di lavoro sul server e migliorare tempo di risposta e throughput.

Cache di rete

Oltre al caching effettuato sui nodi estremi della linea di comunicazione, si sta sempre più diffondendo un caching all'interno della rete stessa, denominato generalmente *proxy caching*. Questa soluzione si pone come obbiettivo principale la riduzione dei tempi di risposta relativi alle interrogazioni effettuate dai client connessi alla medesima cache di rete. Solitamente vengono utilizzate cache di elevate dimensioni, in modo da poter servire in maniera efficace un elevato numero di utenti. In sostanza nella proxy-cache verranno memorizzati,seguendo opportune politiche, i documenti richiesti in precedenza dai client appartenenti ad una

determinata area di riferimento. In questo modo una successiva richiesta di tali informazioni sarà caratterizzata da tempi di attesa decisamente inferiori.

Praticamente la cache del proxy server consente di aumentare notevolmente la velocità di accesso al web, di risparmiare in termini di larghezza di banda, in quanto diminuiscono le richieste effettivamente inoltrate sul web, e di ridurre il carico di lavoro sul lato server.

1.4 Multimedia Proxy Caching

Entriamo ora in un ambito più specifico, ed in particolare ad occuparci del proxy caching relativo a materiale di tipo multimediale. Quando si parla di materiale multimediale è necessario introdurre il concetto di *flusso* (stream), intendendo un'astrazione introdotta per indicare il recapito ed il rendering di un singolo oggetto multimediale.

Un oggetto multimediale può essere inteso come un particolare tipo di dato (ad esempio: video o audio) che annovera tra le sue caratteristiche principali il fatto di variare con continuità e di essere soggetto a vincoli temporali.

La qualità di trasmissione di uno stream multimediale in un'architettura end-to-end di tipo client-server risulta essere limitata da alcuni cosiddetti colli di bottiglia. Questi possono essere individuati dalla larghezza di banda, dalle prestazioni della CPU e dalla quantità di memoria disponibile. Delle tre problematiche appena indicate quella che pone vincoli più stringenti è sicuramente la larghezza di banda della connessione tra server e client. La larghezza di banda identifica praticamente la portata del canale trasmissivo ed è solitamente indicata in bit/secondo (bps). Se la connessione del client al proprio punto di accesso alla rete è caratterizzata da un'ampia larghezza di banda, questo si aspetterà di ricevere stream multimediali di elevata qualità. Queste aspettative possono non essere soddisfatte, in quanto in quella parte di rete che si trova tra il punto di accesso del client ed il server su cui risiedono le informazioni, posso avere collegamenti (anche solamente uno) con larghezza di banda limitata, che mi fungono da colli di bottiglia. Chiaramente se il client possiede un

accesso alla rete avente larghezza di banda limitata, non è possibile ottenere contenuti multimediali di qualità elevata, a causa principalmente dei tempi di download necessari.

Una tecnica dinamica, per eludere le limitazioni imposte dalla larghezza di banda nella connessione tra server e client dotati di accessi a banda molto ampia, è appunto il *multimedia proxy caching*. Viene coinvolta una nuova entità, introdotta in precedenza e denominata proxy server, che risulta essere un server collocato nelle più immediate vicinanze di un più ristretto numero di client. Le dimensioni della memoria cache sul proxy sono proporzionali al numero di client che ad esso fanno riferimento. Una caratteristica dei contenuti multimediali da tenere in considerazione, nel dimensionamento della cache, è anche che la loro dimensione può variare a seconda del livello di qualità con cui sono memorizzati.

Tutto il materiale multimediale richiesto dai client viene ad essi trasmesso passando sempre attraverso il proxy, che può intercettarlo e memorizzarlo.

L'utilizzo di queste stazioni intermedie permette un miglioramento della qualità della trasmissione nei confronti di quei client che hanno a disposizione una buona quantità di banda. Il proxy può inoltre adattare la qualità degli stream multimediali alla larghezza di banda disponibile ed allo stato della connessione fra esso ed il client interessato. Questa soluzione introduce comunque vantaggi sia per i client che hanno a disposizione una banda ampia sia per quelli che usufruiscono di una larghezza di banda inferiore. Si hanno infatti miglioramenti di vario genere, come ad esempio:

- riduzione dei tempi di risposta relativi al controllo della riproduzione dei contenuti multimediali da parte dei client che usufruiscono del servizio;
- il supporto di accessi contemporanei alle risorse;
- riduzione del tempo di latenza iniziale introdotto dalla linea di trasmissione.

Tutto questo senza dimenticare la considerevole riduzione del carico di lavoro sul server e sulla stessa infrastruttura di rete.

I client inoltrano la richiesta al proxy corrispondente, il quale controlla la presenza o meno delle informazioni richieste all'interno della propria cache. In caso di cache miss la richiesta viene inoltrata o ad un'altra entità intermedia o al server di origine, che si occuperà direttamente di soddisfarla. Se consideriamo l'intervento del server, la qualità della trasmissione e del contenuto multimediale inviato sono limitate dalla larghezza di banda media della connessione tra server e client. Se si verifica un cache miss il client non trae alcun beneficio dalla presenza della proxy cache. In caso di cache hit, il proxy si comporta come un server ed inizia a soddisfare la richiesta inoltratagli dal determinato client. Per evitare problemi di congestione dei canali di trasmissione sarà cura del proxy adattare la qualità degli oggetti multimediali alla larghezza di banda disponibile.

1.5 Parametri di valutazione di una cache

Per poter avere una sorta di valutazione della validità di una memoria cache, esistono una serie di indicatori principali che possono essere analizzati. Definiamo alcune variabili necessarie:

r: il numero totale di richieste;

d_i: la dimensione in byte dell'i-esimo oggetto;

c_i: che assumerà valore 1 nel caso di cache hit e 0 nel caso di cache miss.

Presentiamo ora brevemente questi principali indicatori.

Hit rate

Rappresenta il rapporto tra il numero di richieste soddisfatte dalla cache ed il numero totale di richieste da essa ricevute:

$$W^{j} = \frac{\sum_{i=1}^{r} c_i}{r}$$

Questa semplice tecnica di valutazione risulta essere efficace nel caso in cui gli oggetti presenti all'interno della cache abbiano la medesima dimensione.

Relative hit rate

Identifica il rapporto tra

$$\frac{W^{j}}{W^{j}_{\max}}$$

dove W^{j}_{max} è l'hit rate ottenuto con una cache di dimensioni infinite.

Byte hit rate

Viene espresso come il rapporto tra la quantità di byte trasferiti dalla cache in seguito ad un hit e la quantità totale dei byte trasferiti:

$$W^{b} = \frac{\sum_{i=1}^{r} d_i c_i}{\sum_{i=1}^{r} d_i}$$

Relative byte rate

È il rapporto tra

$$\frac{W^{b}}{W^{b}_{\max}}$$

dove W^{b}_{max} è l'hit rate ottenuto con una cache di dimensioni infinite.

Latency ratio

È definito come il rapporto tra la somma dei tempi di download relativi a situazioni di cache miss e la somma dei tempi totali di download:

$$W^{t} = \frac{\sum_{i=1}^{r} lat_i (1 - c_i)}{\sum_{i=1}^{r} lat_i}$$

dove lat_i identifica il tempo di download dell'i-esimo oggetto.

L'indicatore più significativo per quanto riguarda l'ambito multimediale è forse il *byte hit rate*, in quanto viene presa in considerazione la dimensione degli oggetti presenti nella cache, che nel caso di oggetti multimediali può essere elevata. Senza trascurare naturalmente l'importanza della riduzione dei tempi di download e quindi il *latency ratio*.

1.6 Politiche di gestione

L'effettiva gestione di una memoria cache, posizionata in un proxy server, coinvolge un elevato numero di scelte progettuali necessarie, quali ad esempio la scelta degli elementi da memorizzare, di quelli da eliminare o sostituire e molte altre ancora. Per aver una visione di insieme di queste diverse politiche le presentiamo brevemente, indicandone le principali caratteristiche.

Prefetching policies

Il problema che si propongono di risolvere queste politiche è quello di prevedere le esigenze future degli utenti che usufruiscono del servizio di caching. Questo in modo da poter pre-caricare in memoria quei documenti che si pensa possano essere richiesti da un determinato client, in modo da poter avere un miglioramento delle prestazioni già relativamente ad un primo accesso a queste informazioni. Questo è possibile ad esempio tramite l'analisi statistica dei documenti più richiesti.

Routing policies

Quando l'infrastruttura di rete tende ad espandere notevolmente le sue dimensioni, aumentano in maniera considerevole anche le memorie cache dislocate all'interno dell'architettura. Il problema che si presenta è l'individuazione della cache che contiene il materiale richiesto. La funzione

di queste politiche è quella di ottimizzare questa ricerca, in modo che i tempi di risposta siano i più brevi possibili.

Placement policies

Quando l'espansione del sistema diventa molto elevata, anche l'utilizzo di cache di grandi dimensioni, può non bastare. A questo punto ripensa a strutture di cache cooperanti fra loro, la cui efficacia risulta essere proporzionale al numero di utenti e di oggetti condivisi. Esistono diverse tipologie di collaborazione tra cache, che introducono il problema sostanziale di "dove" memorizzare determinate informazioni. Ovvero, a quale livello ed in quale cache è conveniente memorizzare certi oggetti, per fare in modo che l'efficienza dell'architettura sia massimale.

Coherence policies

Come accennato in precedenza, il problema di un sistema di caching, può essere quello della consistenza delle informazioni gestite. Per questo si rischia di fornire agli utenti informazioni ormai obsolete, che non rispecchiano i contenuti attuali. Questa tipologia di politiche si occupa appunto di gestire il problema della consistenza dei contenuti delle memorie, in modo che questi risultino sempre aggiornati.

Replacement policies

L'efficacia di una singola memoria cache dipende, in maniera preponderante, dalle tecniche di rimpiazzamento dei dati in essa contenuti una volta raggiunta la saturazione della medesima. In sostanza queste politiche dovranno occuparsi di individuare quali elementi possono essere eliminati per liberare spazio e consentire la memorizzazione di nuovi contenuti. Queste politiche rappresentano un argomento fondamentale per la realizzazione di questo progetto di tesi e verranno affrontate in maniera più approfondita nel capitolo successivo.

1.7 Conclusioni

In questo primo capitolo si è cercato di illustrare gli argomenti principali coinvolti nella realizzazione di questo progetto. Sono stati

introdotti inizialmente i concetti basilari riguardanti il caching e le sue possibili diverse tipologie. Si è cercato di individuare i vantaggi fondamentali dovuti all'applicazione di tale tecnica, ma anche di riconoscere le varie problematiche sorte per ottenere soluzioni che risultino efficaci. Illustrando le varie politiche riguardanti la gestione di un sistema di cache abbiamo individuato quelle che più interessano questo lavoro, e di cui ci occuperemo nel capitolo successivo.

CAPITOLO 2

2 Removal policies: analisi e valutazioni

Come è già stato spiegato nel capitolo precedente, i vantaggi introdotti dall'utilizzo di un sistema di caching, sono principalmente:

- riduzione della larghezza di banda necessaria nei collegamenti di rete;
- riduzione dei ritardi percepiti dall'utente finale;
- riduzione della quantità di informazioni circolanti sulla rete;
- riduzione del carico di lavoro sul server originario.

Una delle principali caratteristiche che influenza le prestazioni di una memoria cache è l'utilizzo di una determinata politica di rimozione e/o sostituzione piuttosto che un'altra. Questa tecnica viene utilizzata quando lo spazio libero in cache non è sufficiente a contenere l'oggetto che a questa è destinato. Mirano sostanzialmente a prevenire o risolvere situazioni di saturazione della memoria disponibile. Si rende quindi necessaria la rimozione di uno o più oggetti, considerati ormai obsoleti, per liberare la quantità di memoria sufficiente.

Ciò che influenza le prestazioni del sistema di cache è la scelta di quali oggetti rimuovere e quali mantenere in memoria. Bisogna in particolare cercare di individuare quali documenti potrebbero essere richiesti in un immediato futuro e quali, invece, si ritiene possano essere eliminati senza provocare un deterioramento significativo delle prestazioni globali.

Possiamo pensare di suddividere, in alcune differenti categorie, tutte le varie strategie di rimpiazzamento che prendiamo in considerazione. Questa suddivisione può essere effettuata prendendo in considerazione i principali fattori che caratterizzano gli oggetti da memorizzare in cache e che possono influenzare le differenti politiche considerate.

2.1 Classificazione delle politiche

Come indicato in precedenza, le varie politiche di rimozione o rimpiazzamento che possono essere prese in considerazione, sono

classificabili all'interno di diverse categorie. Sono state fatte negli anni diverse classificazioni, ognuna delle quali mira a prendere in considerazione alcuni aspetti caratterizzanti, delle strategie di rimpiazzamento. Prima di introdurre la classificazione proposta, riteniamo utile presentare le principali proprietà degli oggetti di cui può essere necessario effettuare il caching. Proprietà che possono influenzare l'efficienza del processo di rimozione o sostituzione. Queste caratteristiche sono:

- recency: indica l'istante temporale in cui è stato richiesto l'ultima volta l'oggetto considerato;
- frequency: indica il numero di richieste di cui è stato fatto oggetto lo specifico documento;
- size: indica la dimensione dell'oggetto da memorizzare o memorizzato in cache;
- costo: indica il costo necessario, in termini di tempo, per reperire il determinato oggetto dal server d'origine;
- modification time: indica l'istante di tempo in cui è stata apportata l'ultima modifica al documento;
- expiration time: indica un tempo trascorso il quale l'oggetto preso in considerazione può essere considerato obsoleto e quindi essere sostituito.

Questi sono i principali fattori che possono essere considerati, anche se, la maggior parte delle tecniche conosciute attualmente si limitano ad utilizzare le prime quattro proprietà.

Una possibile classificazione delle politiche di rimozione può essere la seguente:

- strategie "recency": considerano principalmente l'attributo recency, abbinandolo eventualmente a costo e/o size;
- strategie "frequency": considerano principalmente l'attributo frequency, abbinandolo eventualmente a costo e/o size;
- strategie "recency/frequency": considerano entrambi gli attributi recency e frequency ed eventualmente valori fissi o variabili di costo e/o size;

- strategie basate su funzioni: questa classe di strategie è contraddistinta da funzioni che prendono in considerazione più parametri contemporaneamente, assegnando ad essi un opportuno peso;
- strategie random: sono strategie di rimozione di tipo non deterministico, che prendono, ad esempio, in considerazione la casualità oppure determinati calcoli di probabilità.

Molti degli algoritmi presentati in letteratura sono stati studiati per ottimizzare la gestione di cache contenenti oggetti di ugual dimensione, anche se alcuni di questi sono facilmente applicabili ad ambiti più generali, gestendo oggetti con caratteristiche diverse fra loro.

Nei prossimi paragrafi approfondiremo le classi di strategie che qui abbiamo introdotto, andando ad analizzare alcune delle politiche esistenti.

2.2 Strategie Recency

Questa tipologia di strategie agisce considerando il fattore recency come attributo principale. Come già accennato nel capitolo precedente, la località caratterizza l'abilità di prevedere i futuri accessi in base a quelli passati. Esistono due tipi di località: spaziale e temporale. La località temporale si riferisce alla possibilità di avere accessi ripetuti al medesimo oggetto in istanti di tempo immediatamente successivi all'ultima richiesta ricevuta. Questo implica che un oggetto richiesto recentemente abbia un buon numero di probabilità di essere richiesto nuovamente entro un breve lasso di tempo. La località spaziale implica invece che la richiesta di un oggetto aumenti in maniera considerevole la probabilità che un altro specifico oggetto venga richiesto successivamente. Questa tipologia di politiche tiene in considerazione principalmente la località temporale.

Andiamo ora ad illustrare brevemente alcuni algoritmi che ricadono all'interno di questa categoria.

LRU (Least Recently Used)

Questa politica agisce sostanzialmente rimuovendo, in caso di saturazione della memoria, l'oggetto o gli oggetti che sono stati richiesti

meno recentemente. È una delle strategie più classiche ed è basata sulle considerazioni effettuate riguardo la località temporale. Viene utilizzata in diversi ambiti applicativi, quali la gestione di database, il paging e la gestione di buffer su disco.

LRU - Threshold

Questa politica rappresenta una modifica di quella appena presentata, dove viene effettuata una selezione sui documenti da memorizzare oppure no. In particolare se un oggetto *i* ha dimensione (s_i) maggiore di una soglia prestabilita (threshold), questo oggetto, non viene memorizzato nella cache. Tutto questo a prescindere dagli altri parametri. Per il resto opera esattamente come l'algoritmo LRU.

SIZE

La linea adottata da questa strategia prevede la rimozione dell'oggetto di dimensione maggiore, consentendo di liberare un maggior quantitativo di memoria. Nel caso di presenza di oggetti di uguali dimensioni, quello o quelli da rimuovere vengono determinati grazie all'utilizzo della politica LRU.

LRU - Min

Questa politica risulta essere una variante della LRU, che cerca di minimizzare il numero dei documenti rimossi. Introduce due parametri, L_0 e T, dove il primo indica una lista ed il secondo un valore di soglia. Viene impostato il valore di T ad S, che rappresenta la dimensione del documento da memorizzare. Nella lista L_0 vengono inseriti tutti gli elementi aventi dimensioni maggiori o uguali a T, tenendo presente che la lista potrebbe essere vuota. Dopodiché viene applicato l'algoritmo LRU sulla lista creata, rimuovendo oggetti fino a che viene liberato spazio almeno pari a T, oppure fino a che la lista non è vuota. Se a questo punto non ho ottenuto sufficiente spazio libero, imposto T al valore $T/2$ e ripeto i passi appena descritti.

LRU - LSC

Questa strategia utilizza una normale lista LRU per determinare un parametro che identifica l'*attività* degli oggetti mantenuti in cache. Durante il processo di rimozione, gli oggetti contraddistinti da una minor attività vengono posizionati in una seconda lista, la cui dimensione totale deve essere inferiore al valore di ΘB. B rappresenta la dimensione totale della cache e Θ è un parametro ($0 < \Theta < 1$) che rappresenta quale frazione della memoria totale debba essere inserita nella nuova lista. Valore che può essere impostato dinamicamente o staticamente. La nuova lista è ordinata secondo il valore $spc_i = c_i/s_i \bullet t_i$ che rappresenta l'attività degli oggetti, dove c_i è il costo, s_i la dimensione e t_i l'istante di tempo dell'ultima richiesta dell'oggetto i. Vengono rimossi oggetti dalla nuova lista, iniziando da quelli caratterizzati da spc_i minore, fino ad ottenere lo spazio libero necessario.

Partitioned Caching

Questa politica abbastanza generale prevede la suddivisione degli oggetti in tre principali categorie, in base alla loro dimensione, etichettate come small, medium e large. Ogni classe ha un proprio spazio di memoria ed è gestita in maniera completamente indipendente dalle altre due seguendo la strategia LRU. Considerando S_C la dimensione totale della cache e S_{c1}, S_{c2} e S_{c3} le dimensioni delle tre partizioni (small=1, medium=2, large=3) ho che $S_C = S_{c1} + S_{c2} + S_{c3}$, con $S_{c1} < S_{c2} < S_{c3}$. Bisogna da tenere presente che queste considerazioni possono essere ritenute abbastanza generali, in quanto risulta chiaro, come sia possibile utilizzare questa suddivisione in categorie abbinandola a politiche diverse dalla LRU.

Le strategie recency presentate sono accomunate fra loro da diverse caratteristiche che permettono di individuare alcuni vantaggi ed alcuni svantaggi di questa famiglia. Tra i vantaggi possiamo individuare il fatto che considerino la località temporale come fattore di principale influenza e che riescano ad adattarsi in maniera dinamica ai cambiamenti del sistema. Inoltre risultano generalmente abbastanza semplici da implementare e veloci nello svolgimento dei compiti loro assegnati. Inserimenti e

cancellazioni dalla lista, infatti, richiedono solitamente operazioni non molto complesse.

Mentre tra gli svantaggi possiamo considerare il fatto, che le semplici implementazioni di LRU, non diano la giusta importanza alla dimensione degli oggetti da mantenere in memoria. Non viene realizzato un giusto bilanciamento tra il parametro recency e la dimensione dell'oggetto, fatta eccezione, naturalmente, per quanto riguarda la strategia SIZE. Un altro svantaggio individuato risulta essere la mancanza di considerazione nei confronti dell'informazione sulla frequenza degli accessi ad un determinato oggetto.

2.3 Strategie Frequency

Queste politiche di gestione di una memoria cache utilizzano il parametro frequency come fattore decisionale primario. Sono principalmente estensioni della più classica strategia LFU (Least Frequently Used), che verrà indicata in seguito, e si basano sul fatto che un oggetto può essere più o meno richiesto di un altro. Possiamo avere due modi di procedere:

- *Perfect LFU*: in questa modalità vengono conteggiate tutte le richieste relative ad un determinato oggetto e questa informazione è mantenuta memorizzata anche se l'oggetto in questione viene eliminato dalla memoria cache. Il numero delle richieste per tutti quegli oggetti che sono stati memorizzati in cache almeno una volta è mantenuto in memoria anche quando questi oggetti sono stati rimossi dalla cache;
- *In-cache LFU*: in questa modalità il numero di richieste relative a ciascun oggetto viene aggiornato e mantenuto in memoria fino a quando l'oggetto è in cache. Un nuovo oggetto che viene memorizzato nella cache avrà un contatore azzerato. Questo non rivela traccia della sua storia passata, ovvero non è possibile sapere se è già stato memorizzato nella cache in precedenza. In questo modo ho il vantaggio di ridurre notevolmente le

informazioni che devo mantenere in memoria, molte delle quali sarebbero potute essere pressoché inutili.

Per le motivazioni appena evidenziate assumeremo nel seguito della trattazione l'utilizzo esclusivamente della modalità *In-cache LFU*. Illustriamo ora alcune strategie rientranti in questa categoria.

LFU (Least Frequently Used)

Questa è la politica più semplice della categoria ed implementa le scelte tipiche che la contraddistinguono. Per ogni oggetto contenuto all'interno della cache è presente un relativo contatore che verrà incrementato ad ogni richiesta del medesimo. In caso di saturazione della memoria cache verrà rimosso l'oggetto o gli oggetti che caratterizzati dal minor numero di richieste, ovvero da una minor frequenza di accesso.

LFU - Aging

Utilizzando la politica precedente è possibile andare incontro a situazioni di logico malfunzionamento. Se un oggetto contenuto nella cache viene, per un certo periodo di tempo, richiesto molte volte, potrebbe rimanere in memoria anche quando non è più richiesto da molto. Questo è dovuto all'alto valore assunto dal suo contatore degli accessi, ottenuto nel momento di maggior popolarità. Per rimediare a questo inconveniente e limitare l'effetto di una prolungata permanenza nel tempo, eventualmente non giustificata, in cache, si introduce un valore di soglia. Se il valore medio dei contatori delle richieste degli oggetti in cache supera questo valore di soglia, il valore di tutti i contatori viene diviso per due. Inoltre questa strategia prevede un valore massimo per questi contatori.

SwLFU (Server Weighted LFU)

Questa strategia prevede l'utilizzo di contatori di richieste "pesati". Il peso w_i dell'oggetto i rappresenta una stima quanto, il server originario contenente l'oggetto, promuova il caching del medesimo. È quindi il server che influenza direttamente questa strategia. In caso di rimozione viene applicato l'algoritmo LRU agli oggetti aventi il medesimo valore di richieste del contatore pesato.

Anche questa categoria di politiche è caratterizzata da alcuni vantaggi ed alcuni svantaggi. Il principale vantaggio è rappresentato dal fatto che agiscono considerando la popolarità degli oggetti. Questo però comporta un aumento della complessità realizzativa. Possono inoltre verificarsi situazioni non corrette se, come indicato in precedenza, non viene considerato anche il tempo trascorso dall'ultima richiesta rivolta all'oggetto. Può capitare anche di avere più oggetti contraddistinti da un uguale numero di richieste; in tal caso bisogna introdurre un fattore supplementare che intervenga nella politica.

2.4 Strategie Recency/Frequency

Come evidenziato dal nome della categoria, andiamo qui ad illustrare alcuni algoritmi che considerano entrambi i parametri principalmente utilizzati fino ad ora: recency e frequency.

SLRU (Segmented LRU)

Questa strategia prevede la suddivisione della cache in due segmenti: uno non-protetto ed uno protetto riservato agli oggetti caratterizzati dal maggior numero di richieste. Quando un oggetto viene memorizzato nella cache una prima volta, viene inserito nel segmento non-protetto. Quando si verifica un cache hit di un oggetto presente nel segmento non-protetto, questo viene trasferito nel segmento protetto. Entrambi i segmenti sono gestiti con politica LRU, ma solo gli oggetti di quello non-protetto possono essere rimossi dalla cache. Quelli del segmento protetto possono solamente essere trasferiti nell'altro segmento. Per il corretto funzionamento di questa strategia è necessario introdurre un parametro che quantifichi la percentuale di memoria cache da allocare nel segmento protetto.

LRU*

Questa politica di gestione prevede l'utilizzo di una lista di oggetti con strategia LRU. Ogni oggetto ha un proprio contatore di richieste. Quando per un oggetto memorizzato si verifica un cache hit, questo viene spostato in testa alla lista e viene incrementato di uno il suo contatore. Ogni

volta che si rende necessario un rimpiazzamento, viene considerato il contatore dell'oggetto che non è stato richiesto da maggior tempo, posizionato quindi in coda alla lista. Se il valore del suo contatore è zero, l'oggetto viene cancellato, mentre se è diverso da zero, viene decrementato di una unità e l'oggetto viene posto in testa alla lista. Si procede con la rimozione di oggetti, ed eventualmente con la modifica di contatori, fino a che non si libera sufficiente spazio di memoria.

HYPER-G

Questa strategia prevede lo sfruttamento combinato di tre politiche principali: LRU, LFU e SIZE. Quando si presenta la necessità di effettuare una cancellazione e sostituzione nella memoria cache, viene inizialmente utilizzata la politica LFU. Se questo criterio restituisce più di un oggetto con le medesime caratteristiche, viene applicata la politica LRU a questo insieme. Se ancora non si riesce ad individuare un unico oggetto elimino dalla memoria cache, utilizzando il criterio di scelta SIZE, l'oggetto di dimensione maggiore presente in quest'ultimo insieme.

Il principale vantaggio di queste strategie è quello rappresentato dal fatto di utilizzare entrambi i parametri recency e frequency e quindi di risolvere le problematiche evidenziate nei precedenti paragrafi. Questo comporta un aumento della complessità implementativa nella maggioranza di queste politiche. Solamente la strategia LRU* cerca di utilizzare una semplice implementazione di LRU, combinandola con la considerazione riguardo la frequenza di accesso. Anche se non viene considerata la dimensione dell'oggetto.

2.5 Strategie basate su funzioni

Queste strategie prevedono l'utilizzo di funzioni che permettono di assegnare ad ogni oggetto un determinato valore. Il parametro ricavato permette di scegliere quale o quali oggetti possono essere ragionevolmente rimossi. Sono solitamente quelli che hanno il parametro calcolato di valore minore. Ne presentiamo ora alcune.

GD-Size (Greedy Dual - Size)

Questa strategia si propone di calcolare per ogni oggetto un valore caratteristico H_i. Sia una nuova richiesta per un oggetto non ancora contenuto in cache, che una situazione di cache hit richiedono il ricalcolo del parametro appena citato. Questo valore è calcolato utilizzando la formula:

$$H_i = \frac{c_i}{s_i} + L;$$

dove c_i rappresenta il costo dell'oggetto i, s_i la sua dimensione e L è un valore che varia con l'aumentare del tempo di permanenza in cache dall'oggetto i ed è inizialmente posto uguale a zero. Questa strategia prevede la scelta, per la rimozione, dell'oggetto avente il valore H_i più basso. Questo valore viene inoltre assegnato al parametro L.

GDSF

Come si può facilmente intuire dal nome, questa strategia deriva direttamente da quella appena presentata. In particolare viene presa in considerazione anche la frequenza di acceso di un oggetto, ovvero il numero di richieste passate ad esso relative. Al momento del calcolo o del ricalcolo del valore H_i verrà utilizzata la seguente formula:

$$H_i = f_i \frac{c_i}{s_i} + L;$$

dove possiamo notare l'introduzione di un nuovo parametro f_i che rappresenta il numero di accessi passati relativi all'oggetto i. Esiste anche una diversa formulazione di questa strategia, la seguente:

$$H_i = \frac{f_i^{\alpha}}{s_i^{\beta}} + L;$$

dove il valore di c_i viene considerato pari ad uno. I due parametri α e β rappresentano due parametri pesati, che vogliono indicare l'importanza che si vuol dare ai fattori utilizzati.

GD*

Questa strategia si pone l'obbiettivo di assegnare ad ogni oggetto un parametro H_i che viene calcolato mediante la seguente formula:

$$H_i = \left(\frac{(f_i * c_i)}{s_i} \right)^{\frac{1}{\beta}} + L ;$$

dove i parametri utilizzati assumono i seguenti significati riguardo l'oggetto i: f_i è il numero di accessi passati, c_i il costo, s_i la dimensione, L è un parametro relativo al tempo di permanenza ed infine β è un parametro pesato, il cui valore, può essere variato in maniera dinamica. La gestione del parametro L è naturalmente la medesima utilizzata nella strategia originaria GD-Size.

LRV

Questa strategia di rimozione sceglie di rimuovere dall'insieme degli oggetti contenuti nella memoria cache, quello caratterizzato dal più basso valore del particolare parametro assegnato a ciascuno di essi. Questo parametro deriva dal calcolo di una funzione che prende in considerazione la probabilità che un oggetto venga nuovamente richiesto. Il suo valore P_r viene calcolato nel modo seguente:

$$P_r(f_i, T_i, s_i) = \begin{cases} P(1, s_i)\left(1 - \tilde{D}(T_i)\right), & se \quad f_i = 1 \\ P(f_i)\left(1 - \tilde{D}(T_i)\right) & altrimenti \end{cases} .$$

$P(f_i)$ rappresenta la probabilità che l'oggetto i venga richiesto f_i+1 volte, dato che è già stato richiesto f_i volte. Per gli oggetti che presentano al loro attivo una sola richiesta, questa probabilità dipende in maniera diretta

dalla loro dimensione s_i ed è data dalla funzione $P(1,s_i)$. $\tilde{D}(T_i)$ rappresenta la distribuzione degli accessi nel tempo, una sorta di storia del passato, mentre T_i indica il tempo trascorso dall'ultima richieste ricevuta per l'oggetto *i*.

HYBRID

Questa strategia operativa prevede che la funzione applicata all'oggetto *i* proveniente dal server *s* dipenda dalla seguente formula:

$$f(i) = \frac{\left(c_s + \frac{W_b}{b_s} \right) f_i^{W_n}}{s_i};$$

dove abbiamo i soliti parametri f_i ed s_i, che indicano rispettivamente il numero di richieste già riscontrate per l'oggetto e la sua dimensione. Inoltre abbiamo c_s che indica il tempo necessario per contattare il server *s* e b_s che indica la larghezza di banda disponibile verso il server. Infine W_b e W_n che sono parametri variabili a seconda delle esigenze. La stima dei valori di c_s e di b_s viene basata sui tempi di caricamento degli oggetti dal server *s* avvenuti nel recente passato.

Le strategie appartenenti alla famiglia appena descritta sono caratterizzate dall'utilizzo di combinazioni variabili dei parametri considerati. Manipolando questi è possibile ottimizzare alcune delle prestazioni più interessanti per l'utilizzatore. Un altro vantaggio è rappresentato sicuramente dal considerare contemporaneamente un certo numero di fattori. Questa scelta consente di affrontare al meglio le diverse situazioni operative che possono verificarsi. Non riscontriamo solamente vantaggi nemmeno in questa categoria. Risulta, infatti, un'operazione difficoltosa la scelta appropriata dell'importanza da assegnare ai parametri da cui dipendono le mie prestazioni. Ad esempio, il considerare parametri che si riferiscono alla storia passata della connessione, della rete o del traffico circolante su di essa, può, in generale, non essere la soluzione

migliore. Questo considerando il fatto che le condizioni operative possono variare frequentemente. Si pensa quindi alla possibilità di introdurre fattori dinamici in grado di adattarsi alle diverse situazioni operative, ma questo non può non introdurre una maggiore complessità nella realizzazione del processo di rimozione.

L'utilizzo, in questa categoria di politiche, dei tempi di latenza o di accesso nei confronti degli oggetti presenti sul server originario, introduce ulteriori problematiche. Infatti, se recency e frequency sono informazioni facilmente ricavabili dalla recente storia passata del sistema, non si può affermare altrettanto per quanto riguarda, ad esempio, la latenza. La misurazione del valore della latenza viene effettuata sul proxy, ma risulta influenzata da numerosi componenti presenti lungo tutto il cammino che separa il proxy dal server. Tale valore può essere soggetto, quindi, a frequenti fluttuazioni, che complicano ogni decisione a livello di determinazione dei migliori parametri applicabili.

2.6 Strategie random

Questa categoria si basa fondamentalmente sulla casualità della scelta degli oggetti da rimuovere dalla cache. Ne presentiamo ora alcune varianti.

RAND

Questa risulta essere, probabilmente, la più semplice strategia tra quelle illustrate, in quanto prevede l'assoluta casualità. Viene infatti rimosso un oggetto qualsiasi tra quelli presenti nella memoria cache.

HARMONIC

Mentre la strategia RAND considera tutti gli oggetti aventi la medesima probabilità di essere scelti per l'eliminazione, questa prevede che ogni oggetto abbia una propria probabilità caratteristica. Viene rimosso l'oggetto scelto casualmente tra quelli aventi valore di probabilità inferiore; probabilità che risulta essere inversamente proporzionale alla dimensione

dell'oggetto e direttamente proporzionale al costo di reperimento dello stesso.

LRU-C e LRU-S

Questa strategia è in sostanza una versione random della più classica politica LRU. Considero

$$c_{\max} = \max \{c_1, c_2, \ldots, c_N\}$$

come il valore massimo tra i costi di accesso agli *N* oggetti considerati e

$$\tilde{c}_i = c_i / c_{\max}$$

come il costo normalizzato dell'oggetto *i*. Quando l'oggetto *i* viene richiesto, questo viene spostato in testa alla lista degli oggetti in cache con probabilità $\tilde{c}_i$, altrimenti non viene fatto nulla.

La politica LRU-S si differenzia dalla precedente perché il parametro qui utilizzato è la dimensione dell'oggetto considerato. Viene posto

$$s_{\min} = \min \{s_1, s_2, \ldots, s_N\}$$

pari alla dimensione più piccola fra gli *N* oggetti presi in considerazione e

$$d_i = s_{\min} / s_i$$

che rappresenta la densità normalizzata dell'oggetto *i*. Questa politica opera come la più classica LRU, con probabilità d_i e altrimenti lo stato della cache non viene modificato.

Queste due politiche sono state presentate insieme, perché esiste anche una proposta che opera utilizzando entrambi i parametri visti per le singole strategie. Vengono definite la seguenti quantità:

$$\beta_i = \frac{c_i}{s_i}; \qquad \beta_{\max} = \max_i \{\beta_i\};$$
$$\tilde{\beta}_i = \frac{\beta_i}{\beta_{\max}}.$$

Nel momento in cui si verifica una richiesta sull'oggetto *i*, questo algoritmo si comporta come quello LRU con probabilità $\tilde{\beta}_i$ e con probabilità $(1-\tilde{\beta}_i)$ non varierà lo stato interno della memoria cache.

Le strategie random presentano un approccio al problema completamente diverso rispetto a quelli descritti in precedenza. Il loro scopo principale è quello di ridurre la complessità del processo di rimozione senza sacrificare troppo la sua qualità. Il principale vantaggio di queste politiche risulta quindi essere quello della semplicità di implementazione. Uno svantaggio è presentato dal fatto che risultano essere di difficile valutazione, in quanto, una stessa strategia applicata al medesimo sistema, può rispondere con prestazioni differenti tra loro.

2.7 Caching di oggetti multimediali

Nel caso in cui, gli oggetti da memorizzare nella memoria cache, siano dati di tipo multimediale, il caching ha un'importanza abbastanza rilevante. Questo perché, a differenza dei casi in cui si prendono in considerazione, ad esempio, documenti HTML, gli oggetti multimediali possono avere dimensioni notevoli e molto diverse fra loro. Un aspetto da considerare è che, in alcuni casi, può essere possibile ridurre le dimensioni degli oggetti considerati senza sacrificare eccessivamente la loro qualità. Un procedimento possibile se non si deve sottostare a determinati vincoli riguardanti la qualità dei contenuti. A prescindere da questo bisogna senz'altro tenere presente che la dimensione della cache dovrà essere adeguata all'ambiente in cui viene utilizzata.

Esistono alcune proposte di soluzioni specifiche relative al caching di materiale multimediale in ambienti con particolari caratteristiche, ma si basano soprattutto su simulazioni. Per ora non vi sono esperienze di applicazioni operative in ambienti reali e non è ben chiaro quali siano le metriche più appropriate per valutarne l'efficacia.

In base a queste considerazioni, si ritiene che un aspetto fondamentale, in questo ambito operativo, sia rappresentato dal concetto di località temporale. È possibile, infatti, che un documento multimediale attraversi un periodo di maggiore popolarità all'interno della rete in cui è

reperibile. Possiamo, per sostenere queste considerazioni, indicare come esempio gli allegati alle e-mail che si ricevono quasi ogni giorno. Questi allegati, che possono essere anche di tipo multimediale, attraversano periodi di maggior popolarità. Durante tali periodi, vengono diffusi ampiamente tra i diversi utenti, seguendo una sorta di "moda" del momento. Oltre al concetto di località temporale, vi sono altri aspetti che meritano considerazione: la dimensione e la frequenza di accesso, solitamente indicata dal numero delle richieste. Nella fase di progettazione del sistema cercheremo di dare la giusta importanza a questi tre principali aspetti.

2.8 Conclusioni

Nel capitolo appena concluso abbiamo affrontato la fondamentale problematica delle politiche di rimozione e sostituzione (*replacement policies*) applicabili nella gestione di una cache. Abbiamo suddiviso queste strategie in cinque principali categorie e descritto le linee guida di ogni politica considerata. Dalle descrizioni effettuate si è cercato di individuare alcuni dei principali vantaggi e svantaggi che possono presentarsi utilizzandole. Infine, sono state fatte alcune semplici considerazioni riguardanti il nostro ambito applicativo.

Nel capitolo successivo non tratteremo direttamente argomentazioni riguardanti i sistemi di caching, ma cercheremo di introdurre alcune tecnologie necessarie per la realizzazione del nostro progetto.

CAPITOLO 3

3 Gli Agenti Mobili e le tecnologie utilizzate: SOMA, MUM e JMF

Questo terzo capitolo si pone come obbiettivo quello di introdurre il paradigma ad agenti mobili e le tecnologie coinvolte. Nella prima parte affronteremo brevemente il concetto di mobilità di codice e degli agenti mobili. Successivamente prenderemo in considerazione le tecnologie coinvolte, facendo distinzione tra quelle utilizzate e quelle al cui interno si è andato ad operare per la realizzazione di questo progetto di tesi. Verranno descritti due ambienti, all'interno dei quali si va ad operare e di cui il sistema di caching progettato diverrà parte integrante. Questi ambienti sono SOMA e MUM. In seguito si presenterà un framework proposto dalla SUN e identificato dall'acronimo JMF, che rappresenta un'estensione del linguaggio Java riguardante i tipi di dati multimediali. Nei diversi paragrafi di questo capitolo andremo ad evidenziare gli aspetti che maggiormente riguardano il nostro progetto. Non sarebbe infatti possibile, in questo contesto, trattare in maniera completa tutte le caratteristiche delle tecnologie sopra citate, data la vastità degli argomenti.

3.1 La mobilità di codice e il paradigma ad agenti mobili

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

Innanzitutto, cosa si intende per mobilità di codice. In linea generale si tratta di considerare il trasferimento non più dei soli dati, ma anche del codice, in modo che possa essere eseguito localmente dove è necessario. Questo permette di aggiungere adattabilità e flessibilità alle applicazioni distribuite. Per approfondire questi concetti è necessario introdurre alcune definizioni. Con *Execution Unit* (EU) intendiamo un flusso computazionale sequenziale, ovvero l'insieme del codice e del suo stato di esecuzione,

come ad esempio un thread ed il suo contesto. Con *Computational Environment* (CE) intendiamo l'ambiente all'interno del quale le EU operano. Occorre ora fare una distinzione tra mobilità debole e mobilità forte. Si parla di mobilità debole quando ho la migrazione del solo codice tra due diverse EU che operano su diversi CE. Si parla invece, di mobilità forte, quando il trasferimento riguarda un'intera EU, cioè codice e stato di esecuzione, che migra da un CE ad un altro.

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

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

3.2 SOMA

Il nome SOMA, acronimo di *Secure and Open Mobile Agent*, sta ad indicare un ambiente ad agenti mobili realizzato presso il Dipartimento di Elettronica, Informatica e Sistemistica (DEIS) dell'Università degli Studi di Bologna. Come accennato in precedenza, questa trattazione non ha l'obbiettivo di illustrare in maniera approfondita ed esaustiva tutte le caratteristiche di questo ambiente di lavoro, ma solamente di introdurne le caratteristiche architetturali principali.

A questo scopo possiamo vedere come SOMA sia basato sul linguaggio di programmazione Java e ne sfrutti tutte le principali potenzialità. Risulta essere in particolare un linguaggio particolarmente adatto all'implementazione di agenti mobili. Questo grazie ad alcune sue caratteristiche:

- la portabilità; Java è un linguaggio interpretato e per questo può essere eseguito in qualsiasi ambiente, ponendo come unico vincolo che questo possegga una *Java Virtual Machine* (JVM). La JVM si occupa di trasformare il bytecode originario, in linguaggio macchina, permettendone la portabilità su diverse architetture hardware e software;
- l'ereditarietà e l'estendibilità; è un linguaggio di tipo object-oriented che consente una progettazione modulare del codice sfruttando classi messe a disposizione dall'ambiente operativo, modificandole e/o ampliandole;
- la dinamicità; questo linguaggio prevede un meccanismo di caricamento dinamico delle classi, anche da sorgenti remote, e di collegamento dinamico all'applicazione corrente;
- la serializzabilità; è consentito serializzare gli oggetti, cioè è possibile effettuare una rappresentazione di questi come stream di byte, con la possibilità di trasferirli in rete;
- la sicurezza;il linguaggio Java presenta importanti caratteristiche di sicurezza built-in. Nelle sue ultime versioni questa architettura di sicurezza è stata aggiornata, rendendola molto più flessibile ed espressiva. È caratterizzata da domini di protezione, controlli di accesso e permessi da associare sia a codice remoto che a codice locale.

La scelta di questo linguaggio di programmazione porta ad una scelta obbligata riguardante il modello di mobilità da utilizzare nelle fasi di sviluppo. Il linguaggio Java è un linguaggio di programmazione interpretato, e per questo una parte dello stato di esecuzione rimane racchiuso all'interno dell'interprete stesso. Il reperimento di questa porzione dello stato di esecuzione risulta sostanzialmente impossibile.

Questo a meno di modifiche effettuate sull'interprete, che farebbero, però, rinunciare ad una caratteristica fondamentale dei sistemi ad agenti mobili, la portabilità.

Dalle considerazioni appena effettuate si intuisce immediatamente come SOMA possa essere definito un sistema a mobilità debole, dal momento che è basato sul linguaggio Java. Dopo aver considerato questa principale caratteristica, possiamo fare le seguenti affermazioni:

- per *codice* intendiamo le classi Java;
- per *risorse* intendiamo gli oggetti del linguaggio Java;
- per *Execution Unit* (EU) intendiamo i thread Java;
- ai *siti* (in SOMA chiamati place) facciamo corrispondere macchine virtuali Java (JVM);
- le interazioni avvengono tramite chiamate a metodi, scambio di messaggi e, quando i componenti risultano su JVM diverse, tramite scambio di comandi.

3.2.1 Caratteristiche principali dell'ambiente SOMA

L'ambiente SOMA è caratterizzato da alcune principali caratteristiche. Le andiamo ad illustrare brevemente:

- *astrazioni di località*: esistono due diversi livelli di astrazione, indicati come place e dominio, che rappresentano, tramite una semplice interfaccia, le località tipiche del web: il nodo e la rete locale (LAN). Ho poi un ulteriore livello di supervisione che si occupa del coordinamento tra i domini;
- *scalabilità*: questa caratteristica è sviluppata su diversi livelli, a cominciare dalla configurazione del sistema, che può essere esteso a piacimento. Posso parlare di scalabilità anche per quanto riguarda la gestione degli utenti e il numero degli agenti in esecuzione, che non viene in nessun modo limitato;
- *sicurezza*: è realizzata la protezione degli ambienti di esecuzione e la parziale protezione degli agenti, riguardante la verifica di integrità del codice e la riservatezza della comunicazione dello

stato. Le operazioni degli utenti sul sistema, sono controllate, inoltre, mediante l'utilizzo di password;

- *apertura*: essendo questo sistema progettato seguendo gli standard proposti da OMG (Object Management Group), gli agenti sono in grado di interagire con gli agenti di altri sistemi aderenti ai medesimi standard;
- *dinamicità*: la configurazione del sistema è dinamica, consentendo l'inserimento o la rimozione di ambienti di esecuzione ed utenti in qualsiasi istante;
- *prestazioni*: mediante l'utilizzo del paradigma ad agenti mobili si intende superare i limiti del modello client-server, cercando di ottenere migliori prestazioni globali;
- *portabilità*: come già evidenziato in precedenza, l'utilizzo del linguaggio Java, consente di realizzare codice portabile e quindi utilizzabile su diverse piattaforme hardware e software;
- *fault tolerance*: questa caratteristica di tolleranza ai guasti prevede che i nodi facenti parte del sistema possano essere attivi o non attivi.

Abbiamo introdotto alcuni concetti relativi alle astrazioni di località che meritano qualche ulteriore spiegazione. Con il termine place, si vuole rappresentare l'ambiente di esecuzione (CE) degli agenti, come astrazione del concetto nodo. Al suo interno gli agenti possono interagire con le risorse locali e con gli altri agenti residenti. Un place è caratterizzato da un nome che lo identifica all'interno del sistema. Una delle caratteristiche principali di questo concetto è che all'interno di un unico nodo possono esservi più place, ma che ognuno di essi è confinato all'interno di un unico nodo. Un'aggregazione di place, con caratteristiche comuni di tipo fisico (appartenenza alla stessa LAN) o logico (amministrazione e politiche comuni), è individuata dal termine dominio. Infine ho l'astrazione relativa al supervisore, che rappresenta un coordinatore dei domini, per ciò che riguarda la configurazione dinamica della topologia e degli utenti. Esso conosce l'intera composizione del sistema, compresi tutti gli utenti, e può comunicare queste informazioni agli altri componenti, quando necessario.

3.3 MUM

Il sistema MUM, acronimo di *Multimedia agent based Ubiquitous Multimedia middleware*, è un ambiente operante all'interno di SOMA e realizzato anch'esso presso il Dipartimento di Elettronica, Informatica e Sistemistica (DEIS) dell'Università degli Studi di Bologna. È all'interno di questo prodotto che andrà ad integrarsi il sistema di caching progetto di questa tesi.

L'ambiente che andiamo a presentare è stato progettato con l'intenzione di supportare la pubblicazione e la fruizione di presentazioni multimediali di varia natura e diversa qualità operando sia su rete fissa che su rete mobile. Il concetto di presentazione multimediale è stato abbinato all'aggregato di uno o più oggetti multimediali, ognuno dei quali caratterizzato da un attributo di qualità. A sua volta il termine oggetto multimediale sta ad indicare l'astrazione di un'istanza di un determinato tipo di dato multimediale contenuto nel sistema, ovvero un descrittore di tale istanza. In pratica il tipo di dato incapsula al suo interno un solo tipo di media, che può essere audio o video o altro ancora. Dal momento che questa applicazione intende supportare documenti registrati e non filmati in diretta, come ad esempio una videoconferenza, avremo che ogni oggetto multimediale troverà corrispondenza in un determinato file contenuto nel sistema.

Ogni presentazione multimediale è rappresentata da un metadato utilizzato per aggregare i diversi oggetti multimediali che la compongono. Avrò un attributo che indica la sua posizione nel sistema ed uno che ne indica la qualità, entrambi intesi come attributi dell'aggregato. Viene poi introdotta un'altra astrazione: il contenuto multimediale. Questo è introdotto per suddividere le presentazioni multimediali in sottoinsiemi mutuamente esclusivi ed è caratterizzato da un titolo, che lo identificherà all'interno dell'intero sistema. Gli utenti che accedono al sistema hanno la possibilità di richiedere il delivery dei titoli in esso contenuti, potendo comandarne la riproduzione mediante un'interfaccia simile a quella di un videoregistratore. Viene inoltre supportata la mobilità della sessione verso

un diverso terminale e la mobilità dell'utente, nel caso in cui l'accesso al sistema avvenga da una rete mobile. Dal momento che sono previsti diversi livelli di qualità per le presentazioni, sarà il sistema a selezionare quello più adatto alle caratteristiche della piattaforma computazionale dalla quale si sta accedendo. Questo comporta che venga gestito un monitoraggio delle risorse disponibili ed una adeguata prenotazione di quelle necessarie a soddisfare la richiesta ricevuta.

3.3.1 Fruizione del materiale multimediale

Il modello computazionale prescelto per la realizzazione del sistema è il modello client-server, con l'abbinamento del modello ad agenti mobili. Lo scopo è quello di far interagire questi due diversi approcci, cercando di sfruttare al meglio le caratteristiche di entrambi. Le principali entità presenti in questo ambiente sono quattro: *ClientAgent*, *Client*, *ProxyAgent* e *Server*. In particolare saranno collegate fra loro mediante un percorso che avrà come punti terminali un client ed un server e passerà attraverso uno o più proxy. Non tutte queste entità sono state realizzate utilizzando il paradigma ad agenti mobili, alcune sono realizzate come entità fisse. Andiamo ad analizzare brevemente le entità coinvolte.

ClientAgent

Questa entità può essere considerata come l'entry point del sistema dal lato client. È realizzato come agente mobile ed ha come compiti principali quello di inizializzare la sessione e di gestire le interazioni dell'utilizzatore con il resto del sistema distribuito. Il fatto di essere un agente mobile facilità la sua comunicazione con gli altri agenti del sistema e permette di supportare la mobilità degli utenti.

Client

È l'entità fissa che riceve i flussi multimediali richiesti e risiede sulla macchina dalla quale l'utente è entrato nel sistema. Nonostante sia un'entità fissa, non è necessario che il codice da utilizzare sia già presente

sulla macchina dalla quale viene lanciato, in quanto può essere scaricato a run-time, seguendo il paradigma del code on demand (COD).

Server

Come il client è un'entità fissa, ma anch'esso ha la possibilità di scaricare il software necessario a run-time, con il paradigma del COD. Una volta lanciato su di una macchina, sarà lui ad occuparsi della gestione di tutte le richieste che giungono alla medesima. Come si può facilmente intuire è l'altro end-point, insieme al client, della comunicazione di flussi multimediali. In particolare, il server, è colui che si occupa della spedizione dei flussi multimediali richiesti.

ProxyAgent

Il ProxyAgent è realizzato come agente mobile, per supportare la mobilità dei terminali, anche se per la gestione vera e propria dei flussi si avvale di un'entità chiamata Proxy. In particolare il ProxyAgent si muoverà sulla rete fissa seguendo i movimenti del terminale all'interno della rete mobile, tipicamente una rete wireless. Uno dei compiti di questa entità è che agisca da disaccoppiatore tra il client e l'infrastruttura della rete fissa, in modo da garantire flessibilità e modularità per futuri sviluppi riguardanti soprattutto il lato client. Fra questi futuri sviluppi potrebbe ad esempio essere compreso l'utilizzo di protocolli di comunicazione diversi dal classico TCP/IP. Un suo ulteriore compito può essere quello di trasformare, qualora sia necessario, i dati multimediali in arrivo, prima di dirottarli sul client. Operazione che può essere richiesta nel caso di presenza di particolari piattaforme software per lo streaming multimediale poste sul client, oppure nel caso in cui i dati debbano essere trasmessi ad un terminale mobile con basse capacità computazionali.

3.3.2 Gestione dei contenuti multimediali

L'architettura dell'ambiente MUM risulta essere suddivisa su tre livelli: *Middleware Mechanisms Layer*, *Middleware Facilities Layer* e *Application Layer*. Nel Middleware Mechanisms Layer vengono

implementati i servizi di base dell'ambiente, mentre il Middleware Facilities Layer utilizzando quei servizi di base realizza funzionalità più complesse messe a disposizione dello sviluppatore dell'applicativo. L'Application Layer, infine, racchiude al suo interno tutte le entità che realizzano l'applicazione e l'interazione con l'utente del sistema.

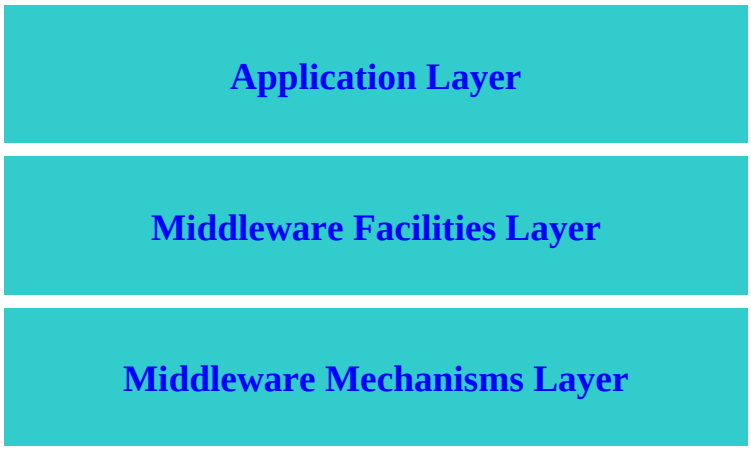


Figura 3.1: layer dell'ambiente MUM.

All'interno del Middleware Mechanisms Layer viene affrontato il problema riguardante la gestione dei contenuti multimediali. Questo servizio si basa su di una classica architettura di tipo client-server. Quando un nuovo nodo viene aggiunto alla gerarchia dei nodi, richiede al nodo padre un riferimento al database (server) che contiene le informazioni relative alle presentazioni multimediali presenti nel sistema. Localmente ogni nodo mette a disposizione uno stub (client) per l'interrogazione del database remoto. Questa risulta essere a grandi linea l'idea di fondo per la gestione.

3.3.3 Gestione dello streaming

Durante la fase di streaming vengono coinvolte le entità fondamentali del sistema, ovvero il client, il proxy ed il server. Ognuna di

queste entità è in grado di gestire uno o più flussi, operando a livello di sessione. Possiamo vederle come insiemi di più oggetti, ed in particolare composte da:

- gestore della sessione;
- uno o più agenti di gestione dei singoli flussi;
- un protocollo per gestire il controllo di sessione;
- un manager della qualità di servizio.

L'architettura del client, del server e del proxy si assomigliano molto fra loro. Ognuna di esse è caratterizzata, ad esempio dalla presenza di uno Streaming Agent che ha il compito di gestire i flussi. La differenza fra le diverse entità consiste nel fatto che, nel caso del client, questo agente dovrà gestire solamente flussi in ingresso. Per quanto riguarda l'entità server dovrà occuparsi unicamente dei flussi in uscita, mentre, nel caso di entità proxy, dovrà riportare in uscita tutti i flussi che arrivano in ingresso.

Avrò poi il gestore della qualità di servizio che deve interagire con il gestore delle risorse, per la prenotazione delle stesse e per il monitoraggio dello stato dello streaming. Il gestore delle risorse può, inoltre, comunicare al gestore della qualità di servizio il verificarsi di determinati problemi, che potrebbero richiedere un adattamento della sessione. All'interno di queste entità è presente anche una *Protocol Unit*, che incapsula i protocolli di comunicazione fra le diverse entità, sia per quanto riguarda il controllo della sessione, sia per lo scambio di informazioni relative alla qualità di servizio.

3.4 Java Media Framework (JMF)

Come accennato nell'introduzione di questo capitolo si tratta di un prodotto opzionale che estende il linguaggio Java, ed in particolare la piattaforma JAVA2SE™, consentendo le gestione di tipi di dati multimediali. Il *Java Media Framework* è sostanzialmente costituito da un insieme di API (Application Program Interface) creato appositamente per consentire di incapsulare dati di tipo multimediale in applicazioni o applet Java. Sun e IBM hanno sviluppato questo componente con l'idea di fornire supporto ai più comuni standard di memorizzazione di contenuti

multimediali, come ad esempio: MPEG-1, MPEG-2, QuickTime, AVI, WAV, AU, MIDI e AIFF.

Una delle caratteristiche principali che sono necessarie quando si vuole gestire materiale multimediale, ad esempio tramite un lettore multimediale, è la velocità di computazione. È richiesta un'elevata velocità computazionale per garantire che operazioni quali, ad esempio, la decompressione delle immagini o il rendering vengano svolte in maniera soddisfacente. Sappiamo che Java utilizza una Java Virtual Machine, che interpreta il byte-code generato dal compilatore del linguaggio. Questo meccanismo, che ha come suo punto di forza la garanzia di portabilità, impone, allo stesso tempo, seri vincoli in termini di prestazioni, qualora si richieda elevata velocità computazionale. Per ovviare a questi problemi, e voler trattare dati multimediali, è necessario ricorrere all'utilizzo di codice nativo della piattaforma su cui si opera. Questo comporta che il programmatore abbia una buona conoscenza delle funzioni native e, cosa ancora più importante, pone un serio vincolo alla portabilità dell'applicazione realizzata. Le API JMF cercano di ovviare a questi problemi.

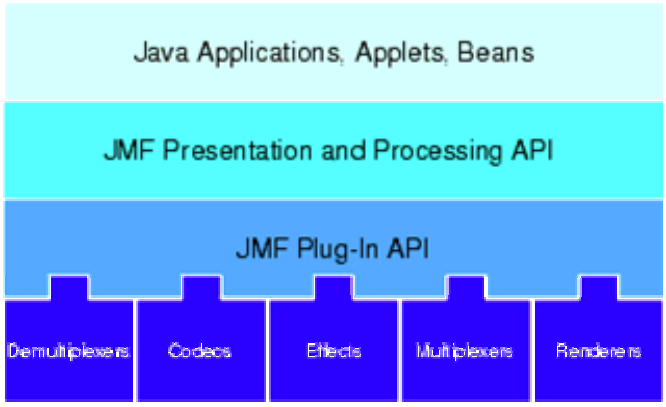


Figura 3.2: architettura del Java Media Framework.

Il componente JMF mette a disposizione degli sviluppatori un insieme di chiamate ad "alto livello" che consentono la gestione di codice

nativo. L'applicazione o l'applet, che integra JMF, non ha bisogno di sapere se e quando deve ricorrere al codice nativo per soddisfare una determinata richiesta. La versione 2.1.1 di JMF rende disponibili classi che consentono di sviluppare applicazioni che catturino dati multimediali e che forniscano la possibilità di controlli ulteriori sull'elaborazione e la riproduzione del materiale stesso.

In particolare il componente JMF 2.1.1 è stato progettato per ottenere i seguenti vantaggi:

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

Andiamo ora a presentare brevemente alcune delle classi che costituiscono questo prodotto.

3.4.1 Il componente *Player*

Definendo con *media stream* il dato multimediale ottenuto da un file locale, acquisito dalla rete oppure da un dispositivo di input (videocamera, microfono, ecc.), possiamo considerare il *Player* come quella struttura che

ne consente la riproduzione. È possibile paragonare il player, per le funzioni di controllo che rende disponibili, ad un semplice videoregistratore. Grazie ad esso, gli sviluppatori di software, possono disinteressarsi delle chiamate al codice nativo, ed allo stesso tempo possono occupare risorse necessarie per la riproduzione e rilasciarle quando non più necessarie.

Queste chiamate a metodi e classi di alto livello rendono trasparente al programmatore la connessione che viene stabilita tra la JVM e le routine specifiche di sistema. Il dato multimediale in ingresso al player viene collegato ad esso mediante una struttura denominata *data source*, che fa riferimento al dato vero e proprio. Quest'ultima struttura potrebbe essere considerata alla stregua di una videocassetta per un videoregistratore, il player. Nella figura che segue viene rappresentato quanto appena descritto.

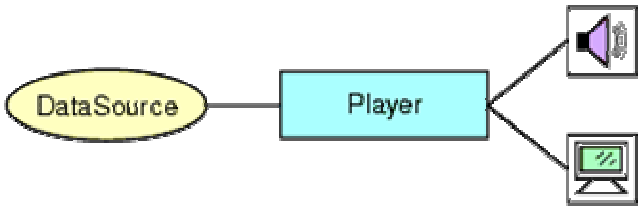


Figura 3.3: ricezione e riproduzione di flussi multimediali.

Come si può notare da questa figura, il media stream riprodotto potrebbe essere di tipo audio o di tipo video o audio e video. Infatti un media stream, anche se identificato da un'unica traccia, potrebbe contenere più canali, come ad esempio un canale audio ed uno video.

Un player può trovarsi, operando normalmente, in uno dei suoi sei stati possibili. Ho cinque di questi stati che individuano una comune situazione di interruzione della riproduzione, vuoi perché è stata interrotta, vuoi perché non è ancora iniziata. Solitamente vengono attraversati questi primi stati fino a giungere al sesto stato, quello dell'effettiva riproduzione

del contenuto multimediale. La transizione fra questi stati avviene solitamente in seguito al verificarsi di determinati eventi.

In generale, il player, una volta inizializzato, può fornire, se presente, un componente visuale utilizzato per la fruizione del dato multimediale. All'interno dell'ambiente MUM il player è quell'entità che permette al client di ricevere e riprodurre i flussi multimediali.

3.4.2 Il componente *Processor*

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

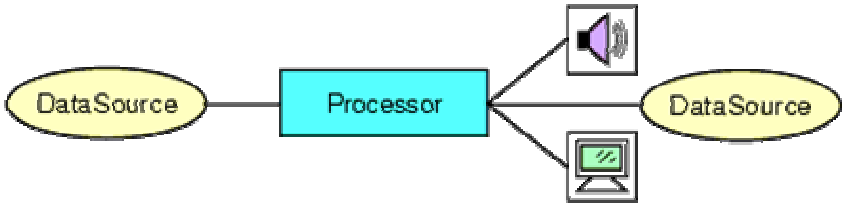


Figura 3.4: ricezione ed invio di flussi multimediali.

Il vantaggio di poter avere un output reindirizzato direttamente su di un data source è rappresentato dal fatto che questa uscita può essere presentata direttamente all'ingresso di un player, di un data sink (che descriveremo nel prossimo paragrafo) o di un altro processor.

Facendo riferimento alle caratteristiche di questo componente fino a qui descritte è intuibile come sia stato utilizzato all'interno dell'ambiente MUM: come parte integrante delle entità server e proxy.

A differenza del player, il processor può trovarsi in uno dei suoi otto stati possibili. Questi otto stati vengono suddivisi in due gruppi principali che individuano due condizioni del processor: *unrealized* e *realized*. Durante le sue fasi di inizializzazione e configurazione è definito unrealized, mentre al termine di queste fasi viene definito realized ed è pronto per avviare la trasmissione dei dati. La fase di configurazione di un processor prevede la connessione del data source, la demultiplazione dello stream di ingresso e il reperimento delle informazioni sul formato dei dati multimediali in ingresso. La demultiplazione consiste sostanzialmente nell'analisi del flusso multimediale in ingresso per l'individuazione di eventuali tracce multiple. Nel qual caso si verifichi la presenza di tracce multiple queste vengono automaticamente suddivise ed inviate separatamente in output. Se l'output è indirizzato verso un data source, le diverse tracce individuate devono essere raggruppate nuovamente in modo da ottenere un unico flusso multimediale d'uscita (come evidenziato nella figura che segue).

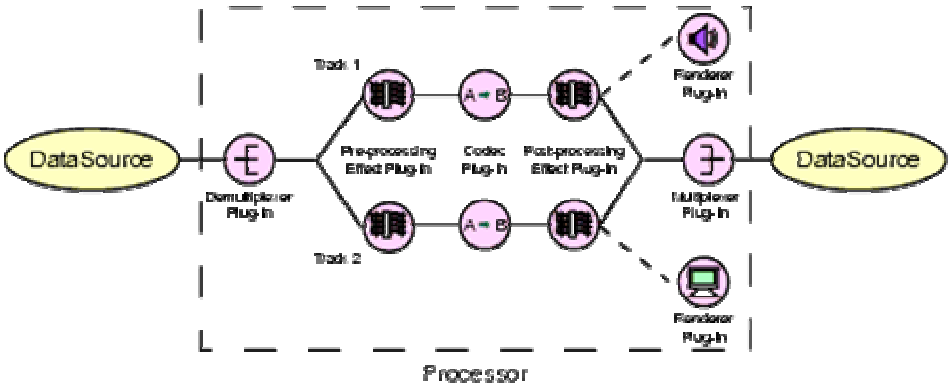


Figura 3.5: demultiplazione e multiplazione del flusso multimediale.

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

3.4.3 Il componente *DataSource*

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

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

3.4.4 Il componente *DataSink*

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

3.5 Conclusioni

In questo capitolo la nostra attenzione è stata focalizzata sulle tecnologie coinvolte nella realizzazione del nostro progetto. Di ognuna di esse abbiamo cercato di fare una panoramica introduttiva, cercando di

fornire informazioni generiche di inquadramento. Successivamente siamo andati ad approfondire quegli argomenti che risultano direttamente coinvolti dal nostro lavoro. Alcuni di questi verranno ripresi anche nei capitoli successivi, trattando le loro più specifiche caratteristiche tecniche ed implementative. In particolare, nel prossimo capitolo andremo ad affrontare l'analisi del sistema di caching da realizzare.

CAPITOLO 4

4 Analisi e progettazione del sistema di caching per materiale multimediale

In questo capitolo andremo ad affrontare le problematiche relative alla realizzazione del sistema di caching per contenuti multimediali. Nei capitoli precedenti si è voluto introdurre le principali argomentazioni riguardanti i diversi aspetti del problema. Prima affrontando i numerosi argomenti che vengono coinvolti dalla realizzazione di operazioni di caching, e poi ponendo la nostra attenzione sulle politiche di rimpiazzamento degli oggetti memorizzati. Infine si è cercato di introdurre brevemente le tecnologie coinvolte in questo progetto, cercando di focalizzare le caratteristiche che più risultano interessanti per i nostri fini. L'obbiettivo che questo capitolo si pone è quello di fare un'analisi del problema, che sia di supporto alle seguenti fasi di progettazione ed implementazione.

4.1 Inquadramento generale del problema

Il problema da affrontare è la realizzazione di un sistema di caching. Come già accennato in precedenza, l'obbiettivo è quello di ottenere un sistema che si integri all'interno dell'ambiente SOMA operando, in particolar modo, nell'ambito del middleware MUM.

Prendendo in considerazione l'architettura del middleware MUM notiamo come il nostro progetto andrà ad interessare l'entità proxy coinvolta. Abbiamo visto come il percorso fra i due end-point della comunicazione, client e server, debba passare attraverso una o più entità proxy. Il mantenimento dei dati all'interno di una memoria cache verrà pertanto effettuato all'interno di tali entità. Il flusso multimediale proveniente dal server viene ricevuto da un proxy ed inoltrato, o ad un altro proxy, o direttamente all'end-point client. È durante questa fase che il dato deve essere oltre che inoltrato anche memorizzato su di un supporto fisico

permanente presente all'interno dell'entità proxy. Tale memorizzazione sarà gestita da opportune politiche di rimpiazzamento, che interverranno in caso di saturazione della memoria o di spazio libero insufficiente. Pertanto il nostro sistema dovrà occuparsi della memorizzazione del flusso multimediale per consentirne un successivo reperimento. In particolare, il materiale multimediale circolante, consisterà in documenti video, in quanto il nostro sistema mira a fornire appoggio ad una applicazione per video streaming.

L'utilizzo delle tecniche di caching, come indicato nei capitoli precedenti, permette di ottenere un miglioramento delle prestazioni globali del sistema. Questi miglioramenti delle prestazioni sono evidenziati principalmente da una riduzione dei tempi di reperimento dei documenti multimediali, da una riduzione del traffico circolante sulla rete e da una riduzione del carico di lavoro sulle entità server.

4.2 Analisi dei requisiti

L'operazione principale, richiesta al sistema di caching, risulta essere quella della memorizzazione del flusso multimediale che transita attraverso le entità proxy. Esula da questo progetto la gestione delle problematiche riguardanti le scelte da effettuare per determinare quali oggetti vadano memorizzati in cache e quali non si ritiene necessario debbano essere mantenuti.

Il gestore della memoria cache dovrà pertanto intercettare il flusso multimediale entrante e indirizzarlo verso un file creato all'interno di un direttorio riservato alla memoria cache. Questa operazione non dovrà interferire sulla normale procedura di inoltro dei dati verso l'entità successiva al proxy, all'interno della gerarchia del sistema. La memorizzazione dei dati prosegue di pari passo con l'andamento del flusso multimediale, e quindi con la riproduzione del documento video sull'entità client.

4.2.1 Politiche di gestione

Alcuni dei gradi di libertà previsti nella gestione di una memoria cache sono individuati dalle politiche utilizzate. Vediamo ora quelle che sono e non sono coinvolte direttamente nel nostro progetto.

Possiamo escludere a priori le *Prefetching policies*, in quanto il materiale multimediale che andiamo a considerare risulta solitamente di dimensioni notevoli. Questo sconsiglia quindi un pre-caricamento in memoria dei documenti, in quanto sarebbe necessaria un'occupazione di risorse prolungata nel tempo. Inoltre, si avrebbe l'occupazione di una quantità di memoria cache considerevole, senza avere la certezza che il video memorizzato venga, poi, effettivamente richiesto.

Le *Routing policies* e le *Placement policies* potrebbero, invece, essere coinvolte nella realizzazione di un sistema di caching. Introducono però, problematiche relative ad un più alto livello di gestione e coordinamento rispetto a quello affrontato nel nostro progetto.

Le *Coherence policies* affrontano problematiche relative alla consistenza dei dati contenuti nella cache. Queste considerazioni non risultano avere importanza nella realizzazione del nostro sistema, data la tipologia di documenti coinvolti nel nostro progetto. Si ritiene infatti, che le informazioni video considerate, non necessitino di frequenti aggiornamenti, e quindi, è pressoché scongiurata la presenza di materiale obsoleto in quanto non aggiornato.

Le politiche di gestione sulle quali ci siamo concentrati risultano essere le *Replacement policies*, descritte nel secondo capitolo di questa tesi. La scelta della politica di rimozione e rimpiazzamento da applicare, influenza le prestazioni dell'intero sistema.

4.3 Problematiche affrontate

In questa parte cercheremo di effettuare un'analisi dei diversi aspetti da prendere in considerazione nella successiva fase di sviluppo. Per consentire la realizzazione delle operazioni richieste, è necessario effettuare delle scelte. Queste scelte riguardano principalmente l'organizzazione interna della memoria cache, le modalità di salvataggio dei flussi

multimediali e le politiche adottate per la gestione del sistema. Senza dimenticare la scelta delle opzioni rese disponibili all'utente per consentire una personalizzazione del sistema, cercando di adattarlo all'ambiente entro il quale sarà operativo.

4.3.1 I formati video

Per rendere possibile il mantenimento in cache degli oggetti multimediali presenti nel sistema bisogna fare alcune considerazioni. Come accennato in precedenza, i contenuti multimediali trattati sono in particolar modo elementi video. In questi ultimi anni c'è stato un notevole aumento della diffusione di contenuti video di tipo digitale all'interno della rete internet. Questo anche grazie alla sempre maggiore diffusione dei supporti dvd ed agli scambi, non sempre legali, di video digitali attraverso la rete. I sempre più aggiornati programmi di compressione riescono a comprimere un film, originariamente in formato dvd, fino a farlo contenere in un normale cd. Questo senza che la qualità del documento venga eccessivamente deteriorata. Vediamo quindi come un film medio possa avere dimensioni pari circa a 700 megabyte o superiori.

La dimensione, in termini di occupazione di memoria, del documento video, risulta quindi essere direttamente collegata alla sua qualità, alla risoluzione, alla durata e al tipo di compressione video utilizzata. Nel nostro caso l'applicazione di video streaming con la quale si deve interagire, prevede tre principali livelli di qualità all'interno dei quali sono raggruppati tutti i documenti presenti sul sistema. Questi tre livelli vengono identificati dalle etichette LOW, MEDIUM e HIGH. I formati dei video sono quelli più comunemente usati, e quindi anche supportati da JMF per il multimedia streaming.

4.3.2 Organizzazione interna della cache

Abbiamo appena visto, nel paragrafo precedente, quale potrebbe essere la dimensione degli oggetti multimediali da gestire. Una prima considerazione da effettuare riguarda, quindi, la dimensione della memoria

cache. È necessario che questa possa contenere un certo numero di documenti, in modo da ridurre la frequenza delle operazioni di rimozione e sostituzione. Anche se questo ci porta a dover gestire una memoria di notevoli dimensioni e a riservarle questo spazio sul proxy.

Avremo, sul supporto di memoria permanente dell'entità proxy, una directory riservata alla memoria cache. È all'interno di questo direttorio che archivieremo i video. Tenendo presente le caratteristiche evidenziate nel precedente paragrafo si è pensato di avere, all'interno della directory appena citata, altre tre sottodirectory. Ognuna di esse farà riferimento ad uno dei tre differenti livelli di qualità introdotti per i contenuti video da memorizzare: low, medium e high. Ogni sottodirectory risulterà essere gestita come una cache indipendente dalle altre, avente una sua dimensione massima ed un proprio insieme di dati nel quale scegliere, eventualmente, quello o quelli da eliminare. Questa visione della cache ha delle caratteristiche comuni con la strategia *Partitioned Caching* presentata all'interno del secondo capitolo. Continuando a considerare la qualità come parametro principale, si potrebbe pensare di considerare anche la dimensione degli oggetti multimediali. Ogni cache, relativa ad un singolo livello di qualità, potrebbe essere suddivisa in *n* sottocartelle, associate a diversi intervalli di dimensioni. Questo consentirebbe di avere prestazioni migliori nella gestione di questi insiemi, in quanto costituiti da oggetti più omogenei relativamente all'occupazione di memoria. All'interno di questo progetto non verrà tenuta in considerazione questa ulteriore suddivisione della cache.

4.3.3 Memorizzazione dei video

Il flusso multimediale da memorizzare arriva in ingresso al proxy e viene girato in uscita sull'entità successiva della gerarchia. Questo flusso viene intercettato e indirizzato anche verso un file locale all'interno della directory di cache. In particolare all'interno di una quarta sottodirectory del direttorio di cache. Questa ulteriore sottodirectory viene introdotta per consentire la temporanea memorizzazione dell'oggetto. Al termine del

salvataggio, l'oggetto considerato, verrà poi trasferito nel direttorio relativo alla propria qualità caratteristica.

Dal momento che trattiamo oggetti la cui durata può essere considerevole, bisogna tenere presente che, il client, potrebbe interrompere in maniera definitiva la riproduzione del video. L'interruzione può avvenire anche nel caso in cui non sia stato raggiunto il termine del video. In questo caso viene mantenuta in memoria la parte di video che è stata già visionata dal client. Non viene scaricata dal server originario la parte rimanente del filmato per non sovraccaricare il server ed aumentare in maniera considerevole il traffico della rete. Anche perché si tratterebbe di effettuare questa operazione, con la possibilità che, il determinato filmato, venga poi eliminato dalla memoria cache senza che sia mai stato visionato completamente.

È nel momento del trasferimento dell'oggetto dal direttorio temporaneo a quello definitivo che saranno, eventualmente, chiamate in causa le politiche di gestione della memoria. Anche perché, solo al termine del salvataggio è conosciuta la dimensione effettiva dell'oggetto da memorizzare. Essendo possibile interrompere, da parte del client, la fruizione del materiale multimediale, è possibile che la dimensione effettiva non corrisponda con quella originaria.

4.3.4 Politiche di rimpiazzamento

Richiamando le politiche descritte nel secondo capitolo, possiamo vedere quante possono essere le scelte possibili, tenendo presente che ne sono state presentate solamente alcune. Abbiamo evidenziato come le differenti politiche privilegino differenti aspetti che caratterizzano gli oggetti e le azioni compiute su di essi. Queste considerazioni ci portano ad individuare l'effettiva difficoltà di selezionare una politica in quanto migliore rispetto alle altre.

Privilegiare un aspetto piuttosto che un altro può incidere sulle prestazioni del sistema, in termini, principalmente, di carico di lavoro sull'entità proxy. In particolare, potrei avere un utilizzo con frequenze elevate delle politiche di rimpiazzamento. Questo carico di lavoro può

gravare in maniera preponderante sul proxy, specialmente nel caso vi siano numerosi altri flussi multimediali transitanti sull'entità.

Per questi motivi si è scelto di implementare più di una politica, rendendo così possibile un minimo adattamento del sistema all'ambiente in cui è chiamato ad operare.

4.3.5 Opzioni di personalizzazione

Dall'analisi dei vari aspetti effettuata nei paragrafi precedenti possiamo notare come siano numerosi i parametri su cui è possibile agire. Per questo motivo si è pensato di rendere possibile al gestore dell'entità proxy una personalizzazione di alcuni di essi.

La prima opzione che viene presa in considerazione, è la scelta effettuabile riguardo la politica di rimpiazzamento della cache. La scelta potrà essere effettuata tra le strategie proposte.

Sarà possibile, poi, definire la dimensione massima delle tre directory destinate a contenere gli oggetti multimediali. In questo modo possiamo adattare le dimensioni della memoria cache in base alle dimensioni dei più diffusi elementi video presenti nel sistema.

Un ultimo parametro modificabile è quello relativo alla directory in cui è contenuta la cache. Possiamo, in particolare, indicarne il percorso ed il nome. Nel qual caso il percorso e il direttorio indicato non esistano verranno creati.

L'obbiettivo di queste opzioni di personalizzazione risulta essere principalmente quello di consentire un adattamento del sistema di caching all'ambiente entro il quale questo sarà operativo. Tutto questo per cercare di ottenere un miglioramento generale delle prestazioni.

4.4 Progettazione

All'interno di questo paragrafo andremo ad illustrare, attraverso le scelte effettuate, la fase di progettazione del sistema di caching.

Il sistema da realizzare deve essere inteso come il sistema che si occupa di operare la memorizzazione del flusso multimediale all'interno dell'entità proxy. In particolare entra in funzione nel momento in cui lo stream di dati arriva sul proxy per essere inoltrato al successivo nodo presente sul cammino. Lo scopo di tale memorizzazione è quello di avvicinare i contenuti multimediali ai client, appartenenti all'ambiente, che ne fanno richiesta.

Andiamo ora a vedere la struttura complessiva del sistema, osservando quali risultano essere i componenti più significativi.

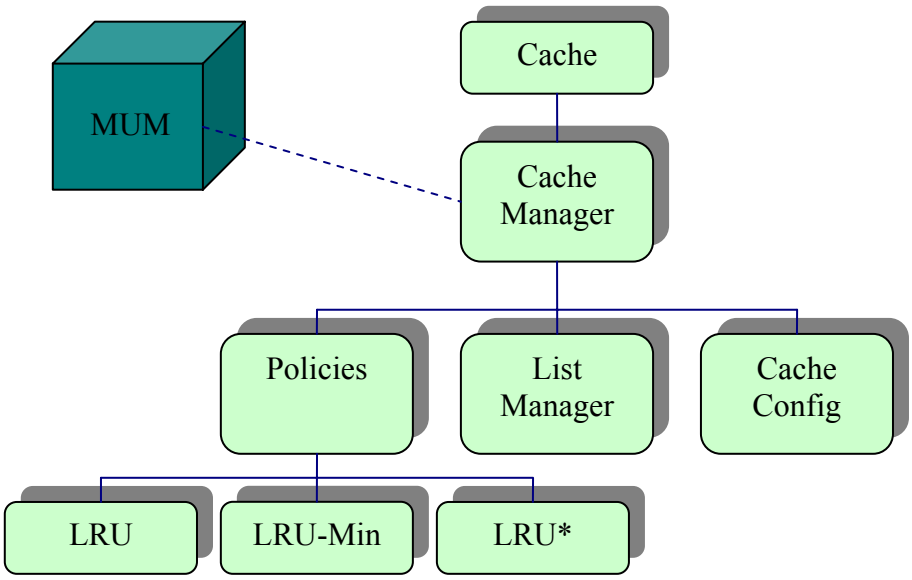


Figura 4.1: diagramma a blocchi del sistema di caching.

Come evidenziato dai vari blocchi contenuti nella figura qui sopra, il sistema risulta composto da diversi moduli. Andremo ora ad illustrare i componenti così individuati.

4.4.1 La cache

La cache è mantenuta sul dispositivo di memorizzazione permanente presente sul proxy che appartiene al cammino considerato. Come individuato nella fase di analisi, questa consta di una directory principale all'interno della quale sono presenti quattro sottodirectory.

Una di queste è relativa alla memorizzazione temporanea del flusso multimediale, mentre le altre tre corrispondono ai tre livelli di qualità presi in considerazione per i documenti video da trattare.

4.4.2 Il gestore del sistema

Il componente che si occupa della gestione del sistema è essenzialmente individuato dalla classe *CacheManager*. Non dobbiamo, infatti, occuparci della effettiva realizzazione della cache, ma della sua gestione. Questa è l'entità che fa da punto di contatto con l'ambiente MUM, essendo chiamata in causa dal metodo che si occupa, sul proxy, della ricezione del documento multimediale. In particolare riceverà dal suddetto metodo le informazioni relative al video ed il *data source* che fa riferimento al documento vero e proprio.

Le prime azioni che compie questa struttura, al momento della sua inizializzazione, riguardano il reperimento delle informazioni relative alla memoria da gestire. Innanzitutto vengono individuati i parametri di personalizzazione della cache. Successivamente, avviene il reperimento delle informazioni relative agli oggetti residenti all'interno della memoria cache.

Una fase successiva riguarda le operazioni necessarie alla configurazione del componente *processor* che verrà utilizzato. In particolare bisogna intercettare il flusso di dati entrante, indicato dal data source sopra citato, e indirizzarlo su di un file creato all'interno del direttorio temporaneo di memorizzazione. Dopo aver configurato adeguatamente il processor, prima di iniziare l'effettiva fase di ricezione e salvataggio dei dati, è necessario introdurre un ulteriore componente chiamato *data sink*. Questa entità lavora parallelamente al processor ed è quella che si occupa della effettiva scrittura del file. Dopo averla creata,

data sink e processor vengono avviati in successione, dando inizio al trasferimento dei dati. Entrambi questi componenti vengono fermati e chiusi quando l'utente dal lato client chiude l'interfaccia che gli permetteva la visione del video. In questo modo viene effettuata l'interruzione del flusso di dati e quindi la chiusura del file posto nel direttorio temporaneo.

Al termine di queste operazioni, il file ottenuto avrà tutte le caratteristiche del file che verrà mantenuto in memoria, salvo il fatto di trovarsi nel direttorio errato.

Successivamente a queste operazioni di chiusura, deve essere effettuato il trasferimento del file nel direttorio corrispondente al proprio livello di qualità. È a questo punto che vengono chiamate in causa le politiche di gestione della nostra memoria cache.

Vediamo ora la rappresentazione Uml della classe CacheManager, analizzando brevemente i metodi introdotti.

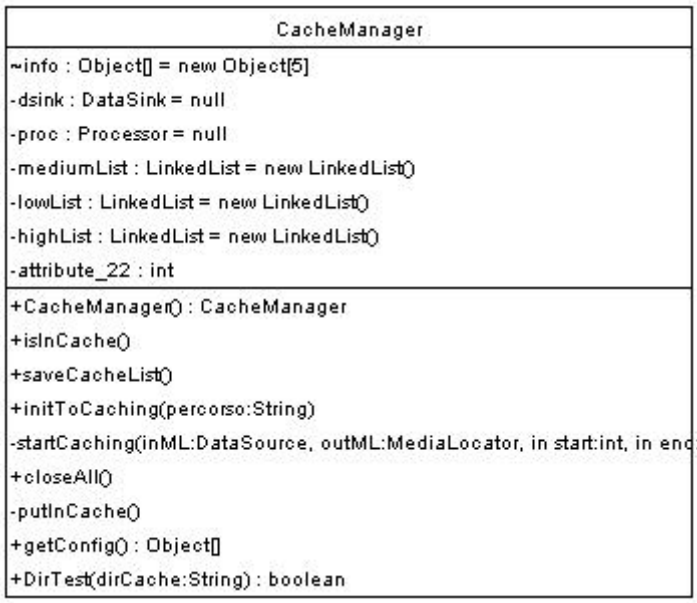


Figura 4.2: classe CacheManager.

I metodi principali della classe rappresentata in figura:

- *CacheManager*: è il costruttore della classe e si occupa del reperimento delle informazioni di configurazione e delle informazioni attuali riguardanti il contenuto della cache;
- *isInCache*: verifica la presenza nella cache di un determinato oggetto;
- *saveCacheList*: memorizza le liste contenenti le informazioni relative ai dati presenti in cache, nei relativi file;
- *initToCaching*: configura ed inizializza l'elemento processor, necessario per l'intercettazione del flusso multimediale;
- *startCaching*: crea ed inizializza il componente data sink; in seguito avvia la fase di memorizzazione del flusso intercettato;
- *closeAll*: pone termine alla memorizzazione, chiudendo i componenti coinvolti, come processor e data sink; avvia la fase di memorizzazione nella cache del file multimediale;
- *putInCache*: grazie a parametri quali la dimensione effettiva dell'oggetto e la sua qualità, ne permette l'inserimento nella relativa lista ed il successivo trasferimento del file nella locazione corretta;
- *getConfig*: richiama i parametri di configurazione della cache memorizzati nell'opportuno file;
- *dirTest*: verifica l'esistenza del direttorio di cache indicato nel file di configurazione; se e directory non esistono vengono create.

4.4.3 La configurazione della cache

Una delle caratteristiche di questo sistema di gestione, è rappresentata dalla possibilità di essere configurato in alcuni suoi parametri. Questa fase di personalizzazione avviene tramite l'utilizzo di un'interfaccia grafica.

Come indicato in precedenza è possibile impostare alcune opzioni del sistema: la politica di gestione della memoria, la dimensione di ognuna delle singole sottodirectory e la directory principale della cache.

I parametri prescelti saranno memorizzati all'interno di un file di configurazione. Quando viene effettuata la configurazione del sistema vengono richiamati i parametri che attualmente lo caratterizzano, i quali risiedono nel file appena citato. Se questi non risultano reperibili, vengono visualizzati valori di default successivamente modificabili.

Il file di configurazione verrà poi letto dal gestore della cache, il componente CacheManager, in modo da conoscere i principali parametri relativi alla memoria da gestire.

Nella figura seguente vediamo illustrata la rappresentazione Uml delle classi coinvolte nel processo di configurazione.

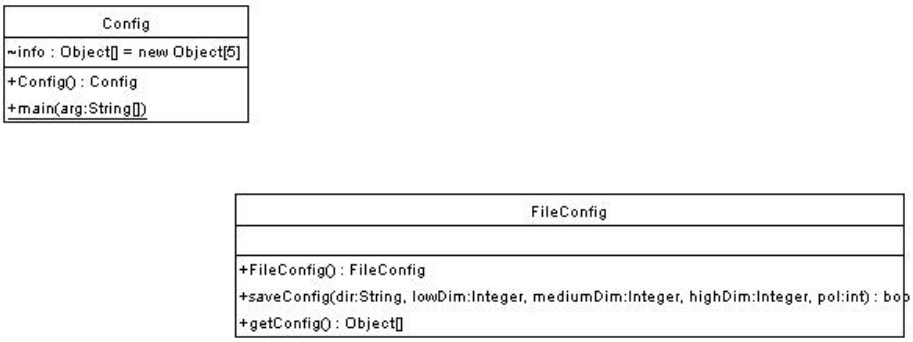


Figura 4.3: classi Config e FileConfig.

Senza entrare nel dettaglio dei metodi relativi a queste due classi, possiamo dire che la classe *Config* rappresenta l'interfaccia grafica mediante la quale si impostano i parametri di personalizzazione della memoria. La classe FileConfig, invece, fornisce i metodi che consentono di salvare tali parametri nel file di configurazione e di reperire le stesse informazioni di personalizzazione dal medesimo file.

4.4.4 Le politiche di gestione

L'organizzazione interna della memoria cache richiama le caratteristiche descritte all'interno del secondo capitolo e riguardanti la strategia *Partitioned Caching*.

Le politiche di gestione considerate agiranno in base al contenuto delle tre sottodirectory descritte in precedenza. Ogni azione verrà descritta riferendosi ad una singola directory, tenendo presente che tutte e tre verranno gestite allo stesso modo.

Riferendoci, ad esempio, alla sottodirectory "Medium" prendiamo in considerazione alcuni principali aspetti. Si ritiene opportuno avere un file che contenga i dati relativi alle presentazioni contenute nella cache. Al momento dell'inizializzazione del gestore della cache, queste informazioni verranno caricate all'interno di una lista ordinata. Questa lista sarà poi utilizzata per le effettive operazioni di gestione della memoria.

Nella scelta delle politiche da implementare si è voluto privilegiare il concetto di località temporale. Questo perché è considerato essere un aspetto rilevante nella gestione di memorie di questa tipologia e per la caratteristica velocità di gestione di queste politiche. Per velocità caratteristica viene inteso il fatto che le operazioni compiute su di un oggetto memorizzato o da memorizzare, non comportano complesse elaborazioni. Andiamo ora ad analizzare le tre politiche prescelte per la successiva fase di implementazione.

LRU

Come prima politica andiamo ad occuparci della politica più semplice che prende in considerazione esclusivamente la località temporale. Considerando la lista delle presentazioni presenti all'interno della memoria cache, ci comportiamo nel modo seguente. In caso di cache hit, l'oggetto richiesto viene individuato, rimosso dalla lista e reinserito in coda alla medesima. Questo per fare in modo che il video richiesto per ultimo sia sempre posizionato in coda alla lista. Qualora si presenti la necessità di memorizzare un oggetto la cui dimensione risulta essere maggiore dello spazio attualmente libero in memoria, si opera nel modo seguente. Vengono rimossi uno o più oggetti, prelevandoli dalla testa della lista, fino a che non è stato liberato spazio sufficiente o fino che la lista non risulti vuota.

Durante ogni operazione di rimozione e/o inserimento viene aggiornato il valore relativo allo spazio occupato dagli oggetti mantenuti in cache.

Nella figura successiva illustriamo la rappresentazione Uml della classe LRUpolicy, che realizza la politica, evidenziando il fatto che implementa un'interfaccia comune a tutte le politiche.

I nomi dei metodi della classe spiegano intuitivamente cosa facciano; in ogni modo li presentiamo brevemente:

- *hit*: opera le opportune modifiche alla lista degli elementi, in seguito al verificarsi di un cache hit;
- *replacement*: realizza l'inserimento di un nuovo elemento nella lista di quelli già presenti in cache, effettuando le opportune rimozioni necessarie per liberare spazio; modifica opportunamente il valore che indica lo spazio utilizzato dagli oggetti memorizzati;

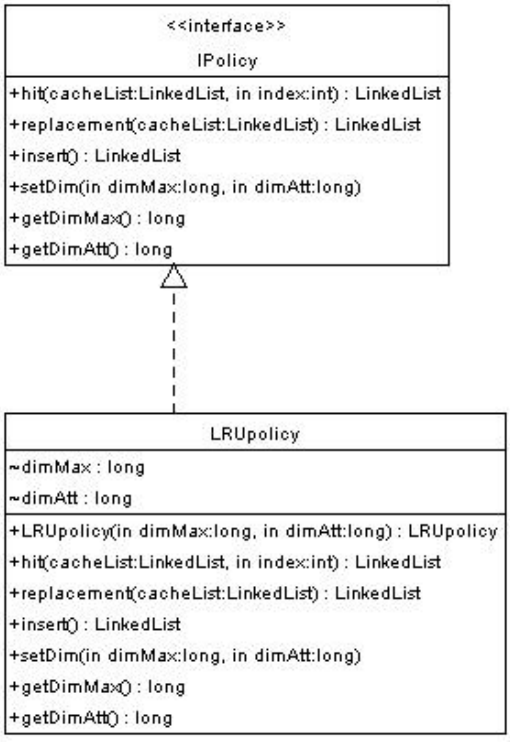


Figura 4.4: classe LRUpolicy e interfaccia IPolicy.

- *insert*: realizza l'effettivo inserimento di un nuovo oggetto nella lista; questo senza fare alcun controllo, ma modificando in maniera opportuna la dimensione attuale degli elementi in cache;
- *setDim*: imposta inizialmente i valori relativi alla dimensione massima della cache ed alla dimensione attuale occupata dagli elementi presenti nella cache;
- *getDimMax* e *getDimAtt*: restituiscono rispettivamente il valore relativo alla dimensione massima della cache e quello relativo allo spazio attualmente occupato.

LRU - Min

Questa politica introduce l'utilizzo di una lista supplementare e di un parametro *T* inizialmente impostato con la dimensione dell'oggetto che vogliamo memorizzare nella cache. La lista principale viene gestita, nel caso di cache hit, esattamente come descritto in precedenza per quanto riguarda la politica LRU. Quando si presenta la necessità di rimuovere uno o più elementi, vengono caricati sulla lista supplementare, da quella principale, solo gli elementi avente dimensione uguale o maggiore a *T*.

Se la lista è vuota si modifica il valore di *T* assegnandogli il valore *T/2* e si ripete la creazione della lista supplementare. Utilizzando la lista creata, rimuovo l'oggetto o gli oggetti dalla testa della medesima, fino a che non ho una quantità di spazio sufficiente o fino a che la lista non risulta vuota. Se ottengo una lista vuota, e non ho liberato spazio sufficiente, opero come appena descritto, modificando nuovamente il valore di *T* e ricostruendo la lista di appoggio. Nel momento in cui un oggetto viene eliminato dalla lista supplementare, questo viene eliminato anche dalla lista principale, in modo da avere una lista principale aggiornata e ordinata.

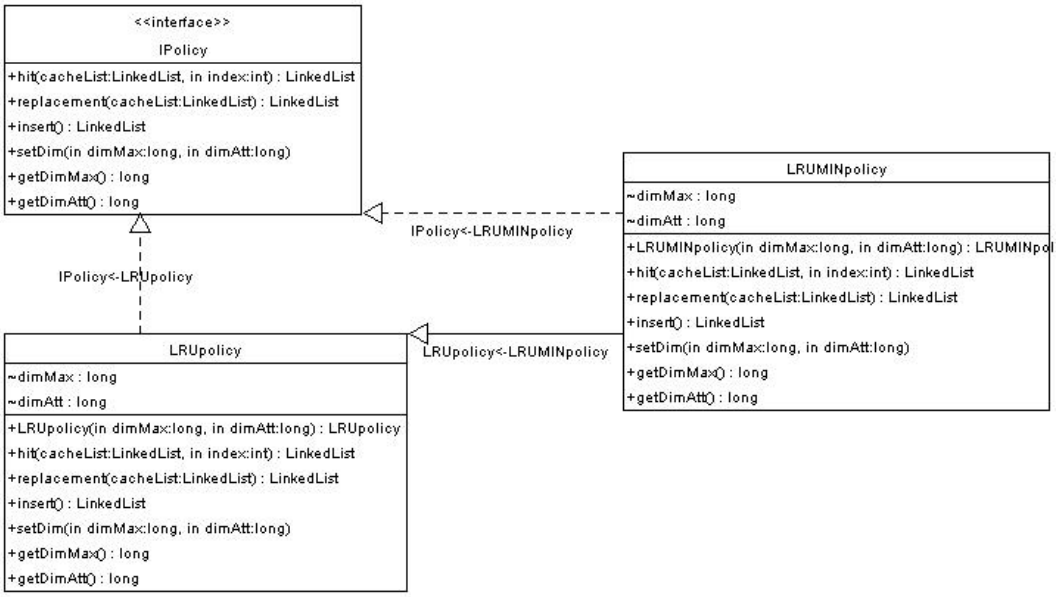


Figura 4.6: classe LRUMINpolicy e relazioni con LRUpolicy e IPolicy.

I metodi della classe LRUMINpolicy sono gli stessi della classe LRUpolicy e ne realizzano le medesime funzioni illustrate in precedenza. La particolarità di questa classe è il fatto che essa estende la classe LRUpolicy per poterne utilizzare il metodo hit. Questo metodo deve, infatti, compiere le stesse azioni per entrambe le politiche.

LRU*

Questa politica prevede l'utilizzo di un ulteriore parametro assegnato ad ognuno degli oggetti posti in cache. Il nuovo parametro introdotto è un contatore delle richieste di cui è stato fatto oggetto il documento presente in memoria. Quando si verifica un cache hit, viene incrementato il valore del contatore e l'oggetto viene posto in coda alla lista considerata.

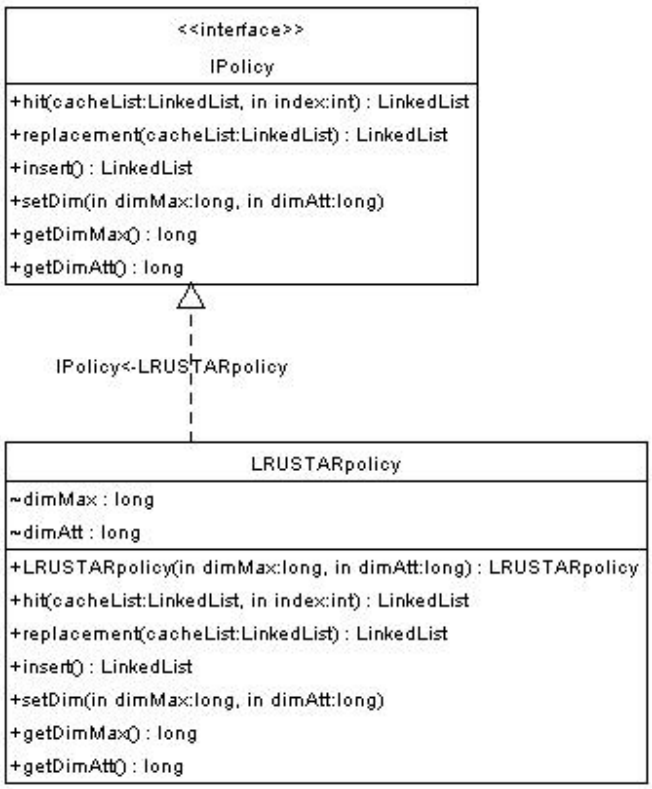


Figura 4.5: classe LRUSTARpolicy e interfaccia IPolicy.

Quando è necessario liberare spazio di memoria, viene preso in considerazione l'oggetto posizionato in testa alla lista. Se il suo contatore di richieste ha valore pari a zero, l'oggetto viene rimosso. Mentre, se il suo valore è diverso da zero, viene decrementato di una unità e l'oggetto viene rimosso dalla testa della lista e trasferito in coda alla medesima. Si prosegue in questo modo fino a che lo spazio di memoria liberato non sia sufficiente a contenere il documento prescelto o fino a che la lista degli elementi non risulti vuota.

Dalla figura posta nella pagina precedente possiamo notare come i metodi relativi a questa politica corrispondano esattamente a quelli visti per le due precedenti. Le funzioni che realizzano sono già state illustrate nelle parti di testo relative alle altre politiche. Le differenze rispetto ai metodi precedenti sono all'interno del corpo di ognuno di essi. Questi devono, infatti, gestire anche il contatore associato ad ognuno degli oggetti contenuti nella memoria cache.

4.4.5 La gestione delle liste

L'ultimo blocco che prendiamo in considerazione, tra quelli rappresentati nella figura relativa all'architettura del sistema di caching, è quello denominato List Manager. Questo componente si occupa di fornire i metodi necessari alla gestione delle liste di oggetti multimediali. Abbiamo, infatti, tre liste ordinate, una per ogni livello di qualità. Queste liste contengono le informazioni relative alle presentazioni contenute in memoria. Le operazioni di gestione della cache, con i relativi inserimenti e le relative rimozioni, vengono effettuate su queste liste e successivamente estese ai file memorizzati su disco. Questa classe offre i metodi che consentono tali operazioni.

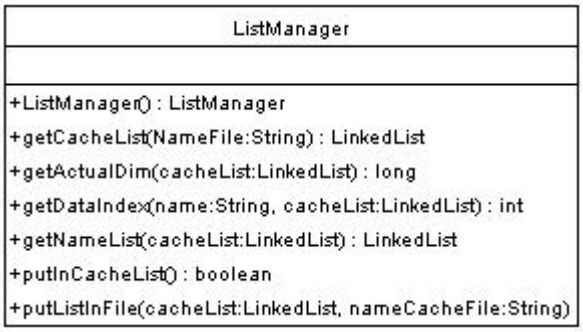


Figura 4.6: classe ListManager.

La classe ListManager implementa i seguenti metodi (Figura 4.6):

- *getCacheList*: questo metodo preleva, dal relativo file, le informazioni riguardanti gli oggetti contenuti nella cache e costruisce una lista ordinata con questi dati;
- *getActualDim*: calcola la dimensione totale degli oggetti contenuti nella lista che riceve come parametro di ingresso;
- *getDataIndex*: restituisce un intero relativo alla posizione di un determinato oggetto all'interno della lista considerata, restituendo

un valore errato (-1) se la lista è vuota o l'elemento non è presente;

- *getNameList*: questo metodo costruisce una lista contenente solo i nomi delle presentazioni contenute nella lista che il metodo riceve in ingresso;
- *putInCacheList*: realizza l'effettivo inserimento del nuovo oggetto nella lista contenente gli elementi presenti nella cache. In base alla dimensioni dell'oggetto ed allo spazio libero in memoria viene inserito l'oggetto o richiamata la relativa politica di rimpiazzamento.

4.5 Conclusioni

In questo capitolo abbiamo affrontato l'analisi del problema di cui si deve proporre una soluzione. Si è cercato di introdurre i principali requisiti richiesti dal sistema di caching, per poi affrontare, nella fase di analisi, le scelte effettuate per la gestione del sistema stesso.

La fase di analisi ci ha portano alla vera e propria progettazione del sistema di caching. In questa parte si è cercato di spiegare l'architettura del sistema e le caratteristiche degli elementi che lo compongono, fino ad arrivare alla definizione di classi e metodi. Queste considerazioni conducono la nostra trattazione verso l'implementazione effettiva del sistema, che verrà trattata nel capitolo successivo.

CAPITOLO 5

5 Implementazione e testing del sistema di caching

In questo capitolo andremo ad illustrare alcuni aspetti particolarmente significativi dell'implementazione del sistema di caching il cui progetto è stato presentato nel capitolo precedente. Nella seconda parte del capitolo, andremo a riportare alcuni valori relativi ai test effettuati sul sistema. In questi test confronteremo prestazioni ottenute, in termini di velocità, dalle tre replacement policy implementate, in diverse condizioni operative.

5.1 Implementazione del sistema di caching

Alcune delle scelte relative a questo specifico sistema di caching, sono state dettate dai vincoli imposti dall'ambiente entro il quale il nostro sistema deve andare ad integrarsi. In particolare, l'implementazione è stata realizzata cercando di sfruttare le potenzialità del linguaggio Java, utilizzato per la realizzazione dell'ambiente MUM. Conseguenza di questo è stato anche l'utilizzo del framework JMF, per la gestione dei contenuti multimediali.

Nelle parti che seguono si è cercato di operare seguendo due principali linee guida di esposizione. Per quanto riguarda la gestione dei contenuti multimediali, o le soluzioni specifiche legate ad un linguaggio, abbiamo fatto riferimento alle strutture offerte dal linguaggio Java utilizzato. Mentre, per quanto riguarda le parti relative all'implementazione delle politiche, si è preferito utilizzare uno "pseudo-linguaggio". Questo per non vincolare tali soluzioni operative ad un particolare linguaggio di programmazione.

5.1.1 Gestione dei contenuti multimediali

In questa sezione descriviamo i componenti coinvolti nella gestione dei contenuti multimediali, facendo quindi espliciti riferimenti al linguaggio Java ed in particolare al framework JMF. I package principalmente coinvolti in questa fase di implementazione sono il *javax.media* ed il *javax.media.protocol*.

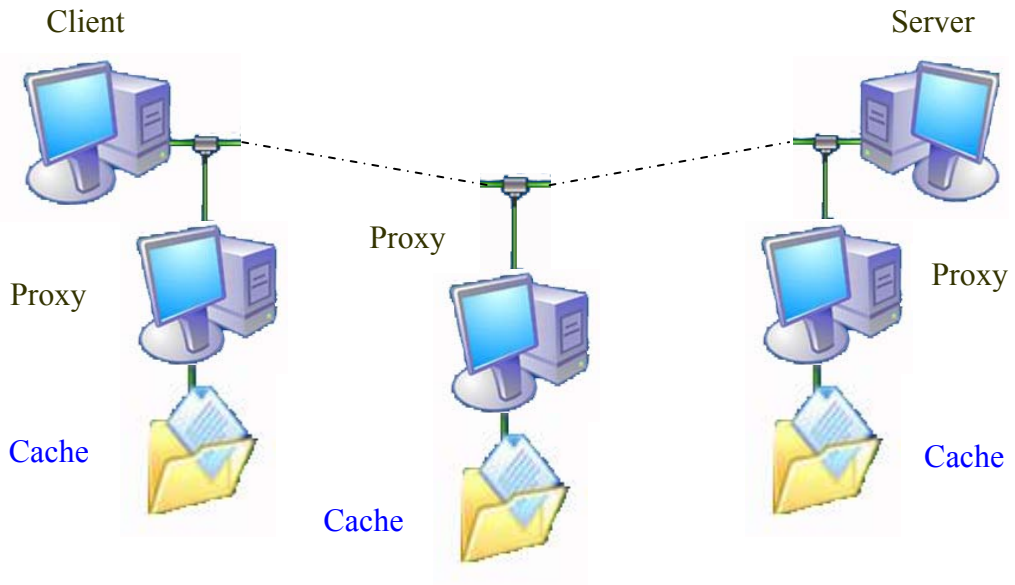


Figura 5.1: modello Client - Proxy - Server.

Prima di approfondire l'argomento riguardante la gestione degli stream multimediali si ritiene opportuno ricordare quale sia lo scenario in cui il sistema di caching andrà ad operare (Figura 5.1). L'entità client richiederà una determinata presentazione al relativo server, che provvederà all'invio dei dati. Queste informazioni transiteranno attraverso una o più entità proxy, che avranno la possibilità di memorizzare questo flusso all'interno delle rispettive memorie cache.

Il primo problema che abbiamo dovuto affrontare è stato quello relativo all'intercettazione di questo flusso multimediale transitante

attraverso l'entità proxy, e la sua redirezione verso un file creato sul disco locale. Il flusso è stato intercettato sfruttando una caratteristica fondamentale del componente data source: la possibilità di essere clonato. È stato creato, infatti, un clone del data source utilizzato nell'entità proxy per inoltrare i dati all'entità successiva del percorso. La riga di codice che segue riguarda la clonazione del data source e *clonedDataSource* indica il data source clonabile da cui il clone viene creato:

```
...
DataSource inML=((SourceCloneable)clonedDataSource).createClone();
...
```

Utilizzando questo clone viene creato il relativo processor, che viene poi configurato in maniera adeguata. Riportiamo la riga di codice relativa alla creazione del processor:

```
...
processor = Manager.createProcessor(inML);
...
```

dove *inML* indica il clone del data source originario. Successivamente si è resa necessaria l'introduzione di un ulteriore componente del framework: il *MediaLocator*. Questo componente, contenuto nel package *javax.media*, viene utilizzato per indicare il file destinazione nel quale il flusso verrà memorizzato. Successivamente a queste operazioni, bisogna occuparsi della creazione del data sink che svolge l'effettivo compito di scrittura dei dati all'interno del file destinazione. Per la creazione di questo componente è necessario sfruttare il processor (*processor*) ed il media locator (*outML*) creati in precedenza:

```
...
dsink = Manager.createDataSink(processor.getDataOutput(),outML);
...
```

Una volta inizializzati questi componenti è possibile avviare l'intercettazione e la relativa scrittura del contenuto multimediale transitante attraverso l'entità proxy. Questo viene realizzato mediante l'invocazione dei metodi *start* del processor e del data sink creati.

Questa fase di implementazione è stata caratterizzata da alcuni problemi iniziali relativi soprattutto al corretto utilizzo dei componenti descritti. Effettuando diverse prove di configurazione ed utilizzo di queste strutture, si è poi arrivati ad ottenere il risultato voluto.

5.1.2 Le politiche di gestione

Per quanto riguarda le replacement policies implementate, andiamo a presentare i principali metodi che caratterizzano ognuna di queste politiche. Per fare ciò ci avvaliamo dell'utilizzo di uno pseudo-linguaggio, in modo da non vincolare le operazioni realizzate al linguaggio Java.

LRU

Andiamo ora a vedere come sono state implementate le operazioni principali realizzate da questa prima politica. Come primo metodo andiamo a prendere in considerazione quello relativo alla situazione di cache hit. Tale metodo si occuperà di rimuovere il determinato oggetto, reperito grazie alla sua posizione, dalla relativa lista e di inserirlo in coda alla medesima. Viene restituita in uscita la lista ordinata. Vediamo ora nel dettaglio queste operazioni:

```
ListaOrdinata hit(ListaDiCache, posizione){

        oggetto = ListaDiCache.rimuovi(posizione);
        ListaDiCache.inserisciInCoda(oggetto);

        restituisci ListaDiCache;
}
```

Un altro metodo importante è quello relativo all'operazione di eliminazione e inserimento necessaria quando bisogna inserire un nuovo elemento all'interno della memoria cache e non si ha spazio libero sufficiente. Viene effettuato un primo controllo relativo alla dimensione dell'oggetto, per verificare che questa non sia maggiore della dimensione massima consentita alla memoria. Se risulta maggiore, la lista delle presentazioni non viene modificata. Se è minore od uguale, viene controllato se la presentazione da inserire è già presente nella lista. Se è presente, viene eliminata e viene liberato lo spazio da essa occupato. Se non è presente non viene eseguita alcuna operazione. Successivamente avviene la rimozione di oggetti fino a che si ottiene sufficiente spazio libero o fina a che la lista non risulta vuota. Dopodiché avviene l'inserimento del nuovo elemento all'interno della lista. Da questo metodo viene restituita la lista, che può essere stata, o meno, modificata. Andiamo a vedere le operazioni descritte:

```
ListaOrdinata replacement(ListaDiCache, nuovoDato){

  se (nuovoDato.dim > dimMax) allora restituisci ListaDiCache;
    altrimenti {
      se (nuovoDato.nome è in ListaDiCache) {
        oggetto = ListaDiCache.rimuovi(nuovoDato);
        libera spazio pari a oggetto.dim;
        }

      finchè ((spazio libero non sufficiente) e (ListaDiCache non
                                                        vuota)) {
          oggetto = ListaDiCache.rimuoviPrimo();
          libera spazio pari a oggetto.dim;
          }

      ListaDiCache.inserisciInCoda(nuovoDato);
      dimAtt = dimAtt + nuovoDato.dim;
      }
```

```
    restituisci ListaDiCache;
}
```

LRU-Min

Per illustrare i metodi relativi a questa seconda politica ci avvarremo di quelli già descritti per la politica precedente. Dal momento che il metodo hit rimane inalterato, non verrà trattato. Per quanto riguarda il metodo replacement, questo risulterà simile al precedente, anche se operante su di una lista diversa. Prima di applicare il metodo di rimpiazzamento descritto in precedenza, è necessario determinare la lista contenente le presentazioni aventi dimensione maggiore o uguale a quella del nuovo oggetto da inserire (il valore di soglia). Successivamente sarà applicato il metodo visto per la politica LRU. Se lo spazio liberato non risulterà sufficiente, verrà creata diversa lista di appoggio, modificando il precedente valore di soglia. Saranno poi ripetute le operazioni appena elencate. Andiamo ora ad illustrare le istruzioni relative all'operazione di creazione della lista di appoggio:

```
...
ListaOrdinata NuovaLista;

per ogni elemento di ListaDiCache ripeti {
                    oggetto = ListaDiCache.Elemento(i);

                    se (oggetto.dim >= soglia) allora {
                        NuovaLista.inserisciInCoda(oggetto);
                        }

                    }
...
```

LRU*

Per quanto riguarda i metodi relativi all'ultima politica implementata, riteniamo opportuno prendere in considerazione solamente le differenze presenti rispetto ai metodi relativi alla politica LRU. In particolare, ci troviamo qui a dover gestire un ulteriore parametro relativo ad ogni presentazione presente nella cache. Questo parametro, che indicheremo con il nome di contatore, viene inizializzato a zero nel momento della prima memorizzazione in cache del determinato oggetto. Quando si verifica una situazione di cache hit, l'elemento richiesto viene rimosso dalla lista, viene implementato il suo contatore e viene reinserito in coda alla lista. Come possiamo notare, l'unica operazione che diversifica questo metodo, rispetto a quello illustrato per la prima politica trattata, è l'incremento di tale contatore:

```
...
oggetto.contatore = oggetto.contatore + 1;
...
```

L'operazione di rimpiazzamento risulta, invece, più complessa. Rimuoviamo, innanzitutto, l'elemento posizionato in testa alla lista di cache. Dopodiché controlliamo il valore del suo contatore. Se è pari a zero, l'oggetto viene rimosso definitivamente e viene liberato il relativo spazio di memoria. Se il valore del contatore è maggiore di zero, viene decrementato di una unità e l'oggetto viene reinserito in coda alla lista. Queste operazioni vengono ripetute fino a che lo spazio libero non è sufficiente o fino a che la lista non risulta vuota. Evidenziamo ora, grazie all'utilizzo di istruzioni espresse mediante uno pseudo-linguaggio, le operazioni appena descritte:

```
...
oggetto = ListaDiCache.rimuoviPrimo();

se (oggetto.contatore = 0) allora {
        libera spazio pari a oggetto.dim;
        }
```

```
        altrimenti {
                oggetto.contatore = oggetto.contatore – 1;
                ListaDiCache.inserisciInCoda(nuovoDato);
                }
...
```

5.1.3 La gestione delle liste

Nella fase di implementazione si è reso necessario effettuare alcune scelte riguardanti le strutture da utilizzare. Una delle scelte più importanti, è stata quella relativa alla struttura da utilizzare per implementare, nel linguaggio Java, le liste ordinate. Liste adibite a contenere le informazioni relative alle presentazioni memorizzate nella cache.

Dopo aver preso in considerazione diverse soluzioni, si è optato per l'utilizzo della struttura *LinkedList*, contenuta nel package *java.util* del linguaggio Java. Questa struttura è caratterizzata dal fornire un ottimo accesso sequenziale e veloci operazioni di inserimento e rimozione dal centro della lista stessa. Offre inoltre metodi che consentono rimozioni ed inserimenti in testa ed in coda alla lista considerata. Di contro, non è la struttura che fornisce un più rapido accesso casuale ai dati in essa contenuti.

Avendo quindi optato per l'utilizzo di questa struttura, sono stati utilizzati i suoi metodi di base per effettuare la gestione delle varie liste coinvolte nella realizzazione del nostro sistema di caching.

5.1.4 La configurazione del sistema

Questa parte di implementazione è relativa alla fase di configurazione del sistema di caching. La personalizzazione del sistema viene realizzata mediante l'utilizzo di una semplice ed intuitiva interfaccia grafica. La finestra che si presenta all'utente è quella presentata nella seguente figura.

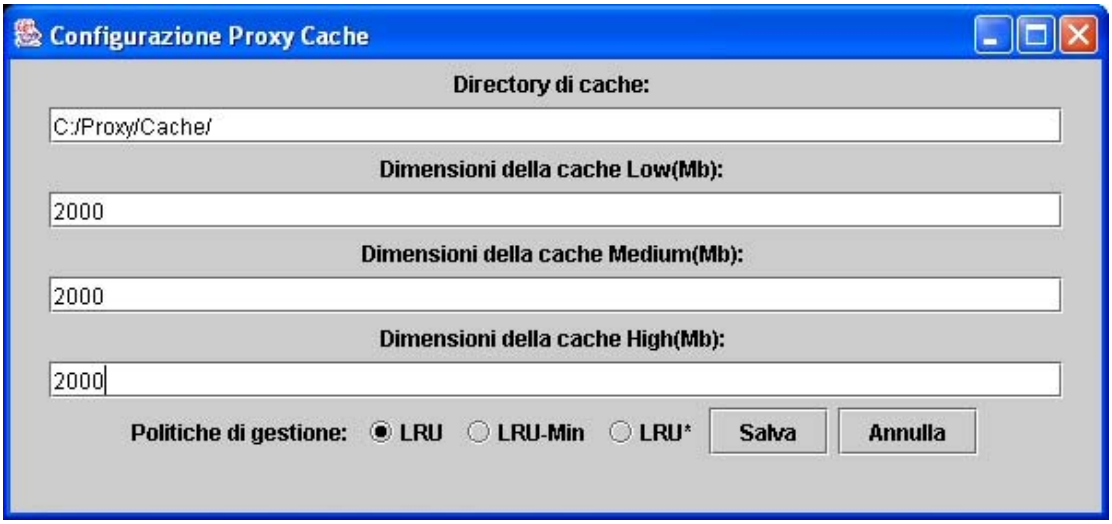


Figura 5.2: finestra di configurazione del sistema.

La classe che implementa questa interfaccia grafica risulta essere sottoclasse della classe base *JFrame*, da cui eredita i metodi. La classe JFrame è contenuta nel package *javax.swing* del linguaggio Java. La classe da noi creata implementa, inoltre, l'interfaccia *ActionListener*, contenuta nel package *java.awt.event* del linguaggio Java. Questa interfaccia permette di associare, all'azione di pressione dei pulsanti visualizzati, le relative operazioni di salvataggio dei dati o di chiusura della finestra.

5.2 Testing

L'ultima fase relativa alla realizzazione del sistema di caching, riguarda il testing del sistema stesso. Questa parte mira a definire le differenze relative alle prestazioni delle tre politiche di rimpiazzamento implementate, in differenti condizioni operative.

Per prima cosa è possibile notare che l'insieme delle tre politiche prende in considerazione, per ogni oggetto, tre parametri principali: la località temporale, l'occupazione di memoria e il numero di accessi. La strategia LRU prende in considerazione solamente la località temporale. La

strategia LRU-Min considera la località temporale e la dimensione degli oggetti. Infine, la strategia LRU*, rileva la località temporale e il numero di accessi relativi ad ogni oggetto contenuto nella memoria cache. Queste considerazioni ci hanno portato ad effettuare test di comparazione prendendo come punto di riferimento la strategia LRU, e confrontandola prima con quella LRU-Min e successivamente con quella LRU*. Le operazioni di inserimento, senza avere problemi di saturazione, e le operazioni conseguenti a situazioni di cache hit, si equivalgono per tutte le strategie. Abbiamo quindi considerato solamente il caso in cui sia necessario effettuare operazioni di replacement.

Per effettuare i test che seguono, è stata utilizzata, come entità proxy, una macchina Olidata così equipaggiata:

- processore Intel Pentium 4 2GHz;
- 256MB di memoria RAM;
- sistema operativo Windows XP Home Edition.

5.2.1 LRU vs LRU-Min

Se la memoria cache si trova in una situazione in cui gli oggetti in essa contenuti risultano di dimensioni omogenee, le prestazioni di queste due strategie si equivalgono. Con il termine prestazioni intendiamo sia il tempo necessario per portare a termine l'inserimento del nuovo oggetto, sia il numero di oggetti rimossi per effettuare questa operazione.

Più interessante è il caso in cui abbiamo, all'interno della memoria cache, numerosi oggetti di piccole dimensioni e pochi oggetti di dimensioni maggiori. I termini relativi alle dimensioni degli oggetti devono essere intesi in riferimento alla dimensione massima della memoria cache.

Abbiamo considerato la cache occupata per il 90% della sua dimensione massima. Occupazione così distribuita rispetto ai singoli file presenti in memoria:

- Presentazione1 = 5%;
- Presentazione2 = 3%;
- Presentazione3 = 5%;
- Presentazione4 = 10%;

- Presentazione5 = 5%;
- Presentazione6 = 30%;
- Presentazione7 = 5%;
- Presentazione8 = 2%;
- Presentazione9 = 6%;
- Presentazione10 = 10%;
- Presentazione11 = 9%.

Tali valori percentuali vanno sempre intesi relativamente alla dimensione massima della cache. Effettueremo ora l'inserimento nella memoria di file di diverse dimensioni, analizzando i comportamenti del sistema. I risultati sono riportati all'interno della tabella 5.1. Dal momento che i tempi relativi alle operazioni testate si sono mostrati molto simili fra loro abbiamo preso in considerazione il numero di oggetti rimossi. Questo è principalmente dovuto al non elevato numero di elementi presenti nella cache.

Dimensione della presentazione inserita (%)	**Numero di oggetti rimossi**	
	LRU	**LRU-Min**
60	6	3
45	6	2
40	6	1
20	3	1

Tabella 5.1: LRU vs LRU-Min.

Come è possibile notare dai risultati relativi ai test, la politica LRU-Min consente, in generale, un minor numero di rimozioni rispetto alla più semplice LRU. Risulta quindi più performante l'utilizzo della strategia che riduce il numero di eliminazioni di oggetti dalla memoria cache.

5.2.2 LRU vs LRU*

Per quanto riguarda il confronto fra queste due strategie, è possibile affermare che è sicuramente preferibile l'utilizzo della strategia LRU*. Questa strategia potrebbe richiedere, come è stato individuato nei test, un maggior numero di interazioni con la lista degli oggetti contenuti nella cache. Allo stesso tempo, però, consente il mantenimento in memoria di quelli richiesti un maggior numero di volte e in tempi abbastanza recenti. Con il passare del tempo, infatti, aumentano le possibilità che un oggetto ha di essere rimosso, anche se è stato precedentemente caratterizzato da un numero elevato di richieste. Le prestazioni della strategia LRU* dipendono in maniera fondamentale dalle richieste che vengono inoltrate al sistema.

Queste caratteristiche influenzano notevolmente, incrementandola, la percentuale delle richieste servite con successo da parte della memoria di cache. Argomento non affrontato nella realizzazione di questo sistema di caching.

I tempi relativi all'operazione di replacement sono risultati pressoché simili, e pertanto non si è ritenuto opportuno riportarli. Questi tempi, come quelli relativi alla politica LRU-Min, potrebbero essere significativi, qualora il numero degli oggetti presenti nella cache fosse molto elevato.

5.3 Conclusioni

All'interno di questo capitolo conclusivo sono state illustrate alcune delle scelte implementative effettuate per la realizzazione del sistema di caching. Sono stati riportati alcuni dei passaggi più significativi del codice prodotto, cercando di descrivere nel modo più comprensivo possibile le relative istruzioni.

È stato infine affrontata una fase di testing del sistema, effettuando alcuni test relativi alle strategie di rimozione implementate, evidenziandone i punti di forza. Si è visto, in particolare, come sia possibile privilegiare un aspetto rispetto ad un altro, scegliendo la strategia più appropriata. Scelta che consente una certa adattabilità del sistema.

Conclusioni

Nel corso di questa trattazione si è avuta la possibilità di constatare quale sia l'effettiva importanza delle tecnologie di caching rivolte al settore dei sistemi distribuiti multimediali. Questa importanza è principalmente dettata dalle tipiche caratteristiche dei contenuti multimediali, prime fra tutte, l'occupazione di memoria e la qualità. L'introduzione di entità intermedie, quali sono i proxy, consente un incremento delle prestazioni globali. Questo grazie alla possibilità di memorizzare i flussi che transitano attraverso di loro, provenienti dal server e diretti verso il client.

Per consentire la realizzazione di questo progetto è stato necessario coinvolgere diversi interessanti aspetti relativi sistemi di caching ed alla gestione di contenuti multimediali.

Nella fase iniziale del progetto, l'attenzione è stata rivolta principalmente verso le varie caratteristiche che bisogna prendere in considerazione, quali ad esempio, le tipologie di caching e i contenuti multimediali. Successivamente ci siamo concentrati sull'analisi di alcune politiche di rimozione, valutando i parametri da queste coinvolti. Si è subito inteso come fosse complicato, se non impossibile, individuare un'unica politica, migliore rispetto alle altre. Prendendo in considerazione le varie politiche siamo giunti ad individuare i parametri che, secondo noi, interessano maggiormente un ambiente in cui vengono diffusi contenuti multimediali. In seguito a queste motivazioni si è subito pensato alla possibilità di implementare più di una politica di rimozione. La panoramica sulle tecnologie ci ha consentito di approfondire la conoscenza relativa ai sistemi entro i quali il nostro progetto è andato ad integrarsi. In questo modo è stato possibile delineare quali fossero i principali aspetti da tenere in considerazione. Le tipiche caratteristiche degli oggetti multimediali hanno poi completato la nostra visione d'insieme. Queste considerazioni ci hanno portato a delineare una soluzione che prendesse in considerazione i tre livelli di qualità, proposti dall'ambiente MUM, per il materiale video. In particolare pensando ad una cache composta da tre "sottocache" identificanti le classi di qualità coinvolte. L'utilizzo delle strutture rese

disponibili dal linguaggio Java, ed in particolare, di quelle fornite dall'estensione Java Media Framework, ha consentito un'agevole gestione del materiale multimediale coinvolto. Dalle considerazioni relative alla scelta delle politiche da implementare, si è giunti alla conclusione che l'aspetto della località temporale, dovesse assumere un ruolo predominante. Oltre a questo aspetto, si è scelto di dare una certa importanza all'occupazione di memoria ed alla popolarità del documento multimediale considerato. Consentendo la configurazione di alcuni parametri relativi al sistema, lo si è reso, sotto certi aspetti, adattabile alle condizioni in cui dovrà operare.

Una successiva e finale fase di testing del sistema nel suo complesso, ha evidenziato le prestazioni delle tre politiche di sostituzione implementate. Questi risultati hanno evidenziato come sia possibile privilegiare un aspetto rispetto ad un altro, scegliendo una diversa strategia di gestione. È risultato evidente come, d'altra parte, sia improbabile riuscire ad indicare una strategia generalmente migliore. Viene così offerta la possibilità di adattare il sistema a diverse possibili condizioni operative.

Il progetto realizzato è andato a coinvolgere ampie problematiche, affrontabili sotto diversi aspetti ed in base a diversi parametri o alla diversa importanza ad essi assegnata. Una delle scelte progettuali è stata infatti quella di privilegiare la qualità del contenuto multimediale, rispetto alla sua occupazione di memoria. Un altro approccio al problema potrebbe considerare l'occupazione di memoria come aspetto primario. Come indicato in fase di analisi, si potrebbero anche coinvolgere entrambe queste caratteristiche. Ad esempio, si potrebbe effettuare una prima suddivisione del materiale rispetto alla sua qualità e successivamente un'ulteriore suddivisione in base all'occupazione di memoria.

Un ulteriore possibile sviluppo futuro riguarda la parte di configurazione. Questa potrebbe essere realizzata in modo che il funzionamento del sistema possa essere modificato run-time. Una soluzione, che consenta questa tipologia di configurazione, permetterebbe al sistema di adattarsi in maniera dinamica ai cambiamenti dell'ambiente in cui opera.

Bibliografia

[CACH] Albini M., "Infrastrutture ad Agenti Mobili per l'Accesso ad Informazioni Distribuite", Tesi di Laurea

[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

[DSYS] Coulouris G., Dollimore J., Kindberg T., "Distribuited systems: concepts and design", 3 ed., Addison Wesley, 2001

[MUM] Foschini L., Tesi di laurea, 2003
http://www.lia.deis.unibo.it/Research/MUM

[JMFPG] Java Media Framework, Programmers Guide:
http://java.sun.com/products/java-media/jmf/2.1.1/specdownload.html

[JMFHP] Java Media Framework Home Page:
http://java.sun.com/products/javamedia/jmf/

[SUR03] Podlipnig S., Böszörmenyi L., "A survey of Web cache replacement strategies", ACM Computing Surveys (CSUR) archive, Vol. 35, Issue 4, pages 374-398, December 2003

[OSYS] Stallings W., Operating systems: internals and design principles, Ed. Upper Saddle River, Prentice Hall, c2001