ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Seconda Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

MODELLI E TECNOLOGIE PER LA VISUALIZZAZIONE DI PROGRAMMI IN ESECUZIONE

Elaborata nel corso di: Sistemi Operativi

Tesi di Laurea di:
MASSIMO SMIRAGLIO

Relatore:
Prof. ALESSANDRO RICCI

Co-relatori:
Prof. ANDREA SANTI

ANNO ACCADEMICO 2012–2013
SESSIONE I

PAROLE CHIAVE

Visualizzazione
Esecuzione di Programmi
Strumenti
Monitoraggio

Alla mia famiglia che mi ha sempre dato supporto.
A Valeria che mi ha aspettato.
Ai miei nipoti.

Indice

Capitolo 1

Introduzione

1.1 Il software al giorno d'oggi

Il mondo è sempre più permeato di sistemi software; software che diventano via via più complessi per rispondere alle naturali esigenze di una società sempre più legata alla tecnologia in ogni aspetto della quotidianità. Ma proporzionalmente alla complessità aumentano anche le difficoltà per coloro che sviluppano il software. I problemi nascono dal fatto di aver a che fare con sistemi molto elaborati, spesso distribuiti geograficamente, ma soprattutto veloci, reattivi, performanti: il mondo va veloce, e i programmi devono tenere il ritmo. Questo significa che qualsiasi programma deve essere multi-processo o multithreading e implica quindi il dover prevedere ed eventualmente risolvere tutti gli aspetti che ne derivano; occuparsi quindi della gestione della concorrenza e delle tecniche di comunicazione tra i diversi threads evitando i "colli di bottiglia" diventa essenziale.

Nel corso degli anni il modo di programmare si è quindi dovuto evolvere per restare al passo con la complessità. Si sono così affinate le tecniche di astrazione concettuale con le quali gli sviluppatori possono immaginare in modo più naturale i programmi durante la fase di progettazione. Si sono evoluti i linguaggi che consentono la rappresentazione dei dati in forma concettuale e organizzabile secondo gerarchie. Sono state inventate tecniche per la generalizzazione del codice in modo tale da renderlo riutilizzabile e facilmente modificabile; in una parola: modulare.

Ma se da un lato questa evoluzione comporta notevoli benefici durante la progettazione e lo sviluppo, in termini di tempo, e fornisce strumenti po-

tenti nelle mani degli sviluppatori, dall'altro aggiunge un'ulteriore strato di complessità. Modularità, ereditarietà e polimorfismo, punti di forza dei linguaggi ad oggetti, comportano l'introduzione di un elevato numero di classi, che, unite all'utilizzo di pattern, utili a rendere indipendenti le varie parti del codice, contribuiscono a rendere il programma altamente estendibile e riutilizzabile, ma aumentano, allo stesso tempo, la difficoltà di analisi del sistema per chi deve carpirne o verificarne il funzionamento.

Se si pensa, inoltre, che il software impiega solitamente una varietà di librerie esterne e interagisce con componenti hardware e software distinti, ci si accorge che esaminare questi sistemi è un compito assai arduo.

Nonostante tutte queste difficoltà, è comunque necessario che **gli sviluppatori debbano comprendere i loro sistemi**.

1.2 Visualizzare l'Esecuzione di Programmi

Per uno sviluppatore, comprendere un software, vuol dire esaminare il modo in cui il programma è strutturato ed analizzare la sua esecuzione.

La struttura descrive l'organizzazione statica del sistema ed è, pertanto, molto importante nei casi in cui si voglia verificare la coerenza del sistema in relazione alla progettazione ed eventualmente apportare modifiche, o semplicemente per avere un punto di partenza nella comprensione del software.

L'analisi dell'esecuzione, invece, consente di esaminare il comportamento dinamico del sistema. Ci sono diverse motivazioni per cui può essere utile l'analisi del comportamento del sistema durante l'esecuzione. Lo scopo essenziale potrebbe essere quello di capire cosa sta succedendo nel sistema in un dato momento e come avviene. Un'eventuale analisi di questo tipo, quindi, potrebbe essere impiegata per diversi obiettivi:

- verificare il corretto funzionamento del sistema anche per esecuzioni in un elevato arco di tempo;
- trovare eventuali cause di comportamenti anomali o bug che vengono riscontrati occasionalmente e di difficile riproduzione;
- risolvere problemi di performance dovuti a eventuali protocolli di comunicazione, errori dovuti al sincronismo tra i diversi thread e eventuali lock inaspettati;

In termini più ampi, lo studio dell'esecuzione di un sistema può essere utile anche per estrapolare dall'implementazione, mediante un'analisi macroscopica, gli elementi progettuali come, ad esempio, le entità principali, diagramma delle interazioni e altro. Questo processo bottom-up, dunque, può essere utile per: risalire ad un modello di astrazione del sistema svincolato dalle particolari scelte implementative e similare al modello progettuale; ottenere informazioni macroscopiche riguardo il comportamento del sistema per verificare il divario in termini di astrazione tra i modelli di comportamento a livello di design e il codice sorgente, che potrebbe eventualmente nascere dal passaggio tra la fase progettazione e la realizzazione; facilitare la manutenzione e l'evoluzione del sistema mostrando quali conseguenze comportano le particolari modifiche dovute ad alterazioni del codice o semplicemente a variazioni di parametri (per esempio il numero di thread, o connessioni ecc. . .), rispettando la coerenza del sistema;

E' chiaro, dunque, quanto sia essenziale, per gli sviluppatori, poter comprendere struttura e comportamento del sistema al fine di effettuarne un'eventuale valutazione, correzione, variazione o innovazione. Per far ciò, però, è necessario che gli strumenti a loro disposizione si evolvano proporzionalmente all'aumento di complessità dei sistemi. Le tecniche già esistenti possono essere di grande utilità in particolari casi, ma diventano tremendamente insufficienti quando si tratta di lavorare su software estremamente ampi: milioni di linee di codice e migliaia di classi rendono questo tipo di analisi eccessivamente costoso in termini di tempo e oneroso in termini di difficoltà. Prendendo in considerazione, ad esempio, una tecnica come il Debugging, si nota subito che, oltre alle limitazioni già accennate, presenta un altro grande difetto: perturba l'esecuzione del sistema. Il Debugging, infatti, consente di bloccare ed analizzare l'esecuzione procedendo istruzione per istruzione; questo comportamento molto mirato può aiutare nella risoluzione di alcuni errori o malfunzionamenti, ma complessivamente è insoddisfacente in quanto, in un programma multithreading, agisce solo su un particolare thread, lasciando al contempo invariata l'esecuzione degli altri. Di conseguenza si ha la possibilità di effettuare un'analisi molto particolareggiata ma di un sistema che non rispecchia il funzionamento reale e non tiene in considerazione le effettive interazioni tra thread. La ricerca, dunque, si sta muovendo verso nuove direzioni al fine di realizzare strumenti il più efficaci possibile e adeguati all'analisi dei sistemi software.

Ciò di cui tratta questo elaborato, dunque, è una particolare categoria di questi innovativi strumenti, tutti relativi ad un'unica idea di fondo: **la visualizzazione dell'esecuzione dei programmi**. Si è pensato, infatti, che visualizzare, sotto forma di una qualche rappresentazione grafica, lo stato del sistema, le interazione dei vari componenti e ulteriori informazioni a differenti livelli di astrazione, possa agevolarne enormemente la comprensione.
Nei prossimi capitoli, dunque, si discuterà su quali siano gli obiettivi ed i requisiti di uno strumento di visualizzazione e si analizzeranno nel dettaglio alcune tecnologie già esistenti.

Capitolo 2

L'obiettivo di un programma di visualizzazione ideale

2.1 Descrivere il funzionamento del sistema

Nel precedente capitolo si è focalizzata l'attenzione sulle motivazioni che hanno portato i ricercatori alla progettazione e alla creazione dei programmi di visualizzazione. In questo capitolo, invece, approfondiremo gli aspetti riguardanti il "cosa" deve essere visualizzato all'utente.

Ciò che lo sviluppatore vorrebbe, quando analizza un programma, è la capacità di poter risalire allo schema concettuale che aveva in mente il progettista durante la realizzazione del sistema; in parole povere, lo sviluppatore è interessato al modello, ovvero una visione semplificata e più accessibile del sistema.

I principali punti di vista utili a descrivere un sistema software sono, dunque, i medesimi di quelli della fase di progettazione:

- struttura
- interazione
- comportamento

Questi tre punti di vista costituiscono tre indispensabili *dimensioni* in cui articolare la descrizione del sistema e consentono, dunque, di esprimerlo in modo concettuale astraendo dalle particolarità dell'implementazione e dal

linguaggio di programmazione utilizzato. L'utilità, per uno sviluppatore, di descrivere il sistema secondo questi modelli ricopre la maggior parte, se non la totalità, degli scenari: la necessità di comprendere un sistema che non si conosce; l'esigenza di verificare che il sistema che si è realizzato sia coerente con i modelli di progettazione; l'eventualità in cui si vogliano analizzare le performance, apportare modifiche o riscontrare ed eliminare comportamenti anomali. Nei prossimi paragrafi analizzeremo cosa si intenda per struttura, interazione e comportamento anche nell'ambito dei programmi di visualizzazione.

2.2 Struttura

Per un programma di visualizzazione, mostrare la struttura del sistema in esame potrebbe essere già un buon punto di partenza per agevolarne la comprensione riguardo il come sia costituito e quante e quali parti lo compongano.

La struttura di un sistema, infatti, è la rappresentazione sotto qualche forma grafica o comunque descrittiva dell'organizzazione del sistema suddivisa in parti.

La struttura descrive in primo luogo l'architettura complessiva del sistema per poi ridefinire ricorsivamente la struttura di ogni sottosistema e di ogni elemento.

Parlando dell'architettura di un sistema si fa riferimento ad un insieme di macro-parti in cui il sistema si articola, mettendo in evidenza le loro responsabilità, le relazioni e le interconnessioni. In parole povere, l'architettura rappresenta il sistema "ridotto all'osso", consentendo, comunque, di descriverne il funzionamento.

La struttura, dunque, delinea la conformazione del sistema spaziando da un livello astratto più generico possibile (l'architettura complessiva) fino ad uno studio capillare di ogni singolo elemento interconnesso con gli altri.

È importante, per un programma di visualizzazione che tenga conto della struttura, riuscire a ricostruire la conformazione del sistema a partire dall'implementazione; in particolare saper individuare e mettere in evidenza solo gli elementi significativi dal punto di vista strutturale, tralasciando i meccanicismi, e poter delineare lo stile architetturale relativo al sistema

preso in esame. A questo scopo è fondamentale ricostruire le interazioni che avvengono all'interno del sistema.

2.3 Interazioni

Se visualizzare la struttura di un sistema, per un programma di visualizzazione, può essere un obiettivo importante, mostrare come le diverse parti scambiano informazione tra loro può esserlo anche di più.

Un modello delle interazioni non è altro che uno schema che racchiude le informazioni riguardanti la comunicazione tra le varie parti del sistema.

Ricostruendo un modello delle interazioni è possibile non solo delineare il funzionamento del sistema, ma sopratutto è possibile individuare quali sono i componenti di maggior rilievo e le varie dipendenze che sussistono. In questo modo è possibile, quindi, selezionare i "punti caldi", escludendo tutto ciò che non è significativo a livello concettuale, permettendo non solo di poter estrapolare l'architettura ma anche descrivere le dinamiche temporali del sistema. L'analisi delle interazioni, inoltre, risulta essere di estrema importanza sopratutto per i sistemi concorrenti. Nei sistemi concorrenti, infatti, è fondamentale individuare le entità "attive" (processi o thread) della struttura e rappresentare le modalità con le quali esse interagiscono tra loro, sia per un'eventuale analisi delle performance, sia per una verifica del corretto funzionamento.

Infine, anche nel caso delle interazioni, come per la struttura, è possibile avere diversi livelli di astrazione che consentono di indagare più o meno a fondo le relazioni che regolano la collaborazione tra le entità in gioco ed avere così un quadro più o meno generico del funzionamento complessivo.

2.4 Comportamento

Con la struttura e le interazioni, dunque, si ha una buona conoscenza del sistema nel suo complesso e dei rapporti che sussistono tra le diverse entità. Lo studio del comportamento, infine, consente di descrivere in modo specifico la logica interna dei diversi elementi.

Come per la struttura e per le interazioni, anche la descrizione del comportamento avviene in modo concettuale e astratto attraverso la definizione

di diversi stati interni in cui una singola entità può trovarsi in un dato momento. Lo studio del comporamento, infatti, mostra l'evolversi di questi stati, quindi le transizioni di stato, in seguito ad eventi specifici provenienti dall'esterno per tutta la durata del ciclo di vita dell'oggetto. Ricostruendo i diagrammi del comportamento si da la possibilità all'utente di intuire il funzionamento particolare dei singoli elementi del sistema senza tuttavia essere costretto ad interpretarne l'implementazione.

2.5 L'importanza delle dimensioni e i problemi relativi ai programmi reali

Come già detto all'inizio di questo capitolo, struttura, interazione e comportamento, indipendentemente dal tipo di rappresentazione che si utilizza per descriverli, consentono una conoscenza più intuitiva, astratta e globale del sistema; un programma di visualizzazione che permettesse di risalire alla descrizione in termini di queste tre dimensioni consentirebbe uno studio del sistema non più legato ad una serie di istruzioni indipendenti, ma ad un modello di ragionamento più elevato ed astratto. Uno strumento di visualizzazione ideale dovrebbe, quindi, poter ricondurre esattamente agli schemi di progettazione, ovvero il modo ottimale di re-interpretare ed eventualmente modificare il sistema.

Purtroppo, però, questo risulta essere un compito arduo; tra le varie difficoltà che si riscontrano, una delle principali nasce dal fatto che ogni programma è realizzato seguendo un particolare paradigma di programmazione. Il paradigma computazionale di un sistema riguarda lo stile di programmazione con il quale un sistema viene progettato e quindi realizzato; esso differisce da sistema a sistema in relazione al tipo linguaggio utilizzato e dalle astrazioni che si prendono in considerazione al momento della progettazione. Ogni linguaggio di programmazione, infatti, è ispirato ad un particolare paradigma e fornisce, pertanto, un insieme di strumenti concettuali che definiscono il modo in cui il programmatore concepisce e percepisce il programma stesso. I linguaggi ad oggetti, ad esempio, si prestano ad un'astrazione che prevede entità concettualmente significative organizzate in gerarchie e interagenti tra loro; ciò non è valido, invece, per altri tipi di linguaggi come ad esempio i linguaggi procedurali o i linguaggi funzionali che si prestano, tuttavia, ad altri tipi di astrazione.

D'altro canto essere a conoscenza del particolare paradigma di programmazione col quale un software è stato realizzato può, in realtà, essere vantaggioso. A causa del particolare contesto implementativo, infatti, il progettista è automaticamente indirizzato verso determinate astrazioni che influenzano le modalità con cui vengono costruiti i modelli concettuali della struttura, dell'interazione e del comportamento; pertanto, per uno strumento di visualizzazione, conoscere il paradigma computazionale del sistema in esame potrebbe già essere un buon punto di partenza per la sua successiva analisi.

Capitolo 3

L'architettura di uno strumento di visualizzazione

3.1 Requisiti di uno strumento di visualizzazione

Finora si è discusso del perché sia necessario comprendere il sistema e quanto sia importante avere una descrizione di esso in termini di struttura, interazione e comportamento.

Come già detto questo può risultare un compito arduo ed è quindi facile supporre che siano necessari degli strumenti che automatizzino l'analisi della morfologia e del funzionamento delle varie parti del sistema. Tradurre questo in pratica non è altrettanto semplice.

Da quanto emerso nei precedenti capitoli, inoltre, è ormai chiaro il fatto che ci siano diversi *livelli di comprensione* utili in differenti ambiti. Lo sviluppatore può essere interessato a visualizzare il funzionamento secondo un modello semplice e astratto che eclissi i meccanicismi dovuti alla traduzione in codice per comprenderlo o per verificarne la coerenza; può voler capire cosa succede durante l'esecuzione in generale o in seguito ad eventi specifici; può voler individuare concretamente i problemi che si presentano.

Secondo *Leroux e Exton*[1] uno strumento di visualizzazione idoneo deve, infatti, rispettare tre requisiti fondamentali (quelli che vengono definiti i *pilastri*): astrazione, enfasi, rappresentazione e navigazione (Figura 3.1).

L'**astrazione** consente di evitare le limitazioni dell'implementazione e

facilita l'individuazione dei pro e dei contro dei vari compromessi a livello di design. L'astrazione tuttavia, deve essere ben calibrata: infatti, sia con un eccessivo filtraggio delle informazioni ma anche includendo troppi dettagli, si rischia di omettere o nascondere il reale funzionamento del sistema.

L'**enfasi** consente di osservare, anche mantenendo il medesimo livello di astrazione, le stesse informazioni focalizzando, però, l'attenzione su uno specifico aspetto, per mettere in luce i particolari fenomeni durante l'esecuzione.

Attraverso l'astrazione e l'enfasi si ha un filtraggio delle informazioni, ma per rendere ottimale uno strumento di visualizzazione è necessaria un'adeguata **rappresentazione**. Una rappresentazione appropriata consente una maggiore comprensione e aumenta l'interesse dell'utente verso lo strumento.

L'interazione tra questa rappresentazione e l'utente, infine, gioca un ruolo centrale nell'ambito della visualizzazione delle informazioni. La **navigazione** è, quindi, un elemento cruciale in ogni applicazione di visualizzazione e deve consentire all'utente di poter esplorare agevolmente le informazioni modificando il livello di complessità agendo su astrazione, rappresentazione ed enfasi.

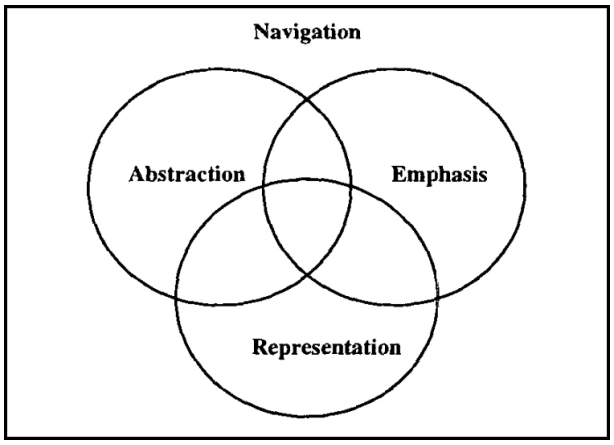

Figura 3.1: I pilastri di uno strumento di visualizzazione

In base all'obiettivo preposto, ed anche in base alla tipologia di sistema in oggetto, è necessario, quindi, scegliere quali dati siano importanti ai fini dell'analisi e quale rappresentazione è più adatta allo scopo.

Un buon strumento di visualizzazione, inoltre, deve essere di facile utilizzo da parte dell'utente; essendo strumenti di supporto non devono ostacolare il programmatore, al contrario devono essere intuitivi e flessibili per incoraggiarne un utilizzo efficace.

3.2 Fasi degli strumenti di visualizzazione

Possiamo, dunque, distinguere tre fasi principali che contraddistinguono il processo di funzionamento degli strumenti di visualizzazione:

- La raccolta dei dati è il primo nonché importante passo per la visualizzazione dell'esecuzione di un programma. Gli strumenti di visualizzazione oggi esistenti utilizzano, spesso, differenti tecniche per il collezionamento delle informazioni; questa moltitudine di metodologie è spiegata dal fatto che questa fase è definibile come la più "pericolosa". La raccolta dei dati, infatti, può essere un'operazione che si prolunga per tutta la durata dell'esecuzione (che quindi può essere molto estesa) e deve relazionarsi con enormi quantitativi di informazioni che devono essere gestite e organizzate nel modo più opportuno. Secondo *Reiss*[3], infatti, il requisito fondamentale di questa fase sta nel *perturbare il meno possibile il sistema*: questo è necessario se si vuole effettuare una corretta analisi del software che ne rispecchia in pieno il reale funzionamento. Se la raccolta dei dati influisce pesantemente sull'esecuzione si rischia di alterarne il comportamento e quindi condizionare le informazioni e l'analisi, soprattutto nel caso in cui la visualizzazione dell'esecuzione deve avvenire parallelamente all'esecuzione stessa. La raccolta dei dati deve essere effettuata in base al livello di *astrazione* e l'*enfasi* dell'analisi che si vuole effettuare; è necessario, quindi, compiere una scelta sulle informazioni che si vogliono memorizzare per rendere questo processo non solo ottimale ma anche il più efficiente possibile. Alcuni strumenti consentono l'interazione dell'utente anche in questa fase: è lo sviluppatore che specifica quali parti del codice monitorare. Altri strumenti più generici, invece, raccolgono informazioni schematiche sull'intero sistema in modo automatico o parzialmente automatico.

- L'analisi dei dati consente di individuare le informazioni rilevanti, relativi a funzionamento, comportamento e struttura del sistema, astraen-

dole dal contesto implementativo. Come per la fase della raccolta dei dati, anche per l'analisi non vi è una tecnica fissata. Innanzitutto l'analisi può essere *generica*, e quindi adattabile ad una moltitudine di programmi estrapolando informazioni generali sul sistema, oppure *specifica*, relativa ad un particolare aspetto o ad una particolare tipologia di software in esame per l'individuazione di problemi mirati. Alcuni strumenti di visualizzazione consentono all'utente di specificare il tipo di analisi da effettuare e di focalizzare l'attenzione solo su alcune parti del sistema. L'analisi può, inoltre, essere effettuata, oltre che dinamicamente, anche staticamente; l'analisi statica si traduce in uno studio del codice per l'individuazione degli elementi del sistema o per un'indicizzazione delle funzionalità (esempio i metodi). Spesso l'analisi statica serve ad includere informazioni di supporto all'analisi dinamica e pertanto viene, generalmente, effettuata prima dell'esecuzione. Nei casi in cui la visualizzazione debba avvenire parallelamente all'esecuzione è necessario che l'analisi, come per la raccolta dei dati, spesso di quantità significativamente estese, sia il più performante possibile.

- La visualizzazione, infine, consente all'utente di esaminare le informazioni acquisite. Visualizzare le informazioni, però, non è un compito semplice. Per rendere la visualizzazione più intuitiva e maggiormente comprensibile per l'utente gli strumenti odierni rappresentano le informazioni sotto forma grafica. Le immagini, infatti, consentono, anche a colpo d'occhio, una visione globale del sistema, permettendo di focalizzare l'attenzione direttamente sui punti d'interesse. Una buona visualizzazione deve essere, come già detto, *navigabile*. L'utente infatti deve poter esplorare i risultati a differenti livelli di astrazione e di enfasi, così da selezionare agevolmente gli aspetti da prendere in esame. A tale scopo alcuni strumenti mettono a disposizione dell'utente una visualizzazione composta da differenti rappresentazioni parallele, ognuna specifica per un particolare aspetto del funzionamento o conformazione del sistema. Secondo *Reiss*[3][4][5], *Dugerdil e Sazzadul Alam*[6][7][8], *Grati, Sahraoui e Poulin*[9] e tanti altri, inoltre, una buona visualizzazione deve possedere un'altra proprietà essenziale: deve essere tempo-dipendente, ovvero deve poter essere navigata anche in funzione del tempo. Secondo questa concezione, infatti, l'utente

deve poter esaminare gli stati del sistema in ogni istante per poter individuare eventuali comportamenti anomali e gli eventi che le hanno causate, o semplicemente per poter comprenderne i meccanismi. Secondo *Dugerdil e Sazzadul Alam*, infatti, la rappresentazione deve essere una sequenza di immagini riproducibili come un vero e proprio film. Se da un lato potrebbe sembrare svantaggioso a causa della quantità di dati in gioco, soprattutto per esecuzioni molto lunghe, tuttavia ciò consente allo sviluppatore di poter esaminare l'esecuzione in un secondo momento, concedendosi tutto il tempo necessario alla comprensione individuando i "momenti cruciali" del sistema.

Capitolo 4

Alcuni strumenti di visualizzazione

Come già detto nel precedente capitolo, gli strumenti di visualizzazione sono innumerevoli e diversificati anche in base ai differenti obiettivi che ognuno di essi si prefigge. In questo capitolo discuteremo solo alcune tipologie di questi strumenti e quindi alcune tecniche particolarmente rilevanti di acquisizione e di rappresentazione delle informazioni.

4.1 Dyview

4.1.1 Descrizione

Il primo strumento analizzato nel dettaglio si chiama Dyview[5]. Dyview è pensato per rappresentare l'esecuzione di sistemi molto complessi, in particolare i server, e quindi per monitorare le transazioni, ovvero le richieste da parte del client, che possono essere suddivise in più task e subtask, e, che a loro volta, possono essere gestiti da più thread.

La particolarità essenziale di questo tool è quella di consentire all'utente di decidere quali aspetti del sistema monitorare. Secondo le esperienze di *Reiss*, infatti, focalizzare la raccolta delle informazioni solo in alcuni ambiti è un approccio molto efficace se si vogliono ridurre i dati, alleggerire l'analisi e sopratutto perturbare l'esecuzione in modo minimo. Un altro punto di forza di Dyview, inoltre, è la possibilità di avere in contemporanea visualizzazione ed esecuzione in modo da permettere all'utente di correlare alcuni eventi

esterni con il funzionamento del sistema. La visualizzazione di Dyview consente di porre attenzione alle interazioni dei thread, alle transazioni e ai task di un server in esecuzione. La visualizzazione (Figura 4.1) è suddivisa in

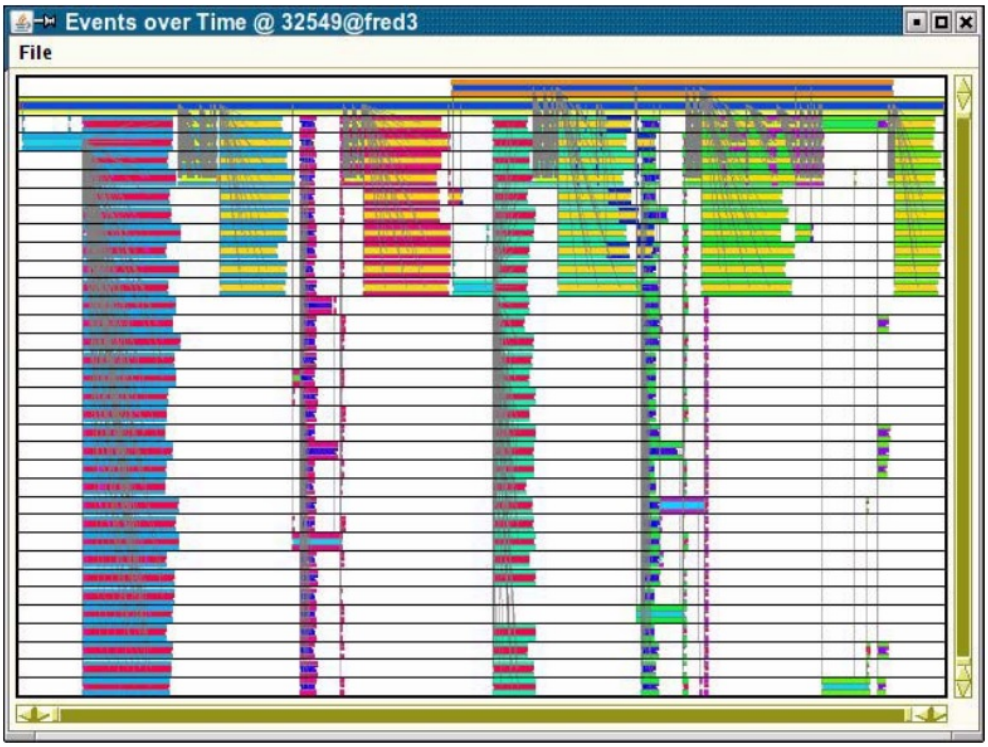

Figura 4.1: Dyview: visualizzazione

righe in cui, ognuna di esse, rappresenta un thread. L'esecuzione all'interno del singolo thread è definita da diversi colori che indicano la transazione e il task in esecuzione; le linee grigie, invece, indicano informazioni riguardanti le relazioni tra task e la creazione della transazione. Per fornire informazioni più dettagliate riguardanti l'esecuzione sono presenti, inoltre, due barre di navigazione per selezionare un particolare lasso di tempo da esaminare e per eseguire uno "zoom" sul o sui thread interessati. Al passaggio del mouse su di un thread apparirà, inoltre, un popup contenente informazioni di supporto riguardanti l'istante temporale in questione, il nome del thread, il task in esecuzione e la transazione corrente; queste informazioni possono essere arricchite tramite particolari specifiche inserite all'interno del codice

sorgente dall'utente stesso per agevolare ulteriormente la comprensione. La vera potenzialità di questo strumento, però, non consiste soltanto nel fatto di avere una visualizzazione specifica a livello di thread, bensì nel fatto che questa visualizzazione venga creata quasi del tutto automaticamente, e quindi con l'intervento minimo da parte del programmatore. Al contrario di VELD[3] (un'esperienza precedente di *Reiss*), infatti, in cui l'analisi del sistema veniva effettuata mediante automi i cui eventi venivano predisposti direttamente dall'utente rendendo il compito molto arduo, Dyview consente al programmatore di scegliere, tra un insieme di classi già selezionate automaticamente (ovvero solo le classi logiche), quelle che includono le transazioni, i task o i thread di maggior interesse. Quindi ricerca i componenti necessari alla visualizzazione mediante un'analisi statica, con la quale determina la gerarchia delle classi, il grafo delle chiamate, la creazione di thread eccetera, e dinamica, in cui vengono individuati gli event handler. A questo punto vengono creati e inclusi nel sistema degli eventi parametrizzati dal thread, la transazione o il task corrente che possono essere, quindi, gestiti in tempo reale e parallelamente all'esecuzione. In questo modo viene creato un archivio dati che consiste in un insieme di tuple contenenti le informazioni sullo stato del sistema e della memoria dipendentemente dal tempo.

4.1.2 Esempio di analisi di un web-crawler

Un esempio tratto da un articolo[5] di *Reiss* propone l'analisi dell'esecuzione di un web-crawler con Dyview.
Un crawler è un software che analizza i contenuti di una rete o di un database; solitamente i motori di ricerca utilizzano web-crawler per analizzare il contenuto di siti web per indicizzarli. Inoltre, tutti gli hyperlink trovati durante questa analisi vengono aggiunti alla lista degli URL per poi poter effettuare l'analisi ricorsivamente anche su di essi. Prima di analizare una pagina, però, il crawler (se programmato adeguatamente) controlla sul file "robots.txt", presente nella root del sito, se sono presenti particolari restrizioni riguardanti pagine da ignorare o specifiche per ridurre il traffico sul sito stesso.

Esaminiamo adesso l'utilizzo di Dyview su un web-crawler in esecuzione (Figura 4.2). Come già detto ogni riga colorata denota un thread distinto il

cui colore interno identifica il particolare processo in esecuzione. Si notano facilmente tre fasi principali di ogni thread:

- Una fase iniziale in cui i thread sono rappresentati quasi esclusivamente come blocchi verdi solidi
- Una fase intermedia in cui prevale il colore celeste
- Una fase finale dove si presenta una situazione variabile

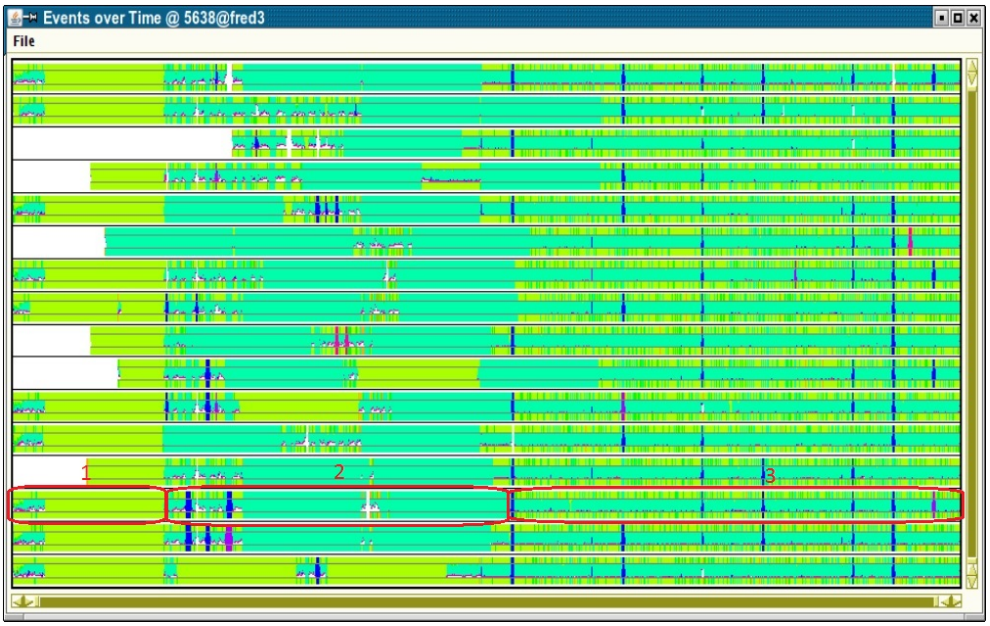

Figura 4.2: Dyview: esecuzione di un crawler web

Il comportamento normale di un web-crawler dovrebbe essere rappresentato come la terza fase, ovvero in cui i thread scaricano e analizzano le pagine in modo continuativo e parallelo. Le prime due, invece, sono fasi anormali. La fase intermedia rappresenta il momento in cui molti thread cercano di scaricare la loro particolare pagina web e il rallentamento che ne deriva. La fase più critica però è quella iniziale; si presenta, infatti, un vero e proprio collo di bottiglia che letteralmente interrompe l'esecuzione, causato dal fatto che tutti i thread tentano contemporaneamente di consultare

il file "robot.txt" per verificare se una pagina è analizzabile. Ciò è causato dai criteri di sincronizzazione che regolano l'accesso e la modifica del file.

Grazie a questa visualizzazione, quindi, è stato possibile riscontrare questo problema ed eventualmente trovare la soluzione più adeguata.

4.1.3 Esempio di analisi di un simulatore di particelle

Un simulatore di particelle è un software che emula l'interazione fisica tra diverse particelle. Per questo esempio osserveremo due immagini.

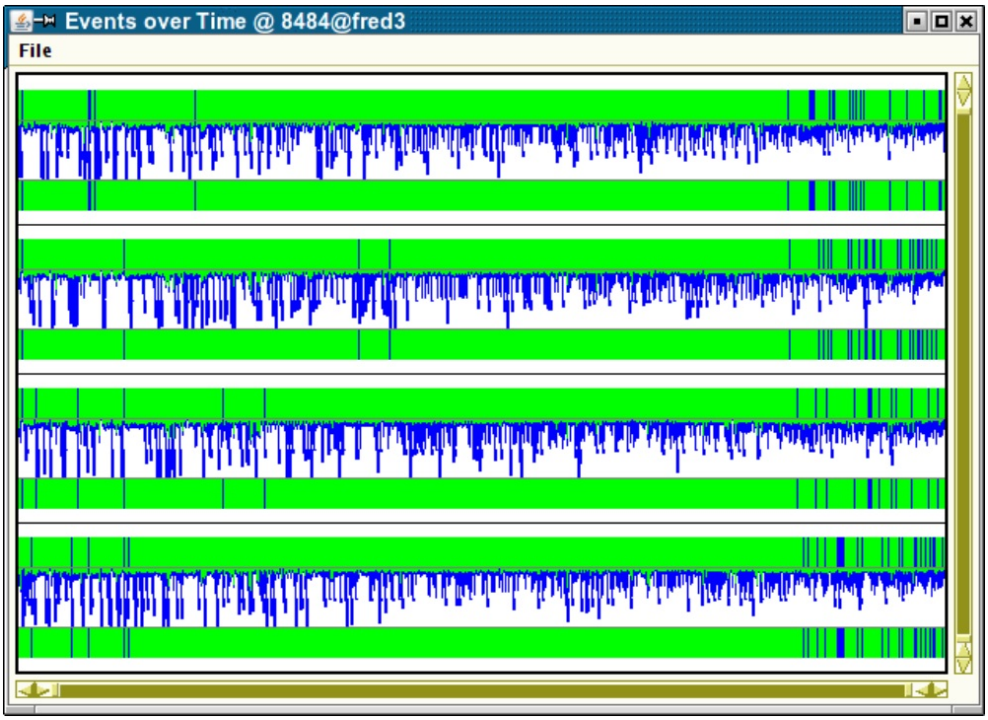

Figura 4.3: Dyview: esecuzione di un simulatore di particelle

La Figura 4.3 mostra la visualizzazione dell'esecuzione del programma in un arco temporale di diversi minuti. Da questa prima visualizzazione più generale si possono subito notare molti spazi bianchi all'interno delle righe che rappresentano i singoli thread; questo sta a significare che i thread sono spesso in stato idle e ciò spiega perché l'applicazione non ha la velocità attesa.

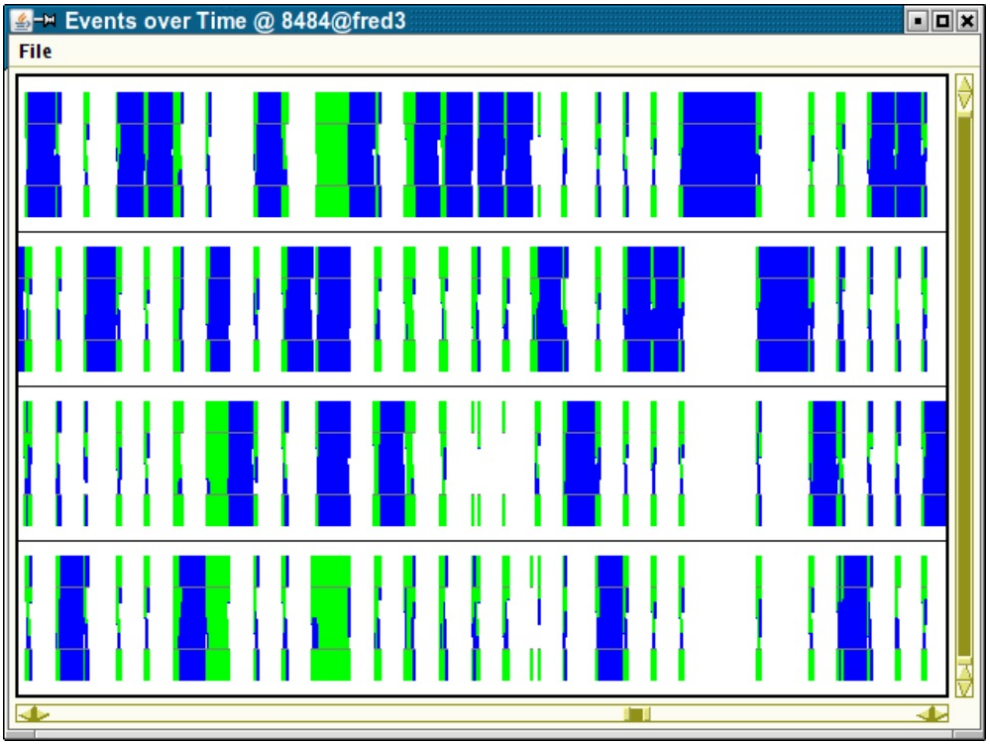

Figura 4.4: Dyview: esecuzione di un simulatore di particelle (nel dettaglio)

La Figura 4.4, invece, mostra il funzionamento più nel dettaglio, ovvero è uno zoom in un particolare istante dell'esecuzione. Le zone blu corrispondono a quel momento del processo in cui viene calcolata la forza di gravità e applicata alle singole forze in gioco; queste computazioni, come emerge dall'immagine, sono predominanti e, inoltre, vengono effettuate sempre da un thread mentre gli altri sono contemporaneamente in stato idle.

4.1.4 Conclusioni Dyview

Dyview, in conclusione, è uno strumento utile allo sviluppatore che, grazie ad un'interazione minima con l'utente e ad un'analisi sia statica che dinamica, consente una visione dettagliata e specifica per il programma in

esame, mettendo in mostra le classi d'interesse e gli aspetti fondamentali del sistema e favorendone una comprensione più approfondita.

4.2 Evospaces

4.2.1 Descrizione

Come già detto nei precedenti capitoli, uno dei problemi fondamentali dei programmi di visualizzazione sta nel fatto di dover rappresentare una grande quantità di dati che, per loro indole, non sono naturalmente rappresentabili.

Un modo particolare di affrontare il problema di come visualizzare questa larga quantità di informazioni, però, è stata affrontata da *Dugerdil e Alam* nel loro sistema di analisi Evospaces[6][8]. Evospaces, infatti, mette a disposizione una rappresentazione molto familiare per l'utente, consentendogli, quindi, di poter concentrare facilmente la sua attenzione sulle informazioni in essa contenute: la cosiddetta *SoftwareCity* è una vera e propria città formata da edifici di diverse forme e dimensioni e dislocati secondo criteri specifici al fine di concentrare in un'unica visualizzazione 3D le caratteristiche principali del sistema in esame.

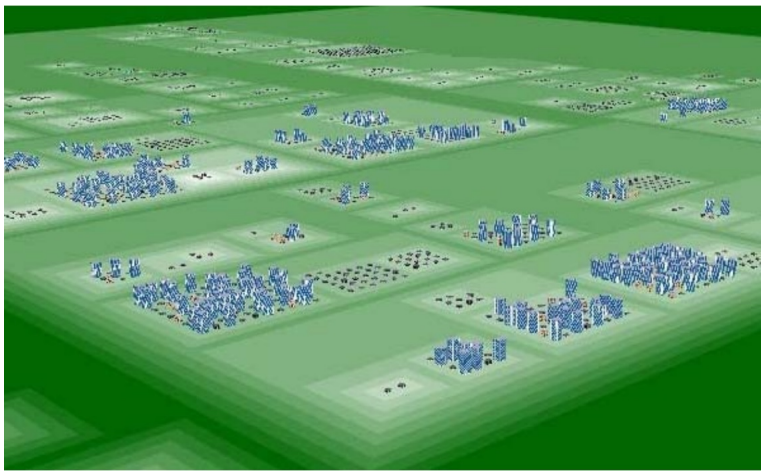

Figura 4.5: Evospaces: la SoftwareCity

Per consentire a questo strumento di visualizzare sistemi scritti in diversi paradigmi di programmazione e per essere in grado di integrare velocemente

eventuali modifiche, Evospaces è stato realizzato in cinque livelli separati (Figura 4.6).

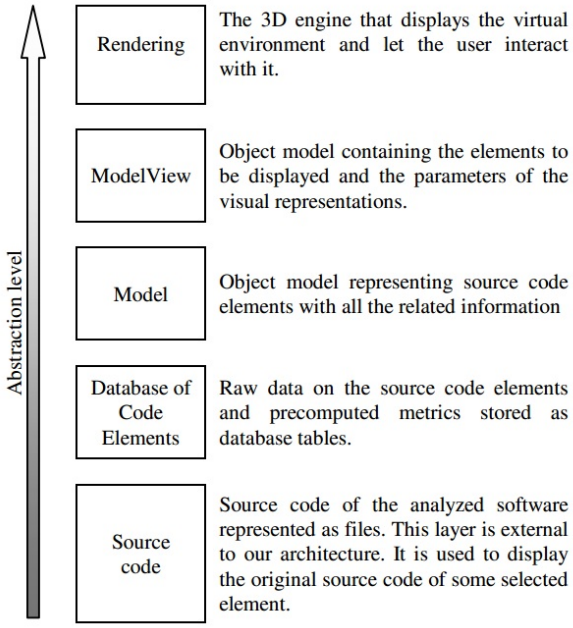

Figura 4.6: Evospaces: livelli di astrazione

4.2.2 Il Source code level

Il primo è il livello Source code del sistema in esame. Questo livello si occupa di esaminare automaticamente i file del codice sorgente per estrarre quante più informazioni possibili.

4.2.3 Il Database level

Tutte le informazioni estratte dal livello del codice sorgente vengono raccolte e memorizzate dal livello Database sotto forma di *entità* e *relazioni*. Gli elementi come classi, metodi, variabili, attributi, package eccetera sono *entità*. I modi in cui questi elementi sono strutturati, comunicano o intera-

giscono sono rappresentati come *relazioni* tra le entità. Viene così costruito un database il cui diagramma ER rappresenta il sistema in esame.

4.2.4 Il Model level

Il livello Model implementa la rappresentazione ad oggetti nel sistema Evospaces delle entità e delle relazioni contenute nel Database. Ogni entità ed ogni relazione, quindi, saranno rappresentate da una specifica classe.

4.2.5 Il Model View level

È il primo livello ad occuparsi degli elementi di visualizzazione e contiene, quindi, tutti i valori e i parametri per la rappresentazione e il rendering delle informazioni in un ambiente 3D.

Gli oggetti del livello Model View sono, infatti, una mappatura degli oggetti del livello Model e, pertanto, anch'essi sono suddivisibili in due gerarchie simili alle entità e le relazioni; a differenza di queste ultime, però, gli elementi del Model View contengono solamente dati visualizzabili.

4.2.6 Il Rendering level

L'ultimo, ma non meno importante, è il livello di Rendering. È questo livello che, grazie alla libreria 3D JOGL (OpenGL for Java), infine, crea, popola ed anima l'ambiente 3D con tutti gli elementi visuali del Model View.

4.2.7 La visualizzazione

Come già detto, Evospaces rappresenta il sistema con una SoftwareCity, ma come le vere città anche questa può essere molto grande e quindi ci si potrebbe facilmente perdere. È necessario, quindi, che la visualizzazione di Evospaces sia strutturata adeguatamente e presenti particolari caratteristiche e punti di riferimento per consentire all'utente di orientarsi. Innanzitutto ogni edificio rappresenta una classe o un file. Gli edifici sono di diverse fogge (Figura 4.7) in relazione a quante linee di codice sono contenute nella classe o nel file; in particolare:

- una casa per file da 0 a 50 linee di codice

- un condominio per file da 51 a 200 linee di codice
- un grattacielo per file con più di 200 linee di codice

Con il municipio, infine, vengono rappresentate i file di interfaccia oppure gli header (.h).

L'altezza degli edifici, invece, denota il numero delle variabili globali dichiarate all'interno del file; in particolare:

- edifici bassi per 0 o 1 variabile
- edifici medi da 2 a 4 variabili
- edifici alti con 5 o più variabili

L'altezza dei municipi dipende, al contrario degli altri edifici, dal numero di funzioni contenuti nell'header a cui esso è riferito.

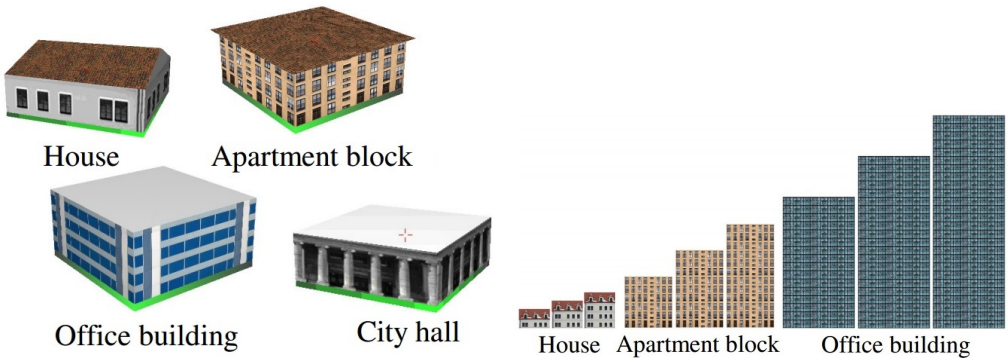

Figura 4.7: Evospaces: tipologie di edifici

Le relazioni tra le diverse classi sono rappresentate tramite dei tubi che collegano gli edifici (Figura 4.8). Alcuni segmenti colorati simulano, inoltre, le interazioni tra classi viaggiando da un edificio all'altra all'interno di questi tubi.

Un'altra caratteristica utile all'utente per orientarsi è la disposizione degli edifici nell'ambiente 3D. Questi possono essere raggruppati in quartieri, in base al package a cui appartiene la classe, oppure presentare una topologia completamente diversa come per esempio a spirale, per ordinare le classi in

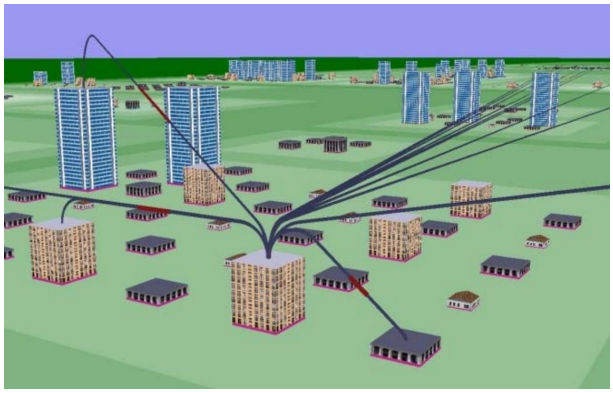

Figura 4.8: Evospaces: relazioni tra classi

senso cronologico rispetto la loro creazione, o a scacchiera, per raggruppare le classi secondo altri criteri.

L'utente può, dunque, spaziare e muoversi per la SoftwareCity liberamente, volando sui grattacieli o camminando per le strade e può, in ogni momento, affidarsi ad una piccola mappa "satellitare", in basso a destra nella schermata, in cui una freccia rappresenta la posizione dell'utente nella città: come avviene nella maggior parte dei videogiochi! È possibile, inoltre, esplorare anche l'interno degli edifici (Figura 4.9); in essi si potranno vedere i "lavoratori", ovvero degli stickman di diversi colori, suddivisi su diversi piani: al piano terra gli stickman saranno di colore rosso e rappresenteranno i metodi della classe; al primo piano saranno blu e rappresenteranno le funzioni; sui piani superiori saranno presenti, inoltre, altri elementi grafici per la rappresentazioni di ulteriori informazioni.

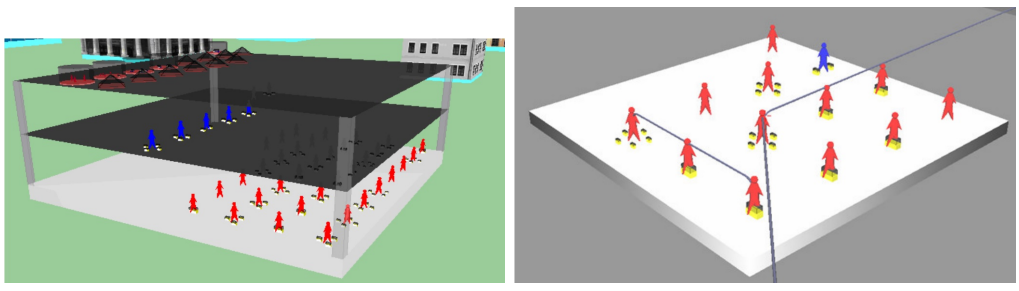

Figura 4.9: Evospaces: interno degli edifici

Staticamente, dunque, il sistema in esame si presenta come una vera e propria città organizzata in settori distinti e popolata da edifici più o meno imponenti. Per quanto riguarda la visualizzazione dinamica, invece, l'obiettivo fondamentale di *Dugerdil e Alam* era quello di avere una rappresentazione dell'esecuzione del programma che si sviluppasse come un film; Evospaces rispecchia questa qualità mantenendo la metafora della SoftwareCity.

Durante la visualizzazione dinamica (Figura 4.10), infatti, sulla nostra città scende la notte; in questo modo solamente i palazzi in cui, in quel particolare momento, avviene "l'azione" si illuminano. L'utente può quindi riprodurre l'esecuzione proprio come un filmato e prendere visione di quali classi prendono parte all'esecuzione ed in quale ordine.

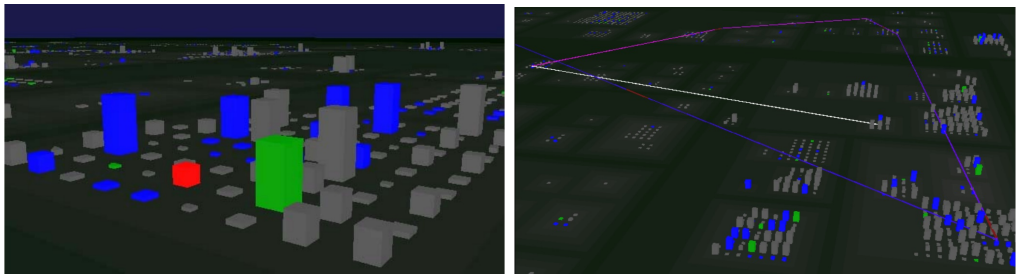

Figura 4.10: Evospaces: visualizzazione dinamica dell'esecuzione

Naturalmente se si dovesse riprodurre un'esecuzione, anche breve, si impiegherebbero ore per visualizzarla interamente per via dell'enorme quantità di operazioni che un computer può svolgere in un solo secondo. Per far fronte alla moltitudine di informazioni, dunque, è necessario introdurre una qualche tecnica di segmentazione. In Evospaces, infatti, la traccia dell'esecuzione viene suddivisa in tanti *segmenti* contigui su cui vengono computate alcune statistiche che riassumono quindi il comportamento globale del sistema in quel lasso di tempo.

Si ha quindi la possibilità di avere due modalità di riproduzione diverse: la riproduzione macroscopica e la riproduzione microscopica. Nella riproduzione macroscopica ogni "fotogramma del filmato" rappresenta un intero segmento: gli edifici che prendono parte all'esecuzione vengono illuminati con colori diversi in base al numero dei metodi eseguiti in quel particolare segmento rendendo visibili, in questo modo, le regioni attive del sistema in ogni momento dell'esecuzione.

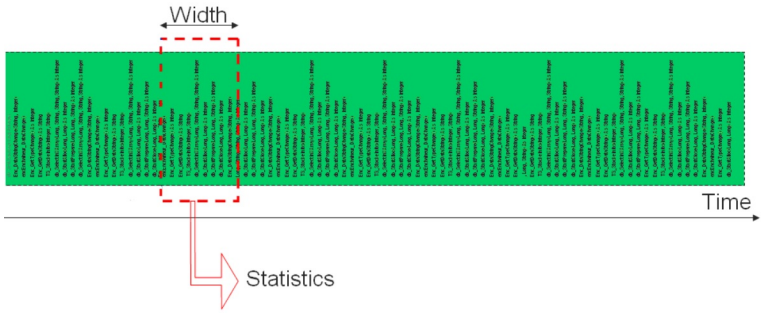

Figura 4.11: Evospaces: segmentazione

Mentre nella riproduzione macroscopica ci si focalizza sull'intera traccia divisa per segmenti, nella riproduzione microscopica l'attenzione si concentra su un singolo segmento. Vengono visualizzati sequenzialmente le chiamate dei metodi, rappresentandole come tubi tra edifici e man mano che si scorrono le singole invocazioni gli edifici si colorano fino ad avere la stessa conformazione della visualizzazione macroscopica.

Durante la visualizzazione dinamica, per navigare i segmenti dell'esecuzione nella parte superiore della schermata appare un pannello con alcuni controlli molto simili all'interfaccia di un qualsiasi riproduttore video; nella Figura 4.12, infatti, sono visibili il pulsante "Play" e la barra temporale della riproduzione macroscopica.

L'utente, inoltre, può impostare la durata di ogni frame in millisecondi, così da poter velocizzare o rallentare la riproduzione. Quando la riproduzione macroscopica è in pausa, invece, l'utente può interagire con i controlli della riproduzione microscopica, del tutto analoghi ai precedenti.

4.2.8 Conclusioni Evospaces

In definitiva, Evospaces è uno strumento dalle enormi potenzialità nell'ambito della comprensione del software. Grazie alla sua interfaccia familiare e intuitiva l'utente può visualizzare istante per istante l'esecuzione in una rappresentazione che non prevede grafici complessi e astratti ma semplicemente una città organizzata e interattiva nella quale potersi muovere e osservare la "vita virtuale" del programma.

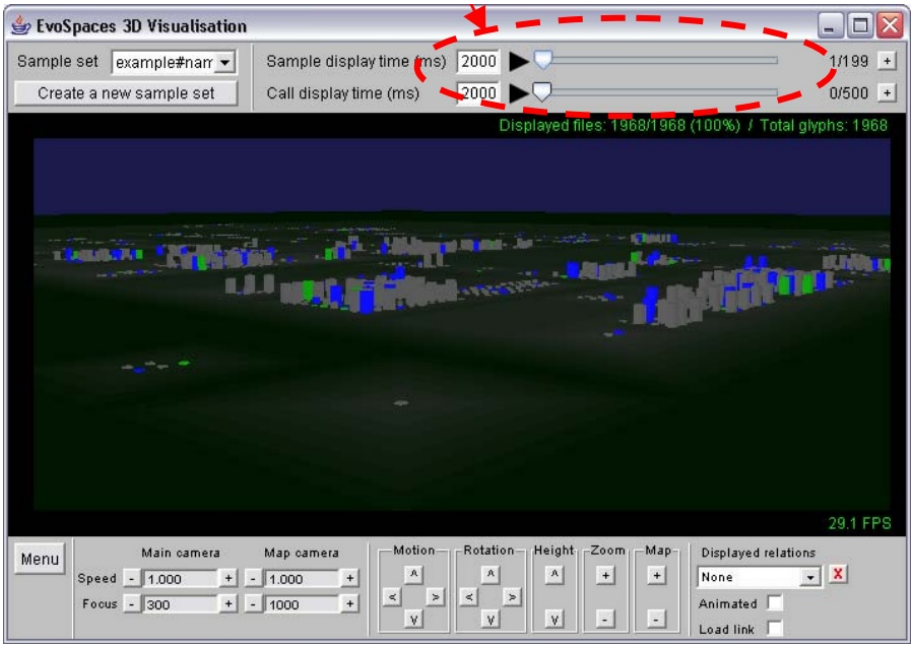

Figura 4.12: Evospaces: interfaccia grafica

Capitolo 5

Realizzare uno strumento di visualizzazione

Nei precedenti capitoli sono stati descritti gli obiettivi e le caratteristiche di un generico strumento di visualizzazione e si è discusso sui vantaggi che l'analisi dell'esecuzione comporta. Sono state, inoltre, esaminate alcune tecnologie già esistenti con le quali si è potuta saggiare l'efficacia e la potenzialità di questi strumenti.
Questo capitolo analizzerà, invece, le tecniche per la creazione di uno strumento di visualizzazione su piattaforma Java e descriverà nel dettaglio la realizzazione di un prototipo funzionante.

5.1 Il controllo dell'esecuzione in Java

Java è una piattaforma software che consente di eseguire qualsiasi programma scritto in linguaggio java.
Il funzionamento di Java è basato sulla Java Virtual Machine, ovvero una macchina virtuale che, indipendentemente dall'hardware e dal sistema operativo, è capace di interpretare ed eseguire i bytecode dei programmi compilati.
Tra le varie infrastrutture messe a disposizione dalla piattaforma Java, ne esiste una in particolare denominata Java Platform Debugger Architecture (JPDA), che fornisce raccolte di API e set di protocolli standard per lo sviluppo di strumenti di debugging e profiler.
La JPDA, dunque, consente di interagire con la JVM per attingere informa-

zioni relative ad una particolare esecuzione ed eventualmente controllarla. Vi sono tre differenti livelli di API inclusi nella JPDA:

- Java Debug Interface (JDI) definisce un'interfaccia Java di alto livello e facile utilizzo che consente di controllare e verificare lo stato dell'esecuzione di un programma java
- Java Virtual Machine Tools Interface (JVMTI) definisce un'interfaccia nativa di basso livello che, similmente alla JDI, consente di ispezionare lo stato di un'esecuzione e di assumerne il controllo
- Java Debug Wire Protocol (JDWP) definisce il protocollo di comunicazione tra il debugger e la Java Virtual Machine su cui è in esecuzione il programma da esaminare

Gli sviluppatori possono, dunque, usufruire indistintamente di questi livelli di astrazione nello sviluppo di uno strumento di debugging o profiling.
La JDI, tuttavia, ha un'ulteriore punto di forza che la rende in alcuni casi essenziale: consente, al contrario della JVMTI, di sviluppare facilmente applicazioni per il debugging in remoto.

5.2 InterView: un prototipo di strumento di visualizzazione

In questo paragrafo verranno descritte brevemente le fasi di realizzazione di InterView, un semplice strumento per la visualizzazione grafica delle interazioni tra oggetti.
La creazione di questo programma è stata effettuata tenendo in considerazione quelle che sono le caratteristiche principali di uno strumento di visualizzazione discusse nei precedenti capitoli; durante l'esposizione dei vari stadi di progettazione, quindi, verrà data maggiore enfasi alle particolari scelte progettuali relative a tale contesto.

5.2.1 Ideazione

La prima fase della realizzazione del prototipo è stata quella di riflettere su quale dovesse essere il suo particolare scopo.
Si è, dunque, pensato che, per uno strumento di visualizzazione, potesse

essere un obiettivo soddisfacente quello di rappresentare dinamicamente la successione delle invocazioni di metodo tra i differenti oggetti istanziati.
Una tale rappresentazione, infatti, consentirebbe di monitorare il flusso di esecuzione di uno o più thread, e, contemporaneamente, delineare quali siano gli oggetti maggiormente coinvolti.
In questo modo, per di più, si avrebbe una raffigurazione del sistema in esame sotto forma di componenti elementari e una riproduzione delle interazioni che intercorrono fra le differenti parti.
È stato, dunque, naturale immaginare una rappresentazione in cui gli oggetti siano visualizzati sotto forma di tanti cerchi e le invocazioni di metodo tra un oggetto e l'altro come segmenti tra due cerchi.

Un punto fondamentale che è emerso fin dall'inizio, inoltre, è stato quello di pensare ad una visualizzazione dinamica tempo-dipendente, che potesse essere navigata proprio come fosse un video; in questo modo sarebbe stato possibile fornire all'utente una serie di controlli molto intuitivi e di semplice utilizzo che gli consentissero di esaminare liberamente parti specifiche dell'esecuzione gestendo anche la velocità di riproduzione.

5.2.2 Progettazione

Come già affermato nei precedenti capitoli, un'importante requisito per uno strumento di visualizzazione è quello di perturbare il meno possibile l'esecuzione del programma in esame.
Partendo da questo requisito fondamentale si è deciso di strutturare l'architettura (Figura 5.1) del sistema in tre macro-componenti:

- un componente attivo (*Monitor*) per il monitoraggio dell'esecuzione del programma in esame
- un componente (*Archivio Dati*) per la gestione e il mantenimento dei dati raccolti
- un componente attivo (*Visualizzazione*) per la visualizzazione dei dati

È importante notare il fatto che i componenti *Monitor* e *Visualizzazione* siano componenti attivi; questo è necessario perché l'esecuzione e la visualizzazione lavorano con tempi estremamente diversi.

In questo modo, dunque, il monitoraggio dell'esecuzione, e quindi la raccolta dei dati relativi ad essa, avviene indipendemente dalla visualizzazione e, pertanto, il sistema in esame non viene influenzato in maniera rilevante.

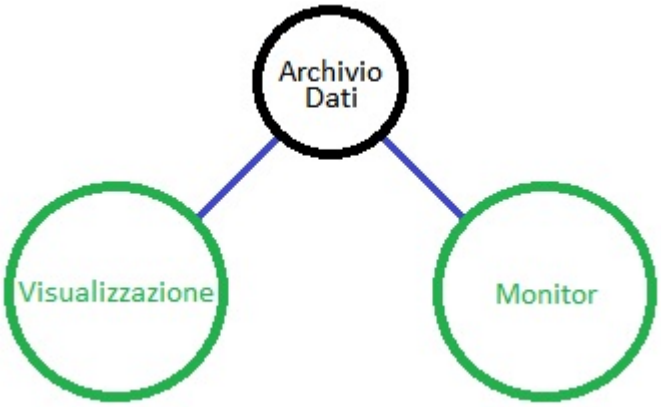

Figura 5.1: InterView: architettura

5.2.3 Implementazione

Nello sviluppo del prototipo si è scelto, per motivi di semplicità, di utilizzare l'interfaccia JDI, discussa precedentemente.
L'interfaccia JDI consente di creare una connessione con la Java Virtual Machine e richiedere la notifica di particolari eventi quando essi si verificano durante l'esecuzione.
Nel caso specifico, il prototipo richiede e gestisce tutti gli eventi relativi alla chiamata di un metodo ed al caricamento di una classe.
A tale riguardo segue qualche dettaglio implementativo che illustra in modo semplificato la gestione degli eventi.

Il codice seguente richiede alla Macchina Virtuale di inviare tutte i futuri eventi di notifica ogni qual volta viene caricata una classe.
L'oggetto "vm" è di tipo VirtualMachine ed è connesso, tramite particolari oggetti Connector, con la Macchina Virtuale e ne rispecchia lo stato corrente.

Registrazione a eventi relativi al caricamento di classi

```
1 EventRequestManager erm = vm.eventRequestManager();
2 ClassPrepareRequest cpr = erm.createClassPrepareRequest();
3 cpr.setEnabled(true);
```

Di seguito, invece, è riportato il metodo "addMethodsWatch" che si occupa di registrarsi agli eventi dovuti alle invocazioni di metodi relativi ad una specifica classe ("ReferenceType").

Registrazione a eventi relativi alle invocazioni dei metodi di una classe

```
1 private void addMethodsWatch(VirtualMachine vm, ReferenceType
    refType) {
2    EventRequestManager erm = vm.eventRequestManager();
3    MethodEntryRequest mer = erm.createMethodEntryRequest();
4    mer.addClassFilter(refType);
5    mer.setEnabled(true);
6 }
```

Viene riportato, infine, il metodo "run" del thread relativo all'ascoltatore degli eventi, ovvero il componente *Monitor* descritto nel precedente paragrafo.

Ascoltatore degli eventi

```
1  public void run(){
2     //CODA DEGLI EVENTI
3     EventQueue eventQueue = vm.eventQueue();
4     while (true) {
5        //PRELEVO GLI EVENTI DALLA CODA
6        EventSet eventSet = eventQueue.remove();
7        for (Event event : eventSet) {
8           if (event instanceof ClassPrepareEvent) {
9           //GESTIONE EVENTI DI CARICAMENTO CLASSE
10             ClassPrepareEvent classPrepEvent =
11                      (ClassPrepareEvent) event;
12
13             ReferenceType refType =
14                      classPrepEvent.referenceType();
15
16             /*REGISTRAZIONE AGLI EVENTI
17               GENERATI DAI METODI DELLA
18                    CLASSE CORRENTE*/
19             addMethodsWatch(vm, refType);
20          }else if (event instanceof MethodEntryEvent) {
21          //GESTIONE EVENTI DI INVOCAZIONE METODO
```

```
22              MethodEntryEvent metEvent =
23                      (MethodEntryEvent) event;
24
25              Method mtd = metEvent.method();
26
27              /*INDIVIDUO IL ''CHIAMANTE''
28                   ED IL ''CHIAMATO''*/
29              ObjectReference caller = null;
30              ObjectReference called = null;
31
32              List<StackFrame> frames =
33                      metEvent.thread().frames();
34
35              for(StackFrame f : frames){
36                 if(called==null)called = f.thisObject();
37                 else if(caller==null)caller = f.thisObject();
38                 else break;
39              }
40
41              //     [..]
42
43          }
44      }
45      eventSet.resume();
46  }
47 }
```

5.2.4 Esempio di analisi di BankSimulator

BankSimulator è un programma sviluppato individualmente dagli studenti durante il corso di Sistemi Operativi al fine di applicare le metodologie di una corretta gestione della concorrenza. Il programma esegue una simulazione di un semplice sistema bancario in cui diversi impiegati svolgono operazioni su un numero limitato di conti corrente; impostando il numero di thread (impiegati), il numero di operazioni da eseguire e il numero di conti corrente è possibile avviare la simulazione.
Monitorando l'esecuzione di BankSimulator con InterView si potrà osservare una splendida riproduzione dell'intera esecuzione. Di seguito qualche screenshot dettagliato.

La Figura 5.2 mostra un fotogramma particolare dell'esecuzione di Bank-

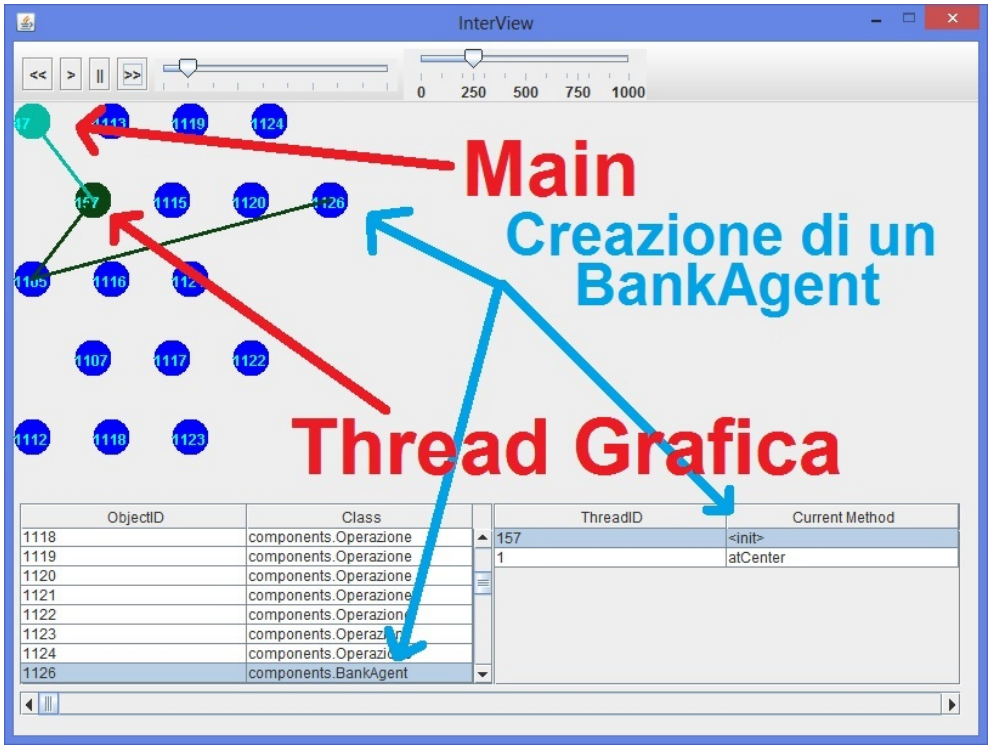

Figura 5.2: InterView: esecuzione di un BankSimulator

Simulator. Si può ricostruire il funzionamento osservando dettagliatamente la visualizzazione.
Inizialmente il thread main crea l'oggetto JFrame (in figura id 157); quando l'utente avvia la simulazione il thread 157 crea l'oggetto SimulatorStarter (id 1105) che, tramite l'invocazione del metodo "startSimulation", procede a creare tutti gli oggetti utili alla simulazione: conti corrente, operazioni e infine i BankAgent (impiegati).
L'istante fotografato in figura, dunque, mostra proprio la creazione del primo BankAgent della simulazione (id 1126).

La Figura 5.3, invece, mostra un istante dell'esecuzione in cui sono già attivi cinque BankAgent differenti che eseguono le loro operazioni autonomamente.
I diversi BankAgent, e quindi i diversi thread, sono evidenziati da colori dif-

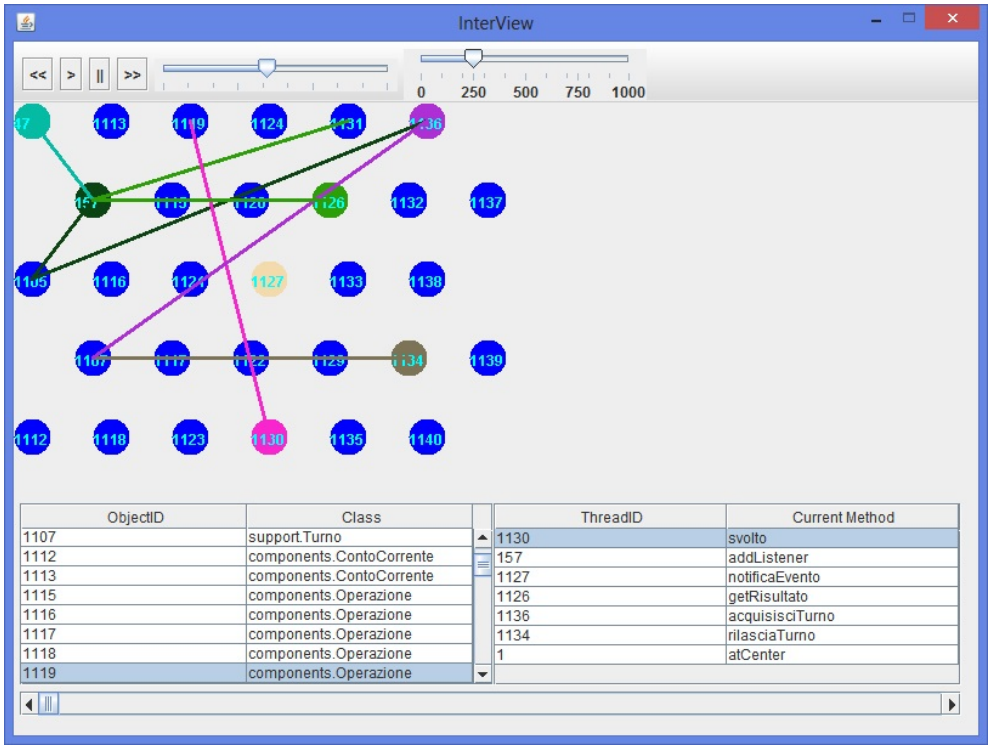

Figura 5.3: InterView: esecuzione di un BankSimulator

ferenti ed è possibile, a colpo d'occhio, esaminare lo stack delle invocazioni percorrendo i segmenti dello stesso colore. Nella tabella in basso, inoltre, viene riportato, per ogni thread, il nome del metodo corrente e, per ogni oggetto, la classe di appartenenza.

5.2.5 Conclusioni InterView

Nonostante sia uno strumento molto semplice, InterView fornisce comunque la possibilità di esaminare l'esecuzione con una visualizzazione di facile utilizzo e comprensione.
Mostrando i differenti thread, inoltre, InterView può essere utilizzato al fine di riscontrare eventuali problemi dovuti alla concorrenza o comportamenti inaspettati riguardo al funzionamento. Con la realizzazione di questo prototipo si sono voluti applicare, seppure in modo semplificato, i prin-

cipi descritti in questa trattazione in modo da valutarne la validità con un'esperienza personale e fornire al lettore una dimostrazione concreta.

Capitolo 6

Conclusioni

Il presente elaborato ha, dunque, posto l'attenzione su un particolare ramo dell'attuale ricerca in campo informatico: la visualizzazione di programmi in esecuzione.

L'argomento è stato trattato, in un primo momento, da un punto di vista più teorico mettendo in evidenza le esigenze degli sviluppatori e le modalità con cui uno strumento di visualizzazione deve soddisfare tali esigenze; pertanto, si è giunti alla conclusione che il modo ideale di agevolare il lavoro dello sviluppatore è di mostrare, con una qualche rappresentazione, struttura, interazione e comportamento del sistema. Ci si è soffermati, inoltre, ad esaminare i principali requisiti e l'architettura di un generico strumento di visualizzazione.

In un secondo momento, invece, sono stati esaminati nello specifico due strumenti: Dyview e EvoSpaces.
I due strumenti analizzati sono stati scelti in modo tale da mostrare il fatto che il problema della visualizzazione di programmi in esecuzione sia estremamente ampio e come ogni strumento focalizzi la propria attenzione su aspetti similari ma che, in base all'obiettivo preposto, potrebbero risultare comunque abbastanza diversificati. Nello specifico, esaminando Dyview, si è notato come questo strumento dia informazioni più qualitative e generiche riguardo sopratutto aspetti relativi alle performance, quindi colli di bottiglia e esecuzione dei singoli processi, e sul funzionamento globale del sistema; con EvoSpaces, al contrario, si nota un'analisi più puntuale dell'esecuzione ponendo l'accento sulle interazioni tra entità più o meno astratte, mantenendo un contensto che rappresenti, a mio parere in modo eccellente,

la struttura del sistema e la suddivisione delle responsabilità tra le sue parti.

Infine, è stato realizzato un prototipo di strumento di visualizzazione che rispecchiasse le caratteristiche e i requisiti principali espressi in precedenza. In questo modo si è potuta constatare la validità delle argomentazioni e dimostrare, con questo semplice prototipo, le potenzialità di questa categoria di strumenti.

In conclusione risulta, quindi, che la visualizzazione dei programmi in esecuzione è un settore molto importante nell'ambito, sempre più complesso, dello sviluppo dei software perché consente di monitorare una corretta produzione dei sistemi informatici e allo stesso tempo garantisce un controllo qualitativo e funzionale nella creazione dei programmi.

Bibliografia

[1] Hugo Leroux and Chris Exton: *A Tool for Representing Concurrent Object-Oriented Program Execution through Visualisation*, (2001)

[2] Chris Exton: *Dynamic Visualization of Concurrent Object-Oriented System*, IEEE (2000)

[3] Steven P. Reiss: *Visualizing Program Execution Using User Abstractions*, ACM (2006)

[4] Steven P. Reiss and Alexander Tarvo: *What Is My Program Doing? Program Dynamics in Programmer's Terms*, (2012)

[5] Steven P. Reiss and Suman Karumuri: *Visualizing Threads, Transactions and Tasks*, ACM (2010)

[6] Philippe Dugerdil and Sazzadul Alam: *Execution Trace Visualization in a 3D Space*, IEEE (2008)

[7] Philippe Dugerdil and Sazzadul Alam: *EvoSpaces Visualization Tool: Exploring Software Architecture in 3D*, IEEE (2008)

[8] Philippe Dugerdil and Sazzadul Alam: *EvoSpaces: 3D Visualization of Software Architecture*, IEEE (2008)

[9] Hassen Grati, Houari Sahraoui, Pierre Poulin: *Extracting Sequence Diagrams from Execution Traces using Interactive Visualization*, IEEE (2010)

[10] Dean F. Jerding, John T. Stasko, Thomas Ball: *Visualizing Interactions in Program Executions*, ACM (1997)

[11] Sravanthi Dandamudi, Sally K. Wahba, and Jason O. Hallstrom: *An Animation Framework for Improving the Comprehension of TinyOS Programs*, ACM (2011)

[12] David Harel, Itai Segall: *Visualizing Inter-Dependencies Between Scenarios*, ACM (2008)