POLITECNICO DI MILANO
Corso di Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

MINERVA: IMPLEMENTAZIONE DEL MUSEO VIRTUALE DELL'ISOLA COMACINA

Relatore: Ing. Francesco AMIGONI
Correlatori: Prof. Stefano DELLA TORRE
Dott.ssa Viola SCHIAFFONATI

Tesina di Laurea di:
Gianluigi Calcaterra, matricola 642545
Massimo Franzosi, matricola 642589

Anno Accademico 2004-2005

Ai miei genitori

Sommario

Il lavoro della presente tesina si inserisce nell'ambito del Progetto Minerva del Politecnico di Milano ed è mirato allo sviluppo di un software per la creazione guidata di un museo virtuale da parte dell'utente. L'applicativo realizzato verrà affiancato al museo vero e proprio del Comune di Ossuccio, permettendo ai visitatori di consultare un numero ragguardevole di opere che, per mancanza di spazio, non verrebbero altrimenti esposte.
Lo scopo di questa tesina è illustrare l'implementazione del software, basata sulle analisi effettuate parallelamente da Andretta ed Erba con la committenza. Il lavoro è stato quindi caratterizzato da una forte interazione tra le due coppie di tesisti, in modo da verificare passo dopo passo la continua attinenza dell'applicativo in via di sviluppo con le specifiche iniziali.
Il sistema è costituito da un client, che visualizza le pagine del museo virtuale e permette all'utente di effettuare delle scelte, e un da server, che elabora le richieste appoggiandosi a un database.
Il risultato finale è stato presentato al nostro committente, il Prof. Della Torre, docente del Politecnico di Milano, che ha confermato il pieno raggiungimento degli obiettivi prefissati. Restano margini di sviluppo soprattutto per quanto riguarda il raffinamento dell'interfaccia utente, legati all'introduzione di eventuali elementi tridimensionali. Abbiamo comunque orientato nostro lavoro in modo tale che possa costituire una solida base di partenza per qualsiasi sviluppo futuro, senza dover apportare modifiche sostanziali alla struttura del progetto.

Ringraziamenti

Sono molte le persone che devo ringraziare per questi anni universitari.
Cominciando da quelli che hanno lavorato insieme a me, voglio ringraziare Gabri, Max e Riki per il supporto durante tutto il periodo della tesina. Ringrazio Ale e Davide per le interminabili chiacchierate e tutti i compagni di "viaggio". Mi piacerebbe poi fare la lista di tutti gli amici universitari, del pino e non, ma evito per paura di dimenticarne qualcuno.
Vorrei ringraziare infine i miei genitori. Con questo lavoro spero di poter ripagare almeno in parte i sacrifici che hanno fatto per permettermi di arrivare a questo traguardo.

Indice

Capitolo 1

Introduzione

Il Progetto Minerva si colloca all'interno di quell'insieme di prodotti informatici a disposizione dei responsabili museali, in particolare di quelli relativi ai *musei virtuali*. I software di questa categoria si occupano della generazione su computer di ambienti tridimensionali, reali o immaginari nei quali vengono esposti oggetti museali. Grazie alla larga diffusione avuta negli ultimi anni da Internet i musei virtuali riscontrano sempre più successo e, grazie all'utilizzo di standard affermati, sono fruibili da un'ampia utenza.
La versione di Minerva presentata in questa tesina trae fondamento da quanto realizzato presso il Politecnico di Milano a partire dal 1995 da un'idea del Prof. Marco Somalvico. Motivo di un ulteriore sviluppo è stato quello di fornire al Comune di Ossuccio uno strumento software in grado di aiutare e guidare i visitatori nella creazione di un museo virtuale, soddisfacendo i propri interessi storici e culturali. Tale applicativo consente di consultare un'ampia quantità di reperti rinvenuti presso l'isola Comacina sul Lario che altrimenti non sarebbero fruibili dal pubblico, date le ridotte aree espositive a disposizione.
Lo scopo della tesina è descrivere le tecnologie e le modalità con cui sono stati implementati i requisiti per la nuova versione di Minerva individuati nel lavoro di Andretta ed Erba. Si è quindi proceduto a una attenta valutazione degli strumenti software da impiegare all'interno del Progetto Minerva che permettessero di adempiere alle richieste del committente. Per individuare gli obiettivi del progetto si sono svolti diversi incontri con il Prof. Della Torre, rappresentante della committenza, nei quali si sono discusse le problematiche da affrontare e le possibili soluzioni. Inoltre si sono tenute riunioni periodiche con il nostro relatore Ing. Francesco Amigoni ed il correlatore Dott.ssa Viola Schiaffonati per approfondire i temi trattati e valutare

il lavoro svolto.
Lo sviluppo dell'applicativo per il Comune di Ossuccio è proceduto parallelamente agli incontri, partendo da un prototipo iniziale che si è trasformato progressivamente in un prodotto valido e corrispondente alle richieste della committenza. Sono state valutate e adottate tutte quelle tecnologie ampiamente diffuse che potessero fornire all'utente un'esperienza gratificante nella creazione del museo virtuale e nella consultazione delle opere, consentendo nel contempo l'aderenza a standard ampiamente diffusi.
Il modello di sviluppo a spirale, costituito dalle fasi di analisi, codifica e valutazione, è stato fondamentale per impostare il lavoro in maniera corretta fin dal principio. Inoltre sono state adottate tecniche di programmazione tali da conferire all'applicativo sia un elevato standard qualitativo, sia un'alta efficienza generale, anche nella gestione di eventuali malfunzionamenti. Infine sono state valutate le prestazioni generali del sistema per garantire all'utente tempi di attesa ragionevoli.
Terminata la fase di sviluppo, è stato illustrato il funzionamento di Minerva alla committenza, che si è dichiarata soddisfatta del risultato ottenuto. Possibili sviluppi del Progetto Minerva riguardano principalmente un miglioramento dell'interfaccia utente e una generalizzazione dell'applicativo.

Il nostro lavoro si è sviluppato a partire da due punti cardine: l'ultima versione disponibile del Progetto Minerva, rilasciata da Fabio Ghidotti [1] e denominata Progetto Minerva05, e l'analisi dei requisiti effettuata dai nostri colleghi Andretta ed Erba [2]. Minerva05 è uno strumento destinato ai curatori dei musei per assisterli durante il processo di allestimento. Attraverso questo software è possibile creare un ambiente di navigazione tridimensionale in modo da fornire all'utente un'anteprima dell'esposizione che verrà poi realizzata in pratica. L'attività di analisi dei requisiti è stata caratterizzata da una serie di incontri con la committenza, atti a definire gli obiettivi del progetto. Nelle prime riunioni sono state illustrate a grandi linee le peculiarità fondamentali del museo dell'isola Comacina e dei reperti esposti, ed è stato analizzato come il nostro strumento informatico avrebbe potuto inserirsi in tale contesto. Nelle riunioni successive è stato affiancato ai nostri colleghi l'Arch. Andrea Bonavita, in qualità di collaboratore del Prof. Della Torre, con il preciso compito di curare la parte relativa ai contenuti. I nostri colleghi, grazie all'aiuto dell'Ing. Amigoni e della Dott.ssa Schiaffonati, hanno successivamente tradotto i concetti emersi in più rigorose specifiche di progetto. Nel frattempo abbiamo intrapreso un'analisi del software Minerva05, per comprenderne appieno la struttura ed il funzionamento. Solamente attraverso una precisa conoscenza di questo applicativo sarebbe stato possi-

bile delineare uno sviluppo adeguato, che non solo ci portasse a raggiungere i nuovi obiettivi, ma che garantisse la retrocompatibilità del nostro lavoro con quelli precedenti.
Minerva05 adottava un'architettura a due componenti secondo il classico paradigma client-server. Dalle nostre analisi è apparso chiaro che la struttura lato server poteva essere funzionale al nostro contesto, e che con opportune modifiche e accorgimenti si sarebbero potute facilmente implementare numerose delle caratteristiche richieste. Il client invece risultava troppo specifico per essere riutilizzato, per cui abbiamo optato per crearne uno ex novo. In effetti gli scopi dei due progetti sono molto diversi fra loro. Mentre il software di Ghidotti nasceva come strumento per gli allestitori come supporto alla creazione di musei virtuali fruibili in maniera passiva dall'utente, il nostro conferisce all'utente un ruolo attivo nella visita permettendogli di creare allestimenti in tempo reale.
A lato client sono state utilizzate tecnologie e standard affermati del web come HTML [3] e CSS [4], per fornire all'utente un ambiente familiare per la consultazione del museo virtuale. La creazione dinamica delle pagine è gestita da Apache Tomcat [5] sfruttando la tecnologia Java Servlet [6]. A lato server è stata mantenuta la struttura ad agenti della versione precedente, basata sul framework JADE [7]. Per quanto riguarda l'archiviazione dei dati è stato adottato il database MySQL [8], che offre l'affidabilità e le prestazioni richieste per questo progetto.
A fronte di queste prime valutazioni, è stato elaborato un prototipo del programma, dalle caratteristiche semplici ed essenziali sia dal punto di vista grafico, sia per quanto riguarda le funzionalità proposte. Questo primo lavoro si è rivelato di grande aiuto, perché ci ha consentito di constatare, a seguito di una riunione con la committenza, che la linea di sviluppo da noi intrapresa stava effettivamente seguendo la direzione desiderata. In un contesto che stava diventando sempre più chiaro, sono stati quindi definiti in maniera più rigorosa gli obiettivi da raggiungere nella versione definitiva del progetto.
Il nostro lavoro ha previsto anche il superamento di alcuni problemi individuati nella tesina dei nostri colleghi Andretta ed Erba, per i quali tuttavia non si era individuata una soluzione appropriata. In particolare è stato risolto il problema della concorrenza degli accessi, permettendo a più utenti di utilizzare l'applicazione simultaneamente e aprendo le porte alla effettiva realizzazione pratica del sistema distribuito. Inoltre sono state omogeneizzate le prestazioni sui diversi sistemi operativi su cui si è effettuato lo sviluppo, in modo tale da fornire alla committenza una più ampia scelta sulle piattaforme hardware e software da adottare nel museo.

Ulteriori sviluppi del lavoro qui presentato si potranno concentrare sull'interfaccia utente, sul processo di allestimento e sul modello distribuito. Per quanto riguarda l'interfaccia, si potrebbe ipotizzare l'introduzione di elementi tridimensionali che possano rendere l'esperienza virtuale dell'utente ancora più suggestiva e coinvolgente. Il processo di allestimento, qualora volesse essere raffinato, potrebbe in futuro appoggiarsi sull'agente Allestitore, già presente in Minerva05, ma che abbiamo deciso di non utilizzare. Infine si potrebbe mettere in pratica il modello distribuito, da noi concepito e implementato però mai collaudato, verificandone la bontà e riscontrandone i possibili vantaggi.

La tesina è strutturata nel modo seguente.

Nel Capitolo 2 si mostrano i mezzi informatici attualmente disponibili per la consultazione di contenuti museali. Inoltre viene descritto lo stato dell'arte a partire da Minerva05 e gli obiettivi della tesina Andretta-Erba.

Nel Capitolo 3 si illustrano i requisti fondamentali del software da utilizzare all'interno del museo dell'isola Comacina e come possano essere effettivamente soddisfatti nella pratica.

Nel Capitolo 4 si descrivono le tecnologie adottate all'interno del Progetto Minerva, gli strumenti di sviluppo utilizzati nel gruppo di lavoro e l'evoluzione del software prodotto. Inoltre è presente la soluzione logica adottata per archiviare le informazioni necessarie al museo virtuale.

Nel Capitolo 5 si mostra l'architettura generale del sistema e le tecnologie utilizzate per la sua implementazione. Inoltre vengono trattate le soluzioni adottate per una migliore efficienza dell'applicativo.

Nel Capitolo 6 si illustrano i risultati ottenuti e i giudizi della committenza. Inoltre vengono descritte le qualità principali del software, come l'efficienza, la tolleranza ai guasti e le prestazioni.

Nel Capitolo 7 si descrivono i possibili sviluppi futuri e le conclusioni del lavoro svolto in questa tesina.

Nell'appendice A si riportano i dati raccolti per effettuare le analisi delle prestazioni consultabili del Capitolo 6.

Capitolo 2

Stato dell'arte

Il presente capitolo vuole illustrare quali mezzi informatici per la fruizione dei contenuti museali siano attualmente disponibili. La trattazione contiene una panoramica sulle principali implementazioni delle tecnologie informatiche nell'ambito dei musei e non vuole essere completamente esaustiva sull'argomento, in quanto richiederebbe una lunga ed approfondita analisi, il che non rientra negli scopi della tesina.
Questo progetto è la successiva evoluzione del software *Minerva05* sviluppato da Fabio Ghidotti [1] e quindi analizzeremo innanzitutto le caratteristiche fondamentali e peculiari di tale versione di Minerva. Presentiamo poi alcuni esempi di software utilizzati nei musei. Essi vengono suddivisi in due categorie: quelli che permettono all'utente di creare un museo virtuale basandosi sulle proprie decisioni e quelli, invece, che offrono solamente una modalità di semplice esplorazione dei contenuti.

2.1 Minerva05

In ambito museale, l'introduzione di strumenti informatici di supporto per la fruizione dei musei da parte del pubblico è sempre stata accolta con una certa diffidenza. Il motivo di questa scarsa accettazione può essere ricercato nella mancanza di strumenti che siano in grado di fornire valore aggiunto e dalla scarsa propensione all'innovazione tecnologica da parte sia degli esperti del settore, sia degli utenti finali.
Il Progetto Minerva, realizzato dal Politecnico di Milano, rappresenta un significativo passo in avanti rispetto agli applicativi più comuni, in quanto esso propone un sistema che si occupa della progettazione automatica di allestimenti museali all'interno di ambienti virtuali. Tale sistema prevede

l'utilizzo di tecniche di intelligenza artificiale e di realtà virtuale che forniscono ottimi strumenti per ottenere un prodotto efficiente ed efficace.
Il software prodotto, Minerva05, offre un supporto alla creazione di allestimenti museali in ambito archeologico attraverso criteri e regole. Ghidotti ha sviluppato Minerva05 partendo dal progetto originario nato nel 1995 da un'idea del Prof. Marco Somalvico [9]: purtroppo il prodotto non si era mai avvalso della collaborazione diretta di esperti del settore e questo ha determinato un progressivo allontanamento dei risultati dalle loro richieste, sebbene il software ottenuto presentasse aspetti interessanti dal punto di vista delle tecnologie impiegate.
Il progetto Minerva05 è stato infatti concepito con l'obiettivo di superare uno dei maggiori vincoli della prima versione di Minerva, ossia la scarsa flessibilità che caratterizzava tutta la fase di allestimento: il programma faceva riferimento a principi di allestimento automatizzato non configurabili, assumendo di poter impiegare regole generali e universali nella delicata operazione di disposizione degli oggetti all'interno dello spazio virtuale.
Minerva05 utilizza un nuovo tipo di approccio con l'utente: la concreta interazione tra il sistema e il curatore dell'esposizione, in cui il primo applica regole e criteri di allestimento specifici per l'ambito archeologico, seguendo le indicazioni fornite dal secondo e non operando in maniera completamente autonoma. Inoltre è in grado di rendere fruibili i risultati ottenuti attraverso la realizzazione di un modello tridimensionale del museo, navigabile interattivamente dall'utente.
Un'altra importante fase del lavoro svolto da Ghidotti è stata la revisione dell'architettura generale di Minerva. Si è così passati da uno struttura prevalentemente ad oggetti ad un sistema omogeneo, che utilizza il paradigma ad *agenti*. Un agente è considerato come una particolare entità software in grado di agire in modo autonomo percependo informazioni dall'ambiente in cui si trova. Esso agisce secondo la propria base di conoscenze, scambiando informazioni con altri agenti o con esseri umani e prendendo l'iniziativa per soddisfare i propri obiettivi. Il paradigma ad agenti implementato in Minerva05 risulta essere un approccio più adatto alle esigenze specifiche del contesto, offrendo una maggiore possibilità di sviluppo e di raffinamento successivo.
Le diverse prove sperimentali effettuate hanno dimostrato che i risultati ottenuti da Minerva05 sono sicuramente soddisfacenti e ciò grazie soprattutto alla stretta collaborazione avvenuta tra Ghidotti ed esperti del settore, che ha permesso di portare il software del Progetto Minerva più vicino alle esigenze degli allestitori museali. Inoltre il software propone un'interattività continua con l'operatore, al quale viene fornita ampia scelta di regole e cri-

teri di allestimento. Il processo perciò non si configura più come totalmente automatizzato ma anzi si avvale del fondamentale apporto di conoscenze e gusto artistico dell'utente.
Oltre a considerare i requisiti richiesti dagli allestitori, Minerva05 non trascura l'utente finale, il quale può effettuare una visita virtuale al museo appena creato: ciò è reso possibile dall'utilizzo di tecnologie che realizzano ambienti tridimensionali realistici.
Avvalendosi di tecnologie collaudate il sistema è robusto e stabile, fornendo una base concreta per le successive evoluzioni e permettendo a Minerva05 di adattarsi alle esigenze delle diverse committenze.

2.2 Musei virtuali con interfaccia web

Il concetto di museo virtuale si sta diffondendo sempre di più nella rete globale, ed è ormai diventato uno strumento frequentemente utilizzato nell'ambito di siti web dei musei. Se tradizionalmente questi visualizzavano solamente informazioni di utilità generale riguardanti il museo, come ad esempio orari, costi e descrizioni delle mostre, l'inserimento di nuove sezioni permette all'utente di esplorare virtualmente il museo stesso. Attraverso la visualizzazione di immagini e la lettura di schede dettagliate, l'utente può in questo caso entrare in contatto con le opere esposte nel museo senza visitarle fisicamente. Scopo di questa visita virtuale è quello di proporre un accesso alternativo al museo, che non vuole essere sostitutivo ma complementare, aumentandone la visibilità e il grado di interesse da parte del pubblico.
Le caratteristiche di queste sezioni aggiuntive sono molto variabili da sito a sito, essenzialmente secondo due parametri principali: il grado di interattività offerta e la veste grafica proposta. Nel caso più semplice la visita virtuale consiste nella consultazione di un elenco di opere semplice ed essenziale; in quello più complesso l'utente può creare il proprio itinerario in un ambiente tridimensionale.
Nei prossimi paragrafi sono brevemente esposti i diversi approcci seguiti nel fornire l'esperienza virtuale classificati in base al variare di questi due parametri, ottenendo quattro categorie principali:

- Catalogo,
- Tour virtuale,
- Museo virtuale,
- Museo virtuale creato dall'utente.

2.2.1 Catalogo

Uno degli approcci più diffusi, sia per la semplicità della realizzazione, sia per la facilità della consultazione, consiste nel fornire ai visitatori dei siti web una catalogazione basilare delle opere esposte tramite schede contenti immagini ed informazioni su ciascuna opera. Vadiamo alcuni esempi di questo tipo di soluzione.
Sul sito del Worcester Art Museum [10] tramite il link *Online Gallery* si accede ad una sezione in cui è possibile consultare le opere esposte raggruppate per periodi storici e provenienze geografiche differenti. Nella parte sinistra dello schermo è possibile scegliere quale gruppo di opere visualizzare oppure fare riferimento ad un motore per la ricerca di un'opera specifica, mentre nella parte centrale è visibile un'anteprima delle opere, con relativo autore, descrizione e datazione dell'opera. Cliccando sull'anteprima si accede ad informazioni più dettagliate dell'opera relativa, come mostrato in Figura 2.1.

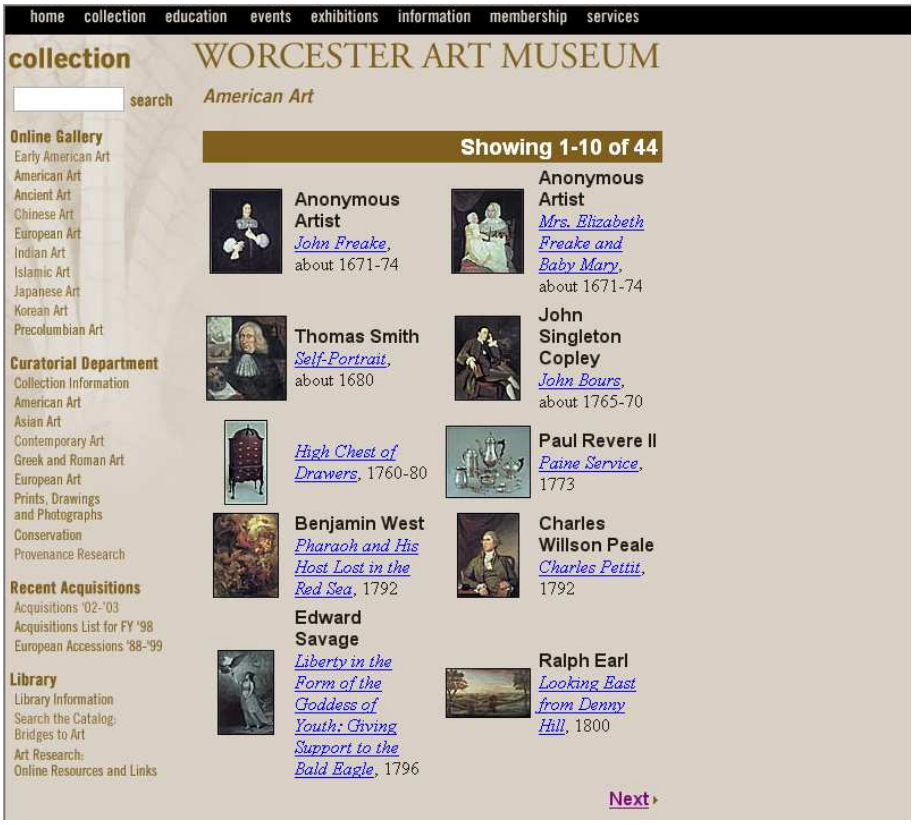

Figura 2.1: Pagine delle opere presenti nel sito del Worcester Art Museum

La consultazione delle gallerie che contano un numero elevato di opere è facilitata dai criteri di ordinamento dei risultati cronologico, per artista o per genere dell'opera. Nell'esempio in Figura 2.2 le opere sono visualizzate in ordine cronologico su un asse temporale orizzontale, al di sotto del quale sono riportati i principali avvenimenti storici di ogni periodo.

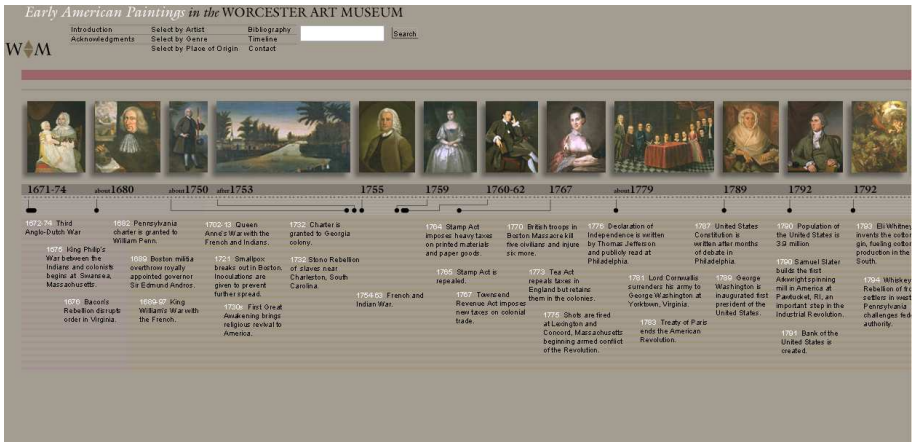

Figura 2.2: Ordinamento cronologico presente nel sito del Worcester Art Museum

Il sito della National Gallery [11] fornisce un altro chiaro esempio di catalogazione delle opere. In questo caso cliccando prima su *Collection* quindi su *Explore the collection online* si accede alla pagina mostrata in Figura 2.3 in cui sono meno schematici i criteri di ordinamento proposti, ma rimane sostanzialmente inalterato il concetto di catalogazione delle opere. Anche su questo sito è visibile un'anteprima dei risultati che porta ad una descrizione più dettagliata delle opere esposte. In Figura 2.4 è mostrata la pagina dedicata alle biografie degli autori delle opere più importanti esposte dal museo.

2.2.2 Tour virtuale

Altri siti web di celebri musei propongono un'interfaccia più dettagliata e coinvolgente rispetto ai casi visti in precedenza, anche se in ultima analisi utilizzano un approccio di catalogazione delle opere. Ciò che differenzia in questo caso l'esperienza dell'utente presso il sito è la modalità di accesso alle singole schede e la veste grafica che assume la visita. Solitamente gli sforzi dei realizzatori di questi siti web si concentrano nel rendere più realistico il percorso di raggiungimento delle pagine dedicate ai singoli oggetti: si potranno ad esempio trovare piantine del museo cliccabili, solitamente

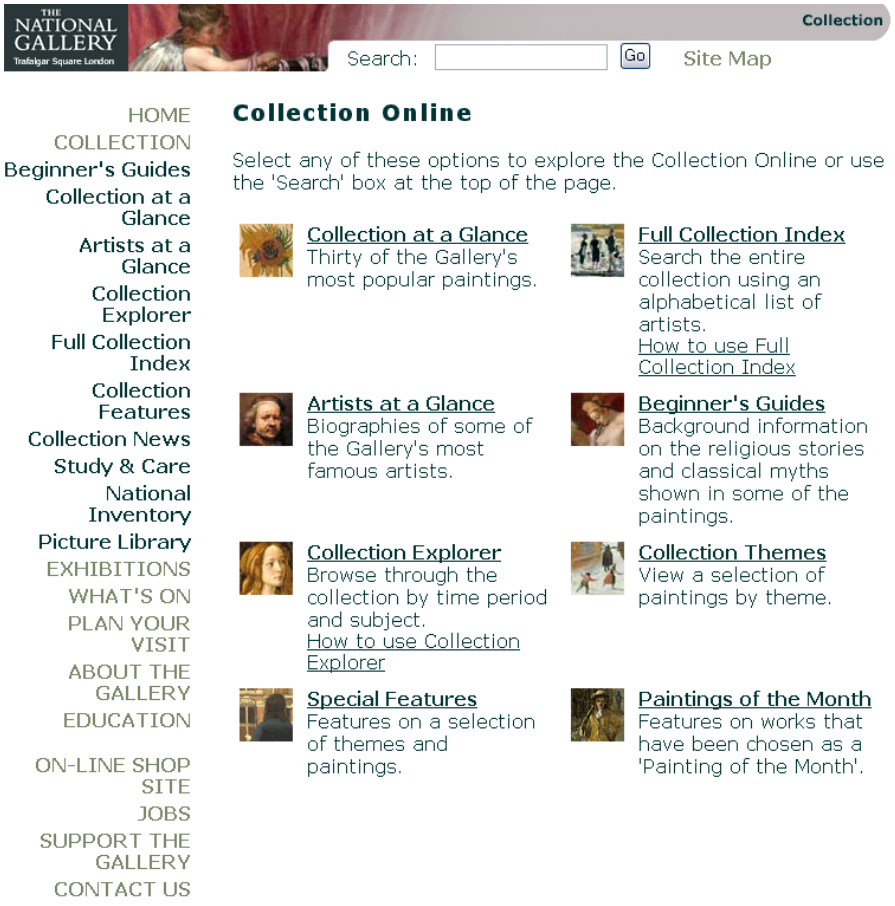

Figura 2.3: Pagina di accesso alle opere della National Gallery

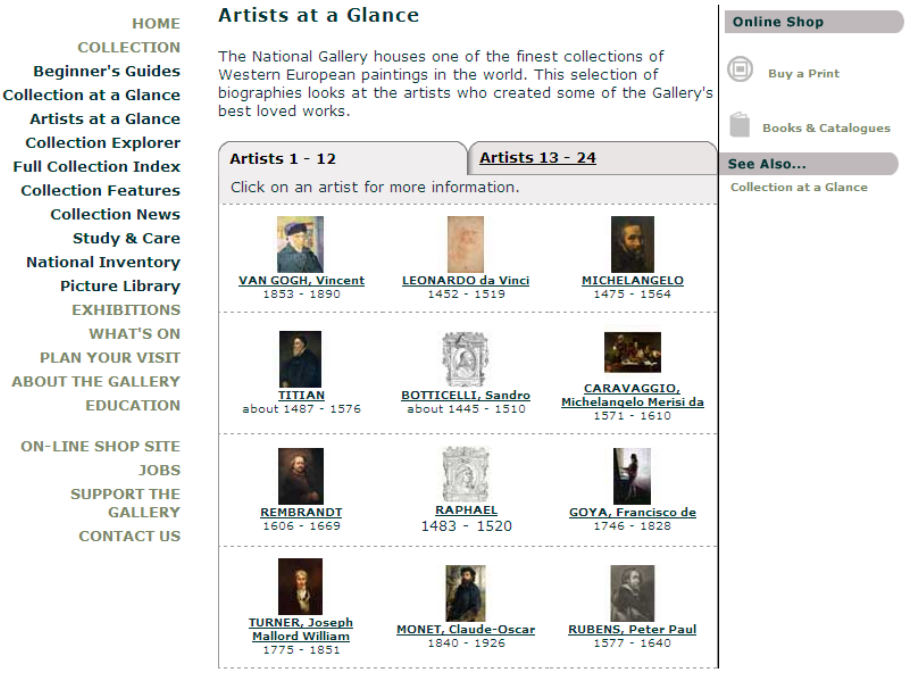

Figura 2.4: Collezione degli artisti presenti nella National Gallery

bidimensionali o in assonometria, dalle quali accedere ai cataloghi delle opere contenute nelle varie stanze. In questo modo si permette all'utente di familiarizzare con l'ambiente di esposizione, considerandolo non come un semplice contenitore ma come qualcosa che contribuisce a valorizzare ulteriormente l'esposizione. Un esempio di questo approccio ci viene offerto dal sito web dello State Hermitage Museum [12] nella cui sezione dedicata alle visite virtuali si accede alla lista dei piani del museo. Successivamente l'utente, selezionando un piano, ha la possibilità di consultarne una piantina con relativa legenda delle stanze in base alle opere ospitate. In Figura 2.5 vediamo un'immagine della piantina del pianterreno.

Un altro metodo per stimolare l'interesse del visitatore virtuale consiste nell'impiego di webcam, ossia di telecamere che ripropongono in tempo reale immagini tratte dall'interno o dall'esterno del museo. In questo modo si aumenta ancora di più il realismo dell'esperienza vissuta dall'utente, che dal proprio computer riesce ad osservare gli ambienti in cui si tengono le esposizioni. Un'applicazione pratica di questa tecnologia si trova ad esempio al Corvette Museum [13], dove sono state collocate oltre una decina di telecamere in diversi punti della struttura. Connettendosi al sito web del

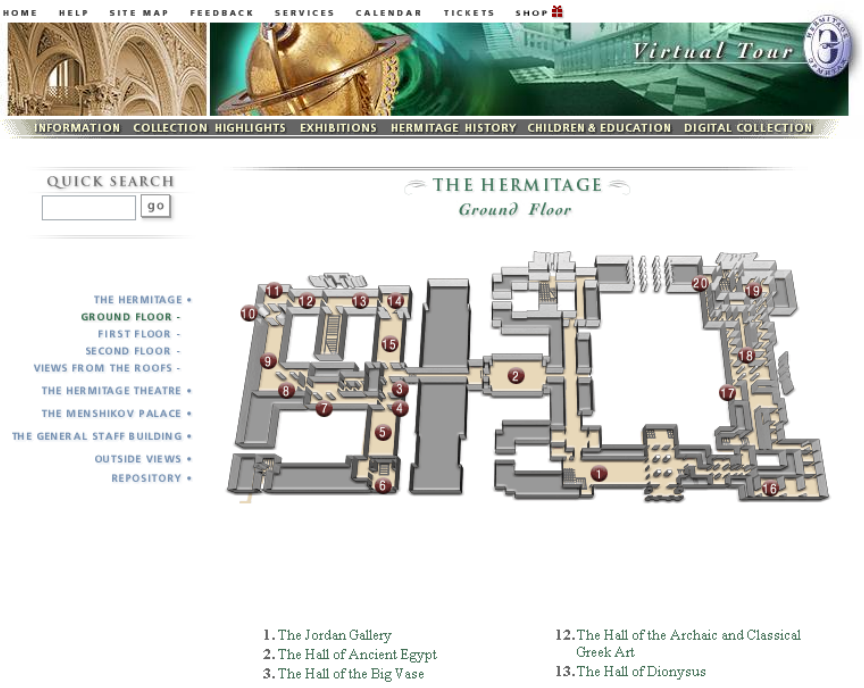

Figura 2.5: Sito dello State Hermitage Museum, piantina del pianterreno

museo e accedendo alla sezione dedicata alle webcam, è possibile osservare immagini delle esposizioni in tempo reale, come in Figura 2.6.

Un'altra interessante tecnologia è quella proposta da Apple denominata Quick Time Virtual Reality (QTVR) [14] che propone all'utente una visione panoramica a 360 gradi dell'ambiente circostante, come se si ponesse al centro di una stanza e ruotasse su se stesso. Solitamente è possibile anche simulare l'avvicinamento alle pareti della stanza tramite una funzione di ingrandimento dell'immagine ed avere una minima escursione verticale della visuale. Nonostante la limitata qualità delle immagini proposte e la minima capacità di spostamento, questa tecnologia riesce indubbiamente a immergere l'utente nell'ambiente reale di esposizione. Tipicamente viene utilizzata in ambienti di dimensione estesa, come stand e capannoni, dove ha senso parlare di visuale panoramica. Un esempio concreto di impiego di tale tecnologia si trova presso il sito web dello Yorkshire Air Museum [15] di cui vediamo due immagini della medesima esposizione vista da angolazioni differenti in Figura 2.7.

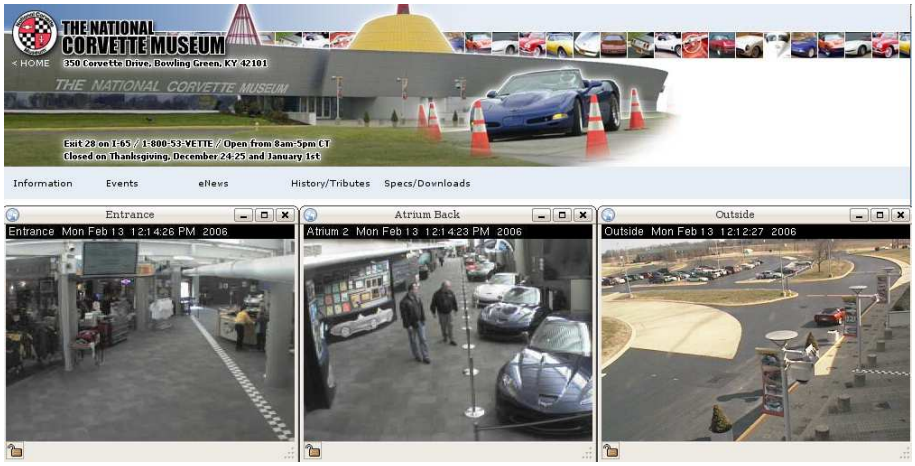

Figura 2.6: Le telecamere che visualizzano l'interno del Corvette Museum

2.2.3 Museo virtuale

Nei paragrafi precedenti sono state illustrate tecniche che tendono ad aumentare sempre più il coinvolgimento dell'utente e il realismo dell'esperienza offerta. Un ulteriore passo in questa direzione è offerto dai musei virtuali. In questo caso l'utente si trova immerso in un vero e proprio ambiente virtuale a tre dimensioni, che può riprodurre in maniera fedele il luogo dove si tengono fisicamente le esposizioni o rappresentarne un qualsiasi altro, realmente esistente o di fantasia. Anche le opere vengono ricostruite tridimensionalmente e collocate all'interno degli ambienti stessi, affinché siano disponibili per la visita virtuale. Il linguaggio più diffuso per questo tipo di applicazioni è Virtual Reality Modeling Language (VRML) [16], ideato per poter essere utilizzato in rete. Questo fa sì che i musei virtuali progettati con questo linguaggio siano accessibili semplicemente tramite browser, senza necessità di installare particolari applicativi lato client. Esempio di museo virtuale è il Virtual Tessellation Museum [17], che espone decorazioni a mosaico di vari artisti. In questo caso l'ambiente di esposizione non è la riproduzione di un luogo realmente esistente ma è fittizio. L'utente si trova in un atrio centrale (vedi Figura 2.8) e ha la possibilità di entrare in diverse strutture nelle cui stanze si trovano le opere suddivise per autore e periodi storici. In Figura 2.9 vediamo un'immagine presa da questo museo virtuale.

Figura 2.7: Tramite la tecnologia QTVR è possibile ruotare di 360 gradi la visuale attorno ad una posizione centrale

Figura 2.8: Veduta dell'atrio centrale del Virtual Tessellation Museum

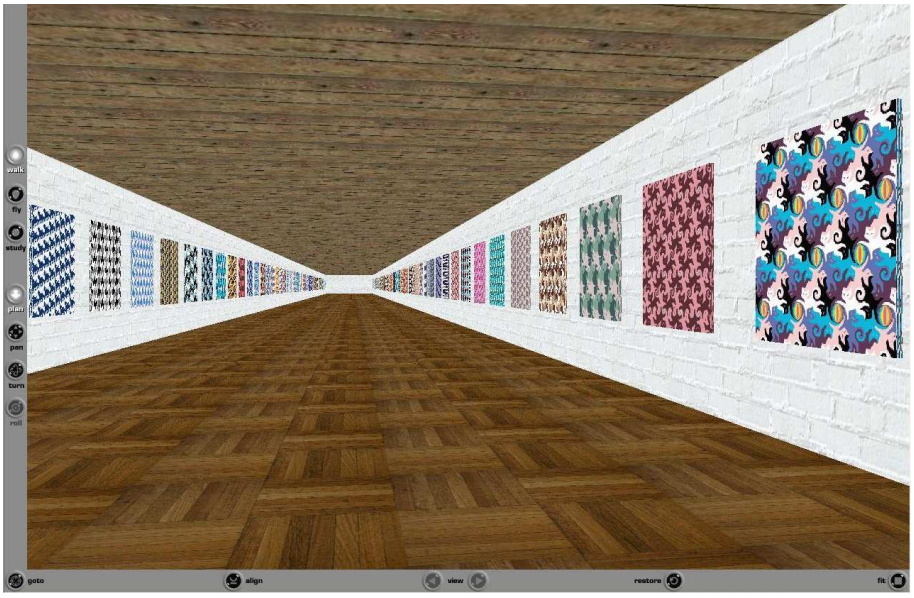

Figura 2.9: Interno di una stanza del Virtual Tessellation Museum

2.2.4 Musei virtuali creati dall'utente

Le tecniche finora analizzate conferiscono al visitatore un ruolo passivo, che pur potendo operare scelte più o meno ristrette, riceve informazioni così come gli vengono proposte nell'esperienza virtuale. La massima interazione fra utente e museo si ottiene attraverso un ribaltamento dei ruoli: il visitatore diventa protagonista attivo nell'operazione di creazione del museo, scegliendo le opere che vuole vedere esposte. L'esperienza virtuale dell'utente sarà quindi personalizzata in base alle sue richieste. Come abbiamo già avuto modo di notare, rientra in questa categoria anche l'oggetto della presente tesina. Un esempio di questo approccio ci viene offerto dal Virtual Museum of Canada [18], in cui l'utente dispone di una galleria personale alla quale può aggiungere opere scegliendole da una collezione di immagini provenienti da diversi musei del Canada. E' inoltre possibile aggiungere in un secondo momento commenti personali a ciascuna opera, in modo tale da renderne più completa la presentazione. Non è invece prevista la tridimensionalità: l'interfaccia segue il funzionale seppur semplice approccio a catalogo già visto in situazioni precedenti. In Figura 2.10 vediamo un esempio di collezione personale.

Figura 2.10: Collezione personale creata presso il Virtual Museum of Canada

2.3 Obiettivi tesina Andretta-Erba

Il lavoro "gemello" svolto da Andretta-Erba [2] è stato motivato dalla necessità di fornire al Comune di Ossuccio un applicativo da affiancare alla struttura fisica in costruzione, che permetta all'utente di creare un museo

virtuale in base a propri interessi e curiosità. Il software deve inoltre permettere agli utenti la fruizione di tutti i reperti rinvenuti presso l'isola Comacina sul Lario, parte dei quali sarebbe destinata a non essere esposta per mancanza di spazio.
Durante la fase di analisi dei requisiti i nostri colleghi hanno individuato le caratteristiche fondamentali del museo virtuale. Esso deve permettere all'utente, come già osservato, la possibilità di consultare reperti non esposti fisicamente e fornire nel contempo al visitatore informazioni aggiuntive che vadano oltre alla specificità del singolo reperto.
Una fase particolarmente importante è stata quella di analisi dei profili di utenza: tenendo conto di questi si è implementato il software in modo da favorire la semplicità di utilizzo, la responsività nella navigazione e l'essenzialità delle informazioni visualizzate.
Dalle analisi svolte è emersa la necessità di ottenere un software facile da utilizzare. Innanzitutto il museo virtuale deve produrre un risultato "utile" in non di più di due passaggi, perché percorsi più lunghi potrebbero disorientare il visitatore. Bisogna considerare che il generico utente utilizza Minerva per la prima volta, incontrando naturali difficoltà a familiarizzare col sistema, per cui si è reso necessario ridurre il più possibile la procedura di creazione del museo. La visita virtuale deve essere fluida e senza soluzione di continuità, in modo da agevolare l'interazione con l'utente e allo stesso tempo facilitare la personalizzazione del museo.
Un aspetto da non sottovalutare è la responsività del sistema, destinato ad essere utilizzato principalmente da un pubblico non esperto, che verrebbe scoraggiato da attese troppo prolungate. Bisogna valutare attentamente i tempi di risposta e di interazione, in modo da ottenere risultati apprezzabili in qualche secondo al massimo. Il progetto è orientato al web e quindi in prospettiva futura è necessario che agli usuali tempi d'attesa dovuti alla infrastrutture di rete non si accumulino eccessivi ritardi di elaborazione.
Particolare attenzione deve essere riservata alla scelta delle informazioni da presentare: esse devono essere concise ma complete, dando maggior importanza all'aspetto visivo legato alle immagini delle opere. Inoltre gli oggetti vengono accompagnati da brevi descrizioni testuali, non eccessivamente dettagliate o impegnative per il visitatore. Altrettanta attenzione va riposta alla coerenza tra le scelte effettuate dall'utente e i risultati prodotti da Minerva, in modo tale che egli possa verificare che le informazioni ottenute siano effettivamente quelle da lui richieste.
Nel lavoro svolto da Andretta-Erba è stata molto importante la stretta collaborazione avuta con gli esperti del settore, che hanno fornito validi suggerimenti per sviluppare e ampliare il software, rendendolo così aderente

alle loro aspettative. A tal fine si sono tenute alcune riunioni con il rappresentante della committenza, Prof. Della Torre, docente del Politecnico di Milano, nel corso delle quali si sono definite le problematiche e discusse le strade percorribili per affrontarle. Nei numerosi incontri con il relatore Ing. Francesco Amigoni, il correlatore Dott.ssa Viola Schiaffonati e l'Arch. Andrea Bonavita, si è via via approfondito e sviluppato nel dettaglio quanto emerso nei colloqui con il Prof. Della Torre, allo scopo di individuare soluzioni appropriate per lo sviluppo del software.

Capitolo 3

Analisi dei requisiti

In questo capitolo richiameremo i principali risultati ottenuti nell'analisi dei requisiti del lavoro di Andretta-Erba, per accennare a come ognuna delle singole caratteristiche del software da loro individuate possa essere effettivamente realizzata nella pratica. Nei prossimi capitoli analizzeremo in dettaglio la loro implementazione.

3.1 Usabilità

Uno dei requisiti fondamentali di Minerva è, come abbiamo già avuto modo di notare, l'usabilità del software. Secondo lo standard ISO 9241 l'usabilità è definita come *l'efficacia, efficienza e soddisfazione con cui specificati utenti raggiungono specificati obbiettivi in particolari ambienti* [19].
La definizione scompone l'usabilità lungo tre assi ortogonali, che definiscono tre proprietà meno sfuggenti (efficacia, efficienza e soddisfazione), e che possono in qualche modo essere misurate. L'efficacia viene definita, nello stesso documento, come *l'accuratezza e completezza con cui specificati utenti possono raggiungere specificati obbiettivi in particolari ambienti.* L'efficienza a sua volta è definita come *le risorse spese in relazione all'accuratezza e alla completezza degli obbiettivi raggiunti.* La soddisfazione, infine, è definita come *il confort e l'accettabilità del sistema di lavoro per i suoi utenti e per altre persone influenzate dal suo uso* e potrà essere misurata in vari modi, ad esempio mediante questionari atti a raccogliere le reazioni dell'utente al sistema.
L'usabilità va perseguita lavorando prevalentemente in due direzioni: progettare un'interfaccia chiara e semplice e limitare le procedure da seguire a pochi passaggi essenziali.

Per quanto riguarda l'interfaccia utente, la chiarezza è ottenuta attraverso la visualizzazione di informazioni concise ma complete, diversificate in base a colori ed altri elementi grafici. Questo ci ha spinto a non utilizzare le interfacce grafiche native del linguaggio di programmazione, potenti ma poco familiari al generico utente. La nostra scelta è invece ricaduta sull'utilizzo degli standard adottati dal *World Wide Web Consortium* [20]. Gli elementi grafici, come ad esempio collegamenti, bottoni radio e barre di scorrimento, sono familiari a quasi tutte le tipologie di utenti. Questa scelta ha comportato il collaterale ma non secondario vantaggio di ottenere un prodotto consultabile via Internet. Un'interfaccia così realizzata inoltre, può essere facilmente modificata e migliorata in un eventuale sviluppo futuro.
Il processo di creazione del museo deve essere piuttosto breve, e i risultati disponibili in pochi clic del mouse. Infatti l'eccessivo prolungarsi nel tempo di questa fase potrebbe confondere o scoraggiare l'utente finale, senza spingerlo ad utilizzare ulteriormente il programma.
Per questo motivo abbiamo progettato un numero ridotto di pagine ed organizzato le informazioni da visualizzare nelle stesse in maniera efficiente. In questo modo l'utente, partendo dalla pagina iniziale, dispone di un museo virtuale già alla terza schermata.
A tutto ciò risulta strettamente correlato il discorso sulle prestazioni del sistema, che influenzano i tempi di attesa percepiti. Discuteremo di questo argomento in maniera esaustiva nel Capitolo 6.

3.2 Collegamenti esterni

La fase di progettazione effettuata da Andretta-Erba ha previsto l'inserimento di collegamenti verso l'esterno, ossia la possibilità di accedere a siti web le cui tematiche siano strettamente connesse a quelle trattate dal museo virtuale. In questo modo si ampliano gli orizzonti del visitatore, arricchendo notevolmente i contenuti esposti durante la visita virtuale. L'unico accorgimento da tenere in considerazione è evitare che l'utente possa perdersi nella navigazione di uno spazio troppo esteso. A questo scopo, è necessario che il visitatore abbia in ogni istante la possibilità di tornare alle pagine del museo.
In termini di realizzazione pratica, collegamenti di questo tipo si sposano perfettamente con le nostre scelte progettuali. Avendo adottato gli standard del World Wide Web inserire link verso pagine di siti web esterni è stato estremamente facile. Per dare all'utente la possibilità di tornare al museo abbiamo suddiviso la pagina in due zone, secondo una tecnologia descritta nel prossimo capitolo. La parte principale, che occupa la porzione

rilevante dello schermo, ospita il sito web esterno mentre quella rimanente un collegamento per tornare alle pagine del museo.

3.3 Portabilità

Dalle analisi svolte nella tesina Andretta-Erba è emersa la neccessità di utilizzare software e linguaggi di programmazione altamente portabili. Per portabilità si intende la facilità di adattamento del programma, o del linguaggio a diverse piattaforme hardware. Questa è stata considerata una caratteristica fondamentale per il Progetto Minerva in quanto essa ne permette l'utilizzo sui principali sistemi operativi sfruttando a pieno le loro peculiarità. Ad esempio sarà possibile adottare *Linux* [21] come server sql sfruttandone a pieno robustezza e affidabilità, mentre potremo utilizzare *Mac OS* [22] lato client data la sua predisposizione alla facilità di utilizzo.
Il progetto Minerva ha adottato, fin dall'inizio, come linguaggio di programmazione *Java* [23] che di sua natura è un linguaggio multi piattaforma: esso è disponibile per *Windows* [24], Mac OS, Linux ed altri sistemi operativi sui quali, tuttavia, non abbiamo effettuato lo sviluppo. Si è quindi deciso di continuare la linea intrapresa dai nostri predecessori vagliando le tecnologie disponibili, scegliendo quelle che offrissero la maggiore integrazione con il software esistente e allo stesso tempo un'alta portabilità.
Per una trattazione dettagliata delle nostre scelte implementative si rimanda al capitolo successivo.

3.4 Affidabilità

Altro requisito evidenziato da Andretta ed Erba è l'affidabilità di Minerva. Un sistema si definisce affidabile se risulta privo di malfunzionamenti di natura sistematica dovuti ad errori di progettazione.
Il soddisfacimento di questo requisito critico ha influenzato tutta la nostra attività. In primo luogo, ci siamo basati su applicativi a loro volta affidabili e collaudati nel tempo. Inoltre l'architettura ad agenti, presente in maniera completa da Minerva05 in avanti, garantisce una maggiore stabilità del sistema. Il comportamento degli agenti risulta infatti facilmente prevedibile perché ognuno di essi ha un repertorio limitato di azioni molto semplici. Un agente è incaricato di controllare che in ogni momento tutti gli agenti siano presenti nel sistema e correttamente funzionanti.
Infine Java è un linguaggio di programmazione fortemente tipizzato, per cui riduce al minimo i casi di errore in fase di esecuzione del programma.

3.5 Concorrenza

Per offrire a più utenti la possibilità di visitare e creare il proprio museo virtuale il sistema deve essere in grado di rispondere a tutte le richieste pervenute simultaneamente. Questo è reso possibile da una corretta gestione della concorrenza da parte di tutti i componenti di Minerva. Nella tesina di Andretta-Erba si era riscontrato un problema per quanto riguardava l'utilizzo del programma da parte di più utenti contemporaneamente. L'anomalia produceva risultati non coerenti disorientando gli utenti con informazioni inaspettate. Nel Capitolo 5 verrà trattato in maniera approfondita la soluzione adottata per eliminare questo problema.

Capitolo 4

Progetto logico della soluzione del problema

In questo capitolo verranno in principio illustrate le tecnologie scelte a fronte dell'analisi dei requisiti effettuata, descrivendo successivamente i prototipi del programma realizzati: il primo semplice e dalla grafica essenziale, mentre il secondo più complesso e aderente alle aspettative della committenza.

4.1 Tecnologie adottate

Il Progetto Minerva è basato sul modello di un sistema distribuito, per cui abbiamo scelto applicativi in grado di dialogare tra di loro (vedi Figura 4.1). Questi applicativi possono eventualmente risiedere sulla stessa macchina. L'architettura è composta da un database, dal già esistente *Minerva Server*

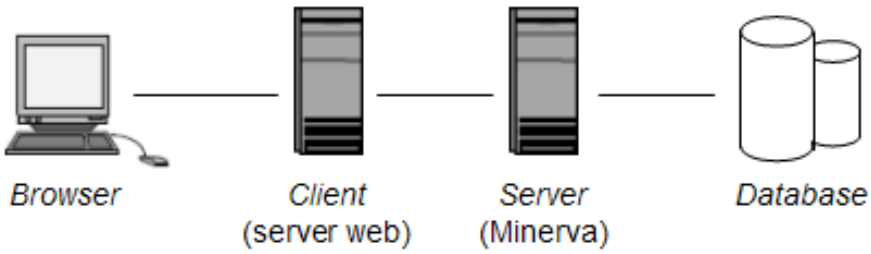

Figura 4.1: Possibile architettura del sistema distribuito

e dal web server, ai quali va ad aggiungersi il browser utilizzato dall'utente finale. Ogni software può essere dislocato su elaboratori differenti in quanto

le comunicazioni sono garantite dall'utilizzo di protocolli standard quali *Hypertext Transfer Protocol (HTTP)* [25] per la trasmissione delle pagine dal web ed il *TCP/IP* [26] per lo scambio di messaggi tra i componenti.
La possibilità di realizzare un sistema distribuito ci consente di utilizzare anche diversi sistemi operativi. Si potrà scegliere quello più adatto ad ottimizzare le prestazioni dell'applicativo in questione.

4.1.1 Database

Minerva05 utilizzava già precedentemente un database per memorizzare le informazioni necessarie alla creazione dei musei virtuali. Si trattava di *Microsoft Access* [27], che dopo le prime analisi è risultato essere disponibile solo per Windows. Inoltre il motore sql in questione poneva allo sviluppo del progetto diversi problemi riguardanti prestazioni e affidabilità.
Sono stati quindi fissati i requisiti fondamentali per il nuovo database: un robusto supporto alle transazioni garantendo le proprietà ACID (atomicità, consistenza, isolamento, persistenza) ed un elevato supporto alla concorrenza. Inoltre il nuovo software deve, non da ultimo, offrire ottime prestazioni per quanto riguarda la velocità di eleborazione delle richieste da parte dei client. Dopo un'attenta analisi dei software presenti sul mercato che offrissero le caratteristiche richieste, la scelta è ricaduta su *MySQL* della casa svedese *MySQL AB* [8]. Questo database inoltre contribuisce ad aumentare la scalabilità di Minerva. Questa proprietà permette di ampliare il sistema mantenendo l'utilizzo di MySQL in quanto è possibile adoperare hardware a basso costo. Infine un'altra caratteristica fondamentale di MySQL è la sua disponibilità sotto licenza *GNU GPL* [28]: oltre a garantirne il libero utilizzo ci permette di ridurre i costi nelle diverse installazioni di Minerva.

4.1.2 Minerva Server

Nell'implementazione delle precedenti versioni di Minerva, si è verificato un progressivo passaggio dal classico paradigma ad oggetti ad un'architettura multiagente. Questo processo è iniziato durante la stesura di Minerva98 e si è ultimato con il lavoro di Fabio Ghidotti, ossia Minerva05.
L'ambiente ad agenti scelto dai nostri predecessori, e successivamente da noi confermato, è quello offerto da *JADE (Java Agent DEvelopment framework)* [7], sviluppato da *TILab S.p.A* [29]. Questo strumento software, rispettando le specifiche *FIPA* [30], fornisce un supporto per la creazione di sistemi multiagente. Attraverso JADE si ha quindi la possibilità di concentrarsi sulla programmazione ad alto livello, senza doversi curare dei dettagli implementativi dei livelli sottostanti, come ad esempio le operazioni necessarie per la

creazione degli agenti o la trasmissione di un messaggio fra di essi.
Gli agenti già esistenti che abbiamo considerato anche nel nostro lavoro sono l'agente supervisore, chiamato "MyLittleServer", e il "DBManager". Il primo si occupa dello smistamento e dell'instradamento dei messaggi, mentre il secondo è responsabile dell'interazione con il database. Abbiamo invece deciso, in accordo con le analisi preliminari di Andretta ed Erba, di introdurre un nuovo agente, denominato "CheckAgent". Scopo di quest'ultimo è controllare che tutti gli altri agenti siano effettivamente esistenti e attivi all'interno del sistema, garantendo una maggiore stabilità e robustezza del server Minerva.
Nel Paragrafo 4.2.4 parleremo di come avvengono le comunicazioni all'interno del server Minerva, mentre nel prossimo capitolo verranno illustrati i molteplici vantaggi che derivano dall'adozione di una architettura ad agenti, nonché una dettagliata analisi della stessa.

4.1.3 Web server

Abbiamo già sottolineato come i requisiti di usabilità del software vengano soddisfatti attraverso la realizzazione di un'interfaccia utente amichevole e di come questa possa essere facilmente ottenuta sfruttando le tecnologie del web. Il componente software che si occupa della generazione delle pagine visibili all'utente è il web server. La scelta relativa a tale componente è stata particolarmente delicata, per via dei molteplici requisiti da soddisfare contemporaneamente. Era necessario individuare un prodotto altamente portabile, in grado di dialogare col server Minerva attraverso le tecnologie Java e in grado di generare in modo dinamico le pagine *HTML (HyperText Markup Language)* [3].
La scelta è così ricaduta su *Apache Tomcat* [5], della *Apache Software Foundation* [31], scritto completamente in Java ed aderente alle specifiche delle tecnologie *Java Servlet* [6] e *Java Service Pages* [32].
Grazie alle caratteristiche di questo pacchetto software, è stato possibile utilizzare JADE all'interno delle Java Servlet per le comunicazioni con il server Minerva. Infine le Java Service Pages potranno permettere in futuro un ulteriore sviluppo della piattaforma utilizzata, introducendo la separazione tra contenuto e visualizzazione e consentendo quindi a persone non esperte del linguaggio di creare pagine accattivanti.

4.1.4 Tecnologie Web

Nel capitolo precedente abbiamo esposto le motivazioni che ci hanno spinto ad adottare gli standard del World Wide Web per la realizzazione dell'in-

terfaccia utente. Le pagine create da Minerva sono espresse in linguaggio HTML e quindi fruibili tramite un comune web browser. Questo ci ha permesso di scorporare completamente la forma dal contenuto: le formattazioni grafiche delle informazioni destinate all'utente vengono aggiunte solo a lato client.
Altra tecnologia adottata, rivelatasi utile e potente, è quella offerta dai *Cascading Style Sheets (CSS)* [4]. Attraverso di essi è possibile definire la veste grafica delle pagine generate, come ad esempio i caratteri, i colori e la disposizione spaziale del testo. Modifiche ai CSS influenzano tutte le pagine che ne fanno uso, senza doverle considerare una alla volta.
Per la visualizzazione di risorse esterne al museo virtuale abbiamo infine utilizzato i *frames*, tecnologia poco diffusa nel web, ma perfettamente adatta al nostro scopo. In questo modo si possono visualizzare nella stessa pagina codici HTML provenienti da fonti diverse.

4.1.5 Strumenti di Sviluppo

Lo sviluppo del Progetto Minerva si è svolto collaborando con Andretta ed Erba al fine di ottenere un prodotto che soddisfacesse appieno la committenza. Questo ha portato alla creazione di un team di sviluppo composto da quattro persone nel quale la divisione dei ruoli non era rigida, permettendo ad ognuno di utilizzare le competenze specifiche al fine di ottenere il miglior risultato possibile.
La necessità di coordinamento tra gli appartenenti al gruppo di lavoro ha portato all'utilizzo di strumenti per lo sviluppo condiviso. Si è quindi deciso di utilizzare un sistema che permettesse di tenere traccia di tutte le modifiche apportate al codice sorgente. Si tratta di *Subversion* [33] che consente una facile gestione dello sviluppo parallelo ed un'ottima integrazione con gli altri strumenti software scelti. Inoltre è stato utilizzato un sistema integrato per la gestione del codice quale *Trac* [34] (in Figura 4.2) ed un *wiki* [35] dove poter condividere informazioni riguardanti il progetto. Infine per facilitare la comunicazione tra gli sviluppatori è stata creata una *mailing list* dove discutere le problematiche relative a Minerva.
Un'ulteriore importante scelta è stata quella relativa all'ambiente di sviluppo da adottare. Il software selezionato è stato *Eclipse* [36] (vedi Figura 4.3) in quanto offre pieno supporto a Subversion ed inoltre integra ottimi strumenti che velocizzano la scrittura del codice ed il debug dell'applicativo.
In Tabella 4.1 vengono riportate le versioni degli applicativi utilizzati nel Progetto Minerva che garantiscono il corretto funzionamento del sistema.

Figura 4.2: Trac System

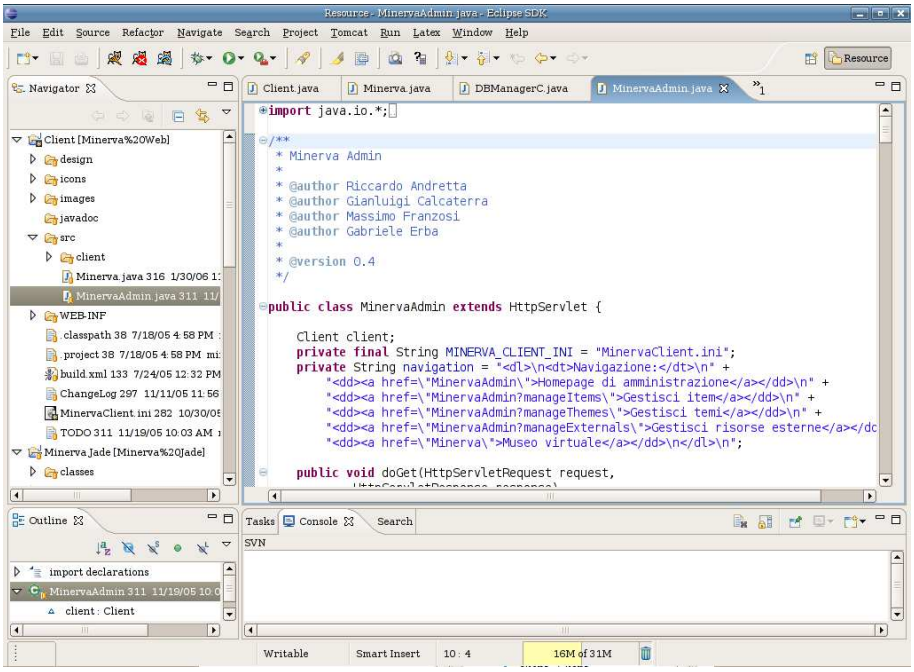

Figura 4.3: Eclipse

OS	JVM	Apache Tomcat	MySQL
Linux Ubuntu 6.04	1.4.2 1.5	5.0.28	4.1.15
Mac OS X 10.4.4	1.4.2 1.5	5.0.28 5.5.15	4.1.14 5.0.16
Windows XP Professional	1.4.2 1.5	5.0.28	4.1.14

Tabella 4.1: Versione dei software utilizzati

4.2 Prototipo

La fase preliminare di sviluppo del Progetto Minerva ha previsto la creazione di un prototipo, realizzato sulla base delle indicazioni da parte della committenza. Esso utilizza tutte le tecnologie sopra elencate, ma le soluzioni adottate sono volutamente semplici e basilari. Abbiamo progettato il prototipo allo scopo di raccogliere preziosi suggerimenti per gli sviluppi successivi.

4.2.1 Prototipo di carta

Alcuni esempi delle pagine del software visibili all'utente finale sono state schematizzate su fogli di carta. È stato quindi stabilito un ordinamento di tali pagine, arrivando a formulare la sequenza di passi da compiere per la creazione del museo virtuale. Un foglio succedeva l'altro, realizzando seppur in maniera primitiva una sorta di *dinamicità*. Il prototipo è stato presentato al Prof. Della Torre per una sessione di testing, durante la quale abbiamo provato a simulare la navigazione di un ipotetico utente all'interno del museo virtuale.
Questo tipo di lavoro ci ha permesso di validare le prime ipotesi relative all'interfaccia utente e, allo stesso tempo, di verificare la bontà della struttura concettuale delle pagine. Sono risultati importanti anche le opinioni e i giudizi espressi dal committente, permettendoci di acquisire importanti indicazioni per la creazione del primo prototipo.

4.2.2 Richiamo delle definizioni di item, temi e criteri

Nella tesina di Andretta ed Erba è stata descritta dettagliatamente la fase di progettazione di Minerva. Essa è stata innanzitutto caratterizzata da una forte e prolungata interazione tra progettisti ed esperti del settore, che ha

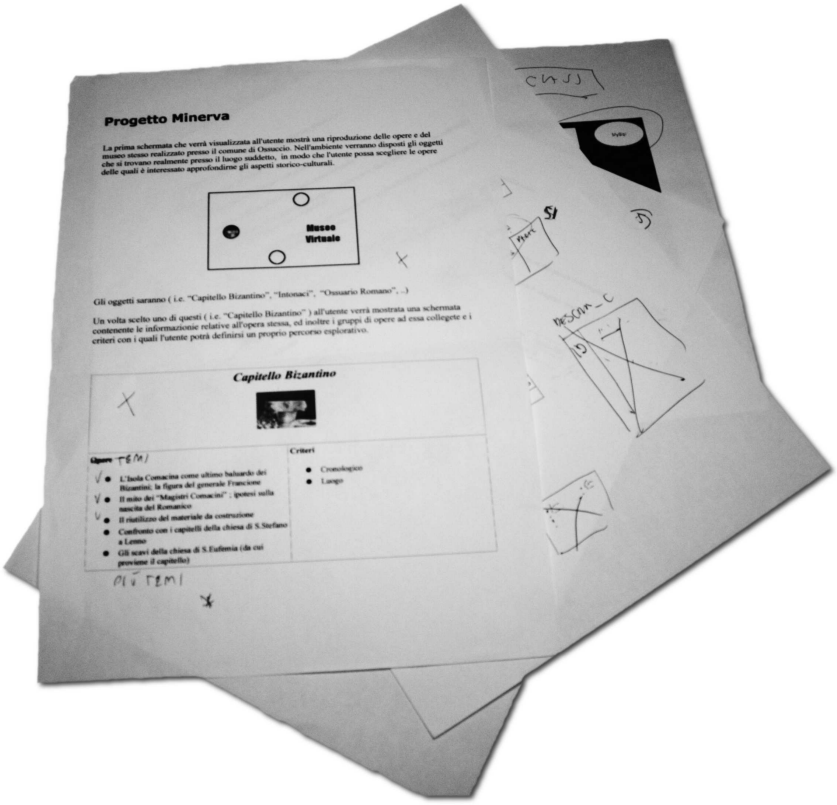

Figura 4.4: Prototipo di carta realizzato nella prima fase del progetto

portato alla realizzazione di un prodotto pienamente adeguato alle esigenze specifiche della committenza.
Durante questa fase si sono definiti i concetti di *item, temi* e *criteri* che richiamiamo brevemente. Con il termine item intendiamo l'astrazione di un oggetto qualsiasi. Nel nostro contesto gli item sono costituiti da reperti archeologici. Ognuno di essi è caratterizzato, nel primo prototipo, da proprietà autoesplicative, come nome, descrizione e priorità.
Per temi invece si intendono argomenti astratti che possono essere associati agli item. Se come tema consideriamo ad esempio l'impiego di un determinato materiale, ad esso saranno legate tutte le opere costituite da quel materiale. La corrispondenza esistente tra item e temi è di tipo "molti a molti", nel senso che un item può appartenere a più temi contemporaneamente e viceversa.
Fin dalle prime analisi della precedente tesina si è considerata l'ipotesi di organizzare il museo in stanze virtuali. Queste avrebbero poi potuto consistere in semplici riquadri contenenti le opere o, nel caso di una realizzazione grafica più complessa, in vere e proprie stanze tridimensionali. I criteri sono stati definiti appositamente per collocare le opere nelle varie stanze del museo virtuale. In fase di progettazione sono stati individuati quattro criteri (cronologico, topografico, tipologia di reperto e modalità di rinvenimento) secondo cui gli item possono essere suddivisi e raggruppati. Se si sceglie ad esempio il criterio cronologico, ad ogni stanza sarà associato un periodo storico e al suo interno verranno allocati gli oggetti di quel determinato periodo.

4.2.3 Database

Il progetto logico del database individua tre entità legate da due relazioni, come mostrato in Figura 4.5. Le entità del diagramma non sono altro che i concetti introdotti precedentemente. La coppia di attributi (name, description) costituisce una chiave primaria per l'item, in quanto assumiamo che non possano esistere due opere con nome e descrizione uguali. Per ogni item inoltre è prevista almeno una foto. Altro attributo dell'item è la priorità (*priority*) che rappresenta l'importanza relativa associata ad ogni oggeto.
La relazione *treatment* lega le entità item e tema: la cardinalità indica che si tratta di una relazione molti a molti, come già notato nel paragrafo precedente. L'altra relazione, chiamata *classification*, è fra l'item e il criterio (*criterion*) ed è caratterizzata da un preciso valore dell'attributo *value*. Un esempio pratico aiuta a chiarire il concetto. Consideriamo un qualsiasi reperto archeologico, come ad esempio un frammento di intonaco appartenen-

te al IV secolo d.c., rinvenuto nell'area San Giovanni dell'Isola Comacina. Se consideriamo il criterio topografico, l'attributo della relazione sarà caratterizzato dal valore "area San Giovanni dell'Isola Comacina", mentre se consideriamo il criterio cronologico, il valore dell'attributo sarà "IV secolo d.c.". Ogni item deve essere in relazione con tutti i criteri presenti. Non può esistere infatti un item che non sia classificabile secondo un certo criterio: un tale oggetto non sarebbe collocabile in una stanza precisa.

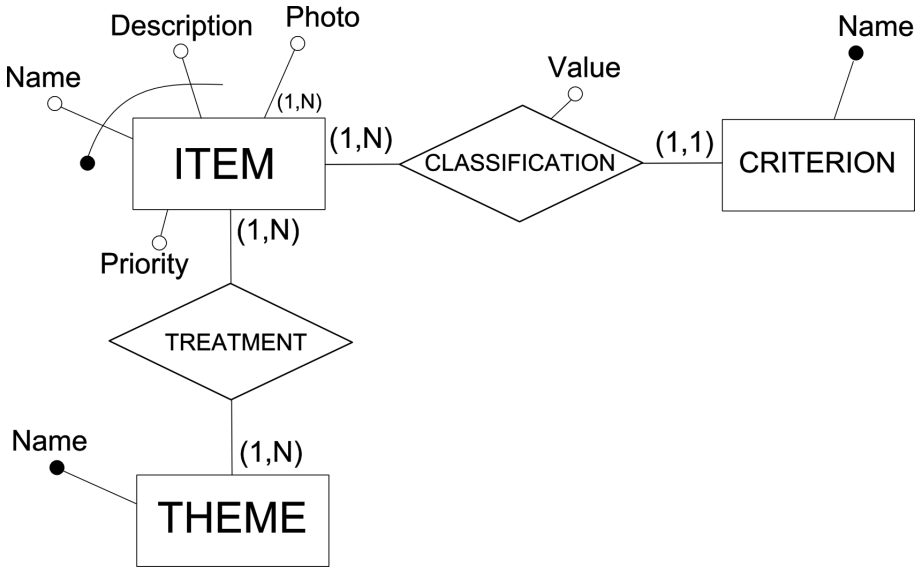

Figura 4.5: Primo progetto logico del database

4.2.4 Flusso e struttura dei messaggi

Come già accennato parlando di Minerva Server, gli agenti comunicano fra loro scambiandosi messaggi. Le informazioni scambiate possono essere di vario tipo, come ad esempio la richiesta di esecuzione di un'operazione o di dati. Potenzialmente ognuno di essi può comunicare con qualsiasi altro, per cui diagrammando l'andamento dei messaggi scambiati fra gli agenti si possono osservare più percorsi. È possibile tuttavia individuare un flusso di messaggi principale, per la gestione delle richieste dell'utente, il quale segue un andamento semplice e lineare. Questi messaggi provengono infatti dalla servlet e giungono all'agente MLS, il quale li inoltra a sua volta al DBManager. Quest'ultimo effettua le query necessarie al database, attende un risultato e lo rispedisce a MLS. Il percorso di ritorno è identico a quello di andata, ma ovviamente inverso.

Per quanto riguarda la struttura dei messaggi, ci siamo mantenuti affini a quella già esistente. Ogni messaggio è costituito da un cappello introduttivo, in cui ne vengono specificate la *performativa* e l'*ontologia*. Col primo termine si vuole indicare il tipo di messaggio in questione. Esempi tipici sono messaggi di richiesta (request), nei quali si richiede al destinatario di compiere un'azione, e di risposta (inform). Con il termine ontologia invece si vuole indicare l'informazione che permette ad un agente di interpretare il contenuto dell'informativa. Se la performativa del messaggio indica ad esempio una richiesta, l'ontologia permetterà all'agente di capire cosa è stato richiesto. Analizziamo un esempio pratico. L'agente DBManager riceve un messaggio di richiesta la cui ontologia è "getItems". Questo valore indica che il messaggio ricevuto è la richiesta di un elenco di item all'interno del museo. Le informazioni necessarie per effettuare la selezione degli item desiderati fra tutti quelli effettivamenti disponibili si trova nella parte rimanente del messaggio. In base a tutte queste informazioni il DBManager elabora una query da effettuare al database. Il messaggio di ritorno è caratterizzato da un'ontologia differente, nel caso specifico "items". Nella parte rimanente della risposta si trova l'elenco degli elementi richiesti. Il discorso è analogo per altri tipi di richiesta, come ad esempio quella illustrata in Figura 4.6.
I nostri sforzi si sono orientati alla ricerca della retrocompatibilità del nostro lavoro con quelli precedenti. Questo giustifica la scelta di soluzioni che apparentemente sembrano più complesse dei problemi posti. Si sarebbe ad esempio potuto evitare di utilizzare l'agente MLS accorciando il flusso dei messaggi, dato il ruolo marginale che esso svolge in questa fase. Questo sarebbe stato in contrasto con le precedenti versioni di Minerva, in cui MLS svolgeva un ruolo fondamentale di coordinatore e supervisore di tutti gli agenti del server. Nel caso di eventuali sviluppi futuri, che prevedono ad esempio l'inserimento di un nuovo agente, la presenza di un punto di riferimento centrale come MLS risulterebbe fondamentale.

In Figura 4.7 si può osservare la schermata che mostra gli elementi del museo virtuale raggruppati in due stanze. Si tratta ancora di una versione molto semplice e aperta a miglioramenti successivi.

4.3 Soluzioni dopo la fase di testing

Dopo la stesura del prototipo si è dato inizio ad una fase di analisi dei risultati ottenuti e della consistenza del lavoro con le specifiche iniziali. Questo è un tipico esempio di sviluppo del software secondo il *modello a spirale* [37], in cui a intervalli regolari si effettua un'operazione di verifica e raffinamen-

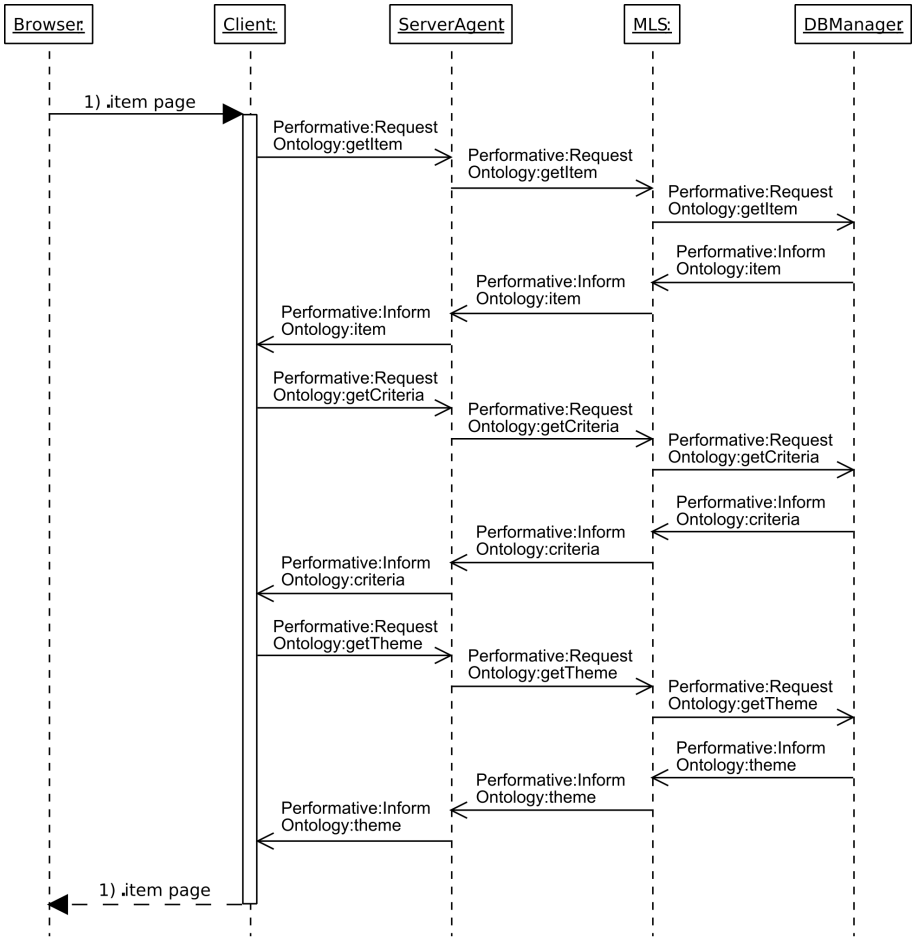

Figura 4.6: Sequence diagram della richiesta di una pagina

Figura 4.7: Schermata tratta dal primo prototipo

to del prodotto. I risultati esposti in seguito non sono stati raggiunti in un unico passo, ma sono il frutto di uno sviluppo incrementale seguendo il paradigma appena menzionato.

4.3.1 Definizione di teche e risorse esterne

Dopo un'attenta analisi del primo prototipo da parte della committenza, è emersa la necessità di facilitare ulteriormente la consultazione delle opere da parte dell'utente. Sono state così definite le *teche*, ossia raggruppamenti di oggetti dalle caratteristiche simili all'interno della stessa stanza. In questo modo l'utente può individuare immediatamente varie tipologie di oggetti esposti, e quindi organizzare la visita del museo in maniera più razionale. É possibile osservare una versione senza teche in Figura 4.8 e la versione definitiva in Figura 4.11.
Inoltre è stata proposta l'idea di approfondire le tematiche trattate dal mu-

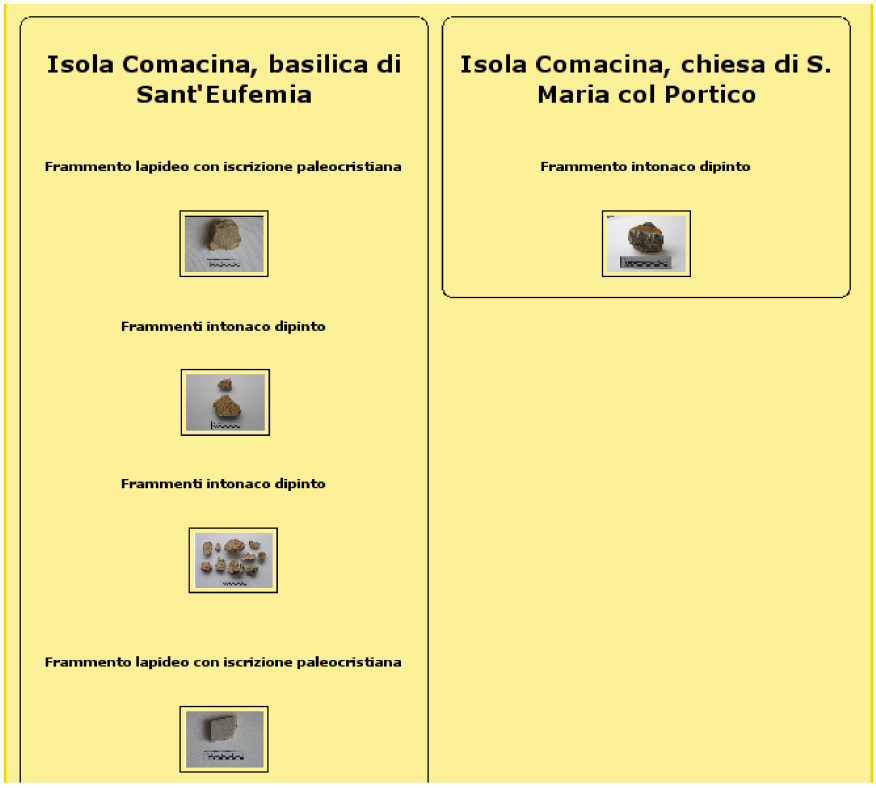

Figura 4.8: Pagina del museo virtuale con le sole stanze

seo mediante risorse provenienti da siti web esterni. Sono stati così definiti i *link esterni* (vedi Figura 4.9), come collegamenti che permettessero al-

l'utente l'accesso a tali risorse. L'implementazione di questa funzionalità del programma è già stata illustrata nel Paragrafo 4.1.4, trattando delle tecnologie Web.

Figura 4.9: Pagina dei link esterni

4.3.2 Database

Osservando il progetto logico della versione definitiva del database (vedi Figura 4.10), si può osservare come esso si basi sulla prima soluzione adottata, confermandone la bontà.
È stato aggiunto l'attributo *icon* all'entità item in maniera da associare una piccola icona ad ogni oggetto visualizzato nelle stanze del museo virtuale. L'attributo *description* dell'entità theme serve per aggiungere ad ogni tema selezionato una breve trattazione, in maniera da fornire all'utente le informazioni necessarie per la consultazione delle opere esposte. In tali descrizioni sono presenti i già menzionati link esterni per poter accedere a risorse non facenti parte del museo virtuale. A questo proposito sono state introdotte le entità *external theme* ed *external link*. La prima di queste identifica la risorsa vera e propria, caratterizzata da un nome, una descrizione ed un insieme

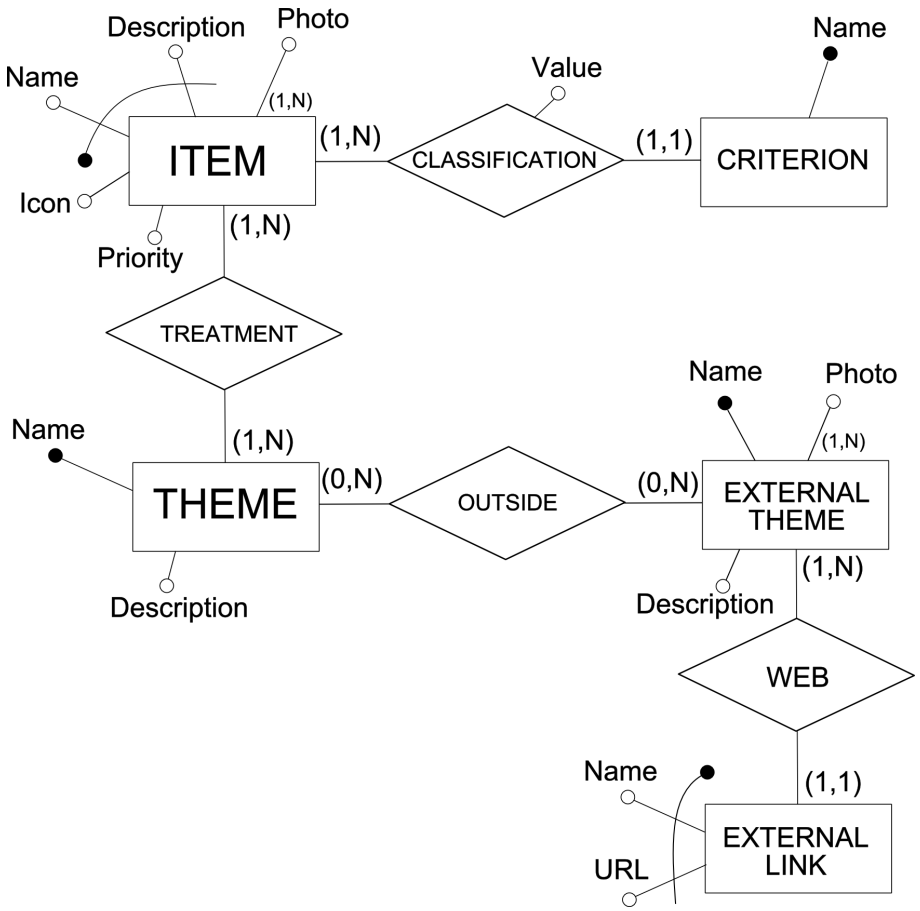

Figura 4.10: Progetto logico del database definitivo

di foto, mentre la seconda il link alla pagina di un sito web, individuato da un nome e relativa *URL* [38].
Le cardinalità delle relazioni che collegano le nuove entità indicano che nella trattazione di un tema può essere presente un numero arbitrario di risorse esterne. Per ognuna di queste a sua volta sono presenti uno o più collegamenti fisici, individuati da un nome visualizzato e la relativa URL.

4.3.3 Pannello di Amministrazione

Nel corso dello sviluppo del Progetto Minerva è apparsa evidente la necessità di fornire al manutentore della base dati quegli strumenti adatti all'inserimento, alla modifica e all'eliminazione degli elementi presenti.
Si è sviluppata un'interfaccia amichevole dai molteplici scopi: velocizzare le operazioni di manutenzione, minimizzare gli errori di inserimento dei dati e rendere accessibile la gestione del database ad utenti privi di conoscenze riguardanti le tecnologie delle basi di dati. Tale strumento risulta molto utile per tutte le operazioni che coinvolgono più di una tabella, delicate e tutt'altro che banali. Ne sono un esempio l'inserimento di fotografie e di link esterni, fasi in cui l'operatore è particolarmente assistito e guidato.
La soluzione tecnica adottata è stata quella di creare *MinervaAdmin* (vedi Figura 4.12), una servlet che effettuasse esclusivamente le operazioni atte a mantenere il database aggiornato e coerente. Tutte le operazioni sono state implementate utilizzando la struttura sviluppata per il museo virtuale e quindi utilizzando le peculiarità sia di Apache Tomcat, sia di JADE.

4.4 Ottimizzazioni

La fase finale del lavoro ha preso in considerazione le problematiche già individuate nella tesina Andretta-Erba, per le quali non si era tuttavia trovata una risposta. Nei paragrafi successivi verranno illustrate le principali soluzioni tecniche adottate per superare tali problemi.

4.4.1 Concorrenza

Nel corso dello sviluppo di Minerva si era tralasciato il problema relativo alla gestione delle richieste concorrenti, in quanto si era stabilito fin dall'inizio che ogni postazione collocata nel museo avesse installato una copia completa del sistema. Successivamente si è valutata la possibilità di poter utilizzare l'applicativo in maniera distributa provvedendo alla corretta implementazione della concorrenza.

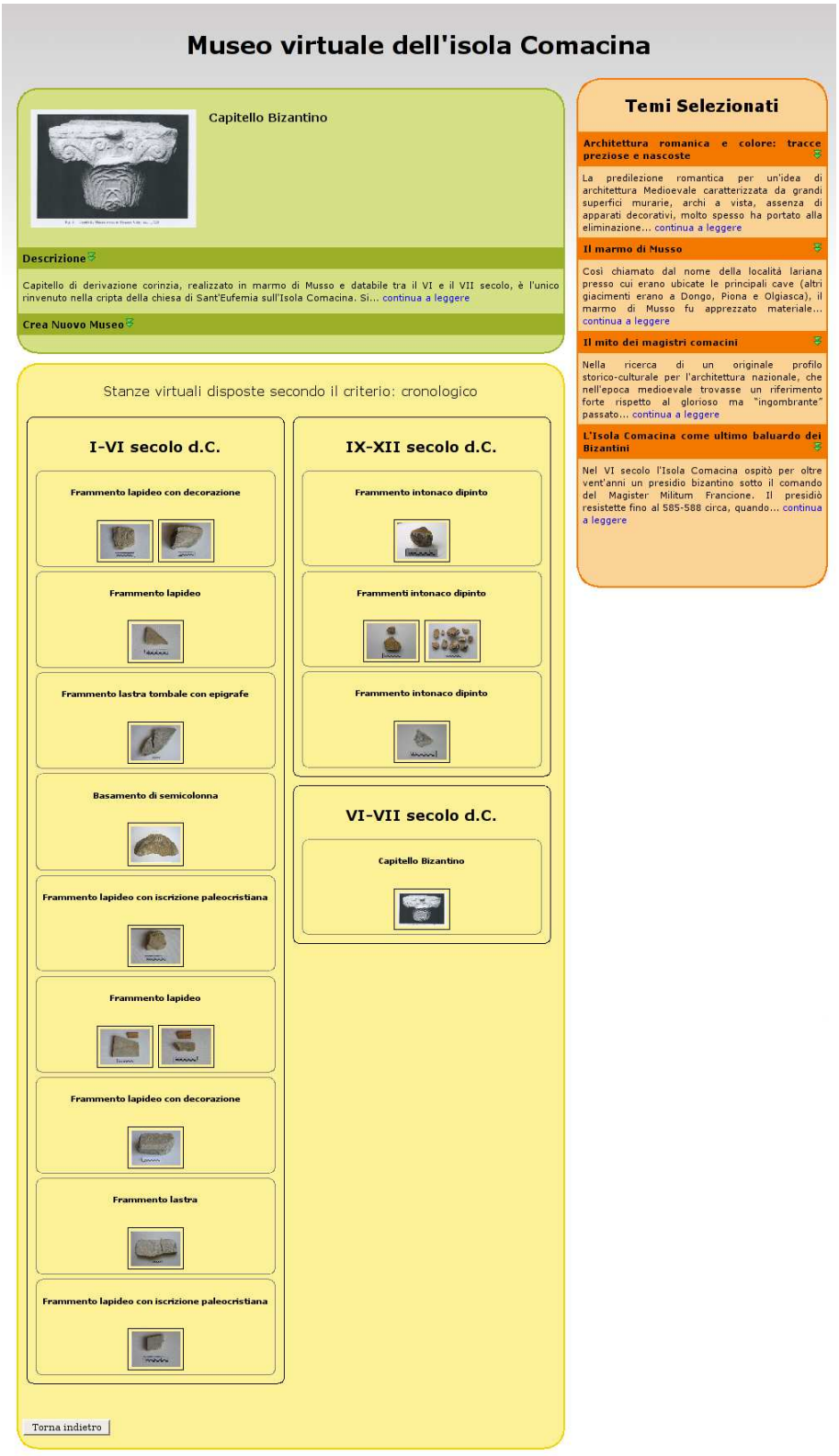

Figura 4.11: Una schermata della versione definitiva del museo virtuale

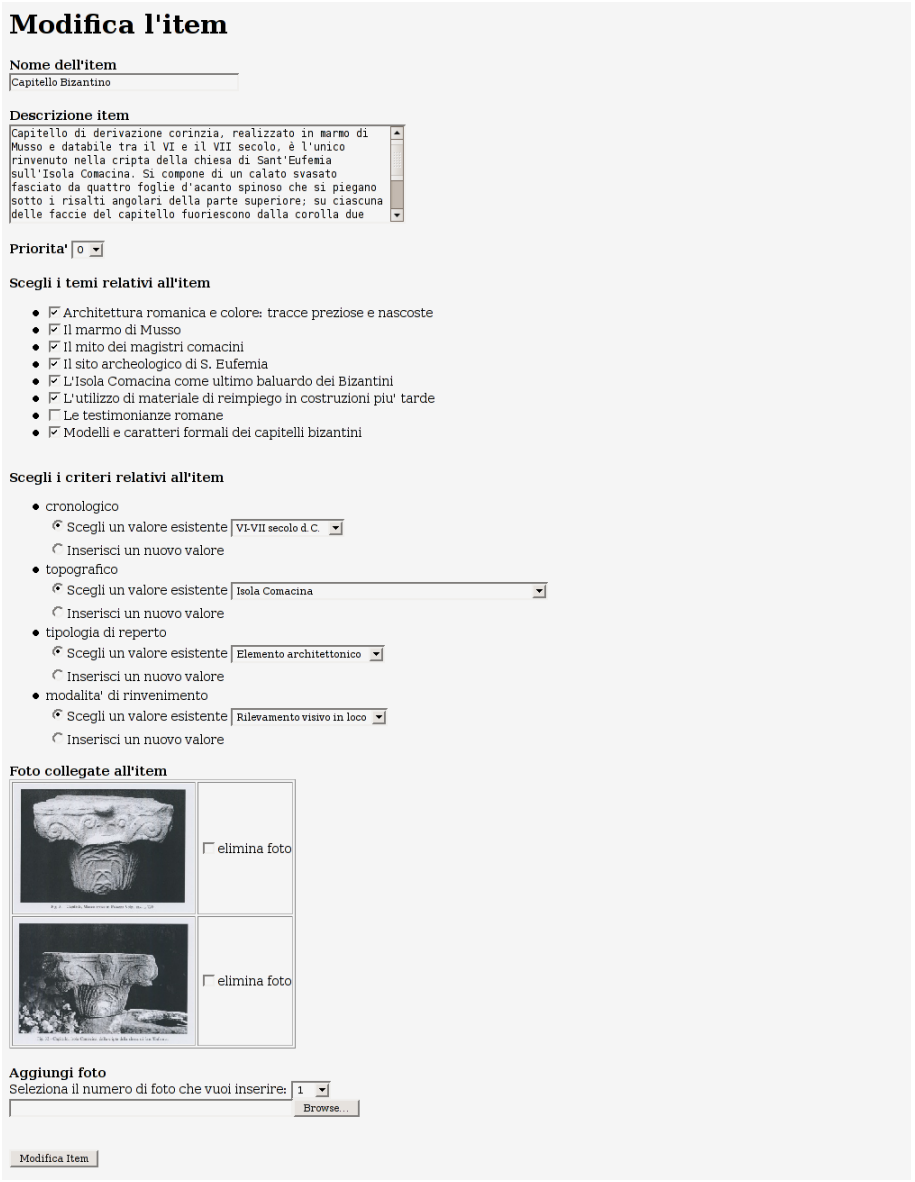

Figura 4.12: Pannello di amministrazione del Progetto Minerva

L'impossibilità di gestire correttamente più richieste simultane era causato dai messaggi *asincroni* utilizzati dagli agenti, mentre le richieste delle pagine web avvengono in modo *sincrono* in quanto la servlet deve ottenere tutti i dati necessari prima di inviare il contenuto al browser. Si sono quindi implementate delle chiamate bloccanti utilizzando dei semafori all'interno dell'agente situato presso il web server in modo che ogni richiesta aspettasse i dati richiesti prima di far proseguire l'esecuzione del programma. Questo è stato ottenuto creando delle operazioni atomiche in grado di garantire la consistenza tra le informazioni richieste e quelle pervenute da Minerva Server.

4.4.2 Prestazioni

Fin dall'inizio del lavoro si sono riscontrati alcuni problemi riguardanti i tempi di esecuzione.
Una prima analisi ha permesso di scoprire un errore di programmazione in Minerva Server all'interno di MLS: quando non vi erano messaggi in arrivo il carico del processore era del 100%. Questo era dovuto all'assenza di una chiamata che bloccasse l'agente e lo risvegliasse solo nel momento in cui fosse necessario eseguire delle operazioni. Implementando tale modifica si è ridotto drasticamente l'utilizzo dell'elaboratore consentendo un impiego prolungato dell'applicativo.
Durante la successiva fase di sviluppo ed implementazione dalle caratteristiche richieste dalla committenza si sono evidenziati notevoli differenze di prestazioni sui diversi sistemi operativi utilizzati. In Mac OS i tempi di risposta era più che accettabili, fornendo all'utente la pagina richiesta nell'ordine di qualche secondo. Al contrario sugli altri due sistemi operativi, Windows e Linux, l'applicativo impiegava un tempo superiore ai trenta secondi per visualizzare le pagine.
La prima ipotesi riguardante il problema era relativa alle comunicazioni tra gli agenti, dovuti ad una differente implementazione dello stack TCP/IP sui diversi sistemi operativi. Si è cosi sviluppato un procedimento per tracciare l'andamento dei messaggi all'interno di JADE che riportasse i tempi di percorrenza delle diverse richieste. Dalle prime analisi di questi dati era emerso che le informazioni rimanevano per un periodo di tempo elevato all'interno di MLS. Questo ci ha fatto ipotizzare che questo agente non avesse a disposizione le risorse necessarie per svolgere le proprie operazioni.
Analizzando i vari componenti che avrebbero potuto essere la causa di questo problema ci si è resi conto che la servlet monopolizzava le risorse di sistema per lunghi periodi di tempo, attendendo i risultati delle richieste

attraverso un ciclo vuoto. Si è quindi aggiunta una funzione che addormentasse la servlet per un lasso di tempo infinitesimo, ma sufficiente a rilasciare alcune risorse da rendere disponibili agli altri componenti. Questo ha portato ad una notevole diminuzione dei tempi di esecuzione delle richieste su tutti i sistemi operativi utilizzati. Lo sviluppo del progetto su diversi sistemi operativi ci ha permesso di individuare un problema di comunicazione tra componenti i cui sintomi sarebbero stati del tutto impalpabili, se avessimo considerato ad esempio il solo Mac OS.

Capitolo 5

Architettura del sistema

In questo capitolo viene illustrata l'architettura generale del sistema e le tecnologie utilizzate per la sua implementazione. Come si può notare in Figura 5.1 il sistema Minerva è composto da due parti principali connesse fra loro, *MinervaWebClient* e Minerva Server. Quest'ultimo è stato presentato nel capitolo precedente ed è stato ereditato delle precedeti versioni del progetto. Questo paragrafo è dedicato alla trattazione di *MinervaWebClient*, ovvero la servlet responsabile della creazione delle pagine HTML visibili all'utente finale. È necessario prima di tutto fare una piccola riflessione in merito al nome di questo componente. Può infatti sembrare strano che una servlet, che come abbiamo più volte asserito risiede sul web server, sia stata chiamata MinervaWebClient. Il ruolo della servlet nel nostro progetto è duplice: può essere vista come client perché effettua delle richieste al server Minerva, assumendo nel contempo le classiche vesti di un server nei confronti del web browser.
Il lato client interagisce con l'utente finale e il gestore del museo ed è composto da due servlet, Minerva e MinervaAdmin, e da un agente JADE. A lato server sono presenti gli altri agenti ed il database server. I primi due paragrafi trattano di risorse che vengono utilizzate da più entità software all'interno del sistema. In quelli successivi, partendo dal lato client fino a giungere al lato server, analizzeremo i singoli componenti e come questi si interfacciano fra loro.

5.1 Preferences

Per permettere la modifica di alcuni parametri di configurazione, anche durante l'esecuzione degli applicativi, è stato sviluppato un sistema per la

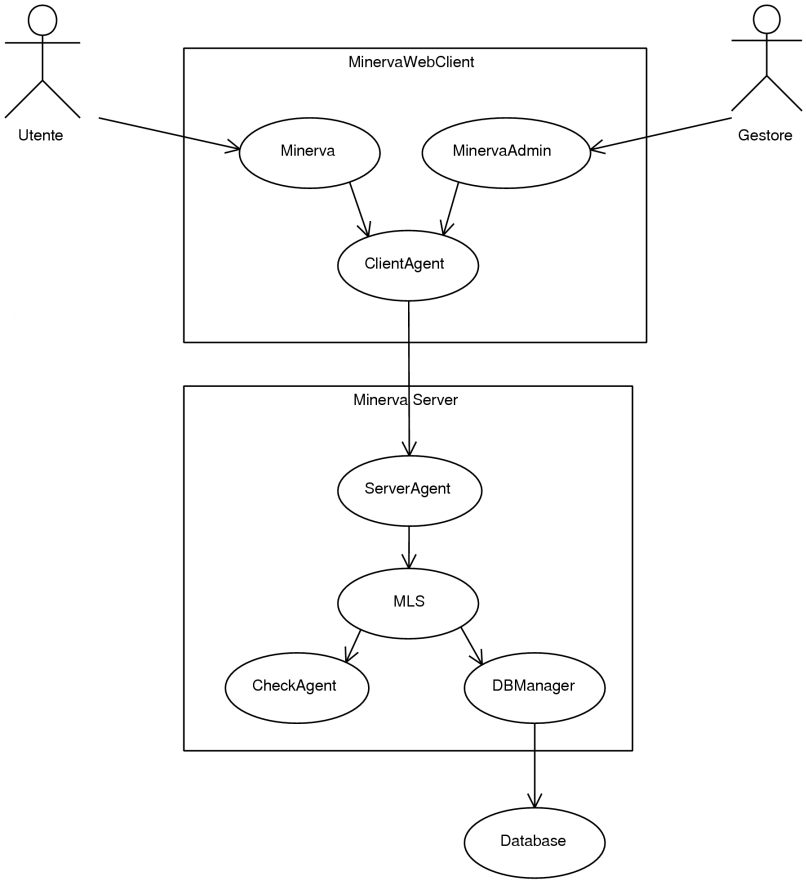

Figura 5.1: Diagramma use case del sistema Minerva

gestione delle impostazioni generali.
Basandoci sulla classe `java.util.Properties` [39] fornita dal linguaggio Java ne abbiamo sviluppata un'altra, *Preferences*, per la gestione delle diverse opzioni presenti in Minerva. Attraverso di essa possiamo utilizzare le impostazioni definite precedentemente nel file di configurazione, senza ogni volta compilare i sorgenti dell'applicativo. Minerva Server fa riferimento al file `Minerva.ini`, che contiene le impostazioni per quanto riguarda il database e il sistema di registrazione degli eventi. Inoltre è possibile specificare la lista degli agenti da attivare all'avvio del programma. Anche in MinervaWebClient le impostazioni sono memorizzate in un file chiamato `MinervaClient.ini`. In questo caso è necessario specificare l'indirizzo della macchina dove risiede Minerva Server per poterlo contattare e richiedere le informazioni da visualizzare all'utente. Sono presenti anche parametri aggiuntivi quali le porte da utilizzare e i percorsi per memorizzare le immagini caricate.
La classe sviluppata fornisce un comodo accesso alle diverse opzioni presenti nei file. Essa mette e disposizione metodi generici, come `get()` e `get_boolean()`, per recuperare valori memorizzati, ma anche più specifici quali `getCreateAgent()` per individuare gli agenti da attivare.

5.2 MinervaLog

Durante lo sviluppo del Progetto Minerva si è riscontrata la necessità di poter registrare delle informazioni riguardanti le azioni compiute dagli agenti. Tali informazioni erano indispensabili sia per la fase di debug dell'applicativo sia per monitorare costantemente lo stato del sistema.
Sfruttando le potenzialità messe a disposizione da Java abbiamo sviluppato la classe *MinervaLog*, sia per Minerva Server sia per MinervaWebClient. Essa utilizza le funzionalità presenti in `java.util.logging.Logger` [40], permettendo una facile e coerente gestione delle informazioni raccolte.
Il sistema creato prevede due differenti log: uno per il server, denominato *MinervaLogServer*, e l'altro per il client, *MinervaLogWebClient*. Questa distinzione permette una migliore gestione delle registrazioni, differenziandole in base ai due principali componenti architturali di Minerva.
La classe Logger permette di associare ad ogni log diversi gestori per quanto riguarda l'output: ne abbiamo così utilizzati due differenti. Il primo permette di registrare tutte le informazioni pervenute in un file di testo, mentre il secondo effettua la stampa dei messaggi direttamente nel terminale di sistema. L'impiego dei gestori non è mutuamente esclusivo ed è quindi possibile attivarli contemporaneamente. La formattazione delle informazioni avviene

impostando un'apposita funzione chiamata dal gestore. Java mette a disposizione la classe `java.util.jogging.SimpleFormatter` [41] che crea un log con le informazioni essenziali e comodamente leggibile. Un'alternativa a questa modalità è quella di utilizzare il formato *Extensible Markup Language (XML)* [42], contenente tutti dati registrati dal Logger: questa soluzione risulta essere molto potente per un'analisi dettagliata del comportamento degli applicativi software.
MinervaLog mette a disposizione dello sviluppatore due metodi differenti per registrare le informazioni sui log. Uno di questi è `info()`, da adoperare per i messaggi a carattere generale riguardanti, ad esempio, lo stato degli agenti o il loro comportamento. L'altro metodo disponibile è `debug()`, consigliato per tenere traccia dei dati necessari alla validazione dell'applicativo.
Grazie all'utilizzo di Preferences (vedi Paragrafo 5.1), il sistema di registrazione può essere abilitato anche durante l'esecuzione degli applicativi e inoltre le opzioni precedentemente esposte sono configurabili senza l'intervento diretto sul codice sorgente. Con l'aggiunta di MinervaLog si è fornito agli applicativi un comodo sistema di gestione delle informazioni di controllo.

5.3 MinervaWebClient

MinervaWebClient è composto dalle servlet *Minerva* e *MinervaAdmin*, che si appoggiano a *ClientAgent*, un agente JADE.
Minerva implementa tutte le pagine del museo, a partire dalla homepage, fino ad arrivare alla pagina delle stanze e dei link esterni. Queste pagine sono *dinamiche*, ossia create in fase di esecuzione basandosi sui dati che giungono dal server Minerva e sulle richieste dell'utente. I metodi utilizzati sono i tradizionali `doGet()` e `doPost()`, che differiscono fra loro in base al modo in cui avviene il passaggio di parametri. Una così netta separazione delle funzioni chiamate secondo queste due categorie contribuisce a mantenere il codice molto chiaro e leggibile. L'implementazione di queste funzioni risiede su un altro modulo, chiamato *Client*, e risulta quindi trasparente alla servlet, a vantaggio dell'*information hiding* fra i componenti. ClientAgent è invece un agente lato client responsabile della comunicazione con il server Minerva, inviando messaggi di richiesta e ricevendo le relative risposte.
All'interno di MinervaWebClient, oltre al già citato linguaggio HTML, sono presenti righe di codice *ECMAScript (Javascript)* [43] per implementare alcuni effetti grafici. Ad esempio è possibile nascondere o visualizzare oggetti come form e porzioni di testo.
Sempre lato client è collocata un'altra servlet, che adopera le stesse tecnologie illustrate nel paragrafo precedente. Questo componente non è visibile

all'utente finale, in quanto si tratta di uno strumento di gestione e manutenzione del museo virtuale. Per questo motivo ci si è maggiormente concentrati sull'usabilità di questo componente piuttosto che sullo sviluppo grafico delle sue pagine.
I metodi utilizzati sono, come per MinervaWebClient, `doGet()` e `doPost()`. Anche MinervaAdmin si appoggia al modulo esterno Client che implementa le funzioni chiamate nel codice sorgente. E sempre in analogia col caso precedente sono presenti porzioni di codice in ECMAScript per rendere le pagine più interattive e comprensibili all'utente.
Attraverso MinervaAdmin si possono gestire tutte le entità individuate nei capitoli precedenti, come gli item, i temi, i criteri e le risorse esterne. Le operazioni permesse sono quelle tipiche, come l'inserimento, la modifica e l'eliminazione dei dati. Scopo di MinervaAdmin è guidare il manutentore del museo virtuale passo dopo passo, facilitando tutte le fasi di gestione e riducendo al minimo le possibilità di inserire valori errati o inconsistenti con la base di dati.

5.4 Minerva Server

Minerva Server è composto da più agenti JADE. Quelli utilizzati nel nostro lavoro sono ServerAgent, MyLittleServer (MLS), DBManager e CheckAgent. Il primo di questi si occupa di tenere traccia delle comunicazione tra MinervaWebClient e lo stesso server in modo da garantire il corretto flusso di messaggi tra i componenti.

5.4.1 MyLittleServer

In questo paragrafo verrà analizzato l'agente MyLittleServer (MLS), il componente fondamentale del server Minerva. Come già accennato, MLS è stato ereditato dalle versioni precedenti del progetto, ma la sua reingegnerizzazione è stata quasi totale.
Come prima cosa è stata utilizzata la classe Preferences (vedi Paragrafo 5.1) dalla quale recuperare i parametri e le impostazioni iniziali necessari all'agente. In questo modo la gestione di questi valori, oltre a essere più rapida, è accessibile anche ai meno esperti in campo di programmazione. Non è più necessario infatti intervenire sul codice sorgente e ricompilarlo.
Compito di MLS è attivare gli altri agenti all'avvio del programma. Se nelle precedenti versioni di Minerva l'attivazione veniva effettuata su tutti gli agenti, adesso è possibile specificare il sottoinsieme di quelli che vengono considerati. Se alcuni di questi sono inutili all'esecuzione, possono essere

esclusi ottenendo un apprezzabile miglioramento delle prestazioni.
A MLS è stato invece revocato il compito di controllare che tutti gli agenti attivati siano effettivamente esistenti e correttamente funzionanti per tutta l'esecuzione del programma. Questa attività è stata devoluta ad un agente ad hoc, chiamato CheckAgent. Il controllo viene ora effettuato ad intervalli di tempo regolari, e non più al transito di un messaggio in MLS. In questo modo si evita che sia aggiunto altro carico in caso di traffico intenso di comunicazione fra i singoli componenti e si ottiene un sistema ancor più robusto.
Il *behaviour* di MLS, ovvero il suo comportamento, è stato scorporato dal codice vero e proprio dell'agente creando un modulo separato. Si ottengono così codici più chiari e suddivisi in base alla loro utilità. Nel modulo relativo al behaviour si trova la porzione di codice per la ricezione e lo smistamento dei messaggi. Questa parte di MLS, che in principio replicava più volte le stesse righe di codice per ogni tipologia di messaggio, è stata notevolmente migliorata. Il messaggio in arrivo viene ora confrontato attraverso un algoritmo di ricerca binaria, implementato dalla funzione `Arrays.binarySearch()` [44], con due array contenenti tutti i possibili messaggi. Il primo array contiene i messaggi provenienti dal client destinati al DBManager, mentre il secondo quelli in verso opposto. In base al risultato della ricerca negli array, MLS inoltra il messaggio nella direzione corretta. In questo modo, se si aggiungono nuove possibili destinazioni, è sufficiente inserire un nuovo elemento negli array, senza scrivere righe di codice per la gestione specifica.

5.4.2 DBManager

L'agente DBManager è responsabile dell'interazione fra il server Minerva e il database. Il suo compito consiste nel formulare al database query in linguaggio sql sulla base delle richieste ricevute da MLS e di restituire un messaggio contenente i risultati dell'interrogazione. Questo agente è l'unico componente che conosce lo schema del database, mascherandone a tutti gli altri la struttura. Si realizza così l'information hiding tra i componenti software. Se si modifica ad esempio l'implementazione del database, sarà sufficiente modificare il DBManager senza intervenire sugli altri agenti. L'interazione tra DBManager e il database è stata realizzata mediante un driver fornito dalla stessa casa produttrice di MySQL, chiamato *MySQL Connector* [45].
Anche questo agente utilizza la classe Preferences (vedi Paragrafo 5.1) per la gestione delle impostazioni di connessione. Attraverso di essa vengono recuperati l'indirizzo del server sql, il nome utente e la password.

Il nostro lavoro riguardo a questo agente si è concentrato maggiormente sulla scrittura delle interrogazioni specifiche per il database relativo al museo del Comune di Ossuccio.

5.4.3 CheckAgent

Al fine di garantire la presenza di tutti gli agenti nel sistema si è sviluppato un metodo di controllo che li riattivasse in caso di errore.
Il controllo degli agenti avveniva in precedenza all'interno di MLS al passaggio di qualsiasi messaggio, causando un leggero ritardo nella trasmissione delle informazioni. Per questo motivo abbiamo creato appositamente l'agente *CheckAgent* per la gestione di tale attività, alleggerendo il carico di lavoro di MLS ed aumentando allo stesso tempo la robustezza del server Minerva. Il comportamento dell'agente in questione è volutamente semplice, in quanto deve esclusivamente verificare la presenza degli altri agenti ed eventualmente ripristinarne lo stato.
CheckAgent sfrutta il metodo `doWait()` disponibile in `jade.core.Agent` [46], una classe presente in JADE, per effettuare ad intervalli regolari di tempo il controllo degli agenti. Per individuare in maniera efficiente quali di questi sono attivi viene utilizzata la classe `jade.domain.DFService` [47]. A partire dagli agenti attivi è poi immediato ricavare per differenza quelli caduti e ripristinare la loro esecuzione.

5.5 Database

A seguito del completamento della definizione logica (vedi Paragrafo 4.3.2) si è proceduto alla realizzazione degli schemi della base di dati, nella quale vengono memorizzate tutte le informazioni necessarie al corretto funzionamento dell'applicativo.
Prima di tutto si è definita la tabella contente gli oggetti presenti nel museo:

`items(id,name,description,priority,icon)`

Successivamente si è pensato a come memorizzare i temi e criteri sulla base dei quali creare il museo virtuale. Si è deciso di utilizzare solamente la seguente tabella per entrambi gli elementi, considerando il tema come un "criterio speciale".

`criteria(id,name)`

Questa tabella contiene quindi, oltre ai già citati criteri cronologico, topografico, tipologia di reperto e modalità di rinvenimento, anche il valore "tema". Ad ognuno di questi elementi è associato un id numerico per renderlo univoco. Per convenzione, assegnamo all'id dell'elemento tema il valore "1". Tutti i record delle altre tabelle che sono in relazione con l'elemento "1" di questa tabella si riferiscono in realtà a un tema e non a un criterio.
Dalle analisi effettuate risulta che un item è associato a tutti i criteri presenti e a un certo numero di temi correlati. Abbiamo quindi provveduto alla creazione di una tabella che collegasse in modo opportuno gli elementi e contenesse il valore del criterio o del tema.

`items_criteria(id_item,id_criterion,value)`

Infine si è definito lo schema per le foto associate ai diversi oggetti presenti nella base di dati.

`photos(id,photo)`

Il campo id corrisponde all'omonimo della tabella items, mentre nel campo photo è contenuto il nome dell'immagine visualizzata dal web browser.
La necessità di fornire una descrizione esaustiva dei temi ha portato alla creazione di una tabella apposita contente le informazioni da visualizzare all'interno del museo virtuale.

`themes(name,description)`

Per quanto riguarda i collegamenti alle risorse esterne sono state definite, sempre facendo riferimento al progetto logico, due ulteriori tabelle. La prima di queste, *external_themes*, contiene nome e descrizione del tema esterno.

`external_themes(id,name,description)`

La seconda invece, *external_link*, associa a ognuno di questi temi un insieme di collegamenti a siti Internet contenti materiale di approfondimento riguardanti le risorse esterne.

`external_link(id,name,url)`

L'ultima tabella definita archivia le foto da associare ai temi esterni. La sua struttura è identica a quella della tabella già definita per le foto relative agli item.

external_photos(id,name)

5.6 Thread Java

L'architettura sviluppata con JADE utilizza principalmente i thread per gestire l'esecuzione simultanea degli agenti. La specifica di Java è volutamente molto lasca nel caso della schedulazione dei thread, proprio per lasciare l'implementazione il più possibile al sistema operativo. Per questo motivo ci possono essere variazioni significative del comportamento e soprattutto delle prestazioni. In generale il programmatore fa riferimento ad una regola empirica, secondo cui un thread non deve monopolizzare l'esecuzione del programma. Se per troppo tempo (qualche decimo di secondo) non viene eseguita un'operazione potenzialmente bloccante, come una lettura, una scrittura o una sleep, il thread deve bloccarsi spontaneamente.
In MinervaWebClient la servlet che attende le informazioni da ClientAgent effettua un'operazione di controllo continuo su una variabile condivisa con il behaviour dell'agente, dove quest'ultimo inserisce le informazioni ricevute da Minerva Server. Questa operazione creava, su alcuni sistemi operativi quali Linux e Windows, un notevole rallentamento. Il problema è stato risolto introducendo una pausa di qualche millisecondo tra due operazioni di controllo adiacenti utilizzando il metodo `Thread.wait()` [48]. L'intervallo di breve durata aggiunto impedisce ad un thread di monopolizzare le risorse del sistema, permettendo di eseguire le richieste in modo responsivo in tutti i sistemi operativi testati.

Capitolo 6

Realizzazioni sperimentali e valutazione

In questo capitolo verranno riportati alcuni giudizi e valutazioni sul risultato espresso sia da parte della committenza, sia da parte degli sviluppatori. Partendo dai numerosi incontri effettuati con l'Arch. Andrea Bonavita, analizzeremo come questi siano stati determinanti al fine di presentare un prodotto che soddisfacesse appieno la committenza. Successivamente illustreremo le principali qualità possedute dal software, come l'efficienza, la tolleranza ai guasti e le prestazioni.

6.1 Test Bonavita

La collaborazione con l'Arch. Andrea Bonavita è stata proficua sotto molteplici punti di vista. In primo luogo il materiale da lui reperito ci ha dato la possibilità di concretizzare un progetto che si stava sviluppando solamente in linea teorica, rendendo quindi più chiari sia gli obiettivi e i requisiti da soddisfare sia le scelte tecnologiche da adottare. I risultati in fase di test hanno acquistato una maggiore leggibilità, perché strettamente inerenti all'oggetto del lavoro, evidenziando in maniera più marcata i legami fra dati inseriti e output generato e consentendoci di verificare l'affidabilità e la consistenza del software. L'inserimento dei dati riguardanti tale materiale si è svolto in maniera incrementale, a fianco dell'attività di sviluppo vera e propria, conferendo così al museo virtuale un numero ragguardevole di opere e tematiche trattate. L'interazione con Andrea Bonavita ha permesso inoltre un costante sviluppo dell'interfaccia utente, sia dal punto di vista grafico, sia nella scelta delle informazioni da visualizzare. Ad ogni riunione abbia-

mo infatti raccolto le sue impressioni nel testare il programma, cogliendo le differenze fra ciò che esso effettivamente visualizzava e ciò che un ipotetico utente finale avrebbe voluto vedere: tutto questo era possibile solamente con una persona che avesse una buona conoscenza del progetto e della materia in questione ma che fosse sufficientemente slegata dagli aspetti tecnologici di Minerva. Subito dopo la riunione stessa procedeva ovviamente uno studio di fattibilità delle nuove modifiche stabilite per analizzarne la compatibilità con le scelte già adottate in precedenza. Tra le principali caratteristiche dell'interfaccia utente sulle quali si è maggiormente lavorato ricordiamo:

- l'essenzialità, ottenuta attraverso la visualizzazione di poche informazioni chiare e precise;
- la diversificazione delle informazioni riportate in base all'utilizzo dei colori;
- l'inserimento di collegamenti esterni, che offrono all'utente la possibilità di visitare siti internet che presentano argomenti correlati ai risultati ottenuti, ampliando notevolmente gli orizzonti di navigazione del programma.

Per una più dettagliata descrizione dell'interfaccia grafica di Minerva si rimanda al Capitolo 4. Il risultato della collaborazione con Andrea Bonavita è stato il raggiungimento di un pacchetto software sufficientemente robusto ed utilizzabile, tale da poter essere presentato alla committenza attraverso una demo che ne evidenziasse tutte le potenzialità.

6.2 Test finale

Dopo aver svolto il lavoro di sviluppo con l'Ing. Amigoni, la Dott.ssa Schiaffonati e l'Arch. Bonavita, il progetto del museo virtuale dell'isola Comacina è stato sottoposto a verifica presso il nostro committente.
Alla presenza del Prof. Stefano Della Torre sono state illustrare tutte le funzionalità implementate nei mesi precedenti. Nella fase iniziale si è realizzato un museo virtuale partendo dal capitello, l'elemento di spicco del museo, e spiegando passo dopo passo le operazioni e le scelte effettuate. Sono state evidenziate tutte le potenzialità della pagina dedicata alle opere, motivando le nostre scelte grafiche e tecnologiche. Inoltre è stato illustrato il comportamento del software alla richiesta di visualizzare i collegamenti esterni e come questo assista l'utente in questa fase esplorativa. Si è mostrato come attraverso l'uso dell'applicativo sia possibile anche per l'utente inesperto

scoprire delle relazioni tra gli oggetti del museo difficilmente ipotizzabili a priori senza un adeguata preparazione in materia.
La stretta aderenza dei risultati da noi ottenuti con le specifiche iniziali ha fatto sì che la committenza fosse pienamente soddisfatta del lavoro svolto. Tutto ciò è stato possibile grazie all'utilizzo di metodologie tipiche dell'ingegneria del software, quali lo sviluppo secondo il modello a spirale, la suddivisione in team del gruppo di lavoro e la continua collaborazione con un referente della committenza.

6.3 Efficienza generale

Il Progetto Minerva è stato sviluppato utilizzando tecniche collaudate nell'ambito dell'ingegneria del software in modo da ottenere un prodotto efficiente, ovvero che fornisca dei risultati il più possibile accurati e completi con il minino utilizzo di risorse.
Il metodo di sviluppo adottato ha seguito il tipico modello a spirale. Questo si distingue rispetto ad altri metodi per offrire un miglior approccio al problema e un risultato pienamente aderente alle specifiche iniziali, grazie alla forte interazione tra programmatori e committenti. Il metodo si suddivide in quattro fasi principali, che si susseguono ciclicamente: al termine della quarta fase, si ritorna alla prima.

- Identificazione degli obiettivi e delle alternative;
- Valutazione della alternative e delle potenziali aree di rischio;
- Sviluppo e verifica del prodotto;
- Revisione dei risultati delle tre fasi precedenti.

Per ottimizzare i tempi di sviluppo si è deciso di suddividere i partecipanti al progetto in gruppi. Questo ha permesso che ognuno di questi affrontasse di volta in volta le problematiche emergenti nelle fasi di verifica. Tale suddivisione ha creato i presupposti per svolgere più attività in parallelo e quindi ridurre i tempi di sviluppo. Come precedentemente analizzato nel Capitolo 4 il lavoro dei team è stato ben supportato attraverso applicativi che ne facilitassero lo sviluppo parallelo e la condivisione delle informazioni.
Nella fase di implementazione sono stati utilizzati alcuni metodi per migliorare la qualità del codice scritto. Il primo, conosciuto come *pair programming* [49], consiste nel lavorare in coppia alla stessa postazione in contemporanea: una persona scrive la porzione di codice concentrandosi sui dettagli di basso

livello, mentre l'altra controlla la coerenza globale di quanto scritto. Naturalmente i ruoli sono interscambiabili in modo da permettere ad entrambi di portare le proprie conoscenze specifiche in materia. Sono stati anche svolti *test di unità* [50] e *test di regressione* [51]. I primi esaminano il comportamento di un singolo componente, mentre i secondi considerano tutta l'applicazione. Grazie a queste verifiche si garantisce che il progetto non si discosti mai dai requisiti della committenza. Durante la stesura del codice è stata ampiamente utilizzata la tecnica del *refactoring* [52], che consiste nel riscrivere parti del programma in modo da semplificarne il codice senza variarne le funzionalità. Anche dopo queste operazioni, i risultati dei test rimangono integri.
Determinante per un corretto sviluppo è stato l'utilizzo del metodo dell'information hiding, che riduce i costi da pagare per eliminare gli errori in fase di scrittura del programma. Questo risultato viene ottenuto strutturando l'intero progetto, e i moduli che lo compongono, in modo che un'errata decisione presa nell'implementazione della singola parte non si ripercuota sul tutto, e possa essere corretta modificando soltanto il modulo in questione. Si può così evitare di dover modificare anche i moduli clienti, che interagiscono con gli agenti soltanto attraverso interfacce. Ogni agente risulta essere autonomo, dialogando con gli altri attraverso protocolli standard di comunicazione.

6.4 Tolleranza ai guasti

Come abbiamo più volte accennato, i nostri sforzi si sono orientati alla creazione di un prodotto robusto ed affidabile. Un sistema si definisce tollerante ai guasti, o *fault tolerant*, se è in grado di operare correttamente, eventualmente con un degrado delle prestazioni, a fronte di un malfunzionamento di uno dei suoi componenti. Un sistema siffatto deve quindi essere in grado di individuare il malfunzionamento, confinarlo in maniera tale che gli effetti negativi non si ripercuotano su altre parti, ed infine ripristinare la situazione di funzionamento ordinaria.
Tra le procedure di ripresa individuiamo due categorie principali. La prima fra queste è la *ripresa a freddo*, se il ripristino è ottenuto riavviando il sistema. Nel caso in cui invece il componente malfunzionante venga sostituito senza interrompere l'esecuzione si parla di *ripresa a caldo*. Il vantaggio offerto dalla prima procedura consiste nella semplicità della sua implementazione. Sfortunatamente però tutti i sistemi che adottano questa soluzione non sono in grado di garantire un servizio all'utente sempre disponibile. La ripresa a caldo invece, a fronte di un'eventuale degrado delle prestazioni,

garantisce in ogni momento il servizio all'utente.
Nel nostro contesto, è possibile che un componente software, come ad esempio un agente, incontri una situazione di malfunzionamento. Questo può essere dovuto ad un errore di programmazione in Minerva o nei livelli software sottostanti. Per quanto riguarda la gestione degli agenti, ne abbiamo introdotto uno, CheckAgent, con il solo scopo di verificare istante per istante lo stato di funzionamento degli altri agenti. Per una trattazione riguardo alla sua implementazione consultare il capitolo precedente. Quando CheckAgent individua un agente caduto, ne crea immediatamente un'altra istanza, per cui effettua una ripresa a caldo. In questo caso si verifica un leggero degrado delle prestazioni di Minerva dovuto alle operazioni di creazione del nuovo agente.
Lo sviluppo del sistema Minerva utilizzando un paradigma distribuito permette una migliore gestione di eventuali guasti che si dovessero verificare in ognuna delle componenti. I tre moduli:

- MinervaWebClient
- Minerva Server
- Database

comunicano tra di loro attraverso protocolli standard e risultano essere indipendenti uno dall'altro. Nel momento in cui fosse necessario riavviare un componente, le altri parti possono rimanere in esecuzione. Si ha così una migliore gestione dello sviluppo dell'applicativo: è possibile lavorare separatamente su ogni singolo componente e riavviare solo i moduli interessati. Ad esempio nel caso di modifiche in MinervaWebClient è sufficiente far ripartire la servlet all'interno di Apache Tomcat senza intervenire sugli altri componenti.

6.5 Analisi delle prestazioni

In questo paragrafo analizziamo le prestazioni della versione finale del sistema Minerva sui differenti sistemi operativi utilizzati. Come abbiamo già avuto modo di notare, l'ottimizzazione del codice sorgente e una più efficiente gestione dei thread ha permesso un notevole miglioramento delle prestazioni sui sistemi Linux e Windows, ora dello stesso ordine di grandezza di quelle di Mac OS X.
Per Linux e Windows ci siamo basati sulla medesima piattaforma hardware, mentre per Mac OS X su una macchina dalle caratteristiche tecniche equivalenti, come mostrato nella Tabella 6.1.

	Mac OS X	Linux	Windows
Versione	10.4.5	Ubuntu 6.04	XP Pro
Macchina	Apple Powerbook	ASUS M 3000	
Processore	G4 1,5 GHz	Intel Pentium M 1,5 GHz	
Cache	512 KByte	1 MByte	
RAM	1,2 GByte	512 MByte	
Hard Disk	Hitachi Travelstar 5K100 60 GByte, 5400 RPM	Hitachi Travelstar 40GN 40 GByte, 4200 RPM	

Tabella 6.1: Caratteristiche tecniche degli elaboratori utilizzati

Sulle macchine di prova sono stati installati tutti software necessari per un corretto funzionamento di Minerva, e in particolare riportiamo in Tabella 6.2 le versioni di tali progammi.

OS	JVM	Apache Tomcat	MySQL
Linux	1.5	5.0.28	4.1.15
Mac OS X	1.5	5.5.15	5.0.16
Windows	1.5	5.0.28	4.1.14

Tabella 6.2: Versione dei software utilizzati nei test

Precisiamo inoltre che questa non vuole essere un'esaustiva analisi statistica delle prestazioni del sistema, che richiederebbe un elevato numero di campioni ed un approccio più scientifico del problema, ma un'indicazione generale sui tempi di attesa visibili all'utente. Innanzitutto è stata determinata una serie fissa di operazioni eseguibili in successione fra loro, come la selezione di item e criteri, in maniera tale da rendere identiche le operazioni compiute dai componenti del sistema. Le interrogazioni effettuate al database e i messaggi scambiati fra gli agenti, ad esempio, sono sempre gli stessi ogni volta che avviamo il test. Tutti i messaggi vengono poi archiviati in un file grazie a MinervaLog, il sistema di logging di Minerva, di cui abbiamo parlato nel capitolo precedente. Ad ogni messaggio è associato un *timestamp*, ossia il valore del clock di sistema relativo all'istante di archiviazione espresso in millisecondi. Attraverso questi valori è possibile ricostruire la tempistica dei messaggi all'interno del sistema. Ad esempio calcolando la differenza tra i timestamp del primo e dell'ultimo messaggio relativo ad un'operazione, si ottiene il tempo impiegato dal sistema per eseguirla. Abbiamo effettuato sei misurazioni del tempo necessario per la creazione del

museo e quattro per la visualizzazione di un item, per un totale di dieci prove per ogni sistema operativo. Questi valori, assieme alla loro media e varianza, sono consultabili nell'Appendice A.
Nei grafici successivi sono riportati i principali risultati dei test. Il primo, mostrato in Figura 6.1, si riferisce al tempo medio impiegato dal sistema per visualizzare un item, ovvero un oggetto con tutte le sue caratterstiche. In questo caso le sotto operazioni da compiere sono piuttosto semplici ed in numero non elevato, per cui i tempi misurati sono bassi.

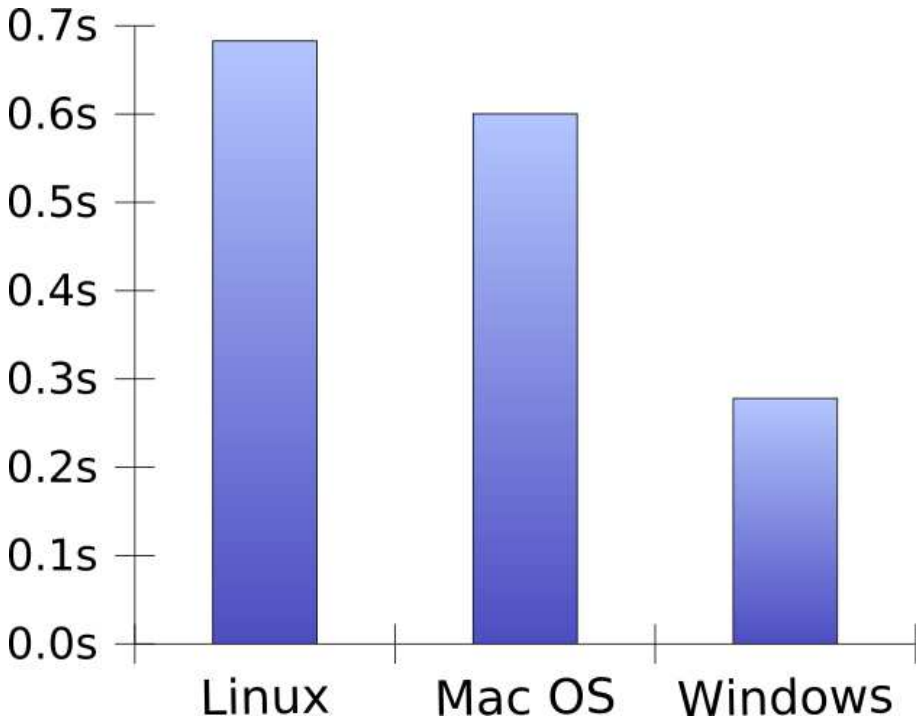

Figura 6.1: Tempo medio per la visualizzazione di un item

Il secondo grafico invece, mostrato in Figura 6.2, evidenzia come il processo di creazione del museo sia più oneroso e complesso. I tempi non sono soltanto mediamente più alti, ma dipendono dal numero di opere che devono essere collocate nelle varie stanze e dal numero maggiore di richieste da effettuare a Minerva Server per ottenre tutte le informazioni da visualizzare nella pagina del muso virtuale.

L'ultimo grafico infine (vedi Figura 6.3) si riferisce al massimo tempo impiegato da Minerva per soddisfare le richieste dell'utente. Si tratta di valori massimi di campioni piuttosto ristretti da noi misurati, per cui non è da escludere che esistano richieste complesse che necessitano di più tempo

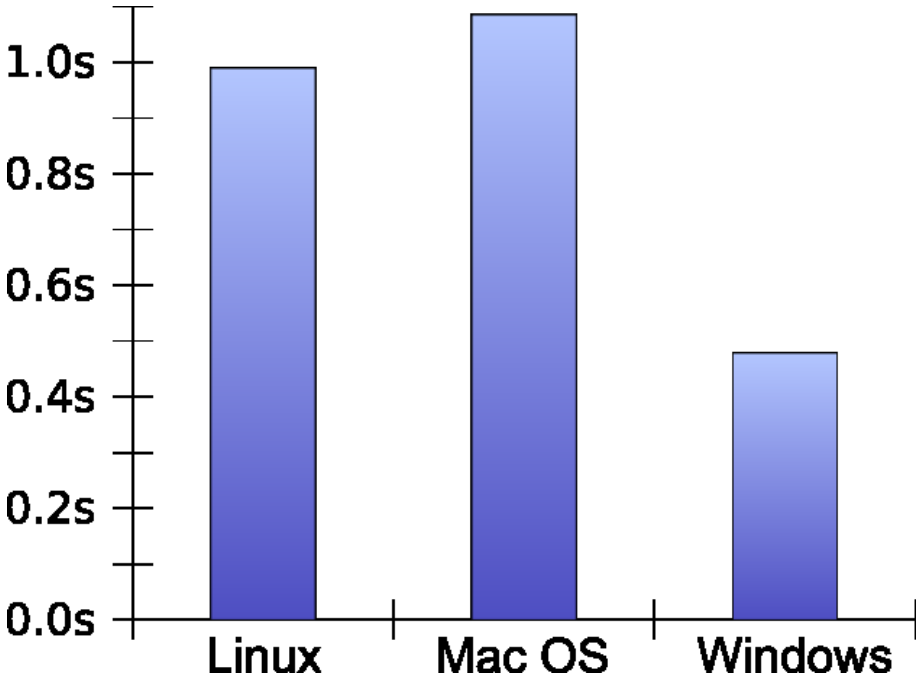

Figura 6.2: Tempo medio per la creazione di un museo

per essere elaborate. Si noti infine come, fra i sistemi operativi in esame, Windows risulti essere il più performante.

Abbiamo anche individuato come si ripartiscono i tempi di comunicazione e di computazione degli agenti che intervengono per soddisfare la richiesta da parte dell'utente, come in Figura 6.4. Si sono svolti alcuni test per riscontrare una eventuale differenza di prestazioni tra le richieste effettuate a freddo, ossia all'avvio del programma, e quelle a caldo, che sfruttano la presenza di dati nella cache. Dai risultati ottenuti si riscontra che non esistono differenze sostanziali nel comportamento di Minerva nei due stati. Inoltre è possibile evidenziare come il DBManager sia il collo di bottiglia nella fase di recupero degli items per la creazione del museo virtuale.

Tutti i tempi rilevati sono pienamente accettabili per l'utente finale e notevolmente al di sotto della soglia di sopportazione, stimata solitamente attorno ai quattro-cinque secondi.

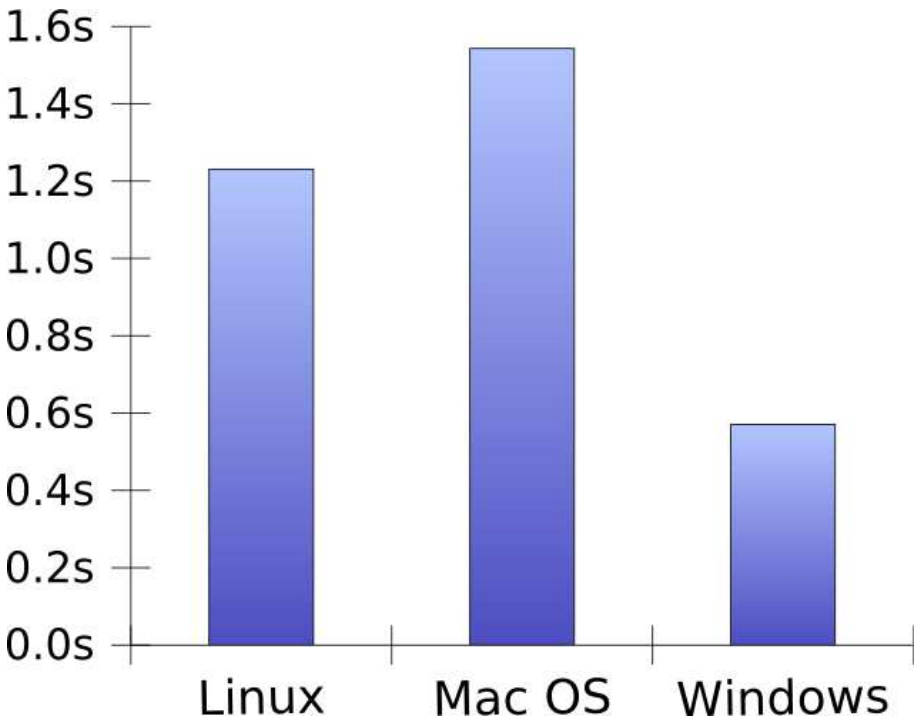

Figura 6.3: Tempo massimo per una richiesta

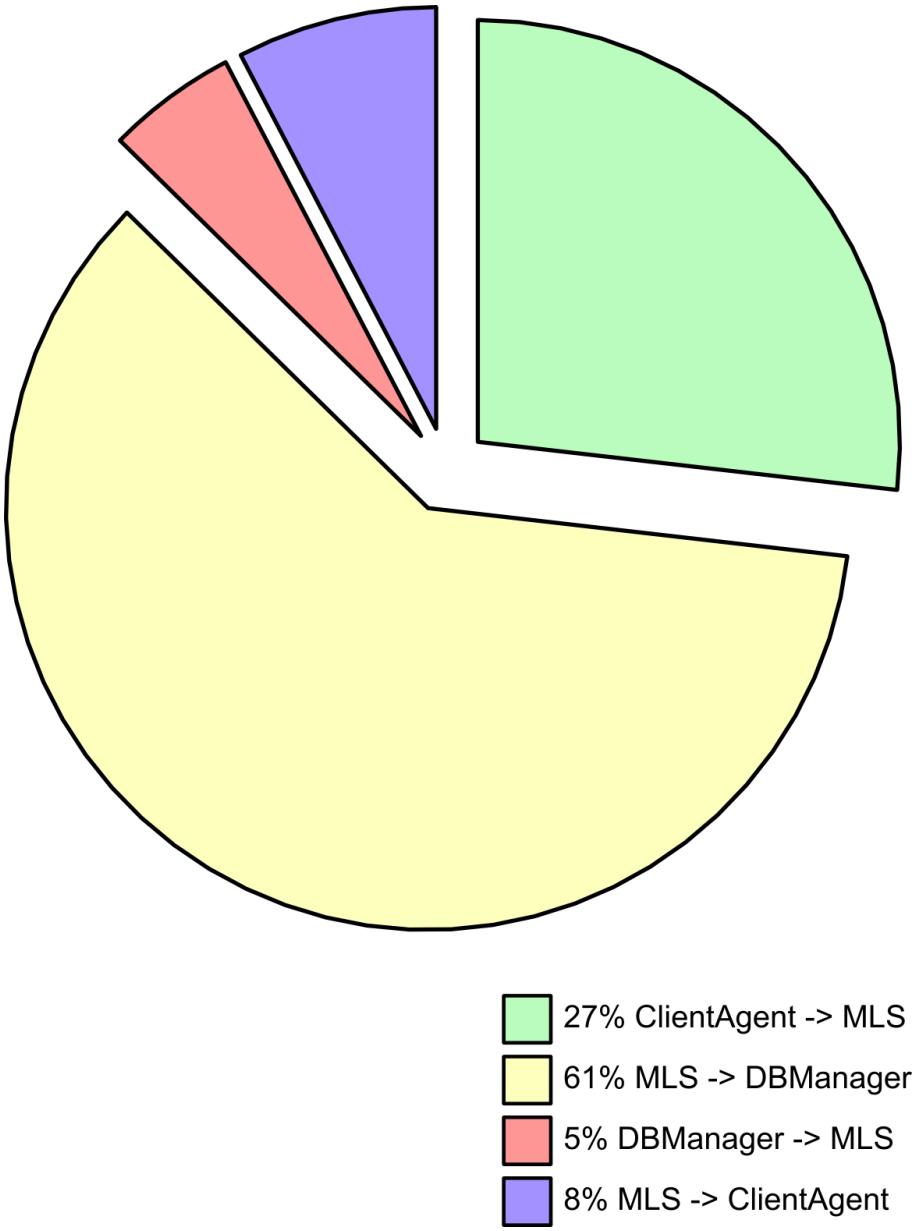

Figura 6.4: Suddivisione dei tempi necessari al recupero degli items

Capitolo 7

Direzioni future di ricerca e conclusioni

Il presente capitolo ha lo scopo di presentare le possibilità di ulteriori sviluppi di Minerva ad opera dei futuri tesisti. Nella parte finale sono inoltre presentate le conclusioni. La metodologia di lavoro adottata, come spesso ricordato, è stata orientata più alla soddisfazione della committenza che all'approfondimento di aspetti tecnologici e teorici, come era invece avvenuto per le precedenti versioni del progetto Minerva. Questi aspetti, oltre ovviamente a nuovi problemi pratici, potranno essere oggetto di studio da parte di lavori futuri.
Ogni paragrafo affronta il possibile sviluppo in un singolo ambito tra quelli che ci sembrano interessanti e che presentano margini di miglioramento.

7.1 Sviluppo dell'interfaccia utente

Lo sviluppo del progetto Minerva nell'ambito di una tesina di intelligenza artificiale si è concentrato principalmente sulla logica di gestione del museo e sui criteri per realizzare un museo a misura d'utente. Il nostro impegno si è focalizzato principalmente sulle funzionalità di interazione con l'utente. Il museo viene presentato tramite una veste grafica leggera e funzionale, basata su tradizionali pagine web generate dinamicamente. Questa soluzione lascia aperte molte possibilità per rendere l'esperienza di navigazione più naturale e coinvolgente. L'attuale versione di Minerva, come richiesto dalla committenza, pone l'accento principalmente sull'ampiezza e sulla libertà di scelta delle informazioni che il sistema fornisce all'utente. È stata quindi trascurata l'esperienza "spettacolare" della realtà virtuale tridimensionale.

La nostra attenzione è stata focalizzata soprattutto sui contenuti e sulla coerenza del sistema. Non è però stato posto nessun ostacolo alla realizzazione futura di un sistema grafico sovrapposto che, utilizzando le informazioni restituite dal nostro "motore", costruisca un museo virtuale tridimensionale con forte impatto visivo.
Le principali strade percorribili per la realizzazione di un motore tridimensionale sono l'uso del VRML o l'impiego delle librerie *Java 3D* [53]. Ricordiamo che le versioni precedenti di Minerva utilizzavano già VRML per creare ambienti virtuali. Questa scelta è diventata a nostro avviso poco percorribile, perché in pratica renderebbe difficile realizzare la continua interazione fra museo e utente che è tipica di questa versione di Minerva. Infatti questa è la differenza sostanziale rispetto al progetto originale, che creava solo musei statici non modificabili in tempo reale dall'utente. Il tipo di approccio tradizionale era lineare: si partiva da una schermata iniziale dove si selezionavano opere e criteri per giungere direttamente a un risultato finale rappresentato dal museo virtuale tridimensionale. L'utente quindi poteva muoversi al suo interno e vedere le opere esposte, ma non poteva in alcun modo modificare l'ambiente circostante se non ricominciando il processo. In questo contesto il ruolo attivo non spettava infatti all'utente finale ma all'allestitore del museo. Minerva era nato allo scopo di supportare gli allestitori nella creazione di musei virtuali esclusivamente fruibili dagli utenti finali. Come già visto in precedenza, la presente versione di Minerva mette invece a disposizione un approccio differente, in cui il visitatore può continuamente creare nuove collezioni a partire da quelle precedenti. Gli eventuali processi di generazione di ambienti tridimensionali sarebbero quindi molto più ricorrenti, incidendo in maniera non trascurabile sui tempi d'attesa.
L'alternativa al VRML potrebbe essere l'impiego delle librerie grafiche native di Java, come le librerie Java 3D. Questa soluzione è omogenea dal punto di vista software perché estenderebbe l'utilizzo della piattaforma Java anche alla visualizzazione, senza introdurre elementi estranei. D'altro canto modificherebbe la struttura del sistema eliminando la navigazione web e introducendo un client Java che sfrutti le librerie sviluppate per interfacciare le pagine web al server. Inoltre il problema legato alle prestazioni, che abbiamo individuato nel caso di VRML, potrebbe rimanere sostanzialmente immutato, a causa dell'elevata frequenza delle fasi di rendering.
Come possibile terza via per la visualizzazione 3D abbiamo pensato di simulare l'ambiente tridimensionale utilizzando il linguaggio *Macromedia Flash* [54]. Questa soluzione offrirebbe un miglior rapporto prezzo/prestazioni del pacchetto software, ma come contrappasso porterebbe con sè un minor rigore e una minore eleganza dal punto di vista strettamente ingegneristico.

Quando si volesse mettere in opera lo sviluppo di una veste grafica tridimensionale andrebbero senz'altro studiati attentamente i requisiti di qualità e performance del risultato e le caratteristiche delle soluzioni precedentemente illustrate, individuando quindi la tecnologia più indicata al caso specifico.
Qualora non si volesse sviluppare un ambiente tridimensionale per l'esposizione delle opere del museo virtuale, si potrebbe arricchire l'esperienza del visitatore attraverso le tecnologie emergenti del web. Utilizzando librerie in ECMAScript che implementano effetti grafici sugli elementi HTML si potrà offrire sicuramente un'esperienza più coinvolgente.
Sempre su questa linea si posiziona la tecnologia *Asyncrynous Javascript And XML (AJAX)* [55], grazie alla quale si possono gestire chiamate asincrone attraverso le pagine web. L'adozione di tale tecnologia permetterebbe il caricamento parziale di dati senza la necessità di effettuare la richiesta di una nuova pagina al web server. Sarebbe così possibile offrire informazioni aggiuntive contestualmente alle richieste dell'utente. Ad esempio al passaggio del mouse sopra un item si potrebbero visualizzare la sua foto e la sua descrizione. Infine la gestione dei dati sarebbe direttamente integrabile nel museo virtuale. L'amministratore di sistema potrebbe modificare le informazioni visualizzate semplicemente cliccando sui campi di testo presenti nella pagina senza dover accedere a MinervaAdmin.

7.2 Allestitore

Come già detto, il nostro sviluppo di Minerva ha portato a una modifica sostanziale dell'architettura del sistema. Ciò è stato dovuto alla nuove esigenze espresse dalla committenza. Una delle conseguenze è stata il mancato utilizzo dell'agente Allestitore. Questo non vuole dire che in futuro non possa essere utilizzato nuovamente. L'agente Allestitore potrebbe quindi ritrovare un ruolo attivo qualora il progetto Minerva venisse applicato a una situazione che preveda una più raffinata operazione di raggruppamento delle opere. In questo caso potrebbe essere necessario ricorrere alla generazione di un albero tassonomico. Con questo termine intendiamo un grafo aciclico che rappresenta la classificazione di concetti tramite gerarchie. Il risultato del processo di allestimento, che consiste in un raggruppamento gerarchico di opere, è rappresentato chiaramente da questa struttura dati.
L'albero tassonomico rappresenta il mezzo ideale per presentare all'utente l'allestimento, ma ha il difetto di essere difficilmente rappresentabile all'interno di un database relazionale. Le cause di questa difficoltà risiedono nelle proprietà strutturali dell'albero. Lo scopo dell'agente Allestitore sarebbe quello di tradurre l'albero tassonomico in una struttura dati, detta

teoria decidibile, rappresentata mediante una base di conoscenza implementata in linguaggio CLIPS [56]. Come nelle precedenti versioni di Minerva si potrebbe usare JESS [57] per impiegare le regole CLIPS all'interno di un programma Java minimizzando gli sforzi di programmazione.
Il risvolto negativo di questo nuovo assetto di Minerva consisterebbe nella lentezza di aggiornamento delle pagine causata da un continuo scambio di dati fra agenti. È pertanto necessario valutare attentamente l'esigenza di ricorrere a questa soluzione, la cui convenienza è garantita solo nel caso si tratti effettivamente di compiere operazioni di allestimento molto complesse.

7.3 Allocazione di stanze e teche

L'ambiente virtuale del Museo di Ossuccio è puramente determinato dal numero di oggetti che "popolano" le varie stanze e teche. Questo è coerente con gli obbiettivi della committenza e con la particolare situazione affrontata nel nostro progetto, ma crea alcuni limiti nell'ipotesi si voglia ambientare il museo in luoghi reali o comunque caratterizzati da una planimetria definita. Sarebbe innanzitutto necessario pensare ad una logica di gestione degli spazi, definendo i vincoli di disposizione delle opere all'interno delle stanze in relazione alla presenza di porte, finestre, punti luce e altri elementi fissi. Sarebbe inoltre fondamentale introdurre dei parametri che tengano conto del grado di "affollamento" di una parete in base alla sua posizione e alle caratteristiche dei reperti presenti.
Con altrettanta attenzione andrebbero poi rispettati i vincoli fisici di ingombro delle superfici e dei volumi delle stanze. Diverrebbe anche necessario apportare consistenti modifiche al database in modo tale che possano essere memorizzati i dati volumetrici di ambienti e oggetti, oltre ai dati concettuali sui quali abbiamo lavorato per questa versione di Minerva.
Il valore di priorità che è stato associato ad ogni oggetto inserito nel database assumerebbe grandissima importanza in questa nuova situazione, in particolare se lo spazio museale virtuale non sia in grado di accogliere tutti gli oggetti della collezione. Attraverso la priorità infatti si potrebbe avere una preziosa indicazione dell'importanza attribuita a ciascun oggetto, con lo scopo di disporre di un ulteriore criterio per decidere se un oggetto meriti o meno di occupare parte delle limitate risorse spaziali del museo.
Tutti i parametri di allocazione dovranno però essere tarati caso per caso da un allestitore esperto che risulta comunque insostituibile. Infatti una procedura totalmente automatizzata porterebbe inevitabilmente a musei privi di quel gusto e quella armonia che solo l'esperto e attento occhio umano può creare.

Questi argomenti sono già stati approfonditi nella tesi di Ghidotti. Uno degli obbiettivi di quella tesi è stato proprio il miglioramento del processo di allestimento grazie a una interazione continua tra il software e l'esperto quando invece in precedenza tutto il processo era automatizzato senza nessun intervento esterno. Quindi lo studio Minerva05 rappresenterebbe senz'altro il punto di partenza per l'implementazione di questa funzionalità.

7.4 JSP, Servlet e il modello MVC

Abbiamo visto come siano le servlet ad occuparsi sia della comunicazione con Minerva Server sia a costruire le pagine HTML da mostrare all'utente. Questo tipo di approccio, semplice e funzionale, non consente la netta separazione tra forma e contenuto. Una soluzione più elegante potrebbe sfruttare le Java Service Pages per la creazione delle pagine HTML, in quanto sono facilmente gestibili dagli sviluppatori che si occupano dell'interfaccia utente, lasciando alle servlet il compito di comunicare con il server.
Per migliorare ulteriormente il lato client di Minerva si può pensare di adottare il modello *Model-View-Controller (MVC)* (vedi Figura 7.1), ovvero un approccio ibrido per servire contenuti dinamici e che si appoggia a JSP, servlet e JavaBean. In questo caso si sfruttano i punti di forza di entrambe

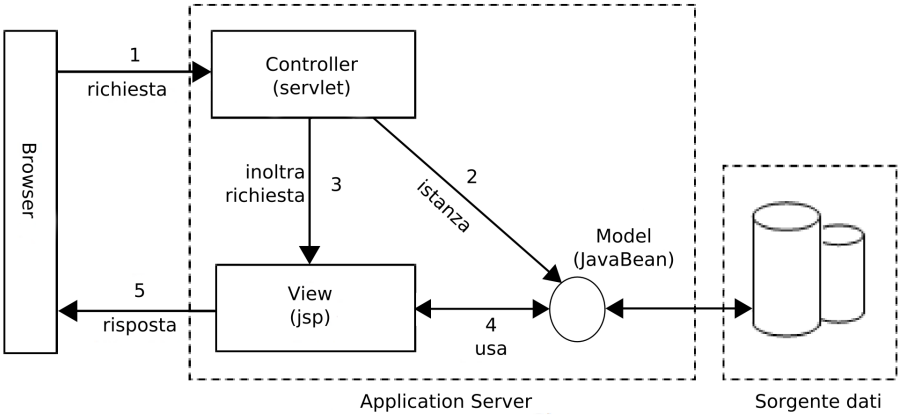

Figura 7.1: Un esempio di Model-View-Controller

le tecnologie, JSP e servlet, utilizzando JSP per generare lo strato di presentazione e servlet per eseguire compiti intensivi di processo. La servlet agisce da controller, ed è incaricata di processare le richieste degli utenti. Di volta in volta in base alla richiesta crea l'istanza di un bean (il model, un componente riutilizzabile, quale potrebbe essere nel nostro caso un item) che

verrà poi utilizzata dalla JSP (la view) a cui la servlet decide di inoltrare la richiesta per recuperare i dati elaborati da Minerva Server. Bisogna notare che non c'è logica di processo all'interno della pagina JSP. Ad essa viene unicamente affidato il compito di recuperare gli oggetti o i bean precedentemente creati dalla servlet, e di estrarne il contenuto dinamico per inserirlo in un template statico. In questo modo viene realizzata una totale separazione tra presentazione e contenuti, con una chiara definizione dei ruoli e delle responsabilità di designer e programmatori, che potranno quindi lavorare con una certa indipendenza. Inoltre, maggiore sarà la complessità dell'applicazione, maggiori dovrebbero risultare i benefici dell'implementazione di questo modello.

7.5 Sistema centralizzato e distribuito

La destinazione d'uso del software prevede il suo utilizzo all'interno del museo di Ossuccio solo su postazioni fisse e non comunicanti tra loro. Questo non implica che in futuro si possa pensare di utilizzarlo anche via web o tramite un sistema distribuito che utilizza un unico database centrale disponibile a tutte le postazioni. La struttura del sistema, pensata per essere scalabile, è stata descritta nel Capitolo 5.
Una prima semplice ed economica politica di gestione sarebbe quella di installare su un server tutto il software di Minerva e il Database MySQL, lasciando quindi ai client solo il browser. Questa scelta è molto comoda perché permetterebbe l'aggiornamento di Minerva su un solo sistema e nello stesso tempo, grazie alle buone performance delle reti locali, non porterebbe a un decadimento delle prestazioni. Il reale limite di questa soluzione diventa palese con un numero elevato di utenti connessi, poiché le risorse hardware del sistema centrale risulterebbero insufficienti a gestire un numero elevato di utenti.
Questo problema può essere risolto tramite la distribuzione del carico di lavoro su più server utilizzando un bilanciatore di carico oppure è possibile suddividere i compiti su più sistemi hardware. Ecco un esempio basato su tre diversi computer:

- Sul primo si installa il web server con MinervaWebClient, in grado di gestire le richieste di accesso provenienti sia dal museo stesso che da qualsiasi altro host su Internet;
- Sul secondo è situato Minerva Server, che si interfaccia con il database e restituisce i risultati a MinervaWebClient;

- Sul terzo sistema è presente il database server, incaricato di recuperare i dati memorizzati localmente.

I server verrebbero messi in comunicazione tra di loro tramite una rete *LAN Ethernet* di classe Gigabit, tecnologia ormai ampiamente diffusa e che grazie ad una banda elevata eliminerebbe il rischio di colli di bottiglia legati alle comunicazioni tra i vari computer. Se il numero di utenti diventa considerevole è inoltre possibile pensare a un sistema più complesso in cui vengano replicati i singoli server, gestito da un'applicazione aggiuntiva per bilanciare il carico di lavoro tra di essi. Questo caso si potrebbe presentare se ci fossero molti accessi dal web, anche se vista la peculiarità del sistema Minerva ciò è improbabile. Segnaliamo infine, nel caso di sistema distribuito, la possibilità di impiegare computer con sistemi operativi diversi in base alle esigenze specifiche, alle prestazioni o alla disponibilità. Ad esempio il server Minerva potrebbe utilizzare Mac OS, MySQL potrebbe essere eseguito su architettura Linux e i client potrebbero essere tutti sistemi Windows.
Nella tesina Andretta-Erba è stato segnalato anche il problema della concorrenza. Questo problema, gestito in modo rigoroso e deterministico, è stato finalmente superato. Un'altra notevole differenza con la situazione della tesina Andretta-Erba è l'omogenità di prestazioni tra i diversi sistemi operativi. Questo risultato, ottenuto tramite una revisione e ottimizzazione del codice, dà più spazio alla costruzione di sistemi misti, mentre in precedenza l'opzione Mac OS risultava essere una scelta praticamente obbligata.

7.6 Miglioramento DBManager

L'aggiunta di tutte le query e delle relative istruzioni nel DBManager necessarie al corretto funzionamento dell'applicativo ha creato un aumento eccessivo di dimensioni del modulo relativo a tale agente. Per migliorare la leggibilità e la manutenzione del suo codice è consigliabile scorporare il comportamento dell'agente in modo da rendere più snella la sua implementazione. Si potrà anche pensare di suddividere la classe che gestisce il comportamento in più sottoclassi che contengano le istruzioni relative ad un solo campo specifico. Ad esempio si potrebbe separare la parte di codice che si occupa delle richieste per il museo dell'isola Comacina da quello utilizzato nelle versioni precedenti. Tutto questo permetterebbe una più facile espansione futura del progetto, facilitando l'aggiunta di nuove base dati.
Si potrebbe addirittura ipotizzare di rivoluzionare completamente la logica con cui questo componente opera all'interno di Minerva. Attualmente l'agente DBManager si occupa di effettuare le query e di inviare i dati a

chi ne ha fatto richiesta senza però che questi siano incapsulati in oggetti. I risultati potrebbero essere resi più astratti attraverso l'utilizzo di una classe che si occupasse direttamente del salvataggio dei dati, realizzando in questo modo l'indipendenza del sistema dall'implementazione fisica dei dati stessi. Ad esempio, a seconda della destinazione d'uso, una classe potrebbe immagazzinare i dati nel dabatabase o in un file XML mantenendo un comportamento trasparente nei confronti di Minerva Server. Il componente diverrebbe allora un gestore di oggetti, per cui sarebbe auspicabile sostituire il nome attuale DBManager con "ObjectManager". Il ruolo rivestito dall'agente nello scenario di tutto il progetto resterebbe comunque invariato.

7.7 Generalizzazione del progetto

Fin dal principio il progetto Minerva è stato concepito come un sistema flessibile e generale. Lo scopo è quello di creare un gestore di musei e non un'applicazione limitata ad un solo contesto in particolare. Infatti è possibile plasmarlo su diversi musei modificando solo un numero relativamente modesto di linee di codice.
Una interessante modifica evolutiva potrebbe essere quella di rendere Minerva adattabile a qualsiasi museo senza intervenire sul codice, ma solo editando i campi di MinervaAdmin. Questo può essere realizzato ampliando MinervaAdmin in modo tale che possa modificare anche la pagina iniziale del museo, tutte le etichette di testo ed anche il foglio di stile CSS. Sarebbe poi necessario modificare anche l'agente DBManager, in modo tale che queste informazioni vengano acquisite in modo dinamico.
La caratteristica che rende questa versione di Minerva innovativa rispetto alle precedenti è il fatto di essere divenuto uno strumento non solo per la creazione di musei virtuali ma anche un vero e proprio "gestore di conoscenza". Con questo termine intendiamo un sistema in grado di organizzare una serie di informazioni, selezionate e catalogate da personale esperto, allo scopo di poter essere esplorate da un utente meno esperto che, in modo dinamico e in tempo reale, crea un percorso di visita in base ai propri interessi. Ciò è possibile perché Minerva è stato realizzato con un elevato grado di astrazione. I concetti base, come spiegato nel Capitolo 4, sono quelli di "item", per sua natura astratto, e di "tema", che non è altro che un insieme di item aventi una particolare caratteristica in comune. Un esempio pratico potrebbe essere la realizzazione di un sistema a scopo didattico per la divulgazione di informazioni riguardo il mondo vegetale in cui i tipi di territori avrebbero la funzione di "tema" e le piante avrebbero la funzione di "item".

7.8 Compatibilità delle linee di sviluppo

Le varie linee di sviluppo che abbiamo qui delineato a volte corrono su binari non compatibili tra loro. Un chiaro esempio è fornito dal conflitto tra la generalizzazione del progetto e la realizzazione di un'interfaccia tridimensionale di tipo classico. Ciò non costituisce un problema ma è anzi una risorsa, perché da un singolo software si possono avere evoluzioni diverse e anche le direzioni di sviluppo possono mutare in base a nuove esigenze specifiche. È però utile lasciare il maggior numero di vie percorribili per evitare di dover ogni volta "reinventare la ruota", ma al contrario trarre maggior profitto possibile dal lavoro precedente.

7.9 Conclusioni

L'attività della presente tesina è stata finalizzata all'implementazione di tutte le funzionalità individuate nella tesina Andretta-Erba per la realizzazione del museo virtuale dell'isola Comacina. Questo ha comportato una prima fase di analisi e revisione del precedente software Minerva, per stabilire come le nuove modifiche potessero inserirsi all'interno di un lavoro già in corso da diversi anni.
Tutto il lavoro si è svolto in stretta collaborazione con gli altri tesisti e la supervisione del nostro relatore. La divisione dei compiti non è mai stata rigida, favorendo un continuo scambio di conoscenze e opinioni all'interno del gruppo di lavoro.
L'architettura del software precedente è rimasta sostanzialmente immutata. Sono stati aggiunti nuovi componenti e modificati quelli esistenti in modo da conferire al sistema nuove caratteristiche senza interrompere la linea evolutiva già intrapresa. A lato server è stato aggiunto il database per la gestione specifica del museo di Ossuccio, e un agente per migliorare la robustezza del software. Gli agenti MLS e DBManager sono stati rivisti e modificati per implementare le nuove caratteristiche richieste. Il lato client è stato sviluppato appositamente per questo progetto scegliendo le tecnologie più appropriate per il contesto specifico.
Risultato di questa tesina è stato una versione del software Minerva ulteriormente migliorata. Oltre ad aver pienamente soddisfatto la committenza, questo lavoro ha risolto i problemi individuati da Andretta ed Erba riguardanti la concorrenza e le prestazioni.

Bibliografia

[1] Fabio Ghidotti. *Minerva: un sistema multiagente per l'allestimento dei musei archeologici*. PhD thesis, Politecnico di Milano, 2003-2004.

[2] Andretta-Erba. *Minerva: Analisi e Progettazione del Museo Virtuale dell'Isola Comacina*. PhD thesis, Politecnico di Milano, 2004-2005.

[3] Hypertext markup language (html), ultimo accesso marzo 2006, http://www.w3.org/markup/.

[4] Cascading style sheets, ultimo accesso marzo 2006, http://www.w3.org/style/css/.

[5] Apache tomcat, ultimo accesso marzo 2006, http://tomcat.apache.org/.

[6] Java servlet, ultimo accesso marzo 2006, http://java.sun.com/products/servlet/.

[7] Jade (java agent development framework), ultimo accesso marzo 2006, http://jade.tilab.com/.

[8] Mysql, ultimo accesso marzo 2006, http://www.mysql.com.

[9] Marco Somalvico, 1941-2002.

[10] Worcester art museum, ultimo accesso marzo 2006, http://www.worcesterart.org.

[11] National gallery, ultimo accesso marzo 2006, http://nationalgallery.org.uk/.

[12] State hermitage museum, ultimo accesso marzo 2006, http://www.hermitagemuseum.org/.

[13] Corvette museum, ultimo accesso marzo 2006,
http://www.corvettemuseum.com.

[14] Quick time virtual reality (qtvr), ultimo accesso marzo 2006,
http://www.apple.com/quicktime/overview/qtvr.html.

[15] Yorkshire air museum, ultimo accesso marzo 2006,
http://www.vryork.com/.

[16] Virtual reality modeling language (vrml), ultimo accesso marzo 2006,
http://www.web3d.org.

[17] Virtual tessellation museum, ultimo accesso marzo 2006,
http://www.tessellation.info.

[18] Virtual museum of canada, ultimo accesso marzo 2006,
http://www.virtualmuseum.ca.

[19] Iso 9241 (ergonomic requirements for office work with visual display terminals). la parte 11 dell'iso 9241, in cui è contenuta questa definizione di usabilità è, in realtè, ancora nello stato di committee draft, e pertanto ancora potenzialmente soggetto a modifiche.

[20] The world wide web consortium, ultimo accesso marzo 2006,
http://www.3w.org.

[21] Linux, ultimo accesso marzo 2006,
http://www.linux.org.

[22] Mac os, ultimo accesso marzo 2006,
http://www.apple.com/macosx/.

[23] Java, ultimo accesso marzo 2006,
http://java.sun.com.

[24] Microsoft windows, ultimo accesso marzo 2006,
http://msdn.microsoft.com/.

[25] Hypertext transfer protocol (http), ultimo accesso marzo 2006,
http://www.w3.org/protocols/.

[26] Transmission control protocol (tcp) , ultimo accesso marzo 2006,
http://www.ietf.org/rfc/rfc793.txt,
internet protocol (ip), ultimo accesso marzo 2006,
http://www.ietf.org/rfc/rfc791.txt.

[27] Microsoft access, ultimo accesso marzo 2006,
http://office.microsoft.com/en-us/fx010857911033.aspx.

[28] Gnu general public license, ultimo accesso marzo 2006,
http://www.gnu.org/copyleft/gpl.html.

[29] Tilab, ultimo accesso marzo 2006,
http://www.telecomitalialab.com/.

[30] Fipa, ultimo accesso marzo 2006,
http://www.fipa.org/.

[31] Apache foundation, ultimo accesso marzo 2006,
http://www.apache.org.

[32] Java service pages, ultimo accesso marzo 2006,
http://java.sun.com/products/jsp/.

[33] Subversion, ultimo accesso marzo 2006,
http://subversion.tigris.org/.

[34] Trac, ultimo accesso marzo 2006,
http://www.edgewall.com/trac/.

[35] Wiki, ultimo accesso marzo 2006,
http://wiki.org/wiki.cgi?whatiswiki.

[36] Eclipse, ultimo accesso marzo 2006,
http://www.eclipse.org/.

[37] Modello a spirale, ultimo accesso marzo 2006,
http://www.analisi-disegno.com/processo/iterativo.htm.

[38] Uniform resource locators (url), ultimo accesso marzo 2006,
http://www.w3.org/addressing/rfc1738.txt.

[39] java.util.properties, ultimo accesso marzo 2006,
http://java.sun.com/j2se/1.4.2/docs/api/java/util/properties.html.

[40] java.util.logging.logger, ultimo accesso marzo 2006,
http://java.sun.com/j2se/1.4.2/docs/api/java/util/logging/logger.html.

[41] Simpleformatter, ultimo accesso marzo 2006,
http://java.sun.com/j2se/1.5.0/docs/api/java/util/logging/simpleformatter.html.

[42] Extensible markup language (xml), ultimo accesso marzo 2006, http://www.w3.org/xml/.

[43] Ecmascript, ultimo accesso marzo 2006, http://www.ecma-international.org/publications/standards/ecma-327.htm.

[44] Arrays.binarysearch, ultimo accesso marzo 2006, http://java.sun.com/j2se/1.4.2/docs/api/java/util/arrays.html.

[45] Mysql connector, ultimo accesso marzo 2006, http://www.mysql.com/products/connector/j/.

[46] jade.core.agent, ultimo accesso marzo 2006, http://jade.tilab.com/doc/api/jade/core/agent.html.

[47] jade.domain.dfservice, ultimo accesso marzo 2006, http://jade.tilab.com/doc/api/jade/domain/dfservice.html.

[48] Theard, ultimo accesso marzo 2006, http://java.sun.com/j2se/1.4.2/docs/api/java/lang/thread.html.

[49] Pair programming, ultimo accesso marzo 2006, http://www.pairprogramming.com/.

[50] Unit test, ultimo accesso marzo 2006, http://en.wikipedia.org/wiki/unit_test.

[51] Regression test, ultimo accesso marzo 2006, http://en.wikipedia.org/wiki/regression_testing.

[52] Refactoring, ultimo accesso marzo 2006, http://en.wikipedia.org/wiki/refactoring.

[53] Java 3d api, ultimo accesso marzo 2006, http://java.sun.com/products/java-media/3d/.

[54] Macromedia flash, ultimo accesso marzo 2006, http://www.macromedia.com/software/flash/flashpro/.

[55] Asynchronous javascript and xml (ajax), ultimo accesso marzo 2006, http://en.wikipedia.org/wiki/ajax.

[56] Clips, ultimo accesso marzo 2006, http://www.ghg.net/clips/clips.html.

[57] jess, ultimo accesso marzo 2006,
http://herzberg.ca.sandia.gov/jess/.

Appendice A

Analisi delle prestazioni

Documentazione delle prove svolte per verificare i tempi di risposta dell'applicativo Minerva. Le analisi dei risulati possono essere consultate nel Capitolo 6.

Minerva - Test	Themes	Criteria	Start	End	Tempo effettivo	Tempo sec.
1) Apparizioene primo item			1141636884017	1141636884696	679	0.679
2) Museo (dal capitello)	1,2,3	1	1141637391837	1141637392871	1034	1.034
3) Frammento lapideo con dec.			1141637638225	1141637639085	860	0.86
4) Museo (dal 3)	1	2	1141637756648	1141637757879	1231	1.231
5) Basimento di semicolonna			1141637892045	1141637892731	686	0.686
6) Museo (dal 5)	1,2	3	1141637990807	1141637991826	1019	1.019
7) Embrice			1141638244941	1141638245447	506	0.506
8) Museo (dal 7)	1	4	1141638344244	1141638345041	797	0.797
9) Museo dal capitello (riazzera)	1,2,3,4,5,6,7	2	1141638483325	1141638484254	929	0.929
10) Museo dal capitelo (form in alto)	5,6,7	1	1141638701584	1141638702510	926	0.926

Figura A.1: Tabella dei tempi rilevati su Linux Ubuntu

Minerva - Test	Themes	Criteria	Start	End	Tempo effettivo	Tempo sec.
1) Apparizioene primo item			1141640896148	1141640896841	693	0.693
2) Museo (dal capitello)	1,2,3	1	1141641066727	1141641068270	1543	1.543
3) Frammento lapideo con dec.			1141641153367	1141641153945	578	0.578
4) Museo (dal 3)	1	2	1141641258955	1141641260102	1147	1.147
5) Basimento di semicolonna			1141641354399	1141641355024	625	0.625
6) Museo (dal 5)	1,2	3	1141641416981	1141641417937	956	0.956
7) Embrice			1141641508295	1141641508800	505	0.505
8) Museo (dal 7)	1	4	1141641592809	1141641593824	1015	1.015
9) Museo dal capitello (riazzera)	1,2,3,4,5,6,7	2	1141641671028	1141641671989	961	0.961
10) Museo dal capitelo (form in alto)	5,6,7	1	1141641731444	1141641732333	889	0.889

Figura A.2: Tabella dei tempi rilevati su Mac OS X

Minerva - Test	Themes	Criteria	Start	End	Tempo effettivo	Tempo sec.
1) Apparizioene primo item			1141718671992	1141718672373	381	0.381
2) Museo (dal capitello)	1,2,3	1	1141718824472	1141718825043	571	0.571
3) Frammento lapideo con dec.			1141718892740	1141718893000	260	0.26
4) Museo (dal 3)	1	2	1141718960878	1141718961339	461	0.461
5) Basimento di semicolonna			1141719024129	1141719024399	270	0.27
6) Museo (dal 5)	1,2	3	1141719073790	1141719074241	451	0.451
7) Embrice			1141719128048	1141719128249	201	0.201
8) Museo (dal 7)	1	4	1141719174856	1141719175286	430	0.43
9) Museo dal capitello (riazzera)	1,2,3,4,5,6,7	2	1141719243314	1141719243800	486	0.486
10) Museo dal capitelo (form in alto)	5,6,7	1	1141719347794	1141719348265	471	0.471

Figura A.3: Tabella dei tempi rilevati su Windows XP

	Linux	Mac OS	Windows
Tempo Medio Creazione Museo	0.989	1.085	0.478
Varianza Creazione Museo	0.025	0.058	0.002
Tempo Medio Item	0.683	0.600	0.278
Varianza Item	0.021	0.006	0.006
Tempo Massimo Creazione Museo	1.231	1.543	0.571

Figura A.4: Tabella dei tempi generale

A freddo

Agente	Ontologia	Timestamp	Tempo	Messaggio	Tempo	Percentuale
ClientAgent	getItems	1142325451591				
MLS	getItems	1142325451661		ClientAgent -> MLS	0.07	27%
DBManager	items	1142325451819		MLS -> DBManger	0.16	61%
MLS	items	1142325451832		DBManager -> MLS	0.01	5%
ClientAgent	items	1142325451852		MLS -> ChekAgent	0.02	8%
			0.261			

A caldo

Agente	Ontologia	Timestamp	Tempo	Messaggio	Tempo	Percentuale
ClientAgent	getItems	1142325924369				
MLS	getItems	1142325924395		ClientAgent -> MLS	0.03	21%
DBManager	items	1142325924457		MLS -> DBManger	0.06	50%
MLS	items	1142325924465		DBManager -> MLS	0.01	6%
ClientAgent	items	1142325924493		MLS -> ChekAgent	0.03	23%
			0.124			

Figura A.5: Tabella dei tempi di recupero degli items