Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea Triennale in Informatica

PROGETTAZIONE E SVILUPPO DI UN INTERFACCIA GRAFICA PER OSMOSIS

Tesi di Laurea in Infomatica

III Sessione
Anno Accademico 2009/2010

Relatore:
Dott. Ugo Dal Lago

Tesi di laurea di:
Davide Mezzetti

Dedicato alla mia famiglia..

Indice

Introduzione

L'oggetto di questo lavoro di tesi è la progettazione e lo sviluppo di un'interfaccia grafica per Osmosis (Open Source and Multi platfOrm Shared Interactive Screen). Osmosis è un applicazione sviluppata come tesi in una laurea triennale di Informatica e fornisce una lavagna virtuale condivisa sulla quale poter disegnare. Nel tempo l'applicazione è cresciuta e sono state aggiunte nuove ed interessanti funzionalità raggiungibili attraverso nuovi bottoni e nuove voci di menù. In questo contesto è opportuno studiare le tecniche che vengono utilizzate per fornire agli utenti un modo semplice ed efficace per poter interagire con l'applicazione. L'Interaction Design viene in nostro soccorso aiutandoci a comprendere quali sono le relazioni tra gli artefatti e le persone che li utilizzano e a definirne, di conseguenza, la forma e il modello di interazione. Questo elaborato di tesi è organizzato come segue: nel capitolo 1 **L'Interaction Design** vedremo il contesto storico e le principali motivazioni che hanno portato alla nascita dell'interaction design. Vedremo come, negli utlimi trent'anni questa si sia raffinata fino a produrre delle importanti linee guida espresse efficacemente dagli obiettivi di usabilità e di esperienza d'uso. Nel capitolo 2 **Analisi dell'Interfaccia Grafica di Osmosis** applicheremo i principi e le tecniche discusse nel capitolo 1 al prototipo già esistente dell'interfaccia grafica per capire quali modifiche sono necessarie. Il capitolo 3 **Il Nuovo Prototipo di Interfaccia** fornisce una dettagliata spiegazione degli interventi che sono stati effettuati per ottenere gli obiettivi dell'interaction design e realizzare la nuova interfaccia. Infine nel capitolo 4 **Il Codice** vedremo la parte implementativa, analizzando il codice Java che realizza gli interventi spiegati nel capitolo 3.

Elenco delle figure

Elenco dei listati

Capitolo 1

L'Interaction Design

Per design dell'iterazione o interaction design, si intende: *la progettazione di artefatti in grado di sostenere le persone nella loro vita quotidiana e professionale*[1]. In pratica progettare l'interazione significa studiare e creare esperienze d'utilizzo con il fine di migliorare e potenziare il modo in cui le persone lavorano e comunicano. In generale l'interaction design è rivolta a dare supporto alle persone, viene infatti definita da Winograd[2] come *la progettazione di spazi per la comunicazione e l'interazione umana.*

Nel contesto informatico questa attività si differenzia dall'approccio dell'ingegneria del software che si concentra sulla realizzazione di soluzioni software per specifiche applicazioni. L'ingegneria del software prende in considerazione i processi produttivi finalizzati allo sviluppo software dal punto di vista tecnologico, (come ad esempio attraverso la definizione di nuovi linguaggi di programmazione) e dal punto di vista metodologico (analizzando e perfezionando il ciclo di vita del software).

L'interaction design viene in aiuto all'ingegneria del software per renderla fruibile agli utenti, fornendo un punto di vista piu olistico dell'applicazione[1]. Vediamo con un esempio di chiarire questo punto. Prendiamo in considerazione la costruzione di una casa. Questa vedrà la partecipazione di architetti e ingegneri civili. Entrambi, pur affrontando lo stesso problema, si comportano in modo differente. Gli ingegneri civili si preoccuperanno delle questioni relative alla realizzazione del progetto, considerando i materiali di messa in opera, la durata dei lavori, gli aspetti strutturali, le normative di sicurezza ecc. Gli architetti invece

penseranno maggiormente agli aspetti dovuti alle persone che vivranno nella casa, considerando l'armonia tra gli spazi condivisi e quelli privati, il posizionamento di porte e finestre negli ambienti, nonchè all'aspetto estetico della casa. Entrambi i punti di vista vanno considerati per realizzare una buona opera.

Da notare che oltre ad architetti ed ingegneri, sono necessari molti altri professionisti alla realizzazione di una casa (come ad esempio muratori, idraulici, elettricisti, ecc), lo stesso accade nella realizzazione di un applicativo software. In base alla sua grandezza e complessità vi parteciperanno grafici, psicologi, pubblicitari, team di informatici e ingegneri ecc.

1.1 Storia dell'Interaction Design

L'ingegneria del software nasce alla fine degli anni sessanta dall'esigenza di sviluppare prodotti sempre più complessi ed evoluti. All'epoca lo sviluppo del software personale era alquanto limitato: molti programmi venivano sviluppati per batch, gli informatici erano pochi ed apprendevano sul campo. Ciò che veniva sviluppato era pensato per un unico cliente, inoltre ad ogni progetto lavorava ed avrebbe lavorato una sola persona, senza scrivere alcuna documentazione.

Gli ingegneri progettavano software che altri ingegneri avrebbero utilizzato. L'interfaccia di un computer era qualcosa di relativamente semplice e lineare, che includeva anche una serie di quadranti ed interruttori che controllavano lo stato interno del sistema.

Con l'introduzione di schermi e nuovi dispositivi hardware di input, come le tastiere, nacquero a cavallo fra gli anni settanta e ottanta le postazioni personali e con esse l'interaction design [3]. Le nuove interfaccie rendevano disponibili nuove operazioni, come lo svolgimenti di calcoli, operazioni bancarie, scrivere documenti ecc che potevano essere svolti anche da persone senza un adeguata preparazione tecnica. Se prima il calcolatore poteva essere utilizzato solo da ingegneri informatici, la sfida ora è quella di sviluppare computer che potessero essere utilizzati anche da altre persone. Per rendere questo possibile informatici e psicologi si confrontarono con la progettazione di interfaccie utente. Informatici e ingegneri svilupparono nuovi linguaggi di programmazione (come Basic o Prolog), nuove architetture di sistema e metodi di progettazione del software per sostenere i com-

piti menzionati sopra, mentre gli psicologi fornivano informazioni sulle capacità cognitive umane (memoria e processi decisionali).

Negli anni seguenti, la ricerca si concentrò sulle interfaccie grafiche (dette anche GUI, da Graphical User Interface) un paradigma di sviluppo che mira a consentire all'utente di interagire con il computer manipolando graficamente gli oggetti. Le GUI fornivano metodi che svincolavano l'utente dall'obbligo di imparare una serie di comandi da impartire da tastiera nella cosiddetta riga di comando della shell di sistema.

Nei prima anni '80 la Xerox Corporation commercializzò il Xerox 8010 Infor-

Figura 1.1: Lo Xerox Star

mation System, comunemente conosciuto come Xerox Star, un sistema informatico costituito da una serie di workstation collegate in lan[4]. Lo Xerox Star era l'evuluzione dello Xerox Alto sviluppato nel 1973 e mai commercializzato. Xerox Alto è stato il primo computer dotato di interfaccia grafica, esso faceva uso anche di un dispositivo di input oggi comunemente utilizzato: il mouse, dando così vita al primo sistema WIMP (Window, Icon, Menù e Pointing Device)[1]. Lo Xerox Star era molto costoso e non ebbe grande diffusione.

In seguito tale paradigma fu ripreso dalla Apple nel 1983, con il poco fortunato Apple Lisa e successivamente nel 1984 con il più fortunato Macintosh (ribattezzato poi Macintosh 128k per distinguerlo dalle versioni successive)[5].

La diffusione di questi nuovi sistemi aumentava di pari passo con lo sviluppo di nuove tecnologie informatiche, comprendenti il riconoscimento vocale, la

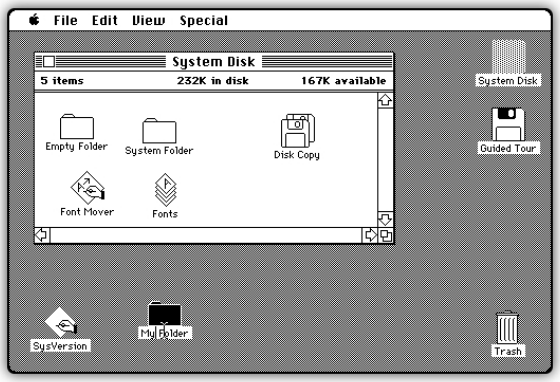

Figura 1.2: Interfaccia grafica del primo Mac OS

Figura 1.3: Macintosh 128K

multimedialità, la presentazione dell'informazione e la realtà virtuale, rendendo disponibili nuove possibilità per lo sviluppo di applicazioni rivolte ad un numero ancora maggiore di persone. Didattica e formazione furono due aree su cui si concentrò molta attenzione. Ambienti interattivi per l'apprendimento, software in supporto alla didattica, simulatori per l'addestramento rappresentarono i maggiori risultati di questo fenomeno. Per realizzare questi nuovi sistemi interattivi, si rendevano necessarie competenze diverse sia da quelle degli psicologi sia da quelle degli sviluppatori di software.

Gli anni '90 furono caratterizzati da nuovi moti di sviluppo tecnologico, reti, computer portatili e porte a infrarossi, la realizzazione di una grande varietà di applicazioni dirette questa volte a tutti divenne una possibilità concreta. Ogni aspetto della vita di un individuo cominciò a essere visto come un'area su cui intervenire con la progettazione e l'integrazione di particolari configurazione tecnologiche. Nuovi modi di imparare, comunicare, lavorare, scoprire e vivere

iniziarono ad essere immaginati.

I team multidisciplinari vennero ulteriormente allargati per includere professionisti con una formazione legata sia ai mezzi di comunicazione che al design, alla produzione grafica, al disegno industriale, alle arti visive e narrative. Sociologi, antropologi, commediografi collaboravano ciascuno con una visione dell'iterazione umana profondamente diversa da quella di uno psicologo. Si pensava che questa pluralità di figure avesse l'insieme di competenze più adatto a progettare i diversi domini di un applicazione di sistemi interattivi di ultima generazione.

Oggi il design dell'iterazione costituisce un grosso business. Consulenti per la realizzazione di siti web, aziende start-up e tutto il settore delle tecnologie mobili ha da tempo compreso come esso giochi un ruolo centrale nella creazione di prodotti interattivi di successo[1]. Gli uffici marketing si stanno rendendo conto di quanto l'usabilità, per esempio di un sito web, abbia un impatto sui risultati delle azioni di valorizzazione della marca, sul numero di accessi, sulla fidelizzazione dei visitatori e sulla soddisfazione del cliente. La presenza o l'assenza di una buona progettazione dell'iterazione ha il potere di decretare il successo o il fallimento di un sito web, un applicazione e persino di un azienda.

1.2 Le Fasi di un Processo di Interaction Design

Per progettare prodotti interattivi usabili è necessario prendere in considerazione chi ne farà uso e il contesto in cui verranno utilizzati. Allo stesso modo è importante la comprensione del tipo di attività che gli utenti svolgeranno in iterazione con quei prodotti. L'adeguatezza di una certa interfaccia, di alcuni dispositivi di input o certi di output dipende dal tipo di attività che deve essere sostenuta dall'artefatto che dovrà essere progettato. Oggigiorno un sistema elettronico fornisce diverse possibilità e una questione cruciale per l'interaction design è: come si può ottimizzare l'iterazione degli utenti con il sistema, l'ambiente o il prodotto in modo tale da sostenere e potenziare lo svolgimento delle attività in cui essi sono coinvolti?

Le scelte progettuali andrebbero prese solo a partire da una profonda comprensione dell'utente. Ciò implica:

- prendere in considerazione ciò che le persone riescono a fare con facilità e ciò che invece fanno con maggiore difficoltà;

- riflettere su cosa potrebbe aiutare le persone nel fare quello che fanno abitualmente;

- capire cosa potrebbe veicolare esperienze qualitativamente di valore;

- ascoltare quello che le persone desiderano e coinvolgerle nel processo di progettazione;

- adottare tecniche consolidate di progettazione basata sull'utente;

Una delle principali ragioni per perseguire un'approfondita conoscenza degli utenti sta nel fatto che utenti diversi hanno bisogni diversi e il design deve essere in grado di rispondere di conseguenza.

Ad esempio Osmosis è stato pensato per l'utilizzo da parte di ricercatori e la sua interfaccia (seppur estremamente minimalista) è stata realizzata per una persona in possesso di requisiti tecnici. Le funzionalità che osmosis offre invece potrebbero essere utili anche a chi non possiede un bagaglio tecnico, per questo sarebbe opportuno progettare e valutare nuovi prototipi che semplifichino l'utilizzo e l'apprendimento dell'interfaccia grafica di osmosis.

La progettazione di nuovi artefatti interattivi è caratterizzata da tre aspetti:

1. coinvolgere gli utenti nello sviluppo;

2. specifici obiettivi di usabilità ed esperienza d'uso dovrebbero essere identificati, concordati e chiaramente descritti fin dall'inizio del processo di design;

3. il processo di design è iterativo;

Nel capitolo successivo vedremo in dettaglio il punto 2 per capire in che modo possiamo migliorare l'interfaccia grafica di osmosis.

1.3 Gli Obiettivi dell'Interaction Design

Sono essenzialmente divisi in due: obiettivi di usabilità che hanno a che fare con specifici criteri di usabilità (per esempio l'efficenza) e gli obiettivi di esperienza d'uso che si riferiscono il larga misura alla qualità che dovranno caratterizzare l'esperienza dell'utente (per esempio la piacevolezza estetica).

1.3.1 Obiettivi di Usabilità

L'usabilità è definita dall' ISO[1], come l'efficacia, l'efficienza e la soddisfazione con le quali determinati utenti raggiungono determinati obiettivi in determinati contesti [6]. In pratica definisce il grado di facilità e soddisfazione con cui l'interazione uomo-strumento si compie. Il livello di soddisfacimento dei requisiti di usabilità di un sistema è rappresentato dal raggiungimento dei seguenti obiettivi come descritto in uno dei testi classici dell'interaction design[1]:

- efficacia;
- efficenza d'uso;
- sicurezza d'uso;
- utilità;
- facilità di apprendimento;
- facilità di ricordo;

Vediamoli più in dettaglio.

L'efficacia rappresenta un obiettivo molto generico, che si riferisce alla capacità del sistema di fare quello per cui è progettato.

L'efficenza riguarda il modo attraverso cui il sistema aiuta gli utenti a portare a termine i propri compiti.

Il problema della **sicurezza** riguarda invece la protezione dell'utente da situazioni pericolose o indesiderabili. Il primo aspetto ergonomico si riferisce alle condizioni dell'ambiente di lavoro delle persone. Mentre il secondo riguarda la

[1] International Organization for Standardization

capacità di fare in modo che qualsiasi utente in qualsiasi tipo di situazione non sia mai esposto al pericolo di compiere accidentalmente azioni indesiderate. Rendere i sistemi più sicuri implica:

1. impedire che l'utente compia errori irreparabili riducendo il rischio che tasti sbagliati vengano attivati per errore;

2. fornire all'utente un modo per porre rimedio a eventuali errori commessi.

Un sistema interattivo sicuro dovrebbe generare fiducia e permettere all'utente di esplorare l'interfaccia per scoprire nuove funzionalità. Fra i meccanismi di sicurezza di una GUI rientrano tutte le opzioni di annullamento dell'input e le finestre di dialogo che richiedono conferma di un'azione fornendo all'utente una possibilità di riconsiderare le proprie intenzioni. Ad esempio premendo l'icona X il sistema dovrebbe chiedere al'utente se vuole salvare il file corrente, questo per prevenire il caso in cui il tasto sia stato premuto per errore e il che il file non sia stato salvato.

L'**utilità** riguarda invece la capacità del sistema di fornire le funzionalità necessarie per compiere le attività che desidera o deve necessariamente portare a termine.

La **facilità di apprendimento** riguarda la semplicità con cui si impara a usare un sistema. E' noto che le persone non amano dover dedicare troppo tempo a capire come usare un sistema. Un chiaro esempio è fornito dalla regola dei 10 minuti introdotta da Nelson[7]. Essa dice che i nuovi utenti dovrebbero imparare ad usare un sistema in meno di dieci minuti. Se non ci riescono il sistema è fallito. Naturalmente questa regola non è adottabile in quei sistemi anche complessi che richiedo uno studio tecnico approfondito.

La **facilità di ricordo** è la proprietà di un sistema legata alla facilità di ricordarne le modalità di utilizzo dopo che sono state apprese. Gli utenti non dovrebbero continuamente imparare come compiere gli stessi task e questo accade quando le operazioni da compiere sono confuse, senza logica o mal disposte. Gli utenti hanno bisogno che gli si ricordi cosa devono fare, per esempio possono essere utili icone e voci di menù particolarmente significative che raccolgano una sequenza di operazioni.

1.3.2 Gli Obiettivi di Esperienza d'Uso

Oltre a concentrarsi sull'innalzamento dell'efficenza e della produttività in ambito lavorativo, l'interaction design sempre di più si confronta con la progettazione di sistemi:

- in grado di dare soddisfazione a chi li usa;
- piacevoli da usare;
- divertenti;
- utili;
- capaci di sostenere le motivazioni delle persone;
- esteticamente gradevoli;
- capaci di alimentare la creatività delle persone;
- gratificanti;
- in grado di soddisfare i bisogni legati alla sfera emotiva delle emozioni;

Tali possibilità sono legate alle caratteristiche dell'esperienza d'uso, vale a dire al modo in cui gli utenti vivono l'iterazione con il sistema. Questo comporta un analisi dell'esperienza d'uso in termini soggettivi differenziandosi così dai più oggettivi obiettivi di usabilità.

Per un designer è quindi opportuno raggiungere il giusto trade-off tra i due valutando le conseguenze delle diverse combinazioni possibili rispetto ai bisogni degli utenti. Per farlo può seguire delle astrazioni di validità generale create grazie ad un mix di conoscenze teoriche, esperienza e buon senso, che danno vita ad un insieme di principi di progettazione.

1.4 Principi di Progettazione e Usabilità

I principi di progettazione nascono proprio con l'intenzione di aiutare i prgettisti nel loro lavoro[8] e non vanno intesi come specifiche per il design di interfacce

concrete, ma piuttosto come promemoria che ricordano al designer alcuni elementi da prendere in considerazione.

Quelli più conosciuti riguardano come determinare che cosa un utente dovrebbe vedere e poter fare nel corso dello svolgimento di una certa attività in iterazione con un sistema. Vediamo i principali che torneranno molto utili nell'analisi del prototipo della GUI[2] di osmosis. Per una trattazione più ampia si rimanda al lavoro di Don Norman, ne *La caffetteria del masochista [9].*

1. **Rendere le cose visibili**. Più le funzioni sono visibili, più l'utente avrà chiaro cosa fare e in che modo farlo;

2. **Fornire feedback**. Il principio del feedback consiste nel fatto che qualsiasi strumento noi utilizziamo deve inviare informazioni di ritorno per confermare che una certa azione è andata a buon fine e permettendo alla persona di continuare nello svolgimento delle sue attività;

3. **Fornire vincoli**. L'idea di vincolare le azioni nasce dall'esigenza di limitare le possibilità di iterazione fra utente e sistema in un dato momento. Lo si può fare in diversi modi, per esempio nella progettazione di interfaccie grafiche è ormai prassi comune quella di disattivare le voci di menù, facendo in modo che l'utente possa eseguire solo le azioni possibili al punto in cui si trova. Uno dei vantaggi di questo modo di stabilire vincoli è quello di impedire all'utente di scegliere azioni inappropriate, riducendo cosi la possibilità di commettere errori. Norman[10] classifica i vincoli in tre categorie: fisici, logici e culturali. I primi si riferiscono al modo in cui gli oggetti fisici impediscono certi movimenti rendendo chiaro il loro utilizzo. I vincoli logici dipendono dalla comprensione delle persone di come funziona il mondo. Essi si affidano al buon senso circa le azioni che sono da svolgere. Ad esempio disabilitare voci di menù quando non appropiate per il compito corrente crea un vincolo logico. I vincoli culturali si fondano su convenzioni apprese, come l'uso di segnali sonori per indicare un pericolo. Una volta appresi e accettati all'interno del gruppo, diventano immediatamente convenzioni apprese. Due esempi di convenzioni già apprese e comuni sono

[2]Graphics User Interface

l'uso di finestre per la presentazione dei contenuti sul desktop e l'uso di icone per indicare le operazioni possibili.

Figura 1.4: Un menù che mostra la disponibilità limitata di scelte come esempio di vincolo logico all'azione. Le voci in ombra rappresentano le opzioni non abilitate.

4. **Fornire un mapping naturale**. Questo principio si inspira alla relazione fra i dispositivi di controllo e i loro effetti nel mondo.

5. **Garantire la consistenza**. Quest principi si riferisce alla buona norma di progettare interfacce che usino operazioni simili mediate da elementi simili, per svolgere compiti simili.
   Le interfacce inconsistenti invece permettono violazioni alla regola, Ad asempio quando certi elementi grafici possono essere selezionati con il click sinistro del mouse e altri con il destro. Il problema in questo caso sta nella sua arbitrarietà che rende difficile agli utenti ricordarsi cosa fare. Pertanto rispettare questo principio porta alla maggior facilità di apprendimento e d'uso.
   Nella progettazione di interfaccie grafiche molto complesse in cui si devono rendere disponibili svariate operazioni si potrebbero creare categorie di comandi che raggruppano sotto-insiemi di operazioni. Per esempio nell'organizzazione di un menù sotto la voce file appaiono tutte le operazioni che hanno a che fare con essi, come salva, chiudi apri ecc.

6. **Fornire affordance**. Il termine affordance indica la proprietà di un oggetto di far capire come deve essere usato. Questo termine è stato cogniato da Norman[10] alla fine degli anni ottanta e viene usato per descrivere come progettare interfacce in modo che sia chiaro come utilizzarle. A un livello base, fornire affordance significa fornire un indizio. Si parla di elementi quali bottoni, icone, link e barre di scorrimento in termini di come far sì che sia evidente come vanno usati: le icone devono ivitare al click, le barre di scorrimento a muoversi verso l'alto o il basso, i bottoni devono farsi premere.

Capitolo 2

Analisi dell'Interfaccia Grafica di Osmosis

2.1 Osmosis

Osmosis (Open Source and Multi PlatfOrm Shared Interactive Screen)[11] è un'applicazione open-source e multi-piattaforma che offre una lavagna virtuale condivisa sulla quale poter disegnare. Il suo scopo è quello di fornire uno strumento con cui poter collaborare in maniera virtuale. Offre infatti uno spazio su cui poter disegnare e scrivere che viene visualizzato dagli altri utenti che condividono la sessione di lavoro.

L'applicazione è composta da diversi elementi principali: l'interfaccia grafica che comprende il menù e la lavagna di disegno, un parser che si occupa di convertire i dati e un protocollo di comunicazione che gestisce il flusso di informazioni trasmesse.

Vale la pena di spendere due parole su questi due ultimi punti. Osmosis si appoggia a Skype per la parte di comunicazione di rete utilizzando delle API[1] sviluppate in Java (Skype4Java)[12] che forniscono tutte le primitive necessarie a gestire la comunicazione di stringhe. Il parser si occupa di tradurre gli oggetti visualizzati sulla lavagna in opportune stringhe (e viceversa) che possono così essere trasmesse.

[1] Application Programming Interface

L'intero sviluppo di Osmosis è il risultato di un lavoro di tirocinio più tesi svolto come prova finale di una laurea triennale in Informatica sotto la supervisione del dott. Ugo Dal Lago. Vista la mole del lavoro e il tempo necessario alla sua realizzazione, il prototipo dell'interfaccia grafica è stato sviluppato in maniera essenziale e fornisce soltanto le funzioni base necessarie per il funzionamento dell'applicazione.

L'interfaccia non è stata oggetto di uno studio approfondito che considerasse i principi di progettazione e di usabilità. Nel paragrafo seguente l'analizzeremo in dettaglio per capire meglio quali sono i suoi punti di forza (e quali no) prima di realizzare un nuovo prototipo.

2.2 Analisi Tecnica

L'interfaccia grafica di osmosis è composta da cinque elementi principali: la barra dei menù posizionata in alta a sinistra, il pannello di disegno posizionato al centro, la barra laterale dei bottoni posizionati sulla sinistra, la barra inferiore con diverse funzioni e i tre pulsanti in alto a sinistra che permettono di gestire la finestra. Di seguito verranno analizzate le funzionalità che vengono fornite (e non) con questo prototipo di gui eseguendo diversi prove. Successivamente verrà studiato come un utente normale (senza conoscenze informatiche) si approccerebbe al suo utilizzo. Questo sarà molto utile per la realizzazione di un prototipo ottimizzato in quanto punto chiave da seguire nei principi di progettazione dell'Interaction Design.

2.2.1 Il Menù

Il menù è composto da tre voci File, Edit e Network ognuna di esse consente di aprire un sottomenù contenente diverse voci che sono non sono state organizzate in maniera logica. Al menù è possibile accedere solo attraverso il click con il tasto sinistro del mouse, non sono presenti scorciatoie da tastiera o combinazioni di tasti. Vedremo le operazioni che vengono fornite e per ognuna sarà fornita un analisi sulle funzionalità che offre.

1. il menù File contiene Load, Save, About Osmosis, Export to, e Exit. Più in dettaglio:

   - Save. Questa voce permette di salvare il disegno su di un file. Alla sua pressione appare una finestra di dialog che consente di inserire il nome del file e la sua posizione nel file system. E' possibile anche selezionare un file già presente nel file system. Questa operazione funziona solo nel caso base in cui si salvi un disegno in un nuovo file. In particolare Save non funziona se:

     · si tenta di salvare più volte uno stesso file modificato.

     · si tenta di salvare un file che era stato aperto in precedenza.

     Inoltre l'operazione non si occupa di:

     · gestire l'estensione dei file. I file salvati non hanno un estensione propria di osmosis;

     · notificare all'utente che il salvataggio è avvenuto;

     · fornire un sistema per gestire la sovrascrittura di un file;

   - Load. Questa voce consente di caricare un file. In realtà funziona solo se la lavagna è vuota e se non lo è disegna il file sovrapponendolo a quello esistente. Quindi load non permette di:

     · salvare il disegno presente sulla lavagna;

     · caricare il nuovo file su di un pannello vuoto;

     Inoltre testandolo in casi particolari che comunque sono molto frequenti si osserva che:

     · è possibile selezionare qualsiasi formato, tuttavia i file non vengono caricati e il sistema non l'utente di nulla (limitazioni, errori, ecc);

     · una volta caricato il file non è possibile sincronizzare la lavagna con eventuali utenti connessi. Gli utenti visualizzano lavagne differenti.

   - About Osmosis. La pressione di questa voce nel menù apre una finestra con al suo interno la licenza GNU/Linux. Per chiuderla è necessario premere il pulsante ok che appare sotto la licenza oppure la X del

gestore finestre. La finestra che si apre non è grande abbastanza per contenere il testo. La licenza non è formattata bene infatti le prime cinque righe sono vuote.

- Export to. Passando con il mouse sopra alla voce se ne attiva un altra (image). Premendola è possibile salvare come immagine il contenuto del pannello. Viene aperta una finestra di dialog che consente di inserire il nome, il percorso. All'utente non viene fornito nessun modo per capire in che formato sta salvando il disegno sulla lavagna. Anche qua non è disponibile nessuna notifica dell'avvenuta esportazione.
- Exit. Chiude Osmosis. Non vengono proposti dei metodi per salvare il lavoro corrente. La pressione involontario ti questa voce causa quindi la perdita del disegno.

2. Il menù edit contiene la voce clean.

   - Clean. Consente di pulire il disegno proponendo una lavagna vuota. Si osserva che non sono forniti sistemi per annullare l'operazione. In oltre l'operazione funziona solo in locale, non avviene nessun aggiornamento della lavagna condivisa e gli utenti connessi visualizzeranno ancora il disegno che è stato appena pulito.

3. Il menù network contiene la voce ap2ap

   - ap2ap. Questa voce permette di connettersi a Skype. Al posto di Ap2Ap si potrebbe segliere un nome più significativo. Una volta connessi viene mostrata una finestra di dialogo in cui sono visualizzati gli utenti con cui è possibile condividere la lavagna. Se Skype non è disponibile o la connessione ha dei problemi viene mostrato un messaggio di errore.

2.2.2 Il Gestore della Finestra

Per gestore della finestra si intendono le tre icone nell'angolo in alto a destra. Il loro uso è ormai assodato nelle interfaccie grafiche basate su prototipo WIMP[2][1].

[2]Window, Icon, Menù e Pointing Device

Vediamo le funzioni che offrono le tre icone:

- la barretta permette di ridurre ad icona osmosis;
- il quadrato permette di ingrandire la finestra a tutto schermo;
- la X permette di chiudere il programma; un'osservazione su quest'ultima. La pressione del tasto chiude irrimediabilmente Osmosis senza offrire la possibilità di salvare il lavoro corrente.

2.2.3 La Barra Laterale Sinistra

Contiene sette bottoni vediamoli in dettaglio.

- 5 bottoni servono per selezionare un colore, sono disponibili il nero (attivo di default) il rosso, il giallo, il verde e il blu.
- Gli altri due hanno una S e una I che significano Inserire e Selezionare. Se premo S posso poi selezionare nella lavagna una linea disegnata. Questa verrà colorata di viola e poi premendo il tasto canc nella tastiera la posso cancellare. Se premo I si attiva la funzione inserimento stringa. Cioè premendo il tasto destro del mouse nella lavagna ci verrà scritto il contenuto della text area sottostante. I due bottoni sono tra loro mutualmente esclusivi (o uno o l'altro) e se nessuno di questi due tasti è attivo allora posso disegnare sulla lavagna.

2.2.4 La Barra in Basso

Consente di svolgere diverse funzioni: Posso selezionare in un menù a tendina la grandezza della linea. Sono disponibili dieci diversi livelli di spessore. Di default ha valore 1. Questo menù è posizionato nell'angolo in basso a sinistra. Il pannello centrale consente, una volta selezionato con il click sinistro del mouse, di scrivere a tastiera nella modalità vista sopra. Sulla destra troviamo un pannellino che ci informa dello stato della connessione con Skype. Ha due possibili valori: Connected e Not Connected. Lo stato Connected ha tre ulteriori valori: Busy, Avaible e Writing che indicano lo stato della lavagna rispettivamente significano

impegnata (l'utente remoto sta utilizzando la lavagna), disponibile oppure che l'utente locale sta scrivendo.

2.3 Ricapitolando

A questo punto dell'analisi possiamo notare che l'interfaccia grafica sia abbastanza minimale. Questo dovrebbe rendere il suo utilizzo e apprendimento molto semplice, purtroppo non è così. Alcune features (come l'inserimento delle stringhe nel pannello di disegno) non sono affatto immediate, altre hanno invece un utilizzo insolito da come ci si aspetterebbe (come ad esempio la funzione Load) e altre ancora non realizzano completamente il loro compito (come ad esempio Save). Nella realizzazione del nuovo prototipo sarà importante garantire che le funzionalità offerte dalla gui eseguano il compito per cui sono state pensate, e che lo facciano offrendo all'utente indizi su come utilizzarle. In questo senso la funzionalità Clean presente nel menù Edit è concettualmente simile al classico Nuovo presente solitamente all'interno del menù File.

L'organizzazione generale della finestra è semplice e pulita, il posizionamento del menù in alto a sinistra, la barra dei bottoni sulla sinistra rendono agevole l'utilizzo di osmosis e l'utente può fin da subito dedicarsi al suo utilizzo scrivendo sulla lavagna che occupa una posizione di primo piano all'interno del layout grafico.

2.3.1 L'Interfaccia e l'Utente

Come si è visto nei primi capitoli, la progettazione di interfacce grafiche pone molta considerazione nei riguardi dell'utente che utilizzerà l'artefatto. In questo senso l'interfaccia di Osmosis non è stata oggetto di uno studio approfondito. Gli utenti non sono stati coivolti nello sviluppo del prototipo, per questo è stato opportuno valutare il loro approccio all'utilizzo dell'interfaccia di Osmosis. Sono state selezionate due persone con differenti bagagli tecnici. Entrambi sono incappati nei problemi di funzionalità già ampiamente discussi precedentemente, tuttavia la loro prima impressione è stata positiva. Il posizionamento del menù, delle barra laterale e la lavagna al centro garantisco un immediata comprensione

generale di quello che è possibile fare.

I problemi principali si sono riscontrati nell'utilizzo della lavagna in modalità inserimento testo e cancellazione linea. E' molto difficile capire come utilizzarli, sopratutto perchè il sistema non fornisce un adeguato feedback. Ad esempio l'icona dell'inserimento testo è un bottone con una I, una volta premuto la funzione inserimento è attiva ma non viene suggerito come fare per utilizzarla, il cursore resta una freccetta e l'utente si ritrova completamente spaesato. Premendo il bottone che ha come icona una S viene attivata la cancellazione linee e per tornare a disegnare sulla lavagna è necessario disattivarla, ripremendo il bottone. Ricapitolando per disegnare linee occorre disattivare i bottoni I e S, per inserire testo occorre selezionare I e per cancellare line occorre selezionare S. Osserviamo quindi che vengono utilizzati due metodi differenti per attivare le diverse opzioni e questo causa inconsistenza e difficoltà di apprendimento.

Nel nuovo prototipo verrà fornito un sistema per renderne più semplice ed intuitivo l'utilzzo, fornendo anche diversi feedback che aiuteranno l'utente a comprenderne il funzionamento.

Capitolo 3

Il Nuovo Prototipo di Interfaccia

In questo capitolo verrà presentato il nuovo prototipo di interfaccia, analizzando dapprima il suo aspetto e le funzionalità che offre ed in seguito verrà spiegato come sono stati ottenuti i principali obiettivi di usabilità.

3.1 I Componenti

La nuova interfaccia grafica appare molto simile alla precedente, con una differenza sostanziale. E' stata eliminata la barra sottostante alla lavagna. L'indicatore di status è stato spostato nella barra laterale e il metodo di inserimento del testo è stato modificato eliminando la text area sottostante la lavagna e rendendo l'interfaccia più semplice da utilizzare. Il layout grafico è composto da quattro elementi principali. In alto sulla sinistra abbiamo un menù composto da quattro voci: File, Tools, Network e Help. Sulla destra, in alto, troviamo tre icone per la gestione della finestra e sulla sinistra appare una barra contenente diversi pulsanti. Molto evidente in posizione centrale abbiamo la lavagna su cui è possibile scrivere o disegnare fin dall'avvio di Osmosis.

3.1.1 Il Menù

Le voci del menù in alto a sinistra consentono di accedere ad altri menù. Le voci in esso contenute sono raggruppate in maniera logica così da garantire più consistenza che porta ad una maggior facilità di apprendimento e d'uso. Ad

alcune voci ci si può accedere utilizzando particolari combinazioni di tasti che sono state scelte considerando le interfaccie grafiche di uso comune. Vediamole in dettaglio:

- **menù File**: contiene tutte le operazioni che hanno a che fare con i file. Ci si accede premendo Alt+F.
  - New: utilizza una nuova board, ci si accede con Ctrl+n.
  - Open: apre un file, ci si accede con Ctrl+o.
  - Save: salva un file, ci si accede attraverso Ctrl+s.
  - Save as: salva un file scegliendo il nome.
  - Export to: consente di esportare il disegno come un immagine.
  - Exit: esce da osmosis, ci si accede anche premendo Ctrl+e.

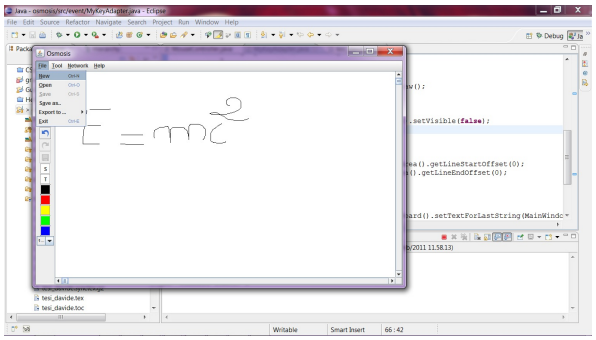

Figura 3.1: Il menù File della nuova interfaccia.

- **menù Tools**: contiene operazioni di utilità per l'elaborazione e la gestione del disegno sulla lavagna. E' accessibile utilizzando Alt+t.
  - Undo: annulla l'ultima modifica fatta alla lavagna, accessibile attraverso Ctrl+z.
  - Redo: ripete l'ultima azione, accessibile attraverso Ctrl+y.
  - Canc line: consente di selezionare una linea o una stringa e di cancellarla premendo CANC, accessibile attraverso Ctrl-c.
  - Write text: consente di inserire del testo sulla lavagna.

- **Menù Network**: contiene operazioni che hanno a che fare con la connessione a Skype e ci si accede tramite Alt+n.
  - Connect Ap2Ap: connette Osmosis a Skype.
- **Menù Help**: contiene operazioni di supporto e che danno informazioni su osmosis, accessibile attraverso Alt+h.
  - About Osmosis: visualizza la licenza GNU/Linux.

3.1.2 Il Gestore della Finestra

In alto sulla destra appaiono tre icone. Rispettivamente esse sono:

- Un trattino: consente di ridurre la finestra di Osmosis ad icona nella barra delle applicazioni aperte.
- Un quadrato: consente di ingrandire la finestra a tutto schermo.
- Una X: consente di chiudere Osmosis.

3.1.3 Il Menù ad Accesso Rapido

Sulla sinistra appare una barra con dei bottoni. Essi consentono di utilizzare particolari opzioni di scrittura, di accedere ad alcune voci del menù utilizzate frequentemente e forniscono informazioni sullo status di Osmosis. Partendo dall'alto esso sono:

- un'icona rappresentante un mondo. Indica lo status e può avere due valori, Connected e Not Connected. Quando l'utente è connesso abbiamo altri tre valori che indicano l'utilizzo della board (Avaible, Busy, Writing);
- un'icona con una freccia a sinistra. Invoca la funzione undo;
- un'icona con una freccia a destra. Invoca la funzione redo;
- un'icona con un dischetto. Effettua il salvataggio;
- un'icona con una X: consente di selezionare una linea dalla lavagna e di rimuoverla premendo il tasto Canc,

- un'icona con una T. Consente di inserire del testo sulla lavagna,
- cinque bottoni colorati. Consentono di cambiare il colore del tratto,
- un piccolo menu a tendina. Consente di selezionare la grandezza del tratto,

Si è cercato di rendere l'interfaccia ancora più semplice da utilizzare, tuttavia le funzionalità che offre sono state ulteriormente approfondite prendendo in considerazione casi inconsueti di utilizzo che possono presentarsi durante il salvataggio dei file, la modifica al documento, ecc.

3.1.4 La Lavagna

La lavagna su cui disegnare è rimasta sostanzialmente invariata rispetto al prototipo precedente. Sono disponibili due barre di navigazione che consentono di scorrere il foglio a destra e a sinistra e in alto e in basso, rendendo virtualmente disponibile una lavagna infinita.

3.2 Funzionalità e Sicurezza

In un'interfaccia grafica si possono presentare diverse situazioni, in ognuna delle quali è importante garantire che l'utente non perda il lavoro svolto (il disegno sulla lavagna) e rendere disponibili dei metodi per consentirgli di annullare l'operazione eventualmente scelta. In termini di usabilità questa proprietà garantisce la sicurezza dell'utente. E' necessario quindi considerare e prevedere le diverse situazioni in cui l'utente potrebbe tentare di utilizzare una determinata operazione, considerando gli effetti che avrebbe sul disegno, sul file e sulla condivisione della lavagna. Questi punti possono essere chiariti da un esempio. L'utilizzo della voce Open apre un file e lo disegna sulla lavagna. Che effetti ha l'operazione sulla lavagna già esistente? Il disegno presente verrebbe cancellato e sostituito con quello rappresentato nel file aperto. In questo caso è necessario fornire un sistema che consenta di salvare il lavoro corrente sulla lavagna **prima** di effettuare l'apertura di un file. Si tratterebbe di una sorta di salvataggio in extremis, invocato per precauzione nel caso l'utente si fosse dimenticato di salvare. Inoltre come si comporterebbe la lavagna dell'utente connesso in remoto? Un

attimo ho un disegno e l'attimo dopo ne ho un altro completamente differente? Questi casi vanno gestiti fornendo un sistema efficiente che consenta agli utenti di salvare il disegno prima di sincronizzare le lavagne.

In seguito verranno distinte le operazioni in due categorie: quelle "critiche" che possono causare una sostanziale modifica al disegno della lavagna (per esempio cancellandola) e che quindi necessitano di più accorgimenti, e quelle più "soft" che offrono servizi di utilità per la gestione del disegno (come la cancellazione di una linea).

3.2.1 Critical Operation

Le voci del menù file New, Open, Exit oppure la pressione del bottone X del gestore delle finestre, invocano delle operazioni che causano una modifica alla lavagna. In particolare attraverso New possiamo visualizzare una lavagna vuota, usando Open possiamo aprire un documento precedentemente salvato e visualizzarlo sulla lavagna mentre utilizzando la voce Exit oppure premendo l'icona X il documento corrente e Osmosis vengono chiusi. In tutti questi casi andrebbe reso disponibile un sistema per:

1. salvare il disegno;

2. annullare l'operazione.

Questi punti sono stati realizzati fornendo delle finestre di dialogo che consentono in qualunque circostanza di evocare le funzioni Save e Save as che servono proprio a soddisfare il punto 1. La gestione dei salvataggi sarà trattata esaustivamente nel prossimo paragrafo.

3.2.2 Soft Operation

Le altre voci del menù e i bottoni presenti nella barra laterale non causano una modifica della lavagna ma offrono importanti (se non essenziali) operazioni, anche le funzionalità offerte di undo e redo vengono definite soft operation in quanto l'uso combinato dei due permette di annullare una modifica alla lavagna. In questi casi non è necessario nessun meccanismo di salvataggio in extremis.

Tuttavia verranno adottati alcuni accorgimenti per migliorare l'iterazione dell'utente con l'interfaccia. Queste operazioni verranno spiegate successivamente in un paragrafo a parte.

3.3 Sicurezza: il Salvataggio e le Finestre di Dialogo

La gestione delle operazioni critiche avviene attraverso delle finestre di dialogo progettate ad hoc, cioè che assumono valori differenti adatti alle diverse situazioni che possono presentarsi. Le finestre di dialogo all'occorrenza propongono il salvataggio dei file, gestiscono la sovrascrittura e guidano l'utente nello svolgere le operazioni in modo sicuro.

3.3.1 Il Salvataggio

L'operazione di salvataggio è essenziale, questo può assumere due differenti comportamenti che variano a seconda dello stato del file. Abbiamo due casi:

1. il file non è stato salvato in precedenza;

2. il file è stato salvato in precedenza.

Il primo caso capita sempre almeno una volta ed è necessario fornire all'utente la possibilità di scegliere un nome e un percorso per effettuare il salvataggio. Questa operazione viene fornita tramite una finestra di dialogo che consente di scegliere il nome e il percorso. Successivamente vengono effettuati dei controlli sull'esistenza di un file con gli stessi parametri, se il file esiste viene proposta un ulteriore finestra di dialogo che avverte l'utente e gli domanda se vuole sovrascriverlo. Per realizzare queste ulteriori finestre è stata creata la classe *SaveOverwriteFileChooser*.

Le finestre di dialogo che vengono utilizzate sono bloccanti, cioè finchè non viene effettuata una scelta tra le opzioni fornite (i bottoni) non è possibile accedere alle altre funzionalità dell'interfaccia (come disegnare). Questo primo caso rappresenta anche il comportamento dell'interfaccia all'invocazione della voce

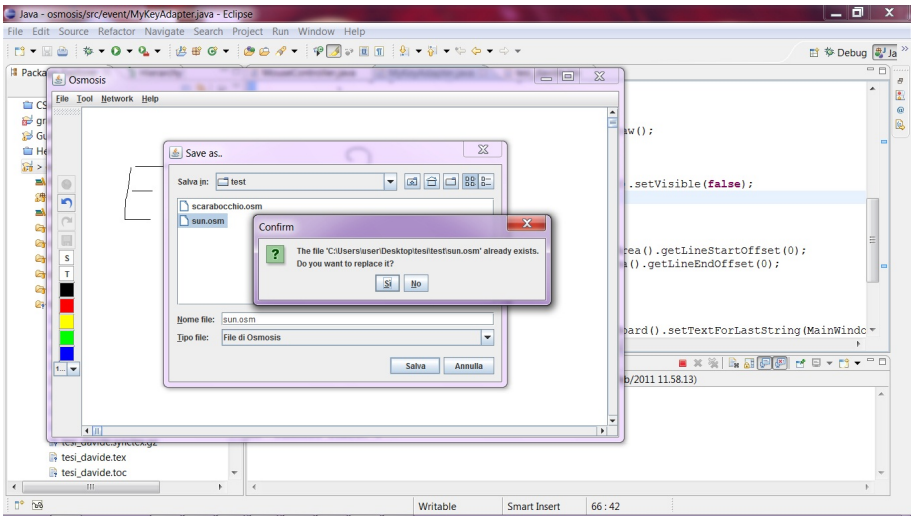

Figura 3.2: La finestra di dialogo che permette la gestione della sovrascrittura.

Save-as.

Nel secondo caso invece esiste già un file in cui è salvato lo stato della lavagna, in questa circostanza l'operazione di salvataggio viene effettuata senza proporre finestre di dialogo. Questo caso invece rappresenta la voce save nel menù file.

Ci si potrebbe domandare perchè sono state fornite due voci che in sostanza realizzano la stessa operazione. In particolare rispetto al prototipo precedente di gui è stata aggiunta la voce Save. Questa velocizza il processo di salvataggio rendendolo immediato e risparmiando così l'utente dal dover ogni volta selezionare il nome del file e sovrascriverlo, risultando quindi estremamente utile ed efficiente.

Assieme alle due operazioni base di salvataggio viene fornita, attraverso la voce Export to, la possibilità di salvare il disegno sulla lavagna come immagine jpeg. Anche in questo caso vengono usate delle finestre di dialogo che assistono l'utente ed evitano la sovrascrittura involontaria di file. Le tre operazioni viste dispongono inoltre di un meccanismo che crea la giusta estensione del file, il quale si rivelerà particolarmente utile nell'operazione di apertura di un file. Nel prossimo sottoparagrafo vedremo perchè.

3.3.2 Le Estensioni ed i Caricamenti

Nel prototipo precedente invocando la voce associata al comando Open era consentito aprire qualsiasi formato. Il file non veniva però realmente aperto e l'interfaccia di Osmosis non si preoccupava di visualizzare alcun genere di messaggio. Se Osmosis era eseguito da un editor di sviluppo, come Eclipse[1], apparivano in output una serie di errori. Per gestire correttamente i file supportati è stato necessario creare un formato per Osmosis. La stringa .osm rappresenta l'estensione scelta e viene aggiunta ogni volta che si effettua il salvataggio al nome del file. L'invocazione di Export To invece produce l'aggiunta dell'estensione .jpeg.

Durante le fasi di salvataggio, apertura ed esportazione vengono utilizzati dei "filtri" che assistono l'utente nelle operazioni. Tali filtri permettono di visualizzare nelle finestre di dialogo i soli file con l'estensione opportuna (.osm o .jpeg) agevolando, per esempio, l'operazione di caricamento in cui l'utente viene incoraggiato a selezionare i file con estensione .osm.

Non viene comunque esclusa la possibilità di tentare di aprire altri file, per questo viene effettuato un ulteriore controllo (sempre sull'estensione) e all'occorrenza viene mostrato un messaggio di errore in una finestra di dialogo che informa l'utente che il formato selezionato non è supportato.

3.3.3 Salvataggi in Critical Operation

Le voci del menù New, Open, Exit e l'icona X invocano operazioni che assumono comportamenti differenti a seconda degli scenari che possono presentarsi. Abbiamo tre possibili scenari:

1. **Il file è stato salvato in precedenza e non ha subito modifiche, oppure è vuoto**. In questa circostanza non rischio di perdere il lavoro, quindi le funzioni invocate tramite New, Open, Exit e il bottone X vengono immediatamente eseguite perchè non ci sono situazioni spiacevoli da prevedere.

2. **Il file è stato salvato e ha subito modifiche**. In questo caso l'utilizzo dei comandi del menù causerebbe la perdita delle ultime modifiche effettuate

[1] Piattaforma di sviluppo open-source

ma non dell'intero documento. L'utente in precedenza ha effettuato un salvataggio scegliendo un nome per il file, quindi una finestra di dialogo gli domanda se vuole risalvarlo, sono possibile due opzioni rappresentate da due bottoni più l' icona di chiusura:

- SI: Salva il file eseguendo le operazioni di salvataggio ed esegue l'operazione scelta (New, Open, Exit, o X).
- NO: non salva il file ed esegue l'operazione.
- Icona X: annulla l'operazione.

3. **Il file non è stato salvato e ha subito modifiche**. Questa situazione è differente dalla seconda perchè si deve avvertire l'utente che il disegno sulla lavagna non è stato salvato. Una prima finestra di dialogo viene mostrata, essa contiene un messaggio di avvertimento "Save current file?", due bottoni e un icona:

   - SI: causa l'apertura della finestra di dialogo utilizzata per Save as e delle procedure a lei associate.
   - NO: causa l'immediata esecuzione dell'operazione richiesta (ad esempio Exit).
   - Premendo invece l'icona X viene annullata l'operazione invocata.

3.4 Soft Operation

Questo tipo di operazioni non necessitano di controlli sullo stato del file e quindi non sono stati utilizzati particolari accorgimenti. Tuttavia si è cercato di semplificare al massimo il loro utilizzo in modo da agevolare l'utente nell'apprendimento e nella memorizzazione dell'uso. Per esempio sono stati aggiunte delle etichette che spiegano brevemente quello che accade se viene invocata l'operazione.

Un'altra importante tecnologia che si è adottata è la disattivazione/attivazione delle voci del menù quando queste non sono consentite. Questo accorgimento assiste l'utente impedendogli di utilizzare alcune operazioni che non sareb-

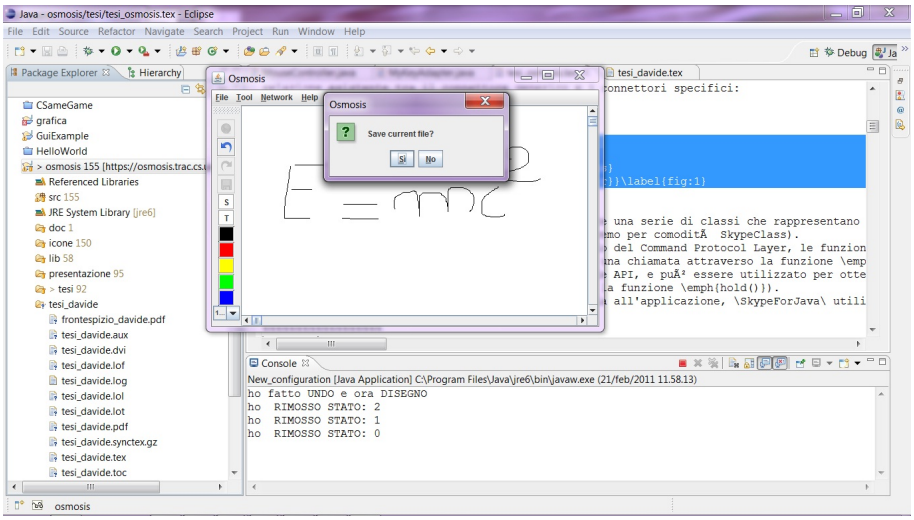

Figura 3.3: La finestra di dialogo che sospende la chiusura dell'applicazione

bero altrimenti appropiate. Disattivare le icone e le voci di menù non pertinenti rappresenta un **vincolo logico** che andrebbe sempre considerato nel progettare un' interfaccia. Vediamo ora in dettaglio le operazioni principali: undo e redo, l'inserimento di testo nella lavagna, la cancellazione di una linea (o testo), e più in generale tutti gli altri.

3.4.1 Undo e Redo

Queste due voci permettono di invocare importanti funzionalità che sono ormai di uso frequente nelle interfacce grafiche. Undo consente di annullare l'ultima operazione eseguita e redo di ripeterla. Nel nostro contesto per operazione si intende il disegno di un tratto sulla lavagna o l'inserimento di testo, quindi utilizzando undo la lavagna apparirà come era prima dell'inserimento dell'ultima linea (o stringa). Inoltre è possibile eseguire n undo in serie riportando la lavagna indietro di n linee e/o stringhe. La funzionalità redo consente di annullare uno o più undo eseguiti in successione.

Undo e Redo sono stati inseriti come voci nel menù tool e come icone nella barra laterale. Il loro posizionamento strategico incrementa l'**accessibilità** delle operazioni.

Date le peculiarità di undo e redo è opportuno valutare i casi in cui questi

possono essere utilizzati dall'utente. Le voci appaiono disabilitate quando il loro utilizzo è sconveniente, porta ad errori nella gestione interna oppure creano inconsistenza nella lavagna condivisa. Vediamo ora le politiche che sono state scelte per rendere disponibile undo e redo. Undo è sempre disponibile tranne i casi in cui:

- **La lavagna è vuota.** Non esiste nessuno stato precedente al disegno corrente.
- **Ho appena aperto un file.** Come posso sapere con certezza qual'è l'ultimo tratto inserito?

Redo è disponibile dopo che ho eseguito almeno un undo e non lo è quando:

- **Dopo aver eseguito almeno un undo disegno sulla lavagna.**
- **Ho eseguito tanti redo fino a tornare allo stato della lavagna come era prima degli undo.** Un esempio può chiarire meglio questo punto. Ho appena disegnato due tratti, posso quindi applicare undo due volte prima di tornare alla lavagna vuota. Di conseguenza posso eseguire al massimo due redo dopo di chè la voce viene disabilitata.

Undo e redo sono implementati nella classe *UndoRedoDescription* contenuta nel package application che fornisce metodi e strutture dati per la memorizzazione e gestione degli stati della lavagna.

3.4.2 Inserimento del Testo

Il bottone che consente di inserire le stringhe era presente anche nel prototipo precedente, tuttavia il suo utilizzo era complicato. L'utente doveva cliccare sul bottone per abilitare l'opzione scrittura testo. In seguito con un click del mouse doveva abilitare l'area di testo in cui avrebbe poi scritto utilizzando la tastiera. Il testo veniva poi trasferito sulla lavagna nel momento in cui l'utente avrebbe cliccato con il tasto destro nella zona in cui voleva scrivere. Questo procedimento è lungo e complesso e il suo utilizzo non è molto intuitivo.

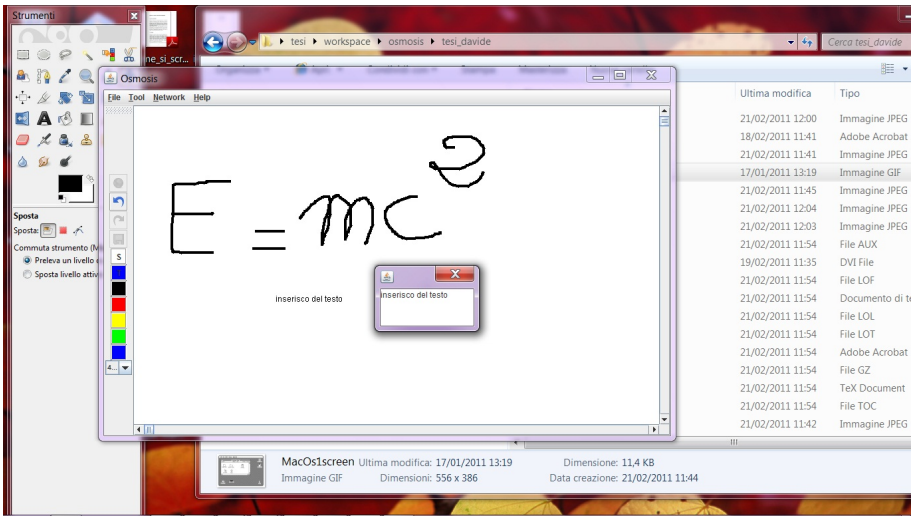

Figura 3.4: Inserimento del testo

E' da notare inoltre come l'utilizzo dell'inserimento del testo si differenzia dall'inserimento delle linee (per cui basta un click) rendendo l'interfaccia inconsistente. Come fare allora per ottenere un sistema efficiente che permetta l'inserimento di stringhe? Si era pensato di rendere questa funzione sempre disponibile premendo il tasto destro del mouse, tuttavia tale sistema risultava estremamente efficiente utilizzando il mouse, mentre avrebbe creato problemi con una penna tablet nella quale potrebbe non essere presente il tasto destro.

Perciò è stato deciso di adottare un pulsante per attivare la funzione inserimento stringa e di eliminare la textarea sottostante la lavagna. Una volta che l'utente ha scelto dove vuole inserire del testo appare una finestra di dialogo che gli consente di scrivere. Questo comportamento è del tutto simile a quello del famoso editor grafico Gimp. Quando viene premuto il bottone viene adottato un cursore di testo comunemente utilizzato nei word processor, questo incoraggia l'utente a cliccare per poi scrivere come farebbe editando un qualsiasi altro documento. Infine premendo il tasto Esc il testo viene trasferito sulla lavagna.

3.4.3 Cancellazione Linee e Testo

Per questa funzione già esistente si è adottato un cursore speciale a forma di "mirino". L'utente può selezionare (cliccando) una linea o una parte del testo e

rimuoverla premendo il tasto Canc.

3.5 Conclusioni di Usabilità

Come abbiamo visto nel paragrafo 3.3 la **sicurezza d'uso** è stata realizzata fornendo, a seconda dei casi, opportune finestre di dialogo che veicolano l'utente nella scelta delle operazioni più appropiate alle sue esigenze fornendo anche dei meccanismi per rendere reversibile qualsiasi scelta. In queste circostanze l'utente si trova protetto da possibili errori logici o involontari e può concentrarsi sul disegno, utilizzando le funzionalità del menù in maniera **efficace** ed **efficiente** e quindi incrementando la sua produttività.

L'uso del menu file in un software che lavora con dei documenti è estremamente utile ed è ormai ampiamente diffuso nei programmi comuni come ad esempio nei browser, nei client di posta e in tutti i sistemi operativi basati su interfaccie WIMP[2]. Possiamo quindi affermare che il suo utilizzo e la maniera in cui questo avviene sia un **vincolo culturale**, discusso nel capitolo riguardante i principi di progettazione di usabilità.

Sono stati aggiunti anche importanti **vincoli logici** che aiutano l'utente a comprendere l'utilizzo del menù. In questo senso le voci delle operazioni che, se attivate, porterebbero a degli errori (come abbiamo visto in dettaglio analizzando undo/redo) o che risulterebbero inutili vengono disattivate. Per esempio la voce Save è disattivata anche quando ho appena aperto un file, sarebbe infatti inutile il suo utilizzo e nel caso volessi salvare lo stesso file con un altro nome dovrei ricorrere a Save as. In questo caso la disattivazione di Save serve proprio a suggerire l'utilizzo di Save As.

Un principio di progettazione molto importante è quello di **fornire feedback**, cioè ogni strumento dell' interfaccia deve fornire informazioni sul suo stato. Pensiamo al caso del salvataggio. Cosa accade se tento di salvare un file? Nel prototipo precedente non avevo nessuna informazione di ritorno, cioè osmosis non mi informava che questa operazione era avvenuta con successo (o che stava avvenendo). In oltre se il file era di grosse dimensioni il cursore veniva bloccato per tutto il tempo necessario al salvataggio.

[2]Window, Icon, Menù e Pointing Device

Per ovviare a questi inconvenienti si è scelto di utilizzare diversi tipi di cursori. Per il salvataggio è stato adottato un particolare cursore del mouse a forma di clessidra che indica all'utente l'avanzamento dell'operazione. Una volta che questa è stata portata a termine il cursore torna quello di default e la voce del menu Save viene disattivata suggerendo che il salvataggio è avvenuto con successo. I metodi per manipolare i cursori sono forniti dalla classe *DrawPanel*.

Avviando Osmosis è subito possibile cominciare a disegnare, per rendere ciò evidente è stato creato un cursore apposito che al passaggio del mouse sulla lavagna visualizza una matita, differenziandosi così dalla freccetta che appare quando si passa sulla barra dei menù. Nel caso vengano attivate le altre funzioni di inserimento o cancellazione il cursore cambia ancora. Questi accorgimenti sono utili perchè coinvolgono l'utente nell'utilizzo dell'interfaccia, inoltre consigliano il modo in cui tali funzioni debbano essere utilizzate sulla lavagna fornendo delle informazioni di ritorno. Il cursore matita invita a disegnare, così come il cursore mirino (utilizzato nella cancellazione di linee) invita a selezionare la linea con precisione.

Capitolo 4

Il Codice

In questo capitolo verranno presentate le principali modifiche effettuate nel codice di Osmosis. Vedremo innanzi tutto l'organizzazione dei package e successivamente verranno presentate le nuove classi realizzate e le modifiche più importanti che implementano le funzionalità della gui che è stata sviluppata utilizzando il framework Swing[13].

4.1 Organizzazione dei Package

Il codice è suddiviso in package, ognuno di essi rappresenta una parte logica di Osmosis:

- GUI: contiene il codice relativo all'interfaccia grafica;
- APPLICATION: main class e funzionalità aggiuntive;
- EVENT: gestione degli eventi;
- ACTION: gestione delle azioni;
- NET: funzionalità di rete;
- PARSER: il parser di osmosis;
- THREADS: codice di definizione dei threads di osmsois;

Vediamo principalmente i primi quattro package che contengono le classi in cui è implementata la nuova interfaccia.

4.2 Il Package Gui

Le due classi principali di questo package sono*DrawPanel e MainWindow*, vedremo anche la classe *SafeOverwiteFileChooser* creata per gestire le finestre di dialogo.

4.2.1 La Classe DrawPanel

La classe *DrawPanel* contiene la definizione della lavagna, sono stati aggiunti i metodi per la creazione (*createMyCursor()*) e gestione dei cursori (metodi *setDefCursor()*, *setTextCursor()* , *setWaitCursor()* e per il caricamento di una lavagna (metodo *setBoard(DrawDescription board)*. Il metodo *paintComponent(Graphics g)* è stato modificato.

```
1  public class DrawPanel extends JPanel {
2  private Cursor defCursor = createMyCursor();
3  [..]
4    private Cursor textCursor = new Cursor(Cursor.TEXT_CURSOR); //cursore per stringhe
5    private Cursor cutCursor = new Cursor(Cursor.MOVE_CURSOR);
6    private Cursor waitCursor = new Cursor(Cursor.WAIT_CURSOR);
7
8    public DrawPanel(){
9      super();
10     /**/
11     this.setPreferredSize(new Dimension(60000,40000));
12     /**/
13     this.setBackground(Color.WHITE);
14     board = new DrawDescription();
15     this.unlock();
16     this.setCursor(defCursor);
17   }
18   /**
19    * crea il cursore penna utilizzato di default nel pannello
20    *  */
21
22   public Cursor createMyCursor(){
23     Toolkit tool = Toolkit.getDefaultToolkit();
24     ImageIcon tmp = new ImageIcon(getClass().getResource("pencil.gif"));
25     Image img = tmp.getImage();
26     Cursor penCursor = tool.createCustomCursor(img,new Point(0,0) , "penCursor");
27     return penCursor;
28   }
29
30   public void setDefCursor(){
31
32     this.setCursor(defCursor);
33   }
34   public void setTextCursor(){
35     this.setCursor(textCursor);
36   }
37   public void setCutCursor(){
38     this.setCursor(cutCursor);
39   }
40   public void setWaitCursor(){
41     this.setCursor(waitCursor);
42   }
43 [..]
```

```
44  /*
45     * Permette di caricare un disegno sulla lavagna
46     * @param board disegno da caricare
47     */
48    public void setBoard(DrawDescription board) {
49      //this.board.clear();
50      this.board = new DrawDescription(board.getVectorPoint(), board.getVectorStrings());
51    }
52    /** paint
53    */
54    public void paintComponent(Graphics g){
55      super.paintComponent(g);
56      Graphics2D g2d = (Graphics2D) g;
57      /*draw points*/
58      for (int i=0; i<board.pointN()-1; i++){
59
60        Point p1 = board.getPoint(i).getPoint();
61        Point p2 = board.getPoint(i+1).getPoint();
62
63        if (p1.getX() > -1 && p2.getX() > -1){
64          if (board.isSelectedInitial(i)){
65            g2d.setColor(Color.MAGENTA);
66          }
67          else{
68            g2d.setColor(board.getPointColorInitial(i));
69          }
70          g2d.setStroke(board.getPointStrokeInitial(i));
71          g2d.drawLine((int)p1.getX(), (int)p1.getY(), (int)p2.getX(), (int)p2.getY());
72        }
73      }
74      /*draw strings*/
75      for (int i=0; i<board.stringN(); i++){
76        String text = board.getString(i).getString();
77        Point p = board.getString(i).getLocation();
78        if (board.getString(i).isSelected()){
79          g2d.setColor(Color.MAGENTA);
80        }
81        else{
82          g2d.setColor(board.getString(i).getColor());
83        }
84        g2d.drawString(text, (int)p.getX(), (int)p.getY());
85      }
86    }
87  [..]
```

Listato 4.1: Alcuni frammenti della classe *DrawPanel*

4.2.2 La Classe MainWindow

La classe principale del package GUI in cui viene inizializzata l'interfaccia. Il codice delle operazioni di salvataggio, usato dalle voci di menù e dai bottoni e stato raggruppato in due particolari metodi *routineSave()* e *routineLoad()*. Questi metodi sono visibili anche all'esterno del package gui perchè vengono utilizzati in particolari occasioni anche dalle classi degli eventi. La routine di salvataggio funziona in questo modo:

- viene settato il cursore clessidra (linea 5 del listato)

- gli oggetti dei vettori stringa e punto vengono tradotti dal parser e inseriti in una stringa (linee da 6 a 15)
- la stringa viene scritta su file (linea 16)
- vengono settate delle variabili interne (linee 17 e 18)
- viene settato il cursore matita (linea 19)

```
1  /** gestisce le operazioni di salvataggio e viene invocata anche dai listener
2   *
3   */
4  public void routineSave(File file){
5    MainWindow.getMainWindow().getBoardPanel().setWaitCursor();
6    DrawDescription dd = MainWindow.getMainWindow().getBoardPanel().getBoard();
7    String toSave="";
8    for (int i=0; i<dd.pointN(); i++){
9      toSave=toSave+newSParser.requestParse(Cmd.add, dd.getPoint(i));
10     toSave = toSave+";";
11   }
12   for (int i=0; i<dd.stringN(); i++){
13     toSave=toSave+newSParser.requestParse(Cmd.add, dd.getString(i));
14     toSave = toSave+";";
15   }
16   SaveAndLoad.save(toSave,file.getAbsolutePath());
17   currentFile = file;
18   lenghtSave=dd.pointN()+dd.stringN(); //elementi memorizzati nei vettori
19   MainWindow.getMainWindow().getBoardPanel().setDefCursor();
20 }
```

Listato 4.2: Il metodo *routineSave()* della classe *MainWindow*

La routine di caricamento invece segue il seguente algoritmo:

- viene settato il cursore clessidra (linea 5)
- viene settata a variabile file (linea 6)
- i vettori vengono svuotati e riempiti con gli oggetti contenuti nei vettori dell nuovo disegno (linee 7 e 8)
- la lavagna viene ri-disegnata (linea 9)
- vengono settate variabili interne (linee 10 e 11
- viene settato il cursore matita (linea 12)

```
1  /** gestisce le operazioni di caricamento e viene invocata anche dai listener
2   *
3   */
4  public void routineLoad(File file){
5    MainWindow.getMainWindow().getBoardPanel().setWaitCursor();
```

```
6       currentFile = file;
7       boardPanel.getBoard().clear();
8       newSParser.requestParse(SaveAndLoad.load(file.getAbsolutePath()));
9       boardPanel.redraw();
10      DrawDescription dd2 = MainWindow.getMainWindow().getBoardPanel().getBoard();
11      lenghtSave=dd2.pointN()+dd2.stringN();
12      MainWindow.getMainWindow().getBoardPanel().setDefCursor();
13    }
```

Listato 4.3: Il metodo *routineLoad()* della classe *MainWindow*

Vediamo ora il codice che crea e posiziona i componenti dell'interfaccia. I menù vengono inseriti nella topBar (metodo *getTopBar()*) attraverso il metodo *add* nelle linee da 36 a 39 del listato 4.4. Ogni menù viene creato da un opportuno metodo, vediamo in dettaglio il menù File creato dal metodo *getFileMenu()*. Alla riga 52 viene settato il testo della voce (metodo *setText(File*, alla riga 53 viene settata la combinazione di tasti (metodo *setMnemonic()*) e successivamente vengono aggiunte tutte le sue voci (linee da 54 a 59).

I metodi successivi creano le voci del menù file, vediamo alcuni particolari della voci New creata dal metodo *getNewItem*. La scorciatoia da tastiere viene impostata alla linea 100 (metodo *setAccelerator()*) e la descrizione della voce alla linea 103 (metodo *setToolTipsText()*). Successivamente viene aggiunto l'actionlistener che consente di associare un azione all'invocazione della voce (metodo *addActionListener(new NewAction())*). Gli oggetti ActionListener, come ad esempio newAction, li vedremo in seguito perchè contenuti nel package action.

```
1   public class MainWindow extends JFrame {
2   [..]
3     /**
4      * This is the default constructor
5      */
6     public MainWindow() {
7       super();
8       initialize();
9     }
10
11    /**
12     * This method initializes mainwindow
13     *
14     * @return void
15     */
16    private void initialize() {
17      this.setName("mainWindow");
18      this.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
19      this.addWindowListener(new MainWindowListener());
20      this.setContentPane(getJContentPane());
21      this.setSize(600,400);
22      this.setTitle("Osmosis␣");
23      this.getBlackDraw().doClick();
24      this.setVisible(true);
25    }
26
27    /**
```

```
28      * This method initializes topBar
29      *
30      * @return javax.swing.JMenuBar
31      */
32     private JMenuBar getTopBar() {
33       if (topBar == null) {
34         topBar = new JMenuBar();
35         topBar.add(getFileMenu());
36         topBar.add(getEditMenu());
37         topBar.add(getNetworkMenu());
38         topBar.add(getHelpMenu());
39       }
40       return topBar;
41     }
42
43     /**
44      * This method initializes fileMenu
45      *
46      * @return javax.swing.JMenu
47      */
48     private JMenu getFileMenu() {
49       if (fileMenu == null) {
50         fileMenu = new JMenu();
51         fileMenu.setText("File");
52         fileMenu.setMnemonic(KeyEvent.VK_F);
53         fileMenu.add(getNewItem());
54         fileMenu.add(getOpenItem());
55         fileMenu.add(getSaveItem());
56         fileMenu.add(getSaveAsItem());
57         fileMenu.add(getExportMenu());
58         fileMenu.add(getExitItem());
59       }
60       return fileMenu;
61     }
62
63     /**
64      * This method initializes editMenu
65      *
66      * @return javax.swing.JMenu
67      */
68     private JMenu getEditMenu() {
69       if (editMenu == null) {
70         editMenu = new JMenu();
71         editMenu.setText("Tool");
72         editMenu.setMnemonic(KeyEvent.VK_T);
73         editMenu.add(getUndoItem());
74         editMenu.add(getRedoItem());
75         editMenu.add(getCancItem());
76         editMenu.add(getInsertStringItem());
77       }
78       return editMenu;
79     }
80     private JMenu getHelpMenu() {
81       if(helpMenu == null) {
82         helpMenu = new JMenu();
83         helpMenu.setText("Help");
84         helpMenu.setMnemonic(KeyEvent.VK_H);
85         helpMenu.add(getAboutItem());
86       }
87       return helpMenu;
88     }
89
90     /**
91      * This method initializes newItem
92      *
93      * @return javax.swing.JMenuItem
94      */
```

```
95    private JMenuItem getNewItem() {
96      if (newItem == null) {
97        newItem = new JMenuItem();
98        newItem.setText("New");
99        newItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N,
100           Event.CTRL_MASK, true));
101       newItem.setMnemonic('N');
102       newItem.setToolTipText("New␣empty␣file");
103       newItem.addActionListener(new NewAction());
104     }
105     return newItem;
106   }
107
108   /**carica un file .osm sulla board.
109    *
110    */
111   private JMenuItem getOpenItem(){
112     if (openItem == null) {
113       openItem = new JMenuItem();
114       openItem.setText("Open");
115       openItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O,
116           Event.CTRL_MASK, true));
117       openItem.setMnemonic('O');
118       openItem.setToolTipText("Open␣file");
119       openItem.addActionListener(new OpenAction() );
120     }
121     return openItem;
122   }
123
124   /**
125    * This method initializes saveItem
126    *
127    * @return javax.swing.JMenuItem
128    */
129   private JMenuItem getSaveItem() {
130     if (saveItem == null) {
131       saveItem = new JMenuItem("Save");
132       saveItem.setText("Save");
133       saveItem.setMnemonic('S');
134       saveItem.setEnabled(false);
135       saveItem.setToolTipText("Save␣file");
136       saveItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S, Event.CTRL_MASK, true));
137       saveItem.addActionListener(new SaveAction());
138     }
139     return saveItem;
140   }
141
142   /**
143    * This method initializes saveAsItem
144    *
145    * @return javax.swing.JMenuItem
146    */
147   private JMenuItem getSaveAsItem() {
148     if (saveAsItem == null) {
149       saveAsItem = new JMenuItem();
150       saveAsItem.setText("Save␣as..");
151       saveAsItem.setMnemonic('A');
152       saveAsItem.setEnabled(false);
153       saveAsItem.setToolTipText("Save␣as␣file");
154       saveAsItem.addActionListener(new java.awt.event.ActionListener() {
155         public void actionPerformed(java.awt.event.ActionEvent e) {
156           fcs.setDialogTitle("Save␣as..");
157           fcs.setFileFilter(new OsmFileFilter());
158           int ret = fcs.showSaveDialog(jContentPane);
159           if (ret == JFileChooser.APPROVE_OPTION){
160             routineSave(fcs.getSelectedFile());
161             MainWindow.getMainWindow().setEnableSave(true);
```

```
162             //MainWindow.getMainWindow().getOsmosisState().resetVectorState();
163           }
164         }
165       });
166     }
167     return saveAsItem;
168   }
169   /**
170    * This method initializes exportMenu
171    *
172    * @return javax.swing.JMenu
173    */
174   private JMenu getExportMenu() {
175     if (exportMenu == null) {
176       exportMenu = new JMenu();
177       exportMenu.setText("Export to ...");
178       exportMenu.add(getImageExportItem());
179     }
180     return exportMenu;
181   }
182
183   /**
184    * This method initializes exitItem
185    *
186    * @return javax.swing.JMenuItem
187    */
188   private JMenuItem getExitItem() {
189     if (exitItem == null) {
190       exitItem = new JMenuItem();
191       exitItem.setText("Exit");
192       exitItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_E,
193           Event.CTRL_MASK, true));
194       exitItem.setMnemonic('E');
195       exitItem.setToolTipText("Exit Osmosis.Bye");
196       exitItem.addActionListener(new ExitAction());
197     }
198     return exitItem;
199   }
200   /**
201    * This method initializes undoItem
202    *
203    * @return javax.swing.JMenuItem
204    */
205   private JMenuItem getUndoItem() {
206     if (undoItem == null) {
207       undoItem = new JMenuItem();
208       undoItem.setText("Undo");
209       undoItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_Z,
210           Event.CTRL_MASK, true));
211       undoItem.getAccessibleContext().setAccessibleDescription("Cancel last action");
212       undoItem.setToolTipText("Cancel last action");
213       //undoItem.setMnemonic('U');
214       undoItem.setEnabled(false); //si attiva dopo la prima qualche modifica
215       undoItem.addActionListener(new UndoAction());
216     }
217     return undoItem;
218   }
219   /**
220    * This method initializes redoItem
221    *
222    * @return javax.swing.JMenuItem
223    */
224   private JMenuItem getRedoItem() {
225     if (redoItem == null) {
226       redoItem = new JMenuItem();
227       redoItem.setText("Redo");
228       redoItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_Y,
```

```
229             Event.CTRL_MASK, true));
230         redoItem.setToolTipText("Repeat␣last␣action");
231         redoItem.getAccessibleContext().setAccessibleDescription("Repeat␣last␣action");
232         redoItem.setEnabled(false); //redo disponibile solo dopo undo
233         redoItem.addActionListener(new RedoAction());
234       }
235       return redoItem;
236     }
237
238     /**
239      * This method initializes cancItem
240      *
241      * @return javax.swing.JMenuItem
242      */
243     private JMenuItem getCancItem() {
244       if (cancItem == null) {
245         cancItem = new JMenuItem();
246         cancItem.setText("Canc␣line");
247         cancItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_C,
248             Event.CTRL_MASK, true));
249         cancItem.setToolTipText("Select␣and␣canc␣line");
250         cancItem.getAccessibleContext().setAccessibleDescription("Select␣and␣canc␣line");
251         //cancItem.setEnabled(false);
252         cancItem.addActionListener(new CancAction());
253
254       }
255       return cancItem;
256     }
257       /**
258        * This method initializes insert Item
259        *
260        * @return javax.swing.JMenuItem
261        */
262     private JMenuItem getInsertStringItem() {
263       if (insertStringItem == null) {
264         insertStringItem = new JMenuItem();
265         insertStringItem.setText("Write␣Text");
266         insertStringItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_T,
267             Event.CTRL_MASK, true));
268         insertStringItem.setToolTipText("Write␣Text␣on␣board");
269         insertStringItem.getAccessibleContext().setAccessibleDescription("Write␣Text␣on␣board");
270         insertStringItem.addActionListener(new  TextAction());
271       }
272       return insertStringItem;
273     }
274       /**
275      * This method initializes imageExportItem
276      *
277      * @return javax.swing.JMenuItem
278      */
279     private JMenuItem getImageExportItem() {
280       if (imageExportItem == null) {
281         imageExportItem = new JMenuItem();
282         imageExportItem.setText("Image␣JPEG");
283         imageExportItem.setToolTipText("Export␣like␣as␣JPEG␣file");
284         imageExportItem.setEnabled(false);
285         imageExportItem.addActionListener(new java.awt.event.ActionListener() {
286           public void actionPerformed(java.awt.event.ActionEvent e) {
287             fcs.setFileFilter(new JpegFileFilter());
288             int ret = fcs.showDialog(jContentPane, "Export␣");
289             File file = null;
290             Image img = new BufferedImage((int)boardPanel.getScrollPane().getViewport().getVisibleRect().
                  getWidth(),(int)boardPanel.getScrollPane().getViewport().getVisibleRect().getHeight(),
                  BufferedImage.TYPE_INT_RGB);
291             Graphics g = img.getGraphics();
292
293             boardPanel.getScrollPane().getViewport().paint(g);
```

```
294
295           if (ret == JFileChooser.APPROVE_OPTION){
296
297             if( !(fcs.getSelectedFile().getName().endsWith(".jpeg") || fcs.getSelectedFile().getName().
                    endsWith(".JPEG"))){
298               file = new File(fcs.getSelectedFile().getAbsolutePath()+".jpeg");
299             }
300             else{
301               file = fcs.getSelectedFile();
302             }
303             ImageImporterExporter.exportImage((BufferedImage)img, file);
304           }
305         }
306       });
307     }
308     return imageExportItem;
309   }
310   /**
311    * This method initializes networkMenu
312    *
313    * @return javax.swing.JMenu
314    */
315   public JMenu getNetworkMenu() {
316     if (networkMenu == null) {
317       networkMenu = new JMenu();
318       networkMenu.setText("Network");
319       networkMenu.setMnemonic(KeyEvent.VK_N);
320       networkMenu.add(getActivateCheckBox());
321     }
322     return networkMenu;
323   }
324
325 [..]
```

Listato 4.4: Alcuni frammenti della classe *MainWindow* che realizzano la barra dei menù

Nel frammento di listato seguente vedremo la creazione della barra laterale (metodo *getSideBar()*). Al suo interno vengono posizionati tutti i bottoni e tutte le icone. In particolare osserviamo alla linea 28 il metodo *getLineWidthsBox()* che crea il bottone che consente di cambiare spessore alla linea e i metodi *getUndoButton()* alla linea 208 e *getRedoButton()* alla linea 219 che creano i bottoni per le funzionalità undo e redo posizionati nella barra laterale.

```
1   /**
2    * This method initializes jContentPane
3    *
4    * @return javax.swing.JPanel
5    */
6   private JPanel getJContentPane() {
7     if (jContentPane == null) {
8       jContentPane = new JPanel();
9       jContentPane.setLayout(new BorderLayout());
10      jContentPane.setName("");
11      jContentPane.add(getTopBar(),BorderLayout.NORTH);
12      //jContentPane.add(getBottomBar(), BorderLayout.SOUTH);
13      jContentPane.add(getSideBar(), BorderLayout.WEST);
14      //jContentPane.add(getBoardPanel(), BorderLayout.CENTER);
15      /**/
16      jContentPane.add(getBoardPanel().getScrollPane(), BorderLayout.CENTER);
```

```
17
18        /**/
19      }
20      return jContentPane;
21    }
22
23    /**
24     * This method initializes lineWidthsBox
25     *
26     * @return javax.swing.JComboBox
27     */
28    private JComboBox getLineWidthsBox() {
29      if (lineWidthsBox == null) {
30        lineWidthsBox = new JComboBox();
31        lineWidthsBox.setFont(new Font("Dialog", Font.BOLD, 10));
32        lineWidthsBox.setSelectedIndex(-1);
33        lineWidthsBox.setToolTipText("Select␣line␣width");
34        lineWidthsBox.setPreferredSize(new Dimension(40, 22));
35        lineWidthsBox.setMaximumRowCount(10);
36        lineWidthsBox.addActionListener(new java.awt.event.ActionListener() {
37          public void actionPerformed(java.awt.event.ActionEvent e) {
38            drawStroke = new BasicStroke (lineWidthsBox.getSelectedIndex()+1);
39          }
40        });
41        //populate jcombobox
42        for (int i=1; i<11; i++){
43          lineWidthsBox.addItem(i+"␣pt");
44        }
45      }
46      return lineWidthsBox;
47    }
48    /**
49     * This method initializes jDialog
50     *
51     * @return javax.swing.JDialog
52     */
53    public JDialog getTextInsertDialog() {
54      if (textInsertDialog  == null) {
55        textInsertDialog = new JDialog(this);
56        textInsertDialog.setSize(new Dimension(100, 100));
57         textInsertDialog.setContentPane(MainWindow.getMainWindow().getJContentPane2());
58        //textInsertDialog.setUndecorated(true);
59               //textInsertDialog.setBackground(Color.BLUE) ;
60      }
61      else
62        textInsertDialog.setLocation(50,50);
63      return textInsertDialog;
64    }
65    private JPanel getJContentPane2() {
66      if (jContentPane2 == null) {
67        jContentPane2 = new JPanel();
68        jContentPane2.setLayout(new BorderLayout());
69        jContentPane2.setName("");
70        jContentPane2.add(getTextArea(), BorderLayout.CENTER);
71      }
72      return jContentPane2;
73    }
74
75    public void setTextInsertDialog(JDialog text){
76      this.textInsertDialog = text;
77    }
78
79    /**
80     * This method initializes textArea
81     *
82     * @return javax.swing.JTextArea
83     */
```

```
84    public JTextArea getTextArea() {
85      if (jTextArea == null) {
86        jTextArea = new JTextArea();
87        jTextArea.setLineWrap(true);
88        jTextArea.addKeyListener(getMyKeyAdapter());
89
90      }
91      return jTextArea;
92    }
93
94    /**
95     * This method initializes lockToggleButton status connessione
96     *
97     * @return javax.swing.JToggleButton
98     */
99    public JToggleButton getLockToggleButton() {
100     if (lockToggleButton == null) {
101       lockToggleButton = new JToggleButton();
102       lockToggleButton.setToolTipText("not␣Connected");
103       lockToggleButton.setIcon(new ImageIcon(getClass().getResource("Globe-icon16px.png")));
104       lockToggleButton.setEnabled(false);
105         lockToggleButton.addActionListener(new java.awt.event.ActionListener() {
106         public void actionPerformed(java.awt.event.ActionEvent e) {
107         //  voglio scrivere sulla lavagna che e disponibile
108           if (lockToggleButton.getToolTipText() == "available"){
109             if (OsmosisBoard.ap2ap_is_connected){
110               Ap2Ap.create().sendDatagram("lock");
111               lockToggleButton.setEnabled(true);
112             }
113             boardPanel.unlock();
114             lockToggleButton.setToolTipText("Connected:␣drawing");
115           }
116           else if (lockToggleButton.getToolTipText() == "drawing"){
117           //  ho finito di disegnare e rendo disponibile la lavagna
118             if (OsmosisBoard.ap2ap_is_connected){
119               Ap2Ap.create().sendDatagram("unlock");
120             }
121             boardPanel.lock();
122             lockToggleButton.setToolTipText("Connected:␣available");
123             lockToggleButton.setEnabled(true);
124           }
125           //non connesso ad ap2ap, lavagna sbloccata per disegni in locale
126           else if (lockToggleButton.getToolTipText() == "not␣Connected"){
127             lockToggleButton.setEnabled(false);
128             boardPanel.unlock();
129           }
130           else{}  //DO NOTHING
131         }
132       });
133     }
134     return lockToggleButton;
135   }
136
137   /**
138    * This method initializes sideBar
139    *
140    * @return javax.swing.JToolBar
141    */
142   private JToolBar getSideBar() {
143     if (sideBar == null) {
144       GridBagLayout gbl = new GridBagLayout();
145       GridBagConstraints gbc = new GridBagConstraints();
146       sideBar = new JToolBar();
147       sideBar.setOrientation( JToolBar.VERTICAL); //JToolBar.VERTICAL
148       sideBar.setLayout(gbl);
149       /*set costraint for component and add to sidebar*/
150       gbc.gridx = 0;
```

```
151
152         gbc.gridy=0;
153         gbl.setConstraints(getLockToggleButton(), gbc);
154         sideBar.add(getLockToggleButton());
155
156         gbc.gridy++;
157         gbl.setConstraints(getUndoButton(), gbc);
158         sideBar.add(getUndoButton());
159
160         gbc.gridy++;
161         gbl.setConstraints(getRedoButton(), gbc);
162         sideBar.add(getRedoButton());
163
164         gbc.gridy++;
165         gbl.setConstraints(getSaveButton(), gbc);
166         sideBar.add(getSaveButton());
167
168         gbc.gridy++;
169         gbl.setConstraints(getSelectToggleButton(), gbc);
170         sideBar.add(getSelectToggleButton());
171
172         gbc.gridy++;
173         gbl.setConstraints(getInsertToggleButton(), gbc);
174         sideBar.add(getInsertToggleButton());
175
176         gbc.gridy++;
177         gbl.setConstraints(getBlackDraw(), gbc);
178         sideBar.add(getBlackDraw());
179
180         gbc.gridy++;
181         gbl.setConstraints(getRedDraw(), gbc);
182         sideBar.add(getRedDraw());
183
184         gbc.gridy++;
185         gbl.setConstraints(getYellowDraw(), gbc);
186         sideBar.add(getYellowDraw());
187
188         gbc.gridy++;
189         gbl.setConstraints(getGreenDraw(), gbc);
190         sideBar.add(getGreenDraw());
191
192         gbc.gridy++;
193         gbl.setConstraints(getBlueDraw(), gbc);
194         sideBar.add(getBlueDraw());
195
196         gbc.gridy++;
197         gbl.setConstraints(getLineWidthsBox(), gbc);
198         sideBar.add(getLineWidthsBox());
199       }
200       return sideBar;
201     }
202
203     /**
204      * This method initializes undoButton
205      *
206      * @return javax.swing.JMenuItem
207      */
208     public JButton getUndoButton() {
209       if (undoButton == null) {
210         undoButton = new JButton("");
211         undoButton.setIcon(new ImageIcon(getClass().getResource("Undo-icon16px.png")));
212         undoButton.setToolTipText("undo");
213         undoButton.setEnabled(false);
214         undoButton.addActionListener(new UndoAction());
215       }
216       return undoButton;
217     }
```

```
218
219     public JButton getRedoButton(){
220       if (redoButton == null) {
221         redoButton = new JButton("");
222         redoButton.setIcon(new ImageIcon(getClass().getResource("Redo-icon.png")));
223         redoButton.setEnabled(false);
224         redoButton.setToolTipText("Repeat␣last␣action");
225         redoButton.addActionListener(new RedoAction());
226       }
227       return redoButton;
228     }
229     /**
230      * This method initializes SelectToggleButton
231      *  seleziona una linea che viene essere rimossa con canc
232      * @return javax.swing.JToggleButton
233      */
234     public JButton getSelectToggleButton() {
235       if (SelectToggleButton == null) {
236         SelectToggleButton = new JButton();
237         SelectToggleButton.setBackground(Color.white);
238         SelectToggleButton.setToolTipText("Select/canc␣line");
239         SelectToggleButton.setText("S");
240         //SelectToggleButton.setIcon(.getClass().)
241         SelectToggleButton.setPreferredSize(new Dimension(24, 24));
242         SelectToggleButton.addActionListener(new CancAction());
243       }
244       return SelectToggleButton;
245     }
246
247     /**
248      * This method initializes insertToggleButton
249      *
250      * @return javax.swing.JToggleButton
251      */
252     public JButton getInsertToggleButton() {
253       if (insertToggleButton == null) {
254         insertToggleButton = new JButton();
255         insertToggleButton.setBackground(Color.white);
256         insertToggleButton.setToolTipText("Insert␣Text␣and␣pres␣ESC");
257         //icona macchina da scrivere o tastiera
258         //insertToggleButton.setIcon(defaultIcon);
259         insertToggleButton.setText("T");
260         insertToggleButton.setPreferredSize(new Dimension(24, 24));
261         insertToggleButton.addActionListener(new TextAction());
262       }
263       return insertToggleButton;
264     }
265     /**
266      * This method initializes saveButton
267      *
268      * @return javax.swing.JMenuItem
269      */
270     private JButton getSaveButton() {
271       if (saveButton == null) {
272         saveButton = new JButton("");
273         saveButton.setIcon(new ImageIcon(getClass().getResource("Save-icon16px.png")));
274         saveButton.setToolTipText("Save␣file");
275         saveButton.setEnabled(false);
276         saveButton.addActionListener(new SaveAction());
277       }
278       return saveButton;
279     }
280
281     /**
282      * This method initializes colorSelected
283      *
284      * @return javax.swing.JButton
```

```
285     */
286   public JButton getColorSelected() {
287     if (colorSelected == null) {
288       colorSelected = new JButton();
289       colorSelected.setBackground(Color.black);
290       colorSelected.setForeground(new Color(238, 238, 238));
291       colorSelected.setPreferredSize(new Dimension(24, 24));
292       colorSelected.setToolTipText("");
293       colorSelected.setMnemonic(KeyEvent.VK_UNDEFINED);
294       colorSelected.setEnabled(false);
295     }
296     return colorSelected;
297   }
298
299   public void setColorSelected(JButton colorSelected) {
300     this.colorSelected = colorSelected;
301   }
302 [..]
```

Listato 4.5: Alcuni frammenti della classe *MainWindow* in cui viene creata la barra laterale

Nel frammento seguente osserviamo il metodo *getBoardPanel()* che crea la lavagna. Alla linea 10 viene effettuato un importante controllo che permette se necessario di aggiungere uno stato alla struttura dati utilizzata da undo e redo (metodo *add.State(boardPanel.board)*). Alla linea 30 viene inizializzata l'interfaccia (metodo *initGui()*) impostanto l'aspetto grafico con il look-and-feel[1] nativo del sistema operativo (linea 33). Si osservi alla linea 50 che il comportamento del gestore finestra alla pressione dell'icona X viene impostato come nullo (*DO_NOTHING_ON_CLOSE*), esso verrà ridefinito nella classe *MainWindowListener* in modo da gestire anche tutti i casi eccezzionali.

```
1  /**
2     * This method initializes boardPanel
3     *
4     * @return javax.swing.JPanel
5     */
6   public DrawPanel getBoardPanel() {
7     if (boardPanel == null) {
8       boardPanel = new DrawPanel();
9     }
10    if(getOsmosisState().stateSize()  == 0){
11      getOsmosisState().addState(boardPanel.board); //la prima board memorizzata: quella vuota
12    }
13    return boardPanel;
14
15  }
16  public UndoRedoDescription getOsmosisState(){
17    if (vettoreStati == null){
18      vettoreStati = new UndoRedoDescription();
19    }
20    return vettoreStati;
21  }
```

[1]Look (come un oggetto appare) and Feel (come un oggetto grafico deve reagire alle azioni dell'utente)

```
22
23    public void setBoardPanel(DrawPanel boardPanel) {
24
25      this.boardPanel = boardPanel;
26    }
27    /**metodo initGui
28     * @param args
29     */
30    public static void initGui() {
31      try {
32        // Set cross-platform Java L&F (also called "Metal")
33        UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
34      }
35      catch (UnsupportedLookAndFeelException e) {
36        // handle exception
37      }
38      catch (ClassNotFoundException e) {
39        // handle exception
40      }
41      catch (InstantiationException e) {
42        // handle exception
43      }
44      catch (IllegalAccessException e) {
45        // handle exception
46      }
47      SwingUtilities.invokeLater(new Runnable() {
48        public void run() {
49          thisClass = new MainWindow();
50          thisClass.setDefaultCloseOperation(JFrame.DO\_NOTHING\_ON\_CLOSE);
51        }
52      });
53    }
```

Listato 4.6: Frammenti di *MainWindow* in cui avviene l'inizializzazione dell'interfaccia

Nel frammento seguente vediamo i metodi utilizzati per attivare e disattivare i bottoni e le icone.

```
1   [..]
2     public void setSelectCanc(boolean b) {
3       this.selectCanc = b;
4     }
5
6     public void setSelectInsert(boolean b) {
7       this.selectInsert = b;
8     }
9
10    public void setSelectToggleButton(JButton selectToggleButton) {
11      SelectToggleButton = selectToggleButton;
12    }
13
14  [..]
15    /*setta la visibilita della voce save nel menu file del bottone.
16     * invocato da mouseController
17     */
18    public void setEnableSave(boolean f){
19      saveItem.setEnabled(f);
20      saveButton.setEnabled(f);
21    }
22    /*setta la visibilita della voce save nel menu file.
23     * invocato da mouseController
24     */
25    public void setEnableExportItem(boolean f){
26      imageExportItem.setEnabled(f);
```

```
27    }
28    /*setta la visibilita della voce save nel menu file del bottone.
29     * invocato da mouseController
30     */
31    public void setEnableUndo(boolean f){
32      undoItem.setEnabled(f);
33      undoButton.setEnabled(f);
34    }
35    /*setta la visibilita della voce save nel menu file e del bottone.
36     * invocato da mouseController
37     */
38    public void setEnableRedo(boolean f){
39      redoItem.setEnabled(f);
40      redoButton.setEnabled(f);
41    }
42
43    public void setEnableSaveAs(boolean b) {
44      saveAsItem.setEnabled(b);
45    }
46    public void setEnableCanc(boolean b){
47      cancItem.setEnabled(b);
48    }
49    public File getCurrentFile() {
50      return currentFile;
51    }
52
53    public void setCurrentFile(File currentFile) {
54      this.currentFile = currentFile;
55    }
56    public static MainWindow getMainWindow(){
57      return thisClass;
58    }
59
60    public int getLenghtSave() {
61      return lenghtSave;
62    }
63
64    public void setLenghtSave(int lenghtSave) {
65      this.lenghtSave = lenghtSave;
66    }
67
68    public boolean getSelectInsert() {
69      return selectInsert;
70    }
71    public boolean getSelectCanc() {
72      return selectCanc;
73    }
74  }
```

Listato 4.7: Frammenti di *MainWindow*

4.2.3 La Classe SafeOverwriteFileChooser

La classe *SafeOverwiteFileChooser* è utilizzata per defnire il comportamento delle finestre di dialogo usate per avvertire l'utente in particolari condizioni di pericolo. Il metodo *approveSelection()* è stato ridefinito in modo da variare il suo comportamento a seconda del tipo di finestra di dialogo che lo ha invocato (linee 21 e 33 del listato).

```
1  public class SafeOverwriteFileChooser  extends JFileChooser {
2
3    private static final long serialVersionUID = 1L;
4
5    private static final String DEFAULTCONFIRMTITLE = "Confirm";
6      private static final String DEFAULTCONFIRMMESSAGE = "The␣file␣''{0}''␣already␣exists.\nDo␣you␣want␣to␣
           replace␣it?";
7
8      private String confirmTitle;
9      private String confirmMessage;
10 /*costruttore*/
11     public SafeOverwriteFileChooser() {
12         this(DEFAULTCONFIRMTITLE, DEFAULTCONFIRMMESSAGE);
13     }
14 /*costruttore */
15     public SafeOverwriteFileChooser(String confirmTitle, String confirmMessage) {
16         this.confirmTitle = confirmTitle;
17         this.confirmMessage = confirmMessage;
18     }
19 /*overwrite*/
20     public void approveSelection() {
21         if ((getDialogType() == SAVE_DIALOG) || (getDialogType() == CUSTOM_DIALOG)) {
22             File f = getSelectedFile();
23             if (f.exists()) {
24                 String msg = MessageFormat.format(confirmMessage, new Object[] { f });
25                 int r = JOptionPane.showConfirmDialog(this, msg, confirmTitle, JOptionPane.YES_NO_OPTION);
26                 if (r == JOptionPane.NO_OPTION) {
27                     //Returns without approve the selection.
28                     return;
29                 }
30             }
31         }
32         else if(getDialogType() == OPEN_DIALOG){
33             File f = getSelectedFile();
34           if(!(f.getName().endsWith(".osm"))){
35                String msg = "Only␣.osm␣format␣are␣accepted!";
36                JOptionPane.showMessageDialog(this,msg,"Error.Can't␣open␣this␣file.",JOptionPane.
                       ERROR_MESSAGE);
37              return;
38              }
39         }
40         // Approves the selection normally.
41         super.approveSelection();
42     }
43 }
```

Listato 4.8: La classe *SaveOverWriteFileChooser*

4.3 Il package Event

I package Event contiene le classe che gestiscono gli eventi del mouse, della tastiera e della finestra principale di Osmosis. Per una trattazione competa degli eventi si veda la documentazione ufficiale della Oracle[2] [14].

[2]Il 27 gennaio 2010, la Sun Microsystem è stata acquistata dalla Oracle Corporation

4.3.1 Gestore Eventi della Finestra

La classe *MainWindowListener* è stata aggiunta in questo package per poter personalizzare le funzioni offerte dal gestore della finestra. Il metodo *windowClosing()* che gestisce gli eventi alla pressione dell'icona X è stato ridefinito permettendo ora il salvataggio della lavagna nei casi visti nel paragrafo 3.3.3.

```
1  public class MainWindowListener implements WindowListener {
2
3    privaPe SafeOverwriteFileChooser fcs = new SafeOverwriteFileChooser();
4    [..]
5    @Override
6    /*pressione X
7     */
8    public void windowClosing(WindowEvent arg0) {
9      /*file non salvato e non modificato -> vuoto*/
10     DrawDescription dd = MainWindow.getMainWindow().getBoardPanel().getBoard();
11     if (MainWindow.getMainWindow().getBoardPanel().getBoard().isEmpty()){
12       System.exit(0);
13     }
14     /*file NON salvato */
15     else if(MainWindow.getMainWindow().getCurrentFile() == null){
16       int n = JOptionPane.showOptionDialog(MainWindow.getMainWindow(), "Save␣current␣file?", "Osmosis",
17           JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null, null, null);
18
19       if (n== JOptionPane.OK_OPTION) {//si
20         fcs.setDialogTitle("Save␣As..");
21         fcs.setFileFilter(new OsmFileFilter());
22         //In response to a button click:
23         int ret = fcs.showSaveDialog(MainWindow.getMainWindow());
24         MainWindow.getMainWindow().getBoardPanel().setWaitCursor();
25         if (ret == JFileChooser.APPROVE_OPTION){
26           MainWindow.getMainWindow().routineSave(fcs.getSelectedFile());
27           System.exit(0); //ho salvato ora esco
28         }
29       }
30       if(n==JOptionPane.NO_OPTION){
31         System.exit(0);
32       }
33     }
34     /* file salvato ma modificato */
35     else {
36       int n = JOptionPane.showOptionDialog(MainWindow.getMainWindow(), "Save␣"+ MainWindow.getMainWindow().
             getCurrentFile()+"?", "Osmosis",
37           JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null, null, null);
38       switch(n){
39       case(0):
40         /*SALVA ED ESCI -SI-*/
41         MainWindow.getMainWindow().getBoardPanel().setWaitCursor();
42       if (n == JFileChooser.APPROVE_OPTION){
43          MainWindow.getMainWindow().routineSave(MainWindow.getMainWindow().getCurrentFile());
44         System.exit(0);
45       }
46       case(1): /* NON SALVA ED ESCI-NO-*/
47         System.exit(0);
48       }
49     }
50   }
51   [..]
```

```
52 }
```

Listato 4.9: La classe *MainWindowListener* definisce il comportamento del gestore della finestra principale di Osmosis

4.3.2 La Classe MyKeyAdapter

La classe *MyKeyAdapter* è un estensione di KeyAdapter che descrive gli eventi associati alla pressione di un tasto sulla tastiera.E' stato definito il comportamento per due stasti: Canc (linea 7) che consente di rimuovere una linea precedentemente selezionata, ed Esc (linea 23) che viene utilizzato per inserire il testo sulla lavagna. Questo oggetto viene aggiunto ai ricevitori degli eventi della lavagna quando vengono attivate le funzioni di inserimento testo e cancellazione stringa, mentre viene rimosso quando vengono disattivati.

```
1  public class MyKeyAdapter extends KeyAdapter{
2
3    @Override
4    public void keyPressed(KeyEvent e) {
5      // TODO Auto-generated method stub
6      String toSend = "";
7      if (e.getKeyCode() == KeyEvent.VK_DELETE){
8        Vector<Object> obj = MainWindow.getMainWindow().getBoardPanel().getBoard().deleteSelected();
9        if (OsmosisBoard.ap2ap_is_connected){
10         for (int i=0; i<obj.size();i++){
11           if (obj.get(i) instanceof DescriptedPoint){
12             toSend = newSParser.requestParse(Cmd.del, (DescriptedPoint)obj.get(i));
13           }
14           if (obj.get(i) instanceof LocatedString){
15             toSend = newSParser.requestParse(Cmd.del, (LocatedString)obj.get(i));
16           }
17           Ap2Ap.create().sendDatagram(toSend);
18         }
19       }
20       MainWindow.getMainWindow().getBoardPanel().redraw();
21     }
22     if (e.getKeyCode() == KeyEvent.VK_ESCAPE){
23       System.out.println("premuto␣ESC");
24       MainWindow.getMainWindow().getTextInsertDialog().setVisible(false);
25       //mette solo la prima stringa
26       int start = -1;
27       int end = -1;
28       try {
29         start = MainWindow.getMainWindow().getTextArea().getLineStartOffset(0);
30         end = MainWindow.getMainWindow().getTextArea().getLineEndOffset(0);
31       } catch (BadLocationException e1) {
32         // TODO Auto-generated catch block
33         e1.printStackTrace();
34       }
35        MainWindow.getMainWindow().getBoardPanel().getBoard().setTextForLastString(MainWindow.getMainWindow
              ().getTextArea().getText().substring(start,end));
36       MainWindow.getMainWindow().getBoardPanel().redraw();
37       MainWindow.getMainWindow().getTextArea().setText("");
38     }
39     //invia la stringa
40   }
```

```
41 }
```

Listato 4.10: La classe *MyKeyAdapter*

4.3.3 La Classe MouseController

La classe MouseController gestisce gli eventi che si verificano alla pressione dei tasti del mouse. Il metodo *mouseReleased()* è stato ridefinito e vi è stato aggiunto il controllo e la gestione dell'inserimento degli stati nella struttura dati utilizzata dalle operazioni di undo e redo (linee da 12 a 15).

```
1  public class MouseController extends MouseAdapter{
2
3    [..]
4    public void mouseReleased(java.awt.event.MouseEvent e){
5      //attivo le voci nei menu
6      MainWindow.getMainWindow().setEnableUndo(true);
7      MainWindow.getMainWindow().setEnableSaveAs(true);
8      MainWindow.getMainWindow().setEnableExportItem(true);
9      MainWindow.getMainWindow().setEnableCanc(true);
10     // aggiunge lo stato nel vettore di stati
11     if(MainWindow.getMainWindow().getOsmosisState().getNumberBoard() > MainWindow.getMainWindow().
           getOsmosisState().getCurrent()+1){
12       //ho fatto un undo n volte e ora disegno -> elimino (n-current) board dal vettore
13       while(MainWindow.getMainWindow().getOsmosisState().getNumberBoard() > MainWindow.getMainWindow().
             getOsmosisState().getCurrent()+1){
14         MainWindow.getMainWindow().getOsmosisState().removeState(MainWindow.getMainWindow().
               getOsmosisState().getNumberBoard()-1);
15       }
16       MainWindow.getMainWindow().setEnableRedo(false);
17     }
18      MainWindow.getMainWindow().getOsmosisState().addState(MainWindow.getMainWindow().getBoardPanel().
          getBoard());
19   }
```

Listato 4.11: La classe *MouseController*

4.4 Il Package Application

Questo package esporta le classi che vanno ad implementare le strutture dati *DrawDescription*, *DescriptionPoint* e *LocatedString* e le classi che implementano le funzionalità aggiuntive di Osmosis come *SaveAndLoad*, *ImageImporterExporter*, *UndoRedoDescription*, *OsmFileFilter* e *JpegFileFilter*. In seguito vedremo meglio queste ultime.

4.4.1 La Classe UndoRedoDescription

La classe *UndoRedoDescription* fornisce una struttura dati in cui vengono memorizzati gli stati della lavagna. Uno stato è dato dalle linee e dalle stringhe di testo presenti nella lavagna. L'aggiunta di una nuova linea o di una stringa coincide perciò con la creazione di un nuovo stato. Tutti gli stati sono ordinati temporalmente e tramite un indice (*currentState*) è possibile accedere allo stato corrente (cioè la rappresentazione della lavagna attuale). Il metodo *addState(DrawDescription board)* permette di inserire un nuovo oggetto DrawDescription che contiene due vettori: uno di *DescriptedPoint* e uno di *LocatedStrings*. Di seguito viene riportato il costruttore della classe *DrawDescription* che si occupa di creare il nuovo oggetto prendendo in input i due vettori.

```
1  [..]
2  /*costruttore usato da UndoRedoDescription
3
4    public DrawDescription(Vector<DescriptedPoint> punti,Vector<LocatedString> stringhe ){
5      //creo due NUOVI vettori
6      mousePoints = new Vector<DescriptedPoint>(punti.size());
7      //trovare modo piu veloce per copiare i vettori
8      for(int i=0; i<punti.size(); i++){
9        mousePoints.add(punti.get(i));
10     }
11     strings = new Vector<LocatedString>(stringhe.size());
12     for(int i=0; i<stringhe.size()-1; i++){
13       strings.add(stringhe.get(i));
14     }
15   }
```

Listato 4.12: Il costruttore della classe *DrawDescription* effettua la copia degli oggetti dei vettori

I metodi *Undo()* e *Redo()* del listato successivo diminuiscono o incrementano l'indice in modo da poter poi selezionare la lavagna (metodo *getCurrentState()* precedente o successiva.

La struttura dati principale che si occupa di ordinare e memorizzare gli oggetti è un Vettore che viene fornito nella classe *java.util.Vector*. Un vettore cresce dinamicamente incrementando lo spazio riservato agli oggetti con il crescerne del numero. Inoltre sono disponibili dei metodi che permettono di aggiungere, rimuovere e prelevare oggetti (rispettivamente *add()*, *remove()*, *get(index)*).

```
1  public class UndoRedoDescription {
2
3    private Vector<DrawDescription> osmosisState= null;
4    private int currentState = -1; //indice dello stato attivo al momento
5    private int numberBoard = 0;
6
7    /*costruttore
```

```
8     *
9     */
10    public UndoRedoDescription(){
11      osmosisState = new Vector<DrawDescription>();
12    }
13
14    public Vector<DrawDescription> getOsmosisState() {
15      return osmosisState;
16    }
17
18    public void setOsmosisState(Vector<DrawDescription> osmosisState) {
19      this.osmosisState = osmosisState;
20    }
21
22    public int getCurrent() {
23      return currentState;
24    }
25
26    public void setCurrentState(int currentState) {
27      this.currentState = currentState;
28    }
29
30    public int getNumberBoard() {
31      return numberBoard;
32    }
33
34    public void setNumberBoard(int numberBoard) {
35      this.numberBoard = numberBoard;
36    }
37
38    /**aggiunge un drawDescription al vettore osmosisBoard
39     *
40     */
41    public void addState(DrawDescription board){
42      osmosisState.add(new DrawDescription(board.getVectorPoint(), board.getVectorStrings()));
43      /*incremento variabili */
44      currentState++; //aggiunge la prima board(vuota) nella cella 0 e currentState va a 0
45      numberBoard++;
46    }
47    /**rimuove l'ultimo stato del vettore
48     *
49     */
50    public void removeState(int i){
51      osmosisState.remove(i);
52      numberBoard--;
53    }
54
55    /****UNDO**** restituisce lo stato prcedente
56     * @return lo stato(drawDescription) che rappresenta i vettori
57     */
58    public void undo(){
59      currentState--;
60    }
61    /**ritorna lo stato corrente
62     *
63     */
64    public DrawDescription getCurrentState(){
65      return osmosisState.get(currentState);
66    }
67
68    /*****REDO**** seleziona lo stato successivo
69     * @return lo stato(drawDescription) che rappresentai vettori
70     */
71    public void redo(){
72      currentState++;
73    }
74
```

```
75    /* restituisce il numero di elementi del vettore (conta gli elementi per ogni DrawDescription)
76     *
77     */
78    public int stateSize(){
79      return osmosisState.size();
80    }
81
82    /*cancella tutti gli stati presenti nel vettore
83     *
84     */
85    public void resetVectorState(){
86      osmosisState.clear();
87      currentState=-1;
88      numberBoard=0;
89    }
90  [..]
91  }
```

Listato 4.13: La classe *UndoRedoDescription* implementa le strutture dati e i metodi per gestire le funzionalità undo e redo

4.4.2 La Classe SaveAndLoad

La classe *SaveAndLoad* gestisce la scritture su file. Al codice già presente è stato aggiunto l'inserimento delle estensioni che avviene alla linea 5 invocando il metodo *makeExtension()*.

```
1   public class SaveAndLoad {
2     [..]
3     public static void save (String obj, String name){
4       File file = new File(makeExtension(name));
5       FileWriter fw;
6       BufferedWriter out;
7       try {
8         fw = new FileWriter(file,false);
9         out = new BufferedWriter(fw);
10        out.write(obj);
11        out.close();
12      } catch (IOException e) {
13        // TODO Auto-generated catch block
14        e.printStackTrace();
15      }
16    }
17    private static String makeExtension(String name){
18      String suffisso =".osm";
19      if(name.endsWith(suffisso))
20        return name;
21      else {
22        return name+suffisso;
23      }
24    }
25  [..]
```

Listato 4.14: Frammento della classe *SaveAndLoad*

4.4.3 La Classe ImageImporterExporter

Questa classe consente di salvare la lavagna in formato disegno. Il metodo *MakeExtension()* inserisce, se necessario, l'estensione .jpeg

```
1  public class ImageImporterExporter {
2  [..]
3    public static void exportImage(BufferedImage img, File file){
4      try {
5        ImageIO.write(img, "JPEG", file);
6      } catch (IOException e) {
7        // TODO Auto-generated catch block
8        e.printStackTrace();
9      }
10   }
11   public static String makeExtension(String name){
12     String suffisso =".jpeg";
13   //  System.out.println(name);
14     if(name.endsWith(suffisso)){
15       return name;
16     }
17     else {
18       return name+suffisso;
19     }
20   }
21 [..]
22   public static Image importImage (File file){
23     Image image = null;
24     try {
25       image = ImageIO.read(file);
26     } catch (IOException e) {
27       // TODO Auto-generated catch block
28       e.printStackTrace();
29     }
30     return image;
31   }
32 }
```

Listato 4.15: La classe *ImageImporterExporter*

4.4.4 Le Classi per i Filtri e le Estensioni

Le due classi *OsmFileFilter* e *JpegFileFilter* vengono utilizzate per filtrare i file con diverse estensioni durante le fasi di salvataggio, caricamento ed esportazione. Entrambe estendono la classe FileFilter e ridefiniscono il metodo *accept(File file)*.

```
1  public class OsmFileFilter extends FileFilter{
2     public boolean accept(File file) {
3          if (file.isDirectory()) return true;
4          String fname = file.getName().toLowerCase();
5          return fname.endsWith("osm");
6     }
7     public String getDescription() {
8          return "File␣di␣Osmosis";
9     }
10 }
```

Listato 4.16: La classe *OsmFileFilter*

```
1  public class OsmFileFilter extends FileFilter{
2     public boolean accept(File file) {
3          if (file.isDirectory()) return true;
4          String fname = file.getName().toLowerCase();
5          return fname.endsWith("jpeg");
6     }
7     public String getDescription() {
8          return "File␣jpeg";
9     }
10 }
```

Listato 4.17: La classe *JpegFileFilter*

4.5 Il Package Action

Il package Action contiene le classi che definiscono le azioni da compiere quando si invocano (per esempio da menù) certe operazioni. Tutte le classi estendono la classe astratta *AbstractAction* che fornisce un implementazione di default per l'interfaccia Action. Sono quindi già presenti i metodi get e set per le proprietà degli oggetti Action (icon, text, enabled) ed è necessario implementare il metodo *ActionPerform()*.

4.5.1 La Classe NewAction

L'oggetto NewAction viene creato nella classe *MainWindow* quando viene selezionata la voce new del menù file. Nel seguente listato possiamo osservare tutti i controlli necessari per realizzare le discriminazioni viste nel paragrafo 3.3.3. Alla linea 8 abbiamo i controlli per capire se il file è stato già salvato e se è stato modificato (linea 9) oppure no (linea 39). Alla linea 42 viene controllato se il file è stato modificato, mentre se non lo è stato viene eseguito il codice dalla linea 47. In tutti questi casi vengono svolte operazioni differenti che permettono il salvataggio del file, la modifica della lavagna, la disattivazione delle voci nel menù e il reset dei dati contenuti nella struttura dati che gestisce undo e redo.

```
1  public class NewAction extends AbstractAction{
2    [..]
3    private static final long serialVersionUID = 1L;
4    private final SafeOverwriteFileChooser fcs = new SafeOverwriteFileChooser();
5    public void actionPerformed(java.awt.event.ActionEvent e) {
6      DrawDescription dd = MainWindow.getMainWindow().getBoardPanel().getBoard();
7      /*file salvato */
8      if (MainWindow.getMainWindow().getLenghtSave() > 0) {          /* poi  modificato */
9        if(MainWindow.getMainWindow().getLenghtSave() != dd.pointN()+dd.stringN()){
```

```
10          int n = JOptionPane.showOptionDialog(MainWindow.getMainWindow(), "Save␣"+ MainWindow.getMainWindow
                ().getCurrentFile().getName()+"?", "Osmosis",JOptionPane.YES_NO_OPTION, JOptionPane.
                QUESTION_MESSAGE, null, null, null);
11          /*case yes e no: salva/non salva e poi apri*/
12          if(n == 0 || n == 1){
13            if (n == 0){
14              /*salva senza mostrare dialog*/  MainWindow.getMainWindow().routineSave(MainWindow.
                    getMainWindow().getCurrentFile());
15              MainWindow.getMainWindow().setEnableSave(false);
16            }//non salva e apre
17            /*nuovo*/
18            MainWindow.getMainWindow().getBoardPanel().getBoard().clear();
19            MainWindow.getMainWindow().getBoardPanel().redraw();
20            MainWindow.getMainWindow().setCurrentFile(null);
21            MainWindow.getMainWindow().setEnableSave(false);
22            MainWindow.getMainWindow().setEnableUndo(false);
23            MainWindow.getMainWindow().setEnableRedo(false);
24            MainWindow.getMainWindow().getOsmosisState().resetVectorState();     MainWindow.getMainWindow().
                  getOsmosisState().addState(MainWindow.getMainWindow().getBoardPanel().getBoard());
25          }
26        }
27        /**poi NON  modificato -> nuovo */
28        else{
29          MainWindow.getMainWindow().getBoardPanel().getBoard().clear();
30          MainWindow.getMainWindow().getBoardPanel().redraw();
31          MainWindow.getMainWindow().setCurrentFile(null);
32          MainWindow.getMainWindow().setEnableSave(false);
33          MainWindow.getMainWindow().setEnableUndo(false);
34          MainWindow.getMainWindow().setEnableRedo(false);
35          MainWindow.getMainWindow().getOsmosisState().resetVectorState();   MainWindow.getMainWindow().
                getOsmosisState().addState(MainWindow.getMainWindow().getBoardPanel().getBoard());
36        }
37      }
38      /* file NON salvato */
39      else{
40        /* poi NON modificato quindi vuoto--> nuovo*/
41        if(dd.isEmpty()){
42          MainWindow.getMainWindow().getBoardPanel().getBoard().clear();
43          MainWindow.getMainWindow().getBoardPanel().redraw();
44          MainWindow.getMainWindow().setCurrentFile(null);
45        }
46        /* poi modificato-> save current file?*/
47        else{
48          int n = JOptionPane.showOptionDialog(MainWindow.getMainWindow(), "Save␣"+ "Save␣current␣file?", "
                    Osmosis",
49              JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null, null, null);
50          if(n==0){//SI save as dialog
51            fcs.setDialogTitle("Save␣file␣as..");
52            fcs.setFileFilter(new OsmFileFilter());
53            int ret = fcs.showSaveDialog(MainWindow.getMainWindow().getContentPane());
54            if (ret == JFileChooser.APPROVE\_OPTION){
55              MainWindow.getMainWindow().routineSave(fcs.getSelectedFile());
56              /*nuovo */
57               MainWindow.getMainWindow().getBoardPanel().getBoard().clear();
58              MainWindow.getMainWindow().getBoardPanel().redraw();
59              MainWindow.getMainWindow().setCurrentFile(null);
60              MainWindow.getMainWindow().setEnableSave(false);
61              MainWindow.getMainWindow().setEnableUndo(false);
62              MainWindow.getMainWindow().setEnableRedo(false);
63            }
64          }
65          if(n==1){ //non salva il file corrente
66            /*nuovo*/
67            MainWindow.getMainWindow().getBoardPanel().getBoard().clear();
68            MainWindow.getMainWindow().getBoardPanel().redraw();
69            MainWindow.getMainWindow().setCurrentFile(null);
70            MainWindow.getMainWindow().setEnableSave(false);
```

```
71            MainWindow.getMainWindow().setEnableRedo(false);
72            MainWindow.getMainWindow().setEnableUndo(false);
73          }
74        }
75      }
76    }
77  }
```

Listato 4.18: La classe *NewAction* implementa il comportamento dell'azione associata alla voce New

4.5.2 La Classe OpenAction

OpenAction definisce l'azione associata alla selezione della voce open nel menù file. Nel metodo *actionPerformed()* sono stati definiti tutti i controlli necessari per gestire i salvataggi di sicurezza che sono logicamente simili a quelli della classe *NewAction*. Possiamo inoltre osservare alla linea 24 l'aggiunta del filtro per le estensioni.

```
1  public class OpenAction extends AbstractAction{
2    private static final long serialVersionUID = 1L;
3    final javax.swing.JFileChooser fc = new javax.swing.JFileChooser();
4    final SafeOverwriteFileChooser fcs = new SafeOverwriteFileChooser();
5
6    public void actionPerformed(java.awt.event.ActionEvent e) {
7      DrawDescription dd = MainWindow.getMainWindow().getBoardPanel().getBoard();
8      /*file salvato */
9      if (MainWindow.getMainWindow().getLenghtSave() > 0){
10       /* poi stato modificato */
11       if(MainWindow.getMainWindow().getLenghtSave() != dd.pointN()+dd.stringN()){
12         int n = JOptionPane.showOptionDialog(MainWindow.getMainWindow(), "Save␣"+ MainWindow.getMainWindow
               ().getCurrentFile().getName()+"?", "Osmosis",
13             JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.QUESTION_MESSAGE, null, null, null);
14         /*case yes e no: salva/non salva e poi apri*/
15         if(n == 0 || n == 1){
16           if (n == 0){
17             /*salva senza mostrare dialog*/  MainWindow.getMainWindow().routineSave(MainWindow.
                    getMainWindow().getCurrentFile());
18             MainWindow.getMainWindow().setEnableSave(false);
19             MainWindow.getMainWindow().setEnableUndo(false);
20             MainWindow.getMainWindow().setEnableRedo(false);
21              MainWindow.getMainWindow().getOsmosisState().resetVectorState();  MainWindow.getMainWindow().
                    getOsmosisState().addState(MainWindow.getMainWindow().getBoardPanel().getBoard());
22           }
23           /*apri*/
24           fcs.setFileFilter(new OsmFileFilter());
25           int ret = fcs.showOpenDialog(MainWindow.getMainWindow().getContentPane());
26           if (ret == JFileChooser.APPROVE_OPTION){
27              MainWindow.getMainWindow().routineLoad(fcs.getSelectedFile());
28             MainWindow.getMainWindow().setEnableSave(false);
29             MainWindow.getMainWindow().setEnableUndo(false);
30             MainWindow.getMainWindow().setEnableRedo(false);
31              MainWindow.getMainWindow().getOsmosisState().resetVectorState();
32              MainWindow.getMainWindow().getOsmosisState().addState(MainWindow.getMainWindow().getBoardPanel
                    ().getBoard());
33           }
34         }
35         /*case annulla: non salva non apri*/
```

```
36          if (n == 2){
37          }
38        }
39        /* poi NON  modificato-> apri */
40        else{
41          fcs.setFileFilter(new OsmFileFilter());
42          int ret = fcs.showOpenDialog(MainWindow.getMainWindow().getContentPane());
43          if (ret == JFileChooser.APPROVE_OPTION){
44            MainWindow.getMainWindow().routineLoad(fcs.getSelectedFile());
45            MainWindow.getMainWindow().setEnableSave(false);
46            MainWindow.getMainWindow().setEnableUndo(false);
47            MainWindow.getMainWindow().setEnableRedo(false);
48             MainWindow.getMainWindow().getOsmosisState().resetVectorState();
49             MainWindow.getMainWindow().getOsmosisState().addState(MainWindow.getMainWindow().getBoardPanel()
                    .getBoard());
50          }
51        }
52      }
53      /* file NON salvato */
54      else{
55        /* poi NON modificato quindi vuoto --> apri */
56        if(dd.isEmpty()){
57          fcs.setFileFilter(new OsmFileFilter());
58          int ret = fcs.showOpenDialog(MainWindow.getMainWindow().getContentPane());
59          if (ret == JFileChooser.APPROVE_OPTION){
60            MainWindow.getMainWindow().routineLoad(fcs.getSelectedFile());
61            MainWindow.getMainWindow().setEnableSave(false);
62            MainWindow.getMainWindow().setEnableUndo(false);
63            MainWindow.getMainWindow().setEnableRedo(false);
64             MainWindow.getMainWindow().getOsmosisState().resetVectorState();
65             MainWindow.getMainWindow().getOsmosisState().addState(MainWindow.getMainWindow().getBoardPanel()
                    .getBoard());
66          }
67        }
68        /* poi modificato-> save as*/
69        else{
70          int n = JOptionPane.showOptionDialog(MainWindow.getMainWindow(), "Save␣current␣file?", "Osmosis",
71              JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null, null, null);
72          switch(n){
73          case(0):{ // si
74            fcs.setDialogTitle("Save␣as..");
75            fcs.setFileFilter(new OsmFileFilter());
76            int ret = fcs.showSaveDialog(MainWindow.getMainWindow().getContentPane());
77            if (ret == JFileChooser.APPROVE_OPTION){
78               MainWindow.getMainWindow().routineSave(fcs.getSelectedFile());
79              MainWindow.getMainWindow().setEnableSave(false);
80              MainWindow.getMainWindow().setEnableUndo(false);
81              MainWindow.getMainWindow().setEnableRedo(false);
82               MainWindow.getMainWindow().getOsmosisState().resetVectorState();
83               MainWindow.getMainWindow().getOsmosisState().addState(MainWindow.getMainWindow().getBoardPanel
                      ().getBoard());
84            }
85          }
86          /*apri*/
87          case(1):{}
88          fcs.setFileFilter(new OsmFileFilter());
89          int retOp = fcs.showOpenDialog(MainWindow.getMainWindow().getContentPane());
90          if (retOp == JFileChooser.APPROVE_OPTION){
91
92            MainWindow.getMainWindow().routineLoad(fcs.getSelectedFile());
93            MainWindow.getMainWindow().setEnableSave(false);
94            MainWindow.getMainWindow().setEnableUndo(false);
95            MainWindow.getMainWindow().setEnableRedo(false);
96             MainWindow.getMainWindow().getOsmosisState().resetVectorState();
97             MainWindow.getMainWindow().getOsmosisState().addState(MainWindow.getMainWindow().getBoardPanel()
                    .getBoard());
98          }
```

```
99          }
100       }
101     }
102   }
103 }
```

Listato 4.19: La classe *OpenAction* gestisce l'azione del caricamento di file

4.5.3 La Classe SaveAction

La classe SaveAction implementa l'azione del salvataggio. Questa può essere invocata sia attraverso il menù file sia attraverso la barra laterale di accesso rapido.

```
1  public class SaveAction extends AbstractAction{
2    private static final long serialVersionUID = 1L;
3
4    @Override
5    public void actionPerformed(ActionEvent e) {
6      if(MainWindow.getMainWindow().getCurrentFile() == null){
7        SafeOverwriteFileChooser fcs = new SafeOverwriteFileChooser();
8        fcs.setDialogTitle("Save␣as..");
9        fcs.setFileFilter(new OsmFileFilter());
10       int ret = fcs.showSaveDialog(MainWindow.getMainWindow().getContentPane());
11       if (ret == JFileChooser.APPROVE_OPTION){
12         MainWindow.getMainWindow().routineSave(fcs.getSelectedFile());
13         MainWindow.getMainWindow().setEnableSave(false);
14   //      MainWindow.getMainWindow().getOsmosisState().resetVectorState();   //MainWindow.getMainWindow().
           getOsmosisState().addState(MainWindow.getMainWindow().getBoardPanel().getBoard());
15       }
16     }
17     else { /* file gia salvato in precedenza, risalvo*/      MainWindow.getMainWindow().routineSave(
           MainWindow.getMainWindow().getCurrentFile());
18       MainWindow.getMainWindow().setEnableSave(false);
19     //  MainWindow.getMainWindow().getOsmosisState().resetVectorState();     //MainWindow.getMainWindow().
           getOsmosisState().addState(MainWindow.getMainWindow().getBoardPanel().getBoard());
20     }
21   }
22 }
```

Listato 4.20: La classe *SaveAction* definisce l'azione di salvataggio

4.5.4 La Classe ExitAction

La classe ExitAction implementa l'azione della voce exit nel menù file.

```
1  public class ExitAction extends AbstractAction{
2
3    private static final long serialVersionUID = 1L;
4    final SafeOverwriteFileChooser fcs = new SafeOverwriteFileChooser();
5    public void actionPerformed(java.awt.event.ActionEvent e) {
6      DrawDescription dd = MainWindow.getMainWindow().getBoardPanel().getBoard();
7      /*file salvato senza ulteriori modifiche anche vuoto*/
8      if (MainWindow.getMainWindow().getLenghtSave() == (dd.pointN()+dd.stringN())){
9        System.exit(0);
10     }
11     /*file non salvato in precedenza*/
```

```
12      else if(MainWindow.getMainWindow().getCurrentFile() == null){
13        int n = JOptionPane.showOptionDialog(MainWindow.getMainWindow(), "Save␣current␣file?", "Osmosis",
14            JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null, null, null);
15        if(n==0) {//SI
16          fcs.setDialogTitle("Save␣As..");
17          //In response to a button click:
18          int ret = fcs.showSaveDialog(MainWindow.getMainWindow());
19          if (ret == JFileChooser.APPROVE_OPTION){
20            MainWindow.getMainWindow().routineSave(fcs.getSelectedFile());
21            System.exit(0);
22          }
23        }
24        else {//NO non salva e esci-> annulla
25          System.out.print("sono␣qui␣non␣salvo␣ed␣esco");
26          System.exit(0);
27        }
28      }
29      /* file salvato ma modificato. chiedo solo di salvare*/
30      else {
31        int n = JOptionPane.showOptionDialog(MainWindow.getMainWindow(), "Save␣"+ MainWindow.getMainWindow().
              getCurrentFile().getName()+"?", "Osmosis",
32            JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null, null, null);
33        switch(n){
34        case(0):
35          /*SI:SALVA ED ESCI */
36          if (n == JFileChooser.APPROVE_OPTION){         MainWindow.getMainWindow().routineSave(MainWindow.
                getMainWindow().getCurrentFile());
37            System.exit(0);
38          }
39        case(1): /*NO: NON SALVA ED ESCI*/
40          System.exit(0);
41        }
42      }
43    }
44  }
```

Listato 4.21: La classe *ExitAction* definisce l'azione da eseguire invocando la voce Exit

4.5.5 La Classe UndoAction

La classe *UndoAction* fornisce l'azione associata sia alla voce undo del menù tools, sia all'icona presente nella barra laterale a sinistra. L'azione implementata nel metodo *ActionPerform()* può essere invocata sia in locale che in remoto. Se eseguita in locale (linea 13) allora viene inviato un datagram agli utenti connessi (linee 14 e 15) che permetterà di eseguire l'azione undo anche nella loro lavagna.

```
1   public class UndoAction extends AbstractAction{
2
3     private static final long serialVersionUID = 1L;
4
5     @Override
6     public void actionPerformed(ActionEvent arg0) {
7       //controllo se ho board memorizzate
8       if(MainWindow.getMainWindow().getOsmosisState().getCurrent()> 0){
9         MainWindow.getMainWindow().getOsmosisState().undo();//setta currentBoard-1
10          MainWindow.getMainWindow().getBoardPanel().setBoard(MainWindow.getMainWindow().getOsmosisState().
              getCurrentState());
```

```
11  [..]
12        /***  invio datagram UNDO solo se sono io ad utilizzare la lavagna*/
13        if(MainWindow.getMainWindow().getLockToggleButton().getToolTipText() == "writing"){
14          if (OsmosisBoard.ap2ap_is_connected){
15            Ap2Ap.create().sendDatagram("undo");
16          }
17        }
18        /*****************************/
19        MainWindow.getMainWindow().getBoardPanel().redraw();
20        MainWindow.getMainWindow().setEnableRedo(true);
21      }
22      if(MainWindow.getMainWindow().getOsmosisState().getCurrent()<1){
23        MainWindow.getMainWindow().setEnableUndo(false);
24      }
25    }
26  }
```

Listato 4.22: La classe *UndoAction* implementa l'azione dell'operazione undo

4.5.6 La Classe RedoAction

La classe *RedoAction* fornisce l'azione associata sia alla voce undo del menù tools, sia all'icona presente nella barra laterale a sinistra. L'azione implementata nel metodo *ActionPerform()* può essere invocata sia in locale che in remoto e la sua implementazione è del tutto simile a quello di redo.

```
1   public class RedoAction extends AbstractAction{
2
3     private static final long serialVersionUID = 1L;
4
5     @Override
6     public void actionPerformed(ActionEvent arg0) {
7       //controllo redo disponibile
8       if(MainWindow.getMainWindow().getOsmosisState().getNumberBoard() > MainWindow.getMainWindow().
            getOsmosisState().getCurrent()+1){
9         MainWindow.getMainWindow().getOsmosisState().redo();
10        MainWindow.getMainWindow().getBoardPanel().setBoard(MainWindow.getMainWindow().getOsmosisState().
              getCurrentState());
11        /***  invio datagram REDO se sono io a utilizzare la lavagna*/
12        if(MainWindow.getMainWindow().getLockToggleButton().getToolTipText() == "writing"){
13          if (OsmosisBoard.ap2ap_is_connected){
14            Ap2Ap.create().sendDatagram("redo");
15          }
16        }
17        /*****************************/
18        MainWindow.getMainWindow().getBoardPanel().redraw();
19        MainWindow.getMainWindow().setEnableUndo(true);
20        if(MainWindow.getMainWindow().getOsmosisState().getNumberBoard() <= MainWindow.getMainWindow().
              getOsmosisState().getCurrent()+1){
21          MainWindow.getMainWindow().setEnableRedo(false);
22        }
23      }
24      else{}
25      //  System.out.println("REDO non disponibile");
26    }
27  }
```

Listato 4.23: La classe *RedoAction* implementa l'azione dell'operazione redo

4.5.7 La Classe CancAction

La classe *CancAction* implementa le operazioni che vengono realizzate per cancellare una linea o del testo. Per attivare la funzione è necessario aggiungere il ricevitore di eventi per la tastiera (metodo *addKeyListener* alla linea 7) e rimuovere il ricevitore di eventi del mouse (metodo *MouseController* all linea 8). Se invece la funzione era già attiva le due operazioni sopra vengono invertite: viene rimosso il ricevitore della tastiera e aggiunto quello del mouse (metodi *removeKeyListener() e addMouseMotionController*). Le altre istruzioni permettono di settare i cursori utilizzati (linee 9 e 22) e di impostare delle variabili di controllo.

```
1  public class CancAction extends AbstractAction{
2
3    private static final long serialVersionUID = 1L;
4    public void actionPerformed(ActionEvent e) {
5      if (!MainWindow.getMainWindow().getSelectCanc()){ //  era disattivato lo attivo
6         MainWindow.getMainWindow().getBoardPanel().addKeyListener(MainWindow.getMainWindow().getMyKeyAdapter
               ());
7         MainWindow.getMainWindow().getBoardPanel().removeMouseMotionController();
8        MainWindow.getMainWindow().getBoardPanel().setCutCursor();
9        MainWindow.getMainWindow().getBoardPanel().requestFocus();
10       MainWindow.getMainWindow().setSelectCanc(true);
11       MainWindow.getMainWindow().setSelectInsert(false);
12        MainWindow.getMainWindow().getSelectToggleButton().setBackground(Color.blue);
13        MainWindow.getMainWindow().getInsertToggleButton().setBackground(Color.white);
14     }
15     else{ //era attivo lo disattivo
16       MainWindow.getMainWindow().setSelectCanc(false);
17       MainWindow.getMainWindow().setSelectInsert(false);
18        MainWindow.getMainWindow().getSelectToggleButton().setBackground(Color.white);
19        MainWindow.getMainWindow().getBoardPanel().removeKeyListener(MainWindow.getMainWindow().
               getMyKeyAdapter());
20       MainWindow.getMainWindow().getBoardPanel().addMouseMotionController();
21       MainWindow.getMainWindow().getBoardPanel().setDefCursor();
22     }
23   }
24 }
```

Listato 4.24: La classe *CancAction*

4.5.8 La Classe TextAction

La classe *TextAction* implementa le operazioni che permettono di inserire del testo. Per attivare la funzione è necessario aggiungere o rimuovere il ricevitore di eventi della tastiera e del mouse nella maniera vista nel paragrafo precedente.

```
1  public class TextAction extends AbstractAction{
2
3    public void actionPerformed(java.awt.event.ActionEvent e) {
4      if (!MainWindow.getMainWindow().getSelectInsert()){  // non era attivo lo attivo
5         MainWindow.getMainWindow().getBoardPanel().removeMouseMotionController();
```

```
6         MainWindow.getMainWindow().getBoardPanel().setTextCursor();
7          MainWindow.getMainWindow().getBoardPanel().addKeyListener(MainWindow.getMainWindow().getMyKeyAdapter
               ());
8         MainWindow.getMainWindow().setSelectCanc(false);
9         MainWindow.getMainWindow().setSelectInsert(true);
10         MainWindow.getMainWindow().getInsertToggleButton().setBackground(Color.BLUE);
11         MainWindow.getMainWindow().getSelectToggleButton().setBackground(Color.white);
12      }
13      else{ //era attivo lo disattivo
14         MainWindow.getMainWindow().getInsertToggleButton().setBackground(Color.WHITE);
15        MainWindow.getMainWindow().getBoardPanel().addMouseMotionController();
16         MainWindow.getMainWindow().getBoardPanel().removeKeyListener(MainWindow.getMainWindow().
               getMyKeyAdapter());
17        MainWindow.getMainWindow().getBoardPanel().setDefCursor();
18        MainWindow.getMainWindow().setSelectCanc(false);
19        MainWindow.getMainWindow().setSelectInsert(false);
20      }
21    }
22  }
```

Listato 4.25: La classe *TextAction*

Conclusioni

In questo lavoro di tesi si è descritto lo sviluppo della nuova interfaccia grafica di Osmosis. In primo luogo sono stati presentati i principi base dell'interaction design necessari per comprendere il dominio del problema. Si è ampiamente discusso di quali parametri è necessario considerare per ottenere i principali obiettivi di usabilità e d'esperienza d'uso. Tali principi sono stati poi applicati per analizzare il prototipo già esistente dell'interfaccia e capire in che modo era necessario intervenire. Nel capitolo 1 sono state quindi presentate tutte le modifiche realizzate per garantire i principi base dell'interaction design, argomentando i motivi che le hanno rese necessarie. Infine dopo una completa analisi del codice già esistente si è intervenuti alla sua modifica.

Attualmente l'interfaccia fornisce diversi strumenti che sono accessibili sia dal menù principale sia dalla barra laterale di accesso rapido. Le voci dei menù sono state raggruppate in maniera logica in modo da consentire di aggiungere eventuali nuove funzionalità semplicemente inserendo la nuova voce nel menù a lei più appropiato. E' anche possibile inserire facilmente nuove icone e bottoni nella barra laterale di accesso rapido mantenendo invariata la struttura e l'usabilità dell'interfaccia.

Bibliografia

[1] Helen Sharp Jennifer Preece, Yvonne Rogers. *Interaction Design*. APOGEO, 2004.

[2] Winograd T.. *From computing machinery to interaction design*. Beyond Calculation: the Next Fifty Years of Computing, 1997.

[3] Grudin R. *The computer reaches out: the historical continuity of interface design*. 1990.

[4] Xerox-corporation. `http://www.wikideep.it/xerox-corporation/`.

[5] Macintosh 128k. `http://en.wikipedia.org/wiki/Macintosh_128K`.

[6] Iso. `http://www.iso.org/iso/about.htm`.

[7] Nelson T. *The right way to think about the software design*. Addison-Wesely, 1990.

[8] Thimbleby H. *User Interface Design*. Addison Wesley, 1990.

[9] Norman Don. *La caffetteria del masochista*. Giunti, 1988.

[10] Norman Don. Affordance, conventions and design, 1999.

[11] Osmosis Trac Page. `http://osmosis.trac.cs.unibo.it`.

[12] Matteo Baccan. Skype 4 Java. `www.baccan.it/javaday2007/javaday_skype4java.pdf`, 2007.

[13] Swing Graphics. `http://www.d.umn.edu/~gshute/java/swing/graphics.html`.

[14] Java Event Handling. `http://download.oracle.com/javase/6/docs/api/`.

Ringraziamenti

Desidero innanzitutto ringraziare il Dott. Ugo Dal Lago per i suoi preziosi consigli, per avermi fornito il materiale di studio e per il tempo dedicato allo sviluppo della mia tesi.

Un grande ringraziamento va alla mia famiglia, a mio padre Stefano, mia madre Cinzia e mia sorella Valentina, ai nonni e agli zii per il sostegno e l'aiuto in questi anni di studio all'università.

Un ringraziamento particolare va alla mia ragazza Laura che mi ha sopportato con pazienza nella preparazione degli ultimi esami.

Un sentito ringraziamento va a miei compagni di corso (Marco, Dario, Federico, Enrico, Davide, e tutti gli altri) con i quali ho preparato e superato i corsi più difficili. Un ricordo speciale va alle ore (e ore) passate a scrivere righe e righe di codice nei vari progetti del corso di laurea.

Ed infine ringrazio i miei amici (quelli pazzerelli) che mi hanno visto sparire per mesi e nonostante tutto ci sono sempre stati.