La progettazione di applicazioni Toolbook con UML

Massimo Cossentino[(1)], Umberto Lo Faso [(2)]

[(1)] Dipartimento di Ingegneria Elettrica – Università degli studi di Palermo – Viale delle Scienze, 90128 Palermo (Italy). E-mail: maxco@unipa.it
[(2)] Dipartimento di Ingegneria Automatica ed Informatica – Università degli studi di Palermo – Viale delle Scienze, 90128 Palermo (Italy). E-mail: lofaso@unipa.it

La metodologia di progettazione qui presentata cerca di conciliare la architettura del Toolbook (notoriamente non orientata agli oggetti nel senso più completo del termine) con quelle del linguaggio di progettazione del software UML. Da questo approccio discende un procedimento in 6 fasi che, utilizzando i diagrammi dell'UML, a partire da una descrizione testuale (e quindi informale) dei requisiti del software procede verso una formalizzazione delle funzionalità e quindi ad una descrizione sia dei percorsi previsti nell'ipertesto che delle interazioni in ogni singola pagina. Da questi elementi sarà quindi possibile dedurre la struttura dell'applicazione rappresentata secondo un classico diagramma delle classi.

1 Introduzione

Il Multimedia Toolbook è un ambiente di sviluppo che nel corso degli anni è stato sempre più utilizzato per la realizzazione di applicazioni multimediali soprattutto ad indirizzo didattico. I cosiddetti courseware (parola composta da course cioè corso e software) sviluppati in Toolbook sono numerosi e sempre più complessi. Spesso la realizzazione di queste applicazioni è frutto di un lavoro di gruppo in cui al grafico si aggiungono gli autori dei testi e, ovviamente, i programmatori.

Al crescere della complessità e delle dimensioni di questi software è naturale che si senta l'esigenza di uscire dalla fase di sviluppo 'amatoriale' a quella 'professionale' che richiede un rigoroso progetto.

Nel campo della ingegneria del software (e quindi del progetto del software), gli ultimi anni hanno visto l'affermarsi dell'UML [5]come linguaggio per la progettazione delle applicazioni scritte in linguaggi orientati agli oggetti.

Parecchie metodologie sono state sviluppate nel passato per il progetto delle applicazioni ipermediali; tra le altre possiamo citare quella di Grønbaek e altri riferita al modello di Dexter [11], quella messa a punto da Nanard e Nanard [12], la variante ad oggetti del noto HDM [13], la OOHDM [14] di Schwabe, Rossi e Barbosa. Esse pur rimanendo ampiamente valide non sono pensate in funzione dell'UML e quindi è necessario ripensare ad esse per ottenere un nuovo approccio.

Quella che si presenterà nel seguito è una metodologia di progetto che utilizza l'UML e tiene conto delle particolari caratteristiche del Toolbook che in effetti non può essere considerato un vero linguaggio orientato agli oggetti. Per supportare questa affermazione basterebbe, infatti, pensare alla mancanza della nota relazione di ereditarietà (vedasi [10] pag.196) fra gli elementi del linguaggio. Per questo si descriverà preliminarmente la possibile architettura di una applicazione che tenendo conto dei limiti del Toolbook sia strutturata secondo i concetti propri dello sviluppo object-oriented.

Si procederà quindi ad illustrare un processo in 6 fasi che a partire da una descrizione testuale (e quindi informale) dei requisiti del software procede ad una formalizzazione delle funzionalità e quindi ad una descrizione sia dei percorsi previsti nell'ipertesto che delle interazioni in ogni singola pagina. Da questi elementi sarò possibile dedurre la struttura dell'applicazione dal punto di vista dell'elenco di pagine ed oggetti da disporre nelle stesse nonché del comportamento degli stessi elementi.

2 Lo Unified Modeling Language (UML)

UML è un linguaggio che può essere usato per analizzare, specificare, documentare un artefatto software [6]. Le sue origini possono essere ritrovate in parecchi approcci di modellazione e nei linguaggi object-oriented di alcuni anni fa (l'OOADA di Booch, l'OOSE di Jacobson, l'OMT di Rumbaugh e altri).

Verso la fine del 1997 la versione 1.1 dell'UML fu approvata dall'OMG (Object Management Group). In questa occasione l'UML divenne il primo linguaggio standard di modellazione e ben presto il più supportato dall'industria.

Le più grandi aziende nel settore informatico hanno contribuito alla definizione di questo standard e sono interessate alla sua crescita. Il mondo accademico è coinvolto nella esplorazione dei vari problemi connessi con la introduzione dell'UML nel processo di progettazione e sviluppo del software.

UML non è stato concepito per supportare un ben preciso processo di sviluppo e può essere utilizzato sia con vecchie che con nuove metodologie. Questo è un grande campo di ricerca che è ancor aperto.

In questo contesto abbiamo studiato l'applicazione di UML alla progettazione di software basati sul Toolbook. Tenendo presenti i classici approcci alla progettazione nonché alcune nuove metodologie emergenti come lo Unified Process [1] abbiamo sviluppato un processo che si adatta alle specifiche esigenze della progettazione di un software che è nella sua essenza prevalentemente visuale ma non di meno presenta spesso problemi architetturali che necessitano di un adeguato studio.

2.1 I diagrammi dell'UML

Poiché la maggior parte degli strumenti di progetto del software (CASE tools, strumenti per la Computer Aided Software Engineering) sono interamente in inglese ed adoperano le denominazioni originali dei vari elementi dell'UML, tali termini entrano ben presto a far parte del linguaggio condiviso dagli addetti ai lavori. Per questa ragione in questa trattazione si useranno speso i nomi in inglese affiancando ad essi, le prime volte, le corrispondenti traduzioni in italiano.

Per la fase di analisi con particolare riguardo al problema della descrizione del sistema da sviluppare, del suo contesto, dell'interazione delle entità esterne, l'UML offre due diagrammi: lo Use Case Diagram (diagramma dei casi d'uso) e il tradizionale, ben noto, Class Diagram (diagramma delle classi). Il primo è centrato sui casi d'uso (che illustrano gli aspetti funzionali del sistema) e sugli attori (entità esterne al sistema e che interagiscono con esso). Il diagramma dei casi d'uso è stato probabilmente il maggior contributo di Jacobson [6] allo sviluppo dell'UML. Il diagramma delle classi contiene sia le classi (entità del sistema) che le relazioni tra esse. Le classi possono possedere attributi e metodi (in relazione al comportamento del sistema).

Dal punto di vista dei sistemi software, gli elementi che costituiscono l'UML possono essere raggruppati in cinque aree concettuali: la struttura statica, il comportamento dinamico, l'implementazione, il modello organizzativo ed i meccanismi di estensione.

Per descrivere la struttura statica di una applicazione si può usare il diagramma delle classi. In esso le classi sono rappresentate mediante un rettangolo contenente tre campi. Nel primo si trova il nome della classe, nel secondo l'elenco degli attributi, nel terzo i metodi. Inoltre vengono anche indicate le relazioni tra le classi stesse.

Il comportamento dinamico del sistema che discende dall'analisi compiuta nei diagrammi dei casi d'uso, può essere descritto usando i diagrammi di sequenza (sequence diagram) e i diagrammi di collaborazione (collaboration diagram).

Questi presentano punti di vista diversi dello stesso scenario. Gli scenari sono percorsi tra i casi d'uso che illustrano i diversi comportamenti del software. Nei diagrammi di collaborazione l'attenzione è focalizzata su quali messaggi una certa entità del sistema scambia con le altre. I

diagrammi di sequenza invece, mostrano gli stessi messaggi ma stavolta organizzati in ordine temporale.

I diagrammi a stati finiti (finite state diagram) ed i diagrammi d'attività (activity diagram) possono essere usati per descrivere una certa procedura o la vita della classe.

Per supportare gli aspetti implementativi del progetto, UML offre il diagramma dei componenti (component diagram) e quello di dislocazione (deployment diagram). Nel diagramma dei componenti, le classi sono associate con i componenti (file eseguibili, librerie, …) che verranno creati. Nel diagramma di dislocazione vengono rappresentati i processi da eseguire, i nodi (unità di esecuzione e altre periferiche) coinvolti e le loro interconnessioni.

Dei contenitori (package) possono essere usati per organizzare la struttura dei sistemi più grandi; essi permettono di semplificare la gestione di un sistema che coinvolge un gran numero di elementi organizzandoli in modo gerarchico.

Fanno parte dell'UML anche alcuni meccanismi di estensione: gli stereotipi (stereotype, elementi del modello simili a quelli standard ma con vincoli o proprietà addizionali), le etichette (tagged value, che possono essere attaccate a qualunque elemento del modello per contenere informazioni addizionali) ed i vincoli (constraint, che possono essere usati per creare relazioni semantiche tra gli elementi del modello specificando condizioni e proposizioni logiche). I vincoli sono descritti mediante il linguaggio OCL (object constraint language) che è stato appositamente associato all'UML.

3 Struttura di una applicazione Toolbook

Come è ben noto OpenScript, il linguaggio di programmazione utilizzato dal Toolbook è basato sugli oggetti. Esso unisce una grande flessibilità lessicale ad una accettabile flessibilità nella rappresentazione di situazioni tipiche del software multimediale didattico.

Indubbiamente la mancanza di alcune strutture tipiche di linguaggi più complessi può talvolta condizionare alcune soluzioni ma nello specifico contesto applicativo in cui il Toolbook si colloca il problema sorge alquanto di rado.

Le principali strutture dati presenti sono dal punto di vista statico gli array e da quello dinamico gli stack. Mancano tipi strutturati più avanzati come i record e soprattutto la possibilità di creare nuovi tipi di dati.

In effetti tutto l'ambiente di programmazione è più orientato alla gestione del comportamento degli oggetti grafici che al supporto di soluzioni eleganti dal punto di vista algoritmico. Esiste comunque la possibilità di creare procedure ricorsive.

Si esporrà adesso la struttura semplificata di una tipica applicazione Toolbook trattando i vari elementi della stessa come classi in un diagramma delle classi conforme allo standard UML.

In fig. 1 si può notare che l'applicazione (classe application) è l'elemento di più alto livello. Una applicazione è in relazione di tipo 'aggregate' (aggregazione) con i book che la costituiscono e che possono essere da 1 ad n. La classe Book può essere per certi versi considerata il baricentro della intera architettura. Il book come è noto può 'ereditare' dei comportamenti da uno o più system book, può includere risorse (resources) di vario tipo (immagini, cursori, font di caratteri, …) mediante una relazione di aggregazione con questi elementi. E' noto che mentre le risorse entrano a far parte del book che le contiene e vengono salvate al suo interno così non è per le clip. Esse infatti sono soltanto dei riferimenti per dei file fisicamente distinti dal book che sono posti in cartelle che possono anche essere diverse da quella dove è posto il book stesso. Questa diversità spiega il fatto che la relazione tra la classe book e quella Clips è di associazione e non è di un tipo più forte come la aggregazione propria del legame con le risorse.

Il book inoltre aggrega al suo interno altre tre classi: i background (ossia gli sfondi delle pagine), i foreground (ossia i primi piani delle pagine) ed i viewer che permettono di visualizzare le pagine stesse. Si noti che un book nella sua espressione più semplice conterrà sempre almeno una istanza delle classi anzidette.

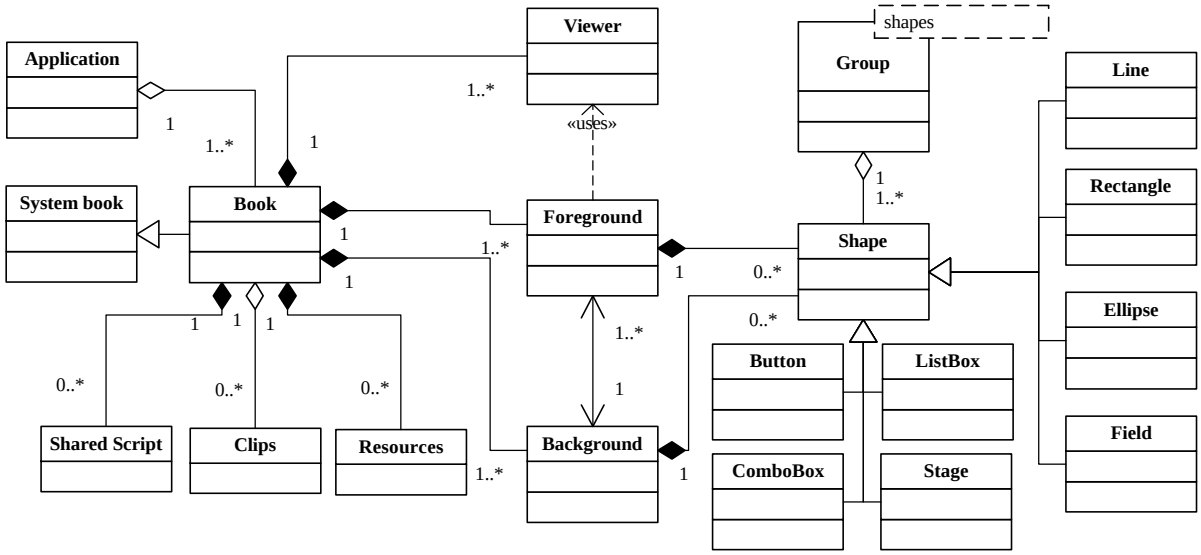

Fig. 1 – La struttura di una applicazione in Toolbook

Sia lo sfondo che il primo piano delle pagine possono contenere gli oggetti grafici (shape) la cui facilità di gestione è stata determinante nel successo del Toolbook.

La classe shape è stata introdotta al solo scopo di raggruppare logicamente alcune proprietà che sono comuni a tutti gli elementi grafici del Toolbook. Si pensi ad esempio alle coordinate, ai colori di bordo e riempimento e così via. Essa è una classe 'virtuale', non può cioè essere utilizzata (instanziata) direttamente ma soltanto le classi da essa derivate possono essere inserite nel programma.

Dalla classe Shape si ricavano i ben noti elementi grafici (bottoni, linee, field, …) mediante una semplice relazione di ereditarietà.

I vari elementi grafici possono essere raggruppati e quindi posti in relazione con la classe Group. Il gruppo è a tutti gli effetti un elemento autonomo e può avere un proprio comportamento (con la relativa porzione di codice OpenScript). Nel diagramma è rappresentato come una classe parametrica; il tipo di elementi che lo costituiscono darà luogo al valore di tale parametro.

Un discorso a parte meriterebbe la classe SharedScript. Essa è l'unica possibilità che si ha in Toolbook di far condividere uno stesso comportamento ad oggetti diversi. Essa in pratica concretizza la possibilità di introdurre una sorta di relazione di ereditarietà (almeno nei comportamenti). Un book può contenere diversi shared script che possono essere assegnati a qualunque elemento della applicazione (che quindi ne erediterà il codice).

4 Progettare con UML per Toolbook

Nel seguito, mediante un esempio concreto si illustrerà una possibile metodologia di progettazione di una applicazione Toolbook.

I courseware, per produrre i quali spesso il Toolbook viene utilizzato, si caratterizzano per la ricorrente presenza di alcuni elementi fondamentali (menu, trattazione degli argomenti, esercizi, …) che vengono ripetuti parecchie volte.

Una adeguata progettazione aiuta ad identificare gli elementi comuni al fine di ottimizzare gli sforzi. Sarà più facile riutilizzare il software già prodotto ed identificare le parti comuni a punti diversi della applicazione se si ha a disposizione una semplice rappresentazione grafica dell'applicazione stessa nei suoi diversi aspetti (funzionali, strutturali, …)

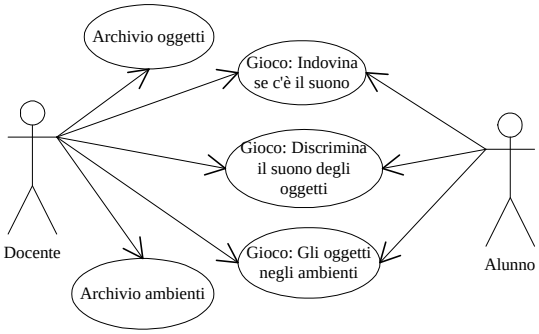

Fig. 2 – Il primo diagramma dei casi d'uso descrive le funzionalità principali del sistema e mostra le interazioni con esse degli attori coinvolti.

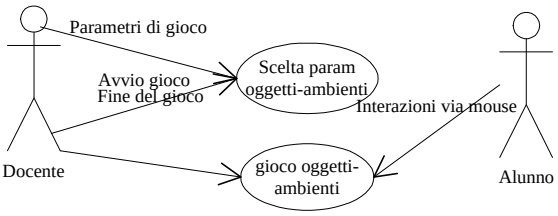

Fig. 3 – Il dettaglio delle funzionalità del caso d'uso "Gioco: gli oggetti negli ambienti" di fig. 2.

Ciò che è importante sottolineare, comunque, è la importanza fondamentale della fase di analisi da cui scaturisce la identificazione delle funzionalità da implementare nel software. Tali funzionalità verranno descritte in appositi diagrammi dei casi d'uso. Da queste funzionalità discenderà la parte cruciale del progetto cioè la struttura del programma così come verrà descritta nei diagrammi delle classi.

Come è noto l'UML è un linguaggio di specifica del software che dichiaratamente non supporta alcuna particolare metodologia di progetto. Dipende dal progettista in base al contesto nel quale opera scegliere il proprio modo di procedere.

La metodologia che descriveremo si articola in 6 passi che iniziano con la analisi dei requisiti del software e arrivano alla identificazione della struttura dell'applicazione e alla sua modularizzazione.

4.1 Descrizione testuale dei requisiti

Tale descrizione deve comprendere tutte le funzionalità ed illustrare come gli utenti interagiranno con il software. Si noti che nel caso in cui il programma che si sta progettando sia un courseware, in questa descrizione non bisogna dettagliare i contenuti delle varie pagine. Infatti il progetto dei testi (e dei media) va distinto dal progetto della struttura del software che li conterrà.

4.2 Formalizzazione delle funzionalità

Si utilizzano gli use case diagram (diagrammi dei casi d'uso, per caso d'uso si intende una sequenza di azioni che il sistema può compiere). Sulla base della descrizione testuale verranno identificati i vari casi d'uso (rappresentati nel diagramma da un ovale) che saranno utilizzati per tracciare dei diagrammi a differenti livelli di dettaglio: al livello più alto si illustra il software ed i suoi utenti (gli utenti, cioè gli attori, vengono rappresentati con delle figure umane stilizzate). Ai livelli successivi si dettagliano le sue parti. Gli use case (casi d'uso) di primo livello vengono dettagliati fino a quando è possibile associare i vari use case a singole pagine Toolbook.

Spesso non è necessario dettagliare ulteriormente in quanto in genere una singola schermata non presenta molte funzionalità diverse. Quello che è importante ricordare è che una notazione grafica come quella che si sta usando non può essere auto-descrittiva. Ad essa vanno associati ampi commenti testuali che in genere vengono immessi negli appositi campi delle note associati ai vari elementi del disegno. Così facendo ogni caso d'uso conterrà la descrizione delle funzionalità che implementa; ciò permette una maggiore facilità di condivisione del progetto e di successiva revisione.

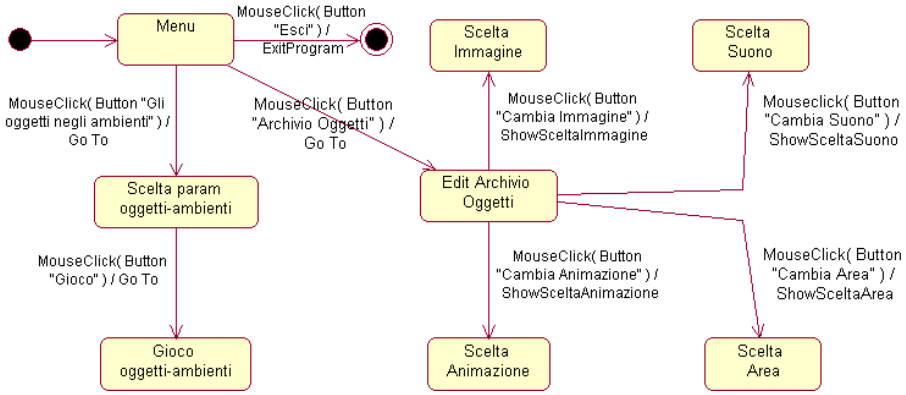

Fig. 4 – Percorsi tra le pagine del programma illustrati con uno statechart diagram

A titolo di esempio si consideri il progetto di un software per il supporto alla riabilitazione di alunni con handicap uditivo. Si può ipotizzare che gli scenari tipici di utilizzo del software prevedano la presenza del docente e dell'alunno. Si potrebbe quindi, iniziare il progetto con un diagramma dei casi d'uso come quello di fig. 2. In esso sono rappresentati i casi d'uso che rappresentano le funzionalità emerse dalla descrizione testuale di cui al punto precedente. Inoltre sono anche presenti due attori (l'alunno ed il docente) ed è specificato con quali casi d'uso interagiscono. Ad esempio il docente interagirà con le funzionalità di archiviazione degli oggetti che non verranno invece utilizzate dall'alunno.

Si consideri adesso il caso d'uso "Gioco: gli oggetti negli ambienti". Esso rappresenta un gioco in cui l'alunno esplora con il mouse un ambiente in cui si trovano degli oggetti che vengono programmati per reagire (con un suono o anche con una animazione) al click del mouse e/o al passaggio del mouse sopra di essi. Il docente in una prima fase sceglierà i parametri dell'esercizio (ambiente, oggetti presenti, interazione con gli oggetti, …), quindi avvierà il gioco e l'alunno allora potrà esplorare l'ambiente (Fig. 3). In questo ulteriore livello di dettaglio si può procedere alla identificazione dei due casi d'uso presenti con altrettante schermate (del software) nelle quali si attueranno le funzionalità illustrate.

Nella pagina "Scelta param oggetti-ambienti" il docente potrà impostare tutti i parametri relativi alla interazione tra l'alunno ed il gioco. Infine avvierà il gioco. L'alunno interagirà con gli oggetti mediante l'esplorazione ed il click del mouse.

4.3 Descrizione dei percorsi fra le schermate

Si utilizzano gli Statechart Diagram (diagrammi di stato). Come si è prima detto, ad alcuni casi d'uso si è fatto corrispondere una pagina del programma che si andrà a realizzare e così si è ottenuta una lista delle pagine da realizzare. E' altresì evidente che un software multimediale/ ipermediale è caratterizzato oltre che dalle funzionalità offerte dalle sue pagine anche (e talvolta soprattutto) dai cammini che si hanno tra di esse. Se si fa corrispondere ad ogni pagina uno stato dello statechart diagram che si sta per tracciare, allora i cammini tra le varie pagine sono rappresentabili mediante la rete delle transizioni di stato. Tali passaggi di stato vengono attivati mediante degli eventi (ad esempio il click su un determinato bottone) che scatenano la transizione secondo la sintassi: evento/ transizione attivata.

Nel diagramma di fig. 4 (che rappresenta solo una parte del software realmente realizzato) si può notare come lo stato iniziale sia relativo alla pagina "Menu". Da essa mediante il click del mouse su alcuni bottoni si può accedere alle pagine "Scelta param oggetti-ambienti", "Edit Archivio Oggetti" e si può anche uscire dal programma. Ad esempio nel passaggio dal menu alla pagina "Scelta param oggetti-ambienti" la didascalia della transizione di stato ci illustra che l'evento mouseclick sul bottone "Gli oggetti negli ambienti" scatena l'azione GoTo che ci porta appunto alla pagina detta.

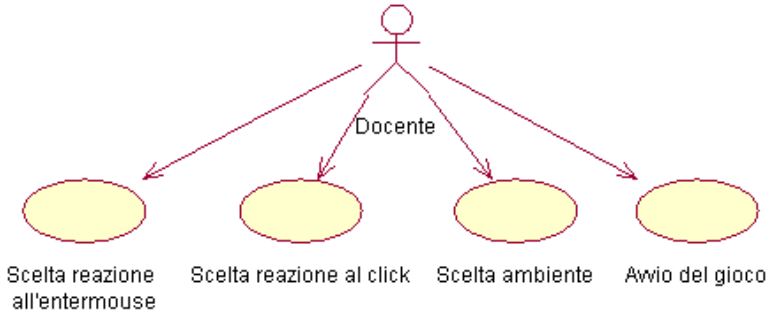

Fig. 5 – Il diagramma dei casi d'uso che descrive le funzionalità della pagina "Scelta param oggetti-ambienti".

In questa metodologia si sta supponendo di avere a che fare con un programma multimediale i cui cammini siano relativamente semplici (e composti da pochi nodi o quanto da gruppi di nodi). La gestione di un ipertesto molto vasto in cui magari le transizioni di stato siano condizionate anche dal valore di alcune variabili potrebbe necessitare di un approccio più complesso in cui si utilizzino più diagrammi organizzati in modo gerarchico.

4.4 Descrizione delle interazioni

Per descrivere le interazioni dell'utente con il software si possono utilizzare i Sequence Diagram (diagrammi di sequenza), i Collaboration Diagram (diagrammi di collaborazione) e gli Activity Diagram (diagrammi delle attività). Scopo di questa fase è esplicitare e dettagliare maggiormente quanto già descritto negli use case con particolare riferimento alla descrizione dei compiti da svolgere (sequenze di interazione ed interdipendenze tra le varie fasi). Si supponga di avere una pagina nella quale bisogna scegliere una immagine da una lista, associarle un nome e quindi una descrizione. In pratica questa pagina presenta una serie di funzionalità diverse con le quali l'attore (l'utente) può interagire. Tali attività devono essere svolte in una sequenza precisa, non avrebbe senso permettere all'utente di inserire il nome di una immagine che non ha ancora scelto. Sarebbe stato possibile tracciare uno use case diagram per descrivere tali funzionalità. In tale diagramma si possono identificare degli "scenari" (uno scenario è una specifica sequenza di azioni che illustrano il comportamento di una determinata porzione del software, può essere usato per descrivere la concretizzazione di uno o più casi d'uso). Esiste una vasta bibliografia sull'utilizzo degli scenari nella fase di definizione dei requisiti del software (vedasi [8],[9]). Il diagramma dei casi d'uso non contiene però informazioni circa le procedure di interazione con il software. Tali fasi possono essere descritte agevolmente mediante i sequence diagram (diagrammi di sequenza). Infatti in essi vengono indicati gli oggetti che concretizzano le funzionalità prescritte dai casi d'uso, gli attori in gioco e le interazioni tra essi. Inoltre è presente un asse temporale che permette di identificare cosa avviene prima e cosa dopo.

Gli activity diagram possono essere visti come una evoluzione dei flow-chart che negli anni passati sono stati molto usati per la progettazione dei software procedurali. Per questa origine sono ora molto usati nella descrizione dei processi.

Adesso illustreremo un possibile approccio a questa fase del progetto. Si consideri il seguente use case diagram che illustra il dettaglio delle funzionalità offerte dalla pagina "Scelta param oggetti-ambienti" già presente nelle figg. 3 e 4.

La fase di scelta dei parametri si concretizza in una serie di interazioni che il docente compie con il programma. Le scelte fatte possono dar luogo a scenari di gioco diversi. Si immagini ad esempio che il docente scelga di far reagire gli oggetti del gioco con un testo al passaggio del mouse e con un suono al click del mouse; inoltre egli sceglierà l'ambiente in cui il gioco avrà luogo (tale ambiente è già completo degli oggetti con cui interagire).

Questa sequenza di operazioni può essere descritta come nel sequence diagram (diagramma di sequenza) di fig. 5.

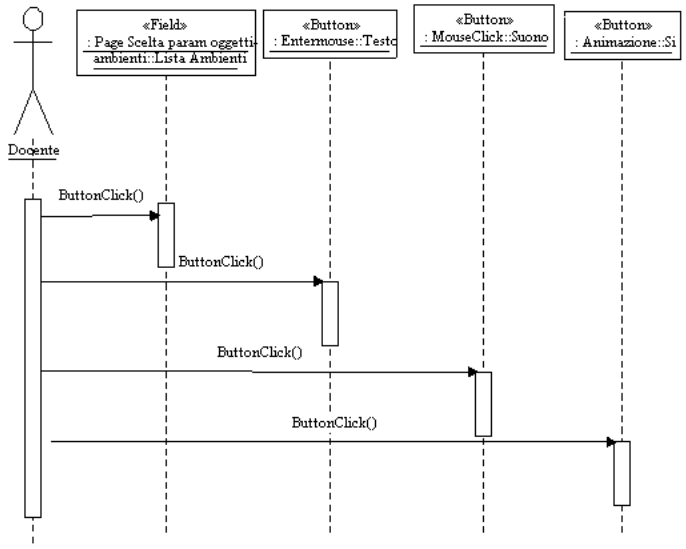

Fig. 5 – Questo diagramma di sequenza illustra i messaggi inviati dal docente al programma per impostare i parametri di gioco.

Nel diagramma si possono notare quattro classi (oltre al docente). Le prime tre sono dei bottoni, la quarta è un field che contiene la lista degli ambienti. E' inoltre presente anche se non rappresentato esplicitamente un asse dei tempi che prevede che i messaggi scambiati dagli oggetti siano ordinati cronologicamente dall'alto verso il basso.

La sintassi del diagramma ci indica che le classi si chiamano rispettivamente "lista ambienti", "suono", "si" e "testo" ed ognuna di esse appartiene ad un diverso gruppo di oggetti (rappresentato in UML da un 'package', cioè contenitore, che è stato specializzato mediante lo stereotipo 'group') il cui nome viene posto prima del nome della classe. Come si vede il docente interagisce con ognuna delle classi mediante il click del mouse. Tale evento scatenerà l'esecuzione del conseguente handler (che verrà implementato come uno dei metodi della classe). Ovviamente contestualmente all'utilizzo del Buttonclick in questo diagramma bisognerà provvedere ad inserire il metodo Buttonclick nelle classi indicate.

Come si può notare il diagramma prescrive che l'utente prima selezioni l'ambiente e poi gli altri parametri.

La descrizione del metodo può essere effettuata mediante un activity diagram che permette di esplicitare il comportamento del codice in una maniera molto simile a quella dei flow-chart.

Si noti che il sequence diagram di fig. 5 rappresenta solo uno dei possibili scenari che si possono immaginare nella descrizione della pagina "Scelta param oggetti-ambienti". Sarà ovviamente necessario descrivere in modo esaustivo tutti gli scenari significativi e da questi si trarranno tutte le classi ed i loro metodi necessari per realizzare le funzionalità imposte nei diagrammi dei casi d'uso.

Spesso non è necessario descrivere tanti scenari diversi, si può invece descrivere l'intera procedura seguita dall'utente mediante un activity diagram. In effetti è il caso di notare che i due approcci non sono del tutto equivalenti in quanto nei sequence diagram è presente l'asse del tempo mentre negli activity diagram il fattore tempo non viene in alcun modo rappresentato. Molto frequentemente i due metodi vanno utilizzati in modo integrato per descrivere da diversi punti di vista lo stesso contesto.

Si descriveranno adesso le interazioni tra l'utente e la pagina prima vista mediante un activity diagram. Si noterà che in un solo diagramma sono presenti tutte le possibili operazioni compiute dell'utente e la descrizione dei conseguenti comportamenti del software.

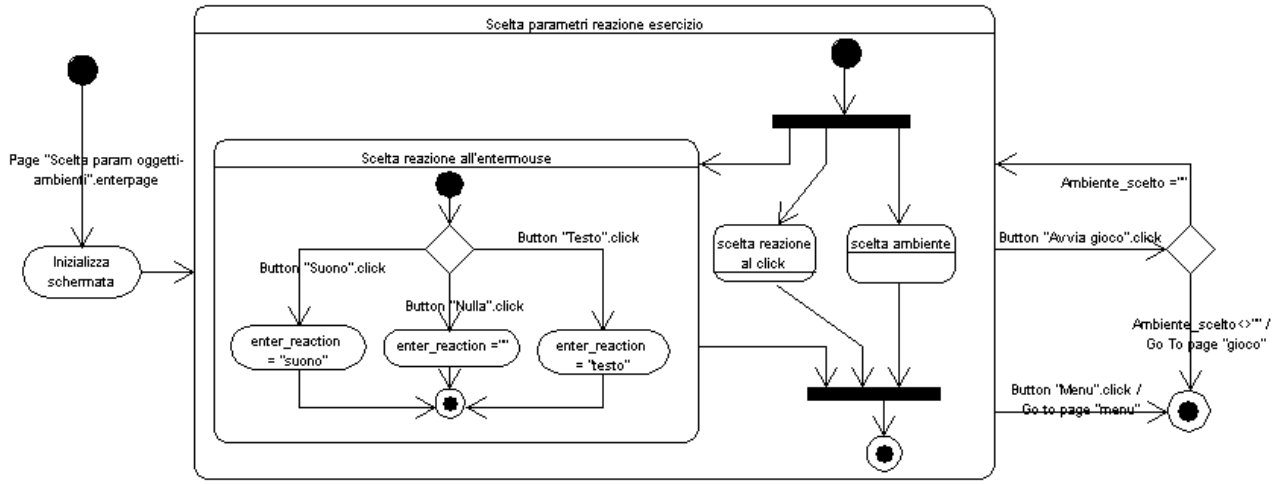

Fig.6 – Il diagramma che illustra il comportamento della pagina “Scelta param oggetti-ambienti”

Il diagramma delle attività (activity diagram) illustrato in fig. 6, descrive l’intero comportamento previsto per la pagina “Scelta param oggetti-ambienti”. Il diagramma è contrassegnato da uno stato iniziale (il cerchio nero in alto a sinistra) dal quale si esce all’arrivo dell’evento “Enterpage” che si ha all’ingresso nella schermata. Viene allora eseguita l’attività “Inizializza schermata”. Essa corrisponderà alla sequenza di operazioni che, all’interno dello handler dell’evento Enterpage, provvederanno ad impostare i valori iniziali delle variabili relative all’esercizio. Al termine di questa procedura si passerà allo stato di “Scelta parametri reazione esercizio”. Si noti che nessuna condizione è associata alla transizione (la linea con la freccia) che porta dalla attività prima vista a questo stato. Infatti una volta terminata l’inizializzazione delle variabili il programma si predispone automaticamente in uno stato di attesa di eventi provenienti dall’utente.

All’interno dello stato “Scelta parametri reazione esercizio” troviamo uno stato iniziale locale (cerchio nero) ed uno stato finale locale (cerchio nero bordato) che indica l’uscita da questo stato. La barra orizzontale nera è una biforcazione (fork). Come indicato dalla fork, i tre stati “Scelta reazione all’entermouse”, “Scelta reazione al click”, “Scelta ambiente” non sono collegati da una dipendenza e quindi l’utente a seconda della sua preferenza potrà accedere ad uno qualunque dei tre. All’uscita dei tre stati il flusso del programma si riunisce in un’altra barra nera che adesso assume il ruolo di ricongiunzione (join).

Di questi ultimi tre stati si è sviluppato il dettaglio del primo soltanto. In esso si nota che vi è una scelta condizionale (indicata dal rombo). A seconda che l’utente faccia click sul bottone “suono” o “testo” o “nulla” si ottiene una diversa assegnazione per la variabile enter_reaction (che verrà utilizzata durante il gioco per determinare la reazione degli oggetti all’evento “entermouse”). La condizionalità nella scelta è realizzabile mediante dei “radio button”.

Non appena l’utente avrà finito di impostare i parametri del gioco potrà uscire da questa schermata. Si può notare che vi sono due possibili vie di uscita dallo stato “Scelta parametri reazione esercizio”. La prima è il click sul bottone “Avvio gioco” che dà effettivamente luogo all’uscita dalla schermata (verso la schermata “gioco”) soltanto se la variabile “ambiente_scelto” ha una valore diverso dalla stringa vuota (cioè se è stato già scelto un ambiente dalla lista). Facendo click sul bottone “menu” nuovamente si esce da questa pagina, stavolta la azione che accompagna questo evento è il passaggio alla pagina “menu”.

Da un diagramma come questo si traggono importanti considerazioni sugli oggetti che bisognerà disporre nella pagina. Quelli con cui l’utente deve interagire vengono menzionati nel diagramma ed il loro comportamento è chiaramente indicato. Si pensi ad esempio ai tre radio button che bisogna usare per la scelta della reazione all’entermouse o al comportamento del bottone “avvio

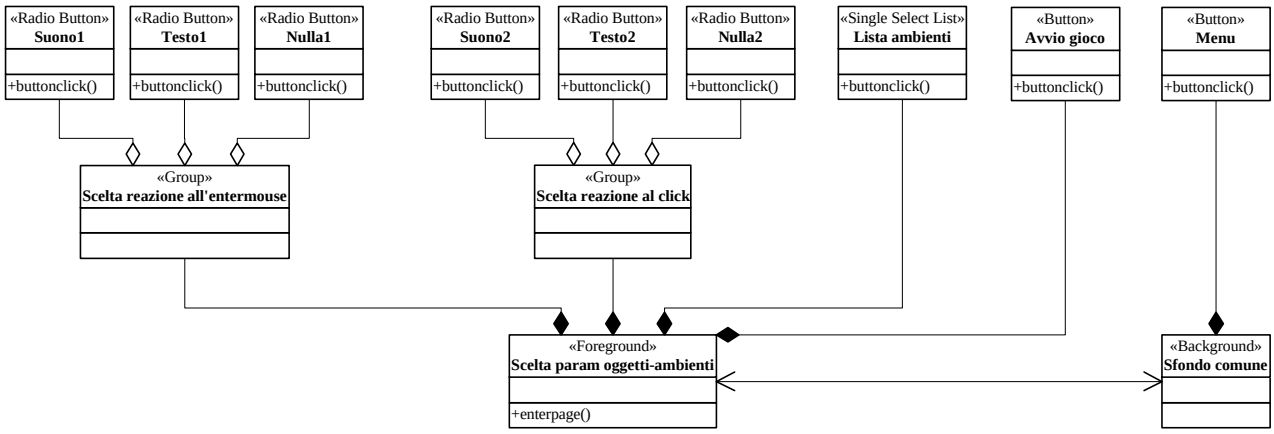

Fig. 7 – Il diagramma che descrive la struttura della pagina “Scelta param oggetti-ambienti”

gioco” che porta effettivamente alla pagina “gioco” solo dopo aver verificato che l’utente abbia scelto l’ambiente in cui giocare.

4.5 Struttura del programma

Dalla descrizione delle interazioni effettuata nel paragrafo precedente discende l’elenco degli oggetti necessari per svolgere le funzioni previste. Proseguendo nell’esempio della pagina “Scelta param oggetti-ambienti” dal diagramma di fig. 6 discende che:

- nella pagina sarà presente un codice di inizializzazione che si può pensare di porre nel primo piano della stessa (foreground), in uno handler di tipo enterpage;
- l’utente per fissare il tipo di reazione ad ognuno degli eventi entermouse e buttonclick potrà scegliere fra tre possibilità (suono, testo, nulla);
- l’ambiente verrà scelto da una lista a selezione singola;
- l’utente potrà uscire da questa schermata con il bottone “avvia gioco” che lo porterà alla pagina “gioco” se è stato scelto un ambiente;
- l’utente potrà tornare al menu principale con il bottone “menu” in qualunque momento.

Questi elementi sono sufficienti per tracciare un diagramma delle classi che descriva completamente la struttura della pagina.

Come si può notare in fig. 7, il primo piano della pagina possiede la procedura enterpage e aggrega il gruppo “scelta reazione all’entermouse” (composto da tre bottoni), il gruppo “scelta reazione al click” (anch’esso composto da tre bottoni), la lista “lista ambienti” ed il bottone “avvio gioco”. Il bottone “menu” è invece posto sul background della pagina perché è prevedibile che esso sia comune ad altre schermate. Sia i bottoni che la lista contengono la procedura buttonclick per reagire al click del mouse secondo quanto precedentemente specificato.

4.6 Modularizzazione

L’ultimo passo del progetto consiste nell’aggregazione delle varie pagine in uno o più book.

Si partirà dall’analisi dello State Transition Diagram di fig. 4; in esso i vari stati (che rappresentano le pagine) possono essere raccolti nei vari book. Spesso in tale operazione risulta molto utile raccogliere le varie classi che costituiscono ogni pagina in un apposito package cui si darà il nome della pagina. Queste pagine andranno poi raccolte in uno o più book.

Il criterio da seguire in questa fase dovrebbe tener conto della struttura delle varie pagine così come descritta nei rispettivi diagrammi delle classi (si veda ad esempio la fig. 6) con alcune accortezze:

a) raccogliere nello stesso book tutte le schermate che appartengono allo stesso caso d’uso;

b) riunire nello stesso book tutte le schermate che presentano strutture simili nei diagrammi delle classi (ciò porta ad una maggiore riutilizzabilità del codice e spesso ad un migliore sfruttamento dei background comuni);
c) raccogliere tutte le schermate che scambiano tra loro grandi quantità di dati o sono strettamente accoppiate nella navigazione.

5 Conclusioni

La metodologia di progettazione che si è illustrata è stata applicata in diversi progetti di applicazioni realizzate con Toolbook. In particolare questi lavori si caratterizzavano per il contemporaneo sviluppo del software da parte di diversi programmatori tutti esperti conoscitori del Toolbook e per la presenza di un committente esterno rispetto alla struttura di sviluppo.

Inizialmente è stato necessario fornire ai componenti del gruppo di lavoro una conoscenza della sintassi dei vari diagrammi e rassicurarli sul fatto che il maggior sforzo iniziale sarebbe stato ampiamente compensato da un risparmio nei tempi di codifica.

Quindi ci si è dedicati alla discussione col committente delle parti iniziali del progetto, quelle relative alle funzionalità, ai percorsi tra le schermate e alle interazioni. Da questa fase è scaturito il progetto nella sua forma finale.

In effetti la previsione di un risparmio di tempi nello sviluppo complessivo del software è stata sempre confermata; la realizzazione del codice è stata più spedita, durante le revisioni con il committente le modifiche necessarie sono state molto minori e i vari programmatori hanno potuto procedere in parallelo in quanto ad ognuno di essi sono state assegnate parti il più possibile indipendenti del software.

La fase di assemblaggio dei moduli prodotti non ha comportato problemi se non per i casi in cui si è evidenziata una carenza nel progetto stesso. A tal proposito un esempio che può essere riportato è quello accaduto in un progetto in cui una variabile di sistema doveva contenere il percorso su disco della cartella in cui si trovava l'applicazione. Poiché la specifica non era abbastanza precisa alcuni programmatori hanno inteso la variabile come qualcosa del tipo "c:\applicazione\" mentre altri la hanno pensata come "c:\applicazione". La differenza è minima ma ha comportato la revisione di ampie porzioni di codice al momento dell'assemblaggio delle parti dei diversi componenti del gruppo di sviluppo.

Quello che si può concludere è che la metodologia è senz'altro utile per minimizzare i tempi di sviluppo (e conseguentemente i costi). In particolare essa riesce a diminuire significativamente i problemi di revisione del progetto in corso d'opera in quanto i requisiti iniziali del software vengono discussi col committente sulla base dei diagrammi frutto della prima parte del progetto stesso. Una volta concordati questi ultimi si è effettivamente riscontrato un elevato livello di accordo sul conseguente risultato software.

Nella stesura di questa metodologia si è evitato di adoperare riferimenti a possibili estensioni dell'UML che sono al momento allo studio. Si pensi ad esempio ai lavori di Manola [1], [3] o di Conallen [4] che pur se riferiti al web potrebbero dare utili spunti anche per la progettazione di ipertesti in Toolbook. Questo perché si voleva ideare un processo che fosse supportato dalla più ampia gamma possibile di strumenti CASE per UML.

Bibliografia

[1] I. Jacobson, G. Booch, J. Rumbaugh, "The Unified Process", IEEE Software, May/June 1999, pp. 96-102.
[2] F. Manola, "Some Web Object Model Construction Technologies", http://www.objs.com/OSA/wom-II.htm, Sept. 1998.
[3] F. Manola, "Technologies for a Web Object Model", IEEE Internet Computing, Jan.-Feb. 1999, pp.38-47.

[4] J. Conallen, “Modeling Web Application Aarchitectures with UML”, Comm. of the ACM, Vol.42, No.10, 1999, pp.63-70.
[5] OMG Unified Modeling Language, version 1.3, June 99, Object Management Group document ad/99-06-08, http://cgi.omg.org/docs/ad/99-06-08.pdf
[6] J. Rumbaugh, I. Jacobson, G. Booch - "The Unified Modelling Language Reference Manual" - Addison Wesley ed.
[7] I. Jacobson, M. Christerson, P. Jonsson and G. Overgaard, “Object-Oriented Software Engineering: A Use Case Driven Approach”, Addison-Wesley, 1992.
[8] A.G. Sutcliffe, N.A.M. Maiden, S. Minocha, D. Manuel, “Supporting Scenario-Based Requirements Engineering”, IEEE Trans. On Soft. Eng., Dec. 98, vol. 24-12, pp. 1072-1088.
[9] M. Jarke, “Scenarios for modelling”, Communications of the ACM, Jan 99, vol. 42-1, pp. 47-78.
[10] A. Fuggetta, C. Grezzi, S. Morasca, A. Morzenti, M. Pezzè, “Ingegneria del software”, edit. Mondadori Informatica, 1997.
[11] Kaj Grønbaek, R. H. Trigg - “Design issues for a Dexter based hypermedia system” - Comm. ACM, 37,2, Feb 1994
[12] J. Nanard, M. Nanard – “Hypertext Design Environments and the Hypertext Design Process” – Comm. ACM, 38,8, Aug. 1995
[13] F. Garzotto, P. Paolini, D. Schwabe - "HDM - A model based approach to hypertext application design". ACM Trans. on Inf. Syst. 11, 1 - Jan 1993.
[14] D. Schwabe, G. Rossi, and S.D.J. Barbosa - "Systematic Hypermedia Application Design with OOHDM" - Proc. ACM Conf. Hypertext ’96 – Mar 1996 - ACM Press